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

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Project Title: Flutter-Amazon Clone/PWA-Wildlife Protection **Roll No. 13**

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

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

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers : Mrs. Kajal Jewani.

Email : kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: Flutter-Amazon Clone/PWA-Wildlife Protection **Roll No. 13**

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	16/1	23/1	14
2.	To design Flutter UI by including common widgets.	LO2	23/1	30/1	14
3.	To include icons, images, fonts in Flutter app	LO2	30/1	6/2	14
4.	To create an interactive Form using form widget	LO2	6/2	13/2	14
5.	To apply navigation, routing and gestures in Flutter App	LO2	13/2	20/2	14
6.	To Connect Flutter UI with fireBase database	LO3	20/2	5/3	14
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4	5/3	12/3	15
8.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA	LO5	12/3	19/3	15
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	19/3	26/3	15
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	26/3	2/4	15
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	5/3	12/3	15
12.	Assignment-1	LO1,LO2 ,LO3	2/2	5/2	5
13.	Assignment-2	LO4,LO5 ,LO6	19/3	21/3	4

MAD & PWA Lab
Journal

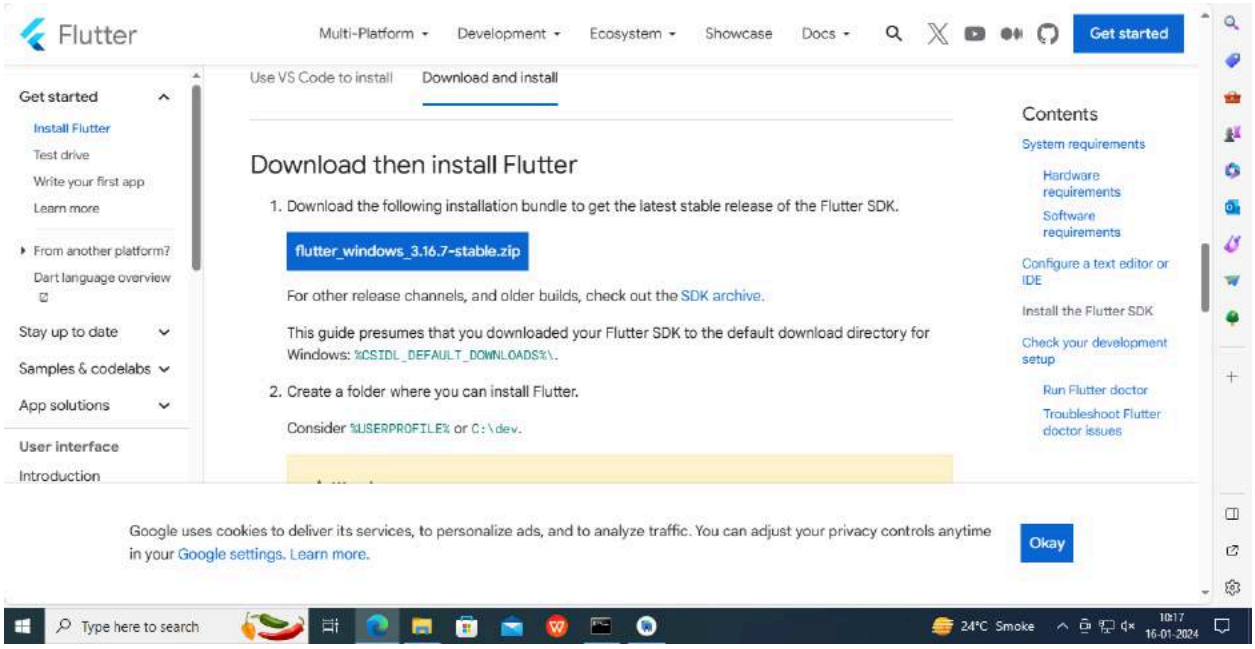
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	13
Name	Swarali Dhobale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	14

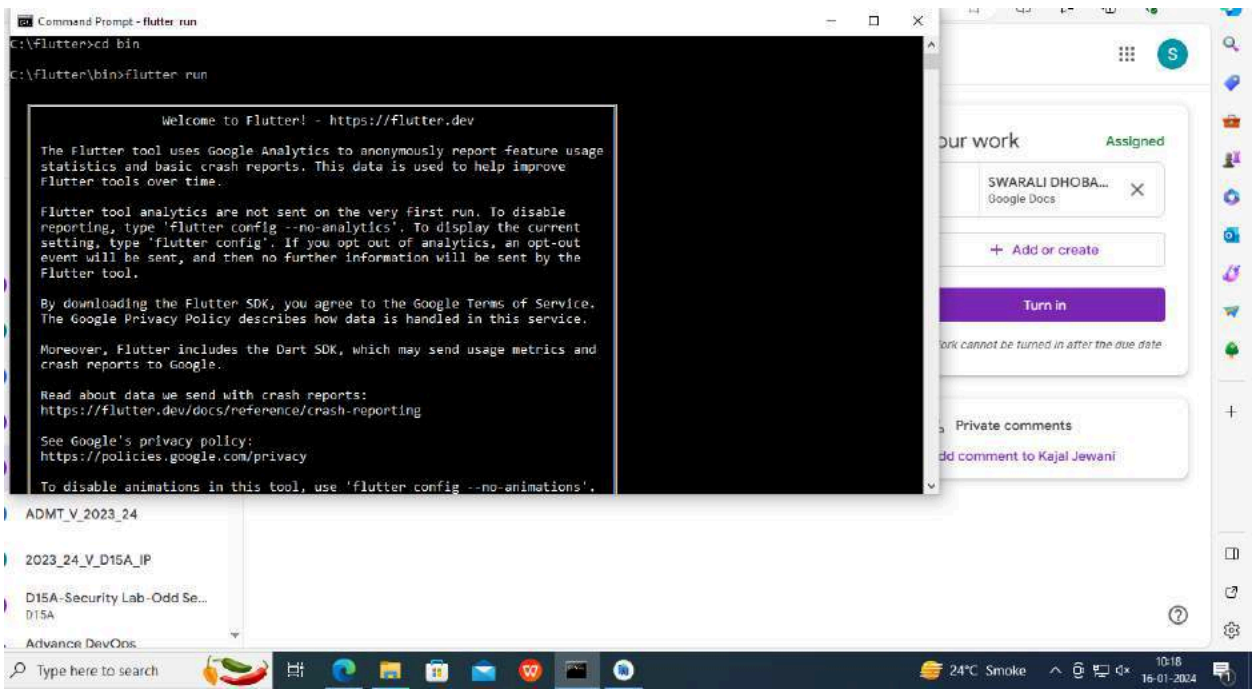
MPL Lab: Experiment No.1

Name: Swarali_Dhobale_D15A_13

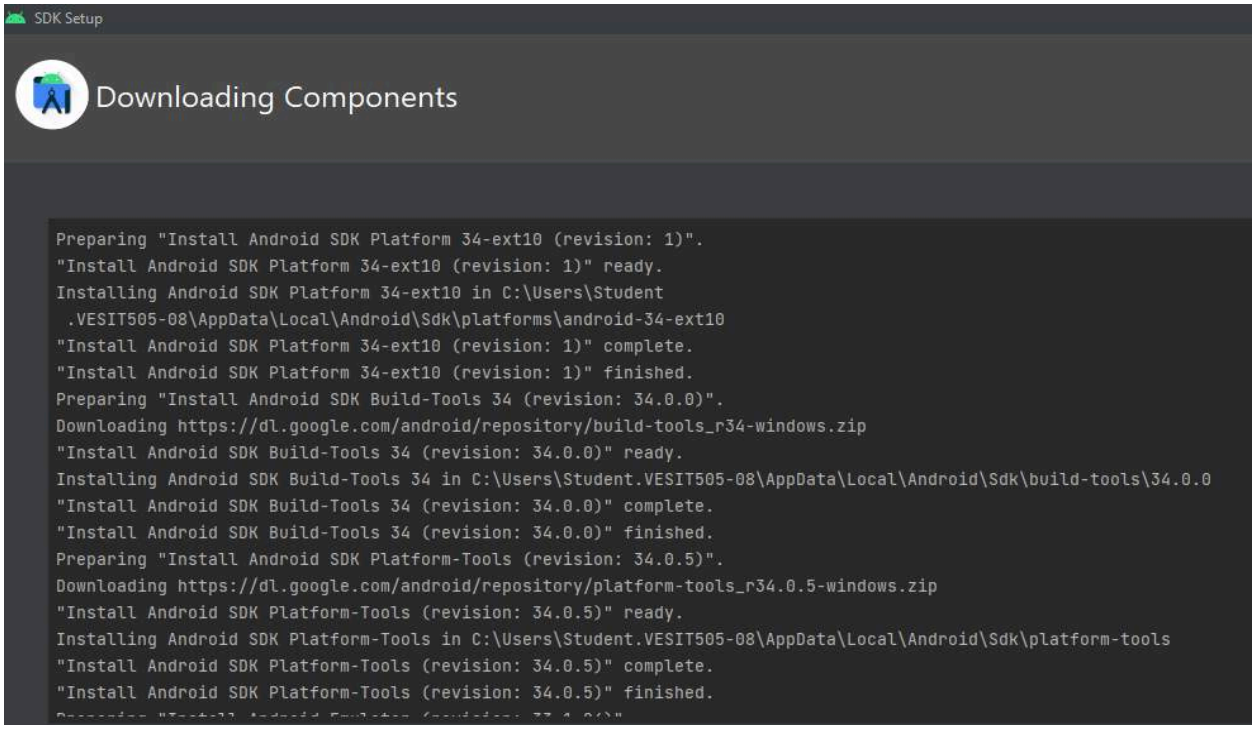
Aim: Installation and Configuration of Flutter Environment.

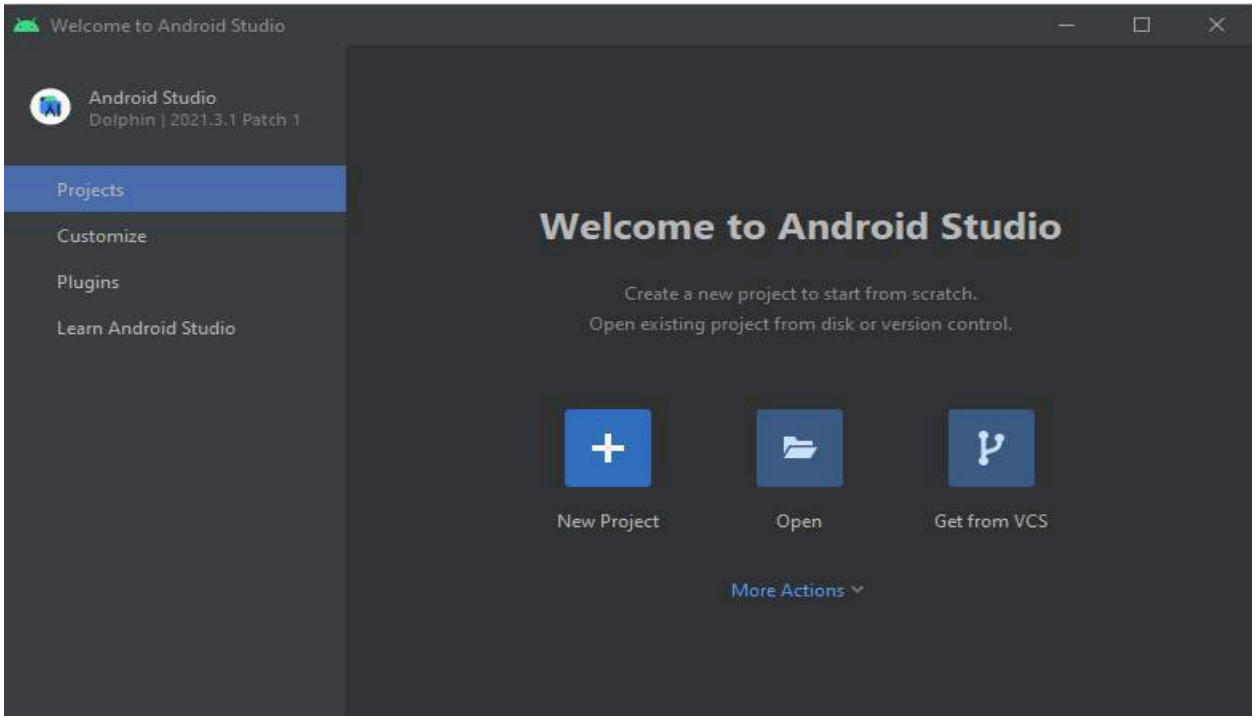
Flutter download and install:



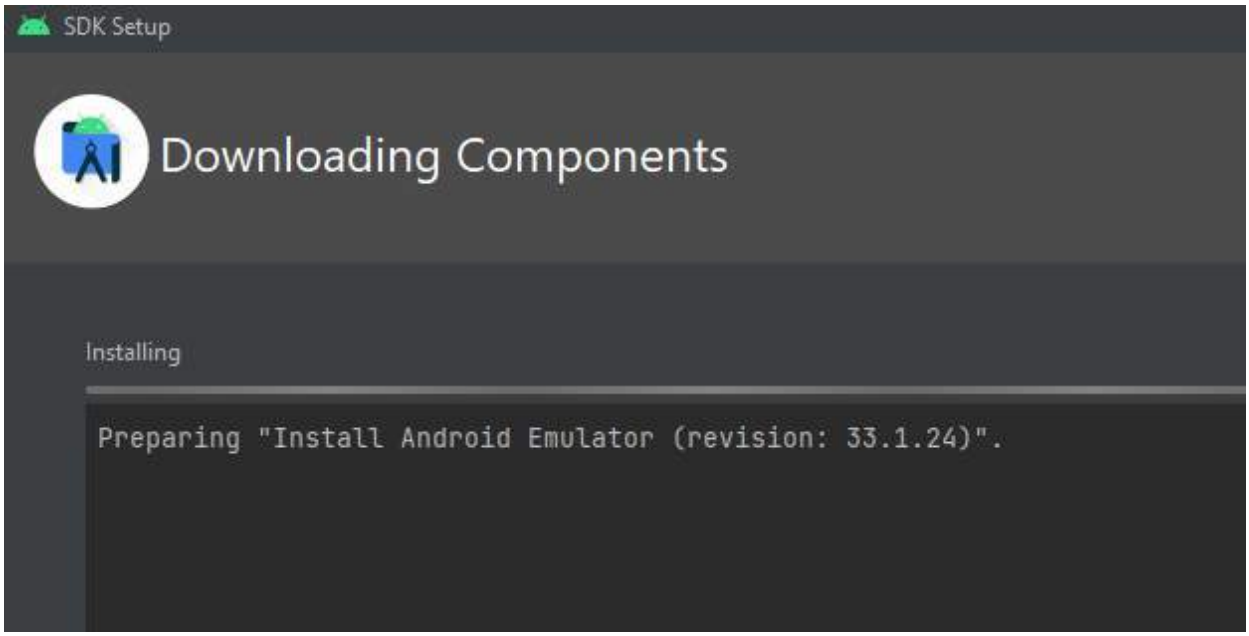


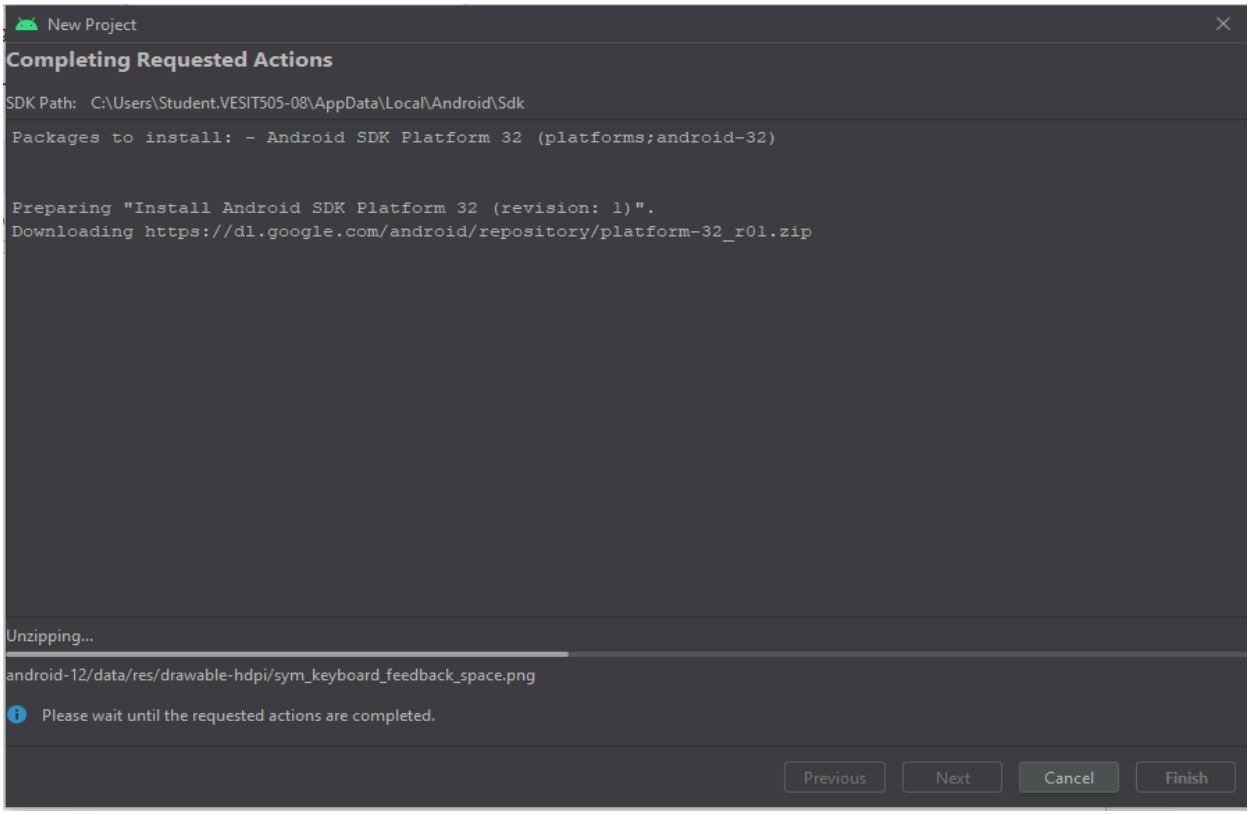
Android Studio installation:





Installation for Android Emulator:


SDK Setup
Downloading Components
Installing
Preparing "Install Android Emulator (revision: 33.1.24)".

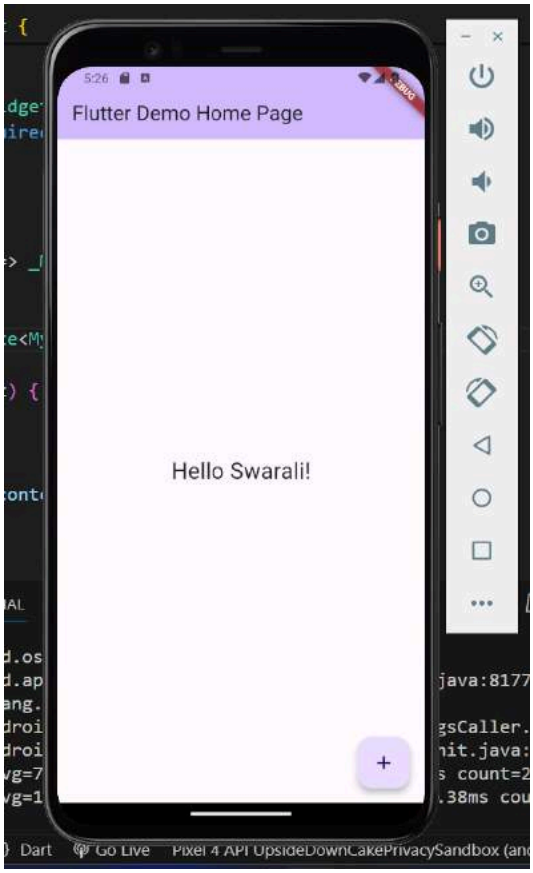

New Project
Completing Requested Actions
SDK Path: C:\Users\Student.VESIT505-08\AppData\Local\Android\Sdk
Packages to install: - Android SDK Platform 32 (platforms;android-32)
Preparing "Install Android SDK Platform 32 (revision: 1)".
Downloading https://dl.google.com/android/repository/platform-32_r01.zip
Unzipping...
android-12/data/res/drawable-hdpi/sym_keyboard_feedback_space.png
Please wait until the requested actions are completed.
Previous
Next
Cancel
Finish

Code:

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(

        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  @override
```

```
Widget build(BuildContext context) {

  return Scaffold(
    appBar: AppBar(
      backgroundColor: Theme.of(context).colorScheme.inversePrimary,
      title: Text(widget.title),
    ),
    body: Center(
      child: Column(

        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          const Text(
                    'Hello Swarali!',
                  style: TextStyle(
                  fontSize: 24,
                  ),
          ),
    ],
      ),
    ),
  );
 }
}
```

Output:



Conclusion: Flutter and Android Studio have been installed and configured.

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	13
Name	Swarali Dhobale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	14

MAD LAB EXPT 2

Name:Swarali Dhobale_D15A_13

Aim: To design Flutter UI by including common widgets.

Theory:

Flutter employs a reactive framework, allowing for fast development and expressive, flexible UI designs. Central to Flutter are widgets, which are the building blocks used to construct user interfaces.

Container: A versatile widget used to contain other widgets. It allows you to customize properties such as alignment, padding, margin, color, and more.

Row and Column: Widgets used to arrange child widgets horizontally (Row) or vertically (Column). They automatically size and position their children according to their properties.

Text: Widget used to display text with styling options like font size, color, weight, and alignment.

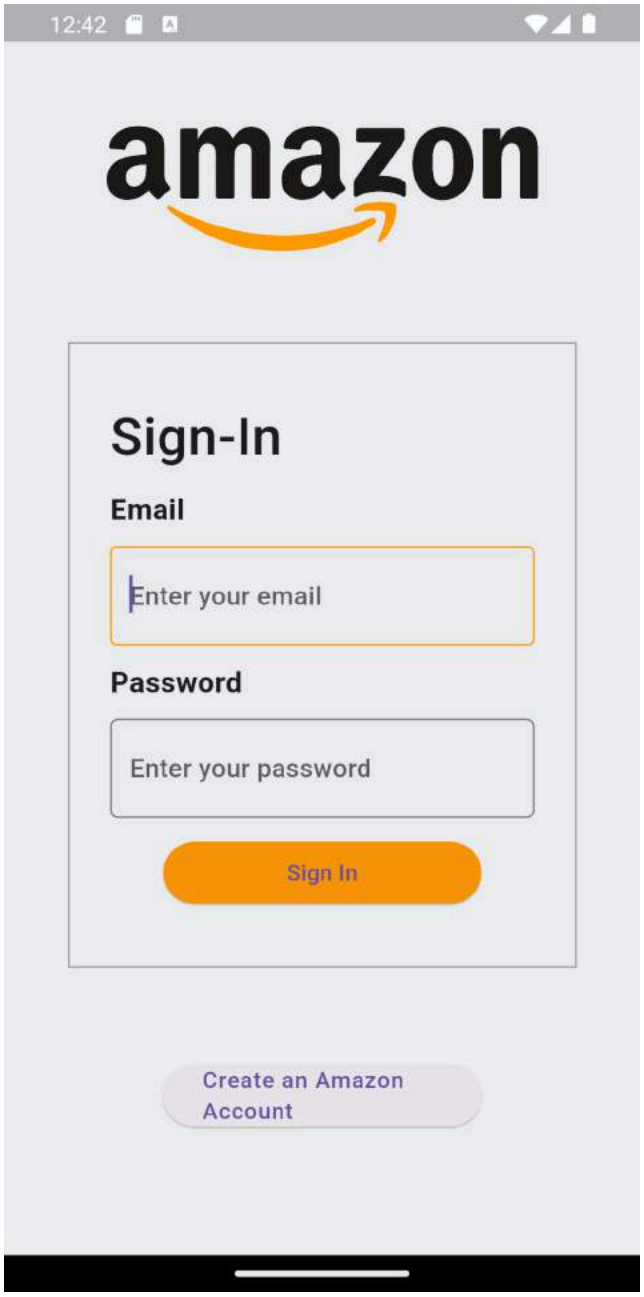
Image: Widget used to display images from various sources such as assets, network, or memory. It supports various image formats and provides options for resizing and scaling.

Icon: Widget used to display icons from the Material Icons or custom icon sets.

Output:

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

class CustomMainButton extends StatelessWidget {
  final Widget child;
  final Color color;
  final bool isLoading;
  final VoidCallback onPressed;
  const CustomMainButton({
    Key? key,
    required this.child,
```

```
    required this.color,
    required this.isLoading,
    required this.onPressed,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    Size screenSize = Utils().getScreenSize();
    return ElevatedButton(
      style: ElevatedButton.styleFrom(
          primary: color,
          fixedSize: Size(
            screenSize.width * 0.5,
            40,
          )),
      onPressed: onPressed,
      child: !isLoading
          ? child
          : const Padding(
              padding: EdgeInsets.symmetric(vertical: 5),
              child: AspectRatio(
                aspectRatio: 1 / 1,
                child: CircularProgressIndicator(
                  color: Colors.white,
                ),
              ),
            ),
    );
  }
}
```



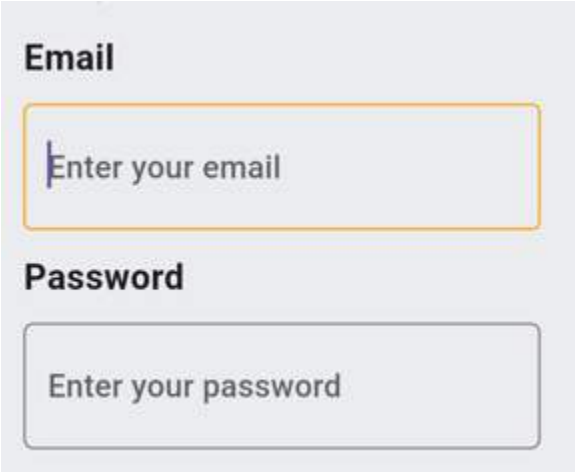
Text field Widget-

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

class TextFieldWidget extends StatefulWidget {
```

```
  final String title;
  final TextEditingController controller;
  final bool obscureText;
  final String hintText;
  const TextFieldWidget({
    Key? key,
    required this.title,
    required this.controller,
    required this.obscureText,
    required this.hintText,
  }) : super(key: key);

  @override
  State<TextFieldWidget> createState() => _TextFieldWidgetState();
}

class _TextFieldWidgetState extends State<TextFieldWidget> {
  late FocusNode focusNode;
  bool isInFocus = false;

  @override
  void initState() {
    super.initState();
    focusNode = FocusNode();

    focusNode.addListener(() {
      if (focusNode.hasFocus) {
        setState(() {
          isInFocus = true;
        });
      } else {
        setState(() {
          isInFocus = false;
        });
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisSize: MainAxisSize.min,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
```

```
Padding(
  padding: const EdgeInsets.only(bottom: 15),
  child: Text(
    widget.title,
    style: const TextStyle(
      fontWeight: FontWeight.bold,
      fontSize: 17,
    ),
  ),
),
Container(
  decoration: BoxDecoration(boxShadow: [
    isInFocus
        ? BoxShadow(
            color: Colors.orange.withOpacity(0.4),
            blurRadius: 8,
            spreadRadius: 2,
          )
        : BoxShadow(
            color: Colors.black.withOpacity(0.2),
            blurRadius: 8,
            spreadRadius: 2,
          )
  ]),
  child: TextField(
    focusNode: focusNode,
    obscureText: widget.obscureText,
    controller: widget.controller,
    maxLines: 1,
    decoration: InputDecoration(
      fillColor: Colors.white,
      filled: true,
      hintText: widget.hintText,
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(3),
        borderSide: const BorderSide(
          color: Colors.grey,
          width: 1,
        ),
      ),
      focusedBorder: const OutlineInputBorder(
        borderSide: BorderSide(
          color: Colors.orange,
          width: 1,
```

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



Conclusion: Common widgets for text and button have been implemented for Flutter application.

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	13
Name	Swarali Dhobale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	14

MAD LAB EXPT 3

Name:Swarali Dhobale_D15A_13

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

Theory:

1.Icons:
Flutter provides the Icon **widget to display icons. Icons in Flutter are based on the material design system, and you can use icons from the Material Icons library or customize your icons.**
Icon(Icons.favorite);

2.Images:
Flutter supports various image formats like JPEG, PNG, GIF, WebP, BMP, and animated WebP/GIF. You can display images from assets or network URLs.
To display an image from assets:
Image.asset('assets/images/my_image.png');
Image.network('http://example.com/my_image.png');

3.Fonts:
You can include custom fonts in your Flutter app to use different typography styles. Flutter supports TrueType (.ttf**) and OpenType (**.otf**) font formats.**

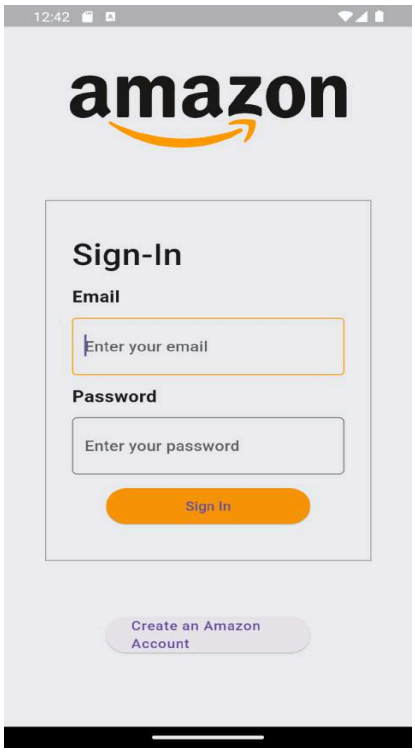
```
Text(
  'Hello, World!',
  style: TextStyle(
    fontFamily: 'MyCustomFont',
    fontSize: 16.0,
  ),
);
```

Output:

```
Widget build(BuildContext context) {
    Size screenSize = Utils().getScreenSize();
    return Scaffold(
      backgroundColor: Colors.white,
      body: SingleChildScrollView(
        child: SizedBox(
          height: screenSize.height,
```

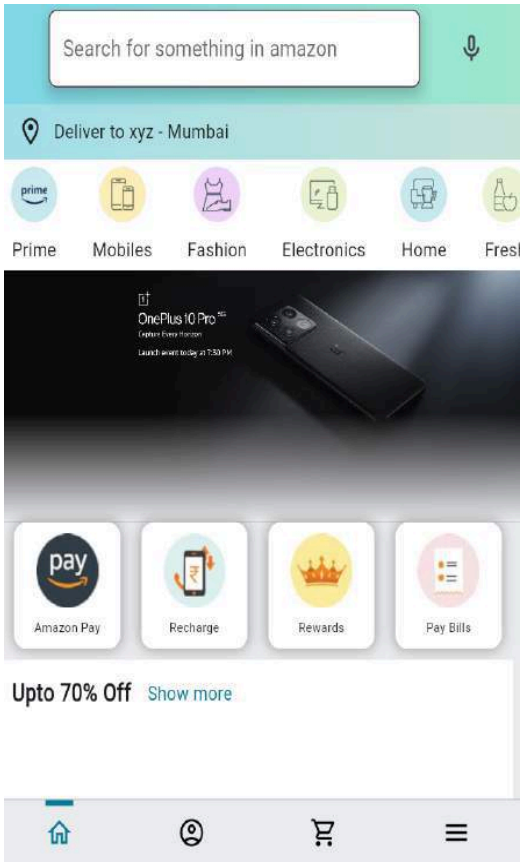
```
width: screenSize.width,
child: Padding(
  padding: const EdgeInsets.symmetric(horizontal: 10, vertical: 20),
  child: Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      crossAxisAlignment: CrossAxisAlignment.center,
      children: [
        Image.network(
          amazonLogo,
          height: screenSize.height * 0.10,
        ),
        SizedBox(
          height: screenSize.height * 0.7,
          child: FittedBox(
            child: Container(
              height: screenSize.height * 0.85,
              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-Up",
                    style: TextStyle(
                        fontWeight: FontWeight.w500, fontSize: 33),
                  ),
                  TextFieldWidget(
                    title: "Name",
                    controller: nameController,
                    obscureText: false,
                    hintText: "Enter your name",
                  ),
                  TextFieldWidget(
                    title: "Address",
                    controller: addressController,
```

```
                obscureText: false,
                hintText: "Enter your address",


              ),
```




```
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,
            ),
          ),
        ),
      ),
    ],
  ),
```



Conclusion: Added icons,images,fonts in Flutter application.

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	13
Name	Swarali Dhobale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	14

MAD LAB EXPT 4

Name:Swarali Dhobale_D15A_13

Aim: To make an interactive form using Widgets.

Theory:

Text Input Fields- Use the TextField widget to allow users to input text. You can customize it with properties like decoration, controller, keyboardType, validator, and onChanged callback.

Form Widget- Encapsulate your input fields within a Form widget. The Form widget manages the form state and provides methods for validation and submission.

Form Fields- Wrap each input field within a TextFormField widget. This widget integrates with the Form widget and automatically handles validation.

Validation-Implement validation logic to ensure that the user input meets certain criteria (e.g., required fields, valid email format). You can use the validator property of the TextFormField widget to specify validation functions.

State Management- Maintain the form's state using either StatefulWidget or state management solutions like Provider, Riverpod, or Bloc. When the user interacts with the form (e.g., typing in a text field), update the corresponding state.

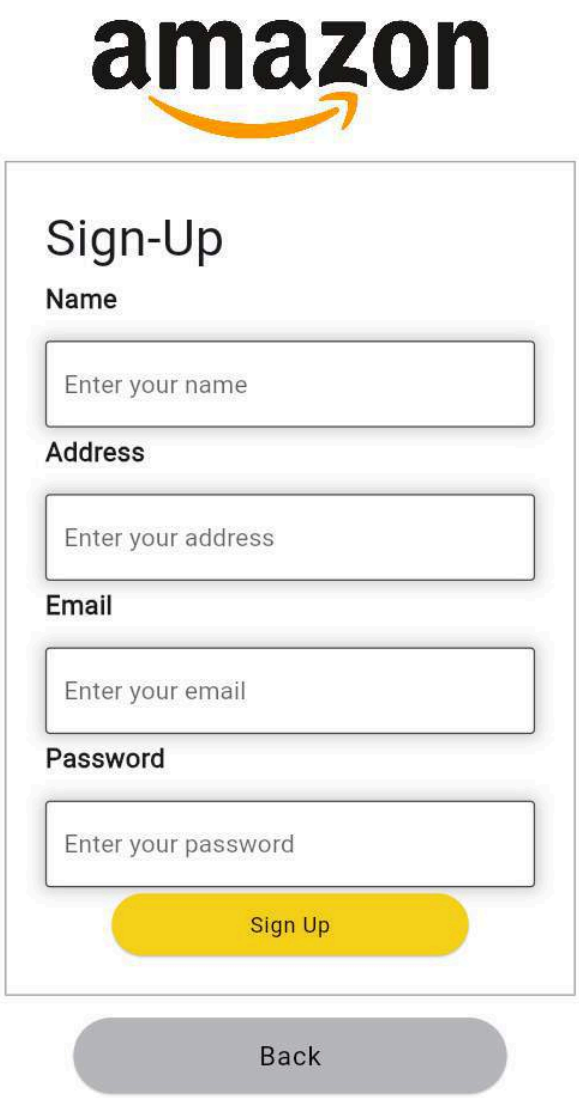
Submit Button- Include a button (e.g., ElevatedButton or TextButton) to allow users to submit the form. Disable the button if the form is invalid.

Handling Form Submission-Define a function to handle form submission. This function should be called when the user taps the submit button. You can access the form data using the FormState object.

Output:

```
import 'dart:math';
import 'dart:typed_data';

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

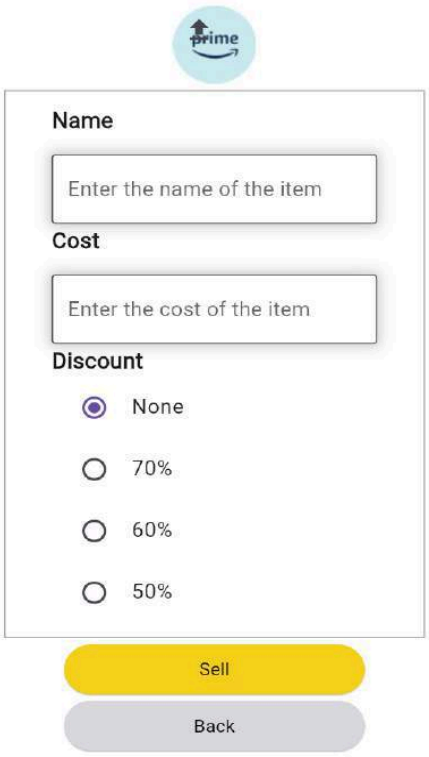
```
class Utils {
  Size getScreenSize() {
    return MediaQueryData.fromWindow(WidgetsBinding.instance!.window).size;
  }

  showSnackBar({required BuildContext context, required String content}) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        backgroundColor: Colors.orange,
        shape: const RoundedRectangleBorder(
          borderRadius: BorderRadius.only(
            topLeft: Radius.circular(10),
            topRight: Radius.circular(10),
          ),
        ),
        content: SizedBox(
          width: getScreenSize().width,
          child: Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text(
                content,
                maxLines: 2,
                overflow: TextOverflow.ellipsis,
              ),
            ],
          ),
        ),
      ),
    );
  }

  Future<Uint8List?> pickImage() async {
    ImagePicker picker = ImagePicker();
    XFile? file = await picker.pickImage(source: ImageSource.gallery);
    return file!.readAsBytes();
  }

  String getUid() {
    return (100000 + Random().nextInt(10000)).toString();
  }
}
```




```
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: An interactive form using Widgets has been created in Flutter application.

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	13
Name	Swarali Dhobale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	14

MAD LAB EXPT 5

Name:Swarali Dhobale_D15A_13

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

Theory:

In Flutter, navigation, routing, and gestures are essential concepts for creating interactive and navigable user interfaces.

Navigation: Navigation refers to the process of moving between different screens or pages within a Flutter app. Flutter provides the Navigator widget for managing navigation and routing.

Routing:Routing is the mechanism used to define the paths or routes between different screens in your app. Each route typically corresponds to a different widget or screen in your app.

Gesture Detection:Gestures allow users to interact with the app by tapping, dragging, swiping, or performing other touch-based actions. Flutter provides various gesture detection widgets to handle user input.

```
GestureDetector(
  onTap: () {
    print('Container tapped');
  },
  child: Container(
    width: 200,
    height: 200,
    color: Colors.blue,
    child: Center(
      child: Text('Tap Me'),
    ),
  ),
)
```

Output:

```
import 'package:amazon_clone/resources/authentication_methods.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';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.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();
  AuthenticationMethods authenticationMethods = AuthenticationMethods();
  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;
            });

            String output =
                await authenticationMethods.signInUser(
                    email: emailController.text,
                    password: passwordController.text);
            setState(() {
              isLoading = false;
            });
            if (output == "success") {
              //functions
            } else {
              //error
              Utils().showSnackBar(
                  context: context, content: output);
            }
          }),
    )
  ],
 ),
),
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();
                  }));
                })
          ],
        ),
      ),
    ),
  ),
 );
 }
}
```

```
class UserDetailsBar extends StatelessWidget {
  final double offset;
  const UserDetailsBar({
    Key? key,
    required this.offset,
  }) : super(key: key);
```

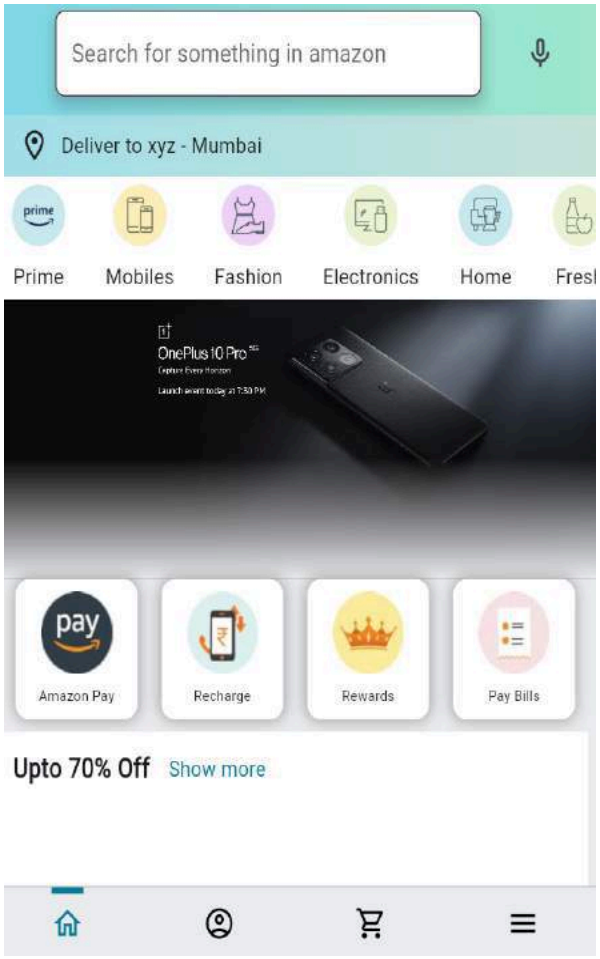
```
@override
Widget build(BuildContext context) {
  Size screenSize = Utils().getScreenSize();
  UserDetailsModel userDetails =
      Provider.of<UserDetailsProvider>(context).userDetails;
  return Positioned(
    top: -offset / 3,
    child: Container(
      height: kAppBarHeight / 2,
      width: screenSize.width,
      decoration: const BoxDecoration(
        gradient: LinearGradient(
          colors: lightBackgroundaGradient,
          begin: Alignment.centerLeft,
          end: Alignment.centerRight,
        ),
      ),
      child: Padding(
        padding: const EdgeInsets.symmetric(
          vertical: 3,
          horizontal: 20,
        ),
        child: Row(
          children: [
            Padding(
              padding: const EdgeInsets.only(right: 8.0),
              child: Icon(
                Icons.location_on_outlined,
                color: Colors.grey[900],
              ),
            ),
            SizedBox(
              width: screenSize.width * 0.7,
              child: Text(
                "Deliver to ${userDetails.name} - ${userDetails.address} ",
                maxLines: 1,
                overflow: TextOverflow.ellipsis,
                style: TextStyle(
                  color: Colors.grey[900],
                ),
              ),
            )
          ],
```

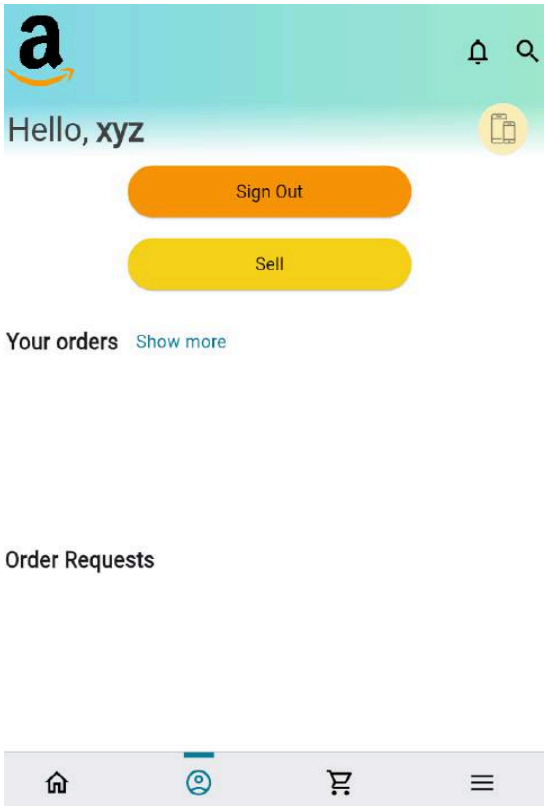
```
          ),
        ),
      ),
    );
  }
}
```

SIGN OUT-

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

  @override
  State<AccountScreen> createState() => _AccountScreenState();
}

class _AccountScreenState extends State<AccountScreen> {
  @override
  Widget build(BuildContext context) {
    Size screenSize = Utils().getScreenSize();

    return Scaffold(
      backgroundColor: Colors.white,
      appBar: AccountScreenAppBar(),
      body: SingleChildScrollView(
        child: SizedBox(
          height: screenSize.height,
          width: screenSize.width,
          child: Column(
            children: [
              IntroductionWidgetAccountScreen(),
              Padding(
                padding: const EdgeInsets.all(8.0),
                child: CustomMainButton(
                    child: const Text(
                      "Sign Out",
                      style: TextStyle(color: Colors.black),
                    ),
                    color: Colors.orange,
                    isLoading: false,
                    onPressed: () {
                      FirebaseAuth.instance.signOut();
                    }),
              ),
```

```
Padding(
  padding: const EdgeInsets.all(8.0),
  child: CustomMainButton(
      child: const Text("Sell",
          style: TextStyle(color: Colors.black)),
      color: yellowColor,
      isLoading: false,
      onPressed: () {
        Navigator.push(
            context,
            MaterialPageRoute(
                builder: (context) => const SellScreen()));
      }),
),
FutureBuilder(
    future: FirebaseFirestore.instance
        .collection("users")
        .doc(FirebaseAuth.instance.currentUser!.uid)
        .collection("orders")
        .get(),
    builder: (context,
        AsyncSnapshot<QuerySnapshot<Map<String, dynamic>>>
            snapshot) {
      if (snapshot.connectionState == ConnectionState.waiting) {
        return Container();
      } else {
        List<Widget> children = [];
        for (int i = 0; i < snapshot.data!.docs.length; i++) {
          ProductModel model = ProductModel.getModelFromJson(
              json: snapshot.data!.docs[i].data());
          children.add(SimpleProductWidget(productModel: model));
        }
        return ProductsShowcaseListView(
            title: "Your orders", children: children);
      }
    }),
```


Search for something in amazon
Deliver to xyz - Mumbai
Prime
Mobiles
Fashion
Electronics
Home
Fresl
OnePlus 10 Pro 5G
Amazon Pay
Recharge
Rewards
Pay Bills
Upto 70% Off
Show more



Conclusion: Navigation,routing and gestures have been applied in Flutter Application.

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	13
Name	Swarali Dhobale
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:	14

MAD LAB EXPT 6

Name:Swarali Dhobale_D15A_13

Aim: To connect Flutter UI with Firebase.

Theory:

Step1:
Go to the Firebase Console (https://console.firebase.google.com/) and create a new project.Follow the instructions to add your app to the Firebase project. You'll need to provide your app's package name (Android) or bundle identifier (iOS).

Step 2:
Add the Firebase SDK dependencies to your Flutter app's pubspec.yaml file. These dependencies vary depending on the Firebase services you want to use (e.g., Firebase Authentication, Firestore, Realtime Database, Cloud Storage).
Run flutter pub get to install the dependencies.

Step 3:
In your Flutter app, initialize Firebase by calling Firebase.initializeApp() in the main() function or at the entry point of your app.
This initialization step is crucial and should be done before accessing any Firebase services.

Step 4:
Once Firebase is initialized, you can start using Firebase services like Firestore (NoSQL database), Realtime Database (JSON database), Cloud Storage (file storage), Cloud Functions (serverless functions), etc.
You'll typically use Firebase APIs to read and write data, handle user authentication, and perform other tasks.

Step 5:
Use Firebase listeners to listen for real-time updates to your data. For example, in Firestore, you can set up listeners to receive updates whenever the data in a collection or document changes.

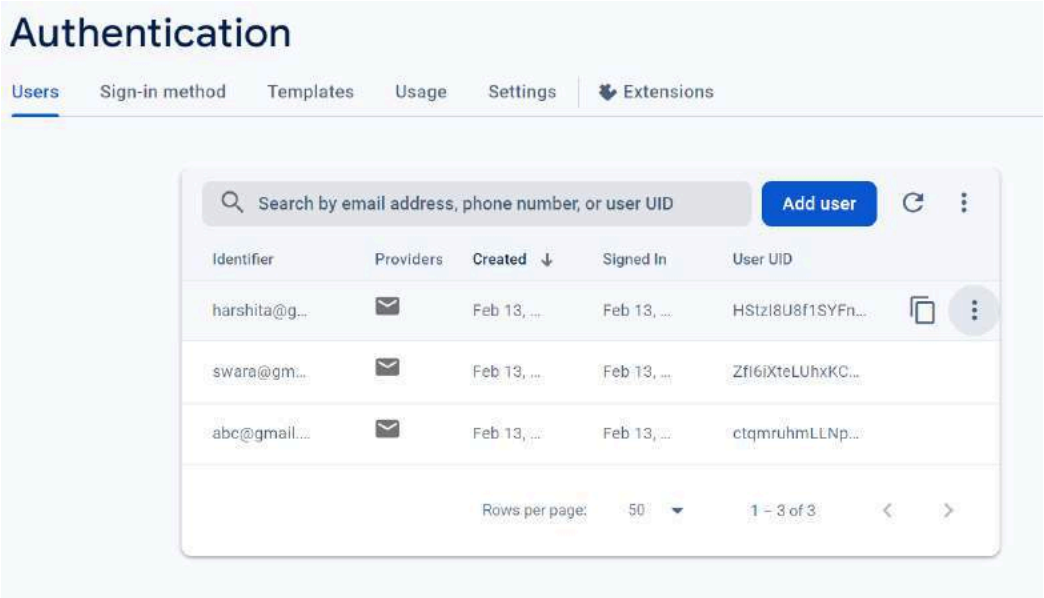
Step 6:
Implement error handling logic to handle exceptions and errors that may occur when interacting with Firebase services.

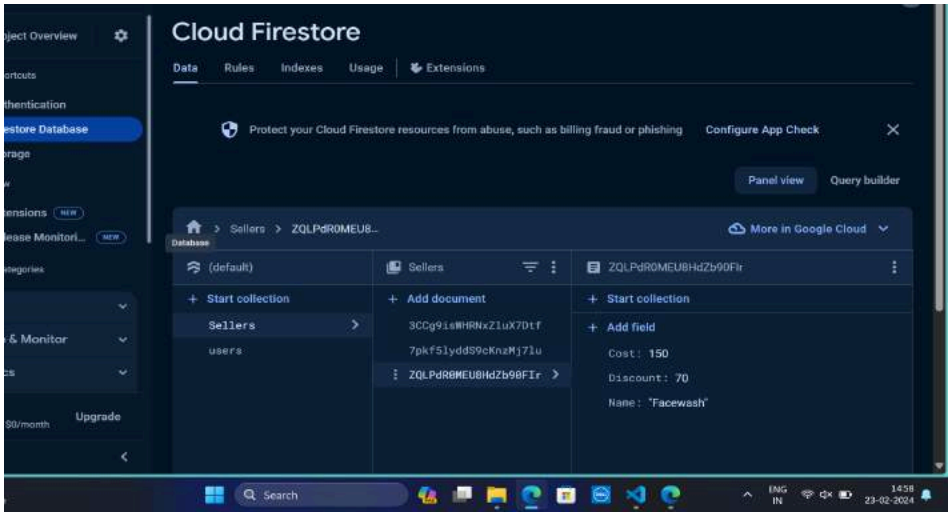
Output:

```
import 'package:amazon_clone/layout/screen_layout.dart';
import 'package:amazon_clone/model/product_model.dart';
import 'package:amazon_clone/providers/user_details_provider.dart';
import 'package:amazon_clone/screens/product_screen.dart';
import 'package:amazon_clone/screens/results_screen.dart';
import 'package:amazon_clone/screens/sell_screen.dart';
import 'package:amazon_clone/screens/sign_in_screen.dart';
import 'package:amazon_clone/utils/color_themes.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  if (kIsWeb) {
    await Firebase.initializeApp(
        options: const FirebaseOptions(
            apiKey: "AIzaSyDvpZFXdfmjxwc0x2oClDw01sNoMrLoF4c",
            authDomain: "clone-12f8a.firebaseapp.com",
            projectId: "clone-12f8a",
            storageBucket: "clone-12f8a.appspot.com",
            messagingSenderId: "413818422314",
            appId: "1:413818422314:web:f7981d7db247b565732f53"));
  } else {
    await Firebase.initializeApp();
  }
  runApp(const AmazonClone());
}

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

  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [ChangeNotifierProvider(create: (_) => UserDetailsProvider())],
      child: MaterialApp(
        title: "Amazon Clone",
        debugShowCheckedModeBanner: false,
        theme: ThemeData.light().copyWith(
```

```
        scaffoldBackgroundColor: backgroundColor,
      ),
      home: StreamBuilder(
        stream: FirebaseAuth.instance.authStateChanges(),
        builder: (context, AsyncSnapshot<User?> user) {
          if (user.connectionState == ConnectionState.waiting) {
            return const Center(
              child: CircularProgressIndicator(
                color: Colors.orange,
              ),
            );
          } else if (user.hasData) {
            return const ScreenLayout();
            //return const SellScreen();
          } else {
            return const SignInScreen();
          }
        }),
    ),
  );
 }
}
```

Connected with firebase.





Conclusion: Firebase has been connected to the Flutter application for user authentication purposes.

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.	13
Name	Swarali Dhobale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

PWA EXPT-07

Name: Swarali Dhobale
Class:D15A-13

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

Theory:

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

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

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

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

3. Faster Services
PWAs can cache the data and serve the user with text stylesheets, images, and other web content even before the page loads completely. This lowers the waiting time for the end-users

and helps the brands improve the user engagement and retention rate, which eventually adds value to their business.

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

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

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

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

Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.

Responsive — They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can later become available.

App-like — They behave with the user as if they were native apps, in terms of interaction and navigation.

Updated — Information is always up-to-date thanks to the data update process offered by service workers.

Secure — Exposed over HTTPS protocol to prevent the connection from displaying information or altering the contents.

Searchable — They are identified as “applications” and are indexed by search engines.

Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
Installable — They allow the user to “save” the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to face all the steps and problems related to the use of the app store.

Linkable — Easily shared via URL without complex installations.

Offline — Once more it is about putting the user before everything, avoiding the usual error message in case of weak or no connection. The PWA are based on two particularities: first of all the ‘skeleton’ of the app, which recalls the page structure, even if its contents do not respond and its elements include the header, the page layout, as well as an illustration that signals that the page is loading.

Code and Output:

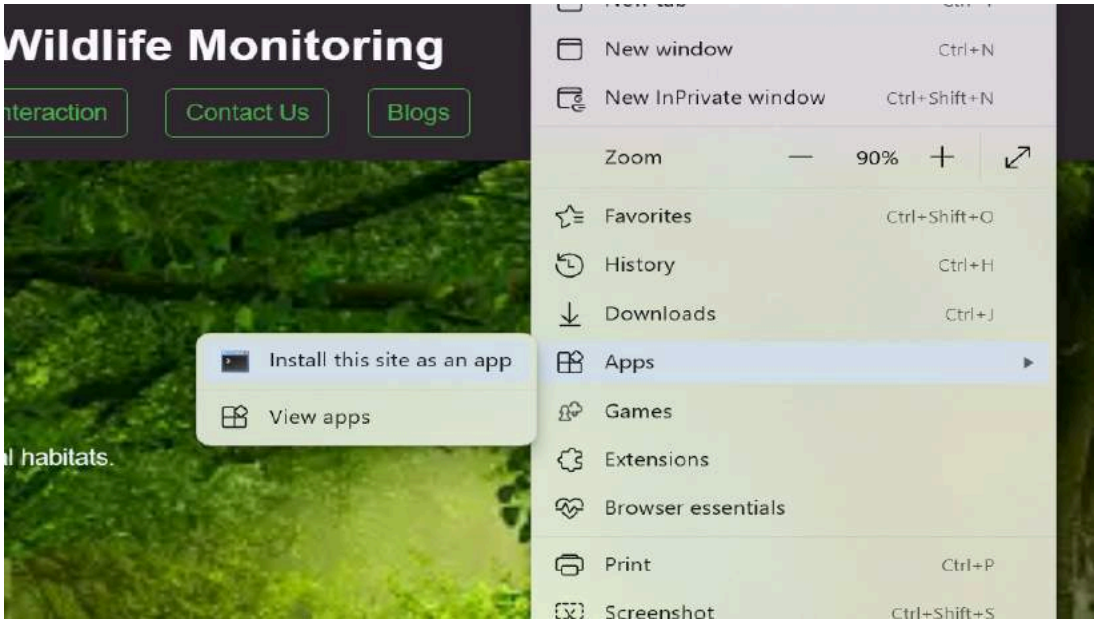
```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Wildlife Monitoring</title>
   <style>
            body {
          font-family: Arial, sans-serif;
          margin: 0;
          padding: 0;
          background:
url("https://media.istockphoto.com/id/537389352/photo/tropical-rainforest.webp?b=1&s=170667
a&w=0&k=20&c=rPxzpapBr16QF47PWJ474qVXE1SjaRImJvt6pClavew=") no-repeat fixed;
          background-size: cover;
       }

       header {
          background-color: #2f292f;
          color: white;
          padding: 9px;
          text-align: center;
       }
```

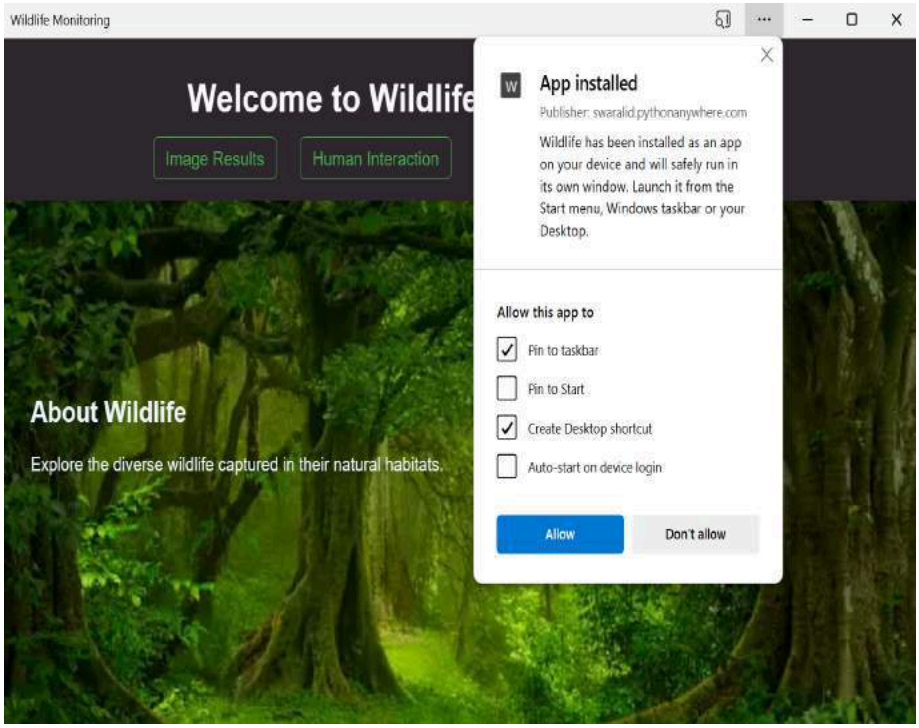
```
nav ul {
    list-style-type: none;
    padding: 0;
}
nav li {
    display: inline;
    margin-right: 20px;
}
.main-content {
    max-width: 800px;
    margin: 120px auto;
    padding: 20px;
    color: aliceblue;
    border-radius: 10px;
}

.predicted-image {
    border: 2px solid red;
    max-width: 100%;
}

footer {
    background-color: #2f292f;
    color: white;
    text-align: center;
    padding: 10px;
    position: fixed;
    bottom: 0;
    width: 100%;
}

nav a {
    text-decoration: none;
    padding: 8px 12px;
    border: 1px solid #4CAF50;
    color: #4CAF50;
    border-radius: 5px;
    transition: background-color 0.3s;
}

nav a:hover {
    background-color: #4CAF50;
    color: white;
}
```

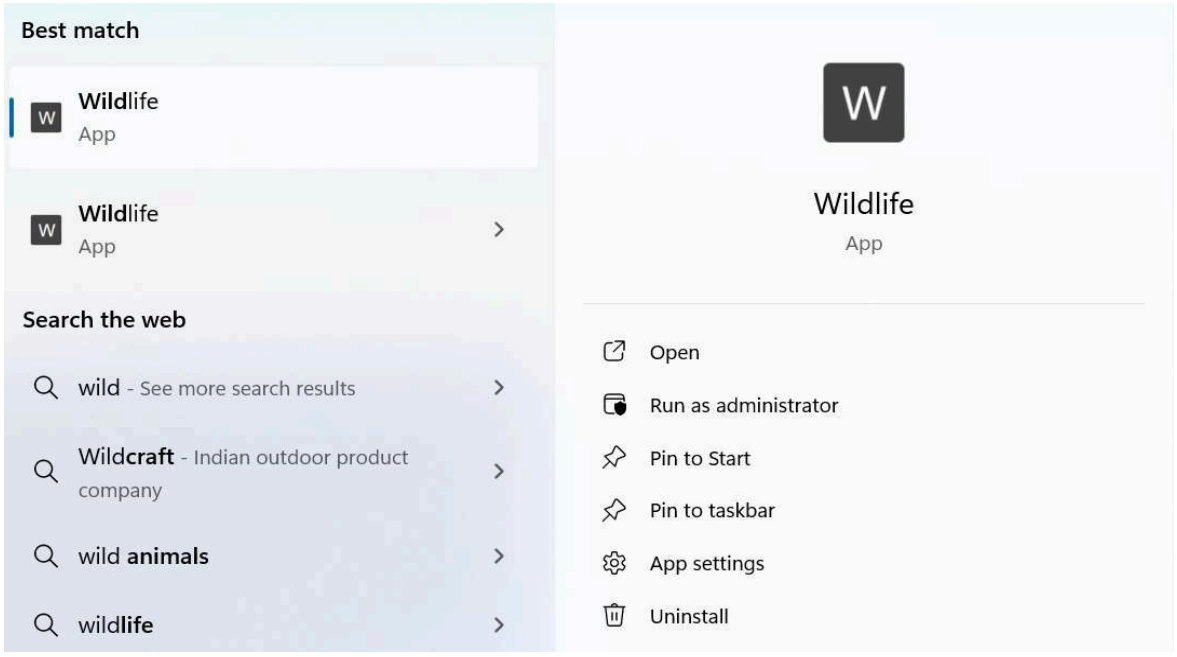
```
        .loading-message-text {
            color: yellow;
            font-size: 18px;
        }
        @media (max-width: 600px) {
            nav ul {
                text-align: center;
            }

            nav li {
                display: block;
                margin: 30px 0;
            }
        }
    </style>
<script>
    document.addEventListener("DOMContentLoaded", function () {
        var imageResultsBtn = document.getElementById("image-results-btn");
        var loadingMessage = document.getElementById("loading-message");

        imageResultsBtn.addEventListener("click", function () {
            loadingMessage.innerHTML = '<p class="loading-message-text">Hang on! Your content
is being loaded...</p>';
            setTimeout(function () {
                loadingMessage.innerHTML = '';

            }, 30000);
        });
    });
</script>
</head>
<body>
<header>
    <h1>Welcome to Wildlife Monitoring</h1>
    <nav>
        <ul>
            <li><a id="image-results-btn" href="/results">Image Results</a></li>
            <li><a href="/human">Human Interaction</a></li>
            <li><a href="/contact">Contact Us</a></li>
            <li><a href="/blogs">Blogs</a></li>
        </ul>
    </nav>
</header>
```

```
<section class="main-content">
    <div id="loading-message"></div>
    <h2>About Wildlife</h2>
    <p>Explore the diverse wildlife captured in their natural habitats.</p></section>
</body>
```

Output:







Conclusion: Created a progressive web app and installed it on the 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.	13
Name	Swarali Dhobale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

PWA EXPT-08

Name: Swarali Dhobale
Class:D15A-13

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.

What can't we do with Service Workers?

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

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

Code and Output:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Wildlife Monitoring</title>
   <style>
          body {
        font-family: Arial, sans-serif;
        margin: 0;
        padding: 0;
        background:
url("https://media.istockphoto.com/id/537389352/photo/tropical-rainforest.webp?b=1&s=170667
a&w=0&k=20&c=rPxzpapBr16QF47PWJ474qVXE1SjaRImJvt6pClavew=") no-repeat fixed;
        background-size: cover;
      }

      header {
        background-color: #2f292f;
        color: white;
        padding: 9px;
        text-align: center;
      }
      nav ul {
        list-style-type: none;
        padding: 0;
      }
      nav li {
        display: inline;
        margin-right: 20px;
```

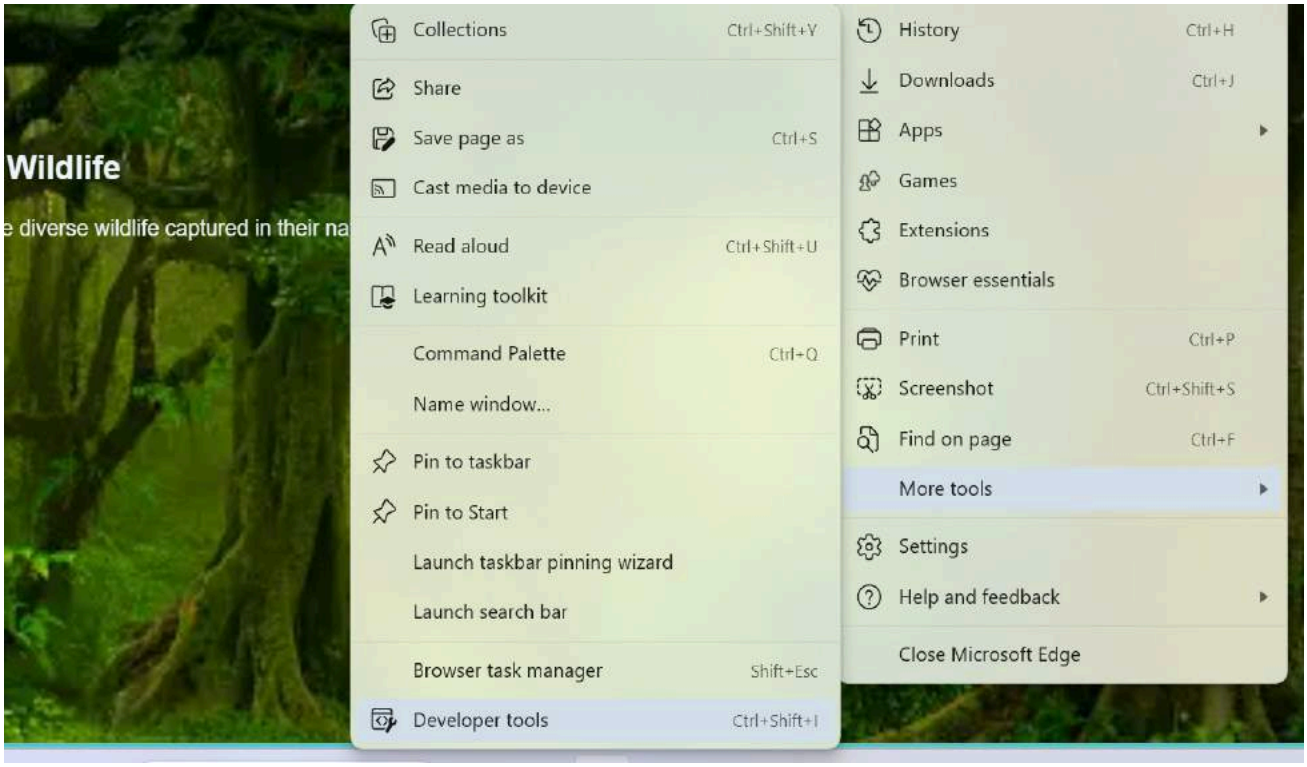
```
}
.main-content {
    max-width: 800px;
    margin: 120px auto;
    padding: 20px;
    color: aliceblue;
    border-radius: 10px;
}

.predicted-image {
    border: 2px solid red;
    max-width: 100%;
}

footer {
    background-color: #2f292f;
    color: white;
    text-align: center;
    padding: 10px;
    position: fixed;
    bottom: 0;
    width: 100%;
}

nav a {
    text-decoration: none;
    padding: 8px 12px;
    border: 1px solid #4CAF50;
    color: #4CAF50;
    border-radius: 5px;
    transition: background-color 0.3s;
}

nav a:hover {
    background-color: #4CAF50;
    color: white;
}

.loading-message-text {
    color: yellow;
    font-size: 18px;
}
@media (max-width: 600px) {
    nav ul {
```

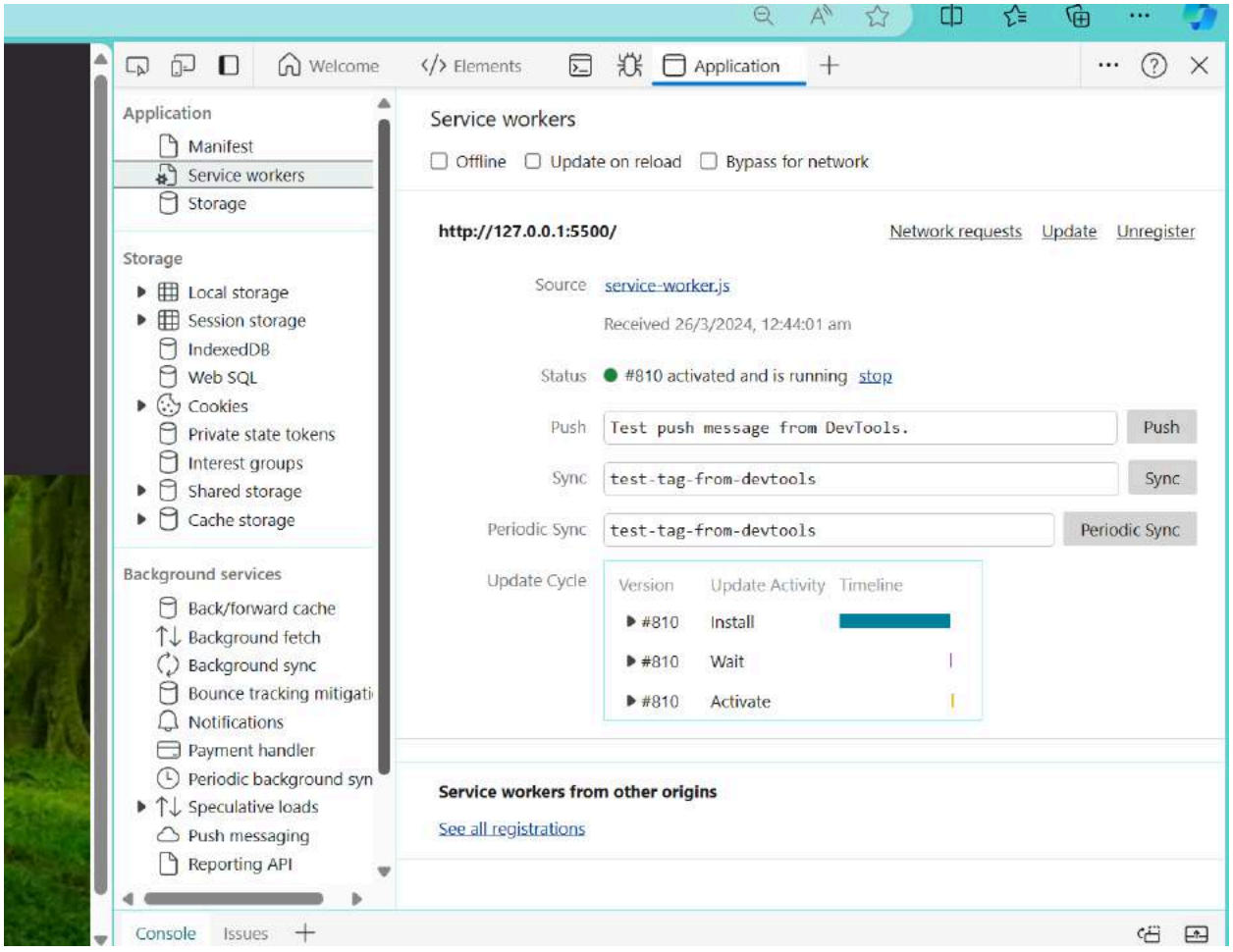
```
            text-align: center;
        }

        nav li {
            display: block;
            margin: 30px 0;
        }
    }
</style>
<script>
    document.addEventListener("DOMContentLoaded", function () {
        var imageResultsBtn = document.getElementById("image-results-btn");
        var loadingMessage = document.getElementById("loading-message");

        imageResultsBtn.addEventListener("click", function () {
            loadingMessage.innerHTML = '<p class="loading-message-text">Hang on! Your content
is being loaded...</p>';
            setTimeout(function () {
                loadingMessage.innerHTML = '';

            }, 30000);
        });
    });
</script>
</head>
<body>
<header>
    <h1>Welcome to Wildlife Monitoring</h1>
    <nav>
        <ul>
            <li><a id="image-results-btn" href="/results">Image Results</a></li>
            <li><a href="/human">Human Interaction</a></li>
            <li><a href="/contact">Contact Us</a></li>
            <li><a href="/blogs">Blogs</a></li>
        </ul>
    </nav>
</header>
<section class="main-content">
    <div id="loading-message"></div>
    <h2>About Wildlife</h2>
    <p>Explore the diverse wildlife captured in their natural habitats.</p></section>
<script src="app.js"></script>
</body>
```

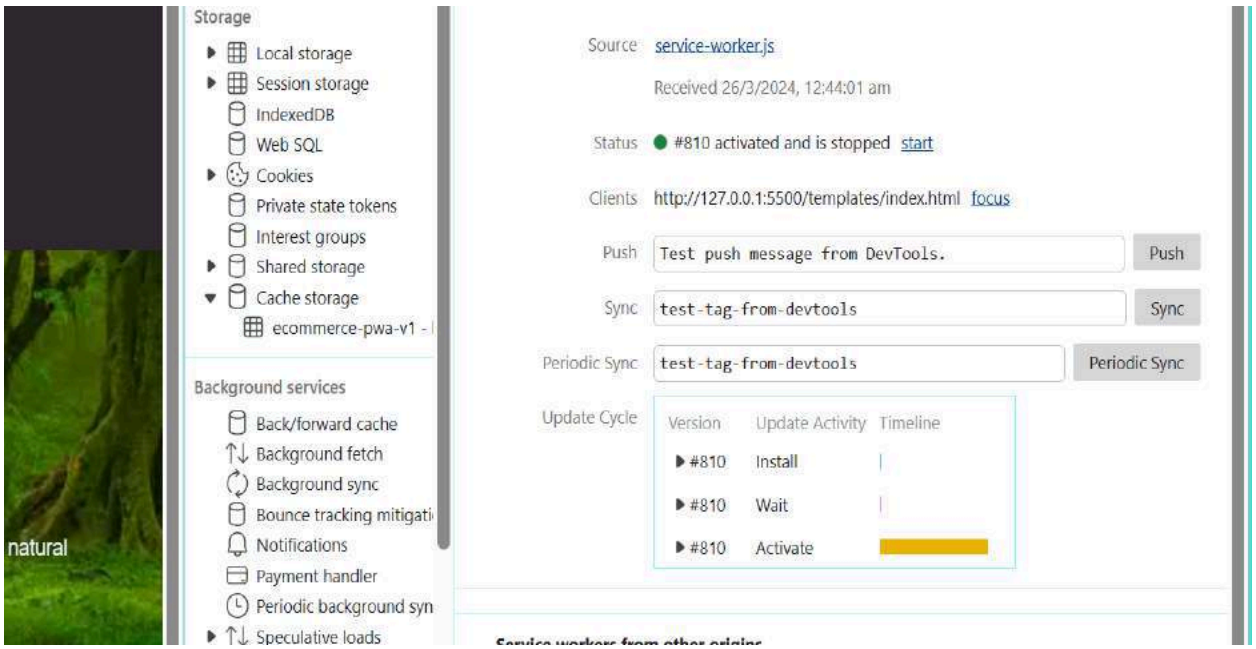
app.js

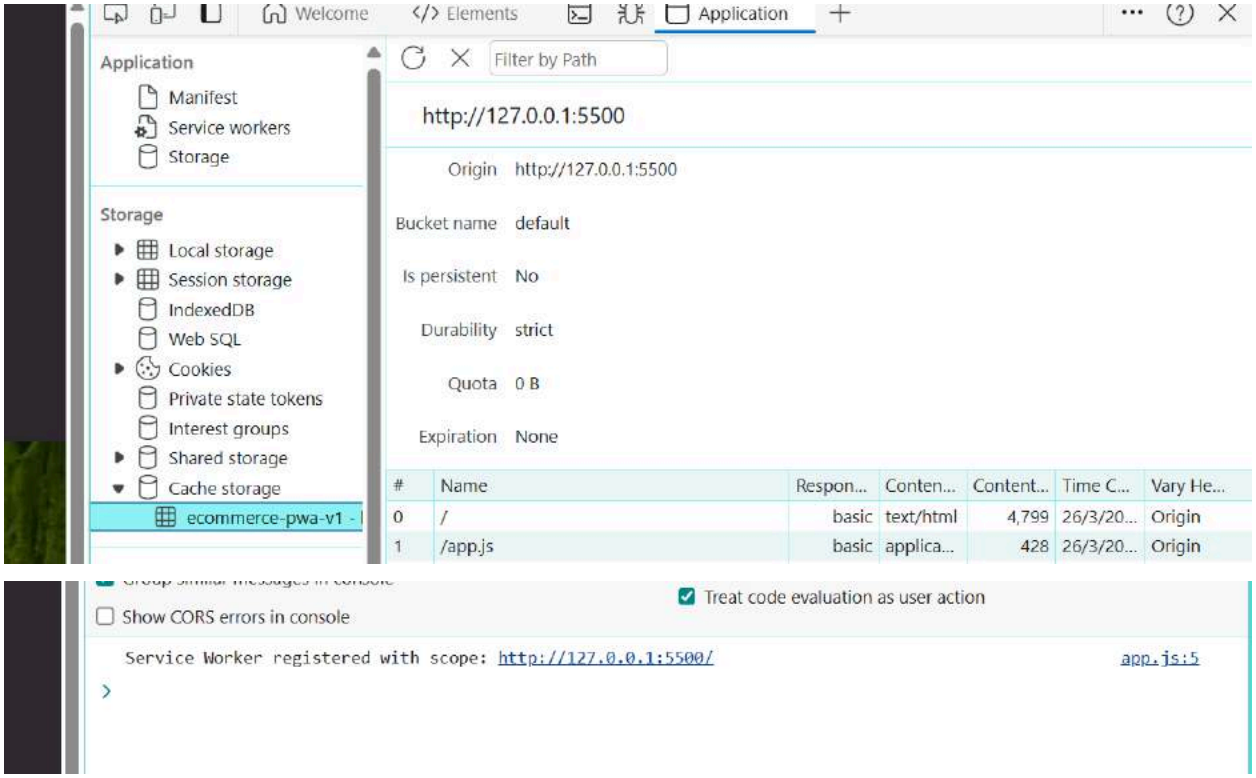
```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', () => {
    navigator.serviceWorker.register('/service-worker.js')
      .then(registration => {
        console.log('Service Worker registered with scope:', registration.scope);
      })
      .catch(error => {
        console.error('Service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

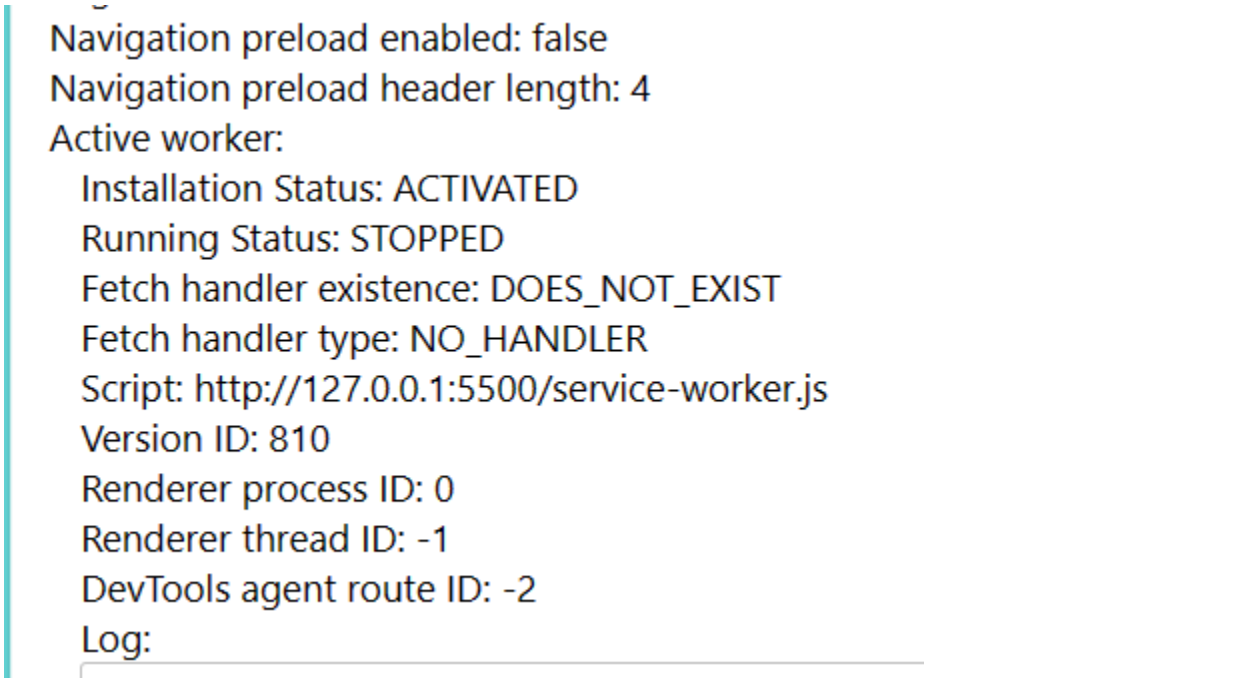
Output:




Welcome
Elements
Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
IndexedDB
Web SQL
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Background services
Back/forward cache
Background fetch
Background sync
Bounce tracking mitigati
Notifications
Payment handler
Periodic background syn
Speculative loads
Push messaging
Reporting API
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5500/
Network requests
Update
Unregister
Source
service-worker.js
Received 26/3/2024, 12:44:01 am
Status
#810 activated and is running
stop
Push
Test push message from DevTools.
Push
Sync
test-tag-from-devtools
Sync
Periodic Sync
test-tag-from-devtools
Periodic Sync
Update Cycle
Version
Update Activity
Timeline
#810
Install
#810
Wait
#810
Activate
Service workers from other origins
See all registrations
Console
Issues


Storage
Local storage
Session storage
IndexedDB
Web SQL
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
ecommerce-pwa-v1 -
Background services
Back/forward cache
Background fetch
Background sync
Bounce tracking mitigati
Notifications
Payment handler
Periodic background syn
Speculative loads
Source
service-worker.js
Received 26/3/2024, 12:44:01 am
Status
#810 activated and is stopped
start
Clients
http://127.0.0.1:5500/templates/index.html
focus
Push
Test push message from DevTools.
Push
Sync
test-tag-from-devtools
Sync
Periodic Sync
test-tag-from-devtools
Periodic Sync
Update Cycle
Version
Update Activity
Timeline
#810
Install
#810
Wait
#810
Activate





<u>Conclusion</u>: Registered a service worker, and completed the installation and activation process for a new service worker for the 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.	13
Name	Swarali Dhobale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

PWA EXPT-09

Name: Swarali Dhobale
Class:D15A_13

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

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

- **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.

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

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

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

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

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

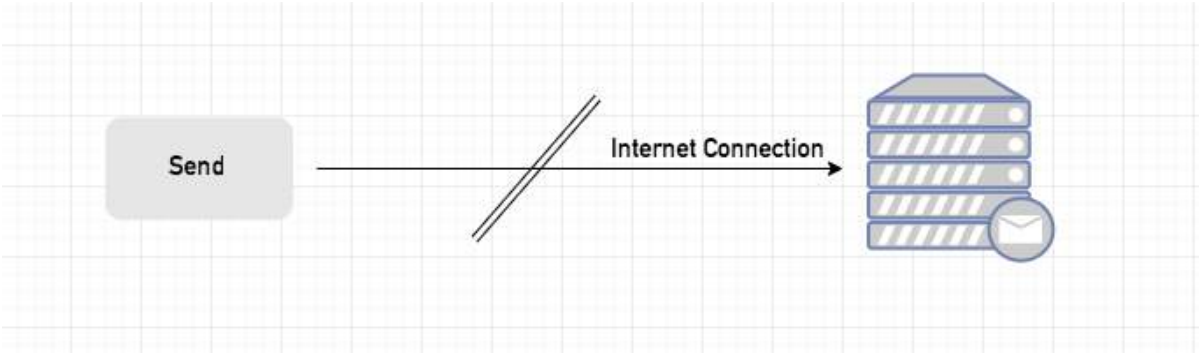
Sync Event

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

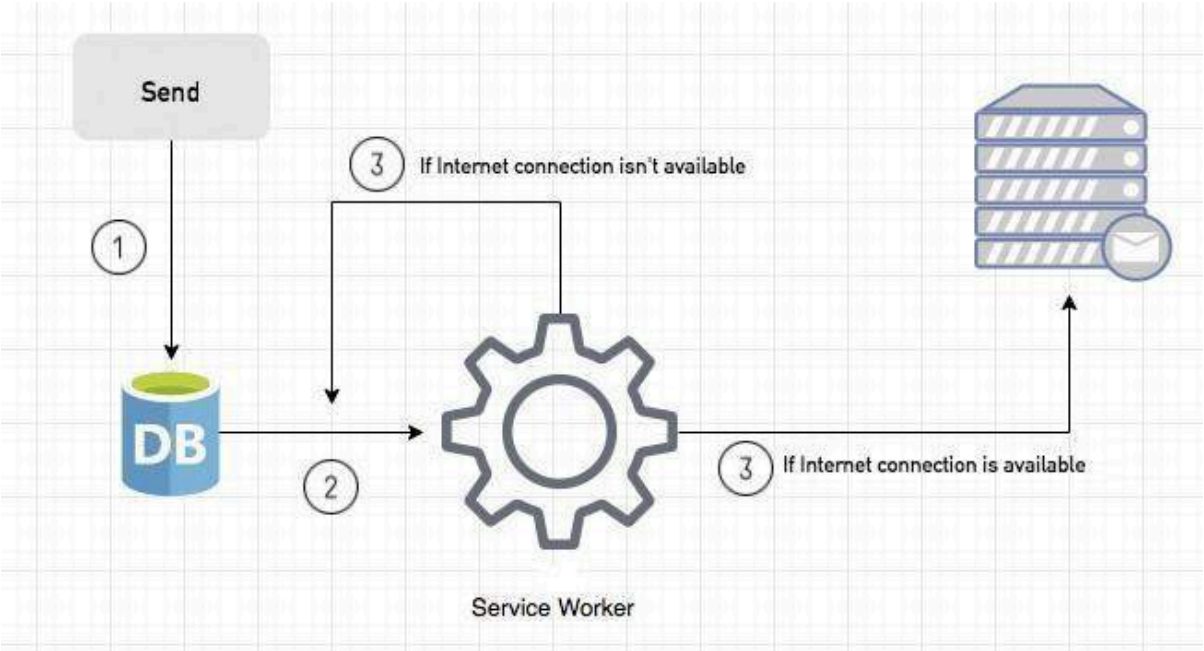
Here is a job for the Background Sync.

The following view shows the classical process of sending email to us. If the Internet

Connection is broken, we can’t send any content to Mail Server.



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



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

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

Push Event

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

“Notification.requestPermission();” is the necessary line to show notification to the user.

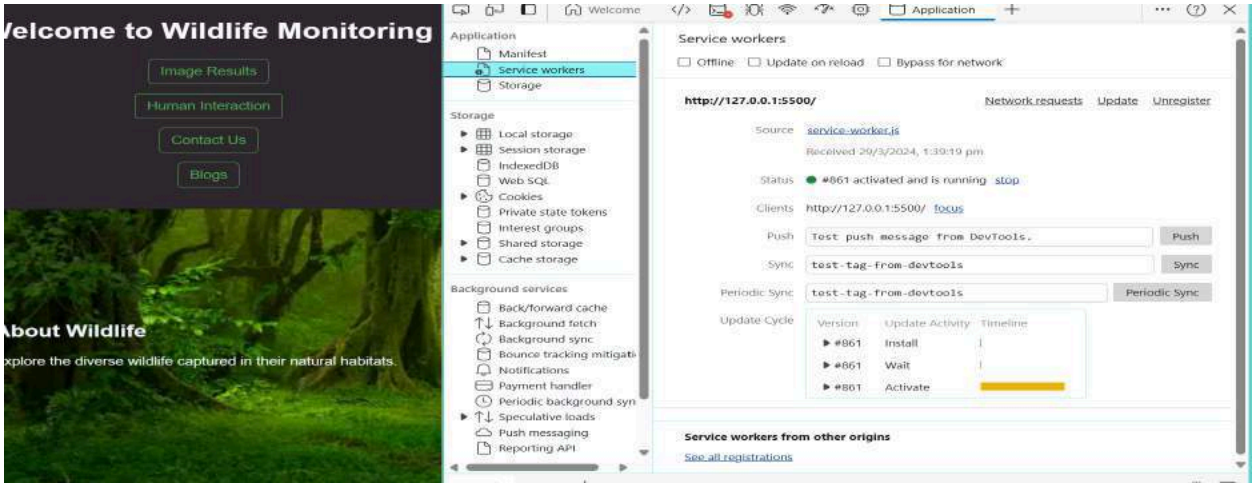
Service Worker Code:-

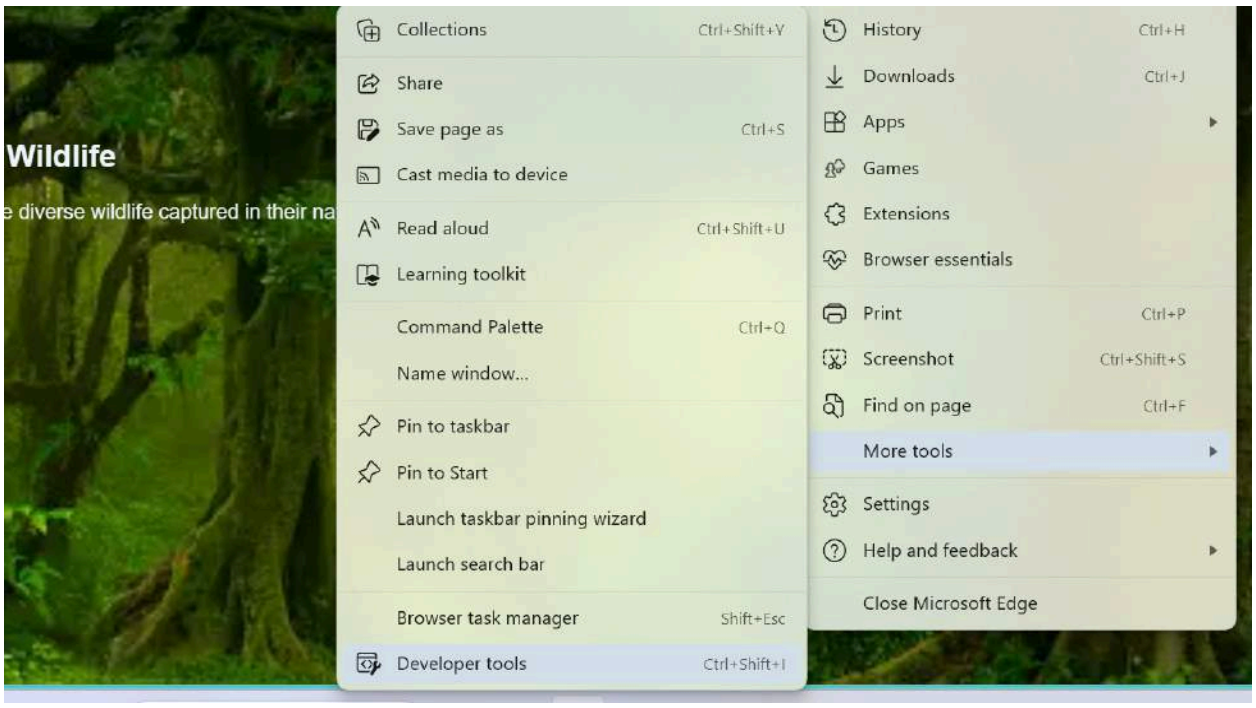
```
// Service Worker Installation
self.addEventListener("install", function (event) {
    event.waitUntil(preLoad());
});
// Fetch Event Listener
self.addEventListener("fetch", function (event) {
    event.respondWith(
        checkResponse(event.request)
            .catch(function () {
                console.log("Fetch from cache successful!");
                return returnFromCache(event.request);
            })
    );
    console.log("Fetch successful!");
    event.waitUntil(addToCache(event.request));
});
// Sync Event Listener
self.addEventListener("sync", (event) => {
    if (event.tag === "syncMessage") {
        console.log("Sync successful!");
    }
});
self.addEventListener("push", function (event) {
    if (event && event.data) {
        try {
            var data = event.data.json(); // Attempt to parse JSON data
            console.log("Push notification data:", data); // Log the received data
            if (data && data.method === "pushMessage")
```

```
    console.log("Push notification sent");
            self.registration.showNotification("New Notification", {
                body: data.message,
                icon: 'img.png',
            }).catch(function(error) {
                console.error("Error displaying notification:", error);
            });
        }
    } catch (error) {
        console.error("Error parsing push data:", error);
    }
  } else {
    console.error("Push event does not contain data.");
  }
});
// Preload Function
function preLoad() {
  return caches.open("offline").then(function (cache) {
    return cache.addAll([
      '/index.html',
      '/manifest.json',
      '/service-worker.js',
      '/app.js'
    ]);
  });
}

// Check Response Function
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);
            });
    });
};

// Return from Cache Function
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) {
```

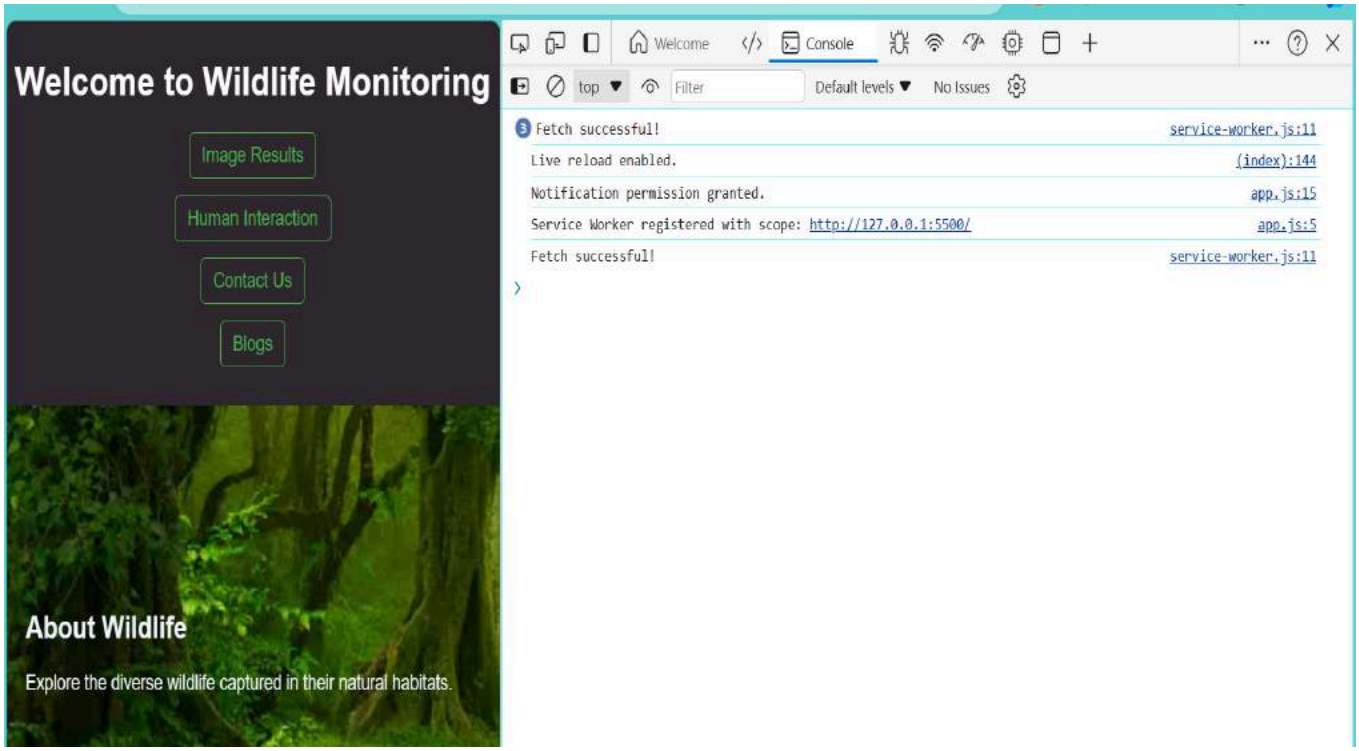
```
        return fetch(request).then(function (response) {
            cache.put(request, response.clone());
            return response;
        });
    });
};
```

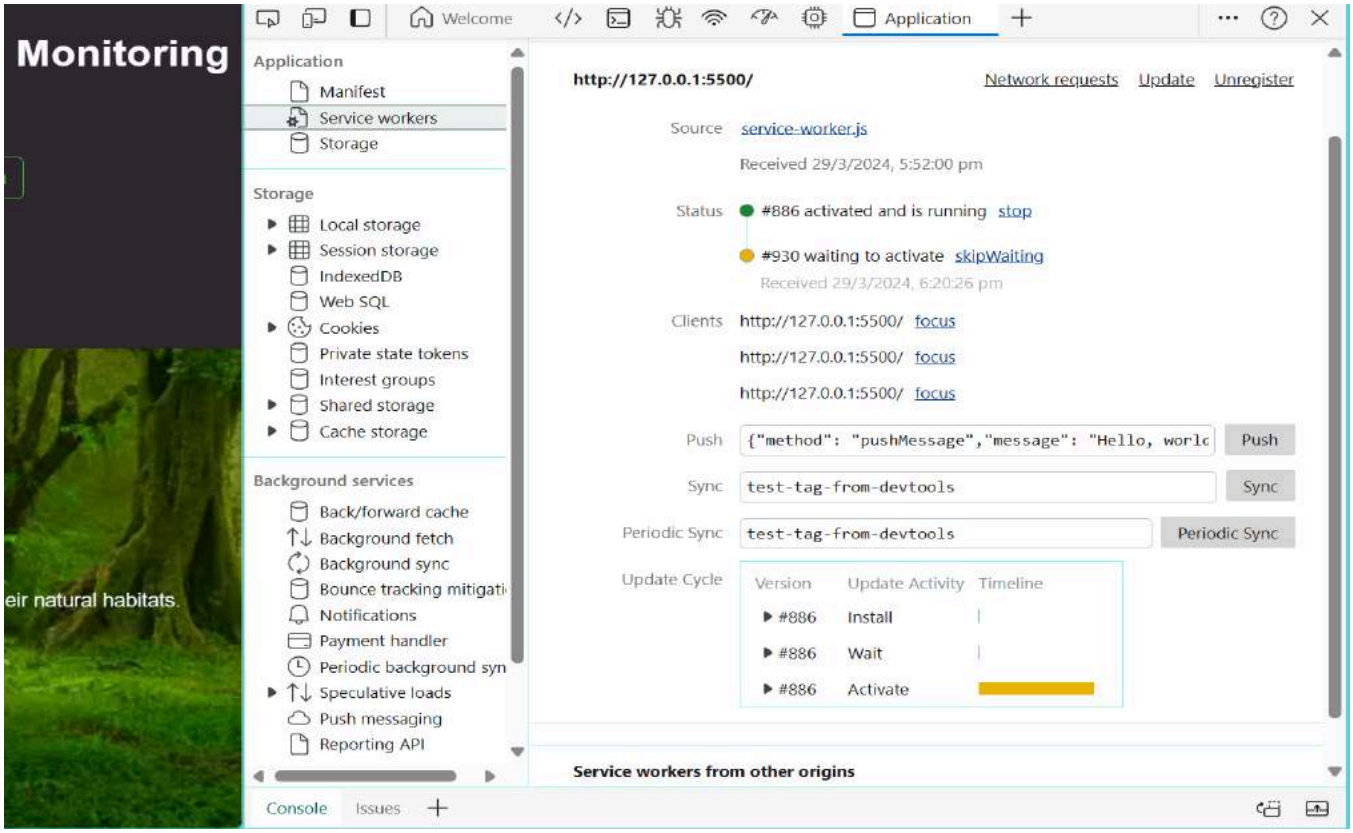





```
[NEW] Explain Console errors by using Copilot in Edge: click  to explain an error. Learn more
Don't show again
Push notification sent                                  service-worker.js:27
```

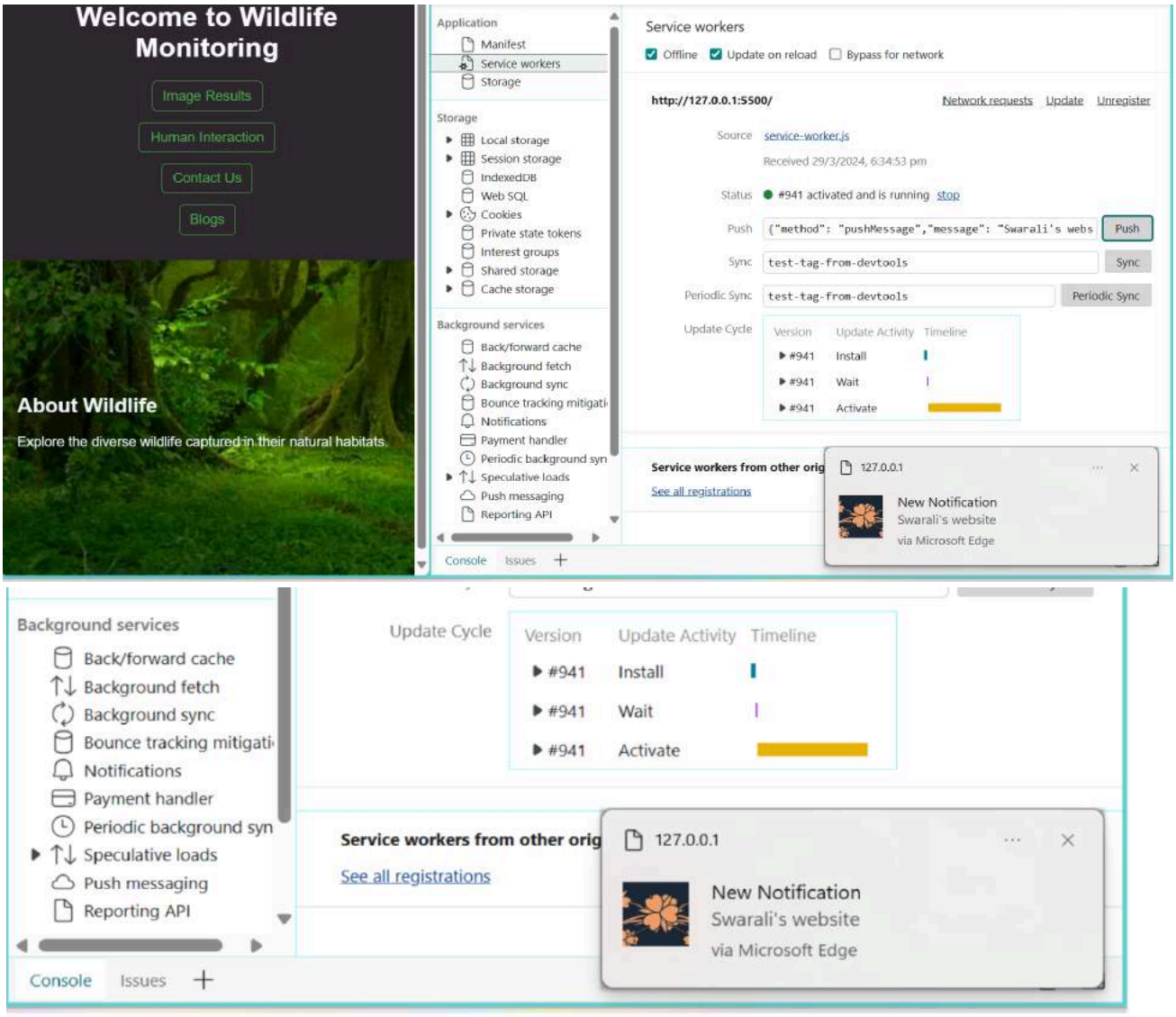
Fetch successful-





NOTIFICATION PERMISSION GRANTED,
PUSH NOTIFICATION SENT-(along with image)

```
3 Fetch successful!                                                    service-worker.js:16
  Live reload enabled.                                                 (index):144
  Notification permission granted.                                     app.js:15
  Service Worker registered with scope: http://127.0.0.1:5500/         app.js:5
  Fetch successful!                                                    service-worker.js:16
  Push notification data: ▼ {method: 'pushMessage', message: "Swarali's website"} i   service-worker.js:32
                            message: "Swarali's website"
                            method: "pushMessage"
                          ▶ [[Prototype]]: Object
  Push notification sent                                               service-worker.js:34
```



Conclusion: Service worker events have been implemented successfully.

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	13
Name	Swarali Dhobale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

PWA EXPT-10

Name: Swarali Dhobale
Class:D15A_13

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

Theory:

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.

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

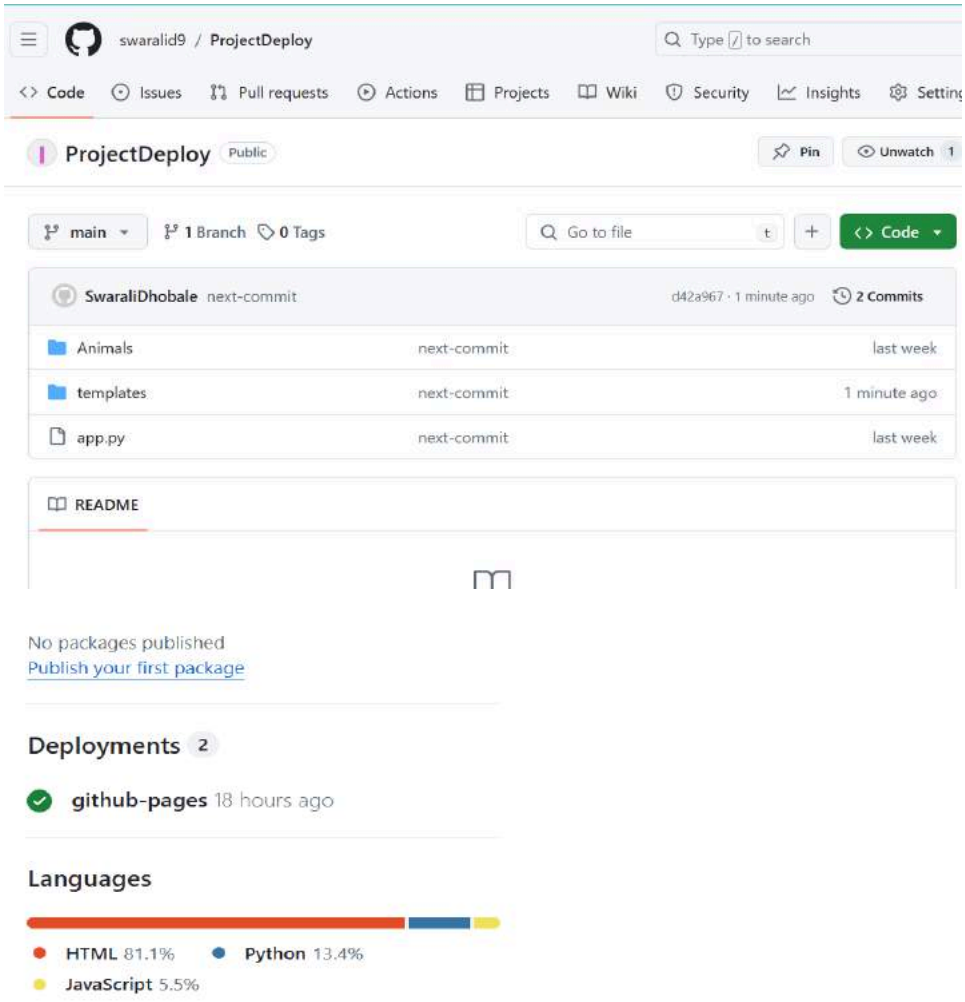
Pros

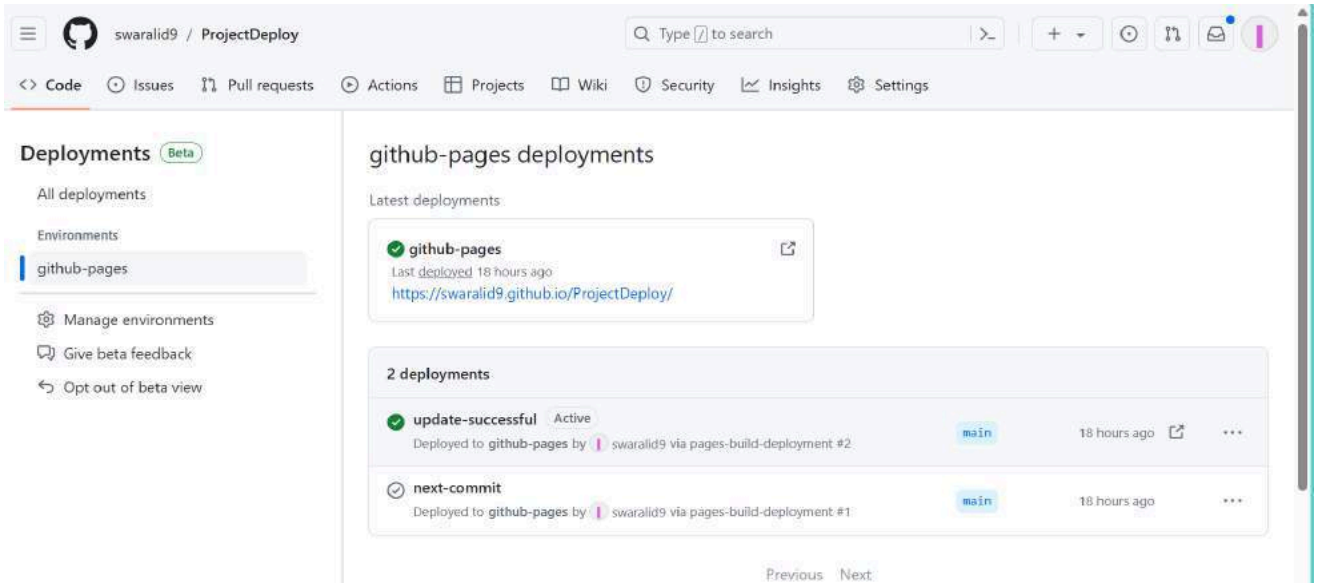
1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5.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

Cons

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

Link to github repo:- LINK





Conclusion: Website has been deployed on Github Pages.

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	13
Name	Swarali Dhobale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

PWA EXPT-11

Name: Swarali Dhobale
Class:D15A_13

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

Theory:

Google Lighthouse :

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

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

Key Features and Audit Metrics

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

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.
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.

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.

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 HTTPSAvoiding the use of deprecated code elements like tags, directives, libraries, etc. Password input with paste-into disabled Geo-Location and cookie usage alerts on load, etc.

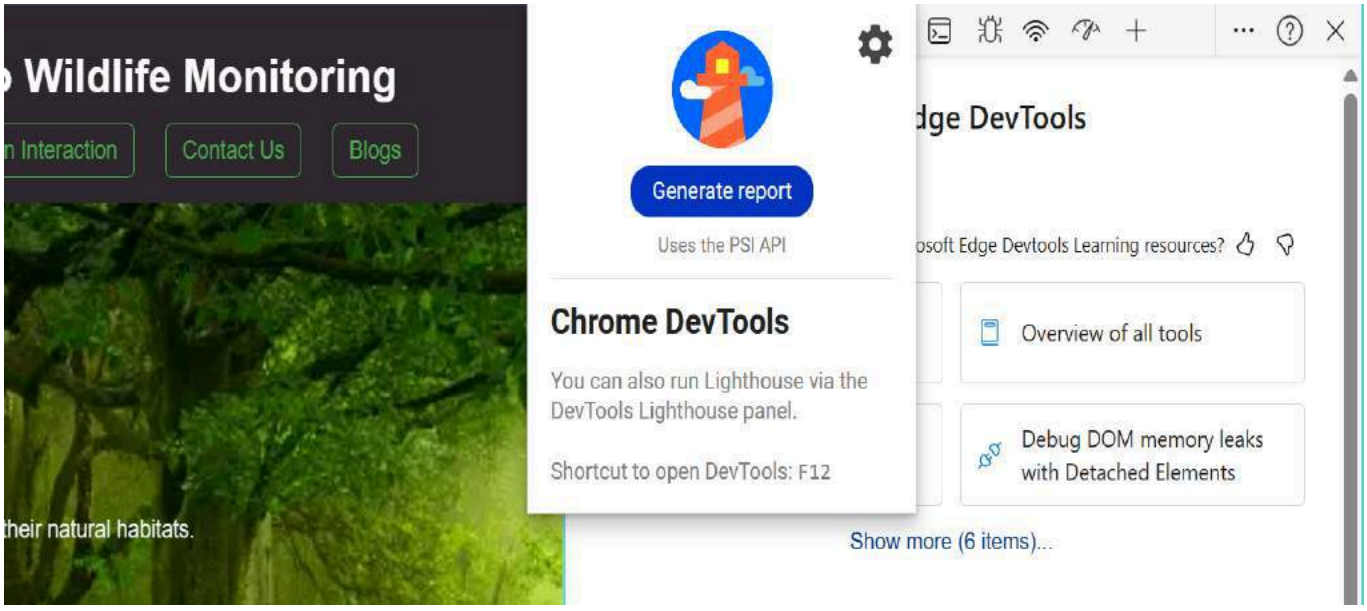
Manifest.json

```
{
  "name": "Wildlife Monitoring",
  "short_name": "Wildlife",
  "description": "Explore the diverse wildlife captured in their natural habitats.",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#4CAF50",
  "icons": [
    {
      "src": "/images/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/images/icon-512x512.png",
      "sizes": "512x512",
```

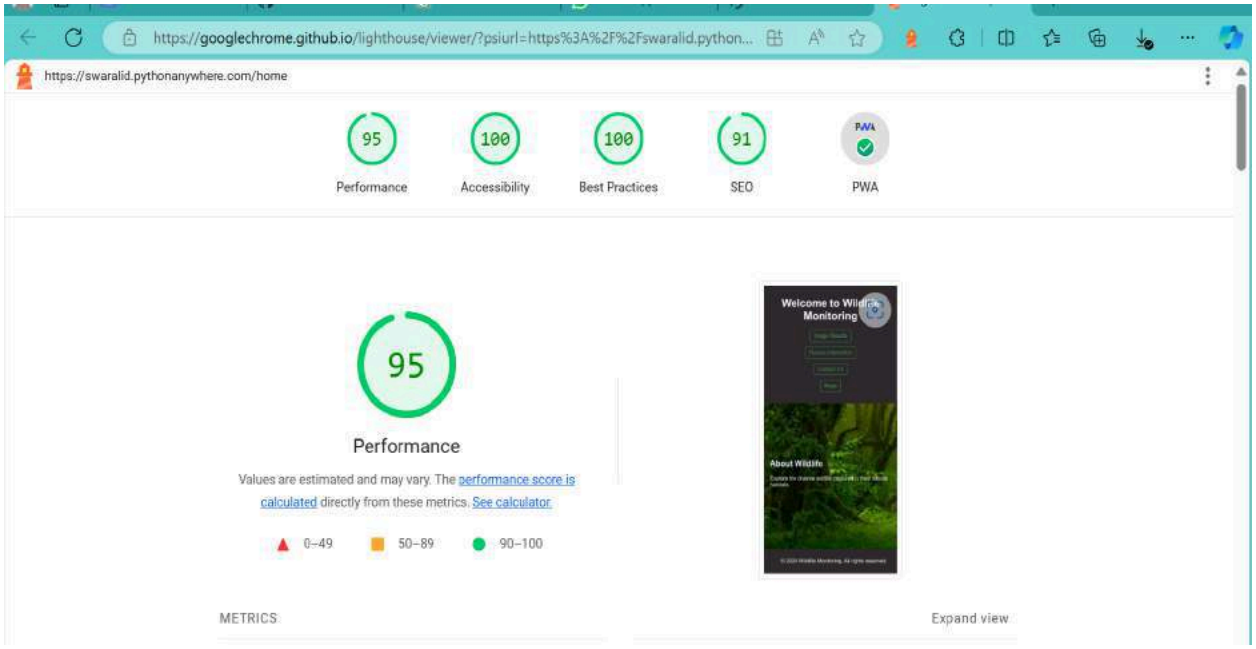
```
      "type": "image/png",
      "purpose": "any maskable"
    }
  ],
  "scope": "/",
  "categories": ["utilities"],
  "lang": "en",
  "dir": "auto",
  "related_applications": [],
  "prefer_related_applications": false,
  "screenshots": [],
  "shortcuts": [],
  "orientation": "portrait-primary",
  "serviceworker": {
    "src": "/service-worker.js",
    "scope": "/"
  }
}
```

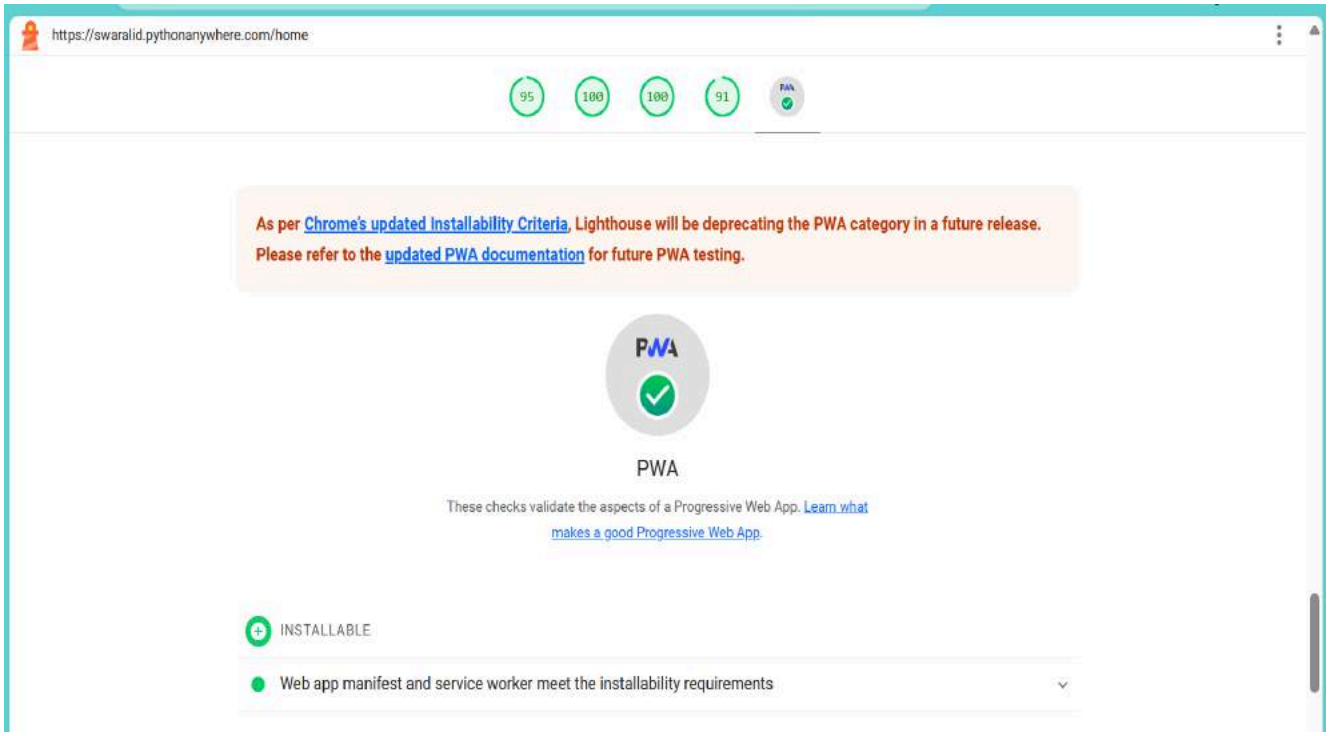
Implementation

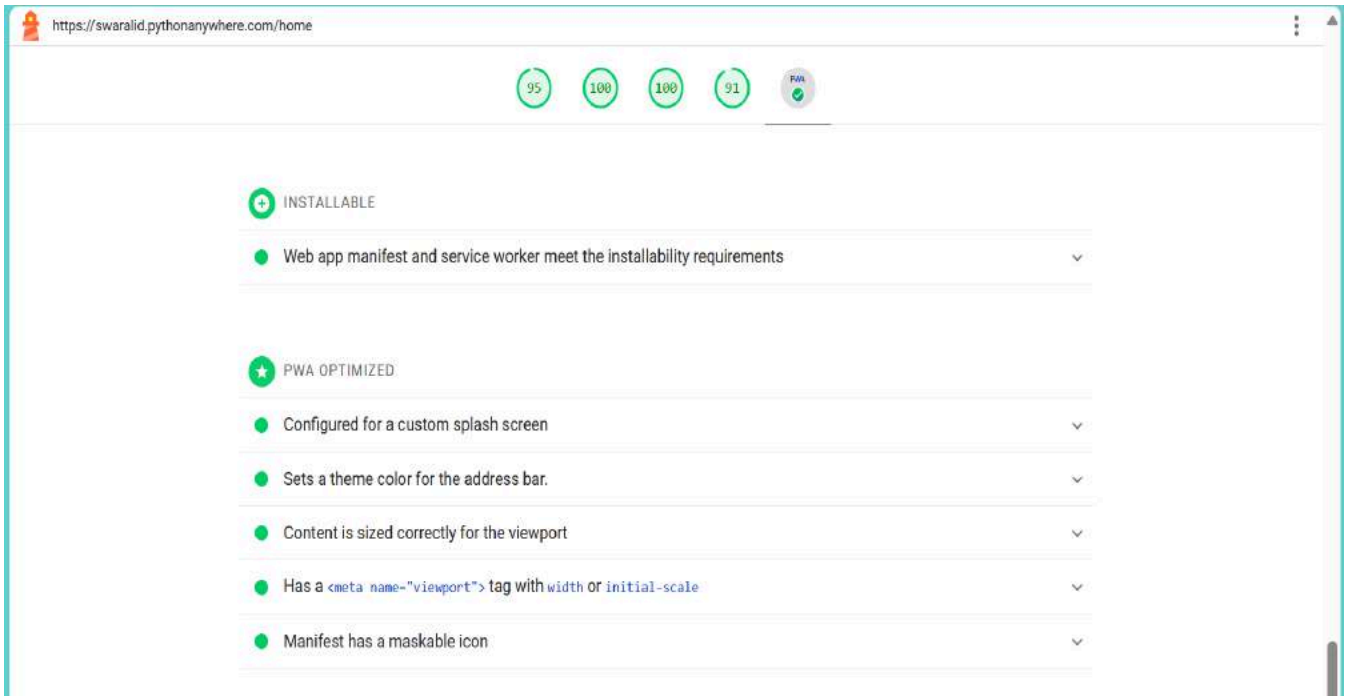
Added extension:-



Report generated:-







Conclusion: Website has been completely configured for PWA optimization, as seen in above image.

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.	13
Name	Swarali Dhobale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	5

Swarali Dhobale
D15A - 13

Assignment 1

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

→ KEY FEATURES OF FLUTTER :—

Flutter is a popular open-source UI software development toolkit created by Google.

① Rich Set of Widgets :—

Flutter comes with a comprehensive set of customizable Widgets that help in creating visually appealing and responsive user interfaces.

② Expressive UI :—

Flutter provides a rich set of design elements animations, and effects, allowing developers to create visually attractive and expressive user interfaces.

③ High Performance :—

Flutter apps are compiled to native ARM code which results in high performance.

* Advantages of flutter include -

1) Faster development
2) Cost-effectiveness
3) Consistent UI
4) Easy maintainance
5) Wide range of plugins

* Differences from traditional approaches :—

1. Cross-platform development
— Traditional approaches often involve separate codebases for different platforms.

2. Compiled code :
— Flutter compiles to native machine code providing better performance compared to interpreted languages used in some traditional approaches.

* Popularity :-

1) Ease of Learning - For developers with a good knowledge of object-oriented programming languag

2) Efficiency - The efficiency gained through Hot Reload and a single codebase appeals to develope and businesses looking to streamline the developm process.

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

→ The widget is a hierarchical structure of UI elements that are used to construct the user interface of an application.
Everything in flutter is a widget, which are combined to create complex and interactive user interfaces.

WIDGET structure :-
1. Root widgets
2. Hierarchy of widgets
3. Leaf widgets
4. Parent and child widgets

How the widgets are widgets are used :—

① Building UI Elements :—
- Every visual element on the screen
- simple text and buttons to complex layouts & animations

② Composition :—
Widgets can be combined and nested to create more complex UI components.
Column → Container → Text widget

Commonly used widgets and their roles :—

① Container: Role: A box model that can contain other widgets. It's often used for layout and styling.

Ex:
```
Container(
  width: 100.0,
  height: 100.0
  child: Text('Hello!'),
)
```

② Column and Row: Role: Widget for arranging children in a vertical or horizontal line.

Ex:
```
Column(
  children: [
    Text('Item 1'),
    Text('Item 2'), ],)
```

③ ListView: Role: a scrollable list of widgets

Ex:
```
ListView(
  children: [
    ListTile(title: Text('Item 1')),
    ListTile(title: Text('Item 2')),
  ],
)
```

3. State management in Flutter:
Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches in Flutter: SetState, Provider, riverpod. Provide scenarios where each approach is suitable.

→ State management is a critical aspect of flutter development as it involves handling and updating the state of an application, which directly influences how the user interface behaves.
Different state management approaches exist in flutter

1. SetState
- It is the most basic and built-in-way of managing state in Flutter. It is suitable for small to moderately complex applications where the state is localized to specific widgets or components.

(2) - + point - suitable for small projects.
- point - limited scalability

2. Provider
- Provider is a popular state management solution that offers a more organizable and scalable approach.

- + point - Centralized state management, minimize boilerpla
- -ve point - may become complex for every large applications.

Teacher's Sign

③ Riverpod :-
It is an advanced state management library built on top of 'Provider'.
It emphasizes simplicity, testability and scalability

- +ve point - Good testability
-ve point - developers may find it overkills for sim projects.

SCENARIOS :-
Use of setState :-
a) small projects
b) state changes are confined to specific widget
c) prioritizing simplicity.

Use of Provider -
a) medium - large scale apps.
b) sharing state efficiently across widgets
c) Organizing codebase

Use of Riverpod -
a) Modern and flexible approach
b) Embracing a more declarative approach
c) Projects requiring scalability.

Q4. Firebase integration in flutter :-
Explain the process of integration of flutter with firebase. 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.

→ Process of integrating flutter with firebase:-

1. Creating a firebase project
2. Registering the app
3. Installing dependencies
4. Initializing Firebase
5. Using Firebase services

Benefits of using firebases:-

1. Real time : Provides a real time NoSQL database or a cloud Firestore

2. Authentication: Firebase offers ready-to-use authenticat-ion solutions supporting providers like Gmail, Facebook

3. Hosting : Firebase hosting enables easy development and hosting of web applications with a CDN.

4. Analytics :- Firebase analytics helps you understand user behaviour and app performance.

Firebase Commonly used services:-

1. Firebase authentication
2. Cloud Firestore
3. Firebase real-time database
4. Firebase Cloud Functions
5. Firebase Cloud Storage
6. Firebase Hosting

Data synchronization in Firebase:-

1) Real time data syne:-
Firebase services like Cloud firestore and the realtime database automatically handle data synchsonization in real time.

2) Offline capabilities:-
Firebase SDK's provide offline support, allowing apps to continue functioning even after the device is offline.

3) Listeners and Streams:-
It uses listeners and streams to notify the application about data changes.

4) Conflict resolution -
Firebase provides built-in conflict resolution mechanisms to handle simultaneous updates from multiple clients.

—X—

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.	13
Name	Swarali Dhobale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	4

21/03/2024

Swarali Dhobale
D15A - 13

Assignment 2

① Define progressive Web app and explain its significance in modern web development. Discuss the key characteristics that differentiate PWAs from traditional

→

A progressive web app is a type of web application that leverages modern web technologies to provide users with a native app-like experience directly through their web browser. PWA's are built using standard web technologies such as HTML, CSS and Javascript but are designed to offer features typically associated with native mobile apps, including offline functionality, push notifications and access to device hardware like cameras and GPS

The significance of PWA in modern web app development lies in their ability to bridge the gap between web and native mobile applications. Some key characteristics that differentiate PWA's from traditional web apps include

① Offline functionality: PWA's can work offline or in low-network conditions by caching essential resources enabling users to access content even when they're not connected to the internet

② Responsive Design: PWAs are built to be responsive ensuring a seamless user experience across various devices and screen sizes, similar to native mobile apps.

Teacher's Sign.:

③ Push notifications :-
PWAs can send push notifications to users allowing businesses to engage with their audience and re-engage ~~with~~ users even when they're not actively using the app.

④ Installation and distribution :-
PWA's can be installed directly from the browser onto a user's device without the need for an app store, streamlining the distribution process and reducing barriers to entry for users

Q2 Define responsive web design and explain its importance in the context of Progressive web apps. Compare and contrast responsive, fluid and adaptive web design approaches.

→ Responsive web design is a web design approach that ensures a website adapt its layers and content to seamlessly function, and display will across various screen size, from desktop to smartphone

Importance of PWA :-

① Foundation for app-like experience :
② PWD ensures proper layout and navigation across devices, mimicking the adaptability of native apps
③ Accessibility
A PWD make PWA accessible wider audience using diverse devices.

FEATURE	RESPONSIVE	FLUID	ADAPTIVE
Layout	Flexible, adaptive	Continuously adjusts based on relative units	Uses multiple fixed width layouts
Development and effort	Moderate	Less effort for basic layout	High initial over layout for each device category
Suitability for PWA's	Ideal founda-tion due to its versatality	Can be used but may require adjustment for optimal experience	Less common for PWA's d to developme overhead.

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

— The lifecycle of services workers involve three main phases

i) Registration :- Service workers are first registered by the web application through javascript code

This registration typically occurs in the main javas file of the application and initiated using the navigator serviceWorker, register(), method.

ii] Installation :—

a) After registeration, the browser begins the installatio phase. During installation, the service worker script is downloaded and cached by the browser.

b) Once script is downloaded, the install event is triggered in the service worker script.

c) This event allows the service worker to perform tasks such as caching static assets or setting up other resources ~~stati~~ needed for offline functional

d) Activation - After the installation phase, the service worker enters the activation phase.

During activation the new version of the service wo takes control of the web page.

- The activate ~~ver~~ event ~~are~~ terminated is triggered du this phase, allowing it to clean up old caches.

Q4

P.T.O.

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

Indexed DB is a low-level API for client-side storage of significant amount of structured data including files.
Its particularly useful in service workers for several reasons.

Asynchronous and non-blocking
Service workers execute in a separate thread from the main browser thread, and indexed DB's asynchronous nature allows service workers to perform data storage operations without blocking the main thread, ensuring smooth user experience.

Persistant storage :-
It provides persistant storage, meaning data stored in indexed DB persists even when the browser is closed.

Large storage capacity :-
Indexed DB supports storing large amounts of data locally, making it suitable for applications that need to store substantial datasets.

P.T.O

iv) Structured data storage:-
Indexed DB stores data in a structured manner allowing developers to organize and query data using indexes.

In the context of service workers, indexed DB can be used to cache resources, such as, HTML, CSS, JS files etc.

—X—