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 _____ of **D15A/D15B** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Project Title: **Roll No.**

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

Year/Sem/Class : D15A/D15B **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: **Roll No.**

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

Project Title: **Roll No.**

Index

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

Project Title: **Roll No.**

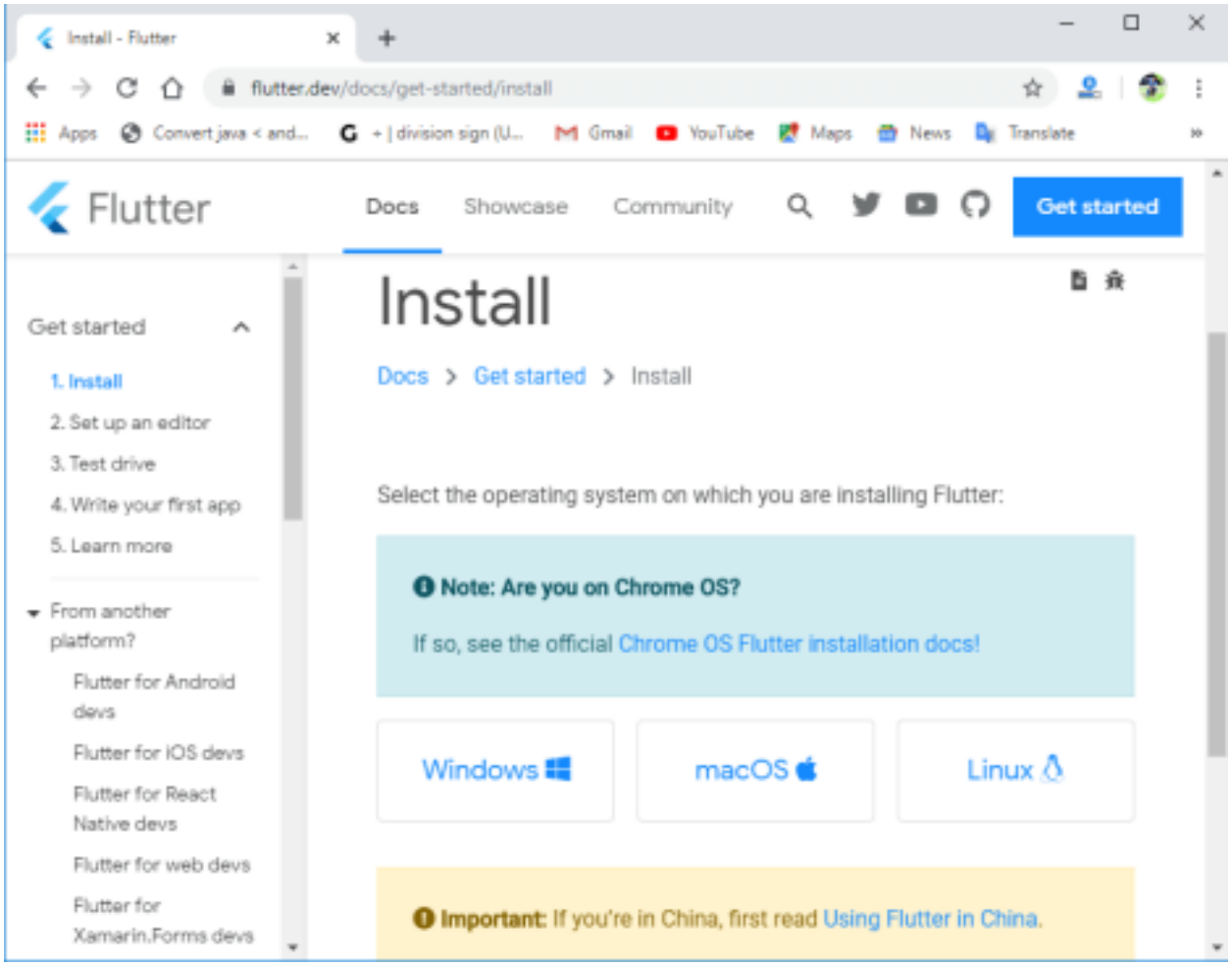
MAD & PWA Lab
Journal

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

Experiment 1

Name:Piyush Tilokani
Div: D15A
Roll no: 63

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



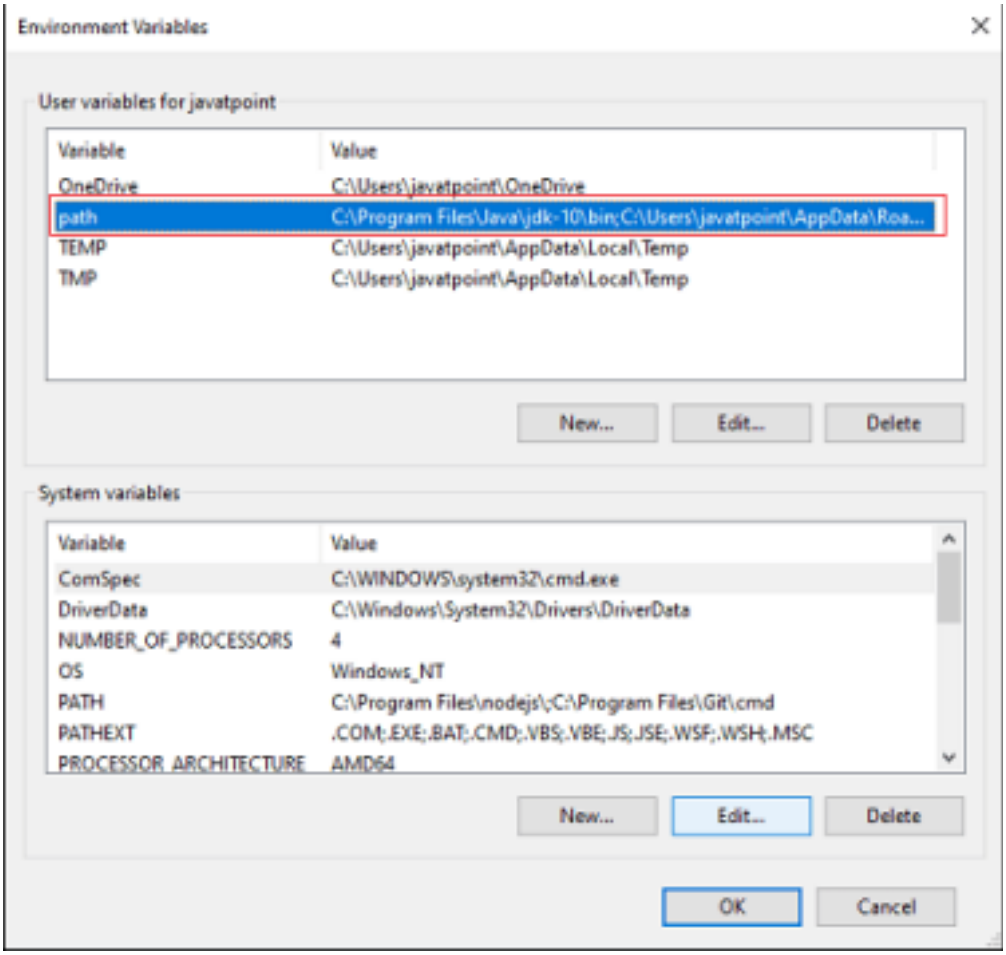
Step 2: Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will find the download link for SDK.

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

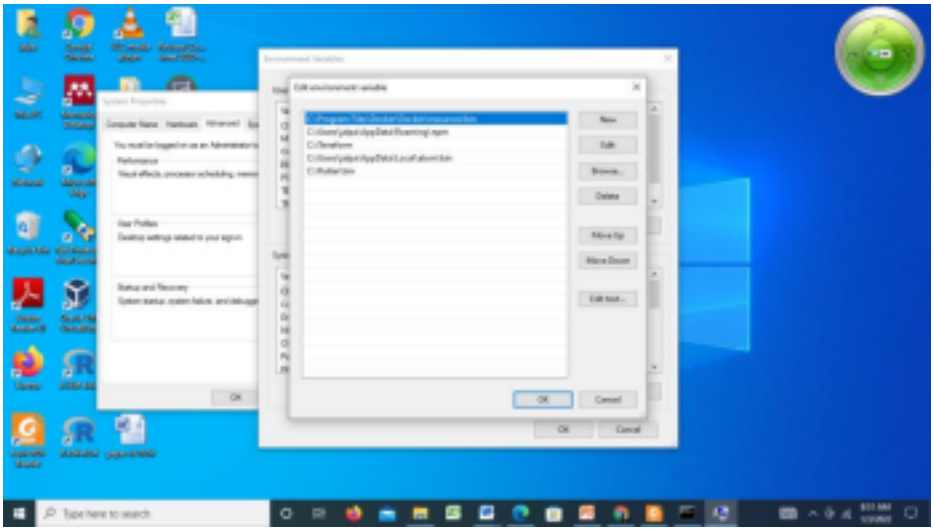
Step 4: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:

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

get the following screen.

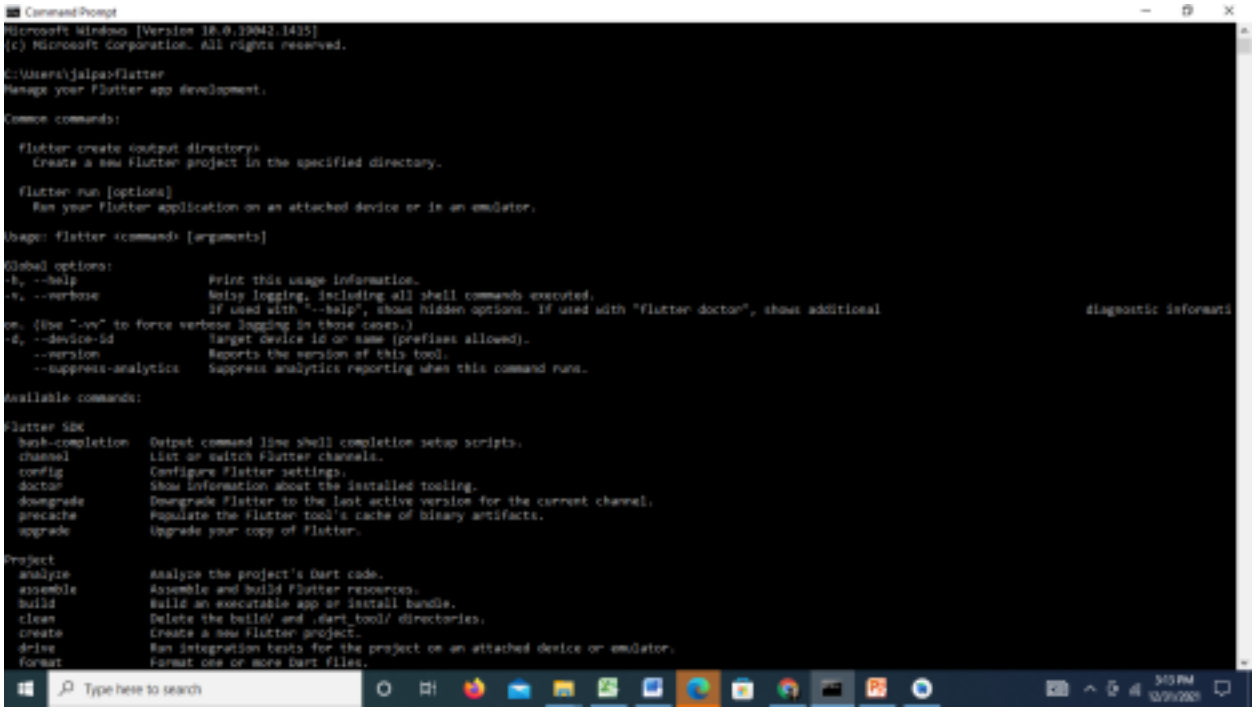


Step 4.2: Now, select path -> click on edit. The following screen appears

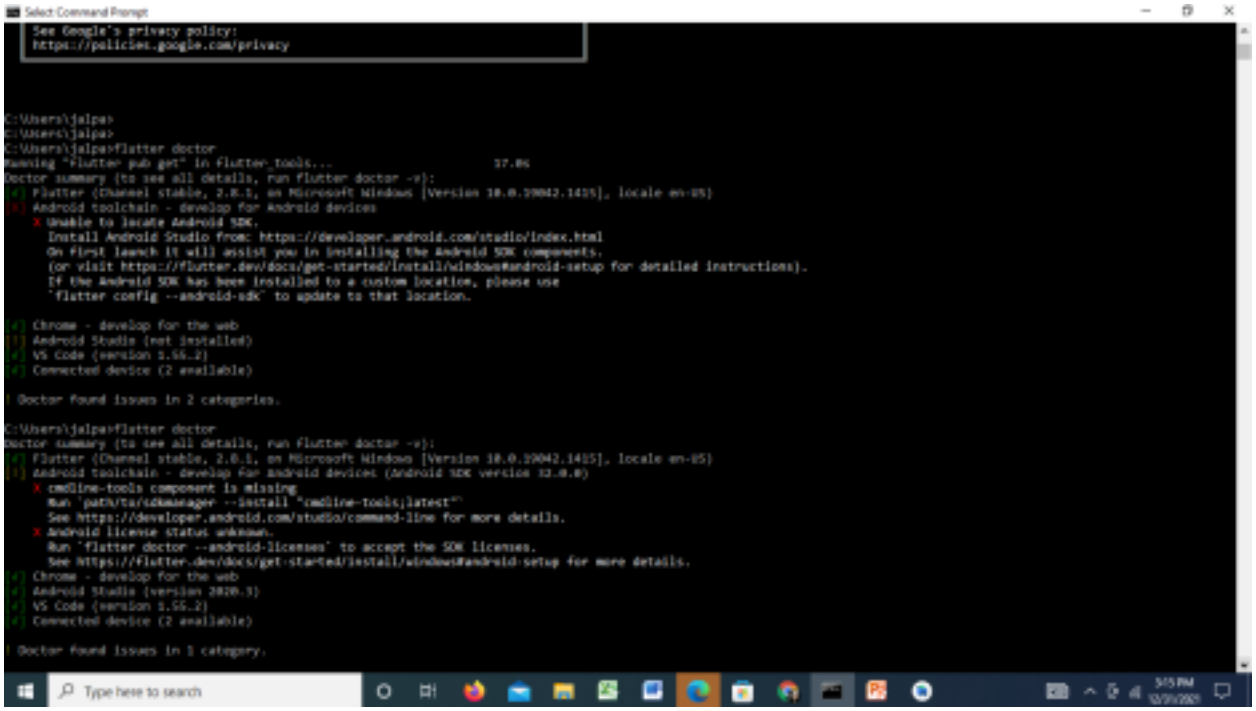


Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value - > ok -> ok -> ok.

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



Now, run the $ **flutter doctor** command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation.



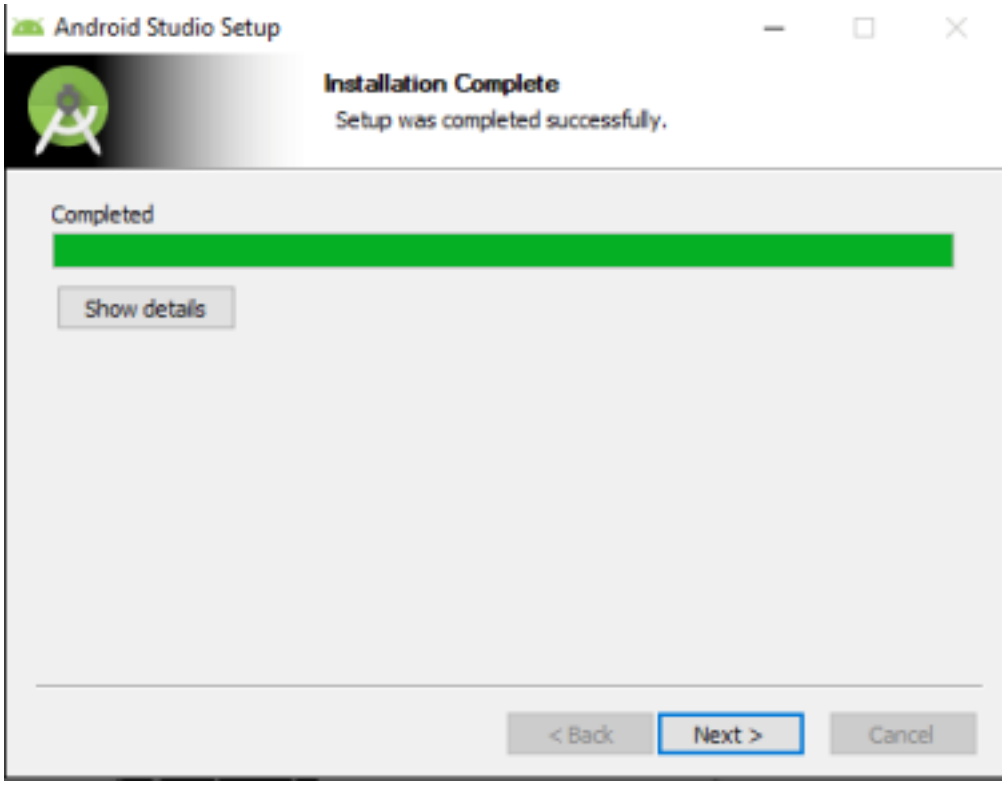
Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

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

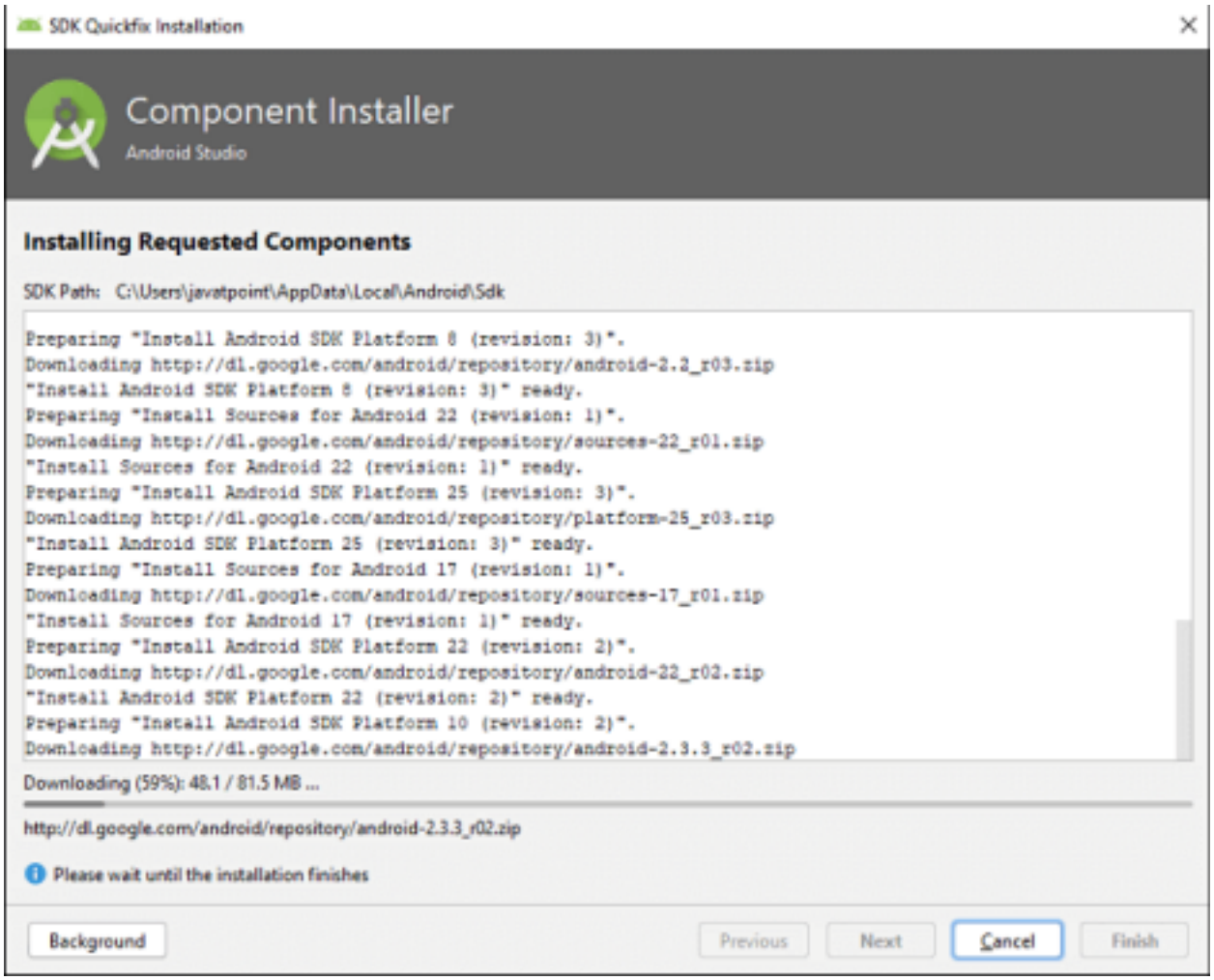
Step 7.1: Download the latest Android Studio executable or zip file from the official site.

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

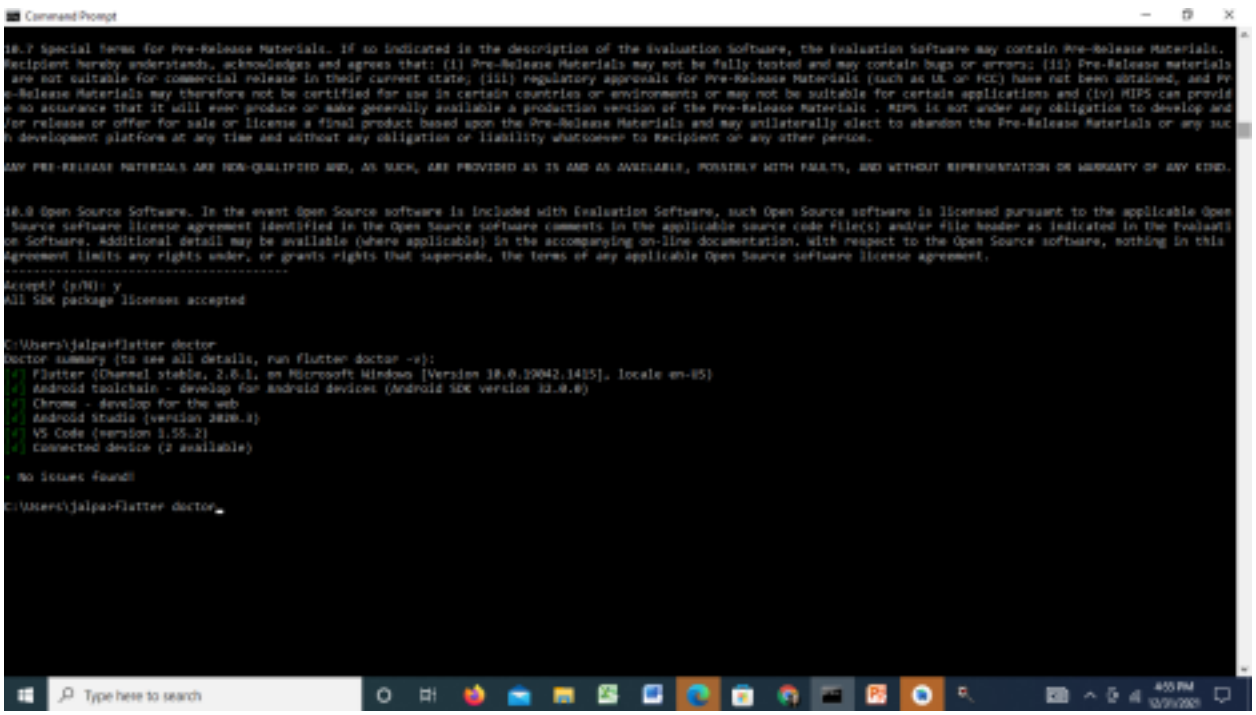
Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



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

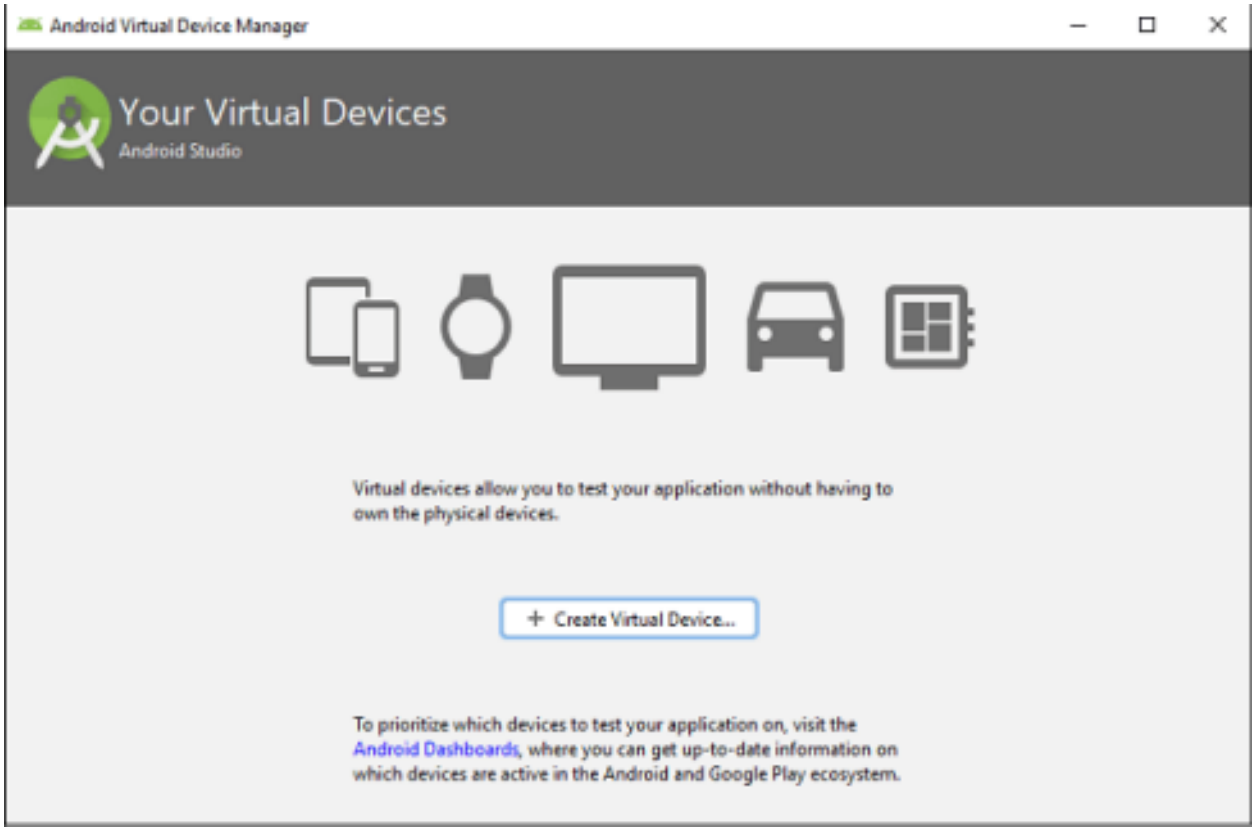


Step 7.5 run the $ **flutter doctor** command and Run flutter doctor --android-licenses command.



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

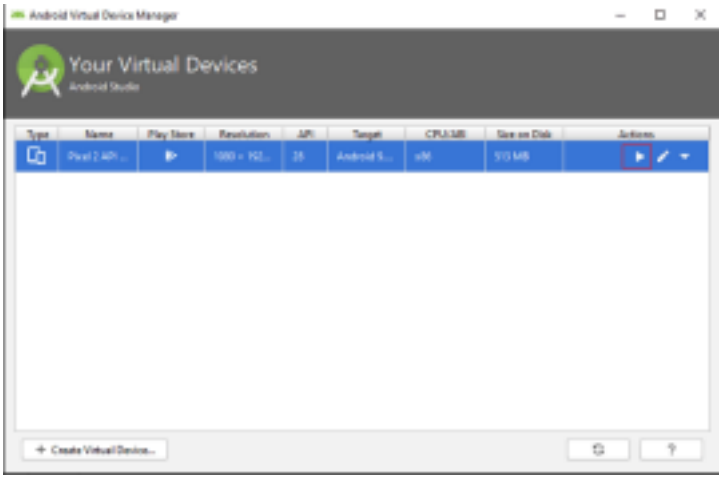
Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.



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

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

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

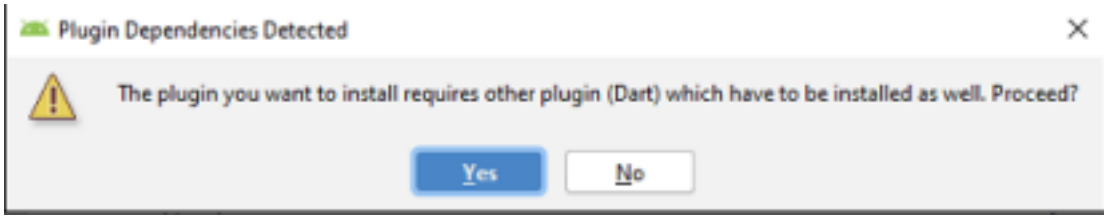


Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

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

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

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



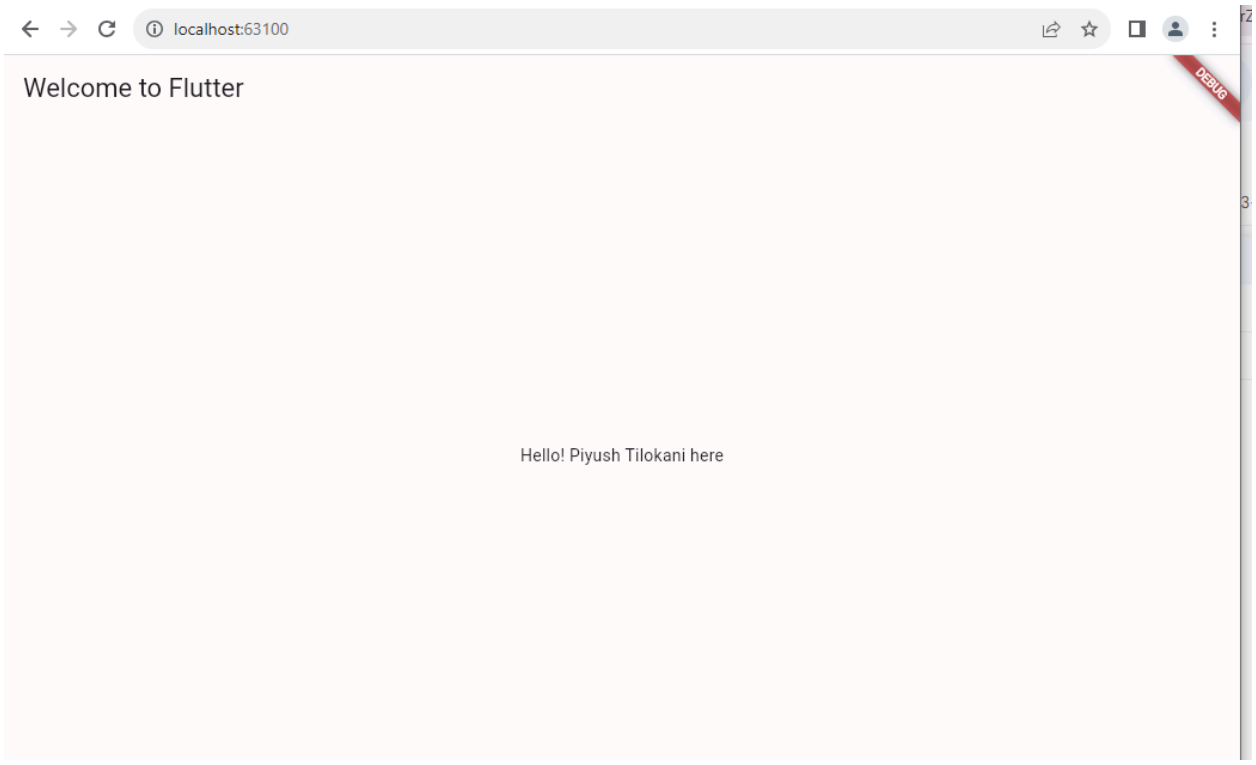
Step 9.3: Restart the Android Studio.

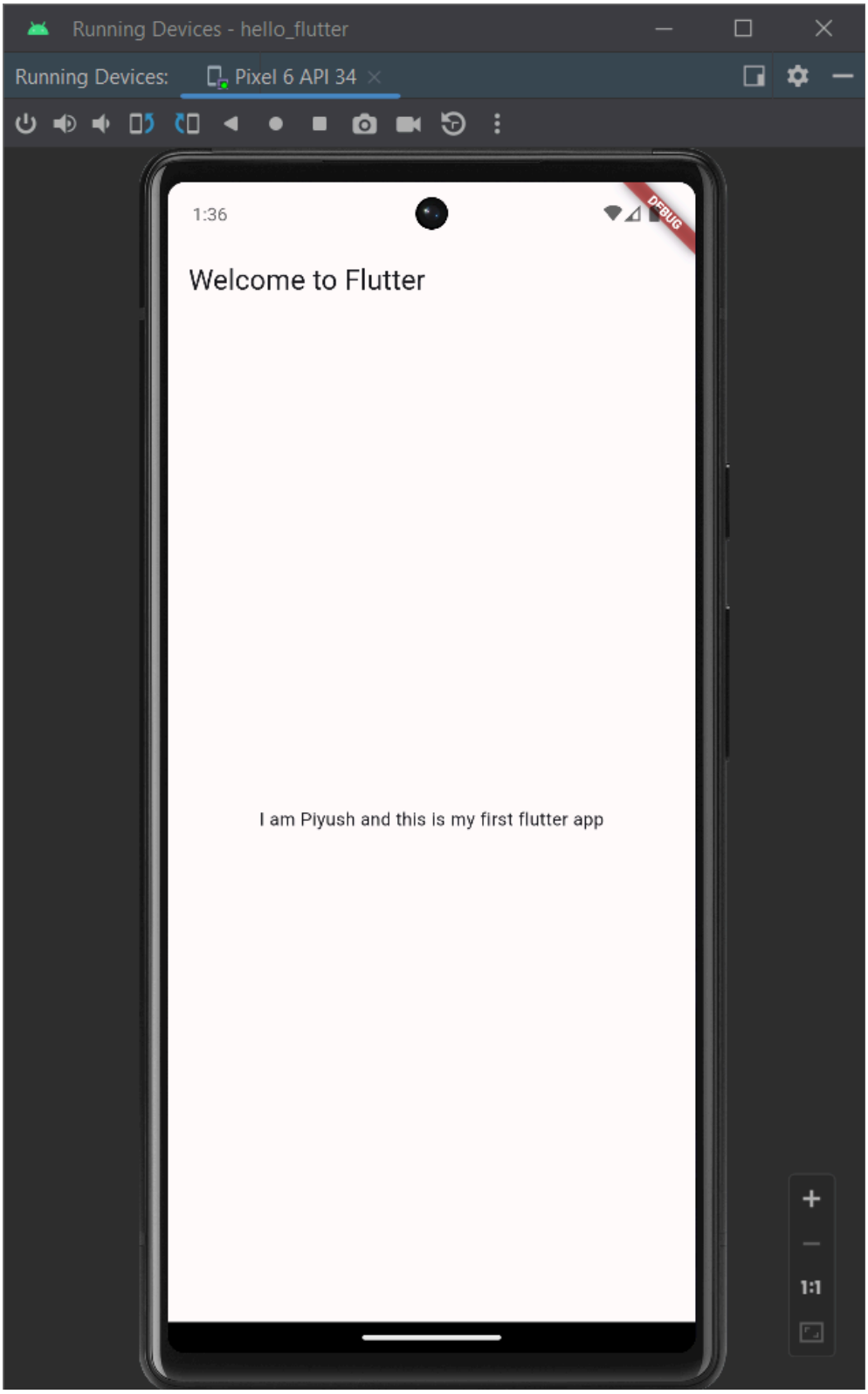
Testing flutter installation on my machine:-

```
C:\Users\acer>flutter --version

┌──────────────────────────────────────────────────────────┐
│ A new version of Flutter is available!                   │
│                                                          │
│ To update to the latest version, run "flutter upgrade". │
└──────────────────────────────────────────────────────────┘
Flutter 3.16.7 • channel stable • https://github.com/flutter/flutter.git
Framework • revision ef1af02aea (8 days ago) • 2024-01-11 15:19:26 -0600
Engine • revision 4a585b7929
Tools • Dart 3.2.4 • DevTools 2.28.5
```

```
C:\Users\acer>flutter devices
Found 3 connected devices:
  Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.3007]
  Chrome (web)      • chrome  • web-javascript • Google Chrome 120.0.6099.225
  Edge (web)        • edge    • web-javascript • Microsoft Edge 120.0.2210.121

Run "flutter emulators" to list and start any available device emulators.

If you expected another device to be detected, please run "flutter doctor" to diagnose potenti
try increasing the time to wait for connected devices with the "--device-timeout" flag. Visit
for troubleshooting tips.
```

Creating first flutter app:-
Code:
main.dart


localhost:63100
Welcome to Flutter
DEBUG
Hello! Piyush Tilokani here



Conclusion: Thus installed android studio, configured my desired Android Virtual Device(AVD) and installed flutter on my machine and created my first flutter project.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Experiment 2

Name:Piyush Tilokani
Div: D15A
Roll no: 63

Aim: To design Flutter UI by including common widgets.

Theory: In Flutter, widgets are the building blocks of the user interface, and several common widgets play crucial roles in creating engaging and interactive applications. Here's a brief overview of some fundamental Flutter widgets:

1. Container: The most basic building block, a container is a box model that can contain other widgets, allowing you to customize its dimensions, padding, and decoration.

2. Row and Column: These widgets help organize children widgets horizontally (Row) or vertically (Column), facilitating the creation of flexible and responsive layouts.

3. AppBar: AppBar is a material design widget providing a top app bar that typically includes the app's title, leading and trailing icons, and actions.

4. ListView: Used to create scrollable lists of widgets, ListView is versatile for displaying a large number of items efficiently.

5. TextField: Enables users to input text, providing a text editing interface with options for validation, styling, and interaction.

6. ElevatedButton is a Flutter widget used to create a button with a raised appearance. It typically represents the primary action in a user interface. The button has a background color, elevation, and responds to user interactions with visual feedback

7. Image: The Image widget displays images from various sources, supporting both local and network images.

8. Scaffold: A top-level container for an app's visual elements, Scaffold provides a structure that includes an AppBar, body, and other optional features like drawers and bottom navigation.

9. Card: Representing a material design card, this widget displays information in a compact and visually appealing format, often used for grouping related content.

10. GestureDetector: Allows detection of various gestures like taps, drags, and long presses, enabling interactive responses to user input.

11. Stack: A widget that allows children widgets to be overlaid, facilitating complex UI designs by layering widgets on top of each other.

12. FutureBuilder: Ideal for handling asynchronous operations, FutureBuilder simplifies the management of UI updates based on the completion of a Future, making it valuable for fetching and displaying data.

These are just a few of the many widgets available in Flutter, each serving a unique purpose in crafting dynamic and user-friendly interfaces.

Code:

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:faker/faker.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';
import 'package:google_fonts/google_fonts.dart';

class DetailPage extends StatelessWidget {
  const DetailPage({super.key, required this.imageUrl});
  final String imageUrl;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Column(
        children: [
          Expanded(
            child: Container(
              decoration: BoxDecoration(
                image: DecorationImage(
                  image: CachedNetworkImageProvider(imageUrl),
                  fit: BoxFit.cover,
                ),
              ),
              child: SafeArea(
                child: Padding(
                  padding: const EdgeInsets.symmetric(horizontal: 18),
                  child: Column(
                    mainAxisAlignment: MainAxisAlignment.spaceBetween,
                    crossAxisAlignment: CrossAxisAlignment.end,
                    children: [
                      Row(
                        mainAxisAlignment: MainAxisAlignment.spaceBetween,
                        children: [
                          InkWell(
                            onTap: () => Navigator.pop(context),
                            child: CircleAvatar(
```

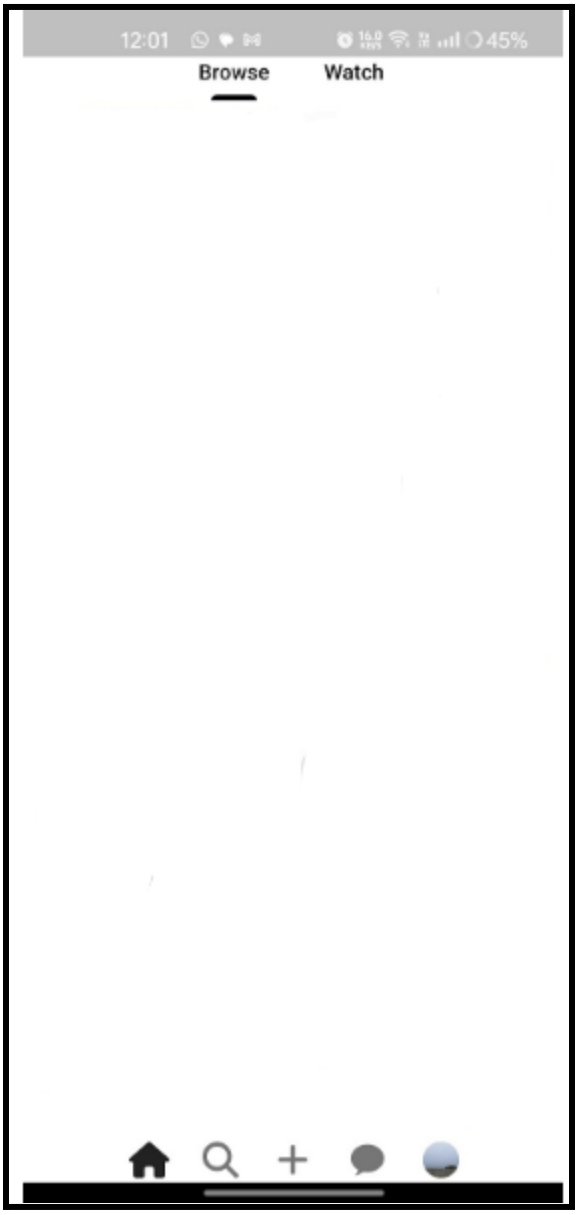
```
    backgroundColor: Colors.black.withOpacity(0.2),
    child: const Icon(
      CupertinoIcons.chevron_back,
      color: Colors.white,
      size: 28,
    ),
  ),
),
InkWell(
  onTap: () {
    showModalBottomSheet(
      context: context,
      builder: (context) {
        return SizedBox(
          height: 330,
          child: Padding(
            padding: const EdgeInsets.symmetric(
                horizontal: 12, vertical: 16),
            child: Column(
              crossAxisAlignment:
                  CrossAxisAlignment.start,
              children: [
                Row(
                  crossAxisAlignment:
                      CrossAxisAlignment.center,
                  children: [
                    const Icon(
                        FontAwesomeIcons.xmark),
                    const SizedBox(
                      width: 12,
                    ),
                    Text(
                      'Options',
                      style: GoogleFonts.notoSans(),
                    ),
                  ],
                ),
                const SizedBox(
                  height: 30,
                ),
                Text(
                  'Follow ${Faker().internet.userName()}',
                  style: GoogleFonts.notoSans(
                    fontSize: 18,
                    fontWeight: FontWeight.w500,
                  ),
```

```
),
const SizedBox(
  height: 15,
),
Text(
  'Copy link',
  style: GoogleFonts.notoSans(
    fontSize: 18,
    fontWeight: FontWeight.w500,
  ),
),
const SizedBox(
  height: 15,
),
Text(
  'Download image',
  style: GoogleFonts.notoSans(
    fontSize: 18,
    fontWeight: FontWeight.w500,
  ),
),
const SizedBox(
  height: 15,
),
Text(
  'Hide Pin',
  style: GoogleFonts.notoSans(
    fontSize: 18,
    fontWeight: FontWeight.w500,
  ),
),
const SizedBox(
  height: 15,
),
Text(
  'Report Pin',
  style: GoogleFonts.notoSans(
    fontSize: 18,
    fontWeight: FontWeight.w500,
  ),
),
Text(
  "This goes against Pinterest's community guidelines",
  style: GoogleFonts.notoSans(
    fontSize: 14,
  ),
```

```
                        ),
                      ],
                    ),
                  ),
                );
              },
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(20.0),
              ),
            );
          },
          child: const Icon(
            CupertinoIcons.ellipsis,
            color: Colors.white,
            size: 28,
          ),
        ),
      ],
    ),
    CircleAvatar(
      backgroundColor: Colors.white.withOpacity(0.8),
      child: const Icon(
        CupertinoIcons.viewfinder,
        color: Colors.black,
      ),
    )
  ],
 ),
 ),
 ),
),
Container(
 padding: const EdgeInsets.only(
  top: 10,
  bottom: 30,
  left: 18,
  right: 18,
 ),
 color: Colors.white,
 child: Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: [
   const Icon(CupertinoIcons.heart_circle_fill),
   Row(
    children: [
```

```
                Container(
                  padding: const EdgeInsets.symmetric(
                      horizontal: 17, vertical: 15),
                  decoration: BoxDecoration(
                    color: const Color(0xFFF1F1F1),
                    borderRadius: BorderRadius.circular(30),
                  ),
                  child: Text(
                    'View',
                    style: GoogleFonts.notoSans(
                      fontWeight: FontWeight.w500,
                    ),
                  ),
                ),
                const SizedBox(
                  width: 6,
                ),
                Container(
                  padding: const EdgeInsets.symmetric(
                      horizontal: 17, vertical: 15),
                  decoration: BoxDecoration(
                    color: Theme.of(context).colorScheme.secondary,
                    borderRadius: BorderRadius.circular(30),
                  ),
                  child: Text(
                    'Save',
                    style: GoogleFonts.notoSans(
                      fontWeight: FontWeight.w500,
                      color: Colors.white,
                    ),
                  ),
                ),
              ],
            ),
            const Icon(Icons.share),
          ],
        ),
      ),
    ],
  ),
 );
 }
}
```

App UI:



Widgets used: Icons, font, bottom navigation bar, image

Conclusion: Thus, understood the use of basic common widgets used in Mobile App Development and used some of them to create the login page for the chosen mini project application.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Experiment 3

Name:Piyush Tilokani
Div: D15A
Roll no: 63

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

Theory:

1. Text Widget:
    - The Text widget is used to display textual content within a Flutter application.
    - It allows you to customize the appearance of text, including font family, size, weight, style, color, alignment, and more.
    - Text widgets support both single-line and multi-line text.
    - You can use Text widgets within various Flutter layout widgets such as Column, Row, ListView, etc., to display text in different parts of the screen.
    - Text widgets can also be styled dynamically using theming or state management techniques.

2. Button Widget:
    - Flutter provides several types of buttons, including ElevatedButton, TextButton, OutlinedButton, and IconButton.
    - Buttons are interactive elements that users can tap or click to trigger actions or events in the application.
    - Each type of button has its own style and appearance, but they all support customization of properties such as text, color, padding, shape, and onPressed callback.
    - Buttons can be placed within Flutter layout widgets like Row, Column, Container, etc., to create interactive user interfaces.
    - Flutter buttons can also be disabled or enabled based on certain conditions, and their appearance can be adjusted accordingly.

3. Image Widget:
    - The Image widget is used to display images within a Flutter application.
    - It supports various image formats such as JPEG, PNG, GIF, WebP, and SVG (using the flutter_svg package).
    - Images can be loaded from different sources including local assets, network URLs, memory, and file paths.
    - The Image widget provides properties to control the image's size, alignment, fit, repeat mode, color filters, and more.
    - Flutter also provides advanced features for image caching, resizing, and processing to optimize performance and memory usage.
    - Images are often used to enhance the visual appeal of an application and to convey information to the user through graphics and icons.

Code:

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:faker/faker.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';
import 'package:google_fonts/google_fonts.dart';

class DetailPage extends StatelessWidget {
  const DetailPage({super.key, required this.imageUrl});
  final String imageUrl;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Column(
        children: [
          Expanded(
            child: Container(
              decoration: BoxDecoration(
                image: DecorationImage(
                  image: CachedNetworkImageProvider(imageUrl),
                  fit: BoxFit.cover,
                ),
              ),
              child: SafeArea(
                child: Padding(
                  padding: const EdgeInsets.symmetric(horizontal: 18),
                  child: Column(
                    mainAxisAlignment: MainAxisAlignment.spaceBetween,
                    crossAxisAlignment: CrossAxisAlignment.end,
                    children: [
                      Row(
                        mainAxisAlignment: MainAxisAlignment.spaceBetween,
                        children: [
                          InkWell(
                            onTap: () => Navigator.pop(context),
                            child: CircleAvatar(
                              backgroundColor: Colors.black.withOpacity(0.2),
                              child: const Icon(
                                CupertinoIcons.chevron_back,
                                color: Colors.white,
```

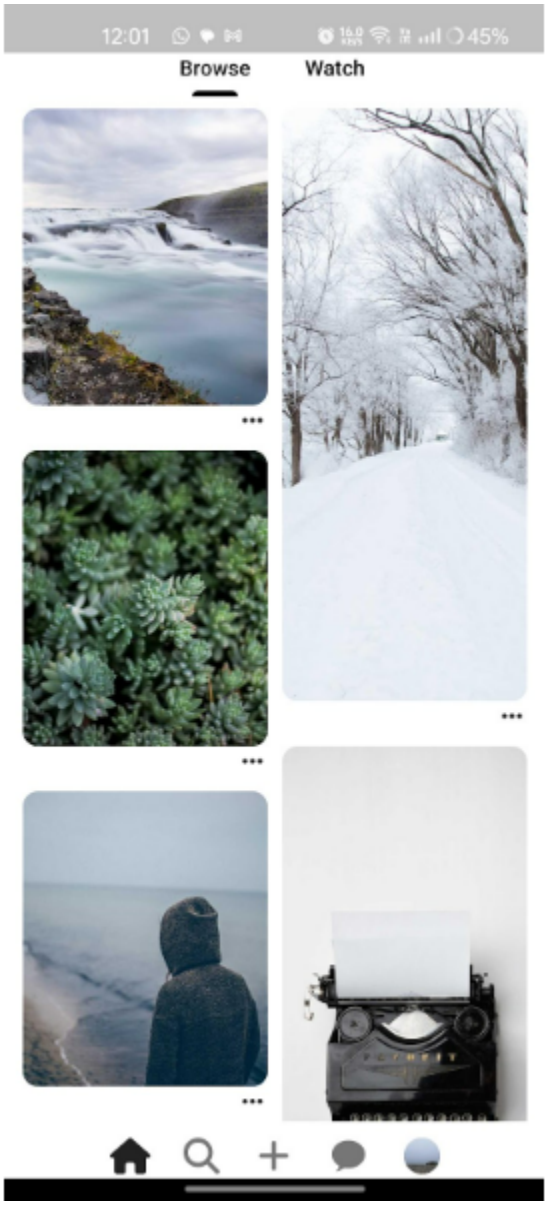
```
      size: 28,
    ),
  ),
),
InkWell(
  onTap: () {
    showModalBottomSheet(
      context: context,
      builder: (context) {
        return SizedBox(
          height: 330,
          child: Padding(
            padding: const EdgeInsets.symmetric(
                horizontal: 12, vertical: 16),
            child: Column(
              crossAxisAlignment:
                  CrossAxisAlignment.start,
              children: [
                Row(
                  crossAxisAlignment:
                      CrossAxisAlignment.center,
                  children: [
                    const Icon(
                        FontAwesomeIcons.xmark),
                    const SizedBox(
                      width: 12,
                    ),
                    Text(
                      'Options',
                      style: GoogleFonts.notoSans(),
                    ),
                  ],
                ),
                const SizedBox(
                  height: 30,
                ),
                Text(
                  'Follow ${Faker().internet.userName()}',
                  style: GoogleFonts.notoSans(
                    fontSize: 18,
                    fontWeight: FontWeight.w500,
                  ),
                ),
                const SizedBox(
                  height: 15,
                ),
```

```
        Text(
          'Copy link',
          style: GoogleFonts.notoSans(
            fontSize: 18,
            fontWeight: FontWeight.w500,
          ),
        ),
        const SizedBox(
          height: 15,
        ),
        Text(
          'Download image',
          style: GoogleFonts.notoSans(
            fontSize: 18,
            fontWeight: FontWeight.w500,
          ),
        ),
        const SizedBox(
          height: 15,
        ),
        Text(
          'Hide Pin',
          style: GoogleFonts.notoSans(
            fontSize: 18,
            fontWeight: FontWeight.w500,
          ),
        ),
        const SizedBox(
          height: 15,
        ),
        Text(
          'Report Pin',
          style: GoogleFonts.notoSans(
            fontSize: 18,
            fontWeight: FontWeight.w500,
          ),
        ),
        Text(
          "This goes against Pinterest's community guidelines",
          style: GoogleFonts.notoSans(
            fontSize: 14,
          ),
        ),
      ],
    ),
  ),
),
```

```
                    );
                  },
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(20.0),
                  ),
                );
              },
              child: const Icon(
                CupertinoIcons.ellipsis,
                color: Colors.white,
                size: 28,
              ),
            ),
          ],
        ),
        CircleAvatar(
          backgroundColor: Colors.white.withOpacity(0.8),
          child: const Icon(
            CupertinoIcons.viewfinder,
            color: Colors.black,
          ),
        )
      ],
    ),
  ),
),
Container(
  padding: const EdgeInsets.only(
    top: 10,
    bottom: 30,
    left: 18,
    right: 18,
  ),
  color: Colors.white,
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      const Icon(CupertinoIcons.heart_circle_fill),
      Row(
        children: [
          Container(
            padding: const EdgeInsets.symmetric(
                horizontal: 17, vertical: 15),
            decoration: BoxDecoration(
```

```
                    color: const Color(0xFFF1F1F1),
                    borderRadius: BorderRadius.circular(30),
                  ),
                  child: Text(
                    'View',
                    style: GoogleFonts.notoSans(
                      fontWeight: FontWeight.w500,
                    ),
                  ),
                ),
                const SizedBox(
                  width: 6,
                ),
                Container(
                  padding: const EdgeInsets.symmetric(
                      horizontal: 17, vertical: 15),
                  decoration: BoxDecoration(
                    color: Theme.of(context).colorScheme.secondary,
                    borderRadius: BorderRadius.circular(30),
                  ),
                  child: Text(
                    'Save',
                    style: GoogleFonts.notoSans(
                      fontWeight: FontWeight.w500,
                      color: Colors.white,
                    ),
                  ),
                ),
              ],
            ),
            const Icon(Icons.share),
          ],
        ),
      ),
    ],
  ),
 );
 }
}
```

App UI:



Widgets used: Image, Text, Icons,

Conclusion: Thus, understood the use of Icons, images and font widgets in Flutter. Implemented Icons, Images and fonts in my Flutter application.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Experiment 4

Name:Piyush Tilokani
Div: D15A
Roll no: 63

Aim: To create an interactive form using the form widget
Theory:

1. Form Widget:
    - The Form widget is a container used to group multiple form fields together.
    - It helps manage the state of the form, including validation, submission, and resetting.
    - The Form widget maintains a FormState object that holds the current state of the form fields.
    - Form widgets facilitate form submission, validation, and error handling.

2. FormField Widget:
    - A FormField widget represents a single form field within a Form.
    - Flutter provides various subclasses of the FormField widget for different types of input fields, such as TextFormField, CheckboxFormField, RadioFormField, DropdownButtonFormField, etc.
    - Each form field widget encapsulates the logic for validating user input and managing its state.
    - Form fields can be customized with properties to specify validation rules, error messages, initial values, input formatting, and more.
    - Form fields automatically register themselves with the Form widget and handle validation and state management transparently.

3. Validation:
    - Flutter's form widgets include built-in support for validation to ensure that user input meets specific criteria.
    - Form fields can be configured with validation functions or validators to check the correctness of user input.
    - Validators can be synchronous or asynchronous functions that return error messages if the input is invalid.
    - Flutter provides a FormFieldState class associated with each form field, which exposes methods to validate the field's value and retrieve validation errors if any.

4. Submission:
    - The Form widget provides a mechanism to submit the form data once it's been filled out by the user.
    - Developers can define an onSaved callback for each form field to specify how the field's value should be processed when the form is submitted.

- When the form is submitted, the onSaved callbacks for all form fields are invoked, allowing developers to collect, process, and submit the form data to a backend server or perform other actions.

signup_page.dart

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

class SignupPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SingleChildScrollView( // Wrap with SingleChildScrollView
        child: Container(
          padding: EdgeInsets.symmetric(horizontal: 20.0, vertical: 100),
          decoration: BoxDecoration(
            gradient: LinearGradient(
              begin: Alignment.topLeft,
              end: Alignment.bottomRight,
              colors: [Colors.black12, Colors.white12],
            ),
          ),
          child: Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Image.asset(
                  'assets/pinterest_logo.png', // Replace with your image file path
                  width: 150.0,
                  height: 150.0,
                ),
                SizedBox(height: 16.0),
                RichText(
                  text: TextSpan(
                    text: 'Welcome to ',
                    style: TextStyle(
                      color: Colors.black,
                      fontSize: 30.0,
                      fontWeight: FontWeight.bold,
                    ),
                    children: [
                      TextSpan(
                        text: 'Pinterest',
                        style: TextStyle(
```

```
            color: Colors.red, // Change the color of "Pinterest" to red
            fontSize: 30.0,
            fontWeight: FontWeight.bold,
          ),
        ),
      ],
    ),
  ),
),
SizedBox(height: 16.0),
TextField(
  decoration: InputDecoration(
    prefixIcon: Icon(Icons.email, color: Colors.black54),
    hintText: 'Email',
    hintStyle: TextStyle(color: Colors.black54),
    filled: true,
    fillColor: Colors.white.withOpacity(0.8),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(12.0),
      borderSide: BorderSide.none,
    ),
  ),
  style: TextStyle(color: Colors.black54),
),
SizedBox(height: 12.0),
TextField(
  decoration: InputDecoration(
    prefixIcon: Icon(Icons.account_circle_outlined, color: Colors.black54),
    hintText: 'Username',
    hintStyle: TextStyle(color: Colors.black54),
    filled: true,
    fillColor: Colors.white.withOpacity(0.8),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(12.0),
      borderSide: BorderSide.none,
    ),
  ),
  style: TextStyle(color: Colors.black54),
),
SizedBox(height: 14.0),TextField(
  obscureText: true,
  decoration: InputDecoration(
    prefixIcon: Icon(Icons.lock, color: Colors.black54),
    hintText: 'Password',
    hintStyle: TextStyle(color: Colors.black54),
    filled: true,
    fillColor: Colors.white.withOpacity(0.8),
```

```
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(12.0),
      borderSide: BorderSide.none,
    ),
  ),
  style: TextStyle(color: Colors.black54),
),
SizedBox(height: 20.0),
ElevatedButton(
  onPressed: () {
    // Add your login logic here
  },
  child: Text(
    'Continue',
    style: TextStyle(
      color: Colors.white,
      fontSize: 18,
    ),
  ),
  style: ElevatedButton.styleFrom(
    backgroundColor: Colors.red,
    padding: EdgeInsets.symmetric(horizontal: 80.0),
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(20.0),
    ),
  ),
),
SizedBox(height: 16.0),

TextButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => LoginPage()),
    );
  },
  child: RichText(
    text: TextSpan(
      text: 'Already have an account? ',
      style: TextStyle(color: Colors.black54, fontSize: 16),
      children: <TextSpan>[
        TextSpan(
          text: 'Login',
          style: TextStyle(color: Colors.red, fontSize: 16),
        ),
      ],
```

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

login_page.dart

```
import 'package:flutter/material.dart';
import 'package:pinterest/main_page.dart';
import './signup_page.dart';


class LoginPage extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   body: SingleChildScrollView( // Wrap with SingleChildScrollView
    child: Container(
     padding: EdgeInsets.symmetric(horizontal: 20.0, vertical: 150),
     decoration: BoxDecoration(
      gradient: LinearGradient(
       begin: Alignment.topLeft,
       end: Alignment.bottomRight,
       colors: [Colors.black12, Colors.white12],
      ),
     ),
     child: Center(
      child: Column(
       mainAxisAlignment: MainAxisAlignment.center,
       children: [
        Image.asset(
         'assets/pinterest_logo.png', // Replace with your image file path
         width: 150.0,
         height: 150.0,
        ),
        SizedBox(height: 16.0),
        RichText(
         text: TextSpan(
```

```
    text: 'Welcome to ',
    style: TextStyle(
      color: Colors.black,
      fontSize: 30.0,
      fontWeight: FontWeight.bold,
    ),
    children: [
      TextSpan(
        text: 'Pinterest',
        style: TextStyle(
          color: Colors.red, // Change the color of "Pinterest" to red
          fontSize: 30.0,
          fontWeight: FontWeight.bold,
        ),
      ),
    ],
  ),
),
SizedBox(height: 16.0),
TextField(
  decoration: InputDecoration(
    prefixIcon: Icon(Icons.email, color: Colors.black54),
    hintText: 'Email',
    hintStyle: TextStyle(color: Colors.black54),
    filled: true,
    fillColor: Colors.white.withOpacity(0.8),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(12.0),
      borderSide: BorderSide.none,
    ),
  ),
  style: TextStyle(color: Colors.black54),
),
SizedBox(height: 12.0),
TextField(
  obscureText: true,
  decoration: InputDecoration(
    prefixIcon: Icon(Icons.lock, color: Colors.black54),
    hintText: 'Password',
    hintStyle: TextStyle(color: Colors.black54),
    filled: true,
    fillColor: Colors.white.withOpacity(0.8),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(12.0),
      borderSide: BorderSide.none,
    ),
```

```
  ),
  style: TextStyle(color: Colors.black54),
),
SizedBox(height: 20.0),
ElevatedButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => MainPage()),
    );
  },
  child: Text(
    'Continue',
    style: TextStyle(
      color: Colors.white,
      fontSize: 18,
    ),
  ),
  style: ElevatedButton.styleFrom(
    backgroundColor: Colors.red,
    padding: EdgeInsets.symmetric(horizontal: 80.0),
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(20.0),
    ),
  ),
),
SizedBox(height: 16.0),

SizedBox(height: 5.0),
TextButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => SignupPage()),
    );
  },
  child: RichText(
    text: TextSpan(
      text: 'Don\'t have an account? ',
      style: TextStyle(color: Colors.black54, fontSize: 16),
      children: <TextSpan>[
        TextSpan(
          text: 'Sign up',
          style: TextStyle(color: Colors.red, fontSize: 16),
        ),
      ],
```

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

Code:

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:faker/faker.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';
import 'package:google_fonts/google_fonts.dart';

class DetailPage extends StatelessWidget {
 const DetailPage({super.key, required this.imageUrl});
 final String imageUrl;

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   body: Column(
    children: [
     Expanded(
      child: Container(
       decoration: BoxDecoration(
        image: DecorationImage(
         image: CachedNetworkImageProvider(imageUrl),
         fit: BoxFit.cover,
        ),
       ),
       child: SafeArea(
        child: Padding(
         padding: const EdgeInsets.symmetric(horizontal: 18),
         child: Column(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          crossAxisAlignment: CrossAxisAlignment.end,
```

```
children: [
  Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      InkWell(
        onTap: () => Navigator.pop(context),
        child: CircleAvatar(
          backgroundColor: Colors.black.withOpacity(0.2),
          child: const Icon(
            CupertinoIcons.chevron_back,
            color: Colors.white,
            size: 28,
          ),
        ),
      ),
      InkWell(
        onTap: () {
          showModalBottomSheet(
            context: context,
            builder: (context) {
              return SizedBox(
                height: 330,
                child: Padding(
                  padding: const EdgeInsets.symmetric(
                      horizontal: 12, vertical: 16),
                  child: Column(
                    crossAxisAlignment:
                        CrossAxisAlignment.start,
                    children: [
                      Row(
                        crossAxisAlignment:
                            CrossAxisAlignment.center,
                        children: [
                          const Icon(
                              FontAwesomeIcons.xmark),
                          const SizedBox(
                            width: 12,
                          ),
                          Text(
                            'Options',
                            style: GoogleFonts.notoSans(),
                          ),
                        ],
                      ),
                      const SizedBox(
                        height: 30,
```

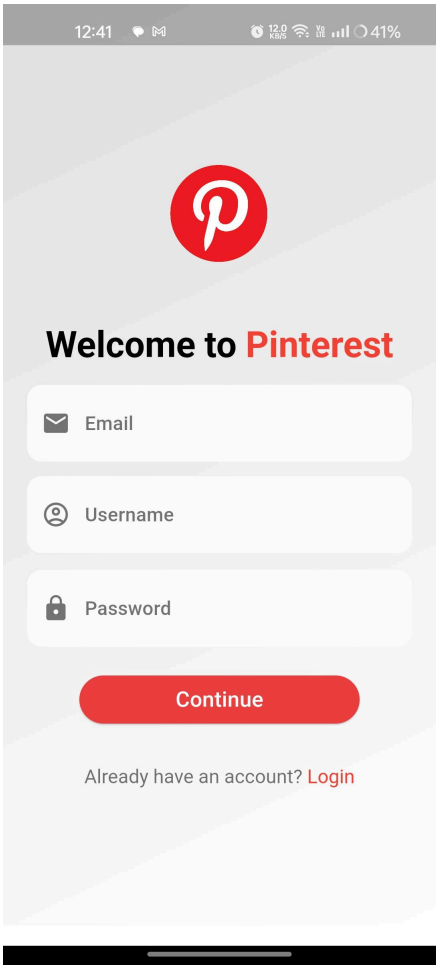
```
),
Text(
  'Follow ${Faker().internet.userName()}',
  style: GoogleFonts.notoSans(
    fontSize: 18,
    fontWeight: FontWeight.w500,
  ),
),
const SizedBox(
  height: 15,
),
Text(
  'Copy link',
  style: GoogleFonts.notoSans(
    fontSize: 18,
    fontWeight: FontWeight.w500,
  ),
),
const SizedBox(
  height: 15,
),
Text(
  'Download image',
  style: GoogleFonts.notoSans(
    fontSize: 18,
    fontWeight: FontWeight.w500,
  ),
),
const SizedBox(
  height: 15,
),
Text(
  'Hide Pin',
  style: GoogleFonts.notoSans(
    fontSize: 18,
    fontWeight: FontWeight.w500,
  ),
),
const SizedBox(
  height: 15,
),
Text(
  'Report Pin',
  style: GoogleFonts.notoSans(
    fontSize: 18,
    fontWeight: FontWeight.w500,
```

```
                    ),
                  ),
                  Text(
                    "This goes against Pinterest's community guidelines",
                    style: GoogleFonts.notoSans(
                      fontSize: 14,
                    ),
                  ),
                ],
              ),
            ),
          );
        },
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(20.0),
        ),
      );
    },
    child: const Icon(
      CupertinoIcons.ellipsis,
      color: Colors.white,
      size: 28,
    ),
  ),
 ],
),
CircleAvatar(
  backgroundColor: Colors.white.withOpacity(0.8),
  child: const Icon(
    CupertinoIcons.viewfinder,
    color: Colors.black,
  ),
)
],
),
),
),
),
),
Container(
  padding: const EdgeInsets.only(
    top: 10,
    bottom: 30,
    left: 18,
    right: 18,
  ),
```

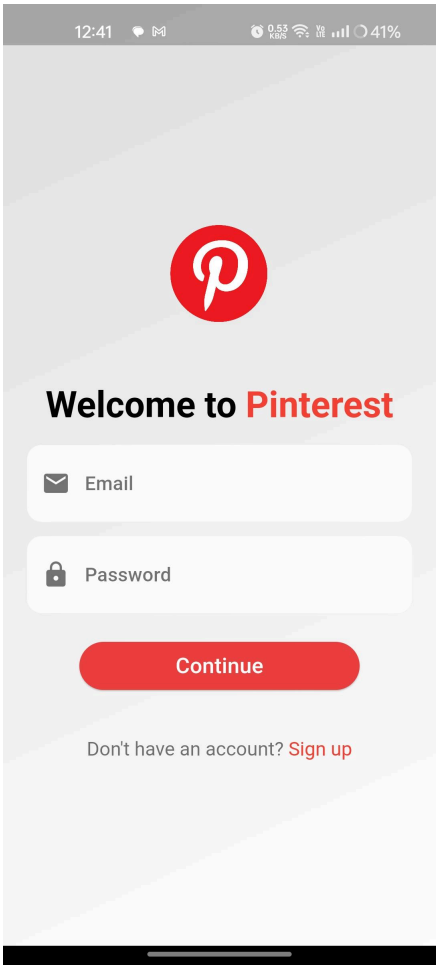
```
    color: Colors.white,
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        const Icon(CupertinoIcons.heart_circle_fill),
        Row(
          children: [
            Container(
              padding: const EdgeInsets.symmetric(
                  horizontal: 17, vertical: 15),
              decoration: BoxDecoration(
                color: const Color(0xFFF1F1F1),
                borderRadius: BorderRadius.circular(30),
              ),
              child: Text(
                'View',
                style: GoogleFonts.notoSans(
                  fontWeight: FontWeight.w500,
                ),
              ),
            ),
            const SizedBox(
              width: 6,
            ),
            Container(
              padding: const EdgeInsets.symmetric(
                  horizontal: 17, vertical: 15),
              decoration: BoxDecoration(
                color: Theme.of(context).colorScheme.secondary,
                borderRadius: BorderRadius.circular(30),
              ),
              child: Text(
                'Save',
                style: GoogleFonts.notoSans(
                  fontWeight: FontWeight.w500,
                  color: Colors.white,
                ),
              ),
            ),
          ],
        ),
        const Icon(Icons.share),
      ],
    ),
  ),
],
```

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

App UI:



Signup page



Login page

Widgets used: Form Widget, Form Widget Fields

Conclusion: Therefore understood the use of form widget in Flutter. Implemented signup and login page using form widget in my Flutter application.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Experiment 5

Name:Piyush Tilokani
Div: D15A
Roll no: 63

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

Theory:

1. Navigation:
    - Navigation refers to the process of moving between different screens or pages within a Flutter app.
    - In Flutter, navigation is typically managed using the Navigator class, which maintains a stack of routes.
    - Each route represents a screen or page in the app, and the navigator manages the navigation stack, allowing users to move forward and backward between routes.
    - Navigation can be triggered by user actions such as tapping buttons, selecting items from lists, or swiping between pages.

2. Routing:
    - Routing is the mechanism used to define and manage the routes within a Flutter app.
    - Routes are defined using route names and associated with corresponding widgets or screens.
    - Flutter provides several routing mechanisms, including named routes, on-the-fly routes, and nested routes.
    - Named routes allow developers to define routes with unique names and navigate to them using the Navigator based on these names.
    - On-the-fly routes are created dynamically at runtime and pushed onto the navigation stack as needed.
    - Nested routes involve embedding navigators within other navigators to create complex navigation structures, such as tab-based navigation or drawer navigation.

3. Gestures:
    - Gestures refer to user interactions such as tapping, dragging, swiping, pinching, and rotating on the screen.
    - Flutter provides a rich set of gesture recognition widgets and APIs to handle user gestures effectively.
    - Common gesture recognition widgets include GestureDetector, InkWell, InkResponse, Draggable, Dismissible, etc.
    - These widgets allow developers to detect various user gestures and trigger corresponding actions or animations in response.

- Gestures can be used to implement interactive UI elements, such as buttons, sliders, swipers, drag-and-drop interfaces, and more.

4. Gesture Detection:

    - Gesture detection in Flutter involves registering gesture recognizers on widgets to detect specific user interactions.
    - Gesture recognizers analyze touch input and determine whether a specific gesture has occurred, such as a tap, double-tap, long-press, drag, etc.
    - Once a gesture is detected, Flutter invokes the corresponding callback function associated with the gesture recognizer.
    - Developers can customize gesture detection by configuring properties such as gesture sensitivity, velocity thresholds, and touch area boundaries.

5. Gesture Handling:
    - After a gesture is detected, developers can handle it by performing various actions, such as updating UI state, navigating between screens, triggering animations, or executing business logic.
    - Gesture handling involves responding to user interactions in a way that provides feedback and enhances the user experience.
    - Flutter's declarative programming model makes it easy to update UI elements in response to user gestures, ensuring a smooth and responsive user interface.

home_screen.dart

```
import 'dart:math';
import 'package:animated_text_kit/animated_text_kit.dart';
import 'package:carousel_slider/carousel_slider.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:flutter_staggered_grid_view/flutter_staggered_grid_view.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:liquid_pull_to_refresh/liquid_pull_to_refresh.dart';
import 'package:pinterest_clone/screens/images_details_screen.dart';
import 'package:pinterest_clone/widget/api_call_waiting_widget.dart';
import 'package:shimmer/shimmer.dart';
import '../api/pexel_api_class.dart';
import '../model/pexel_model.dart';

import '../themes/container_random_colors.dart';
import '../widget/loading_Widget.dart';
import 'onboard_screen.dart';

class HomeScreen extends StatefulWidget {
```

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

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  int currentIndex = 1;
  ScrollController scrollController = ScrollController();
  bool isStarted = false;
  List<Photo> curatedPhotos = [];
  bool isRefreshing = false;

  final GlobalKey<LiquidPullToRefreshState> _refreshIndicatorKey =
      GlobalKey<LiquidPullToRefreshState>();

  final GlobalKey<ScaffoldState> _scaffoldKey = GlobalKey<ScaffoldState>();

  final PexelsApi pexelsApi = PexelsApi(
      apiKey: 'MsB5RufxehOuViswILug2dAoqBtlXvyrRdfA0n1GnLL2MCIMbA2eD8Sk');

  final List<String> quotes = [
    "Exploring moments, capturing dreams where every frame tells a story of "
        "timeless beauty.",
    "Seeking moments in pixels, where every frame tells a story, and each "
        "image is a masterpiece waiting to be discovered.",
    "Discover moments, capture stories. Elevate your world with curated "
        "creativity."
  ];

  void handleRefresh() async {
    setState(() {
      isRefreshing = true;
    });

    try {
      List<Photo> newCuratedPhotos =
          await pexelsApi.getCuratedPhotos(page: 1, perPage: 15);

      setState(() {
        curatedPhotos = newCuratedPhotos;
      });
    } catch (error) {
      if (kDebugMode) {
        print('Error refreshing curated photos: $error');
      }
```

```
  }

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

// void shareImage(String imageUrl, String photographer) {
//   Share.share(
//     'Check out this photo by $photographer: $imageUrl',
//     subject: 'Photo Sharing',
//   );
// }

@override
void initState() {
  super.initState();
  // Connectivity().checkConnectivity().then((ConnectivityResult result) {
  //   if (result == ConnectivityResult.none) {
  //     showNoInternetSnackbar();
  //   }
  // });
  //
  // Connectivity().onConnectivityChanged.listen((ConnectivityResult result) {
  //   if (result == ConnectivityResult.none) {
  //     showNoInternetSnackbar();
  //   }
  // });
}

// void showNoInternetSnackbar() {
//   final snackBar = SnackBar(
//     backgroundColor: Colors.transparent,
//     elevation: 0,
//     padding: const EdgeInsets.only(bottom: 750),
//     dismissDirection: DismissDirection.endToStart,
//     content: Container(
//         width: 190.w,
//         height: 56.h,
//         decoration: BoxDecoration(
//             shape: BoxShape.rectangle,
//             borderRadius: BorderRadius.circular(45),
//             color: Colors.red),
//         child: Center(
//           child: Text(
//             "Hmm... you're not connected to the \n internet",
```

```
//             style: GoogleFonts.poppins(
//                 color: Colors.black,
//                 fontWeight: FontWeight.w500,
//                 fontSize: 17.sp),
//             textAlign: TextAlign.center,
//           ),
//         )),
//     duration: const Duration(seconds: 2),
//   );
//
//   ScaffoldMessenger.of(context).showSnackBar(snackBar);
// }

@override
Widget build(BuildContext context) {
  // Connectivity().checkConnectivity().then((ConnectivityResult result) {
  //   if (result == ConnectivityResult.none) {
  //     showNoInternetSnackbar();
  //   }
  // });

  return LiquidPullToRefresh(
    key: _refreshIndicatorKey,
    onRefresh: () async {
      handleRefresh;
    },
    showChildOpacityTransition: true,
    color: Colors.grey.shade900,
    springAnimationDurationInMilliseconds: 800,
    backgroundColor: Colors.grey.shade600,
    child: Scaffold(
      key: _scaffoldKey,
      backgroundColor: Colors.transparent,
      body: NestedScrollView(
        physics: const AlwaysScrollableScrollPhysics(),
        headerSliverBuilder: (BuildContext context, bool isScrolled) {
          return [
            SliverAppBar(
              expandedHeight: 260.h,
              pinned: false,
              flexibleSpace: Stack(children: [
                FutureBuilder<List<Photo>>(
                  future: pexelsApi.getCuratedPhotos(page: 1, perPage: 10),
                  builder: (context, snapshot) {
                    if (snapshot.connectionState == ConnectionState.waiting) {
                      return Shimmer.fromColors(
```

```
    baseColor: Colors.grey[300]!,
    highlightColor: Colors.grey[100]!,
    child: CarouselSlider(
      options: CarouselOptions(
        height: 380.h,
        initialPage: 1,
        enlargeCenterPage: true,
        viewportFraction: 1,
        enlargeStrategy: CenterPageEnlargeStrategy.scale,
        enableInfiniteScroll: true,
      ),
      items: List.generate(3, (index) {
        return Container(
          decoration: BoxDecoration(
            color: Colors.grey.shade800,
          ),
        );
      }),
    ),
  );
} else if (snapshot.hasError) {
  return Text('Error: ${snapshot.error}');
} else if (snapshot.hasData) {
  final List<Photo> curatedPhotos = snapshot.data!;
  final middleIndex = curatedPhotos.length ~/ 2;
  return CarouselSlider(
    options: CarouselOptions(
      height: 380.h,
      initialPage: middleIndex,
      autoPlay: true,
      viewportFraction: 1,
      enlargeStrategy: CenterPageEnlargeStrategy.scale,
      enableInfiniteScroll: true,
    ),
    items: curatedPhotos.map((photo) {
      return Container(
        decoration: BoxDecoration(
          color: Colors.grey.shade800,
          image: DecorationImage(
            fit: BoxFit.cover,
            image: NetworkImage(photo.src['large']),
          ),
        ),
      );
    }).toList(),
  );
```

```
            } else {
              return YourLoadingWidget(
                child: Center(
                  child: Text(
                    "Error Loading",
                    style: GoogleFonts.poppins(
                        color: Colors.white,
                        fontWeight: FontWeight.w500),
                  ),
                ),
              );
            }
          },
        ),
        Center(
           child: AnimatedTextKit(
          animatedTexts: quotes.map((quote) {
            return TypewriterAnimatedText(
              quote,
              textStyle: GoogleFonts.poppins(
                color: Colors.white,
                fontSize: 20.sp,
                fontWeight: FontWeight.w500,
              ),
              speed: const Duration(milliseconds: 100),
              textAlign: TextAlign.center,
            );
          }).toList(),
        )),
      ]),
    ),
  ];
},
body: Padding(
  padding: const EdgeInsets.all(8.0),
  child: SizedBox(
    height: MediaQuery.of(context).size.height,
    child: FutureBuilder<List<Photo>>(
      future: pexelsApi.getCuratedPhotos(page: 3, perPage: 150),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return const WaitingContainer();
        } else if (snapshot.hasError) {
          return Center(
            child: Text('Error: ${snapshot.error}'),
          );
```

```
} else if (snapshot.hasData) {
  final List<Photo> curatedPhotos = snapshot.data!;
  return MasonryGridView.builder(
     mainAxisSpacing: 10.0,
     crossAxisSpacing: 10.0,
     physics: const AlwaysScrollableScrollPhysics(),
     itemCount: curatedPhotos.length,
     itemBuilder: (context, index) {
      final photo = curatedPhotos[index];
      final double imageAspectRatio =
         photo.width / photo.height;
      return GestureDetector(
         onTap: () {
          Navigator.of(context).push(PageRouteBuilder(
            pageBuilder: (context, animation,
                  secondaryAnimation) =>
              ImageDetailsScreen(
             photo: photo,
             curatedPhotos: curatedPhotos,
             initialIndex: index,
            ),
            transitionsBuilder: (context, animation,
               secondaryAnimation, child) {
             const begin = Offset(0.0, 1.0);
             const end = Offset.zero;
             const curve = Curves.easeInOut;

             var tween = Tween(begin: begin, end: end)
                .chain(CurveTween(curve: curve));
             var offsetAnimation =
                animation.drive(tween);

             return SlideTransition(
                position: offsetAnimation,
                child: child);
            },
            transitionDuration:
               const Duration(milliseconds: 500),
          ));
         },
         child: Column(
          children: [
            AspectRatio(
             aspectRatio: imageAspectRatio,
             child: Container(
              margin: const EdgeInsets.all(5),
```

```
    decoration: BoxDecoration(
      borderRadius: BorderRadius.circular(15),
      color: ColorList.colorList[Random().nextInt(ColorList.colorList.length)],
      image: DecorationImage(
        fit: BoxFit.cover,
        image: NetworkImage(
            photo.src["large"] ?? ''),
      ),
    ),
    child: (photo.url != null)
        ? null
        : const Center(
            child: Icon(
              Icons.image_not_supported,
              size: 50,
              color: Colors.white,
            ),
          ),
  ),
),
Row(
  mainAxisAlignment:
      MainAxisAlignment.spaceBetween,
  children: [
    Flexible(
      child: Text(
        photo.alt,
        style: GoogleFonts.poppins(
          fontWeight: FontWeight.w500,
          fontSize: 14.sp,
          color: Colors.white,
        ),
      ),
    ),
    IconButton(
      icon: const Icon(
        Icons.more_horiz_sharp,
        color: Colors.white,
      ),
      onPressed: () {
        // shareImage(photo.src["large"] ?? '',
        //     photo.photographer);
      },
    ),
  ],
),
```

```
                        ],
                      ));
                },
                gridDelegate:
                    const SliverSimpleGridDelegateWithFixedCrossAxisCount(
                        crossAxisCount: 2));
            } else {
              return YourLoadingWidget(
                child: Container(),
              );
            }
          },
        ),
      ),
    ),
  ),
 ),
 );
 }

}
```

search_screen.dart

```
import 'dart:math';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:flutter_staggered_grid_view/flutter_staggered_grid_view.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:pinterest_clone/screens/search_Screen_bottomsheet.dart';
import 'package:pinterest_clone/widget/api_call_waiting_widget.dart';
import '../api/pexel_api_class.dart';
import '../model/pexel_model.dart';
import '../themes/container_random_colors.dart';
import '../widget/loading_Widget.dart';
import '../widget/loading_indicator.dart';
import 'images_details_screen.dart';

class SearchScreen extends StatefulWidget {
  const SearchScreen({Key? key}) : super(key: key);

  @override
```

```
  State<SearchScreen> createState() => _SearchScreenState();
}

class _SearchScreenState extends State<SearchScreen> {
  List<Photo> _searchResults = [];
  bool _isLoading = false;
  // void shareImage(String imageUrl, String photographer) {
  //   Share.share(
  //     'Check out this photo by $photographer: $imageUrl',
  //     subject: 'Photo Sharing',
  //   );
  // }
  @override
  void initState() {
    super.initState();
  }
  final PexelsApi pexelsApi = PexelsApi(
      apiKey: 'MsB5RufxehOuViswILug2dAoqBtlXvyrRdfA0n1GnLL2MCIMbA2eD8Sk');

  Future<void> _searchPhotos(String query) async {
    try {
      // Existing code...

      final List<Photo> results = await PexelsApi(
          apiKey:
          'MsB5RufxehOuViswILug2dAoqBtlXvyrRdfA0n1GnLL2MCIMbA2eD8Sk')
          .searchPhotos(query);

      if (kDebugMode) {
        print("Search Results: $results");
      } // Add this line for debugging

      setState(() {
        _searchResults = results;
        _isLoading = false; // Move this line inside setState to ensure proper state update.
      });
    } catch (e) {
      // Handle error
      if (kDebugMode) {
        print("Error during search: $e");
      } // Add this line for debugging
      if (kDebugMode) {
        print(e.toString());
      }
      setState(() {
        _isLoading = false;
```

```
    });
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.black,
    appBar: AppBar(
      title: GestureDetector(
        onTap: () {
          showModalBottomSheet(
            context: context,
            isDismissible: true,
            isScrollControlled: true,
            useSafeArea: true,
            shape: const RoundedRectangleBorder(
                borderRadius: BorderRadius.only(
                  topLeft: Radius.circular(15),
                  topRight: Radius.circular(15),
                )),
            builder: (BuildContext context) {
              return StatefulBuilder(
                builder: (BuildContext context,
                    void Function(void Function()) setState) {
                  return _searchBottomSheet();
                },
              );
            },
          );
        },
        child: Container(
          width: double.infinity,
          height: 46.h,
          padding: const EdgeInsets.all(8),
          decoration: BoxDecoration(
              shape: BoxShape.rectangle,
              borderRadius: BorderRadius.circular(45),
              color: Colors.grey.shade900),
          child:  Row(
            children: [
              Icon(Icons.search, color: Colors.grey.shade300,),
              Text(
                "Search Pinterest",
                style: GoogleFonts.poppins(
                    color: Colors.grey.shade300,
```

```
                fontWeight: FontWeight.w500,
                fontSize: 17.sp),
            textAlign: TextAlign.center,
          ),
        ],
      ),
    ),
  ),
  backgroundColor: Colors.black,

),
body: Padding(
  padding: const EdgeInsets.all(8.0),
  child: SizedBox(
    height: MediaQuery.of(context).size.height,
    child: FutureBuilder<List<Photo>>(
      future: pexelsApi.getCuratedPhotos(page: 5, perPage: 150),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return const WaitingContainer();
        } else if (snapshot.hasError) {
          return Center(
            child: Text('Error: ${snapshot.error}'),
          );
        } else if (snapshot.hasData) {
          final List<Photo> curatedPhotos = snapshot.data!;
          return MasonryGridView.builder(
              mainAxisSpacing: 10.0,
              crossAxisSpacing: 10.0,
              physics: const AlwaysScrollableScrollPhysics(),
              itemCount: curatedPhotos.length,
              itemBuilder: (context, index) {
                final photo = curatedPhotos[index];
                final double imageAspectRatio =
                    photo.width / photo.height;
                return GestureDetector(
                    onTap: () {
                      Navigator.of(context).push(PageRouteBuilder(
                        pageBuilder: (context, animation,
                            secondaryAnimation) =>
                            ImageDetailsScreen(
                              photo: photo,
                              curatedPhotos: curatedPhotos,
                              initialIndex: index,
                            ),
                        transitionsBuilder: (context, animation,
```

```
        secondaryAnimation, child) {
      const begin = Offset(0.0, 1.0);
      const end = Offset.zero;
      const curve = Curves.easeInOut;

      var tween = Tween(begin: begin, end: end)
          .chain(CurveTween(curve: curve));
      var offsetAnimation =
      animation.drive(tween);

      return SlideTransition(
          position: offsetAnimation,
          child: child);
    },
    transitionDuration:
    const Duration(milliseconds: 500),
  ));
},
child: Column(
  children: [
    AspectRatio(
      aspectRatio: imageAspectRatio,
      child: Container(
        margin: const EdgeInsets.all(5),
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(15),
          color: ColorList.colorList[Random().nextInt(ColorList.colorList.length)],
          image: DecorationImage(
            fit: BoxFit.cover,
            image: NetworkImage(
                photo.src["large"] ?? ''),
          ),
        ),
        child: (photo.url != null)
            ? null
            : const Center(
          child: Icon(
            Icons.image_not_supported,
            size: 50,
            color: Colors.white,
          ),
        ),
      ),
    ),
    Row(
      mainAxisAlignment:
```

```
                        MainAxisAlignment.spaceBetween,
                        children: [
                          Flexible(
                            child: Text(
                              photo.alt,
                              style: GoogleFonts.poppins(
                                fontWeight: FontWeight.w500,
                                fontSize: 14.sp,
                                color: Colors.white,
                              ),
                            ),
                          ),
                          IconButton(
                            icon: const Icon(
                              Icons.more_horiz_sharp,
                              color: Colors.white,
                            ),
                            onPressed: () {
                              // shareImage(photo.src["large"] ?? '',
                              //     photo.photographer);
                            },
                          ),
                        ],
                      ),
                    ],
                  ));
            },
            gridDelegate:
            const SliverSimpleGridDelegateWithFixedCrossAxisCount(
                crossAxisCount: 2));
        } else {
          return YourLoadingWidget(
            child: Container(),
          );
        }
      },
    ),
  ),
 );
}
Widget _searchBottomSheet() {
  return Container(
    decoration: const BoxDecoration(
      color: Colors.black,
      borderRadius: BorderRadius.only(
```

```
    topLeft: Radius.circular(15),
    topRight: Radius.circular(15),
  ),
),
child: Padding(
  padding: const EdgeInsets.all(8.0),
  child: Column(
    children: [
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          SearchBottomSheetContent(
            onSearch: (query) {
              setState(() {
                _isLoading = true;
              });
              _searchPhotos(query).then((_) {
                setState(() {
                  _isLoading = false;
                });
              });
            },
          ),
          GestureDetector(
            onTap: () {
              Navigator.pop(context);
            },
            child: Text(
              "Cancel",
              style: GoogleFonts.poppins(
                color: Colors.white,
                fontWeight: FontWeight.w500,
                fontSize: 18.sp,
              ),
            ),
          ),
        ],
      ),
      SizedBox(height: 10.h),
      if (_isLoading)
        const Center(child: AnimatedCircularContainer())
      else
        Expanded(child: _buildSearchResults()),
    ],
  ),
),
```

```
  );
}


Widget _buildSearchResults() {
  return MasonryGridView.builder(
      mainAxisSpacing: 10.0,
      crossAxisSpacing: 10.0,
      physics: const AlwaysScrollableScrollPhysics(),
      itemCount: _searchResults.length,
      itemBuilder: (context, index) {
        final photo = _searchResults[index];
        final double imageAspectRatio =
            photo.width / photo.height;
        return GestureDetector(
            onTap: () {
              Navigator.of(context).push(PageRouteBuilder(
                pageBuilder: (context, animation,
                    secondaryAnimation) =>
                    ImageDetailsScreen(
                      photo: photo,
                      curatedPhotos: _searchResults,
                      initialIndex: index,
                    ),
                transitionsBuilder: (context, animation,
                    secondaryAnimation, child) {
                  const begin = Offset(0.0, 1.0);
                  const end = Offset.zero;
                  const curve = Curves.easeInOut;

                  var tween = Tween(begin: begin, end: end)
                      .chain(CurveTween(curve: curve));
                  var offsetAnimation =
                  animation.drive(tween);

                  return SlideTransition(
                      position: offsetAnimation,
                      child: child);
                },
                transitionDuration:
                const Duration(milliseconds: 500),
              ));
            },
            child: Column(
              children: [
                AspectRatio(
```

```
  aspectRatio: imageAspectRatio,
  child: Container(
    margin: const EdgeInsets.all(5),
    decoration: BoxDecoration(
      borderRadius: BorderRadius.circular(15),
      color: ColorList.colorList[Random().nextInt(ColorList.colorList.length)],
      image: DecorationImage(
        fit: BoxFit.cover,
        image: NetworkImage(
            photo.src["large"] ?? ''),
      ),
    ),
    child: (photo.url != null)
        ? null
        : const Center(
      child: Icon(
        Icons.image_not_supported,
        size: 50,
        color: Colors.white,
      ),
    ),
  ),
),
Row(
  mainAxisAlignment:
  MainAxisAlignment.spaceBetween,
  children: [
    Flexible(
      child: Text(
        photo.alt,
        style: GoogleFonts.poppins(
          fontWeight: FontWeight.w500,
          fontSize: 14.sp,
          color: Colors.white,
        ),
      ),
    ),
    IconButton(
      icon: const Icon(
        Icons.more_horiz_sharp,
        color: Colors.white,
      ),
      onPressed: () {
        // shareImage(photo.src["large"] ?? '',
        //     photo.photographer);
      },
```

```
                    ),
                  ],
                ),
              ],
            ));
      },
      gridDelegate:
      const SliverSimpleGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 2));
  }

}
```

upload_screen.dart

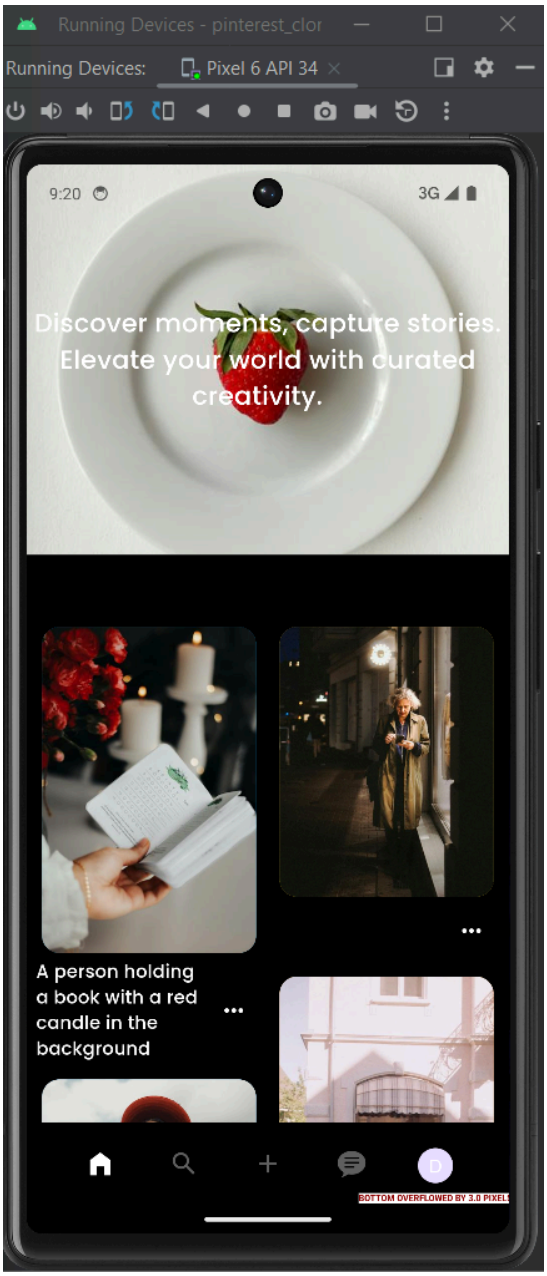
```
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';

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

  @override
  State<MessageScreen> createState() => _MessageScreenState();
}

class _MessageScreenState extends State<MessageScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.black,
      body: Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Center(
            child: Image.asset("assets/images/pinterst.png",
              width: 60.w,
              height: 60.h,),

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

message_screen.dart

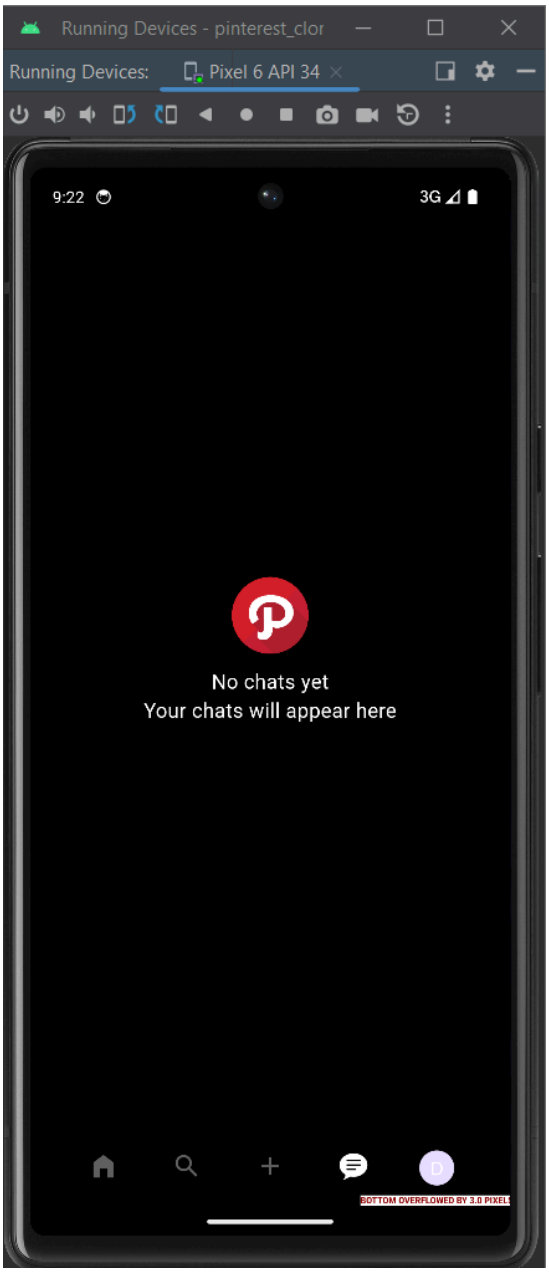
```
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';

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

  @override
  State<ProfileScreen> createState() => _ProfileScreenState();
}

class _ProfileScreenState extends State<ProfileScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.black,
      body: Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Center(
            child: Column(
              children: [
                Image.asset(
                  "assets/images/pinterst.png",
                  width: 60.w,
                  height: 60.h,
                ),
                SizedBox(height: 10.h), // Adding space between image and text
                Text(
                  'No chats yet',
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: 16.sp,
                  ),
                ),
                Text(
                  'Your chats will appear here',
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: 16.sp,
                  ),
                ),
              ],
```

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

App UI:

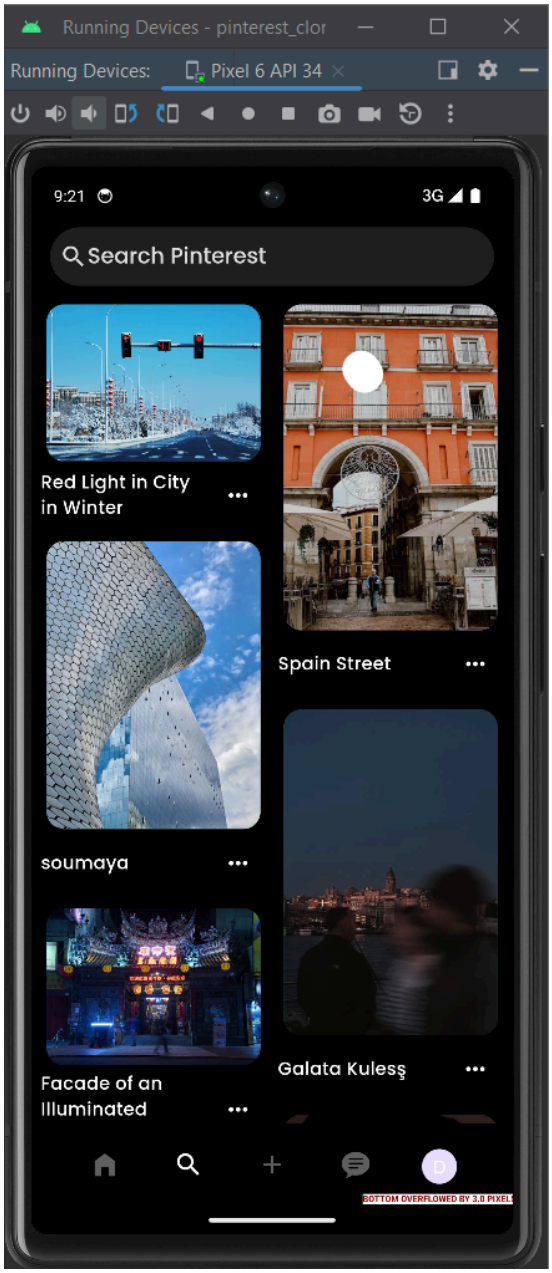


Home Page

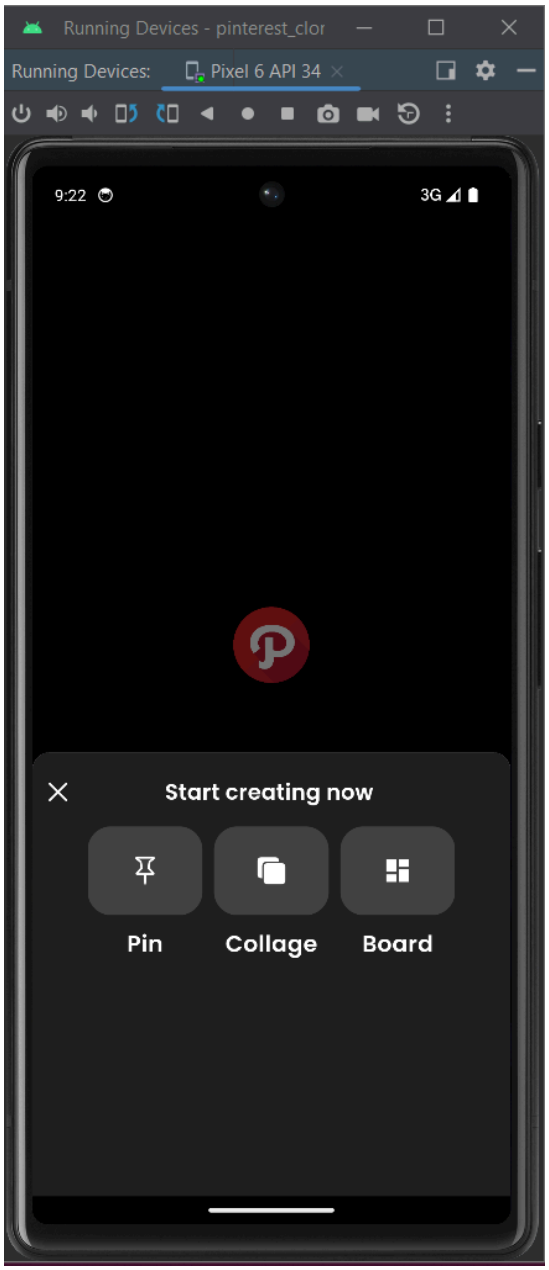


Chat page

Widgets used: Images, Text, Bottom nav bar, Icons



Search page



Upload page

Widgets used: Images, Text, Bottom nav bar, Icons, Bottom sheet, Text field

Conclusion: Therefore understood navigation, routing, gesture detection and gesture handling in Flutter and implemented the same in my Flutter application to route different pages.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Experiment 6

Name:Piyush Tilokani
Div: D15A
Roll no: 63

Aim: How To Set Up Firebase with Flutter for iOS and Android Apps

Theory:

Setting up Firebase with Flutter for iOS and Android apps involves several steps to integrate Firebase services seamlessly into your mobile application. Here's a high-level overview of the process without providing specific code:

1. Step 1: Create a Firebase Project
- Firebase Console: Go to the Firebase Console (https://console.firebase.google.com/) and create a new project.
- Add Your App: In the Firebase Console, add your Flutter app by providing its package name for Android or bundle identifier for iOS.

2. Step 2: Configure Firebase SDKs
- Download Configuration Files: After adding your app, download the configuration files for both Android (google-services.json) and iOS (GoogleService-Info.plist).
- Add Configuration Files: Place the configuration files in the appropriate directories within your Flutter project.

3. Step 3: Configure Your Project
   1. Android Configuration:
- For Android, place the google-services.json file in the android/app directory of your Flutter project.
- Configure the project-level build.gradle file to include the Google Services plugin.
   2. iOS Configuration:
- For iOS, place the GoogleService-Info.plist file in the root of your iOS project's Runner directory.
- Configure the Info.plist file of your iOS project to include necessary settings for Firebase.
4. Step 4: Integrate Firebase SDKs
- Add Firebase Plugins: Add the necessary Firebase Flutter plugins to your pubspec.yaml file. These plugins enable communication with Firebase services such as Authentication, Cloud Firestore, Cloud Messaging, etc.
- Initialize Firebase: Initialize Firebase in your Flutter app's entry point (usually main.dart) using Firebase.initializeApp() method.

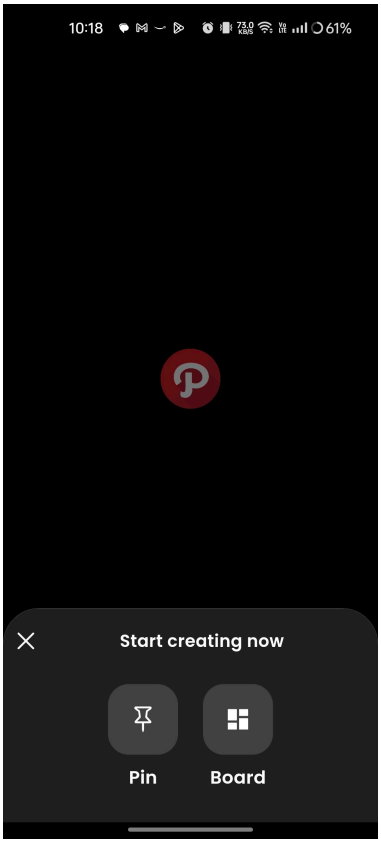
5. Step 5: Use Firebase Services
- Authentication: Implement user authentication using Firebase Authentication services. This allows users to sign up, sign in, and manage their accounts.

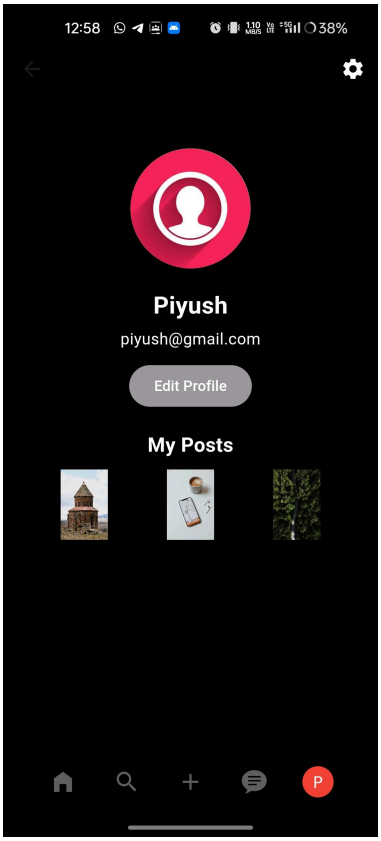
- Database: Utilize Cloud Firestore or Realtime Database to store and retrieve data from the cloud. Design your data model and interact with the database using Firebase SDK methods.
- Cloud Functions: Implement serverless backend logic using Cloud Functions for Firebase. This allows you to run custom backend code in response to events triggered by Firebase features and HTTPS requests.
- Cloud Messaging: Implement push notifications using Firebase Cloud Messaging (FCM) to engage users with timely and relevant messages.

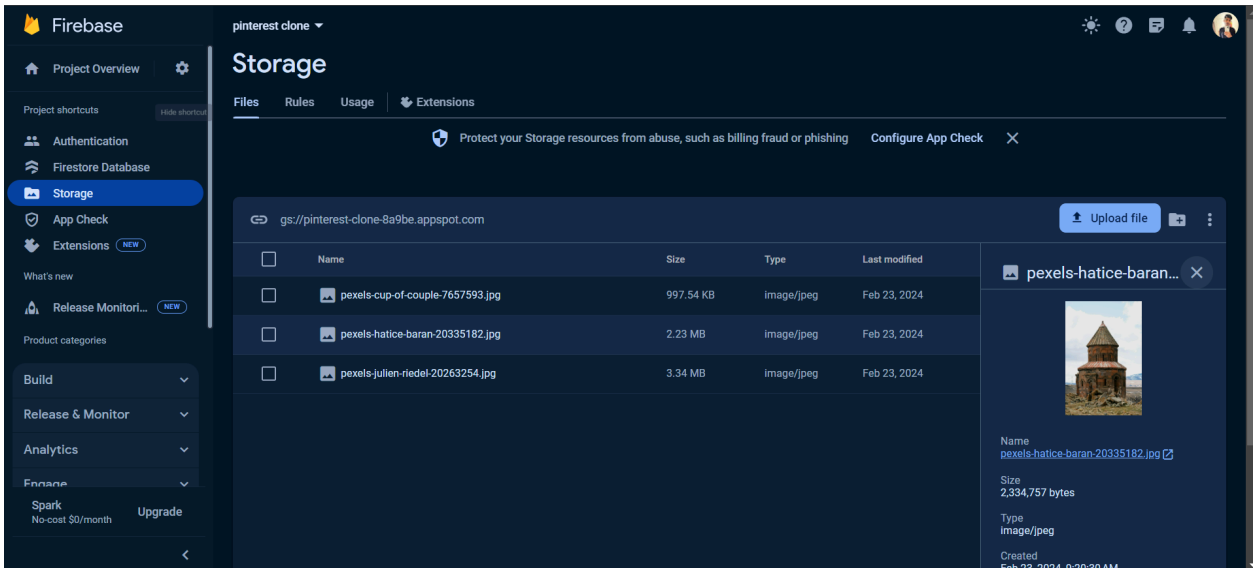
6. Step 6: Test and Deploy

- Test Your App: Test your Flutter app thoroughly to ensure that Firebase services are integrated correctly and functioning as expected.
- Deploy Your App: Deploy your Flutter app to the Google Play Store for Android or the Apple App Store for iOS, making it available to users.
- By following these steps, you can successfully set up Firebase with Flutter for both iOS and Android apps, enabling powerful features and capabilities to enhance your application.

App UI:







References: https://github.com/MujtabaNasir/Pinterest_Clone_Flutter
https://iconscout.com/all-assets/pinterest

Conclusion: Therefore understood setup of firebase and how to use it our flutter app

Project Title: **Roll No.**

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.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

Experiment 7

Name:Piyush Tilokani
Div: D15A
Roll no: 63

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

Theory:-

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

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

- Difference between PWAs vs. Regular Web Apps:

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

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

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

4. Engaging Approach

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

5. Updated Real-Time Data Access
Another plus point of PWAs is that these apps get updated on their own. They do not demand the end-users to go to the App Store or other such platforms to download the update and wait until installed.
In this app type, the web app developers can push the live update from the server, which reaches the apps residing on the user's devices automatically. Therefore, it is easier for the mobile app developer to provide the best of the updated functionalities and services to the end-users without forcing them to update their app.
6. Discoverable
PWAs reside in web browsers. This implies higher chances of optimizing them as per the Search Engine Optimization (SEO) criteria and improving the Google rankings like that in websites and other web apps.

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

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

- Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.
- Responsive — They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can later become available.
- App-like — They behave with the user as if they were native apps, in terms of interaction and navigation.
- Updated — Information is always up-to-date thanks to the data update process offered by service workers.
- Secure — Exposed over HTTPS protocol to prevent the connection from displaying information or altering the contents.
- Searchable — They are identified as “applications” and are indexed by search engines.
- Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
- Installable — They allow the user to “save” the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to face all the steps and problems related to the use of the app store.
- Linkable — Easily shared via URL without complex installations.
- Offline — Once more it is about putting the user before everything, avoiding the usual error message in case of weak or no connection. The PWA are based on two

particularities: first of all the ‘skeleton’ of the app, which recalls the page structure, even if its contents do not respond and its elements include the header, the page layout, as well as an illustration that signals that the page is loading.

Weaknesses refer to:

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

Code:
index.html of the react app

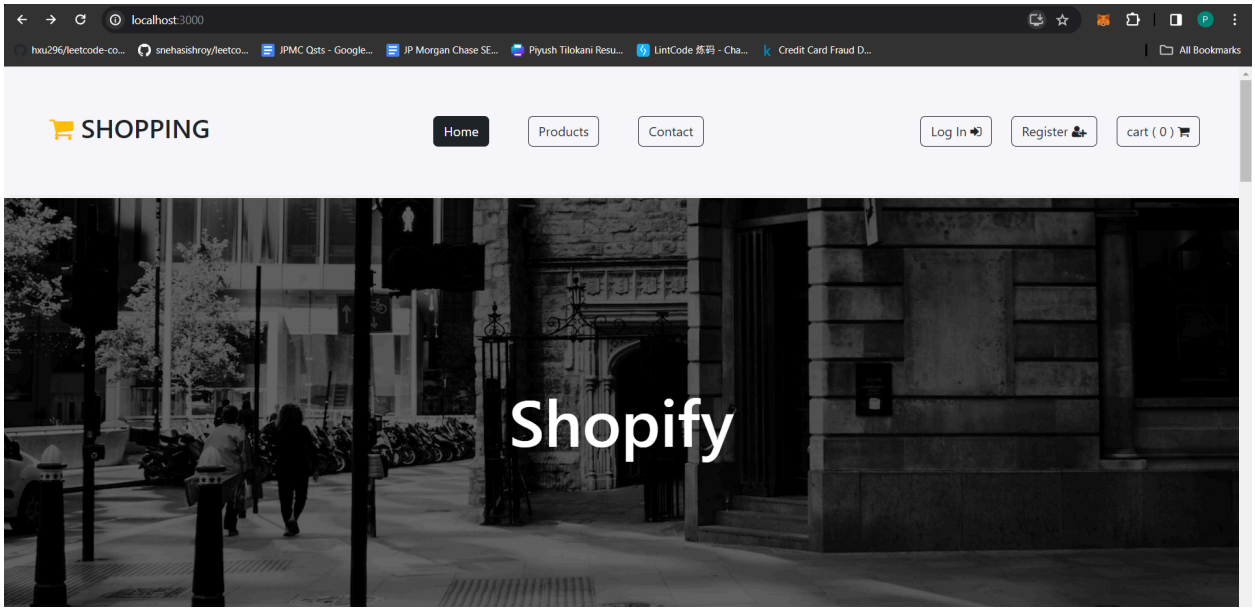
```
<!DOCTYPE html>
<html lang="en">

<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" />
  <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

  <meta charset="utf-8">
  <meta name="viewport"
      content="width=device-width,
            initial-scale=1">
  <meta http-equiv="X-UA-Compatible"
      content="ie=edge">

  <!-- Title -->
  <title>PWA Tutorial</title>
```

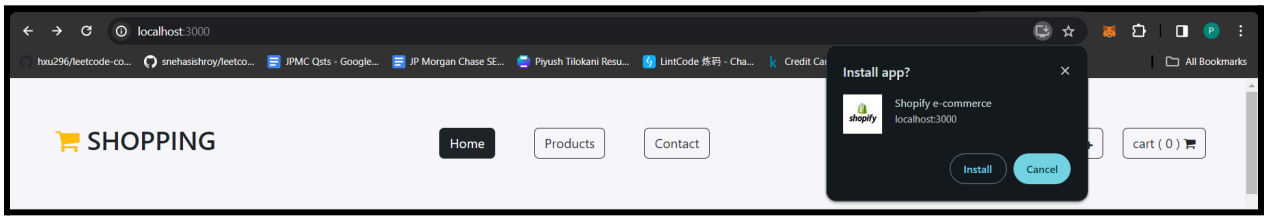
```
  <!-- Meta Tags required for
       Progressive Web App -->
  <meta name=
  "apple-mobile-web-app-status-bar"
        content="#aa7700">
  <meta name="theme-color"
        content="black">

  <!-- Manifest File link -->
  <link rel="manifest"
        href="manifest.json">

  <title>React App</title>
</head>

<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
</body>

</html>
```

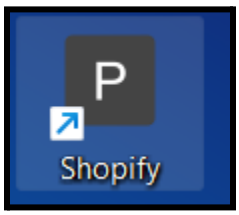
manifest.json

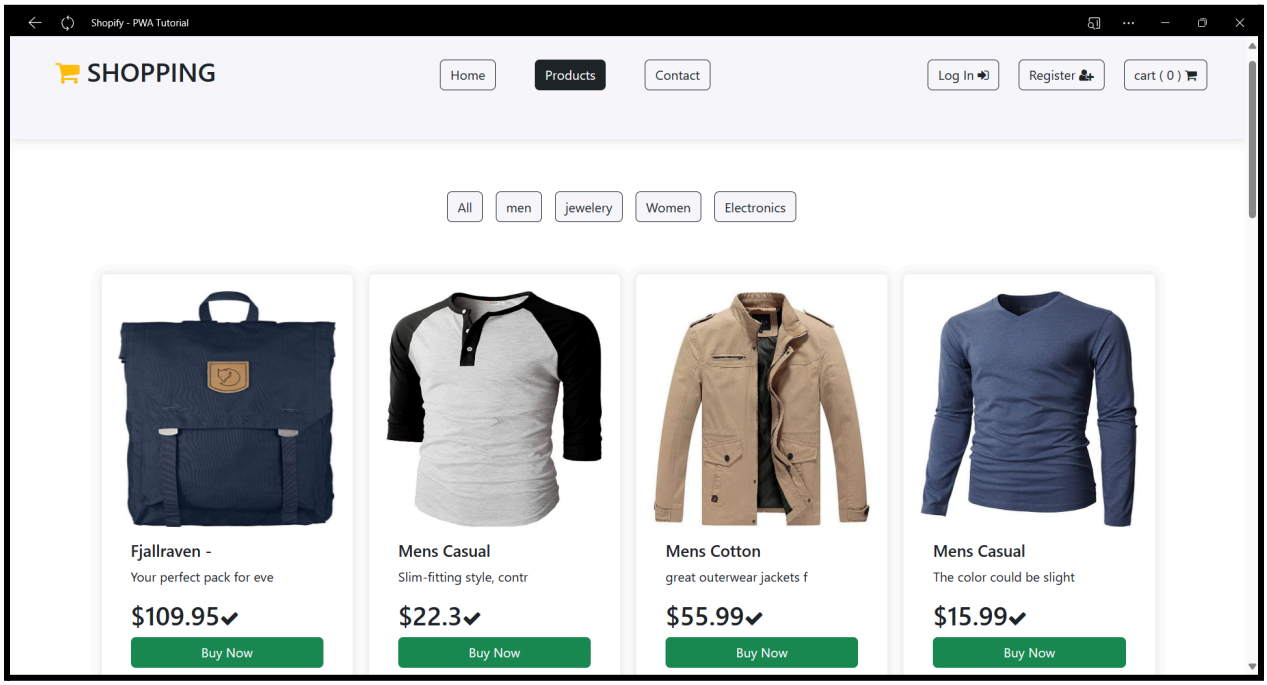
```
{
          "name":"Shopify e-commerce",
          "short_name":"Shopify e-commerce",
          "start_url":"index.html",
          "display":"standalone",
          "background_color":"#5900b3",
          "theme_color":"black",
          "scope": ".",
          "description":"Shopify e-commerce.",
          "icons":[
          {
          "src":"images/icon-192x192.png",
          "sizes":"192x192",
          "type":"image/png"
          },
          {
          "src":"images/icon-512x512.png",
          "sizes":"512x512",
          "type":"image/png"
          }
]
}
```

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

webpack compiled successfully
```

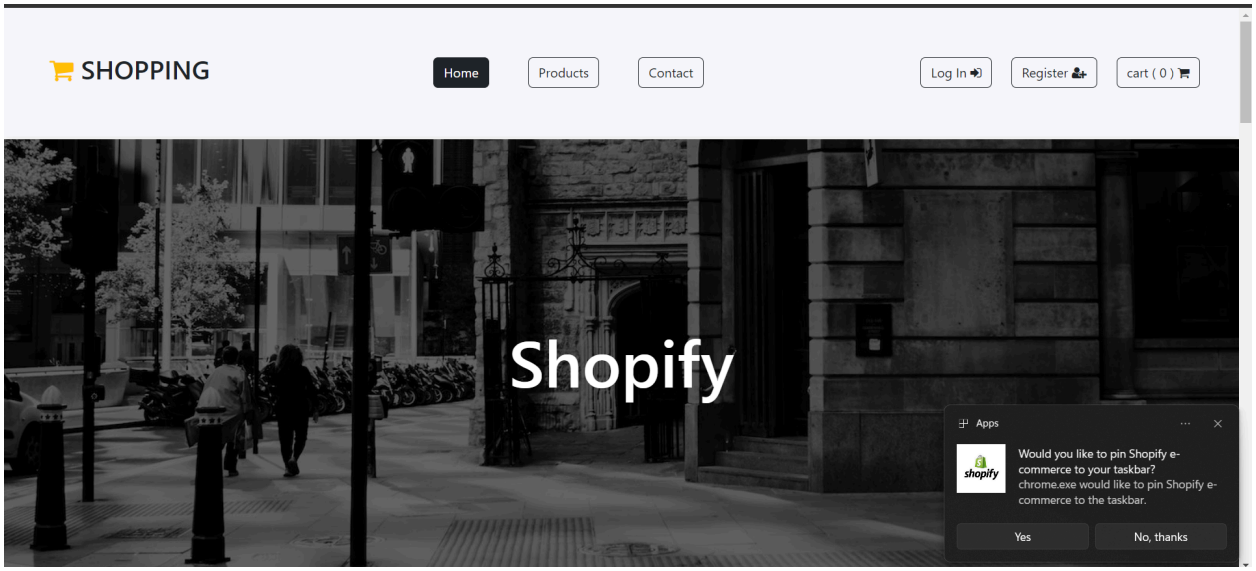








Conclusion: Therefore, I created my first Progressive Web App by adding metadata to the index.html of my e-commerce web app and installed the Application on my local machine.



Project Title: **Roll No.**

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.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment 8

Name:Piyush Tilokani
Div: D15A
Roll no: 63

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

Theory:-

Service Worker

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

Things to note about Service Worker:

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

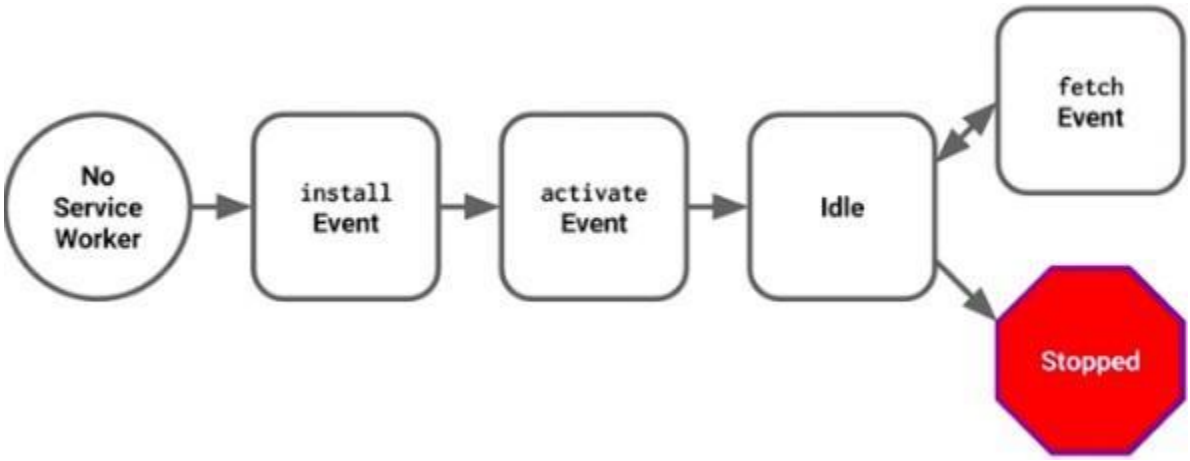
What can we do with Service Workers?

- You can dominate **Network Traffic**
  You can manage all network traffic of the page and do any manipulations. For example, when the page requests a CSS file, you can send plain text as a response or when the page requests an HTML file, you can send a png file as a response. You can also send a true response too.
- You can **Cache**
  You can cache any request/response pair with Service Worker and Cache API and you can access these offline content anytime.
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can’t we do with Service Workers?

- You can’t access the **Window**
  You can’t access the window, therefore, You can’t manipulate DOM elements. But, you can communicate to the window through post Message and manage processes that you want.
- You can’t work it on **80 Port**
  Service Worker just can work on HTTPS protocol. But you can work on localhost during development.

Service Worker Cycle



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

- Registration
- Installation
- Activation

Registration

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

main.js

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

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

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

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

main.js

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

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

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

main.js

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

Installation

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

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

Code:

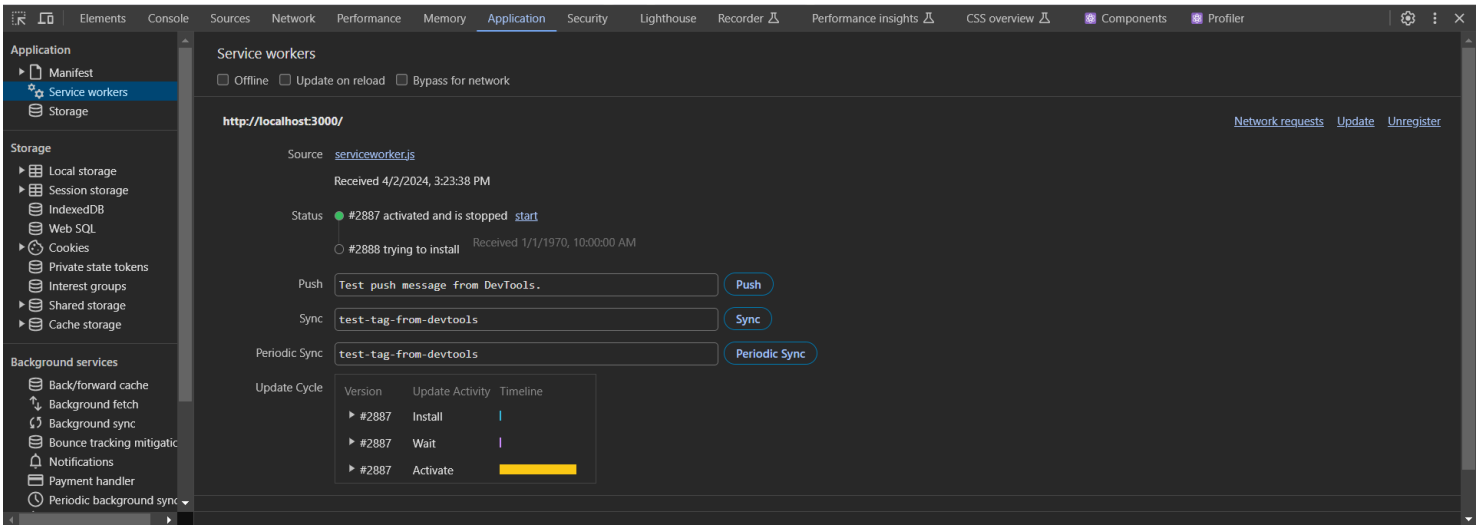
index.html

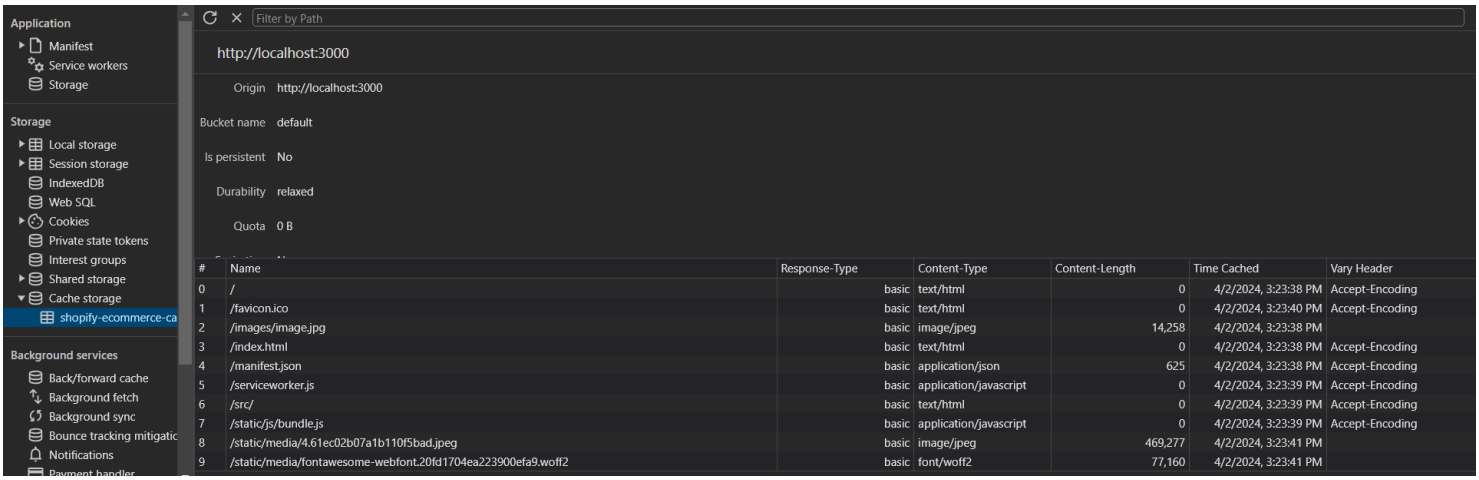
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <script src="serviceworker.js"></script>
    <link rel="manifest" href="manifest.json" />

    <meta charset="utf-8" />
    <meta
      name="viewport"
      content="width=device-width,
               initial-scale=1"
    />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <meta name="theme-color" content="#4285f4" />

    <!-- Title -->
    <title>PWA Tutorial</title>
    <link rel="apple-touch-icon" href="" />
```

```
    <!-- Meta Tags required for
      Progressive Web App -->
    <meta name="apple-mobile-web-app-status-bar" content="#aa7700" />
    <meta name="theme-color" content="black" />

    <!-- Manifest File link -->
    <link rel="manifest" href="manifest.json" />

    <title>React App</title>
  </head>

  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <script>
      window.addEventListener("load", () => {
        registerSW();
      });
      // Register the Service Worker
      async function registerSW() {
        if ("serviceWorker" in navigator) {
          try {
            const registration = await navigator.serviceWorker.register(
              "serviceworker.js"
            );
            console.log("Registered Service Worker:", registration);
          } catch (error) {
            console.log("SW Registration Failed:", error);
          }
        }
      }
    </script>
  </body>
</html>

serviceworker.js
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);
})
);
});
```

Output:





Conclusion: In this experiment, we have registered a service worker, and completed the activation process for a new service worker for the E-commerce PWA.

Project Title: **Roll No.**

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.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment 9

Name:Piyush Tilokani
Div: D15A
Roll no: 63

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

Theory:-

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

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

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

information messages to the page.

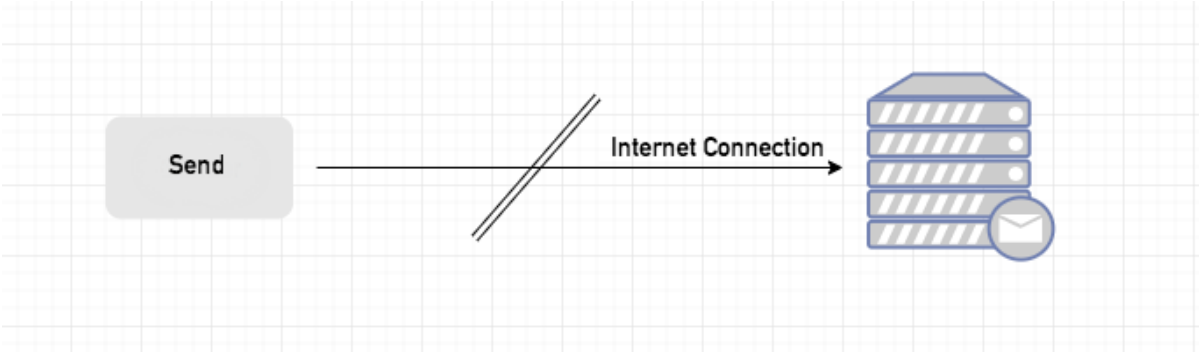
```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);

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

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

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

Sync Event

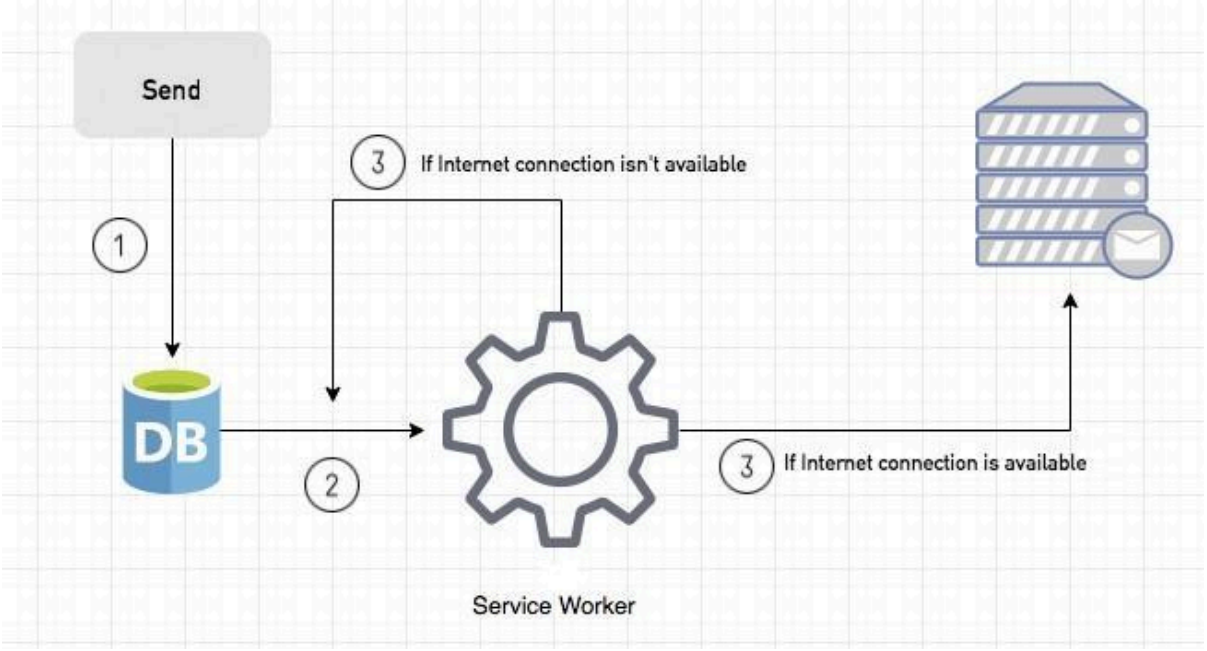
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are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

Here is a job for the Background Sync.

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



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



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

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

Event Listener for Background Sync Registration

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

Event Listener for sw.js

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

Push Event

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

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

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

Code:

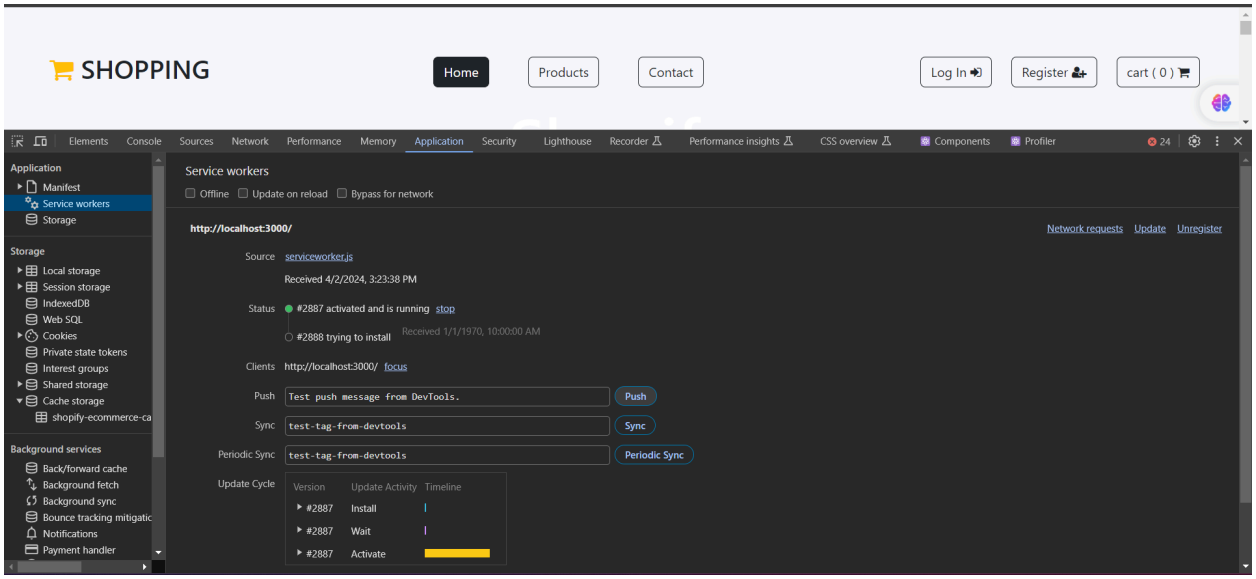
index.html

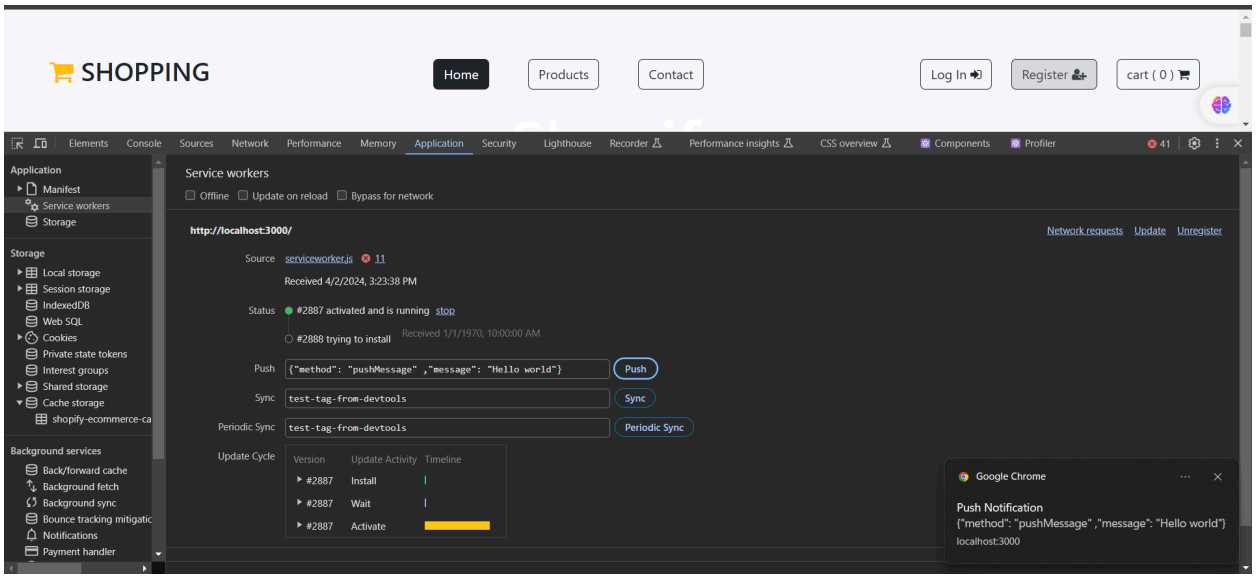
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <script src="serviceworker.js"></script>
    <link rel="manifest" href="manifest.json" />

    <meta charset="utf-8" />
    <meta
      name="viewport"
      content="width=device-width,
```

```
            initial-scale=1"
    />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <meta name="theme-color" content="#4285f4" />

    <!-- Title -->
    <title>PWA Tutorial</title>
    <link rel="apple-touch-icon" href="" />
    <!-- Meta Tags required for
      Progressive Web App -->
    <meta name="apple-mobile-web-app-status-bar" content="#aa7700" />
    <meta name="theme-color" content="black" />

    <!-- Manifest File link -->
    <link rel="manifest" href="manifest.json" />

    <title>React App</title>
  </head>

  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <script>
      window.addEventListener("load", () => {
        registerSW();
      });
      // Register the Service Worker
      async function registerSW() {
        if ("serviceWorker" in navigator) {
          try {
            const registration = await navigator.serviceWorker.register(
              "serviceworker.js"
            );
            console.log("Registered Service Worker:", registration);
          } catch (error) {
            console.log("SW Registration Failed:", error);
          }
        }
      }
    </script>
  </body>
</html>

serviceworker.js
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);
})
);
});
```

Output:





Conclusion: In this experiment, we have registered a service worker, and completed the activation process for a new service worker for the E-commerce PWA.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment 10

Name: Piyush TIlokani
Div: D15A
Roll no: 63

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

Deployed link: https://piyush-tilokani.github.io/PWA-Ecommerce/

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

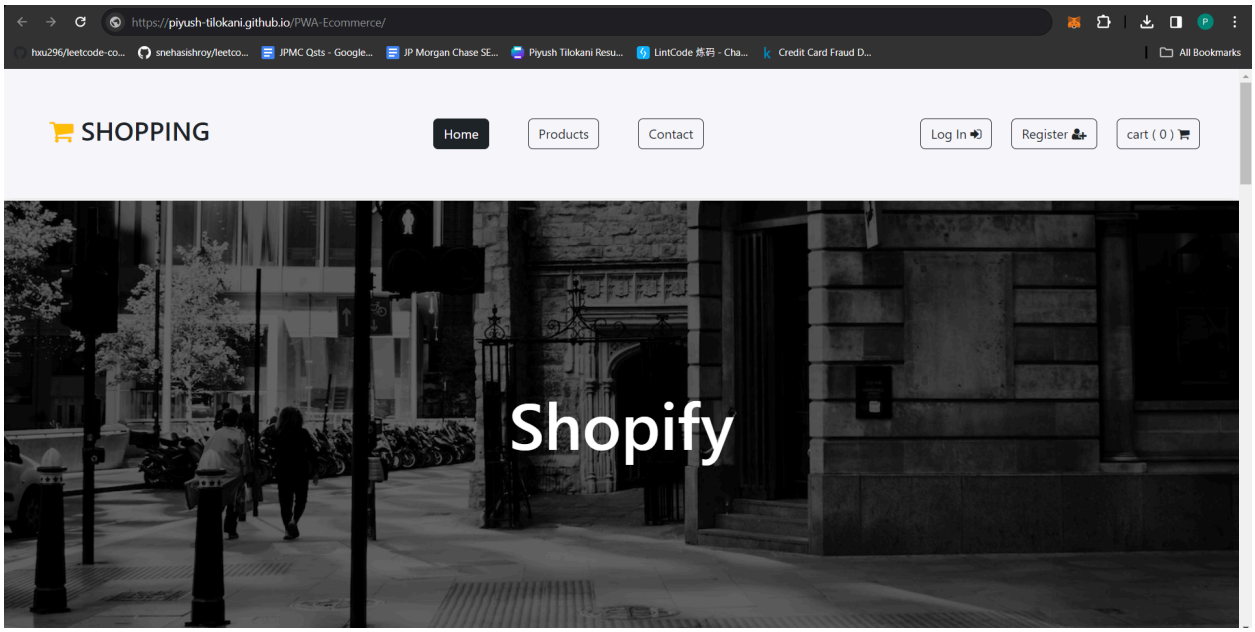
Instacart, 9GAG, and Twitch are some of the popular companies that use Firebase
Firebase has a broader approval, being mentioned in 1215 company stacks & 4651 developers stacks

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.



Conclusion: Thus, we have deployed our app on github pages

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

Experiment 11

Name:Piyush Tilokani
Div: D15A
Roll no: 63

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

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

Google Lighthouse :

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

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

Key Features and Audit Metrics

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

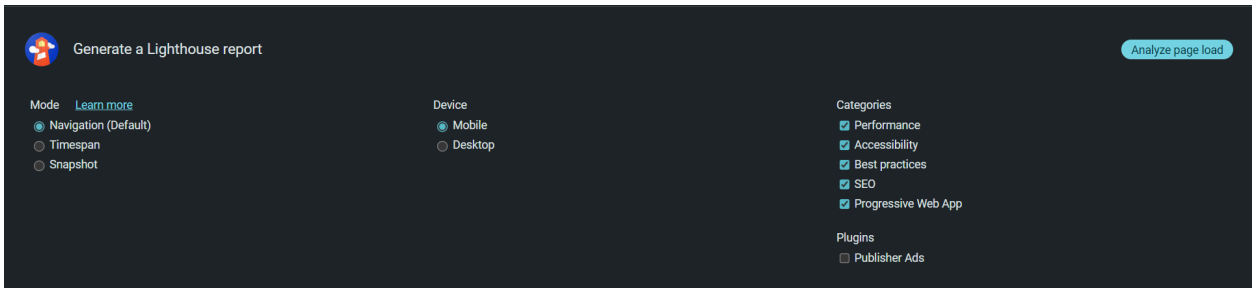
1. **Performance:** This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score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

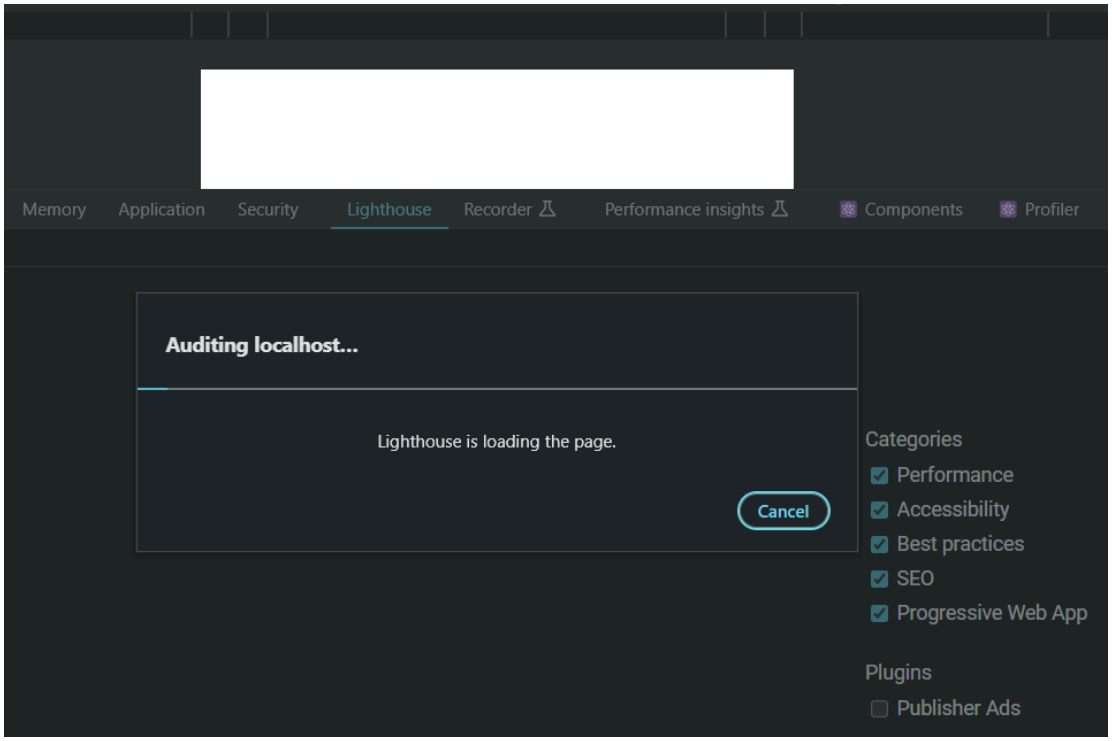
website is, across a plethora of accessibility features that can be implemented in your page (such as the ‘aria-’ attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the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

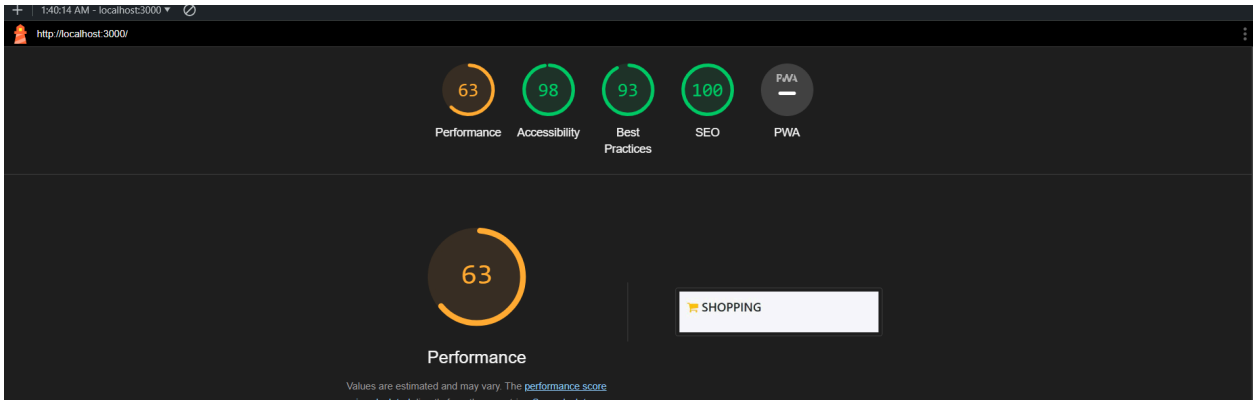
4. **Best Practices:** As any developer would know, there are a number of practices that have been deemed ‘best’ based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS

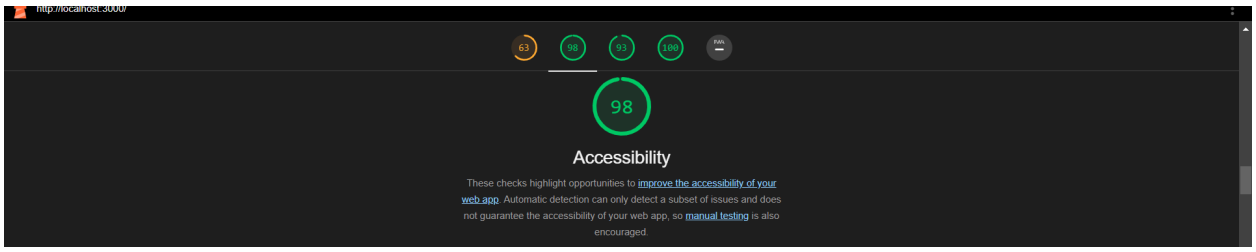
Avoiding the use of deprecated code elements like tags, directives, libraries, etc. Password input with paste-into disabled

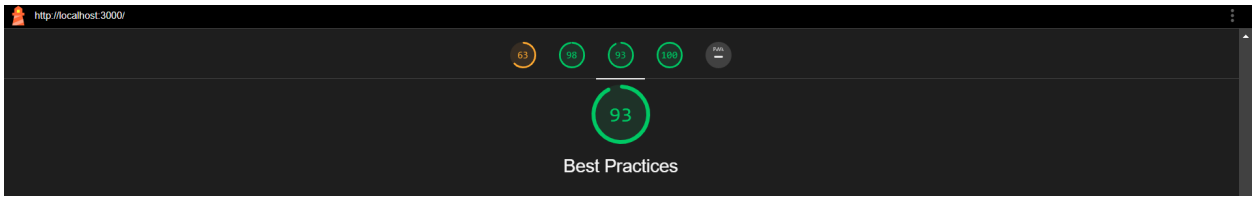
Geo-Location and cookie usage alerts on load, etc.

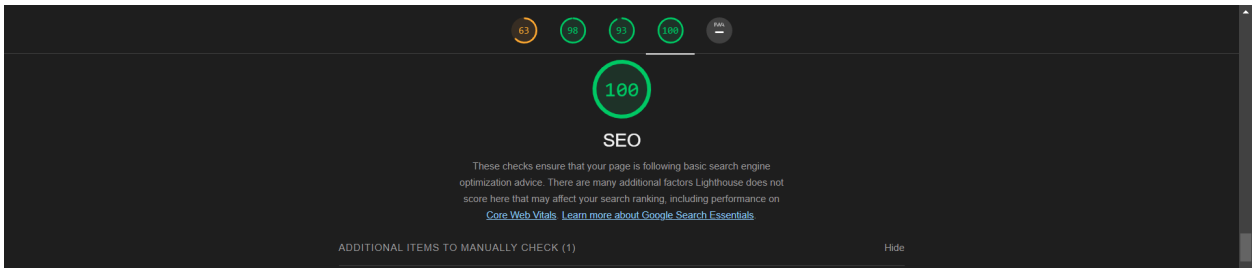


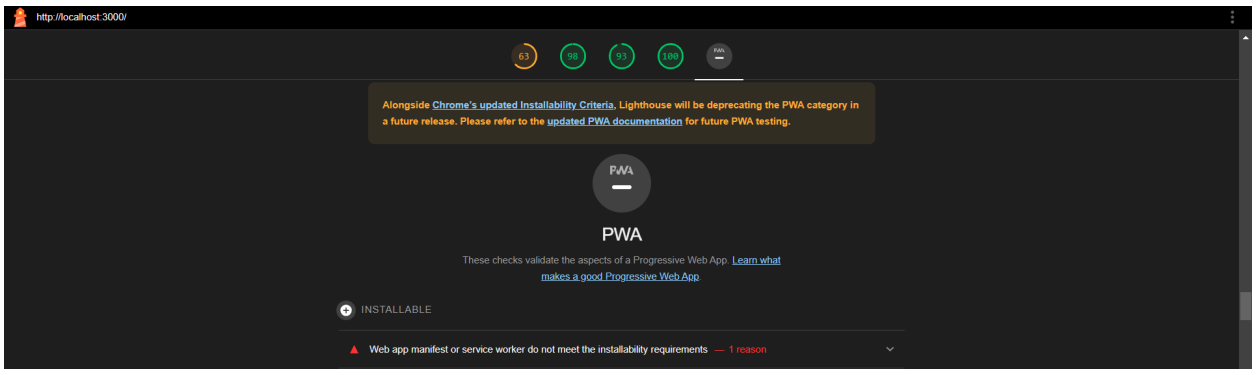








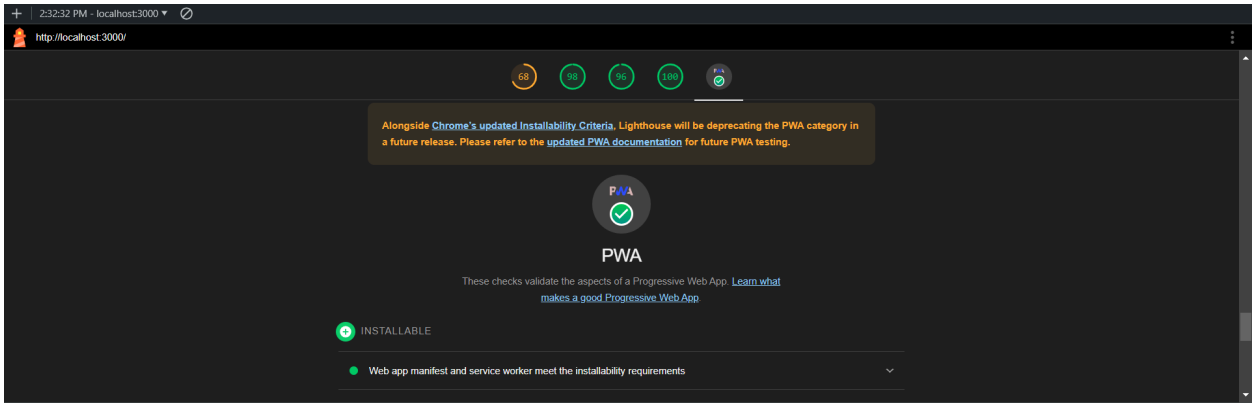




Changes made to manifest.json


```
{
        "name":"Shopify e-commerce",
        "short_name":"Shopify e-commerce",
        "start_url":"index.html",
        "display":"standalone",
        "background_color":"#5900b3",
```

```
        "theme_color":"black",
        "scope": "./",
        "description":"Shopify e-commerce.",
        "icons":[
        {
        "src":"images/image.jpg",
        "sizes":"192x192",
        "type":"image/png",
        "purpose": "maskable"
        },
        {
        "src":"images/image.jpg",
        "sizes":"512x512",
        "type":"image/png"
    ,"purposes": "any maskable"
        },
    {
        "src": "images/image.jpg",
        "type": "image/png",
        "sizes":"144x144",
        "purpose": "any maskable"

    }
]
}
```



Conclusion: Therefore, created my first Progressive Web App by adding metadata to the index.html of my e-commerce web app and installed the Application on my local machine.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Name - Piyush Tilokani
Div - D15A
Roll no. - 63

SDS | Page No. | Date

MAD PWA Lab

Assignment - 1 Flutter

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

Ans Flutter is google's software development kit for building iOS and Android apps. It allows you to create cross-platform apps that provide native performance. Apps created with Flutter feature beautiful and intuitive design and are able to run animation smoothly. Flutter also increases mobile development speed, helping lower costs.

Key features and advantages :-

i) Single Codebase : Code written once can be deployed to iOS and android platforms. This not only reduces ~~deployment~~ development time but also ensures consistent behaviour across different devices.

ii) Hot Reload : Flutter's hot reload feature allows developers to instantly see the impact of change made to the code without restarting app. Enabling quicker iterations and debugging.

iii) Rich widget library : Flutter provides comprehensive set of highly customizable widgets for building UIs.

iv) Performance : Flutter compiles to native ARM code, resulting in high performance applications. Use of Dart language and Skia graphics engine contributes smooth animation and fast execution.

v) Widget-based Architecture : Flutter's architecture revolves around widgets making it modular and easy to organize code. Developers can compose complex UIs by combining smaller, reusable widgets.

vi) Open Source : Flutter encourages collaboration and contribution from the community. This leads to continuous improvement, regular updates and wealth of third-party packages and plugins.

Traditional approach:

i) Developers had to maintain different codebase for iOS and android apps using languages like Java, Swift, Objective-C etc.
ii) Change in code used to require rebuilding and restarting the entire application, leading to slower development cycles.
iii) Achieving a consistent UI across platforms might require using different UI component which was a problem.
iv) Implementing custom animations may be challenging and achieving a highly expressive UI can be time consuming.

Such traditional approach have been tackled and these are some factors that make Flutter framework different from tradition approaches. It's key features and advantages is the reason why it has gained popularity in the developer community.

Q2 Widget tree and Composition- Describe the concept of widget tree in Flutter. Explain how widget composition is used to build complex UI. Provide examples of commonly used widgets & their roles in creating widget tree.

Ans Widget Composition is a powerful concept in Flutter that involves combining multiple smaller widgets to create more complex and sophisticated UI. Instead of creating large and monolithic UI components, developers can break the UI into smaller, reusable widgets and compose them hierarchically to form a comprehensive UI.
Some of its benefits are:-

i) Reusability of code
ii) Modularity
iii) Readability and Maintainability
iv) Flexibility
v) Scalability
vi) Customization

SDS | Page No.
Date / /

- Widget tree is a fundamental concept that represents the hierarchy of UI elements in application. Everything in flutter is a widget. From simple elements to more complex structures like entire screens or even entire application itself.
- Widget tree is organised in a hierarchical structure, with each widget encapsulating a part of the UI. Structure begins with root widget typically representing entire application. This root widget can have child widget and each widget can further have its own children forming a tree-like structure.
- Key points about widget tree : i) Hierarchy ii) Composition iii) Reusability iv) Immutability v) Hot Reload

There are 2 types of widget tree : i) Stateless ii) Stateful

Example of commonly used widget :

i) Material App - Represents root widget of the application.

```
MaterialApp(home: MyHomePage());
```

ii) Scaffold - Provides a basic structure for the visual elements of an app such as app bar, and bottom navigation.

```
Scaffold(
    AppBar: AppBar(
        title: Text('MyApp'),
    ),
    body : MyBody(),
)
```

iii) Column / Row : Arranges child widgets vertically or horizontally.

```
Column(
    children: [
        Text: ('First Child'),
        Text: ('Second Child'),
    ], )
)
```

iv) Container : Basic box model that can contain other widgets and define prop like padding, margin, etc.

Q3 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.

Ans State management is a critical aspect of flutter development, playing a pivotal role in ensuring a responsive and efficient user interface. Effective state management strategies is key to building robust and maintainable applications. Importance of state management in flutter :-

i) Reactivity and UI updates: Fulutter follows reactive programming model and state management is essential for reflecting changes in application state.

ii) Performance optimization: Efficient state management minimizes unnecessary widget rebuilds, improving performance by avoiding redundant UI updates.

iii) Code organization and maintainability: Well managed state promotes clean code architecture, making it easiery to understand, maintain and scale app.

iv) Scalability: As application grows in complexity, proper state management becomes crucial for maintaining a manageable and orgnised database.

v) Testing: Proper state management facilitates unit testing and makes it easier to write test cases for different application states. This ensures reliability and stability of the application under various scenarios.

Comparison of state management approaches :-

i) ~~Set~~ setState(): The simplest and built in way to manage local state with widget

- Use case : a) Suitable for small to medium-sized applications
  b) Ideal for managing simple UI specific states
- Pros : a) Straightforward and easy to use
  b) No external dependencies
- Cons : a) Limited to the scope of a single widget
  b) May lead to widget rebuilds higher in widget tree.

ii) Provider: A popular state management solution that provides global and easily accessible state.

- Use Case: Good for managing global state, dependency injection and sharing data between widgets.
- Pros: a) Simple and easy to implement
  b) Offers good performance and efficiency
  c) Well suited for dependency injection.
- Cons: a) Can be considered overkill for small application.
  b) Limited support for complex asynchronous operations.

iii) Riverpod: An extension of provider that ~~includes~~ introduces improvement in terms of readability, scalability and testability.

- Use case: Offers improved syntax and advanced features compared to provides.
- Pros: a) Provides improved readability and testability
  b) More flexible in terms of state management
- Cons: a) Slightly steeper learning curve compared to provider.

Scenarios:

i) setState(): Ex: Basic calculator app where state changes are isolated to single screen.

ii) Provider: Ex: A weather app with multiple screens where the chosen location and temperature unit need to be shared across different widgets.

iii) Riverpod: Ex: An e-commerce app with complex user applications, shopping cart management and real time updates where riverpod's advanced features can be 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 service commonly used in flutter development and provide a brief overview of how data synchronization is achieved.

SDS | Page No. | Date

Ans Integrating firebase with flutter applications.

i) Create firebase project.
ii) Register your app by clicking on Add app and choose the appropriate platfor
iii) Add firebase SDK to flutter project in pubspec.yaml
dependencies:
firebase_core: ^1.10.6
firebase_auth: ^4.10.6
iv) Initialize firebase in your app Firebase.initializeApp() in main() function
v) Use firebase services in your App.

Benefits of using firebase as a backend solution :-

i) Real-time database
ii) Authentication
iii) Cloud firestore
iv) Cloud storage
v) Easy integration
vi) Scalability
vii) Analytics and Crash reporting
viii) Firebase hosting

Data synchronization in flutter: Firebase cloud firestore employs a real-time synchronization model to ensure that data changes are instantly reflected across all connected devices. The process is as follows:

i) Collections & Documents: Data in firestore is organized into collections which can contain multiple documents. Each document is a set of key-value pairs.
ii) Listeners & Observables: Flutter developers can set up listeners to observe changes in particular collection or document. These listeners are notified whenever data is modifi
iii) Real time updates: When data changes in firestore database, the changes are instantly pushed to all the connected clients.
iv) Offline support: Firestore provides offline support, allowing users to interact with the app even when offline. Data modifications made while offline are stored locally and synced with the server once the device is online.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Name - Piyush Tilokani
Div - D15A
Roll no - 63

SDS | Page No. | Date / /

MAD PWA Lab

Assignment - 2

1) Define Progressive Web App (PWA) and explain its significance in modern web development. Discuss the key characteristics that differentiates PWAs from traditional mobile apps.

Ans A Progressive Web App is a type of web application that uses modern web technologies to provide users with native app-like experience directly through their web browser. PWAs are designed to be reliable, fast, engaging, offering features such as offline access, push notifications, and the ability to install the app on the user's device.

The significance of PWAs in modern web development lies in their ability to bridge the gap between web and native mobile applications. Here are some key characteristics that differentiate PWAs from traditional mobile apps :-

i) Cross-platform compatibility : Built using HTML, CSS and JavaScript, PWAs are compatible with various platforms like desktop, smartphone, tablet.

ii) Responsiveness : PWAs are designed to adapt to different screen sizes.

iii) Offline functionality : PWAs can cache resources and content, enabling users to access the app even when they are offline.

iv) Installability : PWAs can be installed on a user's device directly from the browser, without the need of any app store.

v) Push notifications : PWAs can send push notifications to users, increasing engagement and enabling real-time communication, even when the app is not open.

SDS Page No.
Date / /

2) Define responsiveness and web design and explain its importance in the context of PWAs. Compare and contrast responsive, fluid and adaptive web design approaches.

Ans Responsive web design is an approach to web development aimed at creating websites that provide an optimal viewing and interaction experience across a wide range of devices and screen sizes. The key principle of responsive design is use of flexible layouts, fluids, grids and media queries.
In context of PWAs, responsive design is crucial for ensuring interoperability ~~pla~~. Since PWAs are accessed through web brows they need to be responsive, consistent and user-friendly regardless of devi

1) Responsive Web design:
- Uses flexible layouts and grids to adapt content on different screen sizes
- Achieves responsiveness through CSS media queries base on height, width.
- Optimizes content layout and functionality for each screen size.

2) Fluid Web design:
- Similar to responsive design, it uses fluid designs and flexible layouts.
- Focuses more on fluidity, with elements scaling smoothly to fill available spa
- Does not rely on fixed breakpoints, instead, allows elements to resize to continuously fill the available space.

3) Adaptive web design:
- Involves creating multiple layouts/designs tailored to specific screens
- Uses server-side techniques to detect user's device and deliver appropriate la
- Provides a more targeted and optimized experience for each device category.

SDS | Page No.
Date / /

Q3 Describe the lifecycle of Service Workers, including registration, installation and activation phases.

Ans Lifecycle of service workers :-

i) Registration :

- The first step in the lifecycle of Service Worker is registration.
- This occurs in the main script of a web application, typically the main JavaScript file.
- During registration, the Service Worker script is fetched from the server and registered with the browser using 'navigator.serviceWorker.register()' method
- The registration process typically occurs in the background and does not block the main thread of the web application.

ii) Installation :

- After successful registration, browser downloads service worker script and begins the installation process.
- During installation, the browser fires 'install' event, allowing the service worker to cache static assets and other resources to run the web application offline.
- The service worker can use the 'event.waitUntil()' method to extend the installation process until necessary resources are cached.

iii) Activation :

- Once the Service Worker is successfully installed, the browser fires 'activate' event.
- During activation, the service worker can clean up any old caches or perform other tasks necessary to ensure it is ready to control client pages.
- The 'activate' event is also an opportunity for the service worker to take control of client pages and intercept network requests.

SDS | Page No. | Date / /

4) Explain the use of Indexed DB in the Service Worker for data storage.

Ans IndexedDB is a low-level API provided by modern web browsers for client-side storage of large amounts of structured data.

Use of IndexedDB :-

i) Persistent storage : IndexedDB provides persistent storage, meaning the data stored in the database persists even after the web page or the service worker is closed or refreshed.

ii) Asynchronous API : IndexedDB operations are asynchronous, which means they don't block the main thread of the web application.

iii) Structured data storage : IndexedDB stores data in a structured manner, similar to a NOSQL database. It allows for the creation of object stores, which are collection of JavaScript objects.

iv) Large data handling : IndexedDB is capable of handling large amounts of data efficiently, making it suitable for applications that require storage of substantial datasets.

v) Transaction-based : IndexedDB operations are performed within transactions, ensuring data integrity and consistency.

In the context of Service Worker, Indexed DB can be used for:-

i) Caching : Storing resources, such as HTML, CSS, JS files, images and other assets for offline access. This allows the Service Worker to serve cached resources when the network is unavailable.

ii) Data persistence : Storing application data, such as user preferences, settings or application state, to provide a seamless user experience.

iii) Background Synchronisation : Storing data received from background synchronization tasks, such as periodic updates or push notifications, and processing them asynchronously without interrupting the user's interaction with the web app.