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 **Lisa Sharma** of **D15-A** semester **VI,** has 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 Jewani

Principal

Head of Department

Dr. Mrs. Shalu Chopra

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

Year/Sem/Class : D15A/D15B **A.Y.:** 23-24

Faculty Incharge : Mrs. Kajal Jewani.

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 Airbnb / Purity Plants **Roll No. 53**

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 Analyze PWA Features and deploy it over app hosting solutions	L3, L4

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1	17/1/24	24/1/24	11
2.	To design Flutter UI by including common widgets.	LO2	24/1/24	31/1/24	15
3.	To include icons, images, fonts in Flutter app	LO2	31/1/24	7/2/24	10
4.	To create an interactive Form using form widget	LO2	7/2/24	14/2/24	11
5.	To apply navigation, routing and gestures in Flutter App	LO2	14/2/24	21/2/24	12
6.	To Connect Flutter UI with fireBase database	LO3	21/2/24	6/3/24	11
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4	6/3/24	13/3/24	15
8.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA	LO5	13/3/24	20/3/24	15
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	20/3/24	27/3/24	15
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	27/3/24	27/3/24	15
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	27/3/24	27/3/24	15
12.	Assignment-1	LO1,LO 2,LO3	6/2//24	5/2/24	5
13.	Assignment-2	LO4,LO 5,LO6	20/3/24	21/3/24	4

MAD & PWA Lab
Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	53
Name	Lisa Sharma
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	11

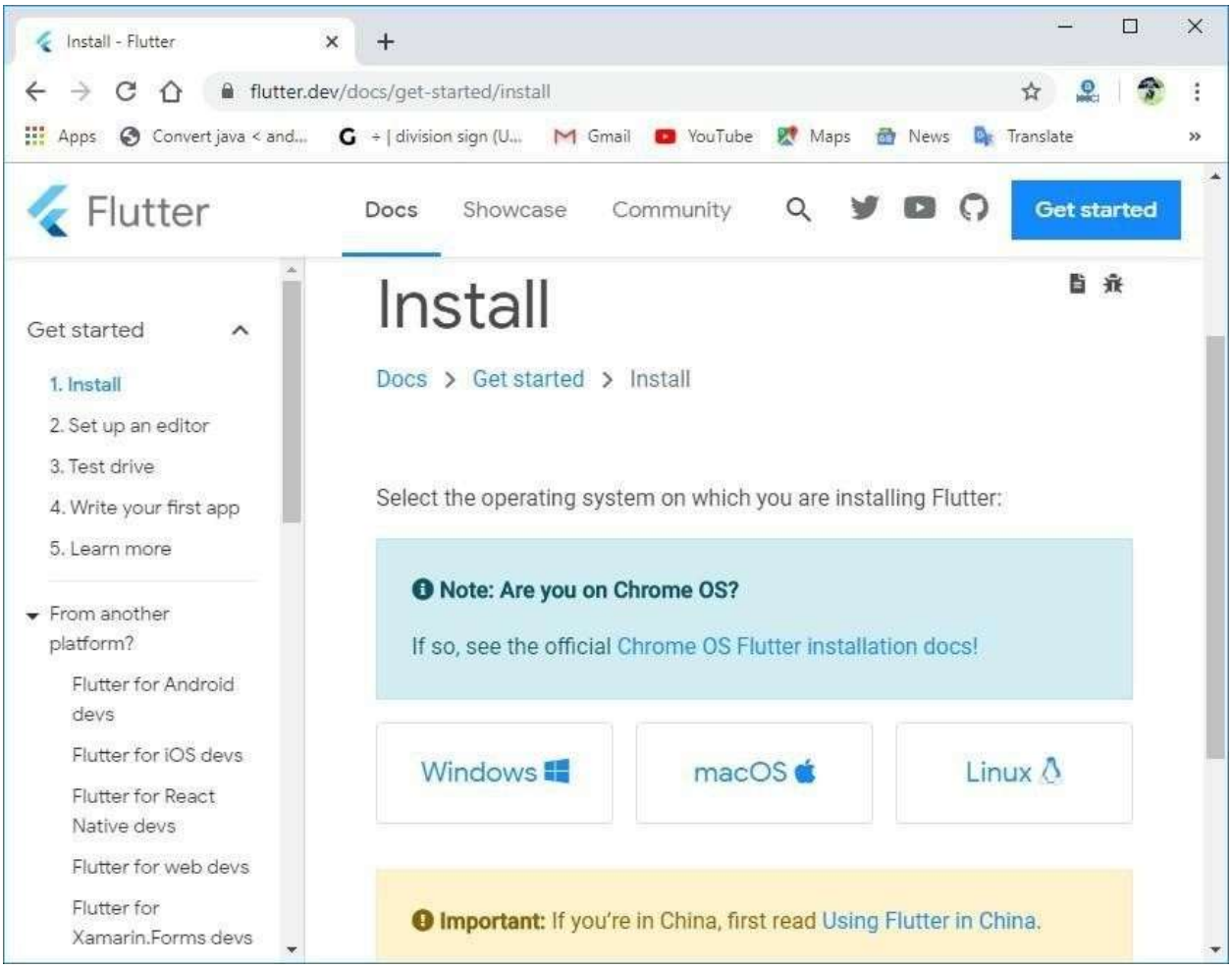
Name: Lisa Sharma
Roll No: 53

Experiment No: 01

Installation and Configuration of Flutter Environment.

Install the Flutter SDK

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

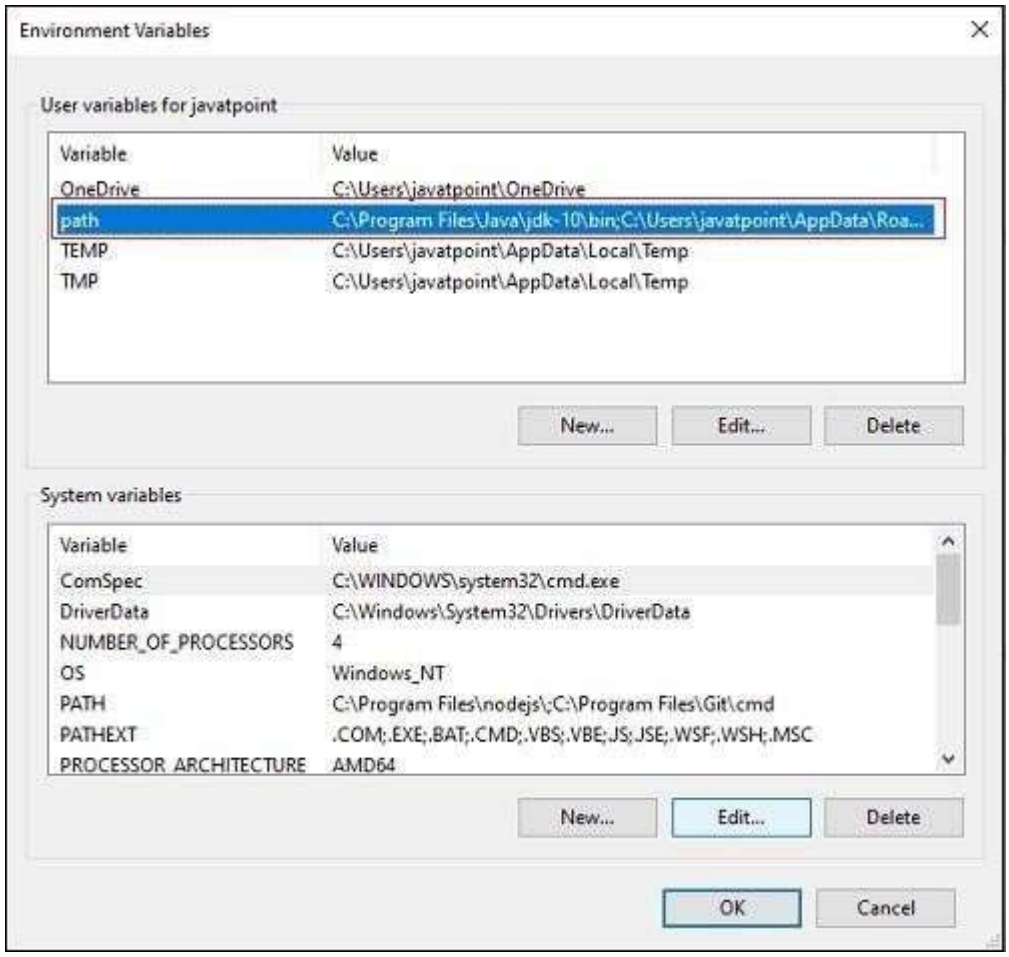


Step 2: Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will find the download link for SDK.

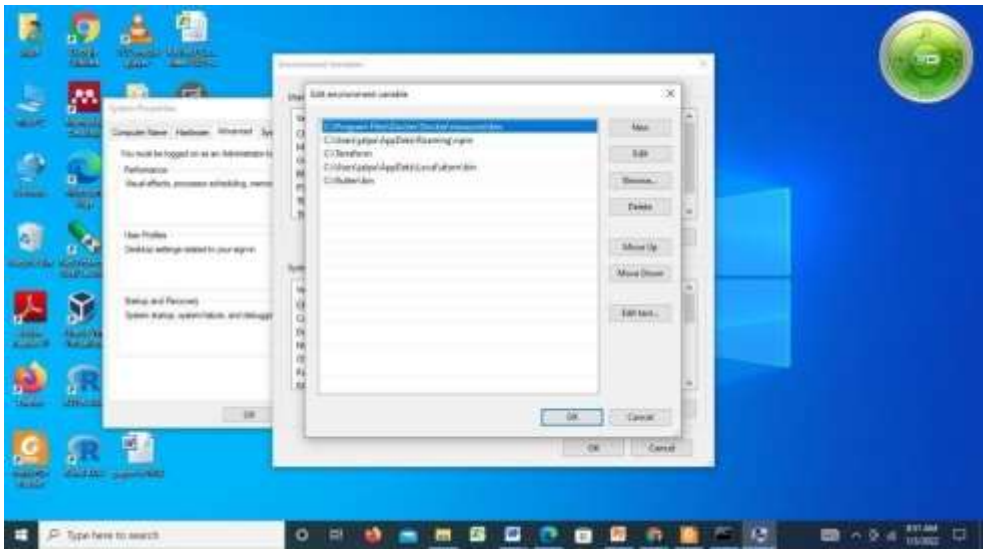
Step 3: When your download is complete, extract the **zip** file and place it in the desired installation folder or location, for example, C: /Flutter.

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

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

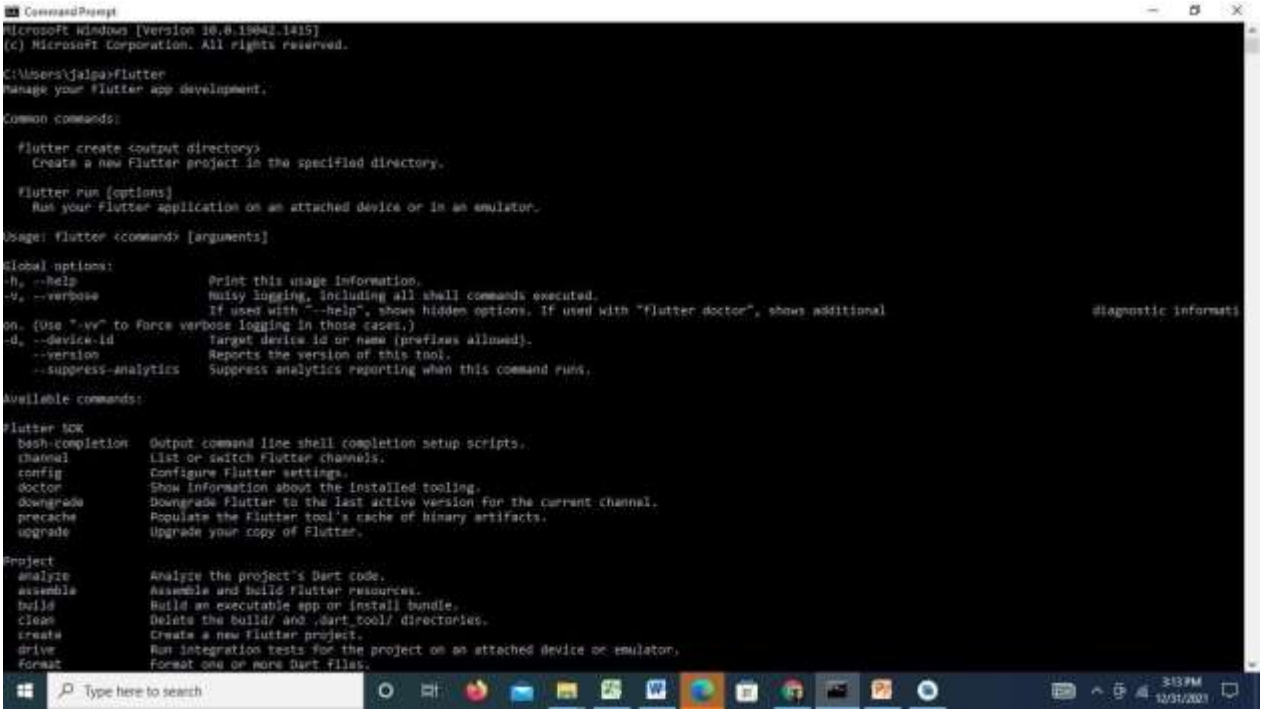


Step 4.2: Now, select path -> click on edit. The following screen appears

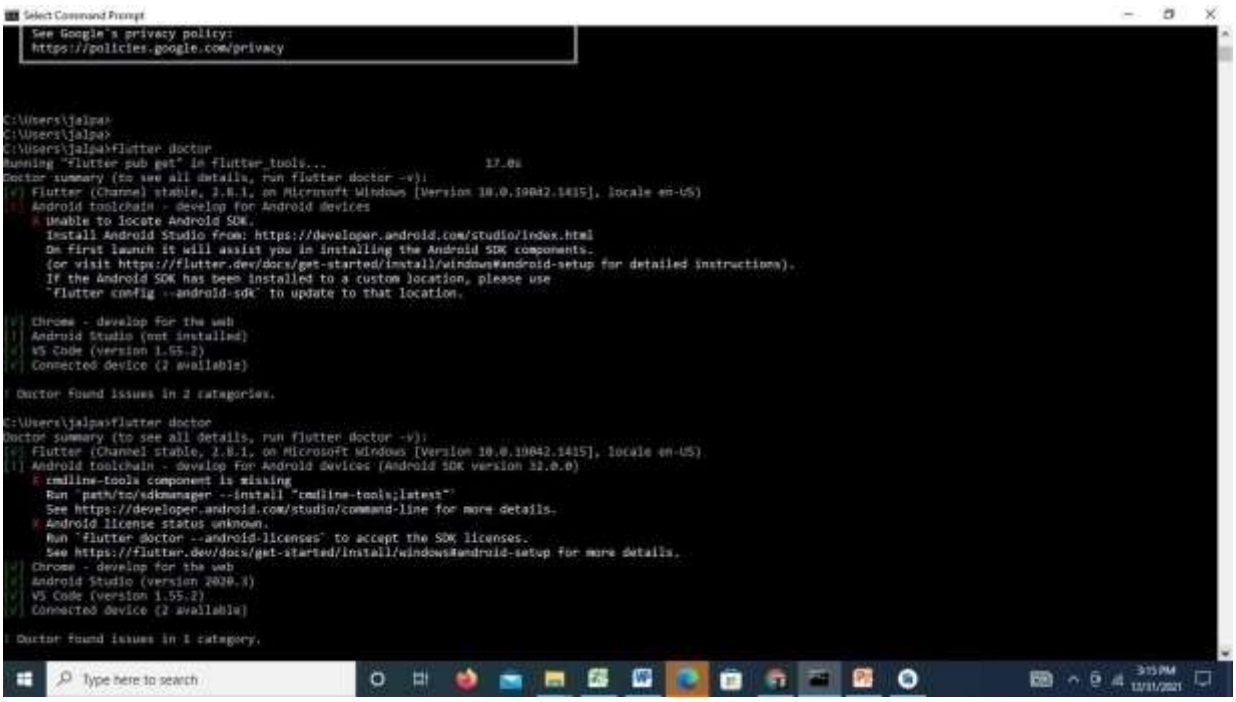


Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value > ok -> ok -> ok.

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



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



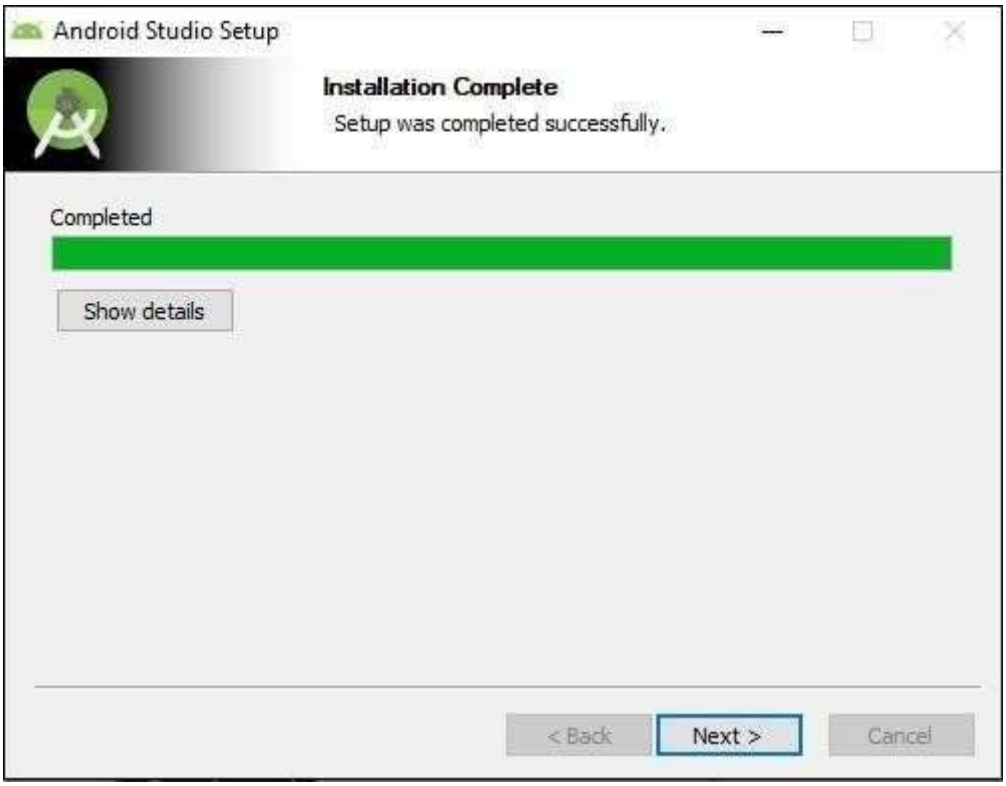
Step 6: When you run the above command, it will analyze the system and show its report, as shownin the below image. Here, you will find the details of all missing tools, which required to run Flutteras well as the development tools that are available but not connected with the device.

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.

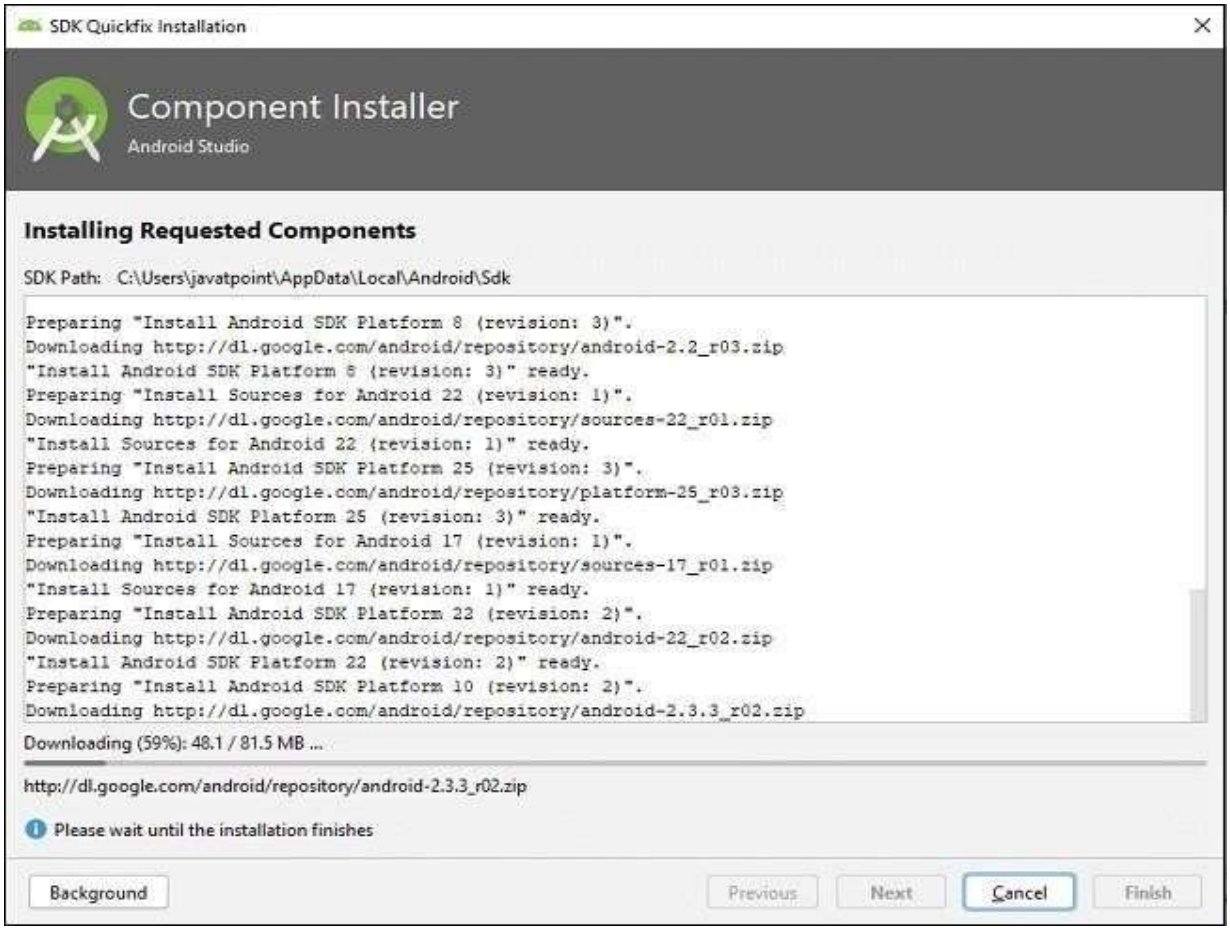
Step 7.1: Download the latest Android Studio executable or zip file from the official site.

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

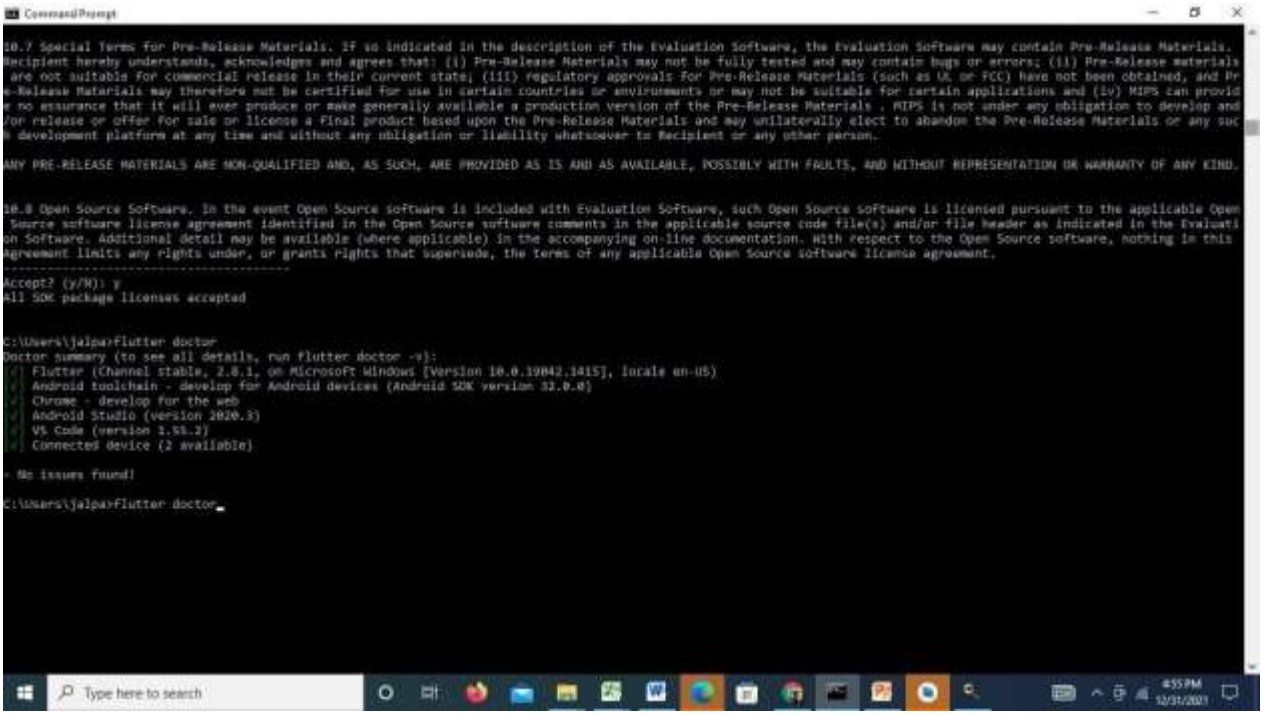
Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



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 theFlutter 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.

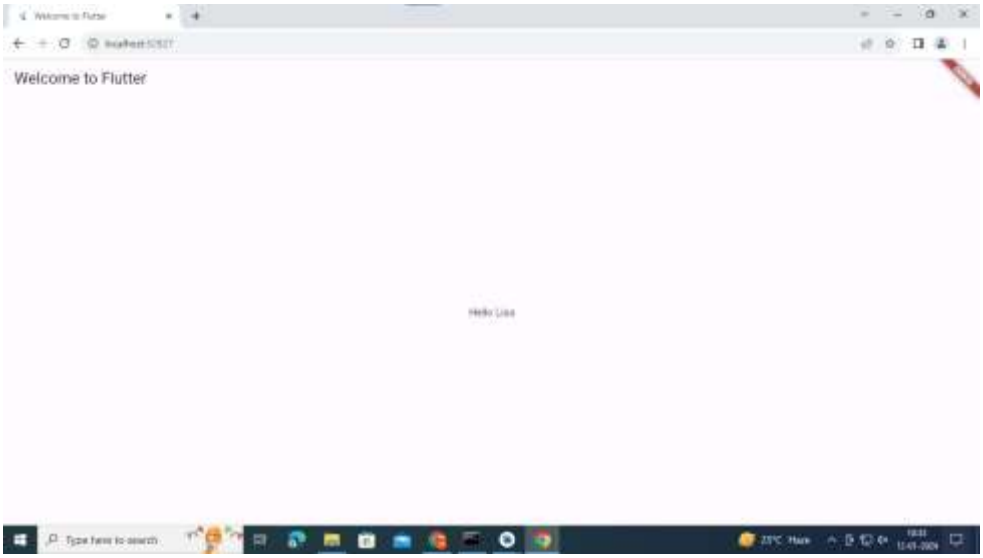
Step 9.2: Now, search the Flutter plugin. If found, select Flutter plugin and click install. When you click on install, it will ask you to install Dart plugin as below screen. Click yes to proceed.

Step 9.3: Restart the Android Studio.

Write Code to print the *hello your_name* in the output.

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

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



MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	53
Name	Lisa Sharma
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Name: Lisa Sharma
Roll No: 53

MAD and PWA Lab

Aim: To design Flutter UI by including common widgets.

Theory:

Flutter widgets are the building blocks of the user interface, representing visual and interactive elements. Widgets can be categorized into two main types: Stateless and Stateful. Stateless widgets are immutable and do not store any internal state, while Stateful widgets can change and maintain their state over time.

Built-in and Custom Widgets:
Flutter comes with a rich set of built-in widgets that cover a wide range of UI elements and interactions. Additionally, developers can create their custom widgets to tailor the user interface to specific requirements. This section explores the anatomy of custom widgets and their integration into the Flutter framework.

Layout and Constraints:
Flutter's layout system is based on the concept of constraints, allowing widgets to adapt to various screen sizes and orientations. Understanding how widgets handle layout constraints is essential for creating responsive and adaptive user interfaces.

Animation with Widgets:
Flutter provides a robust animation framework that seamlessly integrates with widgets, enabling the creation of fluid and engaging user experiences. This section explores how animations can be incorporated into Flutter widgets to enhance the overall user interface.

Testing and Debugging Widgets:
As with any software development, testing and debugging are integral parts of the process. This section discusses strategies for testing and debugging Flutter widgets, ensuring the reliability and quality of the UI components
Container:
The Container widget is a basic building block that can contain other widgets and is often used to define the dimensions, padding, margin, and decoration of a UI element.

Text:
The Text widget is used to display a paragraph or a line of text. It supports various styling options such as font size, color, and alignment.

ListView:
The ListView widget is used to create a scrollable list of widgets. It can display a large number of children efficiently.

Row and Column:
Row and Column widgets are used to arrange children in a horizontal or vertical line, respectively.

Stack:
The Stack widget allows you to overlay multiple widgets on top of each other. It's often used for complex layouts.

AppBar:
The AppBar widget represents the top app bar that usually contains the app's title, icons, and actions.

Code:

```
import 'package:flutter/material.dart';
import '../shared/theme/colors.dart';
import '../shared/widgets/app_nav_bar.dart';

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

  @override
  Widget build(BuildContext context) {
    final textTheme = Theme.of(context).textTheme;

    return Scaffold(
      bottomNavigationBar: const AppNavBar(),
      appBar: AppBar(
        toolbarHeight: 128,
        flexibleSpace: Container(
          decoration: BoxDecoration(
            color: appWhite,
            boxShadow: [
              BoxShadow(
                color: appBlack.withOpacity(0.1),
                blurRadius: 1.0,
                spreadRadius: 1.0,
                offset: const Offset(0.0, 1.0),
              )
            ],
          ),
          child: Stack(
            children: [
              const Positioned(
                bottom: 0.0,
                left: 0.0,
                right: 0.0,
                child: PropertyTypeList(),
              ),
              Positioned(
                top: 70.0,
                right: 8.0,
                child: IconButton(
                  onPressed: () {},
                  icon: const Icon(Icons.tune),
                ),
              ),
              Positioned(
                left: 16.0,
```

```
right: 72.0,
top: 64.0,
child: GestureDetector(
  onTap: () {
    context.pushNamed('booking-details');
  },
  child: Hero(
    tag: 'search',
    child: Container(
      padding: const EdgeInsets.symmetric(
        horizontal: 16.0,
        vertical: 8.0,
      ),
      decoration: BoxDecoration(
        color: appWhite,
        border: Border.all(
          color: appGrey,
          width: 0.5,
        ),
        borderRadius: BorderRadius.circular(32.0),
        boxShadow: [
          BoxShadow(
            color: appGrey.withOpacity(0.5),
            blurRadius: 8.0,
            spreadRadius: 8.0,
            offset: const Offset(0.0, 4.0),
          ),
        ],
      ),
      child: Row(
        children: [
          const Icon(Icons.search),
          const SizedBox(width: 8.0),
          Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text(
                'Where to?',
                style: textTheme.bodyMedium!
                    .copyWith(fontWeight: FontWeight.bold),
              ),
              Text(
                'Anywhere • Any week • Add guest',
                style: textTheme.bodyMedium,
              ),
            ],
          ),
        ],
      ),
    ),
  ),
```

```
              ),
            ),
          ],
        ),
      ),
    ),
    body: Padding(
      padding: const EdgeInsets.only(left: 16.0, right: 16.0, top: 16.0),
      child: ListView.builder(
        itemCount: Property.sampleData.length,
        itemBuilder: (context, index) {
          final property = Property.sampleData[index];
          return PropertyCard(property: property);
        },
      ),
    ),
  );
 }
}

class PropertyCard extends StatefulWidget {
 final Property property;

 const PropertyCard({Key? key, required this.property}) : super(key: key);

 @override
 State<PropertyCard> createState() => _PropertyCardState();
}

class _PropertyCardState extends State<PropertyCard> {
 final controller = PageController();
 var currentPage = 0;

 @override
 void dispose() {
  controller.dispose();
  super.dispose();
 }

 @override
 Widget build(BuildContext context) {
  final size = MediaQuery.of(context).size;
  final textTheme = Theme.of(context).textTheme;
  final colorScheme = Theme.of(context).colorScheme;

  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Stack(
        children: [
          Container(
```

```
      clipBehavior: Clip.antiAlias,
      width: size.width,
      height: size.width - 32.0,
      decoration: BoxDecoration(
        borderRadius: BorderRadius.circular(16.0),
      ),
      child: PageView(
        controller: controller,
        onPageChanged: (value) {
          setState(() {
            currentPage = value;
          });
        },
        children: widget.property.photoUrls.map((imageUrl) {
          return Image.network(imageUrl, fit: BoxFit.cover);
        }).toList(),
      ),
    ),
    Positioned(
      bottom: 8.0,
      left: 0.0,
      right: 0.0,
      child: DotsIndicator(
        dotsCount: widget.property.photoUrls.length,
        position: currentPage,
        onTap: (index) {
          controller.animateToPage(
            index,
            duration: const Duration(milliseconds: 300),
            curve: Curves.easeIn,
          );
        },
        decorator: DotsDecorator(
          color: colorScheme.onSecondary,
          activeColor: colorScheme.secondary,
          size: const Size.square(8.0),
          activeSize: const Size(12.0, 8.0),
          activeShape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(12.0),
          ),
        ),
      ),
    ),
  ],
),
Padding(
  padding: const EdgeInsets.all(16.0),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
```

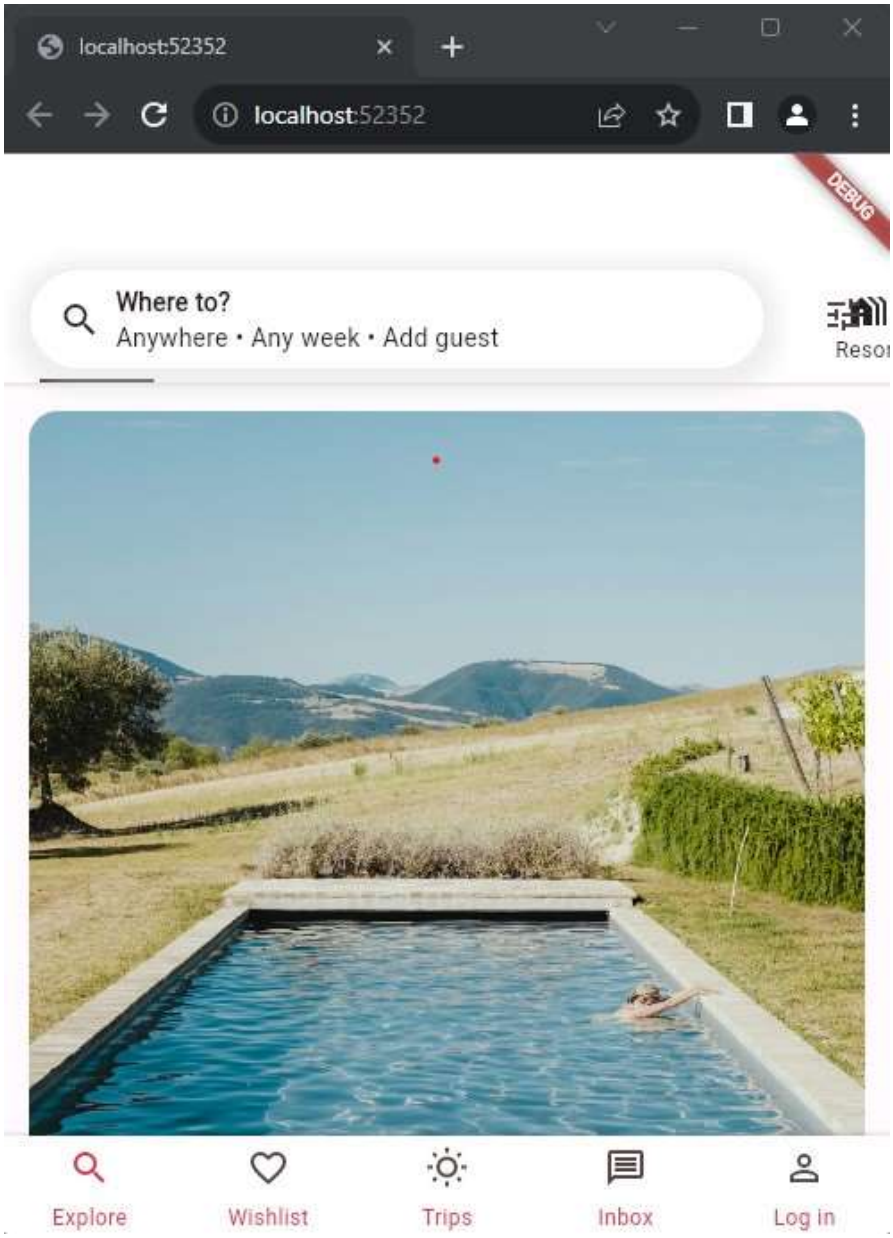
```
            '${widget.property.country}, ${widget.property.city}',
            style: textTheme.bodyLarge!.copyWith(
              fontWeight: FontWeight.bold,
            ),
          ),
          const SizedBox(height: 8.0),
          Text(
            widget.property.description,
          ),
          const SizedBox(height: 8.0),
          Text(
            widget.property.amenities.join(', '),
          ),
        ],
      ),
    ),
  ],
 );
 }
}
```

App Nav Bar

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

import '../theme/colors.dart';

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

 @override
 Widget build(BuildContext context) {
  return Container(
   decoration: BoxDecoration(
    boxShadow: [
     BoxShadow(
      color: appBlack.withOpacity(0.1),
      blurRadius: 1.0,
      spreadRadius: 1.0,
      offset: const Offset(0.0, -1.0),
     )
    ],
   ),
   child: NavigationBarTheme(
    data: NavigationBarThemeData(
     labelTextStyle: MaterialStatePropertyAll(
      Theme.of(context).textTheme.bodySmall!.copyWith(color: appRed),
     ),
    ),
    child: NavigationBar(
     backgroundColor: Colors.white,
     surfaceTintColor: Colors.white,
```

```
      labelBehavior: NavigationDestinationLabelBehavior.alwaysShow,
      onDestinationSelected: (int index) {},
      indicatorColor: Colors.transparent,
      selectedIndex: 0,
      height: 56.0,
      destinations: const [
        NavigationDestination(
          icon: Icon(Icons.search_outlined),
          label: 'Explore',
          selectedIcon: Icon(
            Icons.search,
            color: appRed,
          ),
        ),
        NavigationDestination(
          icon: Icon(Icons.favorite_border_outlined),
          label: 'Wishlist',
          selectedIcon: Icon(
            Icons.favorite,
            color: appRed,
          ),
        ),
        NavigationDestination(
          icon: Icon(Icons.wb_sunny_outlined),
          label: 'Trips',
          selectedIcon: Icon(
            Icons.wb_sunny,
            color: appRed,
          ),
        ),
        NavigationDestination(
          icon: Icon(Icons.message_outlined),
          label: 'Inbox',
          selectedIcon: Icon(
            Icons.message,
            color: appRed,
          ),
        ),
        NavigationDestination(
          icon: Icon(Icons.person_outline),
          label: 'Log in',
          selectedIcon: Icon(
            Icons.person,
            color: appRed,
          ),
        ),
      ],
    ),
  ),
 );
}
```

Output:-



Conclusion:
Flutter's widget architecture offers great flexibility for building complex UIs. Understanding key widgets and concepts is essential for effective Flutter Development

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	53
Name	Lisa Sharma
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	10

Name: Lisa Sharma
Roll No: 53

MAD and PWA Lab
EXPERIMENT - 3

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

Theory:

Fonts:

In Flutter, the TextStyle class is used to define the styling for text within the Text widget or other widgets that involve displaying text. Here's an overview of how you can use the TextStyle class to set various font-related properties

fontSize:
You can set the size of the font using the fontSize property.

fontWeight:
The fontWeight property allows you to set the thickness of the characters in the text.

fontStyle:
The fontStyle property lets you specify whether the text should be in normal, italic, or oblique style.

fontFamily:
You can specify the font family using the fontFamily property. This refers to the specific font you want to use, and it should be available in your project.

decoration:
The decoration property allows you to add decorations to the text, such as underline or overline

Text

A Text widget holds some text to display on the screen. We can align the text widget by using textAlign property, and style property allow the customization of Text that includes font, font weight, font style, letter spacing, color, and many more.

1. Button

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

2. Image

This widget holds the image which can fetch it from multiple sources like from the asset folder or directly from the URL. It provides many constructors for loading image, which are given below:

o Image: It is a generic image loader, which is used by ImageProvider.

o asset: It load image from your project asset folder.

o file: It loads images from the system folder.

o memory: It load image from memory.

o network: It loads images from the network.

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

```
assets:
- assets/images
```

Code:

Booking_details_screen

```
import 'dart:ui';

import 'package:flutter/material.dart';
import 'package:go_router/go_router.dart';
import 'package:models/models.dart';

import '../shared/theme/colors.dart';
import '../shared/widgets/select_date_widget.dart';
import '../shared/widgets/select_destination_widget.dart';
import '../shared/widgets/select_guests_widget.dart';

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

  @override
  State<BookingDetailsScreen> createState() => _BookingDetailsScreenState();
}

class _BookingDetailsScreenState extends State<BookingDetailsScreen> {
  var step = BookingStep.selectDate;

  @override
  Widget build(BuildContext context) {
    final textTheme = Theme.of(context).textTheme;

    return BackdropFilter(
      filter: ImageFilter.blur(sigmaX: 8.0, sigmaY: 8.0),
      child: Scaffold(
        backgroundColor: appWhite.withOpacity(0.5),
        appBar: AppBar(
          backgroundColor: Colors.transparent,
          automaticallyImplyLeading: false,
          leading: IconButton(
            onPressed: () => context.pop(),
            icon: const Icon(Icons.close),
          ),
```

```
  title: Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      TextButton(
        onPressed: () {},
        child: Text(
          'Stays',
          style: textTheme.titleMedium!.copyWith(
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
      TextButton(
        onPressed: () {},
        child: Text(
          'Experiences',
          style: textTheme.titleMedium,
        ),
      ),
    ],
  ),
  actions: const [SizedBox(width: 48.0)],
),
body: SingleChildScrollView(
  child: Padding(
    padding: const EdgeInsets.only(left: 16.0, right: 16.0, top: 16.0),
    child: Column(
      children: [
        GestureDetector(
          onTap: () {
            setState(() {
              step = BookingStep.selectDestination;
            });
          },
          child: Hero(
            tag: 'search',
            child: SelectDestinationWidget(step: step),
          ),
        ),
        GestureDetector(
          onTap: () {
            setState(() {
              step = BookingStep.selectDate;
            });
          },
          child: SelectDateWidget(step: step),
        ),
        (step == BookingStep.selectDate)
            ? const SizedBox()
            : GestureDetector(
                onTap: () {
                  setState(() {
                    step = BookingStep.selectGuests;
```

```
                });
              },
              child: SelectGuestsWidget(step: step),
            ),
        ],
      ),
    ),
  ),
),
bottomNavigationBar: (step == BookingStep.selectDate)
    ? null
    : BottomAppBar(
        padding: const EdgeInsets.symmetric(horizontal: 16.0),
        notchMargin: 0,
        color: Colors.white,
        surfaceTintColor: Colors.white,
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            TextButton(
              onPressed: () {
                // if (step == BookingStep.selectDestination) {
                //   setState(() {
                //     step = BookingStep.selectGuests;
                //   });
                // } else {
                //   setState(() {
                //     step = BookingStep.selectDestination;
                //   });
                // }
              },
              child: Text(
                'Clear all',
                style: Theme.of(context).textTheme.bodyLarge!.copyWith(
                      fontWeight: FontWeight.bold,
                      decoration: TextDecoration.underline,
                    ),
              ),
            ),
            FilledButton.icon(
              onPressed: () {},
              style: FilledButton.styleFrom(
                backgroundColor: appRed,
                minimumSize: const Size(100, 56.0),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(8.0),
                ),
              ),
              icon: const Icon(Icons.search),
              label: const Text('Search'),
            ),
          ],
        ),
      ),
```

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

select_destination

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

class SelectDestinationWidget extends StatelessWidget {
  const SelectDestinationWidget({super.key, required this.step});

  final BookingStep step;

  @override
  Widget build(BuildContext context) {
    final textTheme = Theme.of(context).textTheme;
    return Card(
      elevation: 0.0,
      clipBehavior: Clip.antiAlias,
      child: AnimatedContainer(
        duration: const Duration(milliseconds: 300),
        height: step == BookingStep.selectDestination ? 280 : 60,
        padding: const EdgeInsets.all(16.0),
        child: step == BookingStep.selectDestination
            ? Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    'Where to?',
                    style: textTheme.headlineSmall!.copyWith(
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                  const SizedBox(height: 16.0),
                  TextFormField(
                    decoration: InputDecoration(
                      contentPadding: EdgeInsets.all(16.0),
                      hintText: 'Search destination',
                      prefixIcon: const Icon(Icons.search),
                      hintStyle: textTheme.labelMedium,
                      border: OutlineInputBorder(
                        borderRadius: BorderRadius.circular(16.0),
                      ),
                    ),
                  ),
                  const SizedBox(height: 16.0),
                  SizedBox(
                    height: 128,
                    child: ListView.builder(
                      padding: EdgeInsets.zero,
```

```
          scrollDirection: Axis.horizontal,
          itemCount: 5,
          itemBuilder: (context, index) {
            return Container(
              margin: const EdgeInsets.only(right: 8.0),
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  ClipRRect(
                    borderRadius: BorderRadius.circular(16.0),
                    child: Image.network(
                      'https://picsum.photos/200/300',
                      height: 100,
                      width: 100,
                      fit: BoxFit.cover,
                    ),
                  ),
                  const SizedBox(height: 8),
                  Padding(
                    padding: const EdgeInsets.only(left: 8.0),
                    child: Text(
                      'Placeholder',
                      style: Theme.of(context)
                          .textTheme
                          .bodySmall!
                          .copyWith(
                            fontWeight: FontWeight.bold,
                          ),
                    ),
                  ),
                ],
              ),
            );
          },
        ),
      ),
    ],
  )
    .animate(delay: const Duration(milliseconds: 300))
    .fadeIn(duration: const Duration(milliseconds: 300))
: Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      Text(
        'When',
        style: textTheme.bodyMedium,
      ),
      Text(
        'I\'m flexible',
        style: textTheme.bodyMedium!.copyWith(
          fontWeight: FontWeight.bold,
        ),
      ),
```

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

Select_Date

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

import '../theme/colors.dart';
import 'app_calendar.dart';

class SelectDateWidget extends StatelessWidget {
  const SelectDateWidget({super.key, required this.step});

  final BookingStep step;

  @override
  Widget build(BuildContext context) {
    final size = MediaQuery.sizeOf(context);
    // 112 = app bar height + safe area top padding,
    // 60 = destination selection collapsed height,
    // 32 = top/bottom padding
    // 16 = margin below each card
    var expandedHeight = size.height - 112 - 60 - 32 - 20;
    return Card(
      elevation: 0.0,
      clipBehavior: Clip.antiAlias,
      child: AnimatedContainer(
          height: step == BookingStep.selectDate ? expandedHeight : 60,
          width: double.infinity,
          padding: const EdgeInsets.symmetric(
            vertical: 16.0,
            horizontal: 16.0,
          ),
          duration: const Duration(milliseconds: 300),
          child: step == BookingStep.selectDate
              ? Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      'When\'s your trip?',
                      style:
                          Theme.of(context).textTheme.headlineSmall!.copyWith(
                                fontWeight: FontWeight.bold,
                              ),
                    ),
                    const SizedBox(height: 16.0),
                    const Row(
                      children: [
                        Expanded(child: CalendarOptionsSegmentedButton()),
```

```
        ],
      ),
      const AppCalendar(),
      const Spacer(),
      const Divider(),
      SizedBox(
        height: 48,
        child: ListView(
          scrollDirection: Axis.horizontal,
          children: [
            OutlinedButton(
              onPressed: () {},
              child: const Text('Exact dates'),
            ),
            const SizedBox(width: 8.0),
            OutlinedButton(
              onPressed: () {},
              child: const Text('± 1 day'),
            ),
            const SizedBox(width: 8.0),
            OutlinedButton(
              onPressed: () {},
              child: const Text('± 2 days'),
            ),
          ],
        ),
      ),
      const Divider(),
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          TextButton(
            onPressed: () {},
            child: const Text('Skip'),
          ),
          FilledButton(
            onPressed: () {},
            style: FilledButton.styleFrom(
              backgroundColor: appRed,
              minimumSize: const Size(120, 48),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(8.0),
              ),
            ),
            child: const Text('Next'),
          ),
        ],
      ),
    ],
  )
    .animate(delay: const Duration(milliseconds: 300))
    .fadeIn(duration: const Duration(milliseconds: 300))
: Row(
```

```
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              Text(
                'When',
                style: Theme.of(context).textTheme.bodyMedium,
              ),
              Text(
                'I\'m flexible',
                style: Theme.of(context)
                    .textTheme
                    .bodyMedium!
                    .copyWith(fontWeight: FontWeight.bold),
              ),
            ],
          )),
    );
  }
}
```

Select_guests

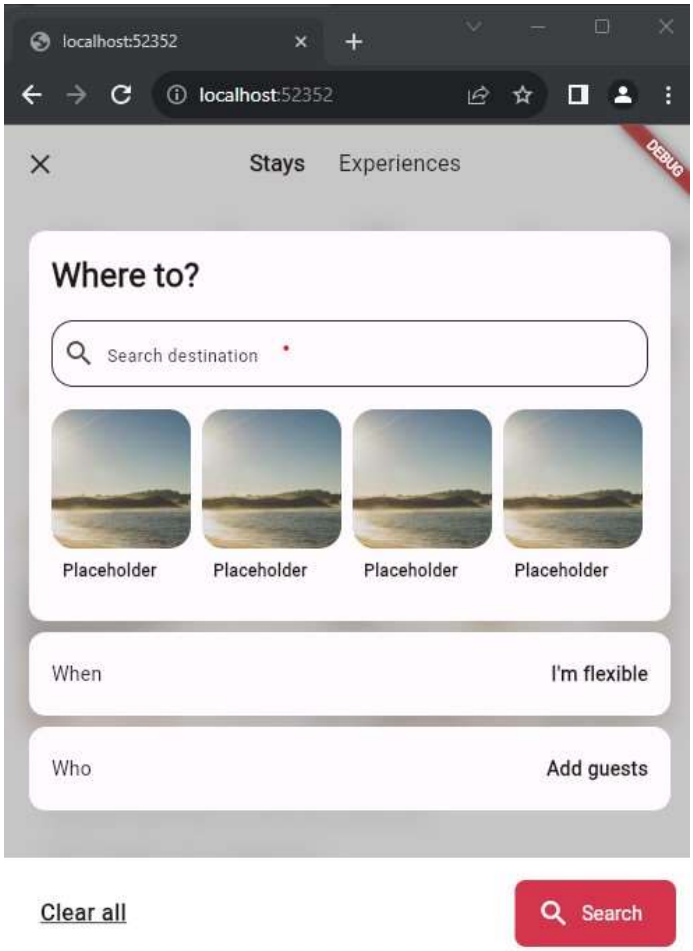
```
import 'package:flutter/material.dart';
import 'package:flutter_animate/flutter_animate.dart';
import 'package:models/models.dart';

class SelectGuestsWidget extends StatelessWidget {
  const SelectGuestsWidget({
    super.key,
    required this.step,
  });

  final BookingStep step;

  @override
  Widget build(BuildContext context) {
    final textTheme = Theme.of(context).textTheme;

    return Card(
      elevation: 0.0,
      clipBehavior: Clip.antiAlias,
      child: AnimatedContainer(
        height: step == BookingStep.selectGuests ? 274 : 60,
        width: double.infinity,
        padding: const EdgeInsets.symmetric(
          vertical: 16.0,
          horizontal: 16.0,
        ),
        duration: const Duration(milliseconds: 300),
        child: step == BookingStep.selectGuests
            ? Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
```

```
      Text(
        'Who\'s coming?',
        style: textTheme.headlineSmall!.copyWith(
          fontWeight: FontWeight.bold,
        ),
      ),
      const SizedBox(height: 16.0),
      SizedBox(
        height: 190,
        child: ListView(
          physics: const NeverScrollableScrollPhysics(),
          padding: EdgeInsets.zero,
          children: [
            _buildGuestsQuanitySelector(
              context,
              'Adults',
              'Ages 13 or above',
              () {},
              () {},
              '0',
            ),
            const Divider(),
            _buildGuestsQuanitySelector(
              context,
              'Children',
              'Ages 2-12',
              () {},
              () {},
              '0',
            ),
            const Divider(),
            _buildGuestsQuanitySelector(
              context,
              'Infants',
              'Under 2',
              () {},
              () {},
              '0',
            ),
          ],
        ),
      ),
    ],
  )
      .animate(delay: const Duration(milliseconds: 300))
      .fadeIn(duration: const Duration(milliseconds: 300))
: Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        Text(
          'Who',
          style: textTheme.bodyMedium,
        ),
```

```
            Text(
              'Add guests',
              style: Theme.of(context)
                  .textTheme
                  .bodyMedium!
                  .copyWith(fontWeight: FontWeight.bold),
            ),
          ],
        ),
    ),
  );
}

Container _buildGuestsQuanitySelector(
  BuildContext context,
  String title,
  String subtitle,
  VoidCallback onDecrement,
  VoidCallback onIncrement,
  String value,
) {
  final textTheme = Theme.of(context).textTheme;

  return Container(
    margin: const EdgeInsets.only(bottom: 8.0),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(title, style: textTheme.bodyLarge),
            Text(subtitle, style: textTheme.bodySmall),
          ],
        ),
        Row(
          children: [
            IconButton(
              onPressed: onDecrement,
              icon: const Icon(Icons.remove),
            ),
            Text(
              value,
              style:
                  textTheme.bodyMedium!.copyWith(fontWeight: FontWeight.bold),
            ),
            IconButton(
              onPressed: onIncrement,
              icon: const Icon(Icons.add),
            ),
          ],
        ),
      ],
```

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

Output:



Conclusion:
In this experiment, we have succesfully imported and inerted image in the flutter and used font style to enter text and succesfully created button for it. All concept of image , font are implemented succesfully.

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	53
Name	Lisa Sharma
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	11

Name: Lisa Sharma
Roll No: 53

MAD and PWA Lab
EXPERIMENT- 4

Aim : To create an interactive Form using form widget

Theory:

Form Widgets:

Form widgets are essential components of interactive forms, offering a range of input elements such as text fields, checkboxes, radio buttons, dropdown menus, and more. These widgets empower developers to design forms that cater to specific data input requirements. The flexibility of form widgets allows for the creation of dynamic and user-friendly interfaces, ensuring that the form adapts to the user's needs.

Form Inputs:

Text Fields:

Purpose: Allow users to input general text information.
Attributes: May include specifications such as maximum length, placeholder text, and input type (e.g., email, password).

Checkboxes:

Purpose: Enable users to make multiple selections from a list of options.
Attributes: Each checkbox typically represents a distinct option, and users can choose multiple checkboxes simultaneously.

Radio Buttons:

Purpose: Provide users with exclusive choices within a group.
Attributes: Users can select only one option from the group, making radio buttons suitable for mutually exclusive selections.

Dropdown Menus:

Purpose: Offer a space-efficient way to present a list of options for selection.
Attributes: Users click on a dropdown menu to reveal a list of choices, selecting one option from the list.

Textareas:

Purpose: Allow users to input multiline text, suitable for longer responses or comments.
Attributes: Can include settings for the number of rows and columns to determine the size of the textarea.

Date Pickers:

Purpose: Facilitate the selection of dates.
Attributes: Users can choose a specific date from a calendar interface, helping to ensure accurate date input.

File Upload:

Purpose: Enable users to submit files (e.g., images, documents).
Attributes: May include file type restrictions, maximum file size, and a browse button for users to locate and upload files from their device.

Code
login_screen.dart

```
import   'package:flutter/material.dart';
import    'package:airbnb_clone/screens/home_screen.dart';

class LoginPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        decoration: BoxDecoration(
          gradient:  LinearGradient(
          begin: Alignment.topCenter,
          end: Alignment.bottomCenter,
            colors: [Colors.transparent,  Colors.redAccent],
          ),
        ),
        child: SingleChildScrollView(
          child: Column(
            children: [
              SizedBox(height: 100.0),
              Image.asset(
              'assets/airbnb_logo.png',
              height: 100,
              ),
              SizedBox(height: 30.0),
              Text(
                'Welcome Back!',
                style: TextStyle(
                  fontSize: 28.0,
                  fontWeight: FontWeight.bold,
                  color: Colors.white,
                ),
              ),
              SizedBox(height: 20.0),
              Text(
                'Login to continue',
                style: TextStyle(
                fontSize:  18.0,
                color: Colors.white,
                ),
              ),
              SizedBox(height: 50.0),
              Padding(
                padding: const EdgeInsets.symmetric(horizontal:  40.0),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    TextField(
                      style: TextStyle(color: Colors.white),
                      decoration: InputDecoration(
                      labelText: 'Email',
                        labelStyle: TextStyle(color: Colors.white),
                        border: OutlineInputBorder(
                          borderSide:  BorderSide(color:  Colors.white),
```

```
    ),
  ),
),
SizedBox(height: 20.0),
TextField(
  style: TextStyle(color: Colors.white),
  obscureText: true,
  decoration: InputDecoration(
    labelText: 'Password',
    labelStyle: TextStyle(color: Colors.white),
    border: OutlineInputBorder(
      borderSide: BorderSide(color: Colors.white),
    ),
  ),
),
SizedBox(height: 20.0),
Align(
  alignment: Alignment.centerRight,
  child: TextButton(
    onPressed: () {
      // Forgot password logic
    },
    child: Text(
      'Forgot Password?',
      style: TextStyle(color: Colors.white),
    ),
  ),
),
SizedBox(height: 20.0),
ElevatedButton(
onPressed: () {
    // Login logic here
  },
  child: Text('Login'),
  style: ElevatedButton.styleFrom(
    primary: Colors.white,
    onPrimary: Colors.blue,
    padding: EdgeInsets.symmetric(horizontal: 40.0),
  ),
),
SizedBox(height: 20.0),
GestureDetector(
onTap: () {
    Navigator.pushReplacement(
      context,
      MaterialPageRoute(
        builder: (context) => HomeScreen(),
      ),
    );
  },
  child: Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Text(
        'Don\'t have an account?',
        style: TextStyle(color: Colors.white),
      ),
```

```
                    SizedBox(width: 5.0),
                    Text(
                      'Sign Up',
                      style: TextStyle(
                        fontWeight: FontWeight.bold,
                        color: Colors.white,
                      ),
                    ),
                  ],
                ),
              ),
            ],
          ),
        ),
        SizedBox(height: 20.0),
        Text(
          'or',
          style: TextStyle(
            fontSize: 18.0,
            color: Colors.white,
          ),
        ),
        SizedBox(height: 20.0),
        ElevatedButton.icon(
        onPressed: () {
            // Social media login logic here
          },
      ),
    ),
  );
 }
}
//
// class HomeScreen extends StatelessWidget {

//  @override
//  Widget build(BuildContext context) {
//    return Scaffold(
//      appBar: AppBar(
//        title: Text('Home'),
//      ),
//      body: Center(
//        child: Text('Welcome to Home Screen!'),
//      ),
//    );
//  }
// }

void main() {
 runApp(MaterialApp(
 home: LoginPage(),
 ));
}
```

Output:

Conclusion: In this experiment , we have successfully created form using form widget and create a login page for my clone application, various properties of form are implemented successfully in the above experiment.

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	53
Name	Lisa Sharma
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	12

Name: Lisa Sharma
Roll No: 53

MAD and PWA Lab
Experiment - 5

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

Theory:

Navigation:
Navigation is a fundamental aspect of mobile app development that involves transitioning between different screens or pages. In Flutter, the Navigator class plays a central role in managing the navigation stack, allowing developers to push and pop routes as users move through the app.

Navigator Class:
The Navigator class handles the navigation stack, maintaining a history of routes. The push method adds a new route to the stack, typically triggered by user actions. The pop method removes the current route from the stack, enabling backward navigation.
Routing:
Routing in Flutter involves defining and organizing the paths or routes within the application. Routes represent different screens or pages, and their effective use is crucial for structuring the app's architecture.

MaterialPageRoute and CupertinoPageRoute:

MaterialPageRoute is employed in apps following Material Design principles, providing a standard Android-style transition between screens.
CupertinoPageRoute is used for iOS-style designs, ensuring a consistent and native user experience.
Named Routes:

Named routes offer a more organized and readable approach to navigation.
By assigning names to routes, such as '/details', developers can easily navigate to

specific screens using Navigator.pushNamed.

Gestures in Flutter:
Gestures enhance user interaction by allowing the app to respond to touch or mouse inputs. In Flutter, the GestureDetector widget is instrumental in recognizing and handling various gestures.

GestureDetector Widget:

The GestureDetector widget is used to detect a variety of gestures, including taps, drags, and long presses.
By wrapping UI components with GestureDetector, developers can specify callback functions to execute when specific gestures are detected.
Gesture Recognizers:

Flutter provides gesture recognizers for more complex gestures, such as panning, pinching, and swiping.
These recognizers, when combined with a GestureDetector, enable the app to respond to a broader range of user inputs.
InkWell and InkWell:

The InkWell and InkResponse widgets bring a material ripple effect to touchable UI components.
Integrating these widgets enhances the visual feedback during user interactions, contributing to a more polished and intuitive user experience.

Code:

Booking_details.dart

```
import 'dart:ui';

import 'package:flutter/material.dart';
import 'package:go_router/go_router.dart';
import 'package:models/models.dart';

import '../shared/theme/colors.dart';
import '../shared/widgets/select_date_widget.dart';
import '../shared/widgets/select_destination_widget.dart';
import '../shared/widgets/select_guests_widget.dart';

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

```dart
  @override
  State<BookingDetailsScreen> createState() => _BookingDetailsScreenState();
}

class _BookingDetailsScreenState extends State<BookingDetailsScreen> {
  var step = BookingStep.selectDate;

  @override
  Widget build(BuildContext context) {
    final textTheme = Theme.of(context).textTheme;

    return BackdropFilter(
      filter: ImageFilter.blur(sigmaX: 8.0, sigmaY: 8.0),
      child: Scaffold(
        backgroundColor: appWhite.withOpacity(0.5),
        appBar: AppBar(
          backgroundColor: Colors.transparent,
          automaticallyImplyLeading: false,
          leading: IconButton(
            onPressed: () => context.pop(),
            icon: const Icon(Icons.close),
          ),
          title: Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              TextButton(
                onPressed: () {},
                child: Text(
                  'Stays',
                  style: textTheme.titleMedium!.copyWith(
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ),
              TextButton(
                onPressed: () {},
                child: Text(
                  'Experiences',
                  style: textTheme.titleMedium,
                ),
              ),
            ],
          ),
          actions: const [SizedBox(width: 48.0)],
        ),
        body: SingleChildScrollView(
          child: Padding(
            padding: const EdgeInsets.only(left: 16.0, right: 16.0, top: 16.0),
            child: Column(
              children: [
                GestureDetector(
                  onTap: () {
                    setState(() {
                      step = BookingStep.selectDestination;
                    });
                  },
                  child: Hero(
                    tag: 'search',
                    child: SelectDestinationWidget(step: step),
                  ),
                ),
                GestureDetector(
                  onTap: () {
```

```
                setState(() {
                  step = BookingStep.selectDate;
                });
              },
              child: SelectDateWidget(step: step),
            ),
            (step == BookingStep.selectDate)
                ? const SizedBox()
                : GestureDetector(
                    onTap: () {
                      setState(() {
                        step = BookingStep.selectGuests;
                      });
                    },
                    child: SelectGuestsWidget(step: step),
                  ),
          ],
        ),
      ),
    ),
    bottomNavigationBar: (step == BookingStep.selectDate)
        ? null
        : BottomAppBar(
            padding: const EdgeInsets.symmetric(horizontal: 16.0),
            notchMargin: 0,
            color: Colors.white,
            surfaceTintColor: Colors.white,
            child: Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                TextButton(
                  onPressed: () {
                    // if (step == BookingStep.selectDestination) {
                    //   setState(() {
                    //     step = BookingStep.selectGuests;
                    //   });
                    // } else {
                    //   setState(() {
                    //     step = BookingStep.selectDestination;
                    //   });
                    // }
                  },
                  child: Text(
                    'Clear all',
                    style: Theme.of(context).textTheme.bodyLarge!.copyWith(
                          fontWeight: FontWeight.bold,
                          decoration: TextDecoration.underline,
                        ),
                  ),
                ),
                FilledButton.icon(
                  onPressed: () {},
                  style: FilledButton.styleFrom(
                    backgroundColor: appRed,
                    minimumSize: const Size(100, 56.0),
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(8.0),
                    ),
                  ),
                  icon: const Icon(Icons.search),
                  label: const Text('Search'),
                ),
              ],
            ),
```

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

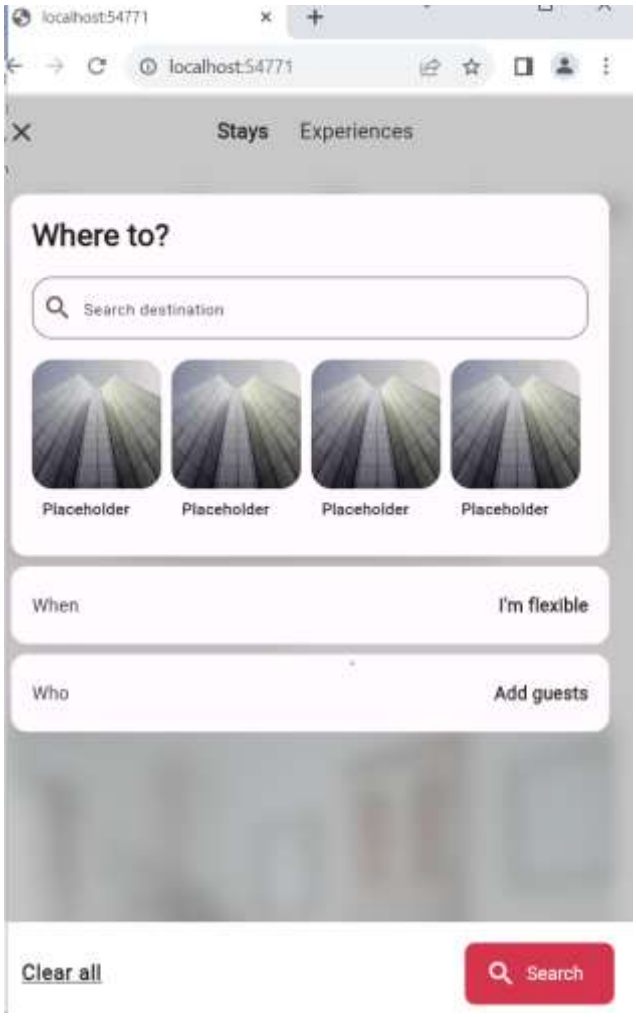
select_guests_widget.dart

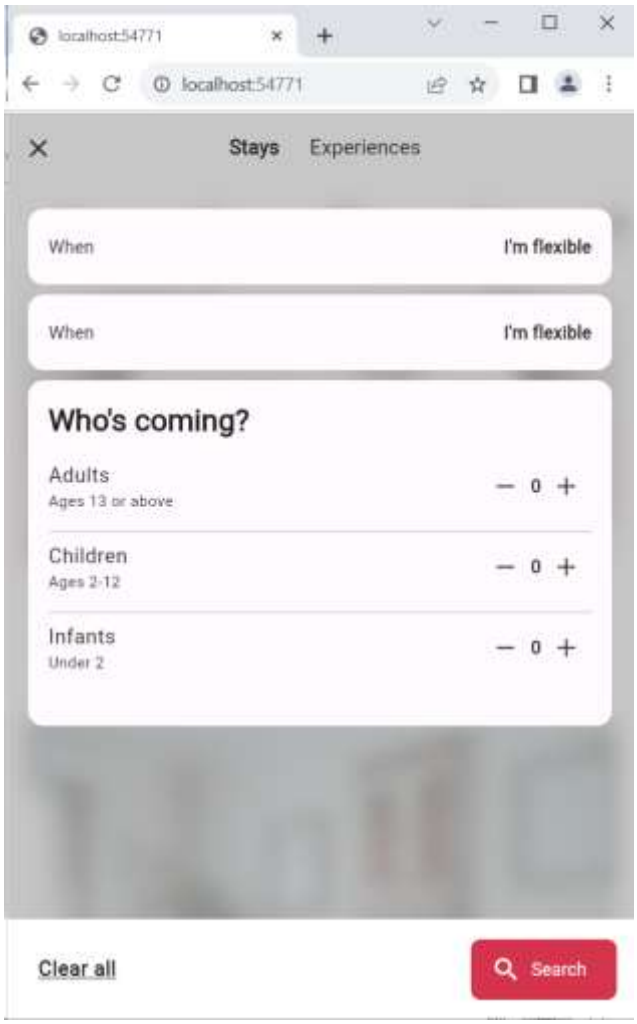
```
import 'package:flutter/material.dart';
import 'package:flutter_animate/flutter_animate.dart';
import 'package:models/models.dart';

class SelectGuestsWidget extends StatelessWidget {
  const SelectGuestsWidget({
    super.key,
    required this.step,
  });

  final BookingStep step;

  @override
  Widget build(BuildContext context) {
    final textTheme = Theme.of(context).textTheme;

    return Card(
      elevation: 0.0,
      clipBehavior: Clip.antiAlias,
      child: AnimatedContainer(
        height: step == BookingStep.selectGuests ? 274 : 60,
        width: double.infinity,
        padding: const EdgeInsets.symmetric(
          vertical: 16.0,
          horizontal: 16.0,
        ),
        duration: const Duration(milliseconds: 300),
        child: step == BookingStep.selectGuests
            ? Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    'Who\'s coming?',
                    style: textTheme.headlineSmall!.copyWith(
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                  const SizedBox(height: 16.0),
                  SizedBox(
                    height: 190,
                    child: ListView(
                      physics: const NeverScrollableScrollPhysics(),
                      padding: EdgeInsets.zero,
                      children: [
                        _buildGuestsQuanitySelector(
                          context,
                          'Adults',
                          'Ages 13 or above',
                          () {},
                          () {},
                          '0',
                        ),
                        const Divider(),
                        _buildGuestsQuanitySelector(
                          context,
```

```
                        'Children',
                        'Ages 2-12',
                        () {},
                        () {},
                        '0',
                      ),
                      const Divider(),
                      _buildGuestsQuanitySelector(
                        context,
                        'Infants',
                        'Under 2',
                        () {},
                        () {},
                        '0',
                      ),
                    ],
                  ),
                ),
              ],
            )
              .animate(delay: const Duration(milliseconds: 300))
              .fadeIn(duration: const Duration(milliseconds: 300))
          : Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                Text(
                  'Who',
                  style: textTheme.bodyMedium,
                ),
                Text(
                  'Add guests',
                  style: Theme.of(context)
                      .textTheme
                      .bodyMedium!
                      .copyWith(fontWeight: FontWeight.bold),
                ),
              ],
            ),
    ),
  );
}

Container _buildGuestsQuanitySelector(
  BuildContext context,
  String title,
  String subtitle,
  VoidCallback onDecrement,
  VoidCallback onIncrement,
  String value,
) {
  final textTheme = Theme.of(context).textTheme;

  return Container(
    margin: const EdgeInsets.only(bottom: 8.0),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(title, style: textTheme.bodyLarge),
            Text(subtitle, style: textTheme.bodySmall),
          ],
        ),
```

```
          Row(
            children: [
              IconButton(
                onPressed: onDecrement,
                icon: const Icon(Icons.remove),
              ),
              Text(
                value,
                style:
                    textTheme.bodyMedium!.copyWith(fontWeight: FontWeight.bold),
              ),
              IconButton(
                onPressed: onIncrement,
                icon: const Icon(Icons.add),
              ),
            ],
          ),
        ],
      ),
    );
  }
}
```

Output:


localhost:54771
localhost:54771
Stays
Experiences
Where to?
Search destination
Placeholder
Placeholder
Placeholder
Placeholder
When
I'm flexible
Who
Add guests
Clear all
Search



Conclusion:

In this experiment, we have successfully created routing in the bottom navigation bar and connected all pages successfully using Navigator class and implemented it successfully.

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	53
Name	Lisa Sharma
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:	15

Name: Lisa Sharma
Roll No: 53

MAD and PWA Lab
Experiment - 6

Aim: To Connect Flutter UI with FireBase database

Theory:

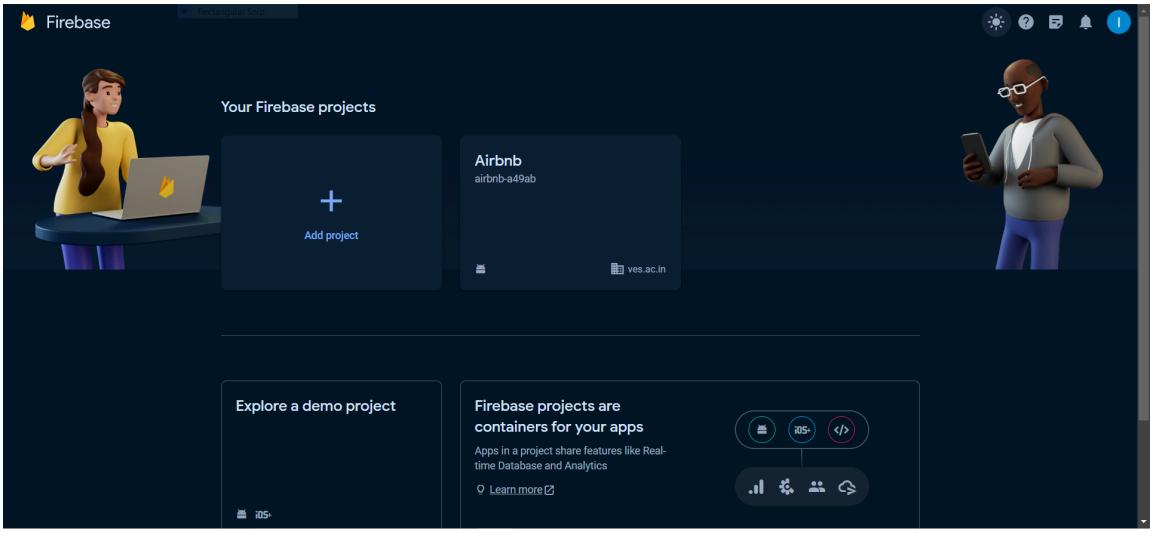
Prerequisites

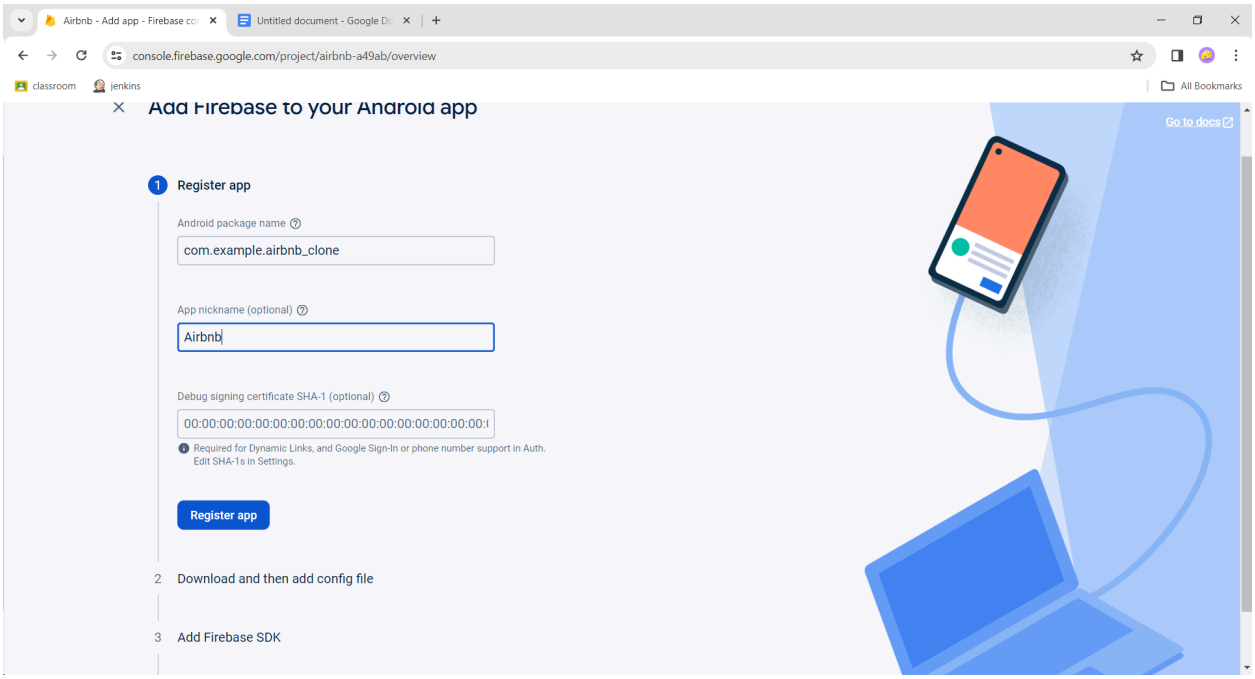
To complete this tutorial, you will need:

- A Google account to use Firebase.
- Developing for iOS will require XCode.
- To download and install Flutter.
- To download and install Android Studio and Visual Studio Code.
- It is recommended to install plugins for your code editor:
    - Flutter and Dart plugins installed for Android Studio.
    - Flutter extension installed for Visual Studio Code.

Create a Firebase Project:

First, log in with your Google account to manage your Firebase projects. From within the Firebase dashboard, select the Create new project button and give it a name:





Go to the Firebase Console and create a new project.
Add your Flutter app to the Firebase project:

Register your app in the Firebase project, and follow the instructions to download the configuration files (google-services.json for Android, GoogleService-Info.plist for iOS).

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

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

com.example.flutterfirebaseexample

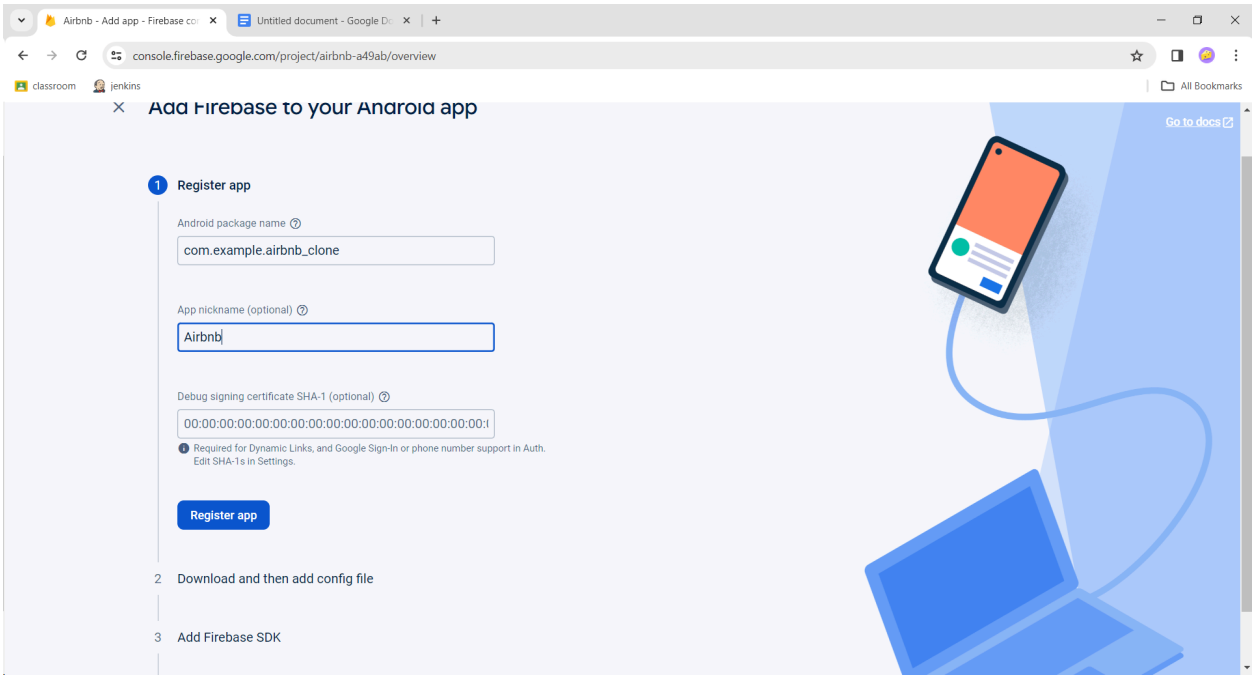
Once you've decided on a name, open android/app/build.gradle in your code editor and update the applicationId to match the Android package name:
android/app/build.gradle

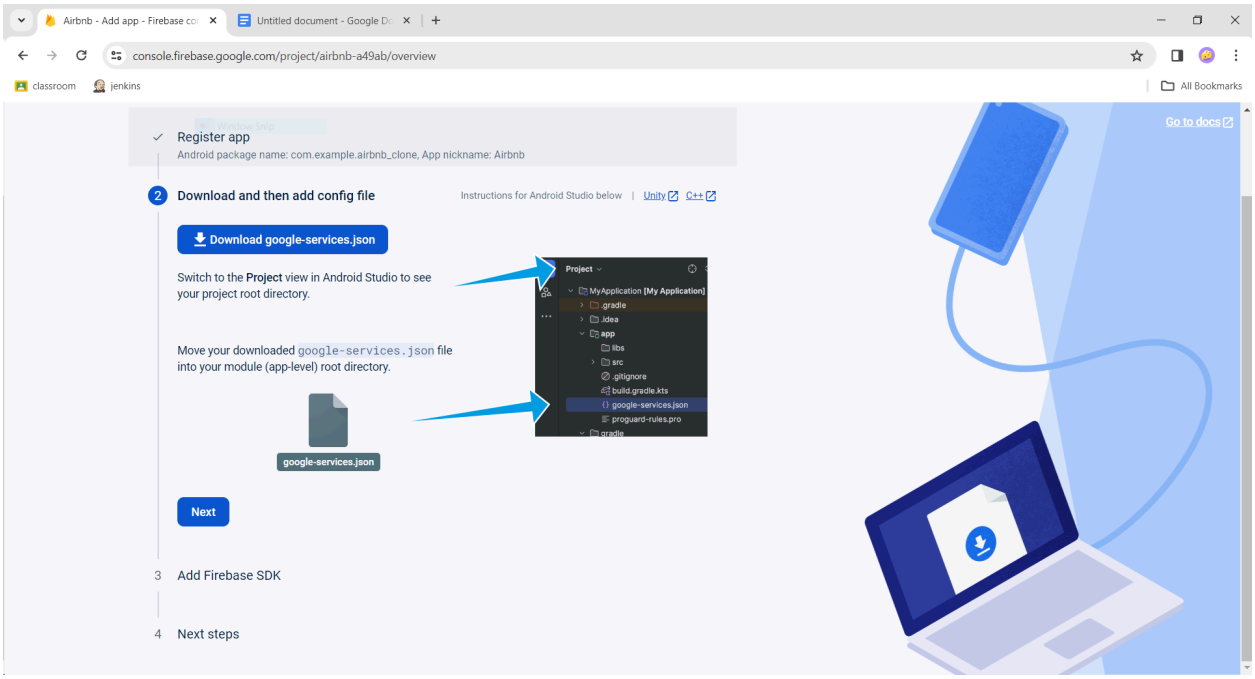
```
...
defaultConfig {
    // TODO: Specify your own unique Application ID (https://developer.android.com/studio/build/application-id.html).
    applicationId 'com.example.flutterfirebaseexample'
    ...
}
...
```

Downloading the Config File

The next step is to add the Firebase configuration file into our Flutter project. This is important as it contains the API keys and other critical information for Firebase to use.

Select Download `google-services.json` from this page:





2. Add Firebase to your Flutter project:
Add Dependencies:

Open your pubspec.yaml file and add the necessary dependencies:

yaml

```yaml
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2
  flutter_animate: ^4.2.0+1
  go_router: ^10.1.2
  syncfusion_flutter_datepicker: ^22.2.12
  intl: ^0.18.1
  modal_bottom_sheet: ^2.1.2
  equatable: ^2.0.5
  uuid: ^4.1.0
  dots_indicator: ^3.0.0
  firebase_core: ^2.25.3
  firebase_auth: ^4.17.3
  cloud_firestore: ^4.15.4
```

Code:

main.dart

```dart
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'shared/navigation/app_router.dart';
import 'shared/theme/colors.dart';
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: FirebaseOptions(
        apiKey: "AIzaSyCDqrgiDjoGFckVQkbx2proOWI_rDBLR3s",
        appId: "1:95463089972:android:e8d8294c8f9d651f136761",
        messagingSenderId: "95463089972",
        projectId: "airbnb-a49ab"),
  );
  runApp(const MyApp());
}

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

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp.router(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        useMaterial3: true,
        colorScheme: ColorScheme.fromSeed(seedColor: appRed),
      ),
      routerConfig: AppRouter().router,
    );
  }
}
```

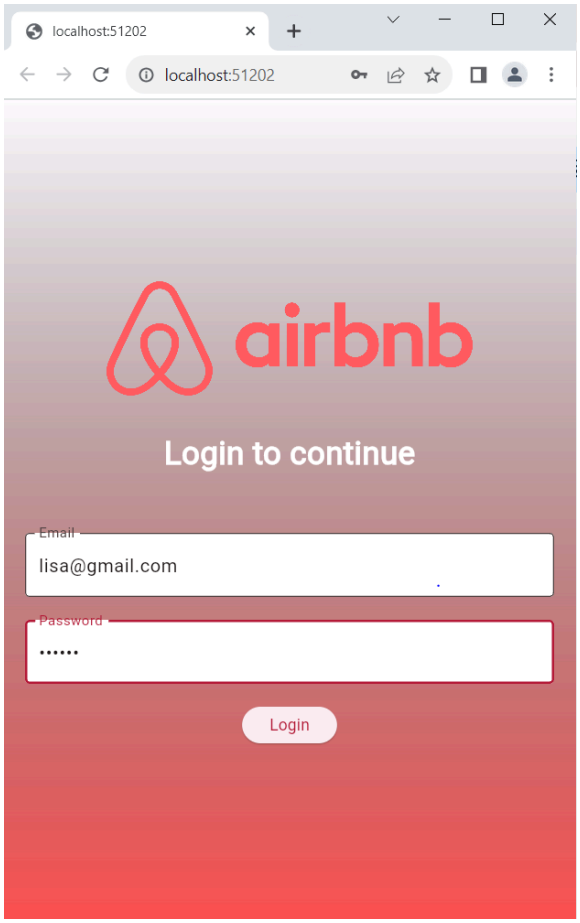
login.dart

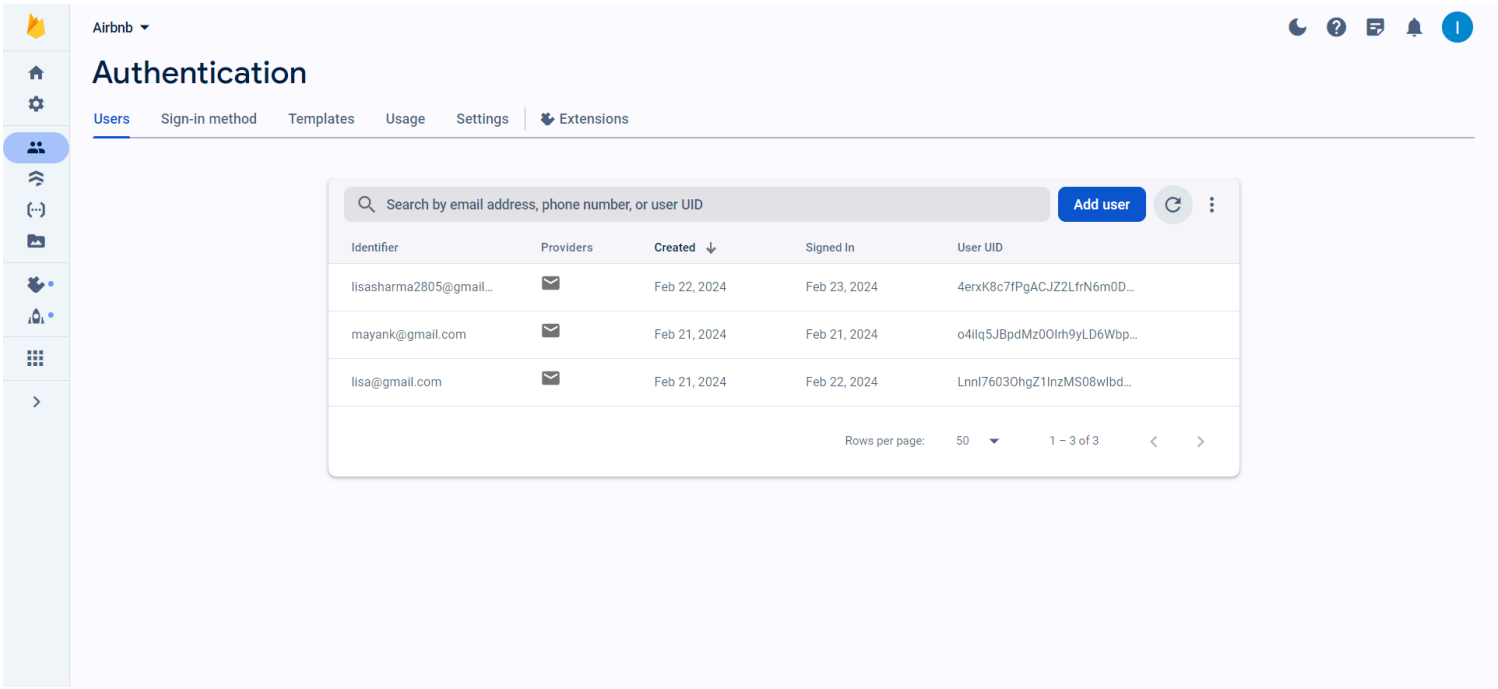
```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:airbnb_clone/screens/home_screen.dart';

class LoginPage extends StatelessWidget {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();

  Future<void> _signInWithEmailAndPassword(BuildContext context) async {
    try {
      final UserCredential userCredential =
      await _auth.signInWithEmailAndPassword(
        email: _emailController.text,
        password: _passwordController.text,
      );
      if (userCredential.user != null) {
        Navigator.pushReplacement(
          context,
          MaterialPageRoute(builder: (context) => HomeScreen()),
        );
      }
    } catch (e) {
      print("Failed to sign in with email and password: $e");
      // Handle error messages here
    }
  }

  @override
```

```
Widget build(BuildContext context) {
  return Scaffold(
    body: Container(
      decoration: BoxDecoration(
        gradient: LinearGradient(
          begin: Alignment.topCenter,
          end: Alignment.bottomCenter,
          colors: [Colors.transparent, Colors.redAccent],
        ),
      ),
      child: Center(
        child: SingleChildScrollView(
          child: Container(
            padding: EdgeInsets.all(20.0),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Image.asset(
                  'assets/airbnb_logo.png',
                  height: 100,
                ),
                SizedBox(height: 30.0),
                Text(
                  'Login to continue',
                  style: TextStyle(
                    fontSize: 28.0,
                    fontWeight: FontWeight.bold,
                    color: Colors.white,
                  ),
                ),
                SizedBox(height: 50.0),
                TextField(
                  controller: _emailController,
                  decoration: InputDecoration(
                    labelText: 'Email',
                    border: OutlineInputBorder(),
                    fillColor: Colors.white,
                    filled: true,
                  ),
                ),
                SizedBox(height: 20.0),
                TextField(
                  controller: _passwordController,
                  obscureText: true,
                  decoration: InputDecoration(
                    labelText: 'Password',
                    border: OutlineInputBorder(),
                    fillColor: Colors.white,
                    filled: true,
                  ),
                ),
```

```
                SizedBox(height: 20.0),
                ElevatedButton(
                  onPressed: () => _signInWithEmailAndPassword(context),
                  child: Text('Login'),
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }
}

void main() {
  runApp(MaterialApp(
    home: LoginPage(),
  ));
}
```

Output:





Conclusion:

In this experiment, we have successfully connected firebase database and authenticated using google signin and email and password with out flutter application successfully.

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.	53
Name	Lisa Sharma
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

Name: Lisa Sharma
Div: D15A
Roll No: 53
Batch: C

PWA EXPERIMENT 7

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

Theory:-

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

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

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

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.).

Folder Structure and icon size

plant 512 X 512.png

Plant 192 X 192.png

Index.html

```
<!DOCTYPE html>

<html>



<head>
   <meta name="apple-mobile-web-app-status-bar" content="#aa7700">
   <meta name="theme-color" content="black">
   <link rel="manifest" href="manifest.json">
   <script src="service-worker.js"></script>
   <title>
       Index
   </title>
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">


   <!--jQuery library-->
   <script
```

```
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>


   <!--Latest compiled and minified JavaScript-->

   <script

src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
   <meta name="viewport" content="width=device-width, initial-scale=1">

   <link rel="stylesheet" href="style.css">

</head>

<body>


   <nav class="navbar navbar-inverse navbar-fixed-top">

       <div class="container">

           <div class="navbar-header">

               <button type="button" class="navbar-toggle" data-toggle="collapse"

data-target="#mynavbar">

                   <span class="icon-bar"></span>

                   <span class="icon-bar"></span>

                   <span class="icon-bar"></span>

               </button>

               <a class="navbar-brand" href="index.html">Purity Plants</a>


           </div>

           <div class="collapse navbar-collapse" id="mynavbar">

               <ul class="nav navbar-nav navbar-right">
```

```
                   <li>

                       <a href="signup.html">

                           <span class="glyphicon glyphicon-user" /> Sign-Up </a>

                   </li>

                   <li>

                       <a href="login.html">

                           <span class="glyphicon glyphicon-log-in" /> Login </a>

                   </li>
```

```
            </ul>
        </div>


    </div>

</nav>
<div class="banner-image">
    <div class="container">
        <div class="banner-content" style="margin-left :25%">
            <h1>All premium plants available here</h1>
            <p>Flat 30% to our new customers</p> <br>
            <a href="product.html" class="btn btn-danger btn-lg active">Shop
Now</a>
        </div>
    </div>
</div>
<footer>
    <div class="container">
        <p style="text-align:center;">Copyright © Purity Plants. All Rights
Reserved and Contact Us: +91 8432777111 </p>
    </div>
</footer>
<script>
    // Add event listener to execute code when page loads
    window.addEventListener('load', () => {
        // Call registerSW function when page loads
        registerSW();
    });

    // Register the Service Worker
    async function registerSW() {
        // Check if browser supports Service Worker
        if ('serviceWorker' in navigator) {
            try {
                // Register the Service Worker named 'serviceworker.js'
                await navigator.serviceWorker.register('service-worker.js');
            }
            catch (e) {
                // Log error message if registration fails
```

```
                    console.error('ServiceWorker registration failed: ', e);
                }
            }
        }
        if ('Notification' in window) {
            Notification.requestPermission().then(function (permission) {
                if (permission === 'granted') {
                    console.log('Notification permission granted.');
                } else {
                    console.warn('Notification permission denied.');
                }
            });
        }
    </script>




</body>


</html>
```

Manifest.json

```
{
    "name":"PWA Tutorial",
    "short_name":"PWA",
    "start_url":"index.html",
    "display":"standalone",
    "background_color":"#5900b3",
    "theme_color":"black",
    "scope": ".",
    "description":"This is a PWA tutorial.",
    "icons":[
    {
    "src":"images/Plant 192 X 192.png ",
    "sizes":"192x192",
    "type":"image/png",
    "purpose":"any maskable"
    },
    {
```

```
   "src":"images/plant 512 X 512.png",
   "sizes":"512x512",
   "type":"image/png",
   "purpose":"any maskable"
   }
]
}
```

Service-worker.json

```
// service-worker.js


const CACHE_NAME = 'my-ecommerce-app-cache-v1';
const urlsToCache = [
 '/',
 'cart.html',
 'index.html',
 'product.html',
 'shop.html',
 'style.css',
 'success.html',
 'service-worker.js',
 'manifest.json',
 'offline.html'
 // Add more files to cache as needed
];



self.addEventListener('install', function(event) {
 event.waitUntil(
   caches.open(CACHE_NAME)

     .then(function(cache) {
       console.log('Opened cache');
       return cache.addAll(urlsToCache)

         .catch(function(error) {
```

```
          console.error('Cache.addAll error:', error);
        });
    })
```

```
 );
});

self.addEventListener('activate', function(event) {
 // Perform activation steps
 event.waitUntil(
   caches.keys().then(function(cacheNames) {
     return Promise.all(
       cacheNames.map(function(cacheName) {
         if (cacheName !== CACHE_NAME) {
           return caches.delete(cacheName);
         }
       })
     );
   })
 );
});


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


// Sync event listener
self.addEventListener('sync', function(event) {
 if (event.tag === 'syncMessage') {
   console.log("Sync successful!");
 }
});
```

```
// Push event listener
self.addEventListener("push", function (event) {
 if (event && event.data) {
   try {
     var data = event.data.json();
     if (data && data.method === "pushMessage") {
       console.log("Push notification sent");
       self.registration.showNotification("Ecommerce website", { body: data.message
});
     }
   } catch (error) {
     console.error("Error parsing push data:", error);
   }
 }
});

self.addEventListener('activate', async () => {
 if (Notification.permission !== 'granted') {
   try { const permission = await
     Notification.requestPermission();
     if (permission === 'granted') {
       console.log('Notification permission granted.');
     } else {
       console.warn('Notification permission denied.');
     }
   } catch (error) { console.error('Failed to request notification
     permission:', error);
   }
 }
});

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

```
        fulfill(response);
      } else {
        reject(new Error("Response not found"));
      }
    })
    .catch(function (error) {
      reject(error);
    });
});
};


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

var addToCache = function (request) {
 return caches.open("offline").then(function (cache) {
   return fetch(request).then(function (response) {
     return cache.put(request, response.clone()).then(function () {
       return response;
     });
   });
});
};
```

Product.html

```
<!DOCTYPE html>
<html>
```

```
   <head>
   <title>
       product
   </title>
       <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" >


       <!--jQuery library-->
       <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>


       <!--Latest compiled and minified JavaScript-->
       <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
       <meta name="viewport" content="width=device-width, initial-scale=1">
       <link rel = "stylesheet" href = "style.css">
```

```
   </head>
       <body>
           <nav class = "navbar navbar-inverse navbar-fixed-top">
                   <div class ="container">
                       <div class ="navbar-header">
                           <button type="button" class ="navbar-toggle"
data-toggle="collapse" data-target="#mynavbar">
                               <span class="icon-bar"></span>
                               <span class="icon-bar"></span>
                               <span class="icon-bar"></span>
                           </button>
                           <a class="navbar-brand" href="index.html">Purity Plants</a>


                       </div>
                       <div class="collapse navbar-collapse" id="mynavbar">
```

```
                        <ul class="nav navbar-nav navbar-right">
                            <li>
                                <a href="cart.html">
                                    <span class="glyphicon
glyphicon-shopping-cart"> Cart </span> </a>
                            </li>
                            <li>
                                <a href="setting.html">
                                    <span class="glyphicon glyphicon-user">
Setting</span> </a>
                            </li>
                            <li>
                                <a href="index.html">
                                    <span class="glyphicon glyphicon-log-out">
Logout</span></a>
                            </li>
                        </ul>
                    </div>

                </div>


            </nav>


            <div class =" container" style="margin-top: 5%;">


            <div class ="jumbotron">
```

```
            <h1> Welcome to our Purity Plants! </h1>
            <p>We have the best quality and rare breed of plants at our botany </p>
            </div>
```

```
<div class="row text-center">
<div class=" col-md-3 col-sm-6 thumbnail " >
<img src="/product-images/1.webp">
<div class="caption">
<h2>Corpse Flower</h2>
<p>Large foul-smelling bloom.</p>

</div>
<div class=" btn btn-primary btn-block btn-md btn-success"> 300
</div>
</div>

<div class=" col-md-3 col-sm-6 thumbnail " >
<img src="/product-images/2.webp">
<div class="caption">
<h2>Jade Vine</h2>
<p>Turquoise flowers in clusters.</p>

</div>
<div class=" btn btn-primary btn-block btn-md btn-success"> 240
</div>
</div>

<div class=" col-md-3 col-sm-6 thumbnail " >
<img src="/product-images/3.webp">
<div class="caption">
<h2>Wollemi Pine</h2>
<p>"Living fossil" from Australia</p>
```

```
</div>
<div class=" btn btn-primary btn-block btn-md btn-success"> 290
</div>
</div>

<div class=" col-md-3 col-sm-6 thumbnail " >
<img src="/product-images/4.webp">
```

```
<div class="caption">
<h2>Ghost Orchid</h2>
<p>Ghostly white floating flowers.</p>
</div>

<div class=" btn btn-primary btn-block btn-md btn-success"> 500
</div>

</div>

</div>

<div class="row text-center">

<div class=" col-md-3 col-sm-6 thumbnail " >
<img src="/product-images/5.webp">
<div class="caption">
<h2>Lithops</h2>
<p>Succulents resembling rocks.</p>
</div>
<div class=" btn btn-primary btn-block btn-md btn-success"> 499
```

```
                </div>
                </div>



                <div class=" col-md-3 col-sm-6 thumbnail " >
                <img src="/product-images/6.webp">
                <div class="caption">
                <h2>Black Bat Flower</h2>
                <p>Dark purple bat-like flowers.</p>
                </div>
                <div class=" btn btn-primary btn-block btn-md btn-success"> 129
                </div>
                </div>



                <div class=" col-md-3 col-sm-6 thumbnail " >
                <img src="/product-images/7.webp">
                     <div class="caption">
                    <h2>Venus Flytrap</h2>
                    <p>Carnivorous plant trapping insects.</p>
```

```
    </div>
                     <div class=" btn btn-primary btn-block btn-md btn-success"> 199
</div>



</div>


<div class=" col-md-3 col-sm-6 thumbnail " >
<img src="/product-images/8.webp">
<div class="caption">
<h2>Kadupul Flower </h2>
<p>Night-blooming Sri Lankan flower.</p>
```

```
</div>
<div class=" btn btn-primary btn-block btn-md btn-success"> 209
</div>


</div>
</div>




<div class="row text-center">



<div class=" col-md-3 col-sm-6 thumbnail " >
<img src="/product-images/9.webp">
  <div class="caption">
<h2>Rainbow Eucalyptus</h2>
<p>Multicolored peeling bark.</p>
</div>
                <div class=" btn btn-primary btn-block btn-md btn-success">299 </div>


</div>



<div class=" col-md-3 col-sm-6 thumbnail " >
<img src="/product-images/10.webp">
 <div class="caption">
<h2>Corkscrew Vine</h2>
<p>Fragrant spiral-shaped flowers.</p>
</div>
                <div class=" btn btn-primary btn-block btn-md btn-success">300</div>
</div>


<div class=" col-md-3 col-sm-6 thumbnail " >
```

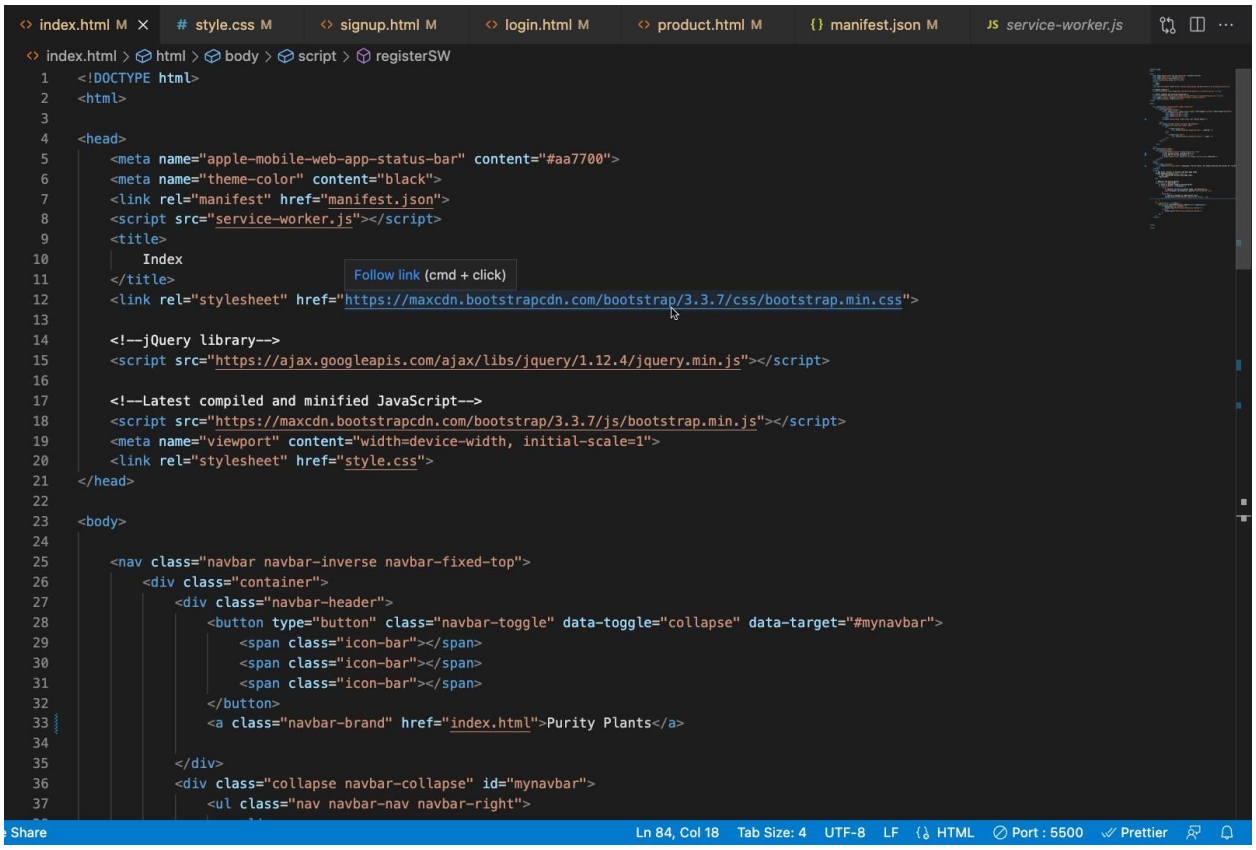
```
                <img src="/product-images/11.webp">
                  <div class="caption">
                <h2>Night-blooming Cereus:</h2>
                <p>Fragrant nocturnal blooms.</p>
                </div>
                 <div class=" btn btn-primary btn-block btn-md btn-success">249 </div>
                </div>


                <div class=" col-md-3 col-sm-6 thumbnail " >
                <img src="/product-images/12.webp">
                  <div class="caption">
                <h2>Bleeding Tooth Fungus</h2>
                <p>Blood-like liquid oozes.</p>
                </div>
                 <div class=" btn btn-primary btn-block btn-md btn-success">249 </div>
                </div>
                </div>
                </div>

                <footer style="margin-top: 5%; margin-bottom:.5%;">
                                <div class="container" >
                                    <p style="text-align:center;">Copyright © Purity
Plants. All Rights Reserved and Contact Us: +91 85321 11111 </p>
                                </div>
                            </footer>



            </body>
        </html>
```

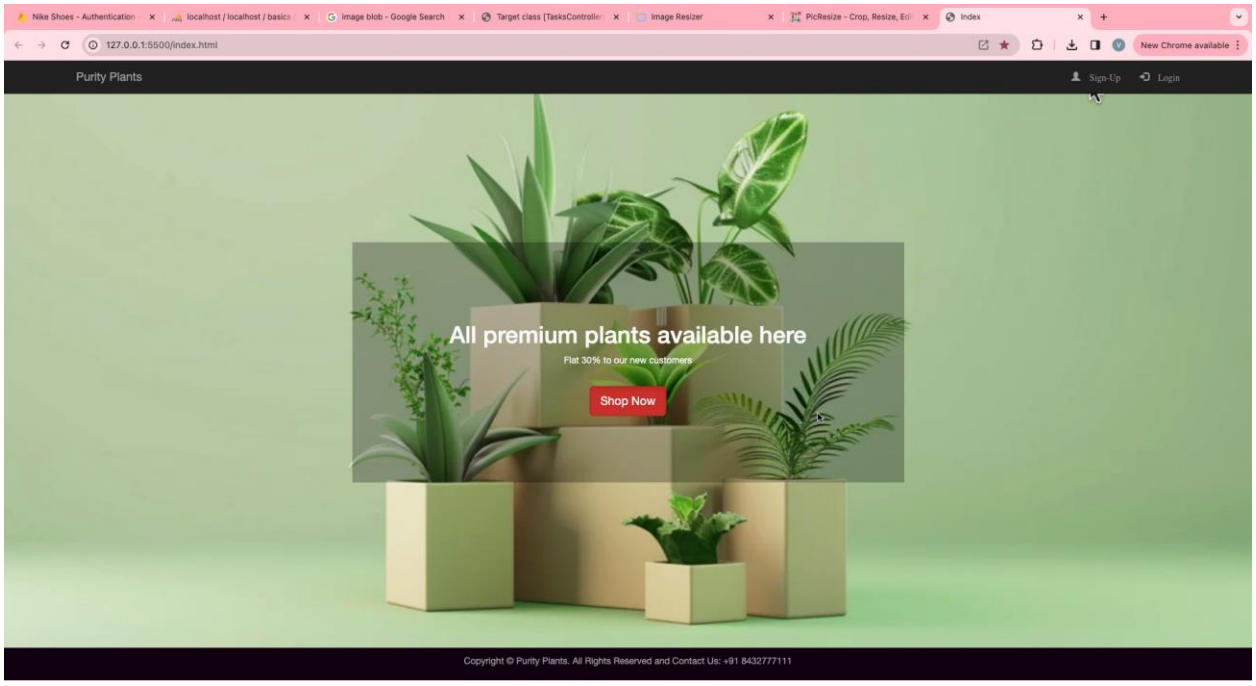
Style.css

```
.banner-image
{padding-top: 75px;
padding-bottom: 50px; text-
align: center;
color: #f8f8f8;
background: url(homepage.png) no-repeat center center;
background-size: cover;
}
```

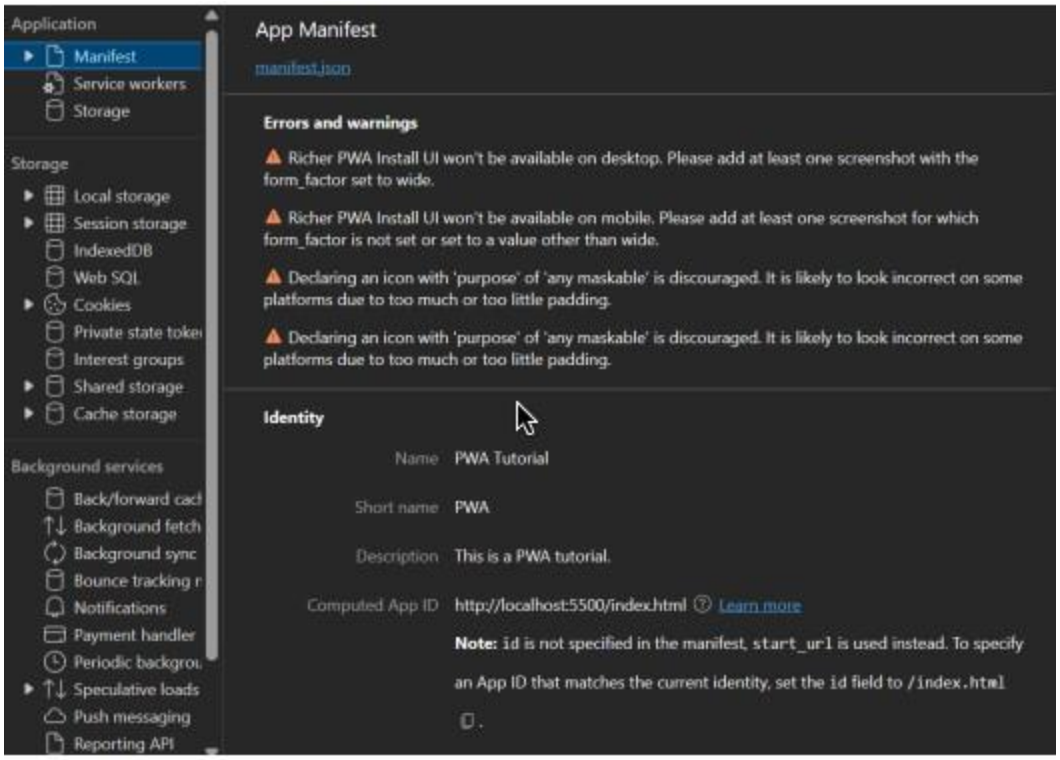
```
.banner-content{
 position: relative; padding-
 top: 6%;
padding-bottom: 6%;


margin-top: 12%; margin-
bottom: 12%;
background-color: rgba(0, 0, 0, 0.3);
width: 50%; text-
align:center;
}
footer
{
padding: 10px 0;
background-color: #110011;
color:#9d9d9d;
bottom: 0;
width: 100%;
}
.container{
 width:90%;
margin:auto;
overflow:hidden;


}
```

Starting the Server
:



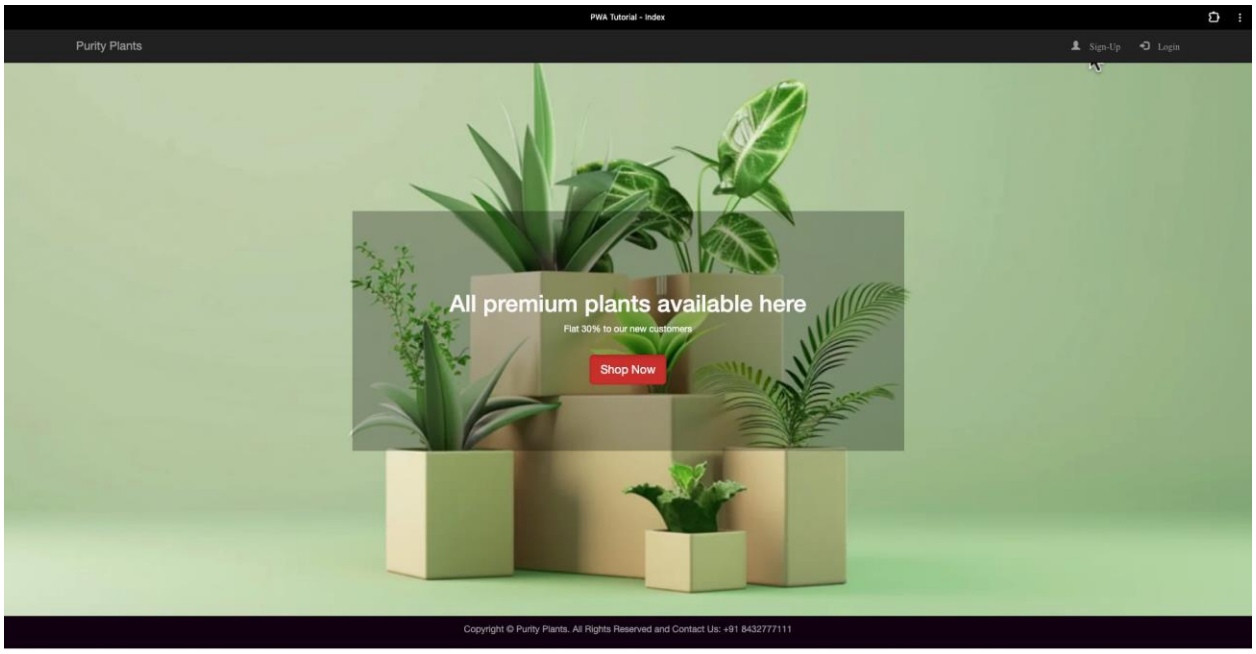


Now go to developer options -> Application->Manifest



Now to install PWA , click on Three dots(...) -> Apps -> Install PWA


flutter 2
Sublime Text
PWA Tutorial
2c4244fe-dcbd-42...1.jpg
ZEB-MAX_NI..._UM.pdf
Simulator Screen Recordi...8.56.mp4
java
removeVowels.java
Other
Screenshots



Conclusion: Hence We wrote meta data of our Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”. And it is currently added Successfully on the Desktop

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.	53
Name	Lisa Sharma
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Name: Lisa Sharma
Division: D15A
Roll No: 53
Batch: C

Experiment No 8

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

Service Worker

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

Things to note about Service Worker:

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

What can we do with Service Workers?

- You can dominate **Network Traffic**

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

- You can **Cache**

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

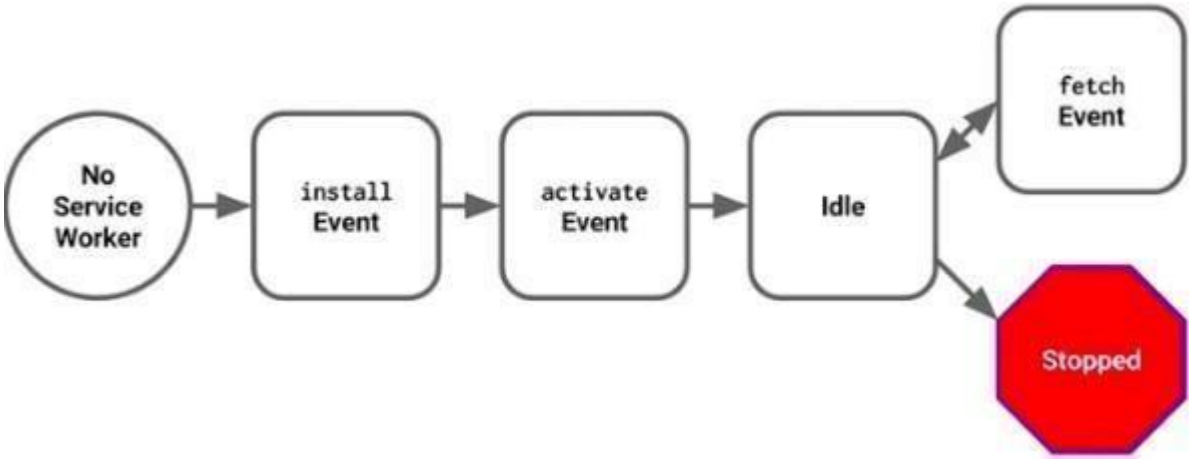
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can't we do with Service Workers?

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

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

Service Worker Cycle



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

- Registration
- Installation
- Activation

Registration

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

main.js

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

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

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

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

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

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

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

main.js

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

Installation

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

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

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

Activation

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

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

service-worker.js

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

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

Code in service-worker.js

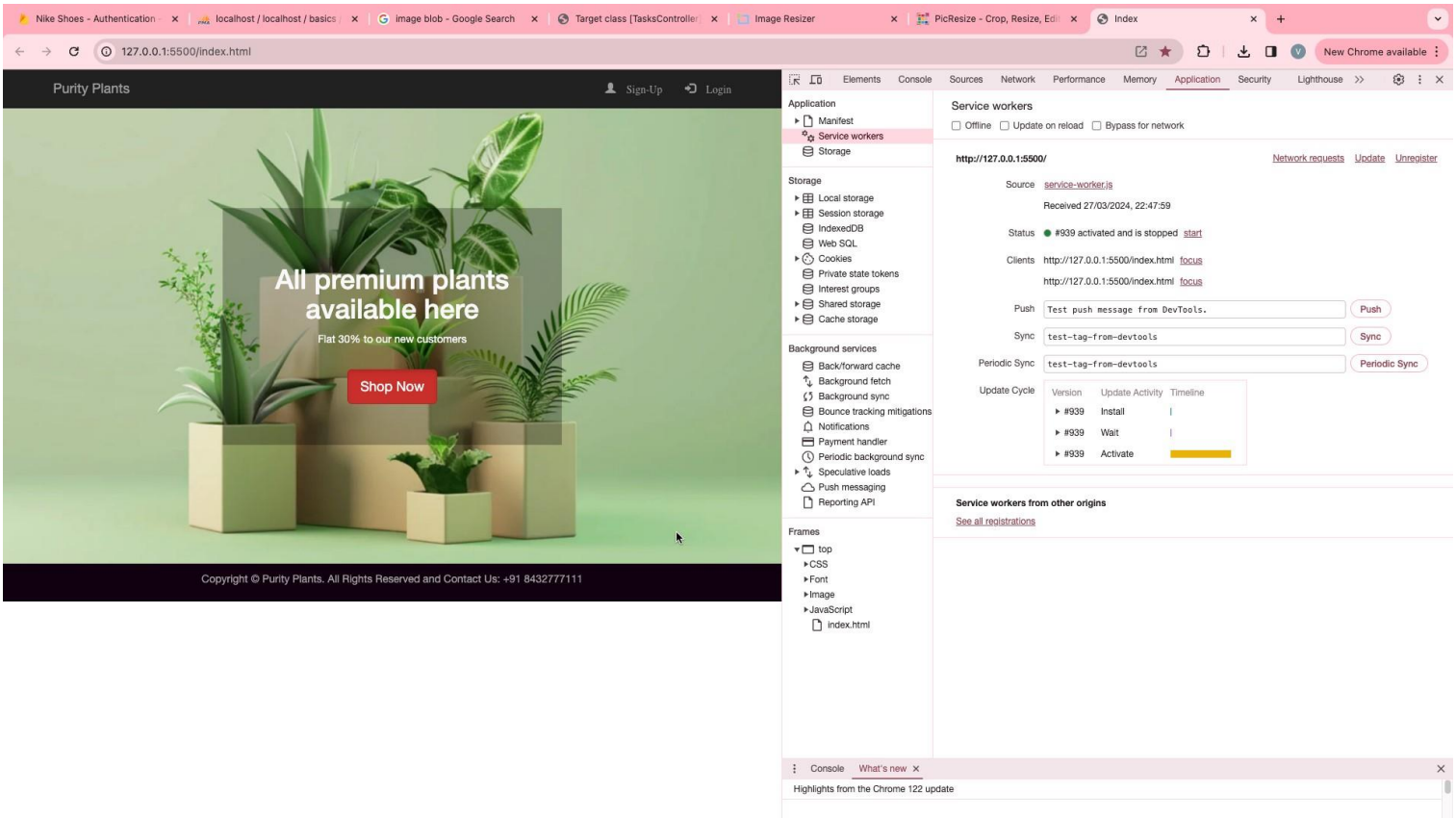
```
self.addEventListener('install', function(event) {

 event.waitUntil(
   caches.open(CACHE_NAME)

     .then(function(cache) {

       console.log('Opened cache');
       return cache.addAll(urlsToCache)

         .catch(function(error) {

           console.error('Cache.addAll error:', error);
         });

     })
 );

});



self.addEventListener('activate', function(event) {
 // Perform activation steps

 event.waitUntil(
   caches.keys().then(function(cacheNames) {

     return Promise.all(

       cacheNames.map(function(cacheName) {
         if (cacheName !== CACHE_NAME) {

           return caches.delete(cacheName);

         }
       })

     );

   })
 );

});
```

Code in index.html

```
<script>
       // Add event listener to execute code when page loads
       window.addEventListener('load', () => {
           // Call registerSW function when page loads
           registerSW();
       });
```

```
// Register the Service Worker
async function registerSW() {
    // Check if browser supports Service Worker
    if ('serviceWorker' in navigator) {
        try {
            // Register the Service Worker named 'serviceworker.js' await
            navigator.serviceWorker.register('service-worker.js');

        }
        catch (e) {
            // Log error message if registration fails
            console.error('ServiceWorker registration failed: ', e);
        }
    }
}
```

Output:



Conclusion: Hence We Successfully Registered our Service Worker on the Progressive Web App and it is activated as well as running

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.	53
Name	Lisa Sharma
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Name: Lisa Sharma
Division: D15A
Roll No: 53
Batch : C

Experiment No 9

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

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

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

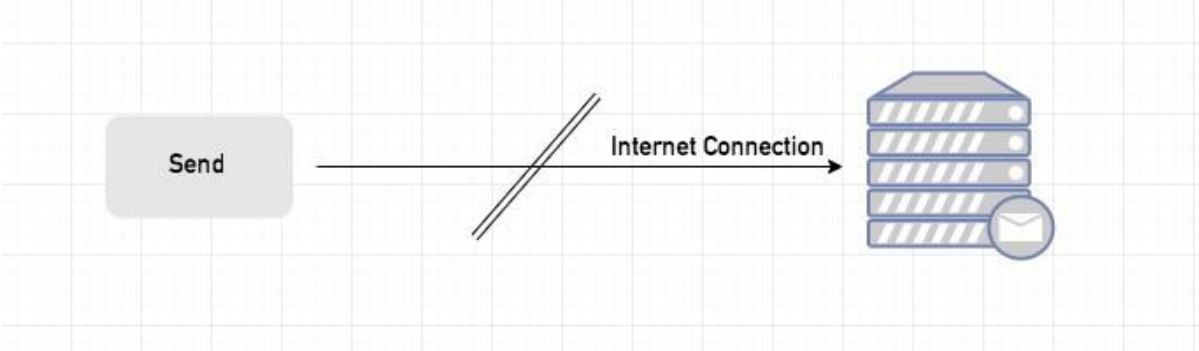
But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

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

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

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

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

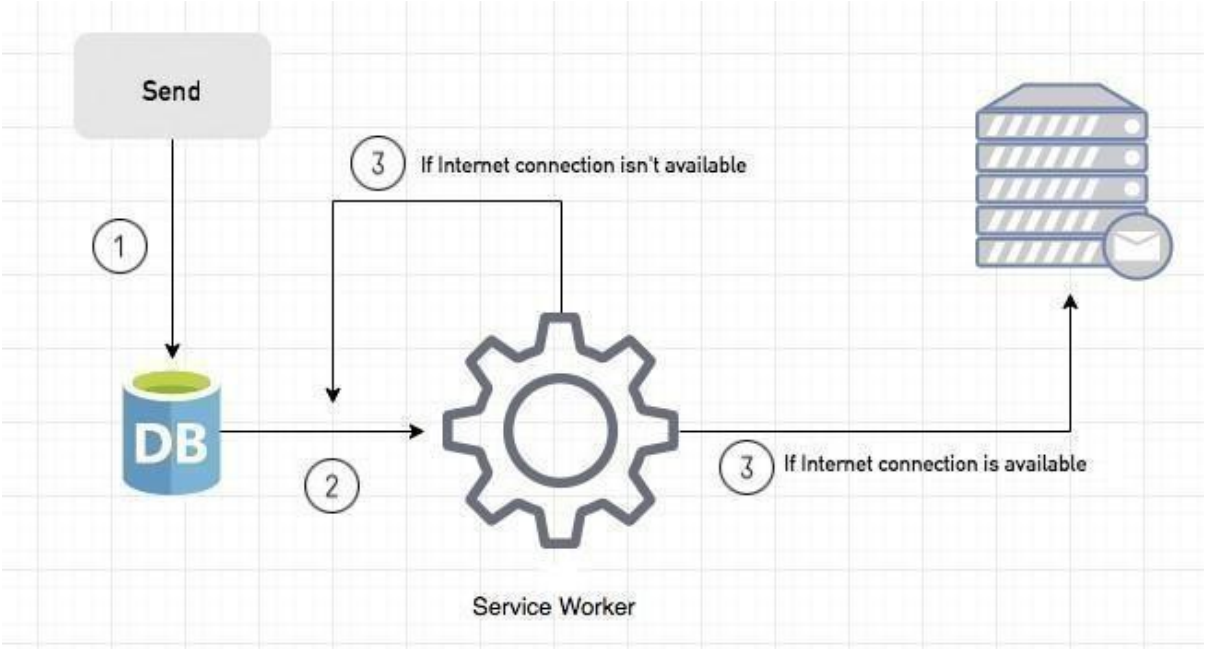
Sync Event

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

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



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



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

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

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.

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

You can use Application Tab from Chrome Developer Tools for testing push notification **Aim**:
To implement Service worker events like fetch, sync and push for E-commerce PWA.

Theory:
Service Worker

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

Things to note about Service Worker:

- A service worker is a programmable network proxy that lets you control how network requests from your page are handled.
- Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.

- The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.
- Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out Promises, an introduction.

Fetch Event

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

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

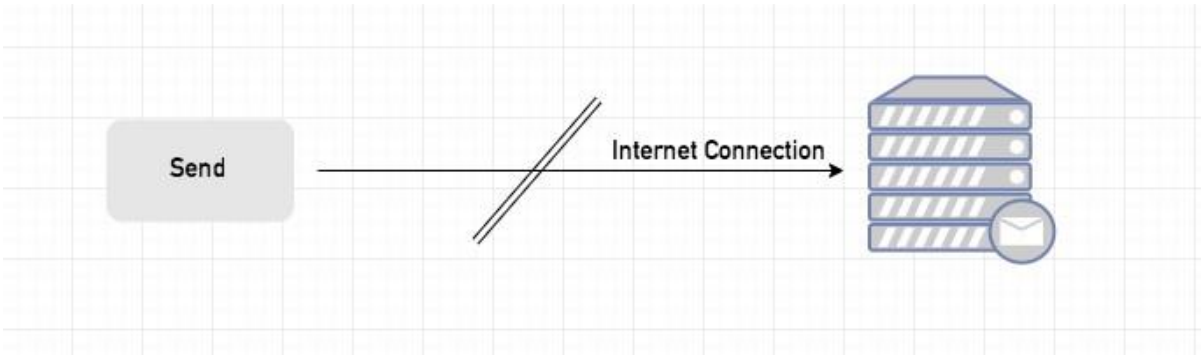
- **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
- **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed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

Sync Event

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

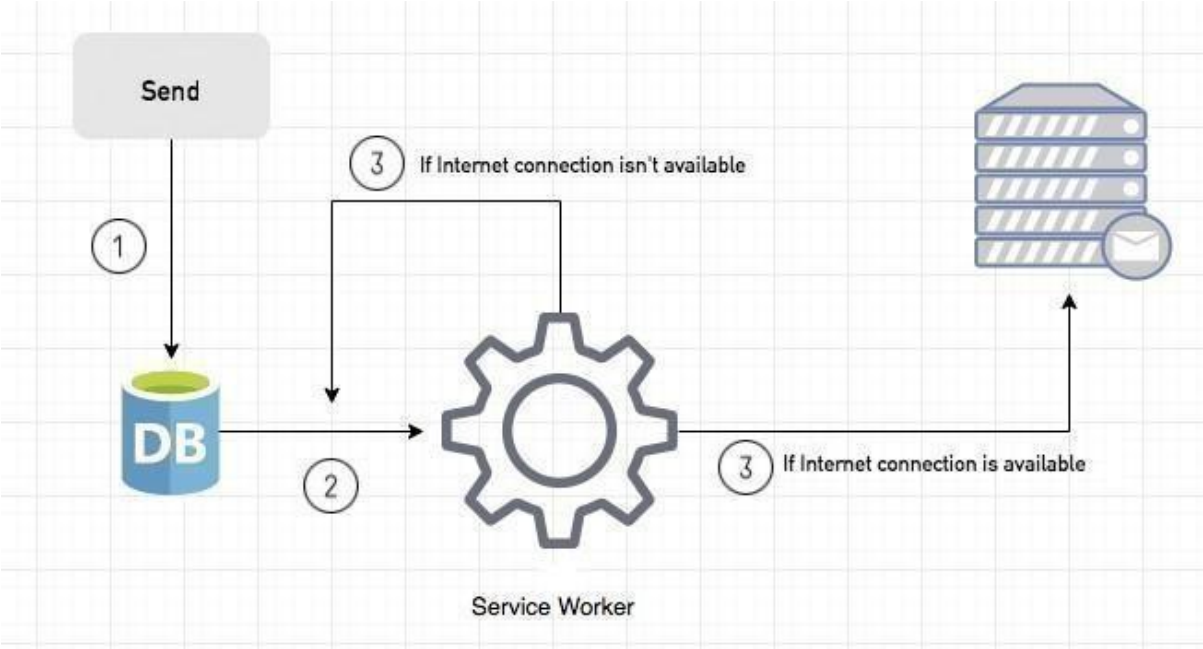
Here is a job for the Background Sync.

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




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

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.

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

You can use Application Tab from Chrome Developer Tools for testing push notification.
Code:

In index.html

```
if ('Notification' in window) {
           Notification.requestPermission().then(function (permission) {
               if (permission === 'granted') {
                   console.log('Notification permission granted.');
               } else {
                   console.warn('Notification permission denied.');
               }
```

```
// service-worker.js



const CACHE_NAME = 'my-ecommerce-app-cache-v1';

const urlsToCache = [

 '/',

 'cart.html',

 'index.html',

 'product.html',

 'shop.html',

 'style.css',

 'success.html',

 'service-worker.js',

 'manifest.json',

 'offline.html'

 // Add more files to cache as needed

];



self.addEventListener('install', function(event) {

 event.waitUntil(

   caches.open(CACHE_NAME)

     .then(function(cache) {

       console.log('Opened cache');

       return cache.addAll(urlsToCache)

         .catch(function(error) {

           console.error('Cache.addAll error:', error);

         });

     })
```

```
            });
        }
```

This code sends notification permission to your Device , and click on allow to send push notification service-worker.js

```
 );
});


self.addEventListener('activate', function(event) {
 // Perform activation steps
 event.waitUntil(
   caches.keys().then(function(cacheNames) {
     return Promise.all(
       cacheNames.map(function(cacheName) {
         if (cacheName !== CACHE_NAME) {
           return caches.delete(cacheName);
         }
       })
     );
   })
 );
});


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


// Sync event listener
self.addEventListener('sync', function(event) {
 if (event.tag === 'syncMessage') {
   console.log("Sync successful!");
 }
});
```

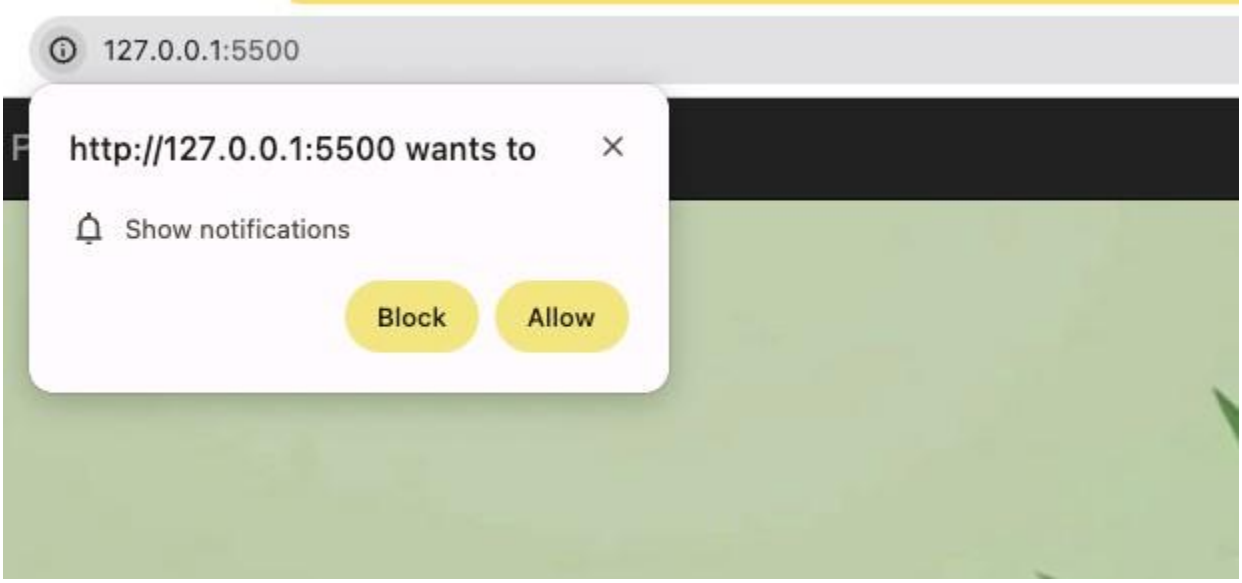
```
// Push event listener

self.addEventListener("push", function (event) {

 if (event && event.data) {

   try {

     var data = event.data.json();

     if (data && data.method === "pushMessage") {
```

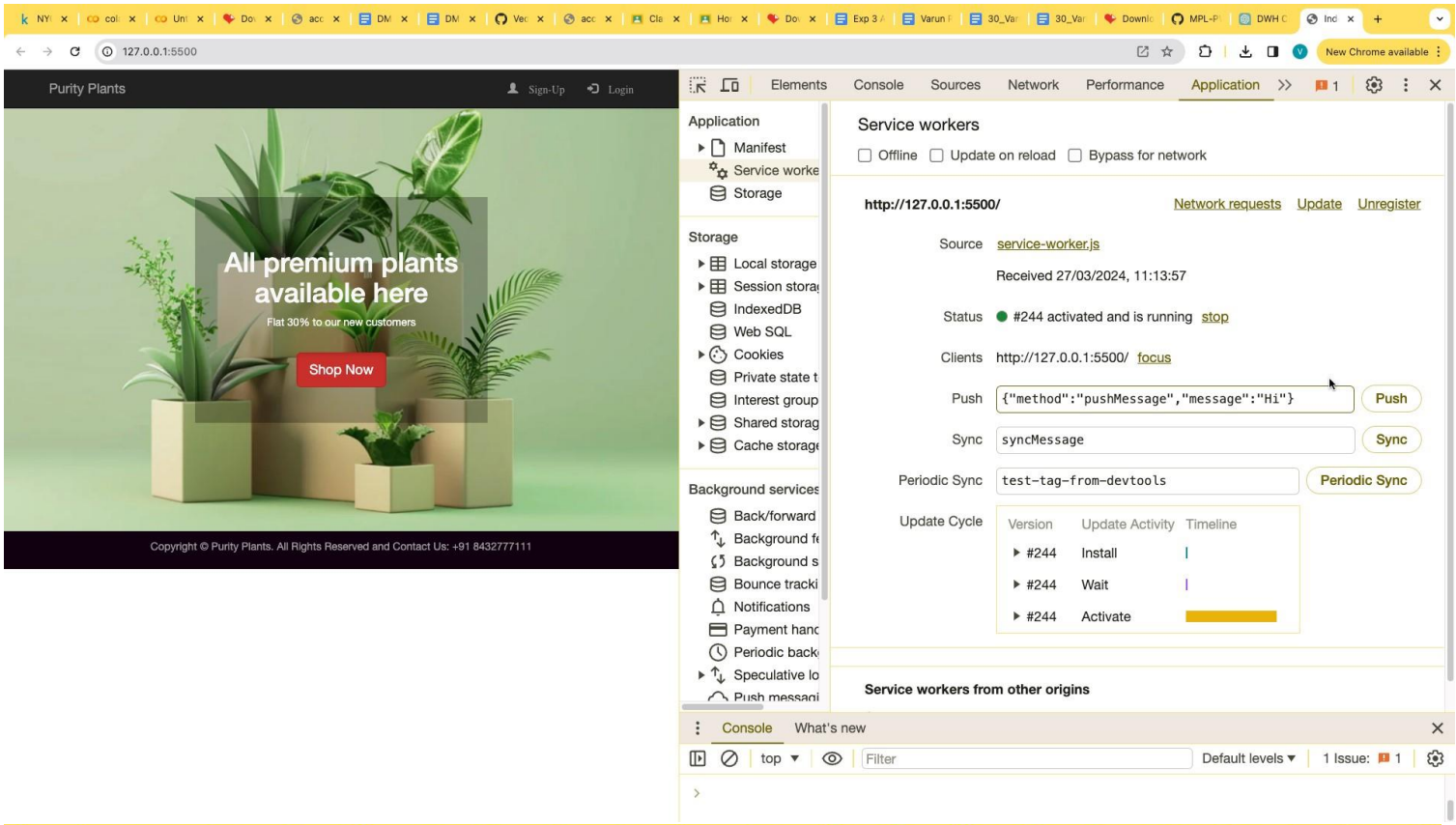
```
      console.log("Push notification sent");
      self.registration.showNotification("Ecommerce website", { body:
data.message });
    }
  } catch (error) {
    console.error("Error parsing push data:", error);
  }
 }
});


self.addEventListener('activate', async () => {
 if (Notification.permission !== 'granted') {
  try {
    const permission = await Notification.requestPermission();
    if (permission === 'granted') {
      console.log('Notification permission granted.');
    } else {
      console.warn('Notification permission denied.');
    }
  } catch (error) {
    console.error('Failed to request notification permission:', error);
  }
 }
});


var checkResponse = function (request) {
 return new Promise(function (fulfill, reject) {
  fetch(request)
    .then(function (response) {
      if (response.status !== 404) {
        fulfill(response);
      } else {
        reject(new Error("Response not found"));
      }
    })
    .catch(function (error) {
      reject(error);
```

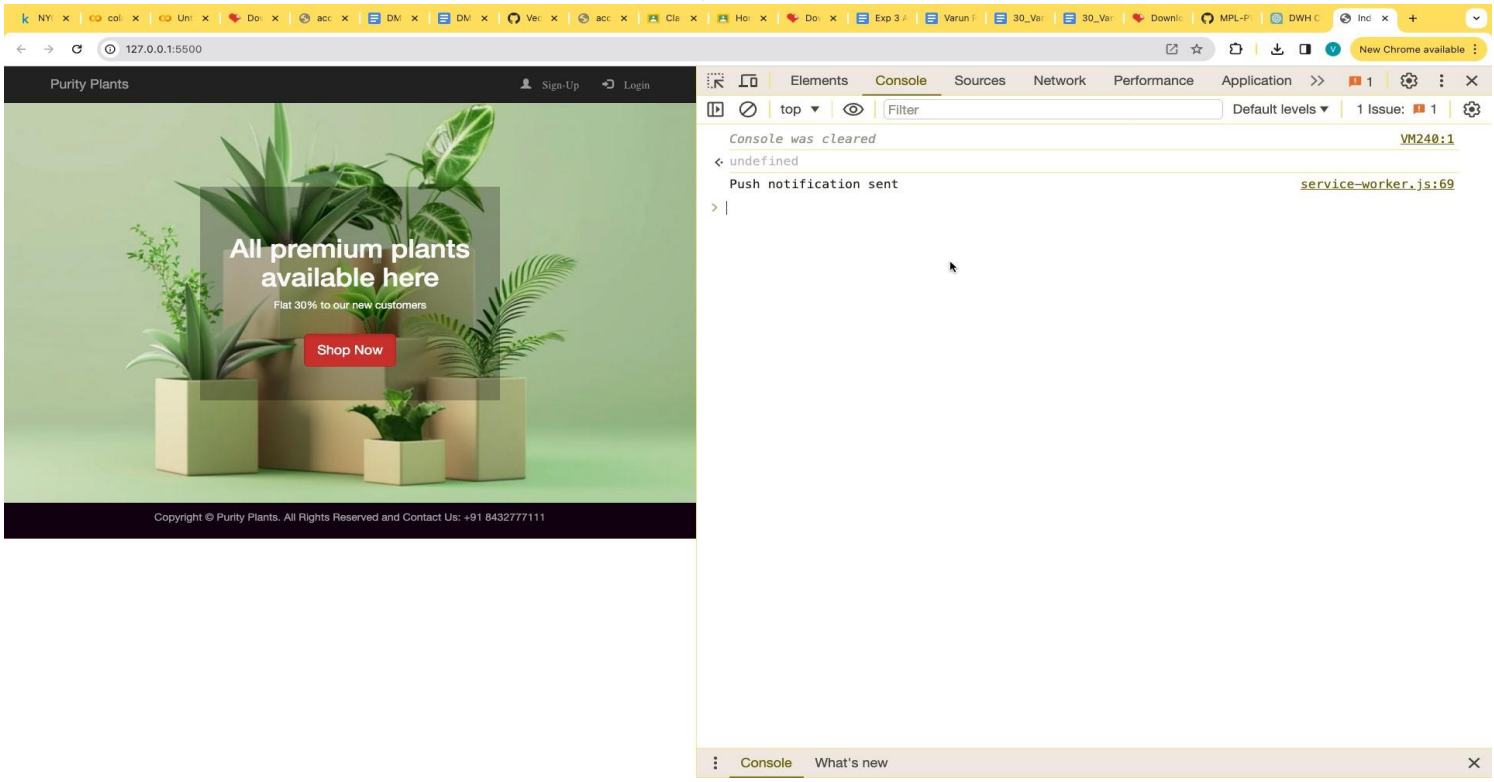
```
      });

  });

};
```

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


var addToCache = function (request) {
 return caches.open("offline").then(function (cache) {
   return fetch(request).then(function (response) {
     return cache.put(request, response.clone()).then(function () {
       return response;
     });
   });
 });
};
```

Output:







Conclusion: Hence we implemented methods like fetch,sync,and push on the service worker , and if we push the message, it says “notification recieved” on the desktop. So the push, sync , and fetch method is implemented successfull.

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	53
Name	Lisa Sharma
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Name:Lisa Sharma

Div: D15A

Roll No: 53

Batch: C

Experiment No: 10

Aim:

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

Theory:

GitHub Pages

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

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

Pros

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

Cons

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

Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

Pros

1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.

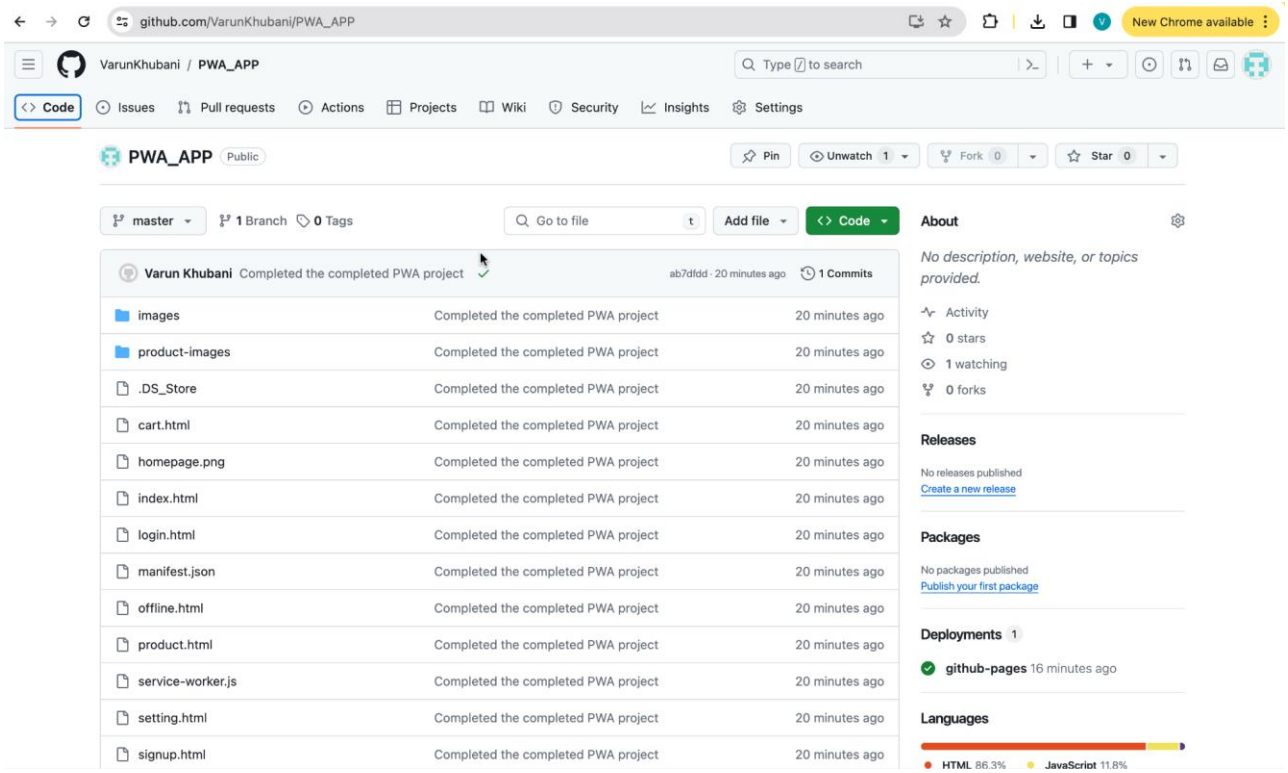
5. Support for HTTPS. A free certificate will be provisioned for your custom domain within 24 hours.

Cons

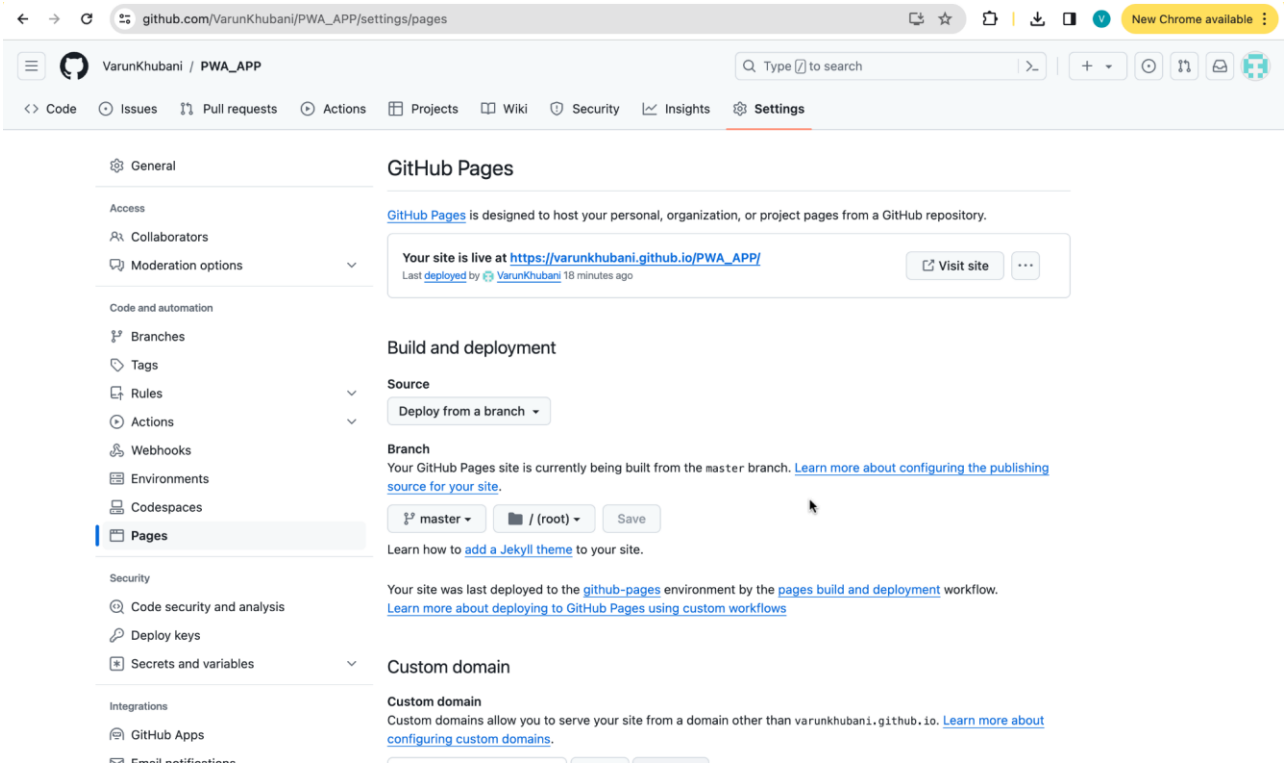
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

Link for GitHub: https://varunkhubani.github.io/PWA_APP/
GitHub ScreenShots

Step1: Make your GitHub Repository public and push your PWA into the repository

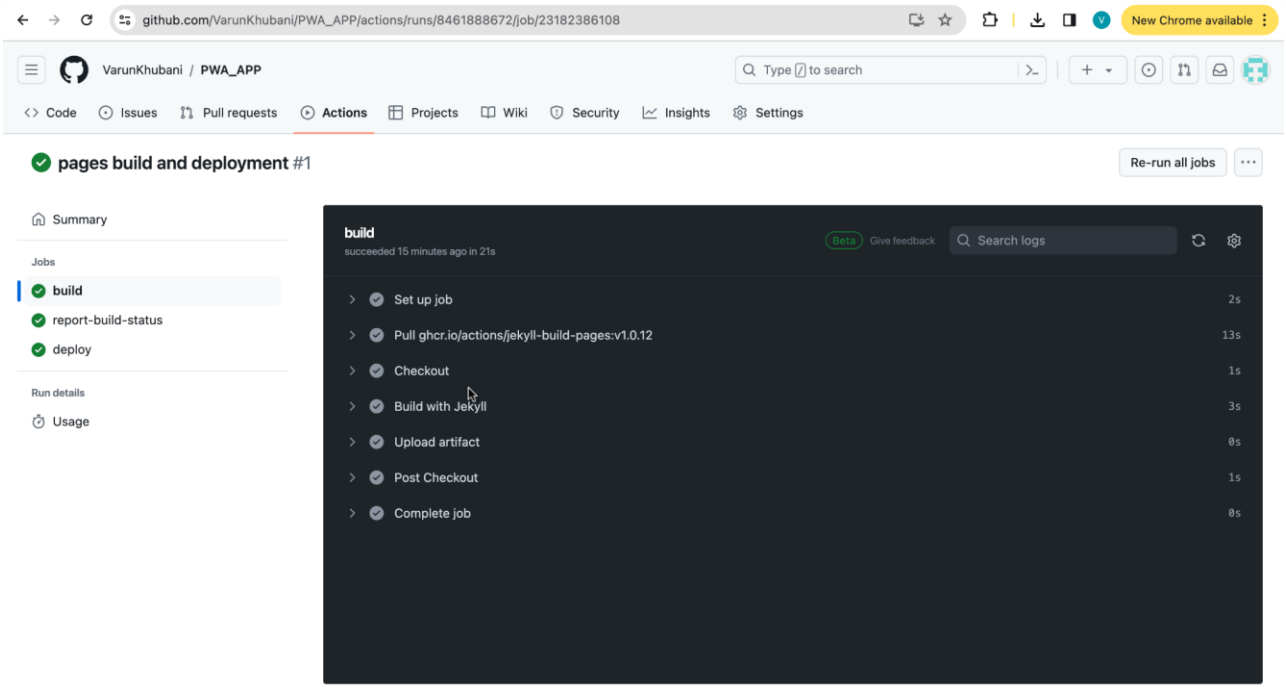


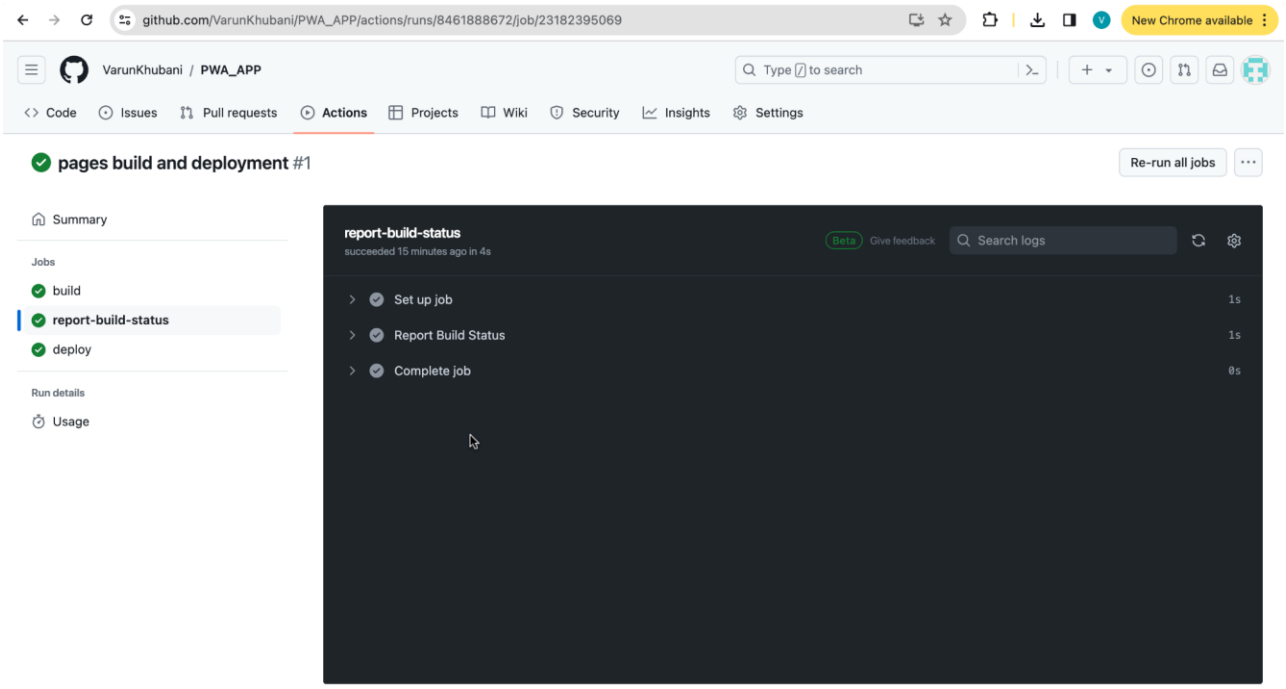
Step 2: Go to settings -> pages and choose your root directory and save it

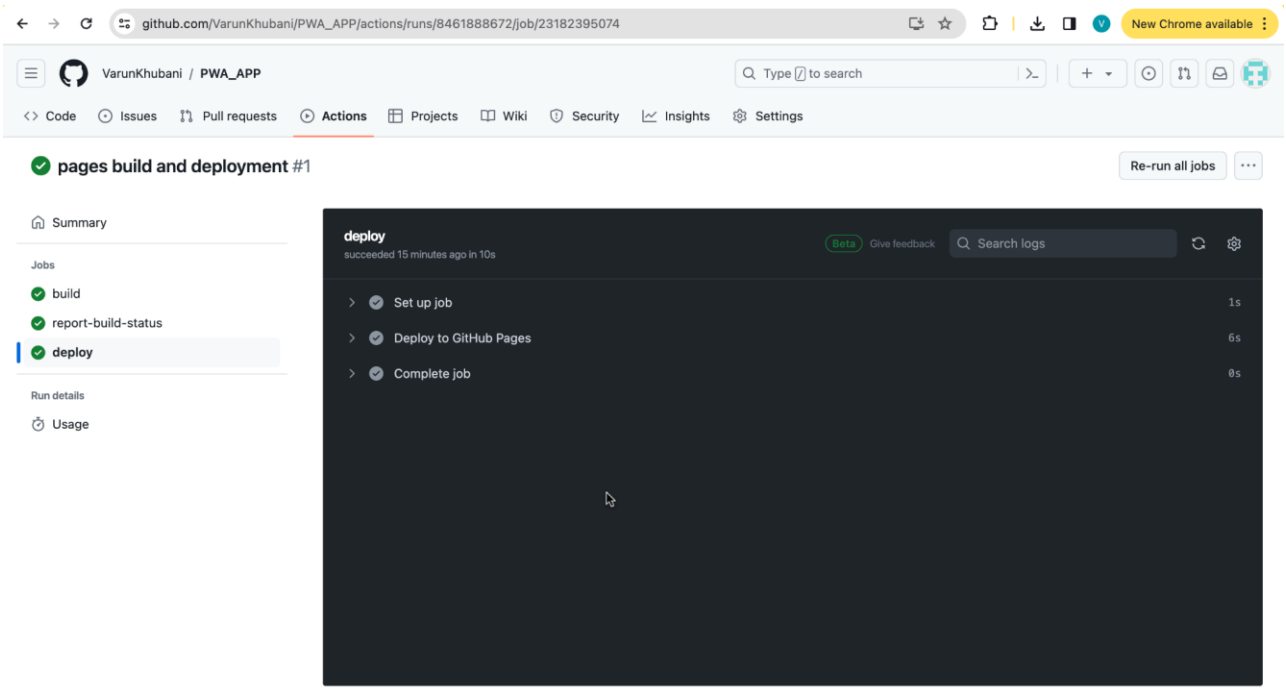


Step 3: Now go to your Code and you will see a small circle near your recent commit(Mine is finished deploying so i am getting a tick-mark sign)

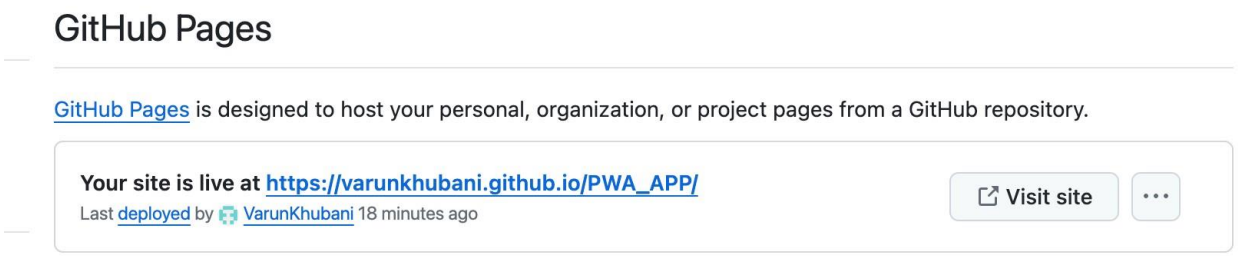
On clicking Logs of all the deployment is shown for convenience

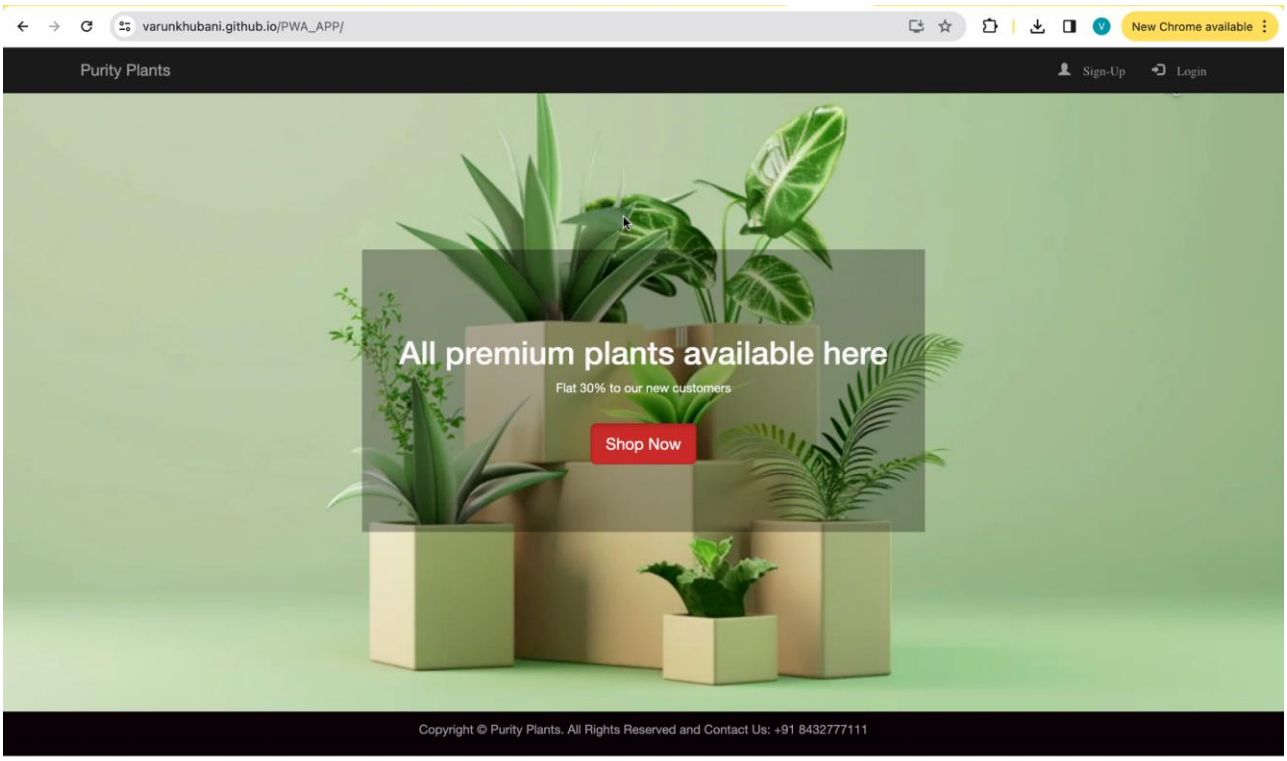

github.com/VarunKhubani/PWA_APP/actions/runs/8461888672/job/23182386108
New Chrome available
VarunKhubani / PWA_APP
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
pages build and deployment #1
Re-run all jobs
Summary
Jobs
build
report-build-status
deploy
Run details
Usage
build
succeeded 15 minutes ago in 21s
Beta
Give feedback
Search logs
Set up job 2s
Pull ghcr.io/actions/jekyll-build-pages:v1.0.12 13s
Checkout 1s
Build with Jekyll 3s
Upload artifact 0s
Post Checkout 1s
Complete job 0s


github.com/VarunKhubani/PWA_APP/actions/runs/8461888672/job/23182395069
New Chrome available
VarunKhubani / PWA_APP
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
pages build and deployment #1
Re-run all jobs
Summary
Jobs
build
report-build-status
deploy
Run details
Usage
report-build-status
succeeded 15 minutes ago in 4s
Beta
Give feedback
Search logs
Set up job 1s
Report Build Status 1s
Complete job 0s



Step4: Go to Settings -> Pages again , and you will see the pages has been deployed and a link is given





Conclusion: Hence we deployed our E-Commerce Progressive Web App Successfully on GitHub Pages

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	53
Name	Lisa Sharma
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

Name: Lisa Sharma
Division: D15A
Roll No: 53
Batch: C

Experiment No 11

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

Theory :

References :
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

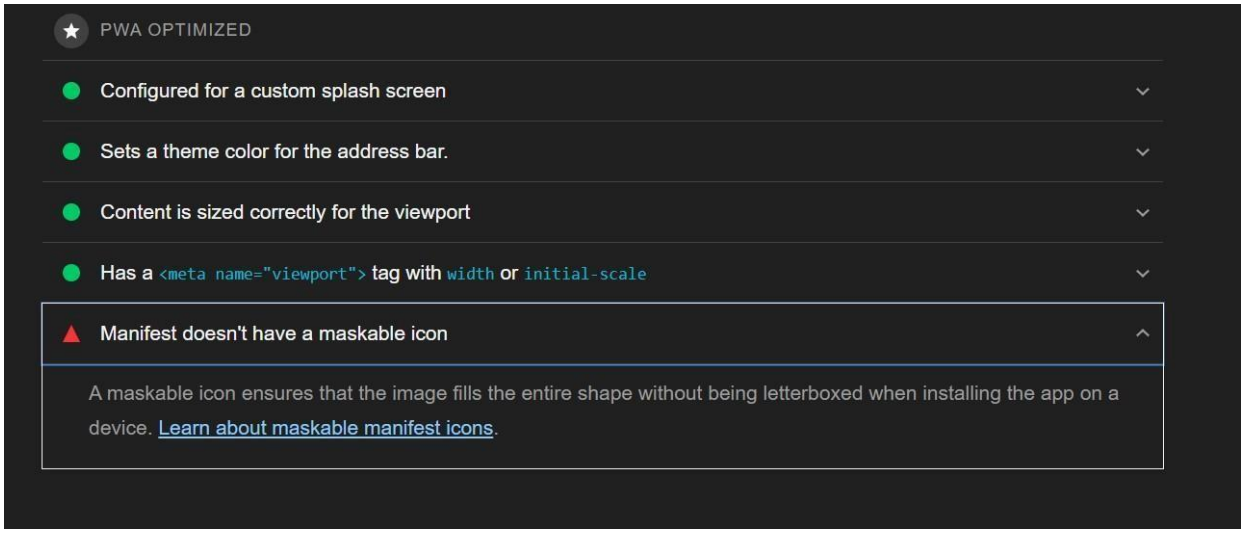
site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th
percentile and so on.

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.

Output:

Before



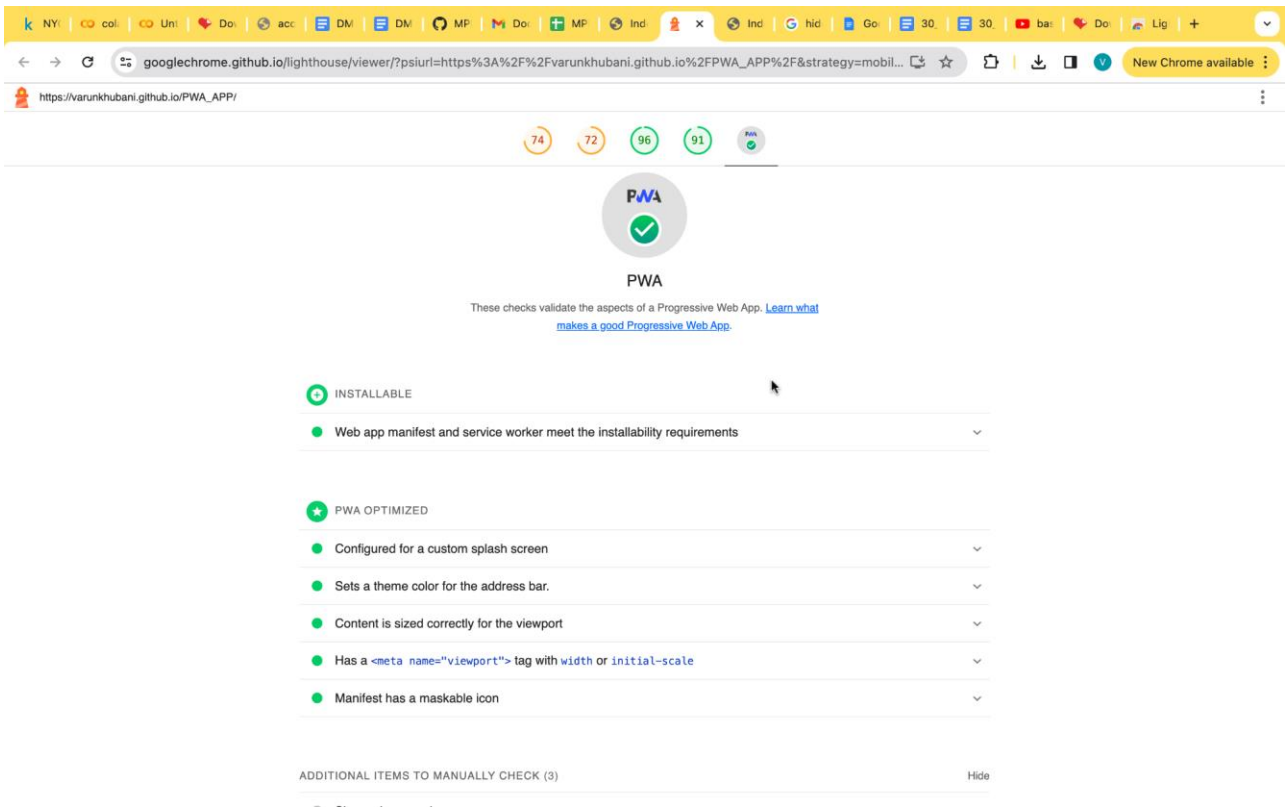
We encountered an issue here , it says “Manifest does not have a maskable icon”

Changes made to the code:

```
{
   "name":"PWA Tutorial",
   "short_name":"PWA",
   "start_url":"index.html",
   "display":"standalone",
   "background_color":"#5900b3",
   "theme_color":"black",
   "scope": ".",
   "description":"This is a PWA tutorial.",
   "icons":[
   {
   "src":"images/Plant 192 X 192.png ",
   "sizes":"192x192",
   "type":"image/png",
   "purpose":"any maskable"
   },
   {
   "src":"images/plant 512 X 512.png",
   "sizes":"512x512",
   "type":"image/png",
   "purpose":"any maskable"
   }
```

```
]
}
```

After:



Conclusion: Hence by making some changes to the code , we did google lighthouse analysis and our PWA is Fully Optimized and ready to go

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.	34
Name	Lisa Sharma
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:	5

LISA SHARMA
53
D15-A

MAD ASSIGNMENT-1

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

→ flutter is an open source UI software development kit created by Google. It is used to developed applications for android, iOS, linux, mac, windows and the web from Single codebase

Key features:-

1. Hot Reload:- This feature allows developers to see the result of their changes almost instantly, without losing the current application state. It enhances productivity and make it easier to experience with the code.
2. Dart programming language:- flutter uses dart, a language optimized for fast apps on any platform. Dart allows for both just in time(JIT) compilation for the development phase.

Advantages:-

1. Reduced Development Time:- The Single codebase and hot reload features significantly reduce development time and efforts as changes can be made in real time and and the app can be deployed in multiple platforms

RMA

2. High performance :- The single codebase, apps compile to have native code, which helps in achieving performance that is comparable to native applications with smooth animations and transitions.

3. Rich Ecosystem :- The flutter ecosystem includes many widgets and tools that cover many usecases. Additionally its easy to use third party plugins to access native features like camera, GPS etc

Differentiation from Traditional Approaches :-

1. Development Efficiency :- Traditional app development often involves writing and maintaining multiple codebases, which is more time consuming and costly

2. productivity tools :- features like hot reload are not typically available in traditional development environments which can slow down development process. flutter's development tools and ecosystem are designed to boost productivity and efficiency.

Popularity Among Developers :-

1. Rapid Development & Iteration :- features like hot reload and a single codebase for multiple platforms makes flutter an attractive option for startup and businesses looking to quickly bring their products to market

2. Strong community and support :- The active and growing developer community among flutter combined with strong support from Google, ensures developers have to access a wealth of resources.

Q2. Widget tree and composition: Describe the concept of widget tree in flutter. Explain how widget composition is used to build complex user interfaces. provide Examples of commonly used widgets and their role in creating a widget tree

⇒ The concept of widget tree in flutter is crucial for understanding how flutter constructs and organises user interfaces (UI) widgets the core building blocks of a flutter app's UI are organised into a hierarchical structure known as the widget tree. The hierarchy represents the arrangement and nesting of widgets, enabling flutter efficiently render the UI.

Widgets in flutter can be either stateful or stateless, depending on whether they depend on some state for their rendering Stateful widgets can update their appearance based on changes on their state. While stateless widgets render once and do not change state.

Flutter emphasis widget composition, where developers build complex UIs by nesting simpler widgets with each other This approach allows for the creation of highly customizable and flexible interfaces. for Example:- A simple UI might consist of a scaffold widget containing a Center widget could be a text widget for a title And Raised Button for an interactive Button.
The structure forms a widget tree starting from Scaffold at the root, down to the text

A SHARMA
ISA
3

Commonly used widgets include:-

Scaffold:- provides the basic material design layout structure.

AppBar:- A toolbar typically used at the top of the top screen.

Text:- displays text with various styling options.

Row/Column:- Organizes children widgets in horizontal or vertical manner.

Container:- A versatile widget for styling, positioning or sizing its child

ListView:- Displays a scrollable list of widgets.

Stack:- layers widgets on top of each other

Through the widget tree and composition, Flutter enables the development of complex, responsive UIs with a single codebase across multiple platforms.

LISA SHAR
M.C.A

Q3- 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.

→ State management in flutter is a critical aspect of web development as it deals with managing the data that changes overtime in your app and how those changes affect the UI.

Importance of state management:-

1. UI Consistency:- Ensures that the UI reflects the current state of app
2. Code Maintainability:- Simplifies the codebase by seperating business logic from UI code, making it easier to maintain and update.
3. Performance optimization:- Efficient state management helps in minimizing the unnecessary widget rebuilds, enhancing app performance.
4. Scalability:- As the app grows, a robust state management can help to manage complexity and flexibility.

Comparison of state management approaches are:-

HARMA

Feature / Aspect	SetState	provider	Riverpod
Description	Built-in feature flutter for managing state	State management library that allows data sharing across widgets	A more flexible and powerful evolution of provider
Complexity	Simple and straightforward for beginners	Moderate complexity requires understanding of provider concepts.	Higher flexi Complexity due to its flexibity and advanced features
Scalability	Not suitable for large apps or extensive state management	Good scalability and maintainability for larger applications	Excellent Scalability suitable for complex and large applications.
performance	Can lead to inefficient in larger apps due to frequent widget rebuilds	optimizes performance by prioritizing unnecessary widgets	High optimized state management, minimizing unnecessary widgets

Scenarios for each approach:-

SetState:- Best for simple or demo apps where you're experimenting with flutter or when the state is confined to a single widget

Provider:- Suitable for most production apps that have a moderate level of complexity. It's a great when the app has a clear seperation between business logic and UI.

Riverpod:- Ideal for complex, large-scale applications with a need for a highly testable, scalable and flexible state management solution.

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 services commonly used in flutter development and provide a brief overview of how data synchronization is achieved.

-> Integrating firebase with flutter:-

1) Set up firebase project:-

- Create a firebase project and register your flutter app in the firebase console

2. Configure flutter app':-
Add firebase dependencies in 'pubspec.yaml' and initialize firebase in the main dart using 'firebase.initialize APP()

3. Using firebase services:-
Include firebase services in your flutter app like Authentication ('~~Auth.app~~' ('firebase_auth'), firestore ('cloud_firestore') and Realtime database ('firbase_database')

4. Authentication Example
Sign in with Email and password using 'firbaseAuth.Instance.signInWithEmailAndPassword'

5) Firestore Example:-
Read data from firestore using 'firbase firestore.instance.collection('users').doc('user_id').get();

Benefits Of using firbase as a Backend solution:-

1) Real-time data sync:-
firbase provides real-time data synchronisation, allowing updates to be immediately reflected in all connected clients.

2 Scalability :-

Firebase automatically scales to handle the growth of your app, ensuring it remains performant as user number increases.

Firebase achieves real-time data synchronisation through websockets. When data changes on the server, firebase sends updates to the connected clients over websockets ensuring that the clients local data sync with the server.

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.	53
Name	Lisa Sharma
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:	4

LISA SHARMA
BISA
53

PWA Assignment-2

Q1. Define progressive web App (PWA) and explain its significance in modern web development. Discuss the key characteristics that differentiate PWA from traditional mobile App

(8) (4)

⇒ A progressive web App (PWA) is a type of web Application that uses modern web capabilities to deliver an app-like experience to users PWAs are built using standard web technologies such as HTML, CSS, and JS but are designed to provide a more native-like experience for users especially on mobile devices.

Key characteristics of progressive web Apps are -

1) Progressive Enhancement - They are built using progressive enhancement principle meaning they provide a basic experience for all users and then enhance based on the capabilities of the user's device and browser

2) Responsive design:- PWAs are designed to feel and behave like native mobile apps. They include multiple features such as Animations, Gestures and transitions to provide a more immersive user experience

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

4) Fast performance

5) Discoverability:- PWAs are easily discoverable and shareable as they are built using standard web technologies and can be accessed via a URL.

6) Security - PWAs are served over HTTPS to ensure security
→ 7) Cross platform compatibility.

The significance of PWAs in modern web development lies in their ability to bridge the gap between web and native app experiences by combining the reach and accessibility of the web with functionality and engagement of native app. They allow developers to create fast reliable and engaging experiences that work seamlessly across wide range.

Q2. Design responsive web design and explain its performance

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

Importance for PWAs -

- Foundation for app like Experience:- PWA ensures proper layout and navigation across devices
- Accessibility:- A responsive design make PWA accessible to wider audience using diverse devices
- SEO benefits:- Responsive design is a primitive ranking factor in search engine design.
- Responsive web design v/s fluid v/s adaptive.

feature	Responsive	fixed	Adaptive
Layout	1) flexibility, adapts to any screen size	continuously adjust based on width	uses multiple fixed width layouts.
Development effort	moderate	less effort for basic layout	more effort to create multiple layout.

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

-> Service workers are crucial components of progressive web apps that enables features such as offline support, push notifications and background synchronization. The lifecycle of service worker involves several key features registration, installation and activation.

1) registration:- It is done using the 'navigator.serviceWorker.register()' method which takes the path to service workers file as a parameter

2) Installation:- Once service worker registered, browser downloads and starts installation process of service worker script

- During installation the service worker script is executed and then install event is fired. Inside install event handler developers can define what resource should be cached for offline access using techniques like caching strategies.

Installation process is critical for setting up the service worker's initial cache and preparing it to control of web applications.

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

- Developers can use the activate event handler to manage cache cleanup, versioning and other activation tasks
- Service remains active until its explicitly unregistered or updated

Its important to note that Service workers run asynchronously

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

Indexed DB lets you store and retrive objects that are objects that are indexed with a key, any objects that are supported by structured clone algorithm can be stored