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

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Name of the Course : MAD & PWA Lab

Course Code : ITL604

Project Title: Amazon Clone / PWA: Ecommerce Website **Roll No. 42**

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

Faculty Incharge : Mrs. Kajal Joseph.

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

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

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: Amazon Clone / PWA: Ecommerce Website **Roll No. 42**

Lab Objectives:

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

Lab Outcomes:

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

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1	17/01/24	24/01/24	15M
2.	To design Flutter UI by including common widgets.	LO2	24/01/24	31/01/24	15M
3.	To include icons, images, fonts in Flutter app	LO2	31/01/24	07/02/24	15M
4.	To create an interactive Form using form widget	LO2	07/02/24	14/02/24	15M
5.	To apply navigation, routing and gestures in Flutter App	LO2	14/02/24	21/02/24	15M
6.	To Connect Flutter UI with fireBase database	LO3	21/02/24	6/03/24	15M
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4	06/03/24	29/03/24	15M
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	13/03/24	29/03/24	15M
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	20/03/24	29/03/24	15M
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	27/03/24	29/03/24	15M
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	27/03/24	29/03/24	15M
12.	Assignment-1	LO1,LO2,LO3	28/01/24	05/03/24	5M
13.	Assignment-2	LO4,LO5,LO6	14/03/24	21/03/24	4M

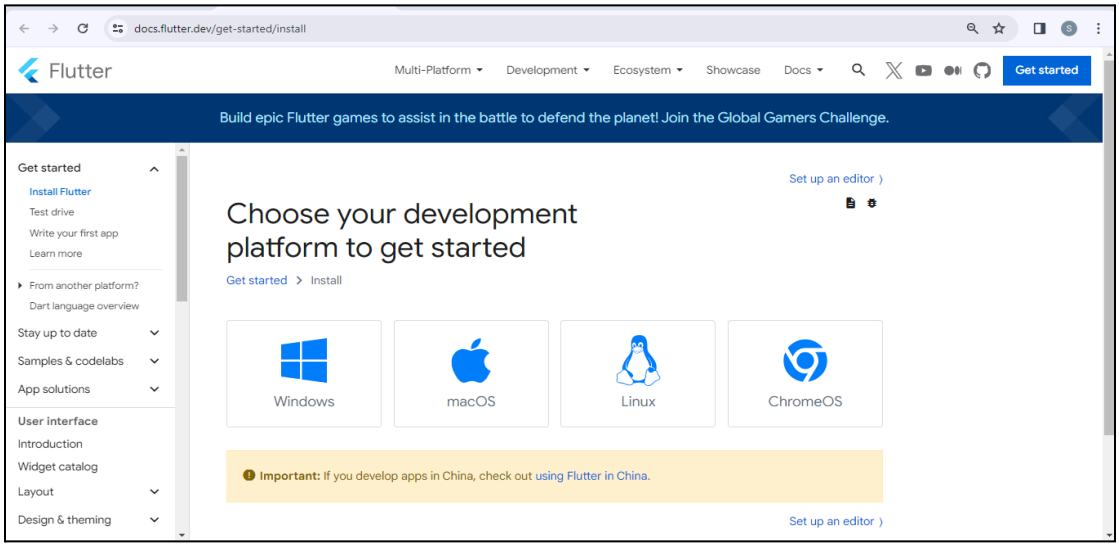
MAD & PWA Lab
Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	15M

Experiment 1

Aim : To install and configure flutter Environment

Install flutter SDK



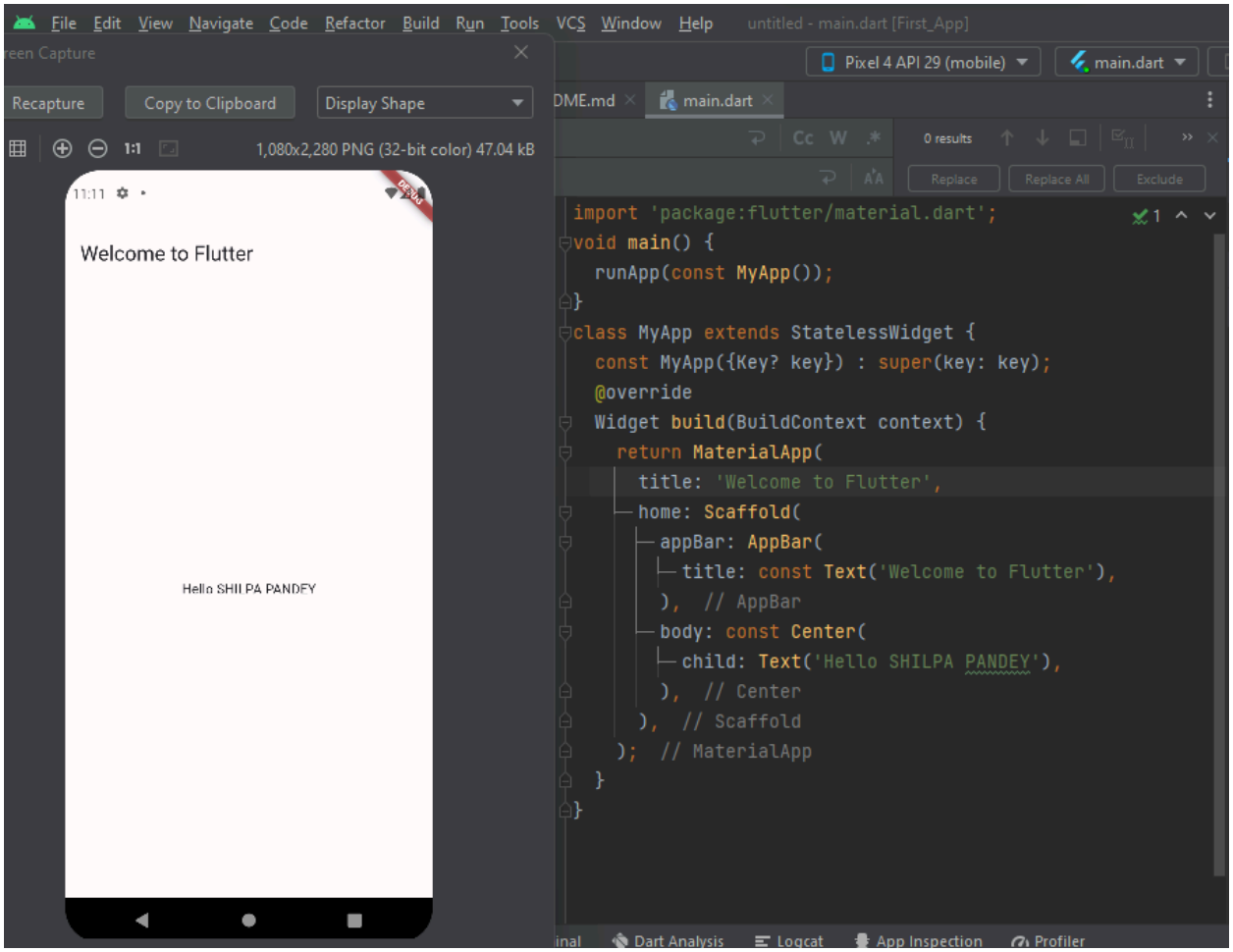
Install Android SDK

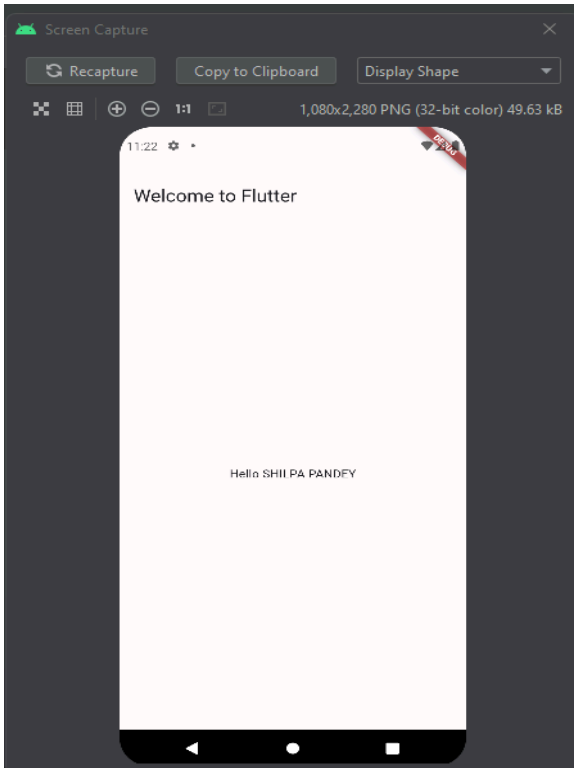
Code:

Flutter App

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

Project Title: Amazon Clone / PWA: Ecommerce Website **Roll No. 42**





Conclusion : In the above experiment , we have successfully configured and installed flutter and android studio in our desktopUnderstand how to run the simple flutter project on a virtual device

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15M

Experiment 2

Aim : To design Flutter UI by including common widgets

Theory:

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.

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**.

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:

- **Image:** It is a generic image loader, which is used by **ImageProvider**.
- **asset:** It load image from your project asset folder.
- **file:** It loads images from the system folder.
- **memory:** It load image from memory.
- **network:** It loads images from the network.

Scaffold

This widget provides a framework that allows you to add common material design elements like AppBar, Floating Action Buttons, Drawers, etc.

Stack

It is an essential widget, which is mainly used for **overlapping** a widget, such as a button on a background gradient.

State Management Widget

In Flutter, there are mainly two types of widget:

- StatelessWidget
- StatefulWidget

StatefulWidget

A StatefulWidget has state information. It contains mainly two classes: the **state object** and the **widget**. It is dynamic because it can change the inner data during the widget lifetime. This widget does not have a **build()** method. It has **createState()** method, which returns a class that extends the Flutters State Class. The examples of the StatefulWidget are Checkbox, Radio, Slider, InkWell, Form, and TextField.

StatelessWidget

The StatelessWidget does not have any state information. It remains static throughout its lifecycle. The examples of the StatelessWidget are Text, Row, Column, Container, etc.

Code :

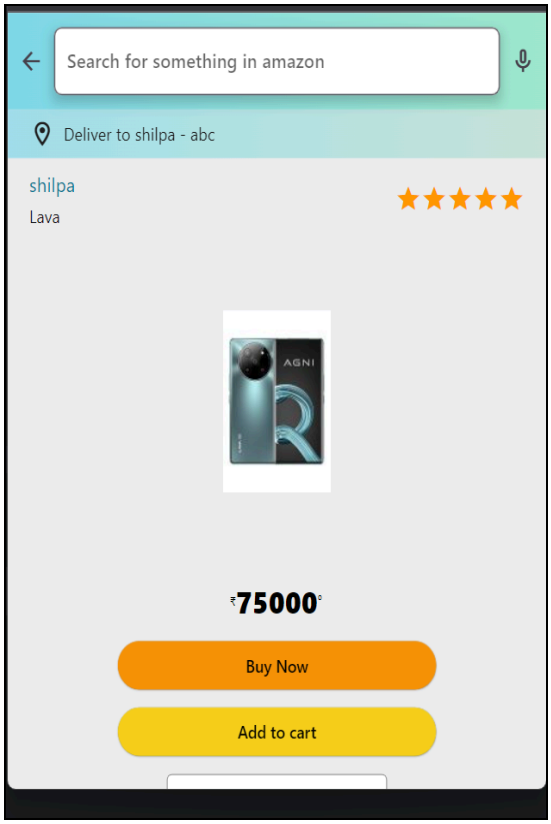
cart

```
import 'dart:ui';
import 'package:amazon_clone/model/product_model.dart';
import 'package:amazon_clone/model/user_details_model.dart';
import 'package:amazon_clone/providers/user_details_provider.dart';
import 'package:amazon_clone/resources/cloudfirestore_methods.dart';
```

```
import 'package:amazon_clone/utils/color_themes.dart';
import 'package:amazon_clone/utils/constants.dart';
import 'package:amazon_clone/utils/utils.dart';
import 'package:amazon_clone/widgets/cart_item_widget.dart';
import 'package:amazon_clone/widgets/custom_main_button.dart';
import 'package:amazon_clone/widgets/search_bar_widget.dart';
import 'package:amazon_clone/widgets/user_details_bar.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';


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


 @override
 State<CartScreen> createState() => _CartScreenState();
}


class _CartScreenState extends State<CartScreen> {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: SearchBarWidget(
       hasBackButton: false,
       isReadOnly: true,
     ),
     body: Center(
       child: Stack(
         children: [
           Column(
             children: [
               const SizedBox(
                 height: kAppBarHeight / 2,
               ),
               Padding(
                   padding: const EdgeInsets.all(8.0),
                   child: StreamBuilder(
                     stream: FirebaseFirestore.instance
                         .collection("users")
                         .doc(FirebaseAuth.instance.currentUser!.uid)
```

```
            .collection("cart")
            .snapshots(),
        builder: (context,
            AsyncSnapshot<QuerySnapshot<Map<String, dynamic>>>
                snapshot) {
          if (snapshot.connectionState ==
              ConnectionState.waiting) {
            return CustomMainButton(
                child: const Text(
                  "Loading",
                ),
                color: yellowColor,
                isLoading: true,
                onPressed: () {});
          } else {
            return CustomMainButton(
                child: Text(
                  "Proceed to buy (${snapshot.data!.docs.length}) items",
                  style: const TextStyle(color: Colors.black),
                ),
                color: yellowColor,
                isLoading: false,
                onPressed: () async {
                  await CloudFirestoreClass().buyAllItemsInCart(
                      userDetails:
                          Provider.of<UserDetailsProvider>(
                                context,
                                listen: false)
                            .userDetails);
                  Utils().showSnackBar(
                      context: context, content: "Done");
                });
          }
        },
      )),
    Expanded(
      child: StreamBuilder(
    stream: FirebaseFirestore.instance
        .collection("users")
        .doc(FirebaseAuth.instance.currentUser!.uid)
        .collection("cart")
        .snapshots(),
    builder: (context,
        AsyncSnapshot<QuerySnapshot<Map<String, dynamic>>>
```

```
                    snapshot) {
                if (snapshot.connectionState == ConnectionState.waiting) {
                  return Container();
                } else {
                  return ListView.builder(
                      itemCount: snapshot.data!.docs.length,
                      itemBuilder: (context, index) {
                        ProductModel model = ProductModel.getModelFromJson(
                            json: snapshot.data!.docs[index].data());
                        return CartItemWidget(product: model);
                      });
                }
              },
            ))
          ],
        ),
        UserDetailsBar(
          offset: 0,
        ),
      ],
    ),
  ),
 );
}
}
```



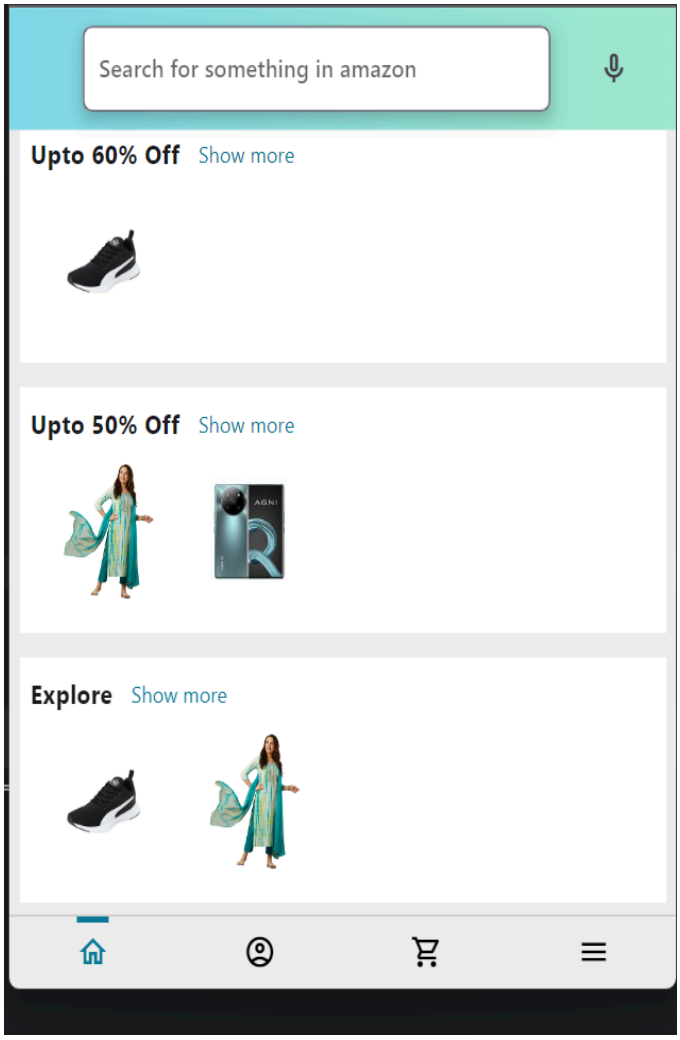
Home

```
import 'package:amazon_clone/model/user_details_model.dart';
import 'package:amazon_clone/resources/cloudfirestore_methods.dart';
import 'package:amazon_clone/utils/constants.dart';
import 'package:amazon_clone/widgets/ad_banner_widget.dart';
import 'package:amazon_clone/widgets/categories_horizontal_list_view_bar.dart';
import 'package:amazon_clone/widgets/loading_widget.dart';
import 'package:amazon_clone/widgets/products_showcase_list_view.dart';
import 'package:amazon_clone/widgets/search_bar_widget.dart';
import 'package:amazon_clone/widgets/simple_product_widget.dart';
import 'package:amazon_clone/widgets/user_details_bar.dart';
import 'package:flutter/material.dart';


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


 @override
```

```
 State<HomeScreen> createState() => _HomeScreenState();
}


class _HomeScreenState extends State<HomeScreen> {
 ScrollController controller = ScrollController();
 double offset = 0;
 List<Widget>? discount70;
 List<Widget>? discount60;
 List<Widget>? discount50;
 List<Widget>? discount0;


 @override
 void initState() {
  super.initState();
  getData();
  controller.addListener(() {
    setState(() {
      offset = controller.position.pixels;
    });
  });
 }


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


 void getData() async {
  List<Widget> temp70 =
  await CloudFirestoreClass().getProductsFromDiscount(70);
  List<Widget> temp60 =
  await CloudFirestoreClass().getProductsFromDiscount(60);
  List<Widget> temp50 =
  await CloudFirestoreClass().getProductsFromDiscount(50);
  List<Widget> temp0 = await CloudFirestoreClass().getProductsFromDiscount(0);
  print("everything is done");
  setState(() {
    discount70 = temp70;
    discount60 = temp60;
```

```
    discount50 = temp50;
    discount0 = temp0;
  });
}


@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: SearchBarWidget(
      isReadOnly: true,
      hasBackButton: false,
    ),
    body: discount70 != null &&
        discount60 != null &&
        discount50 != null &&
        discount0 != null
        ? Stack(
      children: [
        SingleChildScrollView(
          controller: controller,
          child: Column(
            children: [
              SizedBox(
                height: kAppBarHeight / 2,
              ),
              CategoriesHorizontalListViewBar(),
              AdBannerWidget(),
              ProductsShowcaseListView(
                  title: "Upto 70% Off", children: discount70!),
              ProductsShowcaseListView(
                  title: "Upto 60% Off", children: discount60!),
              ProductsShowcaseListView(
                  title: "Upto 50% Off", children: discount50!),
              ProductsShowcaseListView(
                  title: "Explore", children: discount0!),
            ],
          ),
        ),
        UserDetailsBar(
          offset: offset,
        ),
      ],
    )
```

```
        : const LoadingWidget(),
    );
  }
}
```



Conclusion:

Understand the various common widgets like Text , button etc in flutter and also how to use them in flutter App.

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15M

Experiment 3

AIM: To include icons, images, and fonts in a Flutter app.

THEORY:

1. Icons:
Icons play a crucial role in conveying information and improving the user interface. Flutter provides the Icon widget to display various icons from different icon libraries. Commonly used icon libraries include MaterialIcons, FontAwesome, and CupertinoIcons.

To include icons:

Import the necessary icon library in your Dart file:
dart
Copy code
import 'package:flutter/material.dart'; // Example for Material Icons
Use the Icon widget to display an icon:
dart
Copy code
Icon(Icons.favorite);

2. Images:
Images are essential for creating visually appealing interfaces. Flutter supports various image formats, such as JPEG, PNG, and GIF.

To include images:

Place your image assets in the assets directory of your Flutter project.
Update your pubspec.yaml file to include the assets:
yaml
Copy code
flutter:

```
  assets:
    - assets/my_image.jpg
```

Use the Image widget to display an image:

dart

Copy code

```
Image.asset('assets/my_image.jpg');
```

3. Fonts:

Custom fonts can give your app a unique and distinctive look.

To include custom fonts:

Place your font files (e.g., TTF or OTF) in the fonts directory of your Flutter project.

Update your pubspec.yaml file to include the fonts:

yaml

Copy code

```
flutter:
  fonts:
    - family: MyCustomFont
      fonts:
        - asset: fonts/my_custom_font.ttf
```

Use the custom font in your text widget:

dart

Copy code

```
Text(
  'Hello, Flutter!',
  style: TextStyle(
    fontFamily: 'MyCustomFont',
  ),
)
```

Code

Main.dart

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

void main() {
  runApp(const AmazonClone());
}

class AmazonClone extends StatelessWidget {
  const AmazonClone({Key? key}) : super(key: key);


  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: "Amazon Clone",
      debugShowCheckedModeBanner: false,
      theme: ThemeData.light().copyWith(
        scaffoldBackgroundColor: backgroundColor,
      ),
      home:  const HomeScreen());
  }
}
```

home.dart

```
import 'package:amazon_clone/utils/constants.dart';
import 'package:amazon_clone/widgets/ad_banner_widget.dart';
import ‘package:amazon_clone/widgets/categories_horizontal_list_view_bar.dart';
import 'package:flutter/material.dart';


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


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

```
}


class _HomeScreenState extends State<HomeScreen> {
 ScrollController controller = ScrollController();
 @override
 void dispose() {
  super.dispose();
  controller.dispose();
 }
 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: SearchBarWidget(
      isReadOnly: true,
      hasBackButton: false,
    ),
     body: SingleChildScrollView(
         child: Column(
           children: [
               AdBannerWidget(),

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

AdBanner.dart

```
import 'package:amazon_clone/utils/color_themes.dart';
import 'package:amazon_clone/utils/constants.dart';
import 'package:amazon_clone/utils/utils.dart';
import 'package:flutter/material.dart';


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

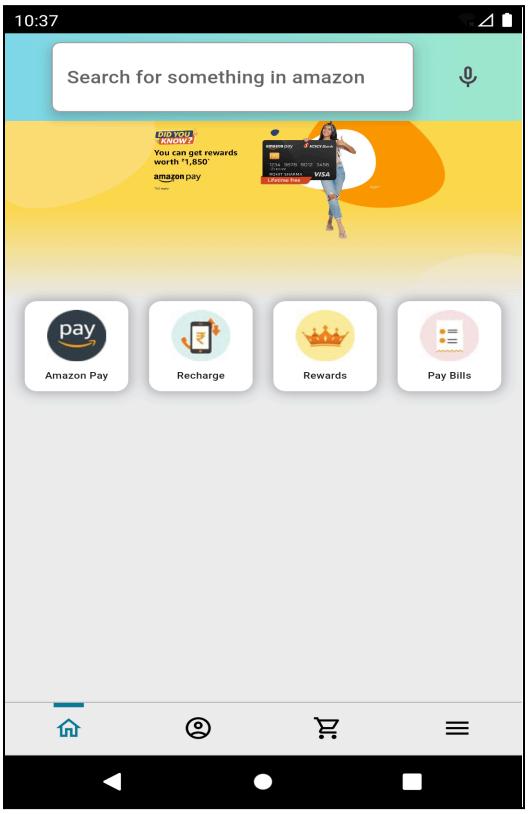
```
 @override
 State<AdBannerWidget> createState() => _AdBannerWidgetState();
}


class _AdBannerWidgetState extends State<AdBannerWidget> {
 int currentAd = 0;
 @override
 Widget build(BuildContext context) {
  Size screenSize = Utils().getScreenSize();
  double smallAdDimension = screenSize.width / 5;
  //Image and gradient
  return Column(
   children: [
    Stack(
     children: [
      GestureDetector(
       onHorizontalDragEnd: (_) {
        if (currentAd == largeAds.length - 1) {
         setState(() {
          currentAd = 0;
         });
        } else {
         setState(() {
          currentAd++;
         });
        }
       },
       child: SizedBox(
        width: double.infinity,
        child: Image.network(
         largeAds[currentAd],
        ),
       ),
      ),
      Positioned(
       bottom: 0,
       child: Container(
        width: screenSize.width,
```

```
        height: screenSize.height / 8,
        decoration: BoxDecoration(
          gradient: LinearGradient(
            colors: [
              backgroundColor,
              backgroundColor.withOpacity(0.000001)
            ],
            begin: Alignment.bottomCenter,
            end: Alignment.topCenter,
          ),
        ),
      ),
    ),
  ],
),
Container(
  height: smallAdDimension,
  width: screenSize.width,
  color: backgroundColor,
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      getSmallAdWidget(
        index: 0,
        side: smallAdDimension,
      ),
      getSmallAdWidget(
        index: 1,
        side: smallAdDimension,
      ),
      getSmallAdWidget(
        index: 2,
        side: smallAdDimension,
      ),
      getSmallAdWidget(
        index: 3,
        side: smallAdDimension,
      ),
    ],
  ),
```

```
      )
    ],
  );
}


Widget getSmallAdWidget({required int index, required double side}) {
 return Container(
    height: side,
    width: side,
    decoration: ShapeDecoration(
      color: Colors.white,
      shadows: [
        BoxShadow(
            color: Colors.black.withOpacity(0.3),
            spreadRadius: 1,
            blurRadius: 8),
      ],
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(10),
      ),
    ),
    child: Padding(
      padding: const EdgeInsets.all(8.0),
      child: FittedBox(
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Image.network(
            smallAds[index],
            ),
            Padding(
              padding: const EdgeInsets.only(top: 10),
              child: Text(
                adItemNames[index],
                style: const TextStyle(
                    fontSize: 13, fontWeight: FontWeight.w500),
              ),
            )
          ],
```

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

Output:



CONCLUSION:

Understood how to include the icons,images and fonts in flutter UI.

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15M

Experiment 4

AIM: To create an interactive Form using form widget

THEORY:

Form Widget: The Form widget serves as the container for form elements. It requires a key parameter, which is essential for accessing the form's state and performing actions like validation and submission.

Form Fields: Inside the Form widget, various form fields can be added. Flutter provides widgets like TextFormField, Checkbox, Radio, DropdownButton, etc., for collecting different types of user input.

State Management: To manage the state of the form, typically a StatefulWidget is used. The state class associated with the widget holds the form's state, including user input and validation status.

Validation: Form validation ensures that user input meets specific criteria before submission. Each form field can have a validator function, which returns an error message if the input is invalid. Flutter's Form widget automatically triggers validation when the form is submitted.

Submission Handling: When the user submits the form, a callback function is invoked to handle the form data. This function can access the form's state using the form key and perform tasks like data processing, database operations, or navigation.

Displaying Errors: Validation errors are displayed to users to guide them in correcting their input. Flutter allows developers to customize error messages and display them below the corresponding form fields.

User Interaction: Interactive elements like buttons are used to trigger form submission. By attaching event handlers to these elements, developers can control the flow of the form and provide feedback to users.

Form Reset: After successful submission or upon user request, the form can be reset to its initial state. This ensures a clean user experience for subsequent interactions.

Code

Main.dart

```
import 'package:flutter/material.dart';
import 'screens/sign_in_screen.dart';
import 'utils/color_themes.dart';


void main() {
 runApp(const AmazonClone());
}


class AmazonClone extends StatelessWidget {
 const AmazonClone({Key? key}) : super(key: key);


 @override
 Widget build(BuildContext context) {
  return MaterialApp(
    title: "Amazon Clone",
    debugShowCheckedModeBanner: false,
    theme: ThemeData.light().copyWith(
      scaffoldBackgroundColor: backgroundColor,
    ),
    home:  const SignInScreen());
 }
}
```

Sigin_in_screen.dart

```
import 'package:flutter/material.dart';
import 'package:amazon_clone/screens/sign_up_screen.dart';
import 'package:amazon_clone/utils/color_themes.dart';
import 'package:amazon_clone/utils/constants.dart';
import 'package:amazon_clone/utils/utils.dart';
import 'package:amazon_clone/widgets/custom_main_button.dart';
import 'package:amazon_clone/widgets/text_field_widget.dart';


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


 @override
 State<SignInScreen> createState() => _SignInScreenState();
}


class _SignInScreenState extends State<SignInScreen> {
 TextEditingController emailController = TextEditingController();
  TextEditingController passwordController = TextEditingController();
   bool isLoading = false;


  @override
 void dispose() {
   super.dispose();
   emailController.dispose();
  passwordController.dispose();
 }


 @override
 Widget build(BuildContext context) {
   Size screenSize = Utils().getScreenSize();
```

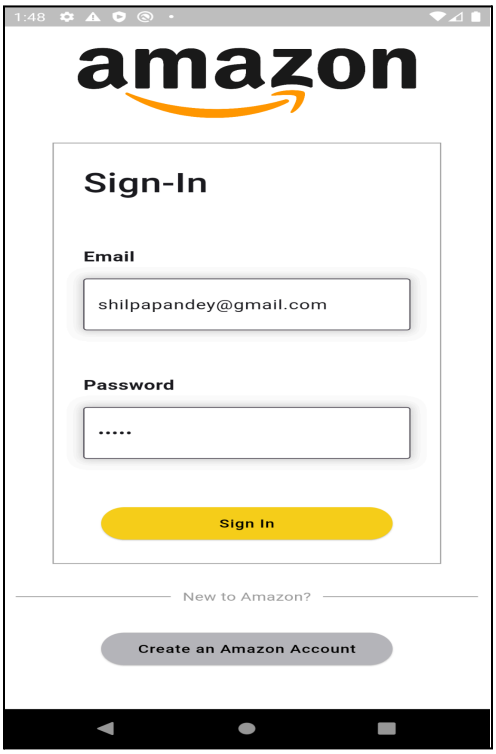
```
return Scaffold(
  backgroundColor: Colors.white,
  body: SingleChildScrollView(
    child: SizedBox(
      height: screenSize.height,
      width: screenSize.width,
      child: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 10, vertical: 20),
        child: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              Image.network(
                amazonLogo,
                height: screenSize.height * 0.10,
              ),
              Container(
                height: screenSize.height * 0.6,
                width: screenSize.width * 0.8,
                padding: const EdgeInsets.all(25),
                decoration: BoxDecoration(
                  border: Border.all(
                    color: Colors.grey,
                    width: 1,
                  ),
                ),
                child: Column(
                  mainAxisSize: MainAxisSize.min,
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    const Text(
                      "Sign-In",
                      style: TextStyle(
                          fontWeight: FontWeight.w500, fontSize: 33),
                    ),
```

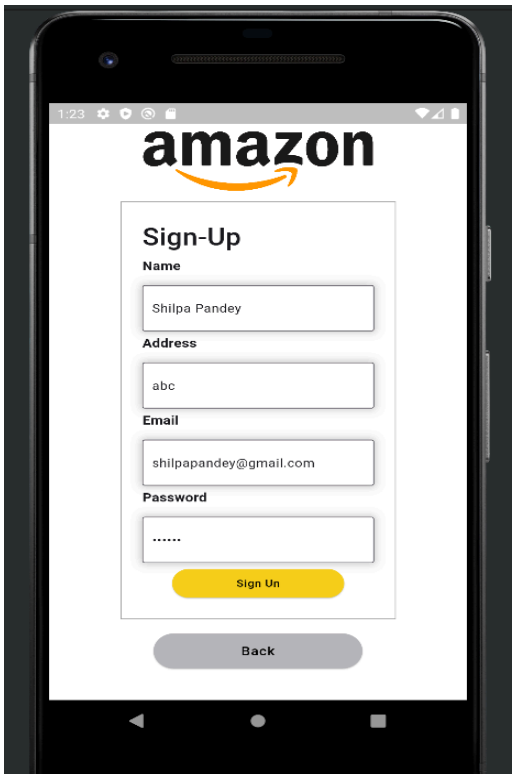
```
        TextFieldWidget(
          title: "Email",
          controller: emailController,
          obscureText: false,
          hintText: "Enter your email",
        ),
        TextFieldWidget(
          title: "Password",
          controller: passwordController,
          obscureText: true,
          hintText: "Enter your password",
        ),
        Align(
          alignment: Alignment.center,
          child: CustomMainButton(
              child: const Text(
                "Sign In",
                style: TextStyle(
                    letterSpacing: 0.6, color: Colors.black),
              ),
              color: yellowColor,
              isLoading: isLoading,
              onPressed: () async {
                setState(() {
                  isLoading = true;
                });


              }),
            )
      ],
    ),
  ),
  Row(
      children: [
        Expanded(
          child: Container(
```

```
                height: 1,
                color: Colors.grey,
              ),
            ),
            const Padding(
              padding: EdgeInsets.symmetric(horizontal: 10),
              child: Text(
                "New to Amazon?",
                style: TextStyle(color: Colors.grey),
              ),
            ),
            Expanded(
              child: Container(
                height: 1,
                color: Colors.grey,
              ),
            ),
          ],
          ),
        CustomMainButton(
            child: const Text(
              "Create an Amazon Account",
              style: TextStyle(
                letterSpacing: 0.6,
                color: Colors.black,
              ),
            ),
            color: Colors.grey[400]!,
            isLoading: false,
            onPressed: () {
              Navigator.pushReplacement(context,
                  MaterialPageRoute(builder: (context) {
                return const SignUpScreen();
              }));
            })
      ],
    ),          ),
```

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

Output:





CONCLUSION:

Understand how to create an interactive Form using form widget

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15M

Experiment 5

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

Theory:

Navigate between screens: Use Navigator methods like push() and pop() to navigate forward and backward through the route stack.

Map route names to their corresponding widgets/screens in the MaterialApp widget using the routes property or use Navigator to push/pop routes dynamically.

Gestures:
Gesture detection: Wrap widgets with gesture detectors such as GestureDetector to detect user gestures.
Gesture handling: Implement callback functions like onTap, onPanUpdate, etc., to respond to specific gestures.

Step 1: First, you need to create two routes.

Step 2: Then, navigate to one route from another route by using the Navigator.push() method.

Step 3: Finally, navigate to the first route by using the Navigator.pop() method.

Code:

Screen layout

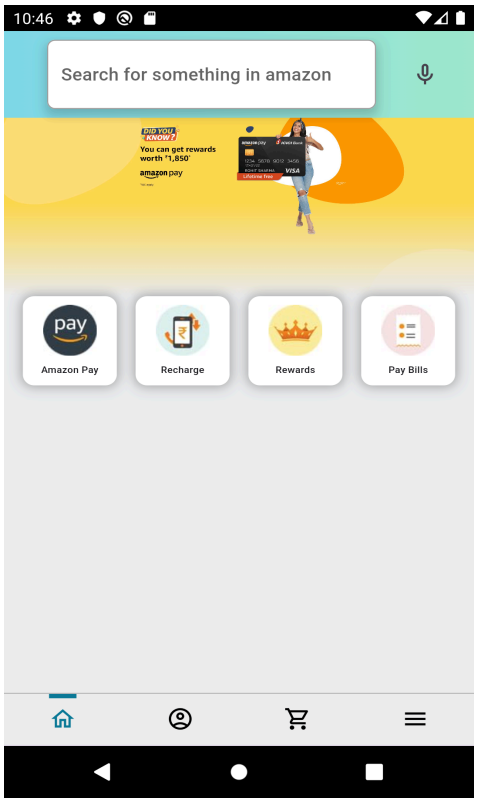
```
 import 'package:amazon_clone/utils/color_themes.dart';
import 'package:amazon_clone/utils/constants.dart';
import 'package:flutter/material.dart';

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


  @override
```

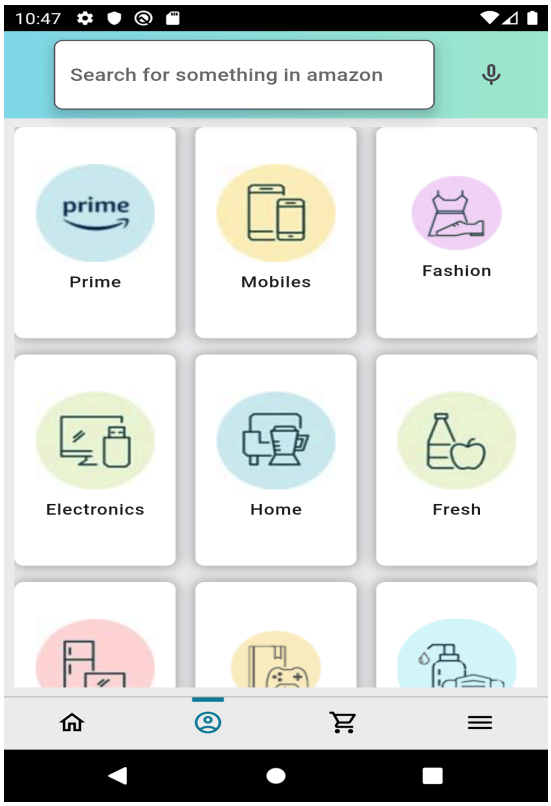
```
  State<ScreenLayout> createState() => _ScreenLayoutState();
}


class _ScreenLayoutState extends State<ScreenLayout> {
  PageController pageController = PageController();
  int currentPage = 0;


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


  changePage(int page) {
    pageController.jumpToPage(page);
    setState(() {
      currentPage = page;
    });
  }


  @override
  void initState() {
    super.initState();
    //CloudFirestoreClass().getNameAndAddress();
  }


  @override
  Widget build(BuildContext context) {
   // Provider.of<UserDetailsProvider>(context).getData();
    return DefaultTabController(
      length: 4,
      child: SafeArea(
        child: Scaffold(
          body: PageView(
            physics: NeverScrollableScrollPhysics(),
```

```
  controller: pageController,
  children: screens,
 ),
 bottomNavigationBar: Container(
  decoration: BoxDecoration(
   border: Border(
    top: BorderSide(color: Colors.grey[400]!, width: 1),
   ),
  ),
  child: TabBar(
   indicator: const BoxDecoration(
    border: Border(
     top: BorderSide(color: activeCyanColor, width: 4),
    ),
   ),
   onTap: changePage,
   indicatorSize: TabBarIndicatorSize.label,
   tabs: [
    Tab(
     child: Icon(
      Icons.home_outlined,
      color: currentPage == 0 ? activeCyanColor : Colors.black,
     ),
    ),
    Tab(
     child: Icon(
      Icons.account_circle_outlined,
      color: currentPage == 1 ? activeCyanColor : Colors.black,
     ),
    ),
    Tab(
     child: Icon(
      Icons.shopping_cart_outlined,
      color: currentPage == 2 ? activeCyanColor : Colors.black,
     ),
    ),
    Tab(
     child: Icon(
      Icons.menu,
      color: currentPage == 3 ? activeCyanColor : Colors.black,
```

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

Output:





Search

```
import 'package:amazon_clone/widgets/search_bar_widget.dart';
import 'package:flutter/material.dart';
```

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


  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: SearchBarWidget(isReadOnly: false, hasBackButton: true),
    );
  }
}
```

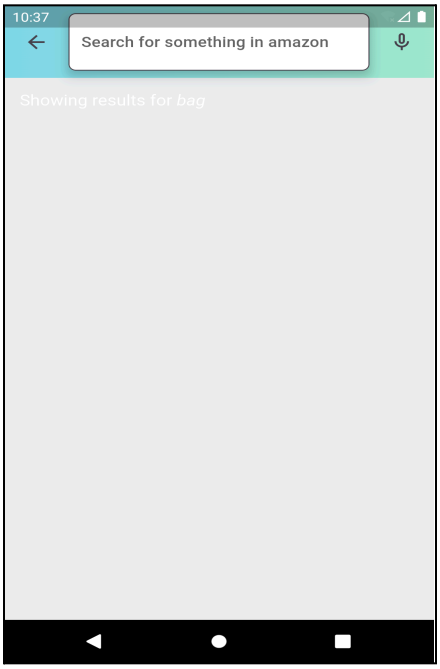
Screen widget

```
import 'package:amazon_clone/screens/results_screen.dart';
import 'package:amazon_clone/screens/search_screen.dart';
import 'package:amazon_clone/utils/color_themes.dart';
import 'package:amazon_clone/utils/constants.dart';
import 'package:amazon_clone/utils/utils.dart';
import 'package:flutter/material.dart';


class SearchBarWidget extends StatefulWidget implements  PreferredSizeWidget {
  final bool isReadOnly;
  final bool hasBackButton;
  SearchBarWidget({
    Key? key,
    required this.isReadOnly,
    required this.hasBackButton,
  })  : preferredSize = const Size.fromHeight(kAppBarHeight),
        super(key: key);
```

```
  @override
  final Size preferredSize;


  @override
  State<SearchBarWidget> createState() => _SearchBarWidgetState();
}


class _SearchBarWidgetState extends State<SearchBarWidget> {
  OutlineInputBorder border = OutlineInputBorder(
    borderRadius: BorderRadius.circular(7),
    borderSide: const BorderSide(
      color: Colors.grey,
      width: 1,
    ),
  );


  @override
  Widget build(BuildContext context) {
    Size screenSize = Utils().getScreenSize();
    return Container(
      height: kAppBarHeight,
      decoration: const BoxDecoration(
        gradient: LinearGradient(
          colors: backgroundGradient,
          begin: Alignment.centerLeft,
          end: Alignment.centerRight,
        ),
      ),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: [
          widget.hasBackButton
              ? IconButton(
                  onPressed: () {
```

```
          Navigator.pop(context);
        },
        icon: const Icon(Icons.arrow_back))
    : Container(),
SizedBox(
  width: screenSize.width * 0.7,
  child: Container(
    decoration: BoxDecoration(
      boxShadow: [
        BoxShadow(
          color: Colors.black.withOpacity(0.2),
          blurRadius: 8,
          spreadRadius: 1,
          offset: const Offset(0, 5),
        ),
      ],
    ),
    child: TextField(
      onSubmitted: (String query) {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => ResultsScreen(query: query),
          ),
        );
      },
      readOnly: widget.isReadOnly,
      onTap: () {
        if (widget.isReadOnly) {
          Navigator.push(
              context,
              MaterialPageRoute(
                  builder: (context) => const SearchScreen()));
        }
      },
      decoration: InputDecoration(
        hintText: "Search for something in amazon",
```

```
                fillColor: Colors.white,
                filled: true,
                border: border,
                focusedBorder: border,
              ),
            ),
          ),
        ),
        IconButton(
          onPressed: () {},
          icon: const Icon(Icons.mic_none_outlined),
        )
      ],
    ),
  );
 }
}
```



Conclusion:

Understand the theoretical concept of navigation, routing and gestures and also how to use them in flutter App.

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	15M

Experiment 6

Aim: To Connect Flutter UI with firebase database

Theory:

Firebase is a Backend-as-a-Service (BaaS) app development platform that provides hosted backend services such as a realtime database, cloud storage, authentication, crash reporting, machine learning, remote configuration, and hosting for your static files.
Firebase is a Backend-as-a-Service (Baas) that provides developers with a variety of tools and services to help them:

- Real-time database: Stores and syncs data in real time in apps
- Cloud messaging: Used to send messages to users' devices in apps
- Authentication: Used to authenticate users in apps
- Database: Stores and syncs data in real-time, sometimes referred to as a real-time document store

Code:

Authentication.dart

```
import 'package:amazon_clone/model/user_details_model.dart';
import 'package:amazon_clone/resources/cloudfirestore_methods.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';

class AuthenticationMethods {
  FirebaseAuth firebaseAuth = FirebaseAuth.instance;
  CloudFirestoreClass cloudFirestoreClass = CloudFirestoreClass();
```

```
Future<String> signUpUser(
    {required String name,
     required String address,
     required String email,
     required String password}) async {
  name.trim();
  address.trim();
  email.trim();
  password.trim();
  String output = "Something went wrong";
  if (name != "" && address != "" && email != "" && password != "") {
    try {
      await firebaseAuth.createUserWithEmailAndPassword(
          email: email, password: password);
      UserDetailsModel user = UserDetailsModel(name: name, address: address);
      await cloudFirestoreClass.uploadNameAndAddressToDatabase(user: user);
      output = "success";
    } on FirebaseAuthException catch (e) {
      output = e.message.toString();
    }
  } else {
    output = "Please fill up all the fields.";
  }
  return output;
}

Future<String> signInUser(
    {required String email, required String password}) async {
  email.trim();
  password.trim();
  String output = "Something went wrong";
  if (email != "" && password != "") {
    try {
      await firebaseAuth.signInWithEmailAndPassword(
          email: email, password: password);
      output = "success";
    } on FirebaseAuthException catch (e) {
      output = e.message.toString();
    }
  } else {
```

```
      output = "Please fill up all the fields.";
    }
    return output;
  }
}
```

cloudFirestore.dart

```
import 'dart:typed_data';
import 'package:amazon_clone/model/order_request_model.dart';
import 'package:amazon_clone/model/product_model.dart';
import 'package:amazon_clone/model/review_model.dart';
import 'package:amazon_clone/model/user_details_model.dart';
import 'package:amazon_clone/utils/utils.dart';
import 'package:amazon_clone/widgets/simple_product_widget.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/cupertino.dart';

class CloudFirestoreClass {
  FirebaseFirestore firebaseFirestore = FirebaseFirestore.instance;
  FirebaseAuth firebaseAuth = FirebaseAuth.instance;

  Future uploadNameAndAddressToDatabase(
      {required UserDetailsModel user}) async {
    await firebaseFirestore
        .collection("users")
        .doc(firebaseAuth.currentUser!.uid)
        .set(user.getJson());
  }

  Future getNameAndAddress() async {
    DocumentSnapshot snap = await firebaseFirestore
        .collection("users")
        .doc(firebaseAuth.currentUser!.uid)
        .get();

    UserDetailsModel userModel = UserDetailsModel.getModelFromJson(
```

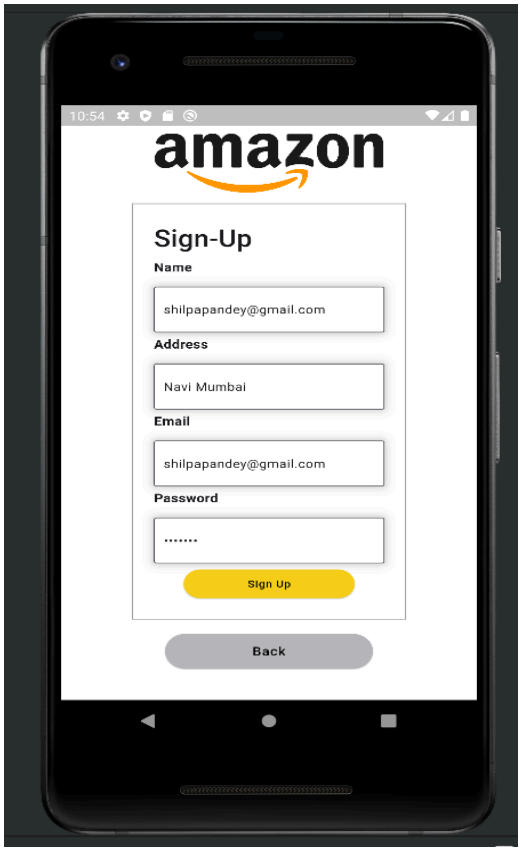
```
    (snap.data() as dynamic),
  );

  return userModel;
}

Future<String> uploadProductToDatabase({
  required Uint8List? image,
  required String productName,
  required String rawCost,
  required int discount,
  required String sellerName,
  required String sellerUid,
}) async {
  productName.trim();
  rawCost.trim();
  String output = "Something went wrong";

  if (image != null && productName != "" && rawCost != "") {
    try {
      String uid = Utils().getUid();
      String url = await uploadImageToDatabase(image: image, uid: uid);
      double cost = double.parse(rawCost);
      cost = cost - (cost * (discount / 100));
      ProductModel product = ProductModel(
          url: url,
          productName: productName,
          cost: cost,
          discount: discount,
          uid: uid,
          sellerName: sellerName,
          sellerUid: sellerUid,
          rating: 5,
          noOfRating: 0);

      await firebaseFirestore
          .collection("products")
          .doc(uid)
          .set(product.getJson());
      output = "success";
```

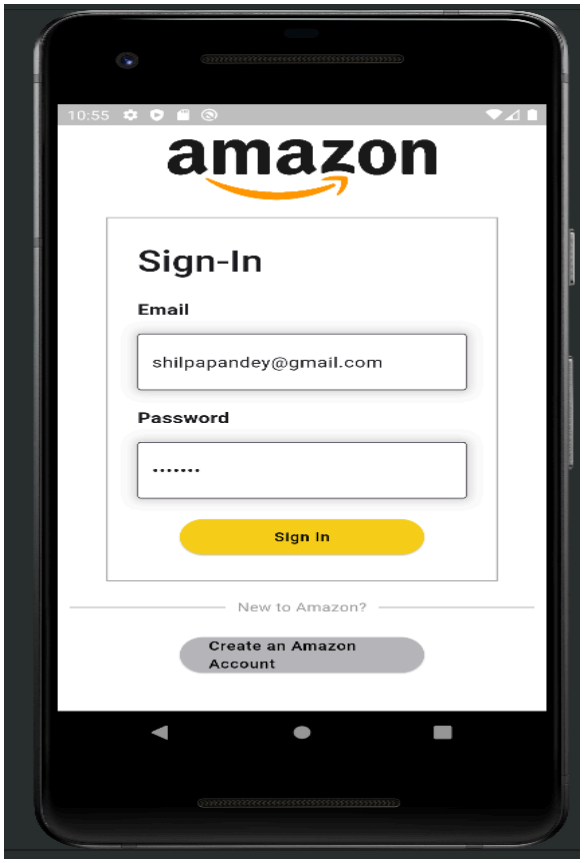
```
    } catch (e) {
      output = e.toString();
    }
  } else {
    output = "Please make sure all the fields are not empty";
  }

  return output;
}

Future<String> uploadImageToDatabase(
    {required Uint8List image, required String uid}) async {
  Reference storageRef =
  FirebaseStorage.instance.ref().child("products").child(uid);
  UploadTask uploadToask = storageRef.putData(image);
  TaskSnapshot task = await uploadToask;
  return task.ref.getDownloadURL();
}

Future<List<Widget>> getProductsFromDiscount(int discount) async {
  List<Widget> children = [];
  QuerySnapshot<Map<String, dynamic>> snap = await firebaseFirestore
      .collection("products")
      .where("discount", isEqualTo: discount)
      .get();

  for (int i = 0; i < snap.docs.length; i++) {
    DocumentSnapshot docSnap = snap.docs[i];
    ProductModel model =
    ProductModel.getModelFromJson(json: (docSnap.data() as dynamic));
    children.add(SimpleProductWidget(productModel: model));
  }
  return children;
}

Future uploadReviewToDatabase(
    {required String productUid, required ReviewModel model}) async {
  await firebaseFirestore
      .collection("products")
      .doc(productUid)
```

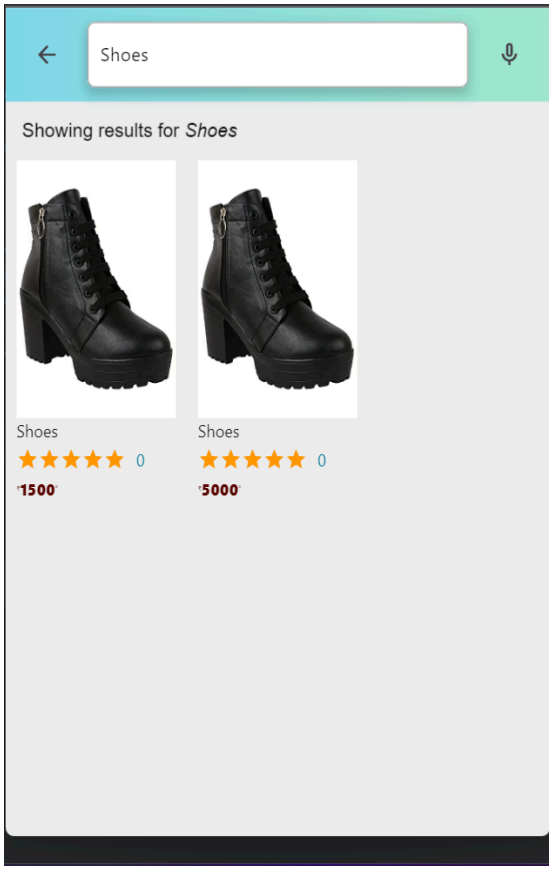
```
      .collection("reviews")
      .add(model.getJson());
  await changeAverageRating(productUid: productUid, reviewModel: model);
}

Future addProductToCart({required ProductModel productModel}) async {
  await firebaseFirestore
      .collection("users")
      .doc(firebaseAuth.currentUser!.uid)
      .collection("cart")
      .doc(productModel.uid)
      .set(productModel.getJson());
}

Future deleteProductFromCart({required String uid}) async {
  await firebaseFirestore
      .collection("users")
      .doc(firebaseAuth.currentUser!.uid)
      .collection("cart")
      .doc(uid)
      .delete();
}

Future buyAllItemsInCart({required UserDetailsModel userDetails}) async {
  QuerySnapshot<Map<String, dynamic>> snapshot = await firebaseFirestore
      .collection("users")
      .doc(firebaseAuth.currentUser!.uid)
      .collection("cart")
      .get();

  for (int i = 0; i < snapshot.docs.length; i++) {
    ProductModel model =
    ProductModel.getModelFromJson(json: snapshot.docs[i].data());
    addProductToOrders(model: model, userDetails: userDetails);
    await deleteProductFromCart(uid: model.uid);
  }
}

Future addProductToOrders(
    {required ProductModel model,
```

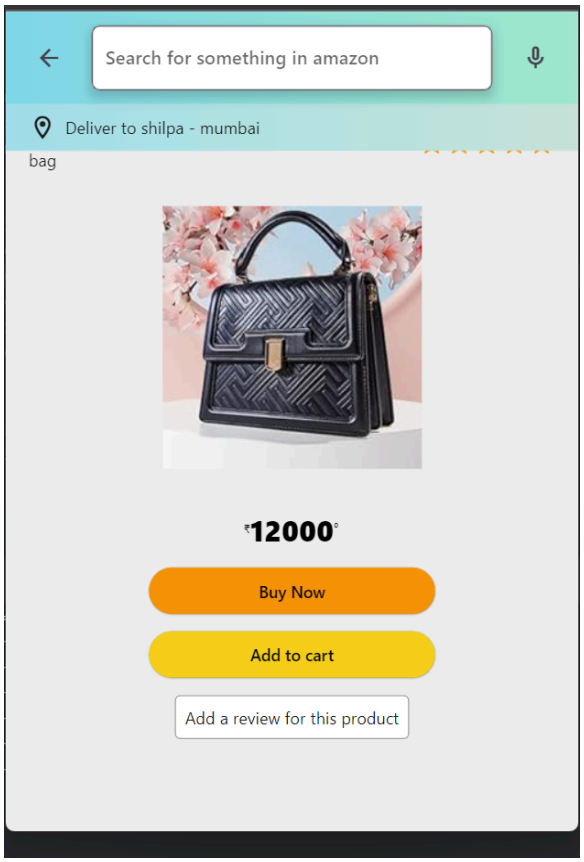
```
      required UserDetailsModel userDetails}) async {
  await firebaseFirestore
      .collection("users")
      .doc(firebaseAuth.currentUser!.uid)
      .collection("orders")
      .add(model.getJson());
  await sendOrderRequest(model: model, userDetails: userDetails);
 }

 Future sendOrderRequest(
     {required ProductModel model,
      required UserDetailsModel userDetails}) async {
  OrderRequestModel orderRequestModel = OrderRequestModel(
      orderName: model.productName, buyersAddress: userDetails.address);
  await firebaseFirestore
      .collection("users")
      .doc(model.sellerUid)
      .collection("orderRequests")
      .add(orderRequestModel.getJson());
 }

 Future changeAverageRating(
     {required String productUid, required ReviewModel reviewModel}) async {
  DocumentSnapshot snapshot =
  await firebaseFirestore.collection("products").doc(productUid).get();
  ProductModel model =
  ProductModel.getModelFromJson(json: (snapshot.data() as dynamic));
  int currentRating = model.rating;
  int newRating = ((currentRating + reviewModel.rating) / 2).toInt();
  await firebaseFirestore
      .collection("products")
      .doc(productUid)
      .update({"rating": newRating});
 }
}
```

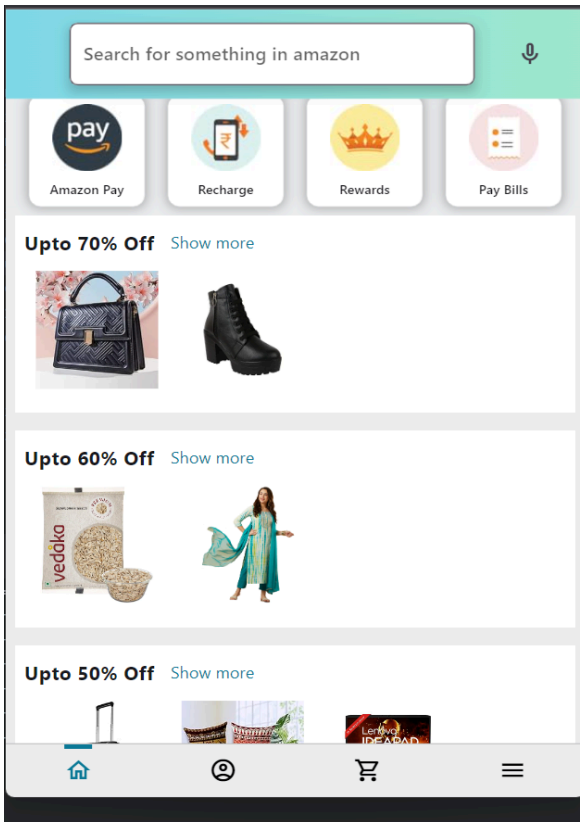
Output:

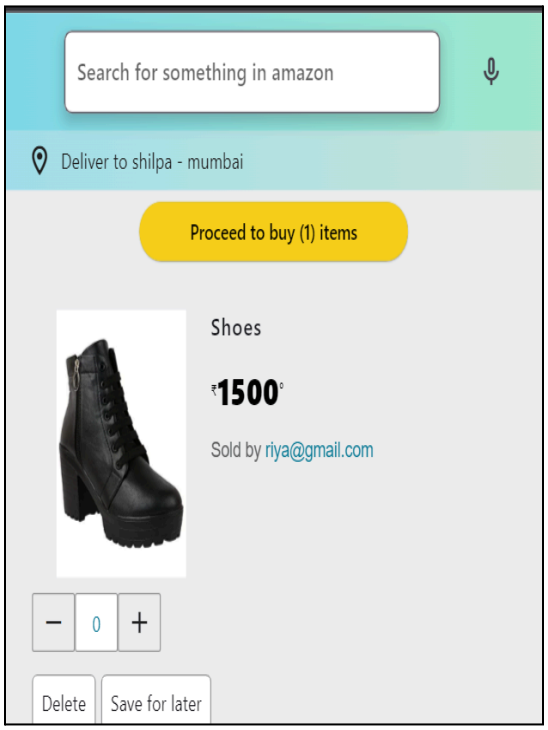

10:54
amazon
Sign-Up
Name
shilpapandey@gmail.com
Address
Navi Mumbai
Email
shilpapandey@gmail.com
Password
.....
Sign Up
Back


10:55
amazon
Sign-In
Email
shilpapandey@gmail.com
Password
.....
Sign In
New to Amazon?
Create an Amazon Account

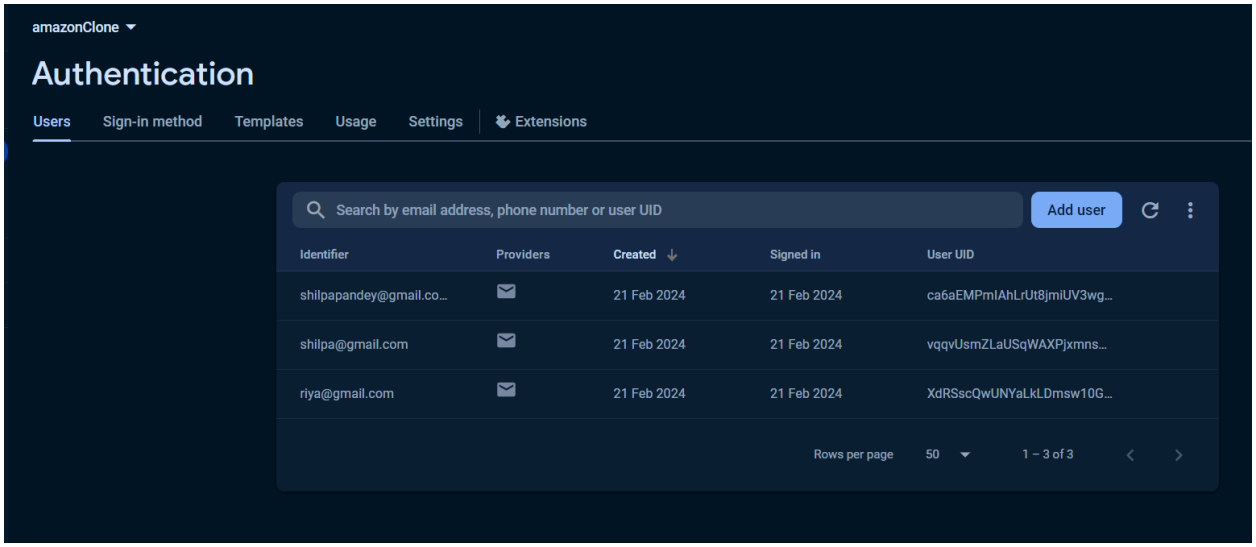

Shoes
Showing results for *Shoes*
Shoes
0
₹1500
Shoes
0
₹5000


Search for something in amazon
Deliver to shilpa - mumbai
bag
₹12000
Buy Now
Add to cart
Add a review for this product

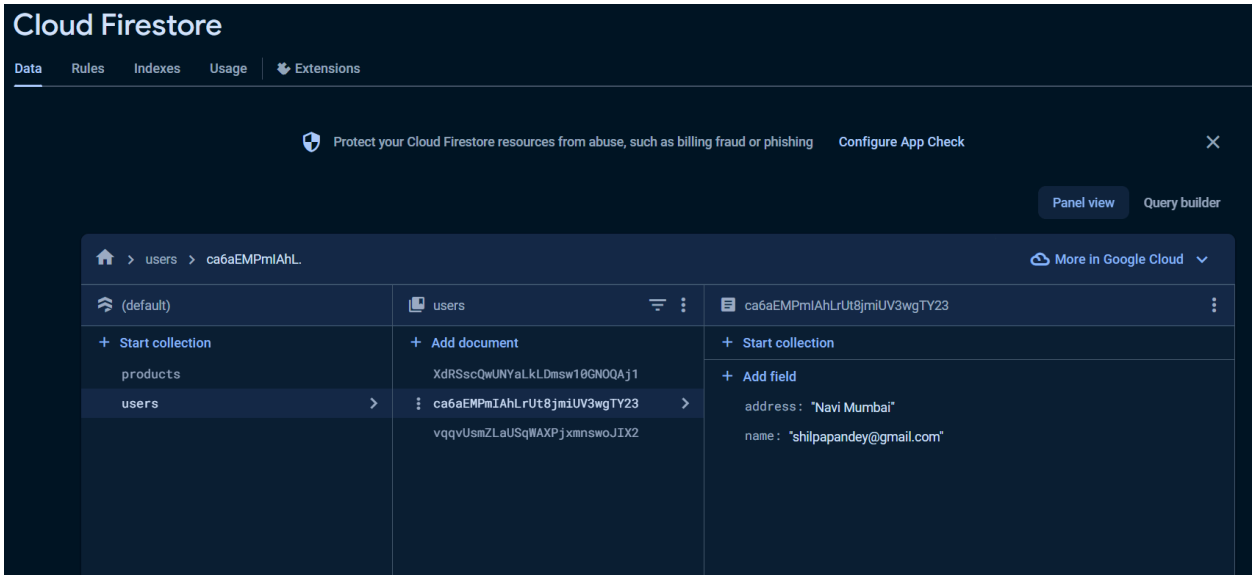

Search for something in amazon
pay
Amazon Pay
Recharge
Rewards
Pay Bills
Upto 70% Off
Show more
Upto 60% Off
Show more
vedaka
Upto 50% Off
Show more

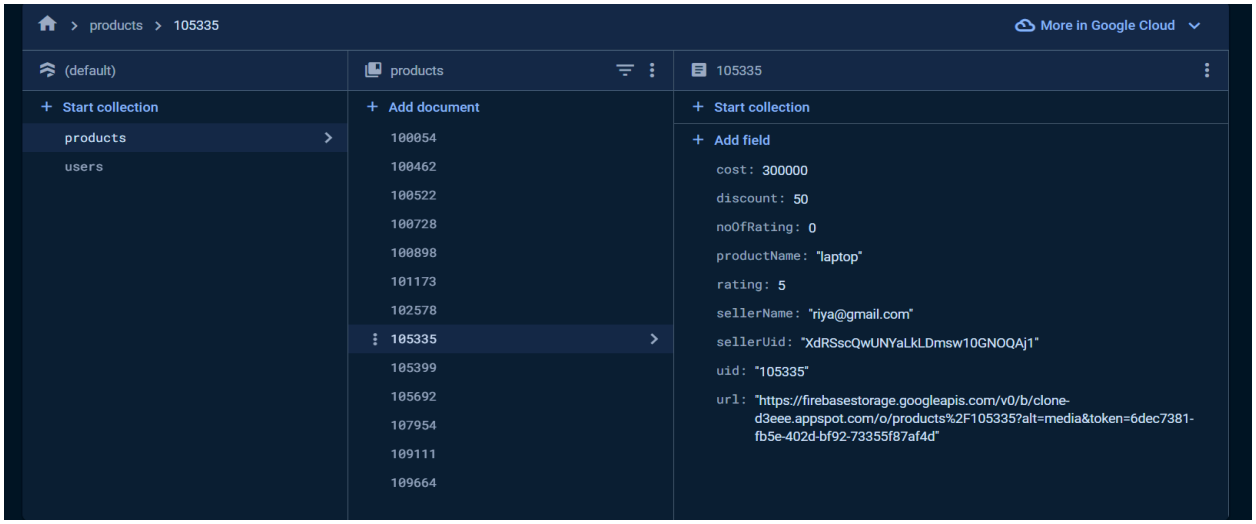

Search for something in amazon
Deliver to shilpa - mumbai
Proceed to buy (1) items
Shoes
₹1500
Sold by riya@gmail.com
0
Delete
Save for later

Firebase Authentication:

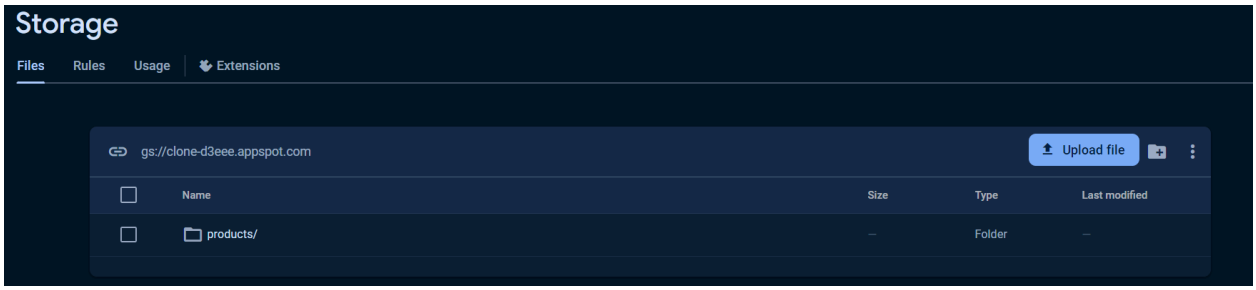


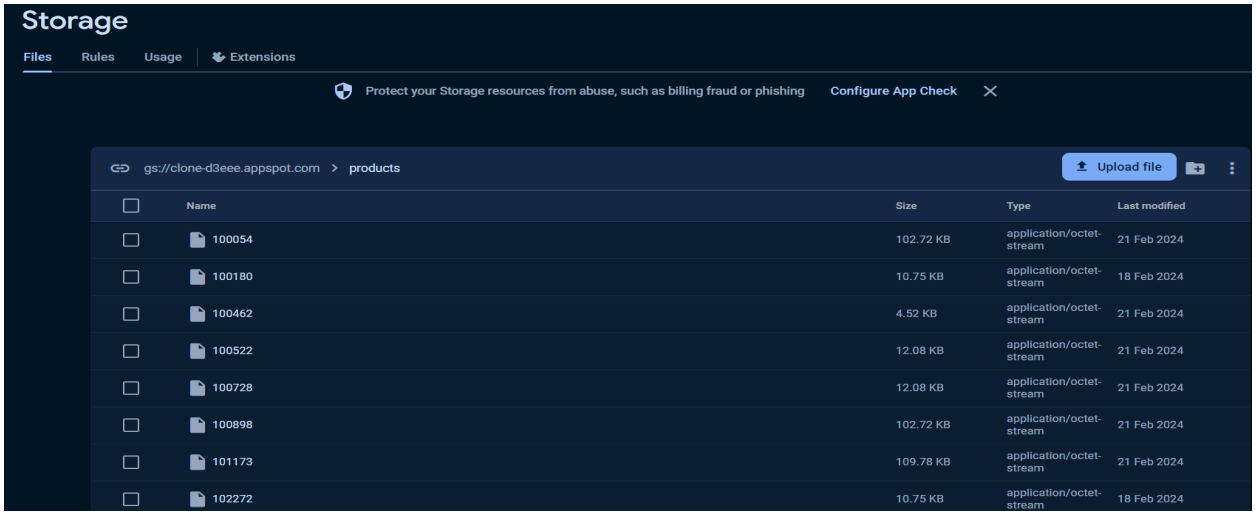
FireStore Database:





Storage:





Conclusion:

Successfully Connected Flutter UI with Firebase database.

MAD & PWA Lab
Journal

Experiment No.	07
Experiment Title.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.
Roll No.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15M

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.

Code:

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>ShoppingMart</title>
    <link rel="stylesheet" href="css/style.css" />
    <link
      rel="stylesheet"

href="https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined:opsz,wght,
FILL,GRAD@20..48,100..700,0..1,-50..200"
    />
  </head>
  <body>
    <header>
      <div class="logo_container">
        <a href="#"
          ><img
            class="myntra_home"
            src="images/myntra_logo.webp"
            alt="Myntra Home"
        /></a>
      </div>
      <nav class="nav_bar">
        <a href="#">Men</a>
        <a href="#">Women</a>
        <a href="#">Kids</a>
        <a href="#">Home & Living</a>
        <a href="#">Beauty</a>
        <a href="#">Studio <sup>New</sup></a>
      </nav>
```

```
  <div class="search_bar">
    <span class="material-symbols-outlined search_icon">search</span>
    <input
      class="search_input"
      placeholder="Search for products, brands and more"
    />
  </div>
  <div class="action_bar">
    <div class="action_container">
      <span class="material-symbols-outlined action_icon">person</span>
      <span class="action_name">Profile</span>
    </div>


    <div class="action_container">
      <span class="material-symbols-outlined action_icon">favorite</span>
      <span class="action_name">Wishlist</span>
    </div>


    <a class="action_container" href="pages/bag.html">
      <span class="material-symbols-outlined action_icon"
        >shopping_bag</span
      >
      <span class="action_name">Bag</span>
      <span class="bag-item-count">0</span>
    </a>
  </div>
</header>
<main>
  <div class="items-container"></div>
</main>
<footer>
  <div class="footer_container">
    <div class="footer_column">
      <h3>ONLINE SHOPPING</h3>
```

```
    <a href="#">Men</a>
    <a href="#">Women</a>
    <a href="#">Kids</a>
    <a href="#">Home & Living</a>
    <a href="#">Beauty</a>
    <a href="#">Gift Card</a>
    <a href="#">Myntra Insider</a>
  </div>


  <div class="footer_column">
    <h3>ONLINE SHOPPING</h3>


    <a href="#">Men</a>
    <a href="#">Women</a>
    <a href="#">Kids</a>
    <a href="#">Home & Living</a>
    <a href="#">Beauty</a>
    <a href="#">Gift Card</a>
    <a href="#">Myntra Insider</a>
  </div>


  <div class="footer_column">
    <h3>ONLINE SHOPPING</h3>
    <a href="#">Men</a>
    <a href="#">Women</a>
    <a href="#">Kids</a>
    <a href="#">Home & Living</a>
    <a href="#">Beauty</a>
    <a href="#">Gift Card</a>
    <a href="#">Myntra Insider</a>
  </div>
</div>
<hr />
```

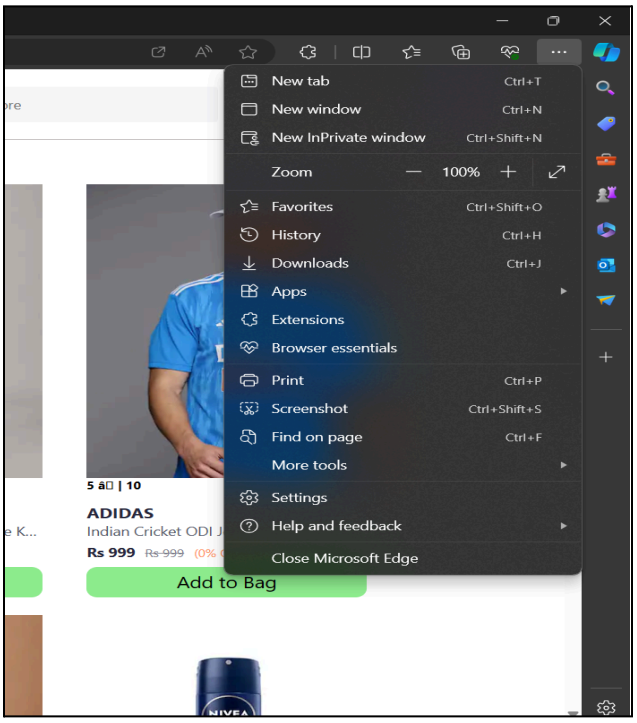
```
    <div class="copyright">© 2023 www.myntra.com. All rights reserved.</div>
  </footer>
  <script src="data/items.js"></script>
  <script src="scripts/index.js"></script>
 </body>
</html>
```

index.js

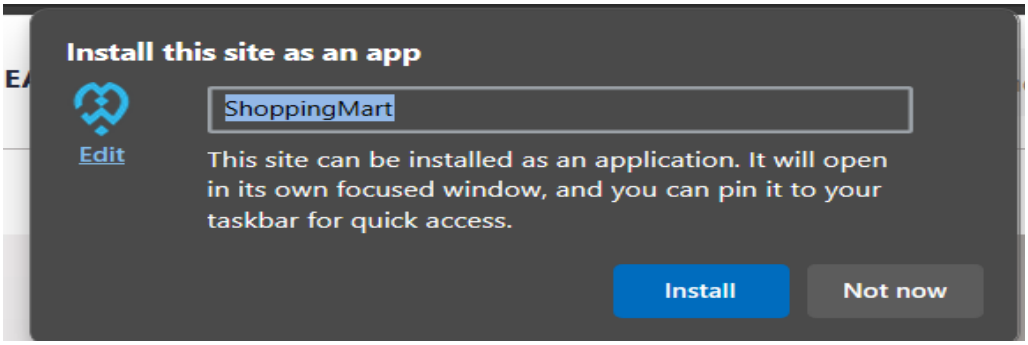
```
let bagItems;
// localStorage.clear();
onLoad();


function onLoad() {
 let bagItemsStr = localStorage.getItem('bagItems');
 bagItems = bagItemsStr ? JSON.parse(bagItemsStr) : [];
 displayItemsOnHomePage();
 displayBagIcon();
}


function addToBag(itemId) {
 bagItems.push(itemId);
 localStorage.setItem('bagItems', JSON.stringify(bagItems));
 displayBagIcon();
}


function displayBagIcon() {
 let bagItemCountElement = document.querySelector('.bag-item-count');
 if (bagItems.length > 0) {
  console.log('I am here');
  bagItemCountElement.style.visibility = 'visible';
```

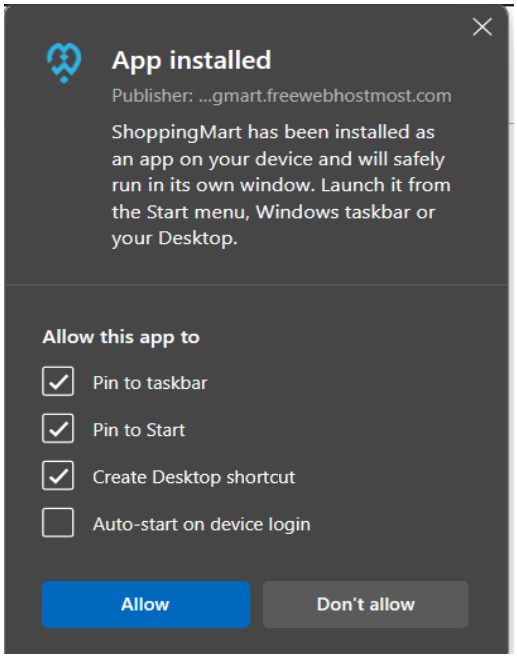
```
    bagItemCountElement.innerText = bagItems.length;
  } else {
    bagItemCountElement.style.visibility = 'hidden';
  }
}


function displayItemsOnHomePage() {
  let itemsContainerElement = document.querySelector('.items-container');
  if (!itemsContainerElement) {
    return;
  }
  let innerHtml = '';
  items.forEach(item => {
    innerHtml += `
    <div class="item-container">
      <img class="item-image" src="${item.image}" alt="item image">
      <div class="rating">
         ${item.rating.stars} ⭐ | ${item.rating.count}
      </div>
      <div class="company-name">${item.company}</div>
      <div class="item-name">${item.item_name}</div>
      <div class="price">
         <span class="current-price">Rs ${item.current_price}</span>
         <span class="original-price">Rs ${item.original_price}</span>
         <span class="discount">(${item.discount_percentage}% OFF)</span>
      </div>
      <button class="btn-add-bag" onclick="addToBag(${item.id})">Add to Bag</button>
    </div>`
  });
  itemsContainerElement.innerHTML = innerHtml;
}
```

Open folder in VS code and click go live at bottom right corner
Open your hosted site on Microsoft Edge

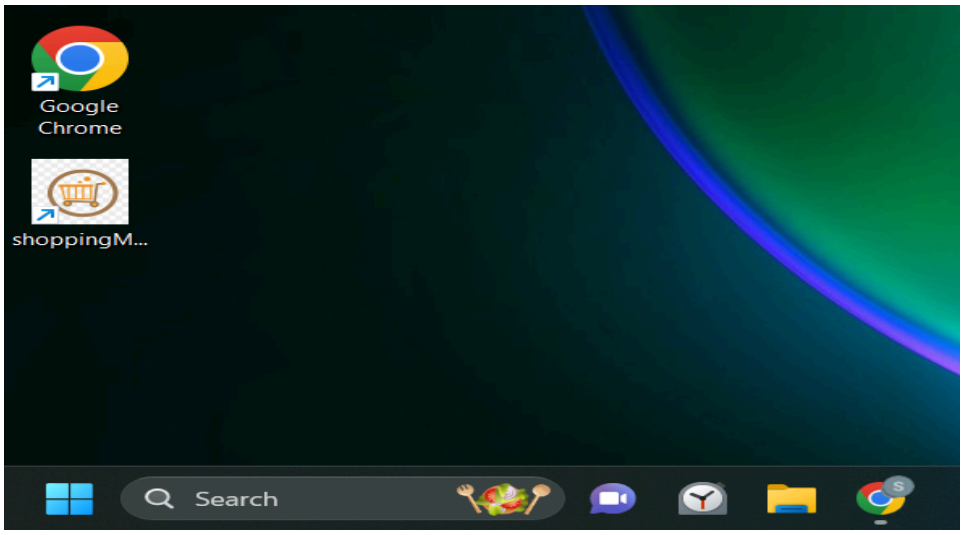


Click on 3 dots
click on Apps
select install app option



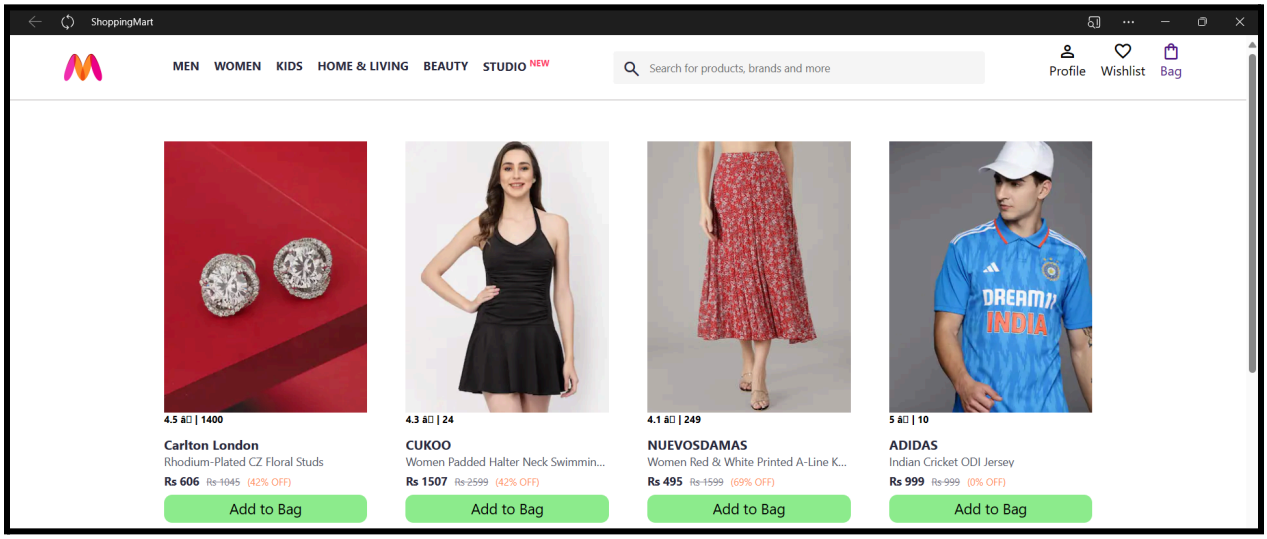


Desktop App Created Successfully



Open Desktop App:

Output:



Conclusion:

In this experiment, we have successfully created a basic progressive web app of our web page and installed it in our desktop successfully

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.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15M

Experiment 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.

Theory:

Service Worker

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

Things to note about Service Worker:

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

What can we do with Service Workers?

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

- You can **Cache**
  You can cache any request/response pair with Service Worker and Cache API

and you can access these offline content anytime.

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

What can't we do with Service Workers?

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

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

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).

Index.html

```
<!DOCTYPE html>
```

```
<html lang="en">
  <head>
    <title>ShoppingMart</title>
    <link rel="stylesheet" href="css/style.css" />
    <link
      rel="stylesheet"

href="https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined:opsz,wght,FILL,GRAD@20..48,100..700,0..1,-50..200"
    />
  </head>
  <body>
    <header>
      <div class="logo_container">
        <a href="#"
          ><img
            class="myntra_home"
            src="images/myntra_logo.webp"
            alt="Myntra Home"
        /></a>
      </div>

      <div class="search_bar">
        <span class="material-symbols-outlined search_icon">search</span>
        <input
          class="search_input"
          placeholder="Search for products, brands and more"
        />
      </div>
      <div class="action_bar">
        <div class="action_container">
          <span class="material-symbols-outlined action_icon">person</span>
          <span class="action_name">Profile</span>
        </div>


        <div class="action_container">
          <span class="material-symbols-outlined action_icon">favorite</span>
          <span class="action_name">Wishlist</span>
        </div>
```

```
    <a class="action_container" href="pages/bag.html">
      <span class="material-symbols-outlined action_icon"
        >shopping_bag</span
      >
      <span class="action_name">Bag</span>
      <span class="bag-item-count">0</span>
    </a>
  </div>
</header>
<main>
  <div class="items-container"></div>
</main>
<footer>
  <div class="footer_container">
    <div class="footer_column">
      <h3>ONLINE SHOPPING</h3>


      <a href="#">Men</a>
      <a href="#">Women</a>
      <a href="#">Kids</a>
      <a href="#">Home & Living</a>
      <a href="#">Beauty</a>
      <a href="#">Gift Card</a>
      <a href="#">Myntra Insider</a>
    </div>


    <div class="footer_column">
      <h3>ONLINE SHOPPING</h3>


      <a href="#">Men</a>
      <a href="#">Women</a>
      <a href="#">Kids</a>
      <a href="#">Home & Living</a>
      <a href="#">Beauty</a>
      <a href="#">Gift Card</a>
```

```
        <a href="#">Myntra Insider</a>
       </div>


       <div class="footer_column">
        <h3>ONLINE SHOPPING</h3>


        <a href="#">Men</a>
        <a href="#">Women</a>
        <a href="#">Kids</a>
        <a href="#">Home & Living</a>
        <a href="#">Beauty</a>
        <a href="#">Gift Card</a>
        <a href="#">Myntra Insider</a>
       </div>
     </div>
     <hr />
     <div class="copyright">© 2023 www.myntra.com. All rights reserved.</div>
    </footer>
    <script src="data/items.js"></script>
    <script src="scripts/index.js"></script>
    <script src="app.js"></script>
  </body>
</html>
```

app.js

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

```
  });
}
```

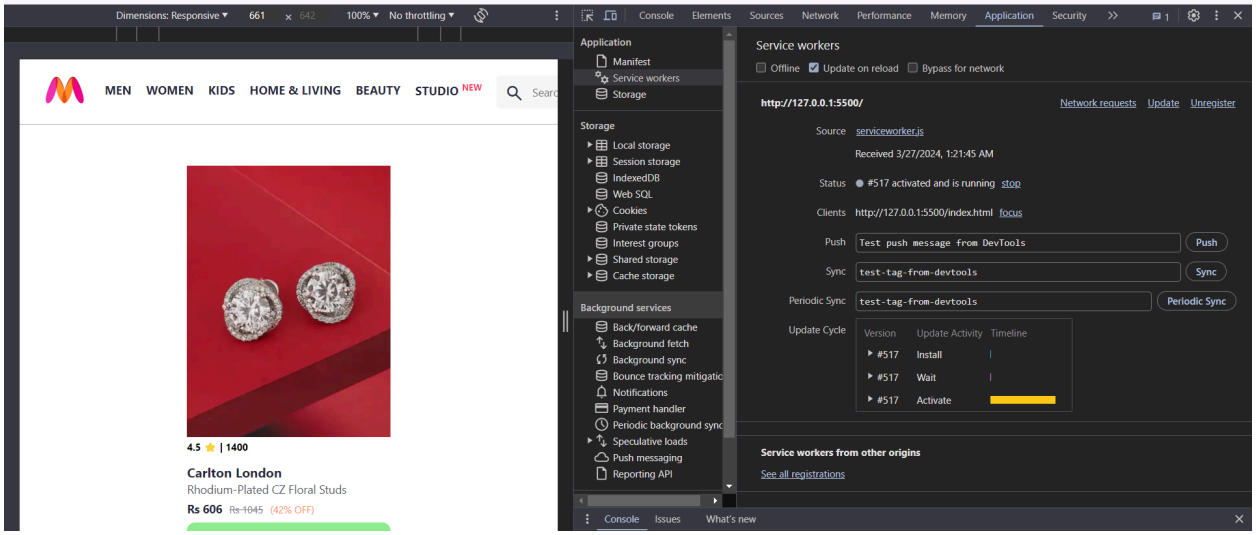
service-worker.js

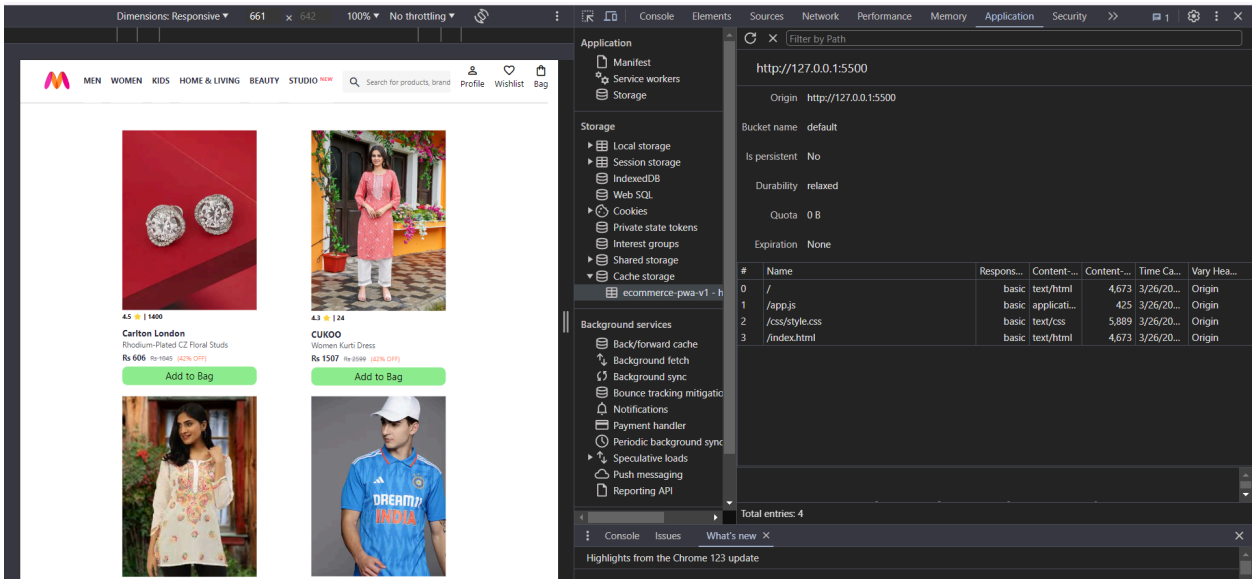
```
const cacheName = "ecommerce-pwa-v1";
const assetsToCache = ["/", "/index.html", "../css/style.css", "/app.js"];
self.addEventListener("install", (event) => {
  event.waitUntil(
    caches.open(cacheName).then((cache) => {
      return cache.addAll(assetsToCache);
    })
  );
});
self.addEventListener("activate", (event) => {
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames
          .filter((name) => {
            return name !== cacheName;
          })
          .map((name) => {
            return caches.delete(name);
          })
      );
    })
  );
});
```

Steps for Execution

- Create a folder and put all 4 files main.css , service-worker.js, app.js, index.html
- open visual studio install extension Live server
- open folder in visual studio open index.html on bottom right corner
- click go Live it will open html page in browser go to developer tools

output:





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

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.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15M

Experiment 9

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

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

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

Sync Event

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

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

Push Event

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

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

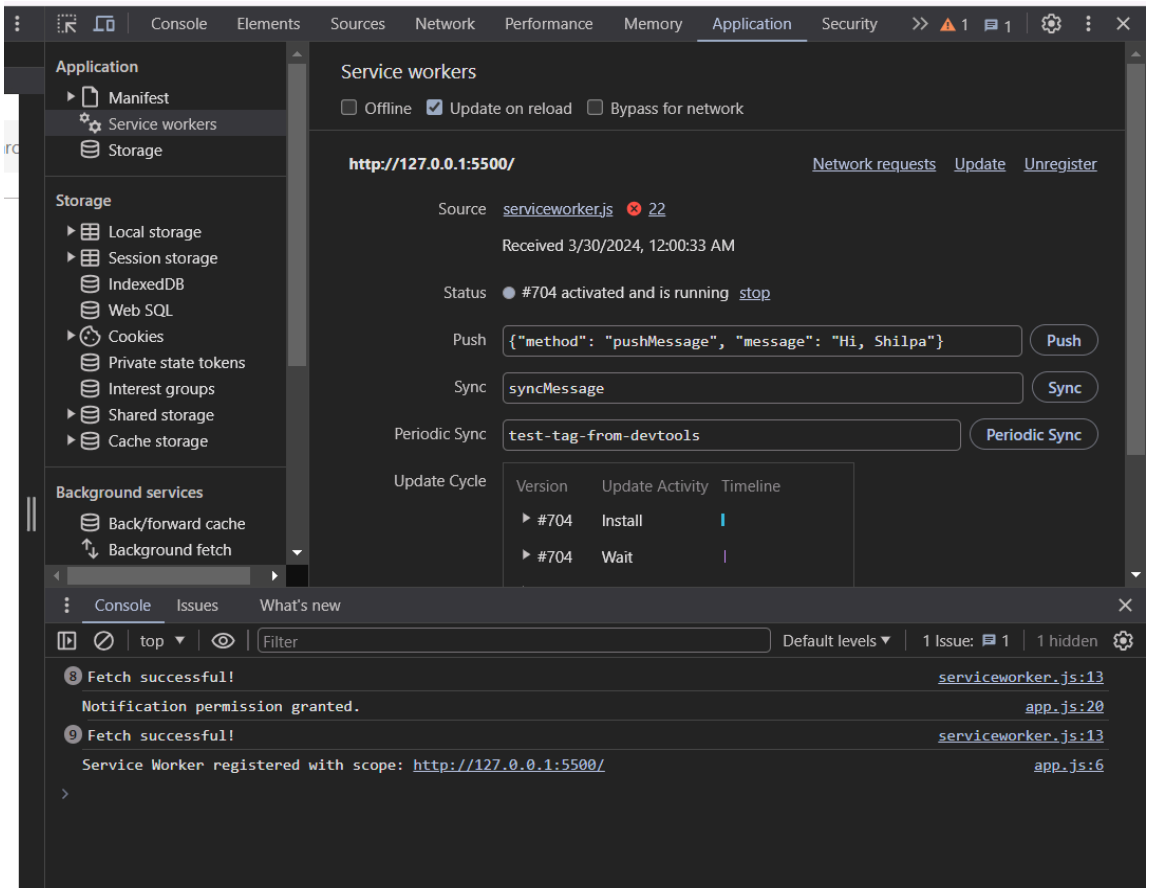
In the following code block is in sw.js file. You can handle push notifications with this event. In this example, I kept it simple. We send an object that has "method" and "message" properties. If the method value is "pushMessage", we open the information notification with the "message" property.

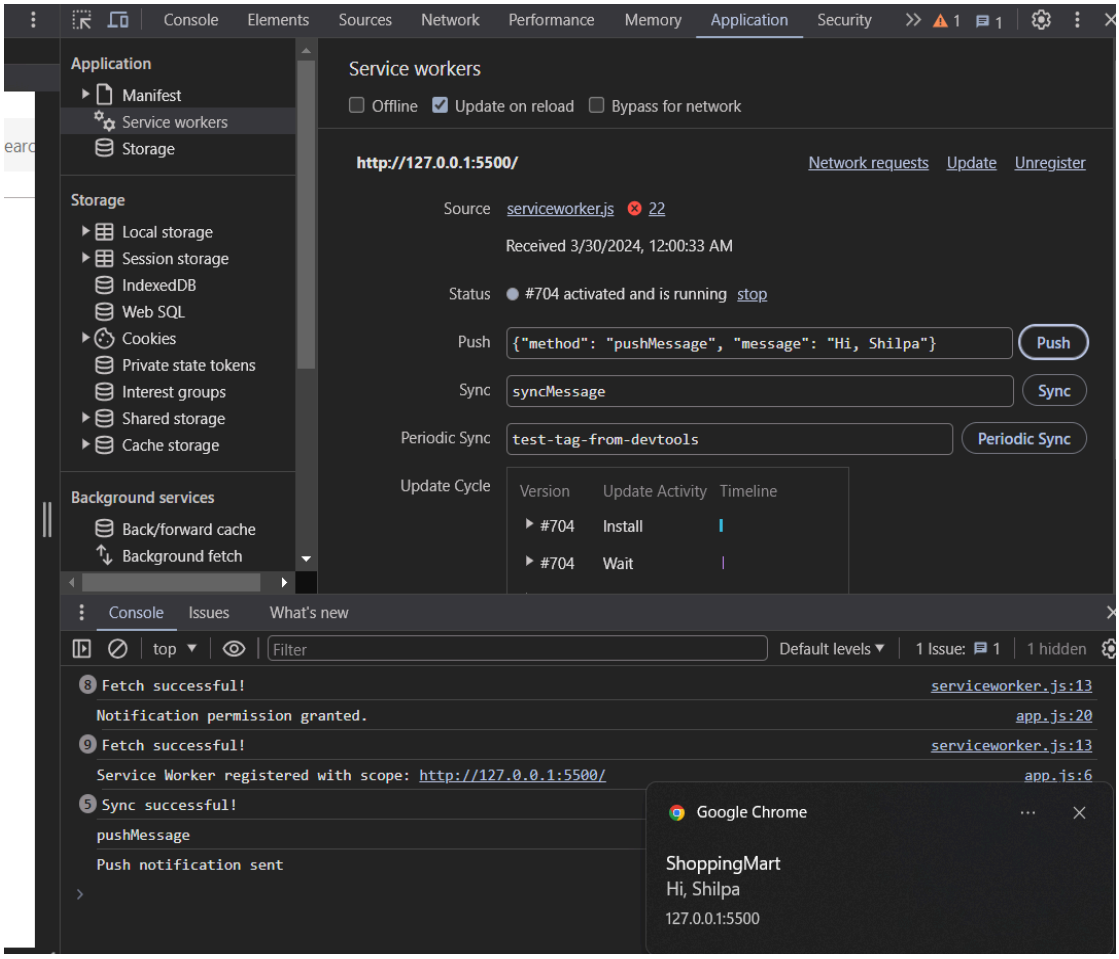
```
//serviceworker
self.addEventListener("install", function (event) {
  event.waitUntil(preLoad());
});

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

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

```
      console.error("Error parsing push data:", error);
    }
  }
});

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

var preLoad = function () {
  return caches.open("offline").then(function (cache) {
    // caching index and important routes
    return cache.addAll(["/index.html", "/images/10.jpg", "/css/style.css"]);
  });
};

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 : In this experiment, we have successfully implemented service worker events like fetch, sync and push for E-commerce PWA and found out output for above implementation

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15M

Experiment 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.

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

Pros

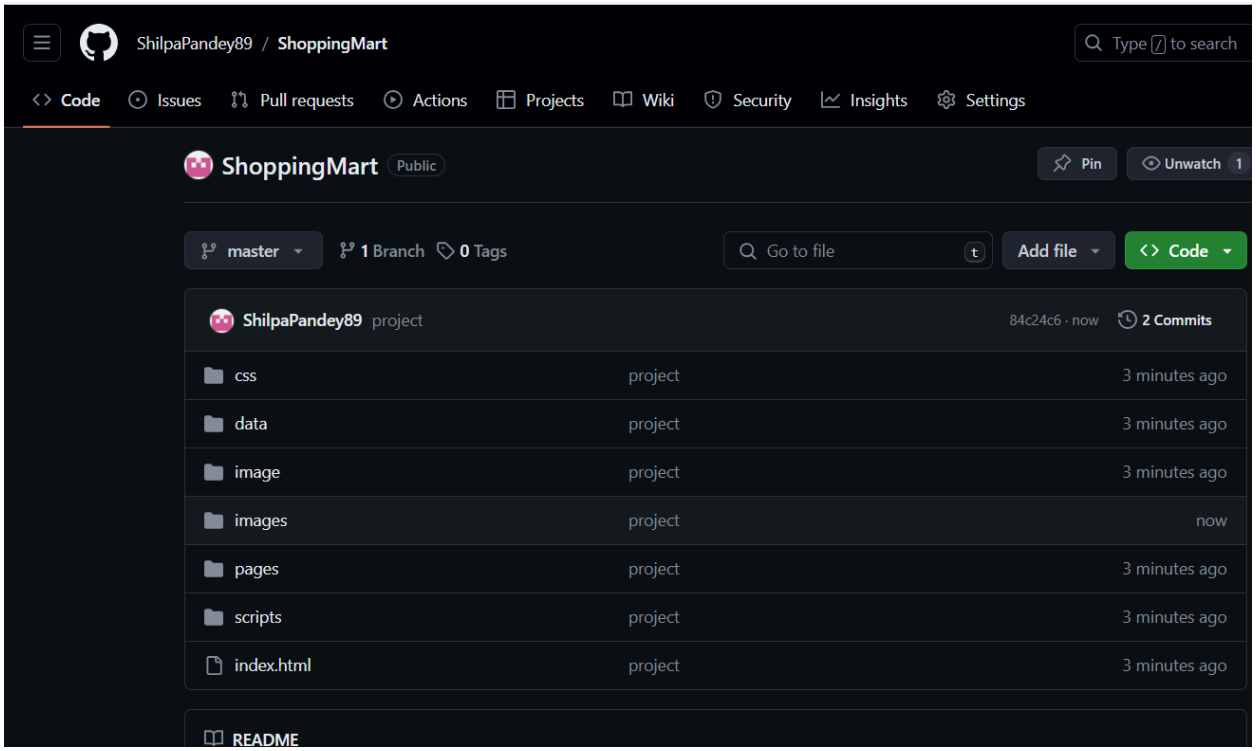
1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5.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

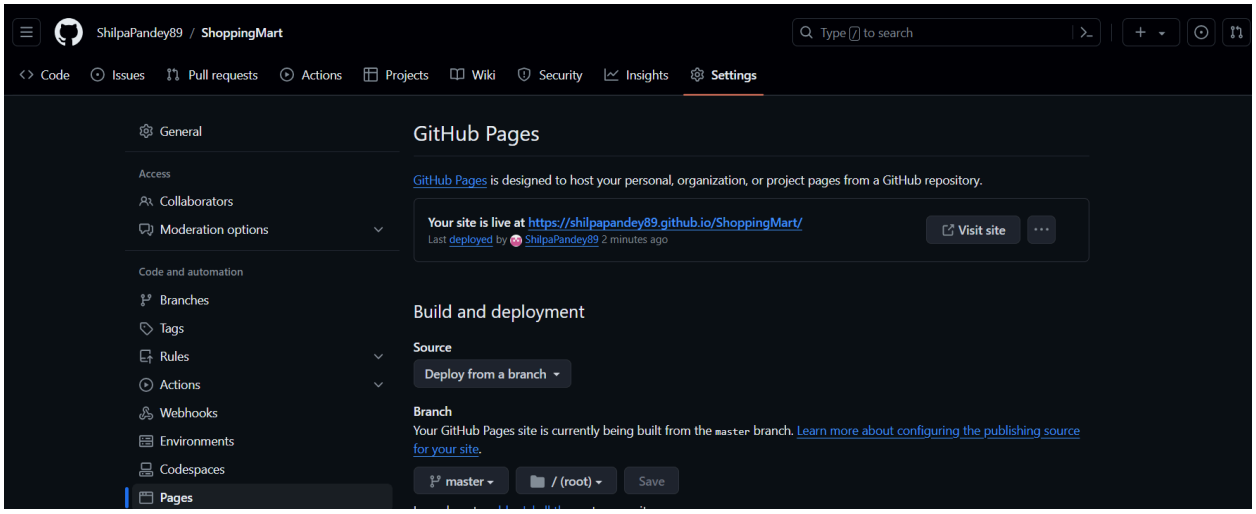
Cons

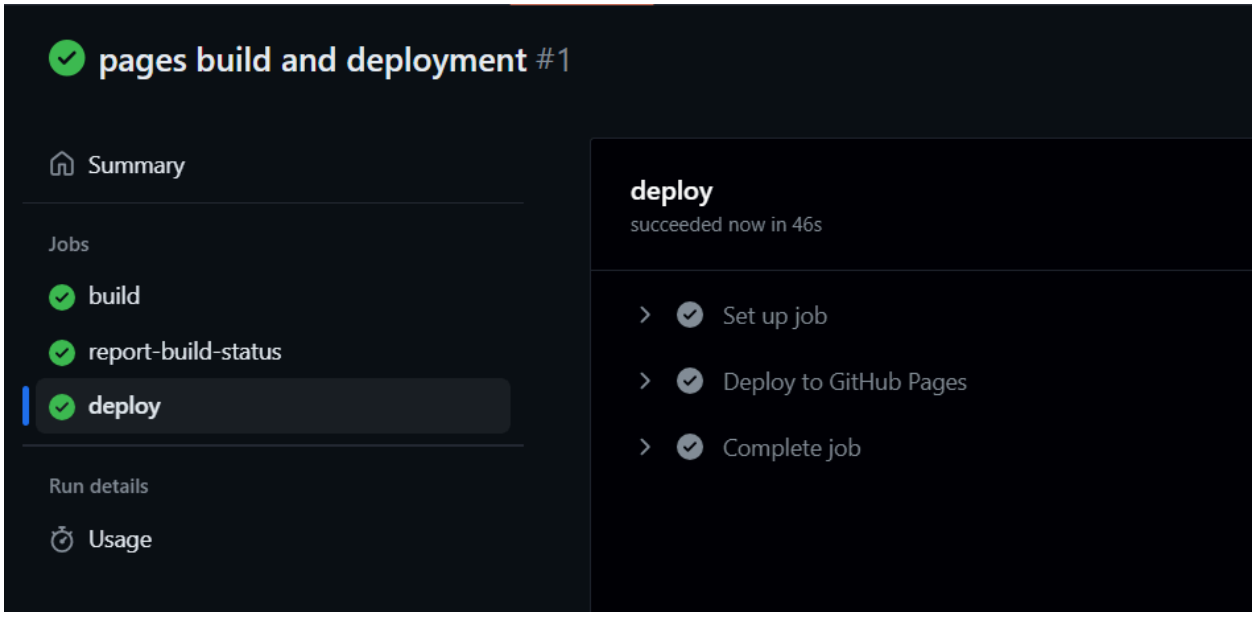
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

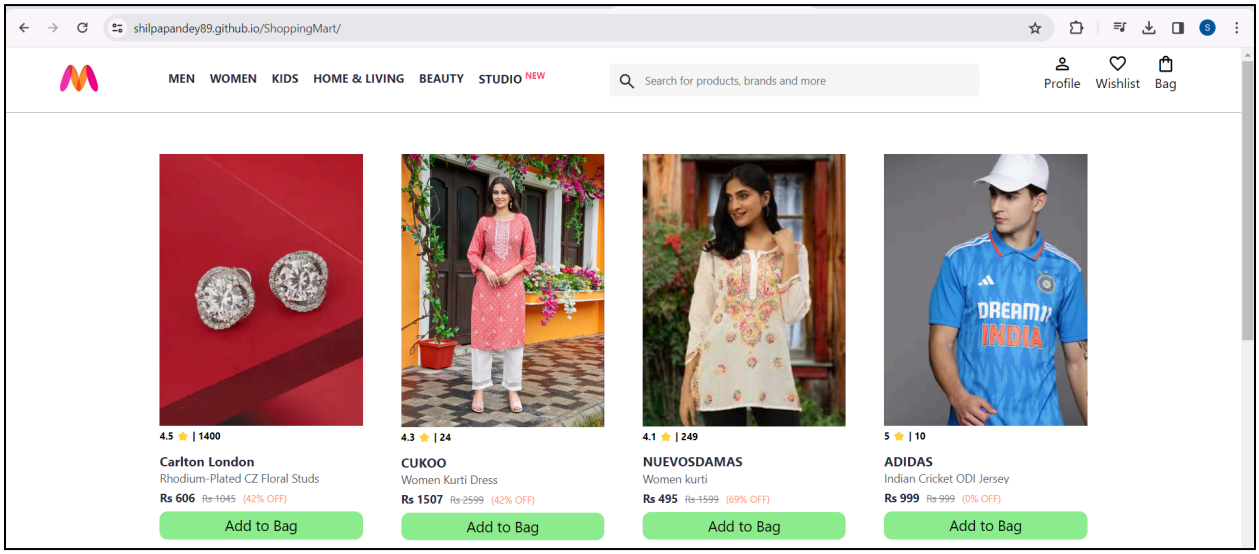
Implementation:

Project Title: Amazon Clone / PWA: Ecommerce Website **Roll No. 42**









Link to GitHub repository:
https://github.com/ShilpaPandey89/ShoppingMart
Hosted Link
https://shilpapandey89.github.io/ShoppingMart/

Conclusion:

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

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15M

Experiment 11

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

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

Google Lighthouse :

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

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

Key Features and Audit Metrics

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

1. **Performance:** This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for

basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th percentile and so on.

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

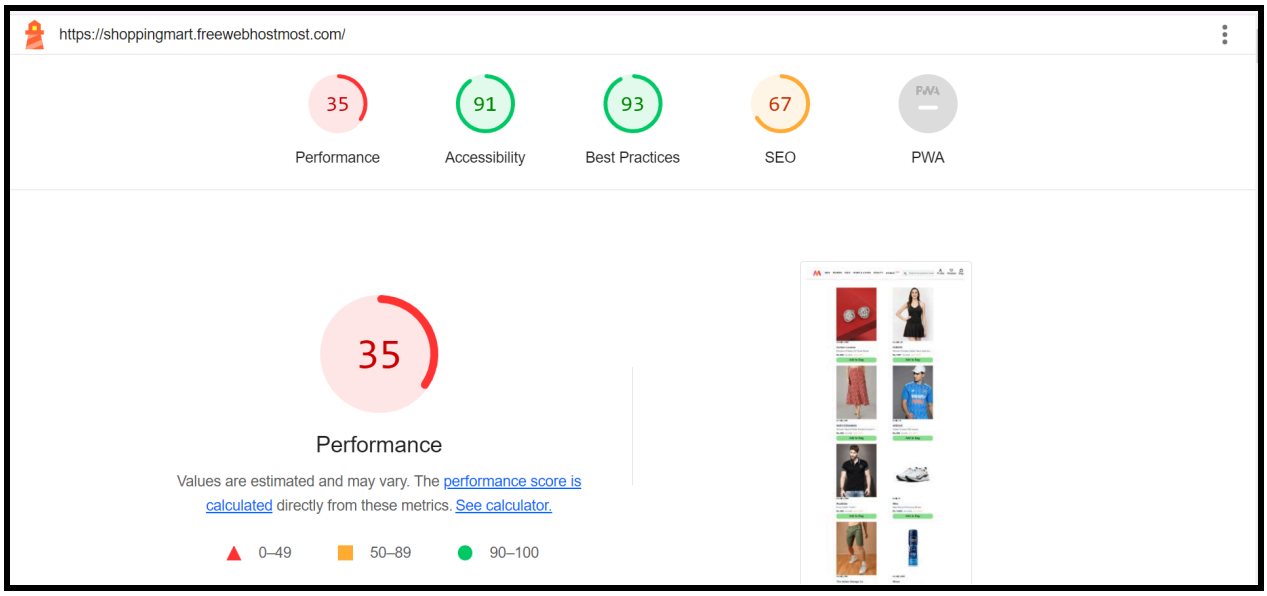
3. **Accessibility:** As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics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

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

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

manifest.json

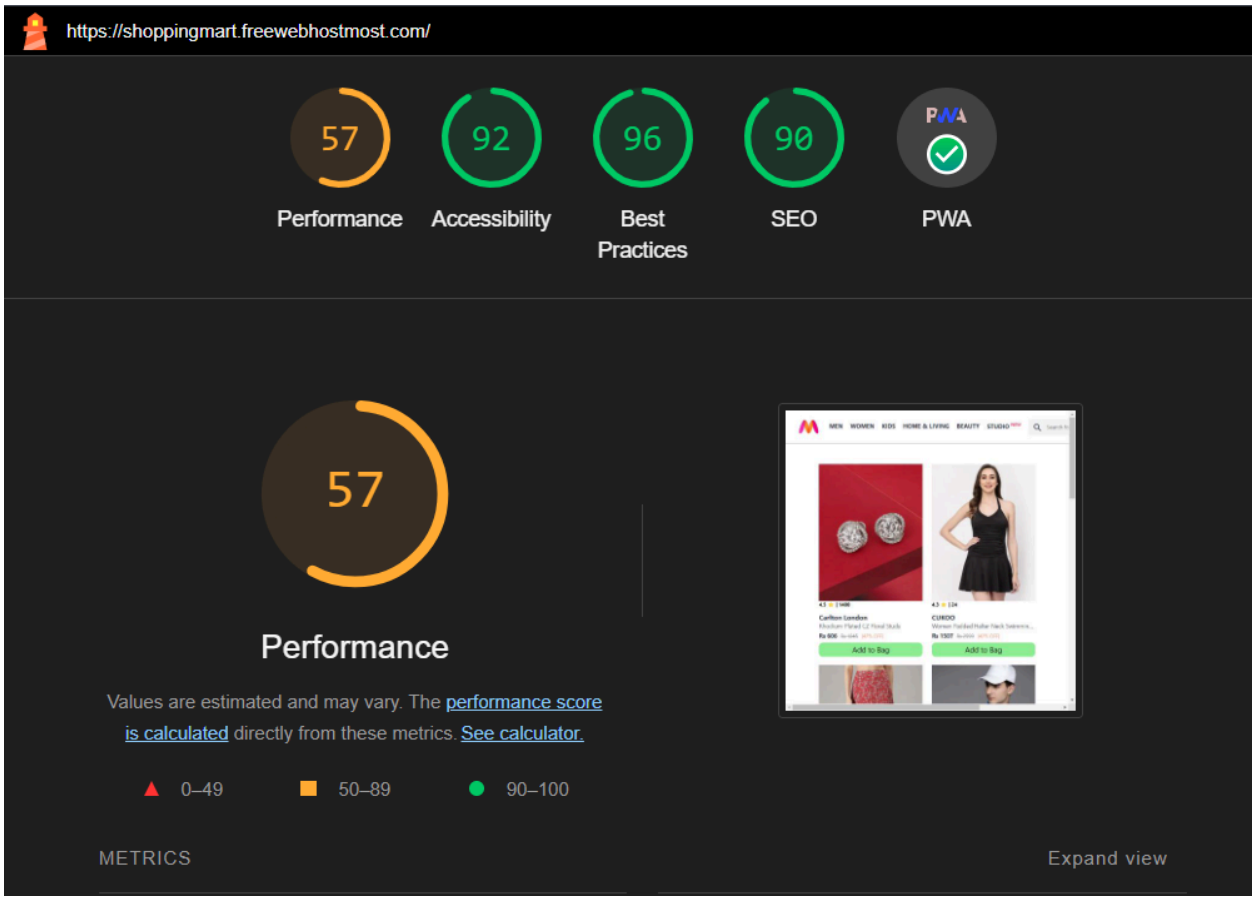
```
{
 "name": "shoppingMart",
 "start_url": "index.html",
 "display": "standalone",
 "background_color": "#5900b3",
 "theme_color": "black",
 "scope": ".",
 "description": "This is an Ecommerce app.",
 "icons": [
   {
    "src": "image/logo.jpg",
    "sizes": "192x192",
    "type": "image/png"
   },
   {
    "src": "image/CompressJPEG.online_512x512_image.jpg",
    "sizes": "512x512",
    "type": "image/png"
   }
 ]
}
```

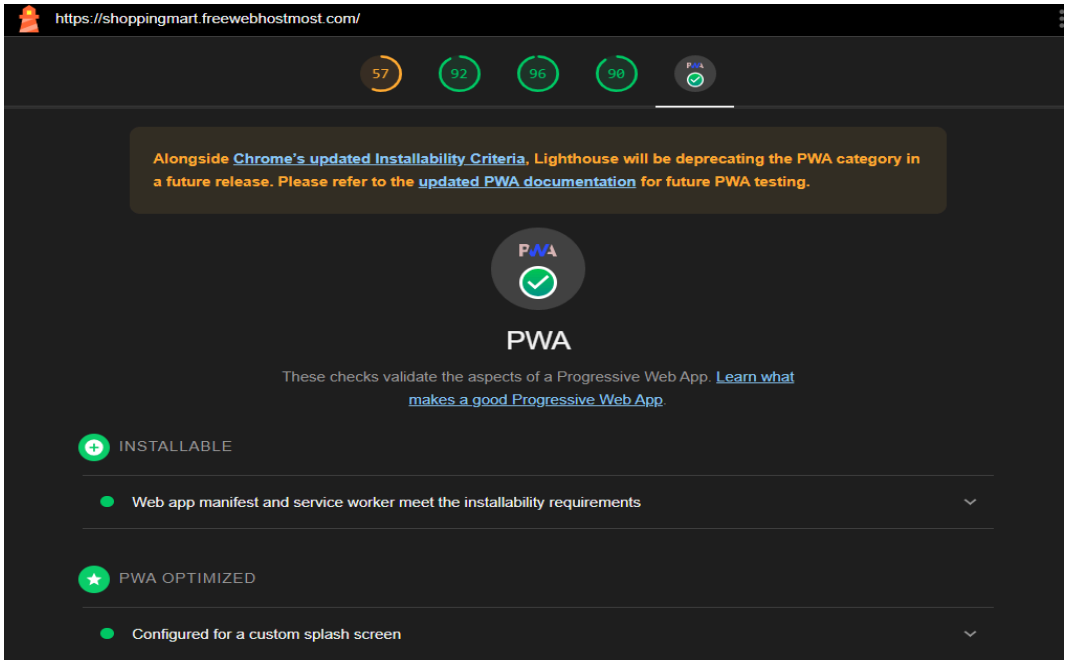


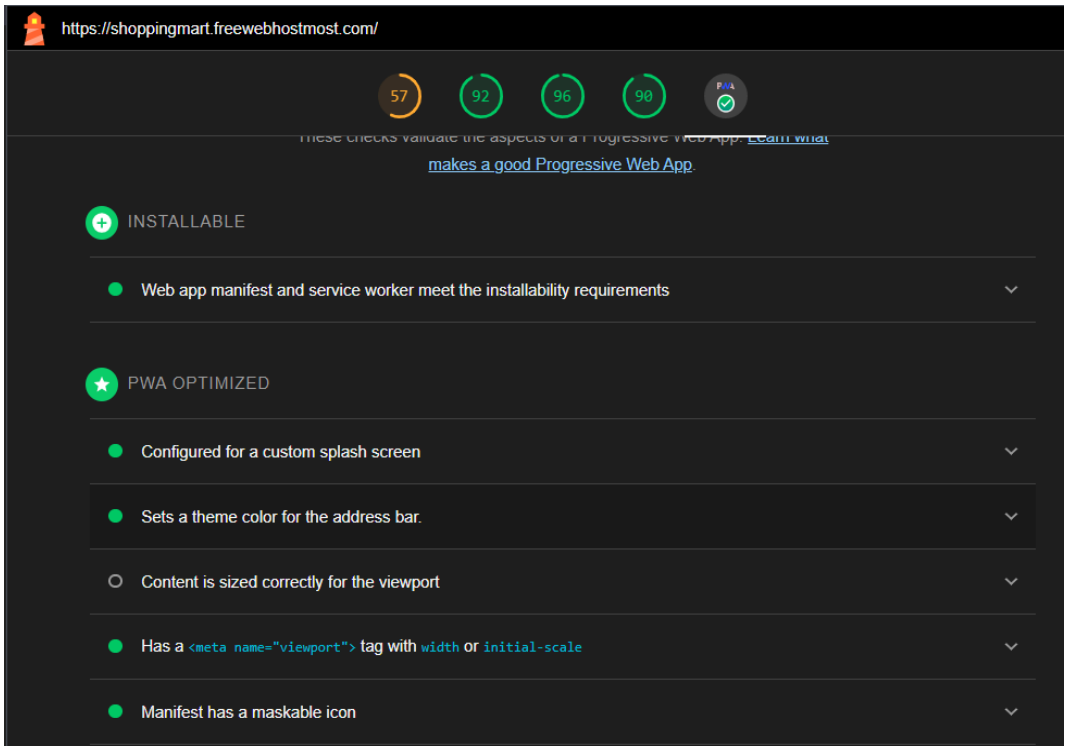
Changes in manifest.json

```
{
 "name": "shoppingMart",
 "start_url": "index.html",
 "display": "standalone",
 "background_color": "#5900b3",
 "theme_color": "black",
 "scope": ".",
 "description": "This is an Ecommerce.",
 "icons": [
  {
   "src": "image/logo.jpg",
   "sizes": "192x192",
   "type": "image/png",
   "purpose": "any maskable"
  },
  {
   "src": "image/CompressJPEG.online_512x512_image.jpg",
   "sizes": "512x512",
   "type": "image/png",
   "purpose": "any maskable"
```

```
    }
  ]
}
```


https://shoppingmart.freewebhostmost.com/
57
Performance
92
Accessibility
96
Best Practices
90
SEO
PWA
PWA
57
Performance
Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.
0–49
50–89
90–100
METRICS
Expand view





Conclusion: Thus we successfully used google Lighthouse PWA Analysis Tool for testing the PWA functioning.

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	5M

Shilpa Pandey
42 D15A

Date
Page

Assignment 1. Flutter

1. 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 UI toolkit developed by Google for building natively compiled applications from mobile, web and desktop from a single codebase.

key features and advantages include :-

1. Single codebase :-
Flutter allows developers to write code and deploy it on both iOS and Android platforms.
2. Hot Reload -
This enables developer to instantly see the result of the code changes, they make.
3. Expressive UI :-
Developers have the flexibility to create expressive and flexible UIs.
4. Integration with other Tools -
Flutter can easily integrate with other popular development tools and frameworks.

Teacher's Sign.:

Advantages OF Flutter:-

1. Faster Development
   uses single codebase for multiple platforms.
2. Consistent UI across Platforms.
   widgets provide a consistent look and feel across different platforms.
3. Cost - Efficiency
   Developing and maintaining single codebase for both iOS and Android reduces development cost and resources.

3) Differ from Traditional Approach:-

(a) Traditional apprach use a hierarchical structure for UI components, whereas Flutter uses a widget-based approach.

(b) Hot Reload allows to see changes made instantly.

Flutter popularity is driven by increased productivity a growing community, flexibility in UI design cross-platform development capabilities and adoption by major companies.

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 example of commonly used widgets and their roles in creating a widget tree.

→ The widget tree represents the hierarchy of UI elements that make up a user interface. widgets are the basic building blocks, and they can be simple, like text or buttons or complex like entire screens.
The widgets tree is structured in a parent-child relationship, forming a tree-like structure.

Widget composition :-

widget composition involves combining and nesting widgets to create more complex UIs.
For example, a 'column' widget can contain multiple 'Text' and 'image' widgets, forming a vertical layout.

Teacher's Sign.: _____

Commonly used widgets.

1. Container -
   A box model for padding, margin and decoration.
2. Column and Row -
   Layout widgets for arranging children vertically or horizontally.
3. Stack -
   overlapping widgets, allowing them to be layered on top of each other.
4. List view -
   A scrollable list of widgets.
5. Grid view -
   A scrollable grid of widgets.
6. AppBar -
   A material design app bar typically at top of screen.
7. TextField -
   An input field for users to enter text.
8. Button widgets -
   Interactive buttons for user actions.

3. State Management in Flutter :-
Discuss the importance of the state management in Flutter application. Compare and contrast the different state management approaches available in Flutter, such as setstate, Provider and Riverpod. Provide scenarios where each approach is suitable.

→ State management is crucial in Flutter to efficiently handle and update the UI based on changes in application data. The choice of state management approach depends on the complexity of the app and its requirement.

setstate :-
- For small to moderately complex applications.
- when managing local state within a widget

Eg :- Simple Form, UI

Provider :-
- For medium to large-sized applications.
- when a centralized state is needed accessible by multiple widgets.

Eg :- Managing user authentication / theme changes or app-wide configuration.

Teacher's Sign.:

③ Riverpod :-
- For large and complex applications
- when testability and maintainability are top priorities.

Eg :- Complex applications with multiple feature dynamic UIs.

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.

→ Integration :-

1. Go to 'Firebase console and create a new project.
2. Add Firebase SDK by including dependencies in pubspec.yaml.

dependencies :-
firebase_core : ^ version
firebase_auth : ^ version
cloud_firebase : ^ version

Teacher's Sign.: _____

3) Run Flutter pub get.
4) Initialise Firebase by calling 'Firebase.initialiseApp() in main.

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

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```

Benefits of using Firebase as Backend

1. Real-time database:-
   - Firebase offers real-time data synchronization with Firebase database allowing seamless updates in your Flutter app.

2. Authentication:-
   Easy integration of Firebase Authentication for secure user management.

3. Cloud Firestore:-
   It notifies clients when data changes allowing for seamless real-time updates on other clients.

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.	42
Name	Shilpa Pandey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	4M

Shilpa Pandey
42 DISA

Assignment 2

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

→ A Progessive Web App (PWA) is a type of web application that utilizes modern web technologies to provide a user experience similar to that native mobile app.
PWAs are designed to be responsive reliable and fast allowing users to access them seamlessly across to access them seamlessly across different devices and network conditions.

Characteristic of PWAs that differentiate PWAs from traditional mobile apps.

1. Cross-platform Compatibility:-
PWAs can run on any device with web browser, including smartphones, tablets and desktops eliminating the need for developing seperate versions for different platforms.

Teacher's Sign :

2. Improved Performance :-
PWA leverage technique like caching and preloading to ensure fast loading times and smooth performance, even on slow or unreliable networks.

3. Offline Functionality :-
PWAs can work limited connectivity, enable offline access to previously visited content.

4. Accessibility -
PWAs are accessible through a URL and can be accessed via any web browser while traditional mobile apps need to be downloaded and installed from an app store.

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.

→ Responsive web design is an approach to building websites that ensures optimal viewing that ensures optimal viewing and interaction across a wide range of devices and screen sizes.

It involves using flexible layouts, fluid grids, and media queries layouts to adapt the layout and design of a website to different viewport sizes, such as those of desktops, laptops tablets and smartphones.

In the context of Progressive Web Apps (PWAs) responsive web design is crucial because PWA aim to provide a seamless user experience across various devices and platforms.
This Flexibility not only enhances user satisfaction but also contributes to the success and adoption of the PWA by catering to a diverse audience.

Responsive web design	Fluid Web design
Dynamic adaptation based on viewport size.	Elements resize proportionally.
Flexible layouts, fluid grids, media queries	Relative unit
Consistent experience across various devices	scalable elements, smooth resizing

Moderate	Moderate.
Adapts to different screen sizes and orientations	Scalable elements, no fixed breakpoints

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

→ The lifecycle of a service worker typically involves three main phases: registration, installation and activation

1. Registration:
The Registration process begins when a Javascript file containing the service workers code is executed by a web browser.
2. This file is typically referenced in the HTML document using the navigator.serviceWorker.register() method.
3. Once registered, the service worker becomes associated with the scope of the document that called registration, meaning it can control the pages within that scope.

2. Installation:-

- After registration, the service worker enters the installation phase.
- During installation, the browser downloads and caches the necessary files specified in the service worker script, such as assets or script needed to run the web application offline.
- If any of the files are successfully cached, the installation is considered complete.

3. Activation:-

- Once installed, the service worker enters the activation phase.
- During activation, the browser checks whether there is an older version of the service worker running.
- After activation, the service worker can intercept network requests, manage caches, and perform other tasks to enable features like offline access and background synchronization.

Throughout its lifecycle, a service worker remains active until it's explicitly unregistered the user clears the browser cache, or the browser decides to terminate it due to memory constraints or other factors.

Teacher's Sign.:

4. Explain the use of IndexedDB in the service worker for data storage.

→ 1. IndexedDB is a powerful browser-based database system that allows web applications to store large amounts of structured data on the client-side.
2. When used within a service worker, IndexedDB enables offline data storage and retrival, providing a robust mechanism for caching data and improving the performance and reliability of web applications, especially in scenarios where network connectivity is unreliable or unavailable.
IndexedDB can be utilized within a service worker for data storage.

1. Offline caching :-
service workers can intercept network requests and cache response, include data fetched from APIs or other web resources.

2. Data synchronization :-
IndexedDB allows service workers to synchronize data between the client and server when the network connection is restored.

Teacher's Sign.:

3. Improved Performance :-
By storing frequently accessed data locally using IndexedDB, service workers can reduce the need for repeated network requests, resulting in faster load times and improved performance for the web application.

4. Rich Querying Capabilities :-
IndexedDB provides powerful querying capabilities, allowing service workers to efficiently retrieve and manipulate data based on various criteria, such as key ranges, indexes and object stores. This flexibility enables developers to build sophisticated data storage and retrieval logic within their web applications.

5. Asynchronous operations :-
IndexedDB operations are performed asynchronously, which ensures that data storage and retrival operations do not block the main thread, thereby preventing UI freezes and providing a smooth user experience.