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.

Since 1962

Department of Information Technology

CERTIFICATE

This is to certify that **Sonali Makhijani** of **D15-A** semester **VI,** has successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Assistant

Subject Teacher

Mrs. Kajal Jewani

Principal

Head of Department

Dr. Mrs. Shalu Chopra

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

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

Faculty Incharge : Mrs. Kajal Jewani.

Lab Teachers : Mrs. Kajal Jewani.

Email : kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

PO10) Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write

effective reports and design documentation, make effective presentations and give and receive clear instructions.

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

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

Program specific Outcomes

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

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

Lab Objectives:

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

Lab Outcomes

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

Index

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

MAD & PWA Lab
Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	11

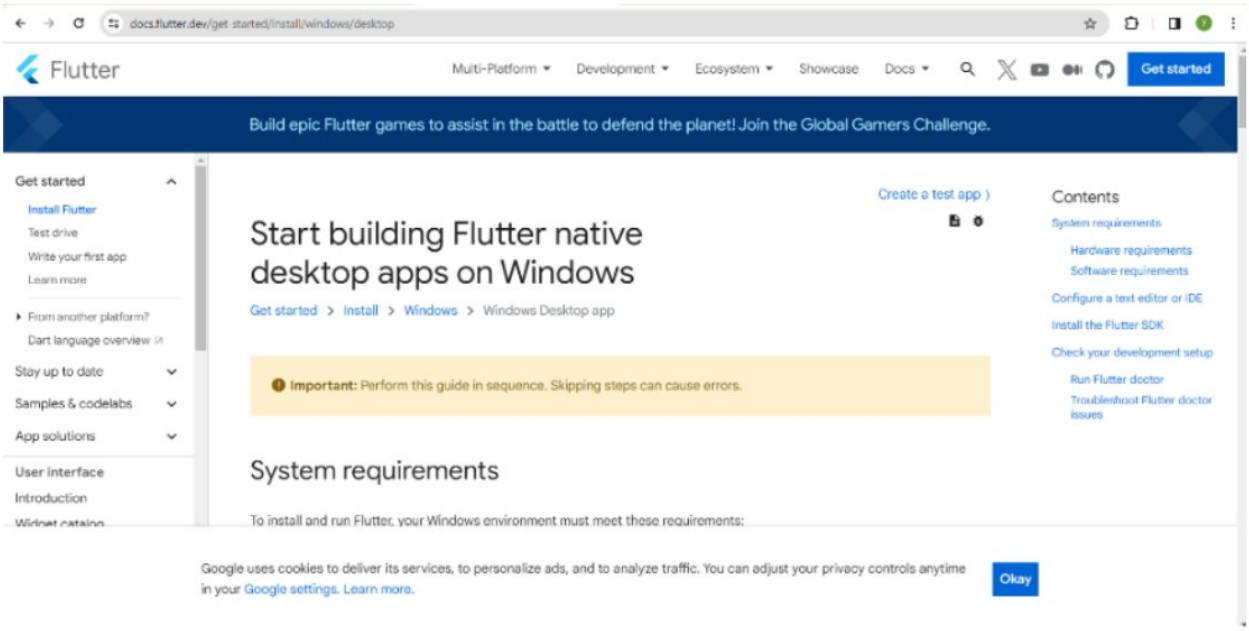
Name:- Sonali Makhijani
Subject:- MPL Lab

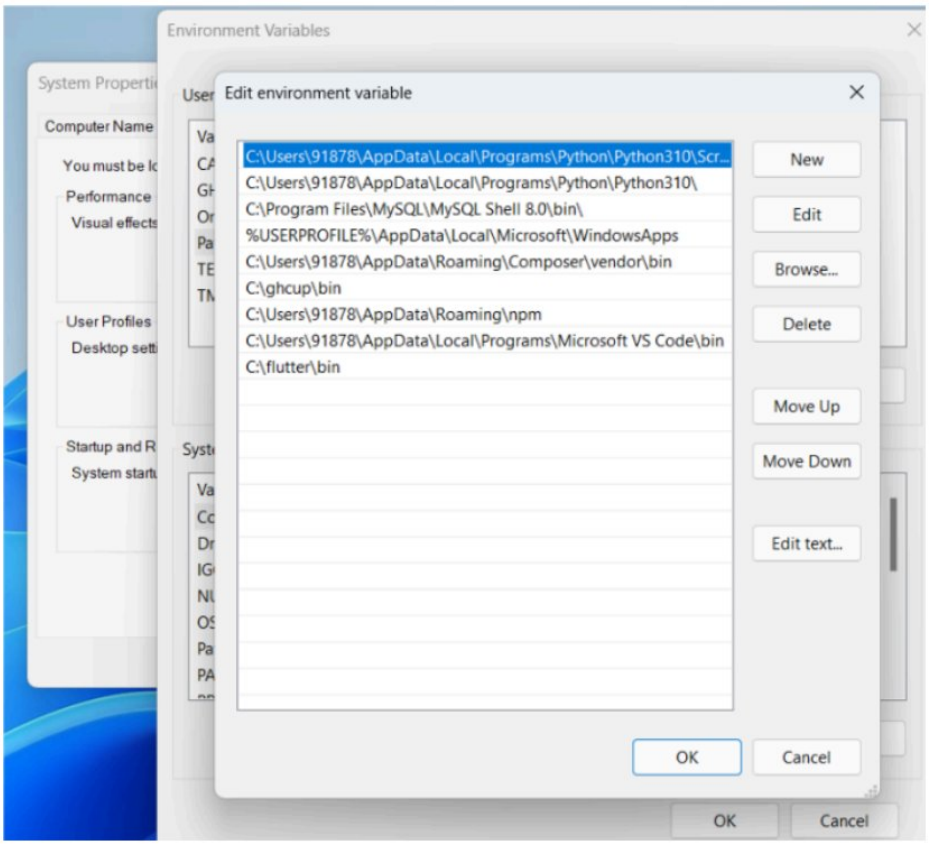
Roll No:- 34
Class:- D15 A - Batch B

Experiment - 1

Aim:- Installation and configuration of Flutter environment.

1. Installing flutter (latest version):-



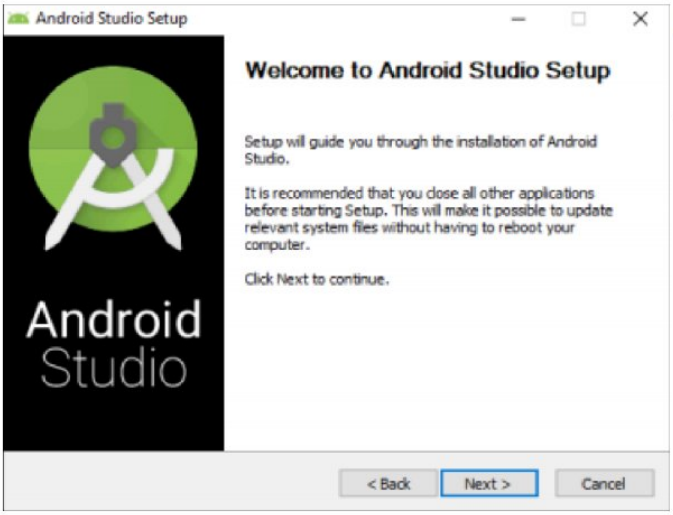


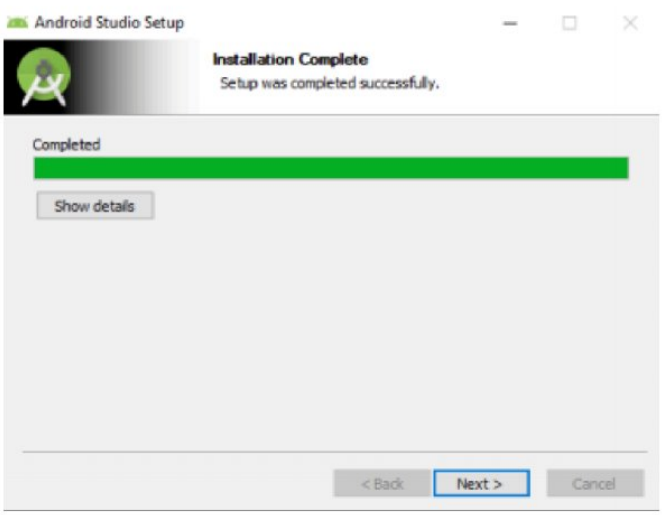
Name:- Sonali Makhijani
Subject:- MPL Lab

Roll No:- 34
Class:- D15 A - Batch B

```
C:\Users\Sonali>flutter --version
Flutter 3.16.8 • channel stable • https://github.com/flutter/flutter.git
Framework • revision 67457e669f (6 days ago) • 2024-01-16 16:22:29 -0800
Engine • revision 6e2ea58a5c
Tools • Dart 3.2.5 • DevTools 2.28.5
```

2. Installing all flutter dependencies including Android Studio:-





3. Running “flutter doctor” command to check if all dependencies are installed correctly

```
C:\Users\Sonali>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.16.8, on Microsoft Windows [Version 10.0.22621.3007]
[√] Windows Version (Installed version of Windows is version 10 or higher)
[√] Android toolchain - develop for Android devices (Android SDK version 34.0.0)
[√] Chrome - develop for the web
[√] Visual Studio - develop Windows apps (Visual Studio Build Tools 2019 16.11.29)
[√] Android Studio (version 2023.1)
[√] VS Code (version 1.85.2)
[√] Connected device (3 available)
[√] Network resources

• No issues found!
```

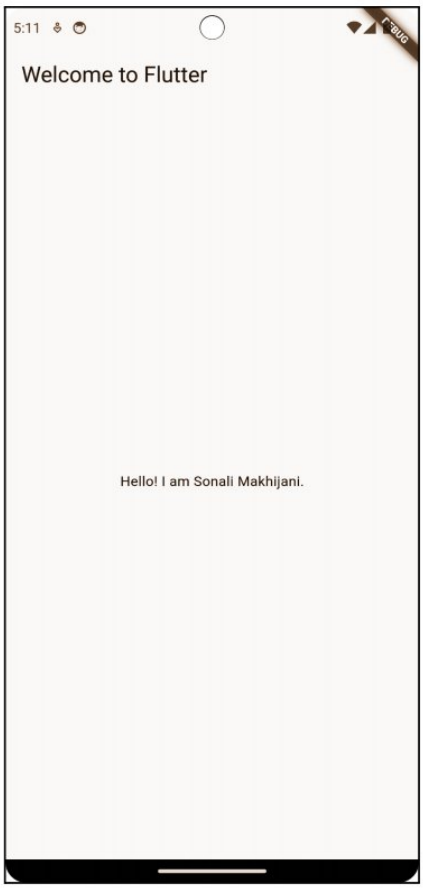
4. Running first flutter project:-

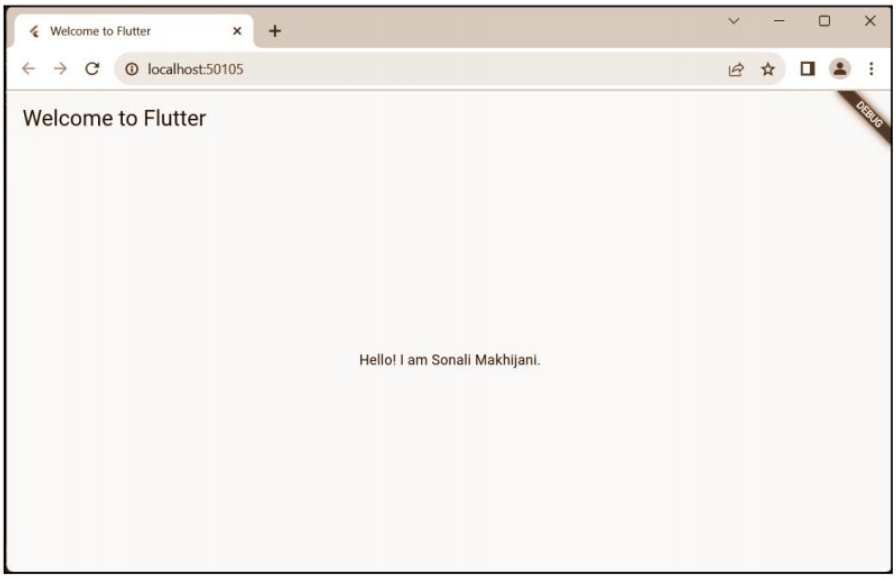
Code:- (main.dart)

```
import 'package:flutter/material.dart';
void main() {
runApp(const MyApp());
}
class MyApp extends StatelessWidget {
```

```
const MyApp({Key? key}) : super(key: key);
@override
Widget build(BuildContext context) {
return MaterialApp(
);
}
}title: 'Welcome to Flutter',
home: Scaffold(
appBar: AppBar(
title: const Text('Welcome to Flutter'),
),
body: const Center(
child: Text('Hello World'),
),
),
```

Output:-





Conclusion:- Installed flutter SDK and its dependencies like Android studio successfully and Implemented my first flutter project.

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	11

Experiment 2

Name: Sonali Makhijani
Class: D15A
Roll no:34

Aim: To design Flutter UI by including common widgets.

Theory:

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

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

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

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

Testing and Debugging Widgets:
As with any software development, testing and debugging are integral parts of the process. This section discusses strategies for testing and debugging Flutter widgets, ensuring the reliability and quality of the UI components.

Container:
The Container widget is a basic building block that can contain other widgets and isoften used to define the dimensions, padding, margin, and decoration of a UI element.

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

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

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

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

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

Experiment 2

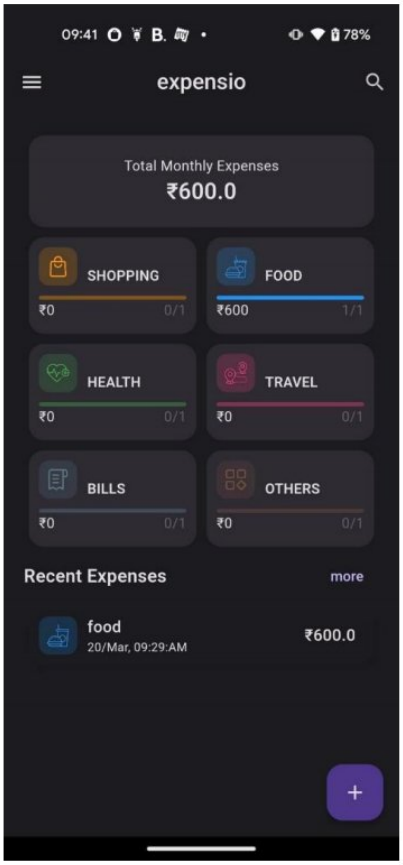
Code:

main.dart :

```
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import '/domain/bloc/expenses_bloc/expenses_bloc.dart';
import '/domain/bloc/config_bloc/config_bloc.dart';
import '/domain/bloc/auth_bloc/auth_bloc.dart';
import '/presentation/screens/splash/splash.dart';
import '/data/repositories/initialize_db.dart';
import '/core/config/app_themes.dart';
import 'core/config/firebase_options.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  await InitializeDb.initLocalDb();
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MultiBlocProvider(
      providers: [
        BlocProvider(
          create: (context) => AuthBloc()..add(AuthEventCheckLoggedInUser()),
        ),


        BlocProvider(
          create: (context) => ConfigBloc()..add(ConfigGetEvent()),
        ),
        BlocProvider(create: (context) => ExpensesBloc()),
      ],
      child: BlocBuilder<ConfigBloc, ConfigState>(
        builder: (context, state) => MaterialApp(
          title: 'Expensio',
          home: const SplashScreen(),
          debugShowCheckedModeBanner: false,
          theme: AppTheme.lightTheme,
          darkTheme: AppTheme.darkTheme,
          themeMode: context.watch<ConfigBloc>().themeMode,
        ),
      ),
    );
  }
}
```

Experiment 2

Output:



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

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	11

Experiment 3

Name: Sonali Makhijani
Div: D15A
Roll no: 34

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

Theory:

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

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

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

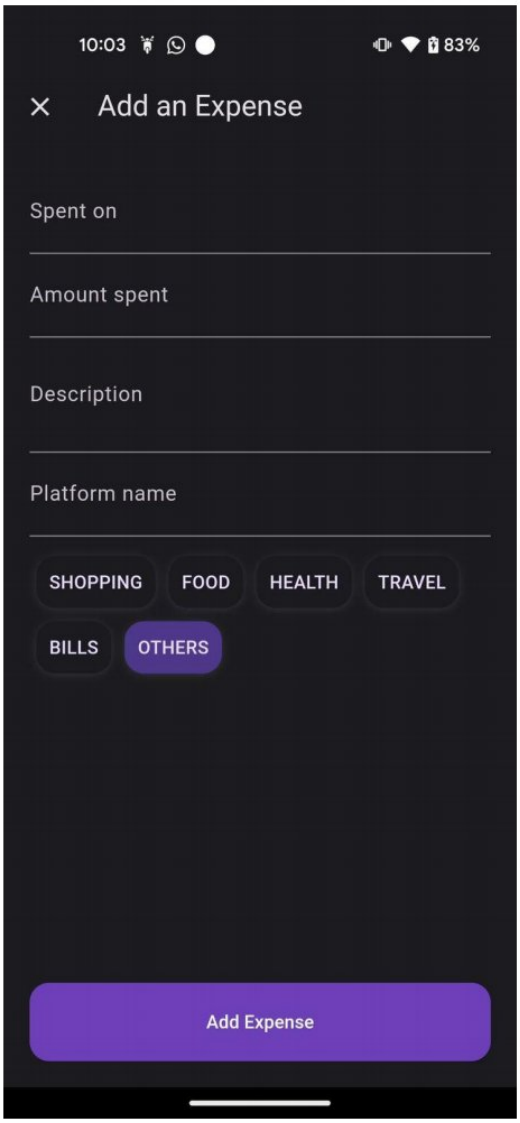
Code:

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

import '/core/constants/styles.dart';
import '/domain/bloc/expenses_bloc/expenses_bloc.dart';
import '../add_or_edit_expense/add_or_edit_expense.dart';
import '/presentation/screens/search/search_page.dart';
import '/data/repositories/common_interfaces/expenses_repo_interface.dart';
import '/presentation/screens/homescreen/components/expense_tile.dart';
import '/presentation/screens/homescreen/components/category_tile.dart';
import '/presentation/screens/homescreen/components/app_drawer.dart';
import '/presentation/widgets/error_widget.dart';
import '/presentation/widgets/loader.dart';
import '../view_expenses/view_expenses_list.dart';

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

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

class _HomeScreenState extends State<HomeScreen> {
  @override
  void didChangeDependencies() {
    context.read<ExpensesBloc>().add(ExpensesLoadEvent());
    super.didChangeDependencies();
  }

  final GlobalKey<ScaffoldState> _key = GlobalKey();
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      key: _key,
      appBar: AppBar(
        title: const Text(
          'expensio',
          style: TextStyle(
            fontWeight: FontWeight.w500,
          ),
        ),
        centerTitle: true,
        leading: IconButton(
          icon: const Icon(Icons.menu),
          onPressed: () => _key.currentState!.openDrawer(),
        ),
        actions: [
          IconButton(
            onPressed: () {
              Navigator.push(context,
                  MaterialPageRoute(builder: (context) => const SearchPage()));
            },
            icon: const Icon(Icons.search_rounded),
          )
        ],
      ),
```

```
drawer: const AppDrawer(),
floatingActionButton: FloatingActionButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (_) => const AddOrEditExpensePage(),
        fullscreenDialog: true,
        barrierDismissible: true,
      ),
    );
  },
  child: const Icon(Icons.add),
),
body: BlocBuilder<ExpensesBloc, ExpensesState>(
  builder: (context, state) {
    if (state is ExpensesLoadingState) {
      return const Loader();
    } else if (state is ExpensesErrorState) {
      return CustomErrorWidget(
        exceptionCaught: state.exception,
        onPressed: () {
          context.read<ExpensesBloc>().add(ExpensesLoadEvent());
        },
      );
    } else {
      return ListView(
        padding: const EdgeInsets.all(paddingDefault),
        physics: const BouncingScrollPhysics(),
        children: [
          Container(
            padding: const EdgeInsets.all(paddingDefault),
            margin: const EdgeInsets.all(5),
            decoration: BoxDecoration(
              color: Theme.of(context)
                  .colorScheme
                  .surfaceVariant
                  .withOpacity(0.4),
              borderRadius: borderRadiusDefault,
            ),
            child: Column(
              children: [
                const Text('Total Monthly Expenses'),
                Text(
                  "₹${context.read<ExpensesBloc>().allExpensesSum.toString()}",
                  style: const TextStyle(
                      fontSize: 22, fontWeight: FontWeight.w600),
                )
              ],
            ),
          ),
          ConstrainedBox(
            constraints:
                const BoxConstraints(maxHeight: 500, maxWidth: 300),
            child: GridView.count(
              physics: const NeverScrollableScrollPhysics(),
              shrinkWrap: true,
```

```
                crossAxisCount: 2,
                childAspectRatio: 4 / 2.4,
                children: List.generate(
                  ExpenseCategory.values.length,
                  (index) => ConstrainedBox(
                    constraints:
                        const BoxConstraints(maxHeight: 150, maxWidth: 300),
                    child: CategoryTile(index),
                  ),
                ),
              ),
            ),
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                const Text(
                  "Recent Expenses",
                  style:
                      TextStyle(fontWeight: FontWeight.w600, fontSize: 18),
                ),
                TextButton(
                  onPressed: () {
                    Navigator.push(
                      context,
                      MaterialPageRoute(
                        builder: (context) => const ViewExpenses(),
                      ),
                    );
                  },
                  child: const Text('more'),
                )
              ],
            ),
            BlocBuilder<ExpensesBloc, ExpensesState>(
              builder: (context, state) {
              return ListView.builder(
                physics: const NeverScrollableScrollPhysics(),
                shrinkWrap: true,
                itemCount:
                    context.watch<ExpensesBloc>().listOfExpenses.length,
                itemBuilder: (context, index) {
                  final expense =
                      context.watch<ExpensesBloc>().listOfExpenses[index];
                  return ExpenseTile(expense: expense);
                },
              );
            }),
          ],
        );
      }
    },
  ),
 );
 }
}
```

App UI:



Widgets used: Text, Icons,

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

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	11

Experiment 4

Name: Sonali Makhijani
Div: D15A
Roll no: 34

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

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

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

3. Validation:
    - Flutter's form widgets include built-in support for validation to ensure that user input meets specific criteria.
    - Form fields can be configured with validation functions or validators to check the correctness of user input.
    - Validators can be synchronous or asynchronous functions that return error messages if the input is invalid.

4. Submission:
    - The Form widget provides a mechanism to submit the form data once it's been filled out by the user.
    - Developers can define an onSaved callback for each form field to specify how the field's value should be processed when the form is submitted.
    - When the form is submitted, the onSaved callbacks for all form fields are invoked, allowing developers to collect, process, and submit the form data to a backend server or perform other actions.

signup_page.dart

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

import '/data/models/user_model.dart';
import '/domain/bloc/auth_bloc/auth_bloc.dart';
import '/core/utils/input_validation.dart';
import '/presentation/screens/splash/splash.dart';
import '/presentation/widgets/loader.dart';

part 'components/signup_text_fields.dart';

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

  @override
  State<Signup> createState() => _SignupState();
}

class _SignupState extends State<Signup> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  final TextEditingController _confirmPasswordController =
      TextEditingController();
  final TextEditingController _firstNameController = TextEditingController();
  final TextEditingController _lastNameController = TextEditingController();
  final _formKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    final colorScheme = Theme.of(context).colorScheme;
    return Scaffold(
      appBar: AppBar(
        title: const Text('SignUp'),
        backgroundColor: colorScheme.background,
      ),
      body: BlocConsumer<AuthBloc, AuthState>(
        listener: (context, state) {
          if (state is AuthStateCreatingUserSuccess) {
            Navigator.pop(context);
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => const SplashScreen(),
              ),
            );
          }
        },
        builder: (context, state) {
          if (state is AuthStateCreatingUser) {
            return const Loader();
          } else {
            return ListView(
              padding: const EdgeInsets.symmetric(horizontal: 20),
              children: [
```

```
                SignupTextFields(
                  emailController: _emailController,
                  passwordController: _passwordController,
                  confirmPasswordController: _confirmPasswordController,
                  firstNameController: _firstNameController,
                  lastNameController: _lastNameController,
                  formKey: _formKey,
                ),
                const SizedBox(height: 30),
                ElevatedButton(
                  onPressed: () {
                    FocusScope.of(context).unfocus();
                    if (_formKey.currentState?.validate() ?? false) {
                      final UserModel user = UserModel()
                        ..email = _emailController.text
                        ..password = _passwordController.text
                        ..firstName = _firstNameController.text
                        ..lastName = _lastNameController.text;
                      context
                          .read<AuthBloc>()
                          .add(AuthEventCreateNewUser(user));
                    }
                  },
                  child: const Text('SignUp'),
                ),
                Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    const Text('Already have an account'),
                    TextButton(
                      onPressed: () => Navigator.pop(context),
                      child: const Text('Login'),
                    ),
                  ],
                )
              ],
            );
          }
        },
      ),
    );
  }
}
```

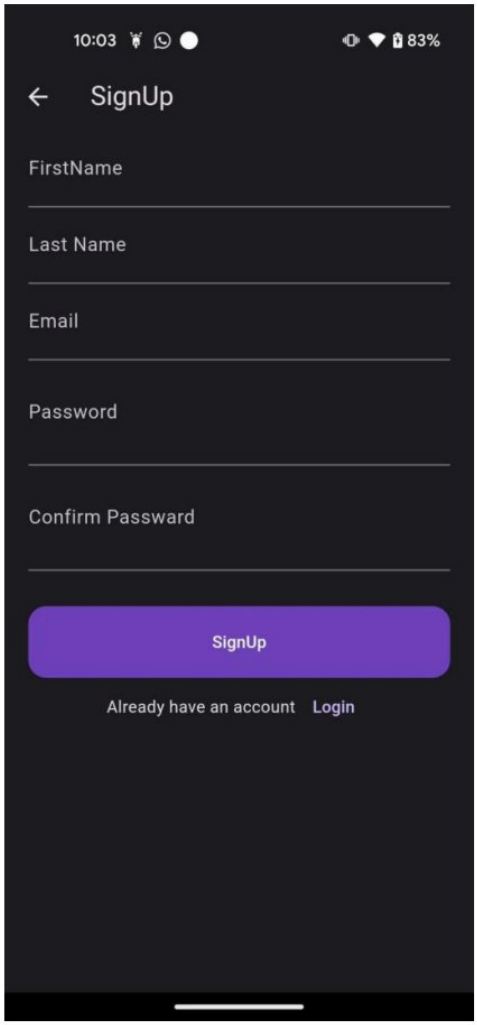
login_page.dart

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

import '/domain/bloc/auth_bloc/auth_bloc.dart';
import '/presentation/widgets/loader.dart';
import '/presentation/screens/authentication/signup.dart';
import '/presentation/screens/homescreen/homescreen.dart';
import '/core/constants/app_assets.dart';
import '/core/utils/input_validation.dart';
```

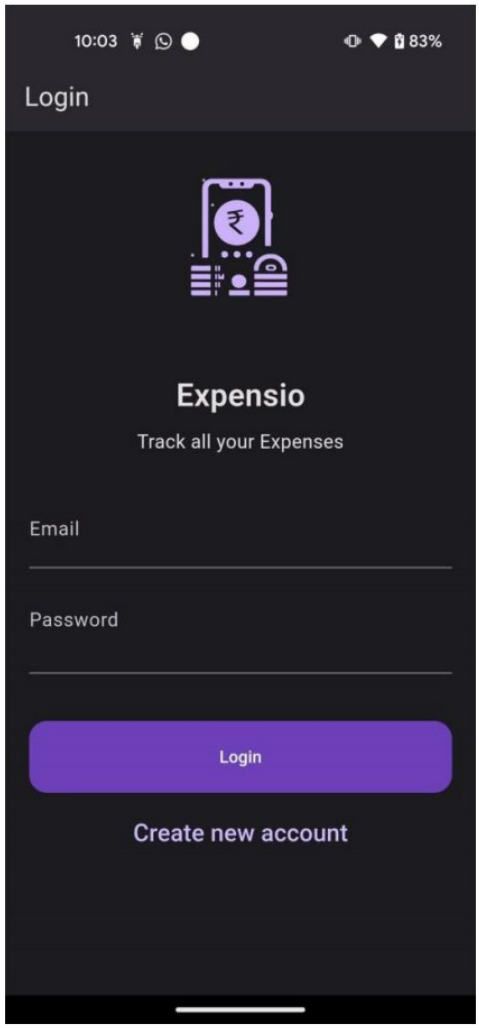
```dart
part 'components/login_icon_texts.dart';
part 'components/login_bottom_buttons.dart';

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

  @override
  State<Login> createState() => _LoginState();
}

class _LoginState extends State<Login> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  final _formKey = GlobalKey<FormState>();
  bool _isPasswordVisible = true;

  @override
  Widget build(BuildContext context) {
    final colorScheme = Theme.of(context).colorScheme;
    return BlocConsumer<AuthBloc, AuthState>(
      listener: (context, state) {
        if (state is AuthStateLoginSuccess) {
          Navigator.pushReplacement(
            context,
            MaterialPageRoute(
              builder: (context) => const HomeScreen(),
            ),
          );
        }
      },
      builder: (context, state) {
        return Scaffold(
          appBar: AppBar(
            title: const Text("Login"),
            backgroundColor: colorScheme.background,
          ),
          body: state is AuthStateLoggingIn
              ? const Loader()
              : GestureDetector(
                  onTap: () => FocusScope.of(context).unfocus(),
                  child: Form(
                    key: _formKey,
                    child: ListView(
                      physics: const BouncingScrollPhysics(),
                      padding: const EdgeInsets.symmetric(horizontal: 20),
                      children: [
                        const LoginIconAndTexts(),
                        TextFormField(
                          controller: _emailController,
                          validator: InputValidator.email,
                          decoration: const InputDecoration(
                            label: Text('Email'),
                            hintText: 'Enter your Email',
                          ),
                        ),
```

```
                    TextFormField(
                      controller: _passwordController,
                      obscureText: _isPasswordVisible,
                      validator: InputValidator.password,
                      decoration: InputDecoration(
                        suffix: IconButton(
                          icon: Icon(
                            _isPasswordVisible
                                ? Icons.visibility
                                : Icons.visibility_off,
                          ),
                          onPressed: () => setState(
                            () => _isPasswordVisible = !_isPasswordVisible,
                          ),
                        ),
                        label: const Text('Password'),
                        hintText: 'Enter your password',
                      ),
                    ),
                    LoginBottomButtons(
                      onLoginPressed: () {
                        FocusScope.of(context).unfocus();
                        if (_formKey.currentState!.validate()) {
                          context.read<AuthBloc>().add(
                                AuthEventLoginWithEmailAndPassword(
                                  email: _emailController.text,
                                  password: _passwordController.text,
                                ),
                              );
                          _emailController.clear();
                          _passwordController.clear();
                        }
                      },
                      onCreateNewAccountPressed: () {
                        Navigator.push(
                          context,
                          MaterialPageRoute(
                            builder: (_) => const Signup(),
                          ),
                        );
                      },
                    ),
                    const SizedBox(height: 20)
                  ],
                ),
              ),
            ),
          );
        },
      );
    }
}
```

App UI:



Signup page



Login page

Widgets used: Form Widget, Form Widget Fields

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

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	11

Experiment 5

Name:Sonali Makhijani
Div: D15A
Roll no: 34

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

Theory:

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

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

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

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

4. Gesture Detection:

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

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

home_screen.dart

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

import '/core/constants/styles.dart';
import '/domain/bloc/expenses_bloc/expenses_bloc.dart';
import '../add_or_edit_expense/add_or_edit_expense.dart';
import '/presentation/screens/search/search_page.dart';
import '/data/repositories/common_interfaces/expenses_repo_interface.dart';
import '/presentation/screens/homescreen/components/expense_tile.dart';
import '/presentation/screens/homescreen/components/category_tile.dart';
import '/presentation/screens/homescreen/components/app_drawer.dart';
import '/presentation/widgets/error_widget.dart';
import '/presentation/widgets/loader.dart';
import '../view_expenses/view_expenses_list.dart';

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

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

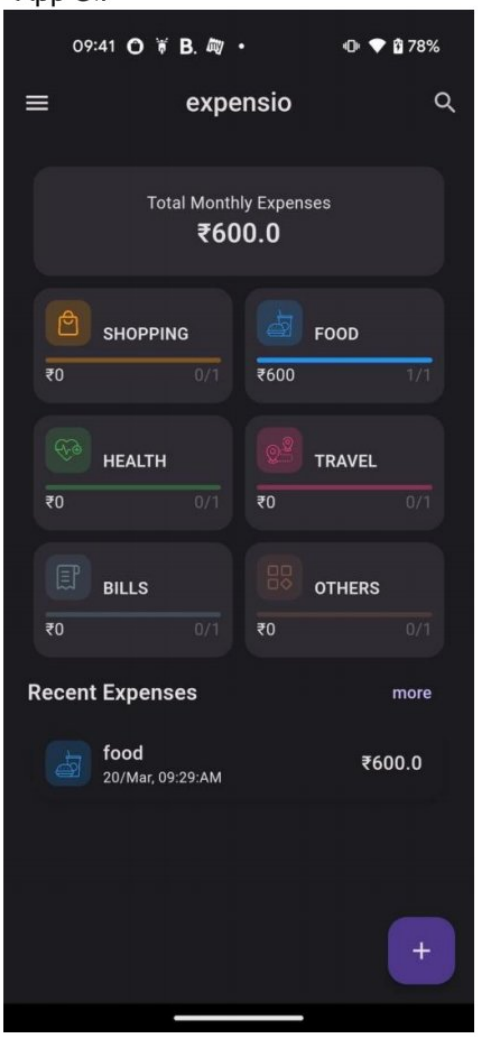
```
class _HomeScreenState extends State<HomeScreen> {
  @override
  void didChangeDependencies() {
    context.read<ExpensesBloc>().add(ExpensesLoadEvent());
    super.didChangeDependencies();
  }

  final GlobalKey<ScaffoldState> _key = GlobalKey();
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      key: _key,
      appBar: AppBar(
        title: const Text(
          'expensio',
          style: TextStyle(
            fontWeight: FontWeight.w500,
          ),
        ),
        centerTitle: true,
        leading: IconButton(
          icon: const Icon(Icons.menu),
          onPressed: () => _key.currentState!.openDrawer(),
        ),
        actions: [
          IconButton(
            onPressed: () {
              Navigator.push(context,
                  MaterialPageRoute(builder: (context) => const SearchPage()));
            },
            icon: const Icon(Icons.search_rounded),
          )
        ],
      ),
      drawer: const AppDrawer(),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          Navigator.push(
            context,
```

```
      MaterialPageRoute(
        builder: (_) => const AddOrEditExpensePage(),
        fullscreenDialog: true,
        barrierDismissible: true,
      ),
    );
  },
  child: const Icon(Icons.add),
),
body: BlocBuilder<ExpensesBloc, ExpensesState>(
  builder: (context, state) {
    if (state is ExpensesLoadingState) {
      return const Loader();
    } else if (state is ExpensesErrorState) {
      return CustomErrorWidget(
        exceptionCaught: state.exception,
        onPressed: () {
          context.read<ExpensesBloc>().add(ExpensesLoadEvent());
        },
      );
    } else {
      return ListView(
        padding: const EdgeInsets.all(paddingDefault),
        physics: const BouncingScrollPhysics(),
        children: [
          Container(
            padding: const EdgeInsets.all(paddingDefault),
            margin: const EdgeInsets.all(5),
            decoration: BoxDecoration(
              color: Theme.of(context)
                  .colorScheme
                  .surfaceVariant
                  .withOpacity(0.4),
              borderRadius: borderRadiusDefault,
            ),
            child: Column(
              children: [
                const Text('Total Monthly Expenses'),
                Text(
                  "₹${context.read<ExpensesBloc>().allExpensesSum.toString()}",
                  style: const TextStyle(
                      fontSize: 22, fontWeight: FontWeight.w600),
                )
```

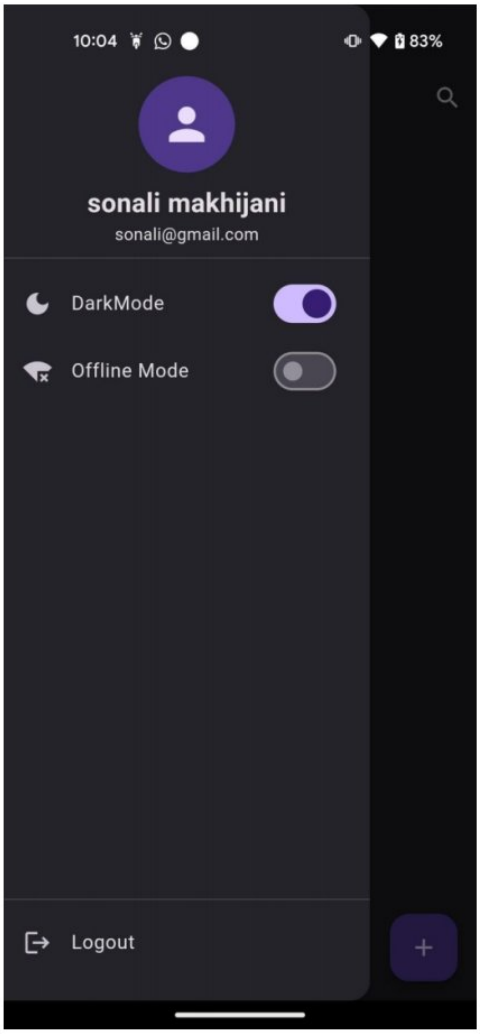
```
    ],
  ),
),
ConstrainedBox(
  constraints:
      const BoxConstraints(maxHeight: 500, maxWidth: 300),
  child: GridView.count(
    physics: const NeverScrollableScrollPhysics(),
    shrinkWrap: true,
    crossAxisCount: 2,
    childAspectRatio: 4 / 2.4,
    children: List.generate(
      ExpenseCategory.values.length,
      (index) => ConstrainedBox(
        constraints:
            const BoxConstraints(maxHeight: 150, maxWidth: 300),
        child: CategoryTile(index),
      ),
    ),
  ),
),
Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: [
    const Text(
      "Recent Expenses",
      style:
          TextStyle(fontWeight: FontWeight.w600, fontSize: 18),
    ),
    TextButton(
      onPressed: () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => const ViewExpenses(),
          ),
        );
      },
      child: const Text('more'),
    )
  ],
),
BlocBuilder<ExpensesBloc, ExpensesState>(
    builder: (context, state) {
```

```
                return ListView.builder(
                  physics: const NeverScrollableScrollPhysics(),
                  shrinkWrap: true,
                  itemCount:
                      context.watch<ExpensesBloc>().listOfExpenses.length,
                  itemBuilder: (context, index) {
                    final expense =
                        context.watch<ExpensesBloc>().listOfExpenses[index];
                    return ExpenseTile(expense: expense);
                  },
                );
              }),
            ],
          );
        }
      },
    ),
  );
 }
}
```

App UI:



Home Page



Side Bar

Widgets used: Text, Side nav bar, Icons

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

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	11

Experiment 6

Name: Sonali Makhijani
Div: D15A
Roll no: 34

Aim: To Connect Flutter UI with Firebase

Theory:

FlutterFire is a set of Flutter plugins that enable Flutter developers to integrate their applications with various Firebase services. Firebase is a comprehensive mobile and web application development platform provided by Google. FlutterFire is specifically designed to provide Flutter developers with a seamless way to interact with Firebase services.

Key features of FlutterFire include:

1. Firebase Authentication: FlutterFire provides plugins to easily integrate Firebase Authentication, allowing developers to implement user sign-up, sign-in, and password recovery features in their Flutter applications. Firebase supports various authentication methods, including email/password, Google Sign-In, Facebook Sign-In, and more.

2. Cloud Firestore and Realtime Database: FlutterFire supports both Cloud Firestore and Firebase Realtime Database, enabling developers to store and retrieve data in real-time. Firestore is a NoSQL document database, while Realtime Database is a JSON-based database.

3. Cloud Functions: Developers can deploy serverless functions using Cloud Functions for Firebase, and FlutterFire allows Flutter apps to trigger and interact with these functions.

4. Cloud Storage: FlutterFire supports Firebase Cloud Storage, allowing developers to upload, download, and manage files in the cloud. This is useful for handling user-generated content, such as images or videos.

5. Firebase Cloud Messaging (FCM): FCM enables developers to send push notifications to their Flutter applications. FlutterFire provides plugins for integrating FCM and handling push notifications.

6. Firebase Performance Monitoring: Developers can monitor the performance of their Flutter applications using Firebase Performance Monitoring. This includes measuring app startup time, screen rendering, and network performance.

7. Firebase Analytics: FlutterFire includes plugins for integrating Firebase Analytics, enabling developers to gain insights into user behavior and app usage.

8. Firebase Remote Config: FlutterFire supports Firebase Remote Config, allowing developers to remotely configure app behavior without publishing updates. This is useful for A/B testing and feature toggling.

9. Firebase Crashlytics: FlutterFire includes support for Firebase Crashlytics, providing real-time crash reporting to help developers identify and fix issues quickly.

10. Firebase AdMob: FlutterFire includes AdMob plugins for integrating advertisements into Flutter applications using Firebase AdMob.

Firebase_options.dart

```
// File generated by FlutterFire CLI.
// ignore_for_file: lines_longer_than_80_chars, avoid_classes_with_only_static_members
import 'package:firebase_core/firebase_core.dart' show FirebaseOptions;
import 'package:flutter/foundation.dart'
    show defaultTargetPlatform, kIsWeb, TargetPlatform;

/// Default [FirebaseOptions] for use with your Firebase apps.
///
/// Example:
/// dart
/// import 'firebase_options.dart';
/// // ...
/// await Firebase.initializeApp(
///   options: DefaultFirebaseOptions.currentPlatform,
/// );
///
class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
      throw UnsupportedError(
        'DefaultFirebaseOptions have not been configured for web - '
        'you can reconfigure this by running the FlutterFire CLI again.',
      );
    }
    switch (defaultTargetPlatform) {
      case TargetPlatform.android:
        return android;
      case TargetPlatform.iOS:
        return ios;
      case TargetPlatform.macOS:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for macos - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      case TargetPlatform.windows:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for windows - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      case TargetPlatform.linux:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for linux - '
          'you can reconfigure this by running the FlutterFire CLI again.',
```

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.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

Name : Sonali Makhijani
Div : D15A
Roll No : 34
Batch B

PWA EXPERIMENT 7

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

Theory:-

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

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

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

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

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

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

time for the end-users and helps the brands improve the user engagement and retention rate, which eventually adds value to their business.

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

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

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

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

Pros and cons of the Progressive Web App The main features are:
Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.

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

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

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

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

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

Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.

Installable — They allow the user to “save” the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to face all the steps and problems related to the use of the app store.

Linkable — Easily shared via URL without complex installations.

Offline — Once more it is about putting the user before everything, avoiding the usual error message in case of weak or no connection

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

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

Folder Structure and icon size

plant 512 X 512.png

Plant 192 X 192.png

Index.html

```
<!DOCTYPE html>

<html>



<head>
   <meta name="apple-mobile-web-app-status-bar" content="#aa7700">

   <meta name="theme-color" content="black">

   <link rel="manifest" href="manifest.json">

   <script src="service-worker.js"></script>

   <title>

       Index

   </title>

   <link rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">



   <!--jQuery library-->

   <script
```

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

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

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

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

```
                   <li>
                       <a href="signup.html">
                           <span class="glyphicon glyphicon-user" /> Sign-Up </a>
                   </li>
                   <li>
                       <a href="login.html">
                           <span class="glyphicon glyphicon-log-in" /> Login </a>
                   </li>
```

```
            </ul>
        </div>


    </div>


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

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

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




</body>


</html>
```

Manifest.json

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

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

Service-worker.json

```
// service-worker.js


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



self.addEventListener('install', function(event) {
 event.waitUntil(
   caches.open(CACHE_NAME)
     .then(function(cache) {
       console.log('Opened cache');
       return cache.addAll(urlsToCache)
         .catch(function(error) {
```

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

        });
    })
```

```
);
});

self.addEventListener('activate', function(event) {
 // Perform activation steps
 event.waitUntil(
   caches.keys().then(function(cacheNames) {

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

           return caches.delete(cacheName);
         }
       })
     );
   })
 );
});


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


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

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

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

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

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


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

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

Product.html

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

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


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


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

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

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

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

                </div>


            </nav>


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


            <div class ="jumbotron">
```

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

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

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

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

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

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

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

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

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

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

</div>

</div>

<div class="row text-center">

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

```
            </div>
            </div>




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



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

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



</div>


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

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



</div>
</div>




<div class="row text-center">



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


</div>



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


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

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


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


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




      </body>
   </html>
```

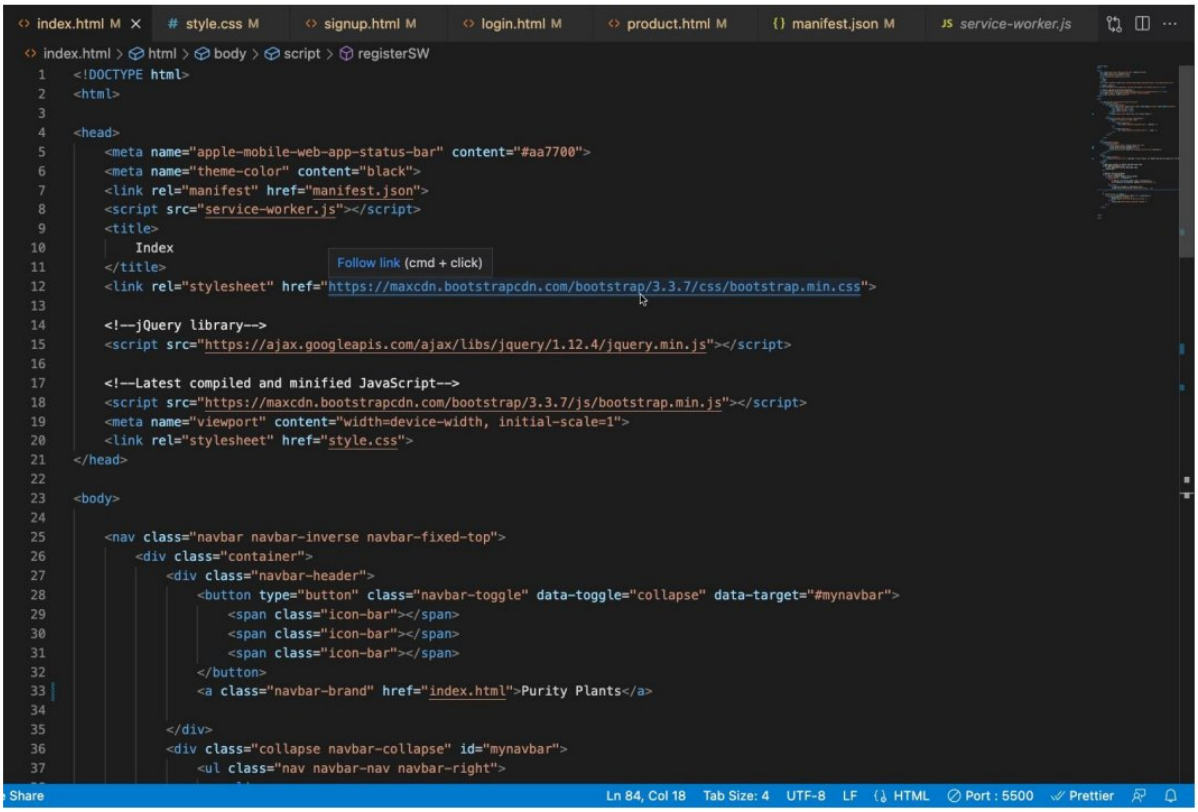
Style.css

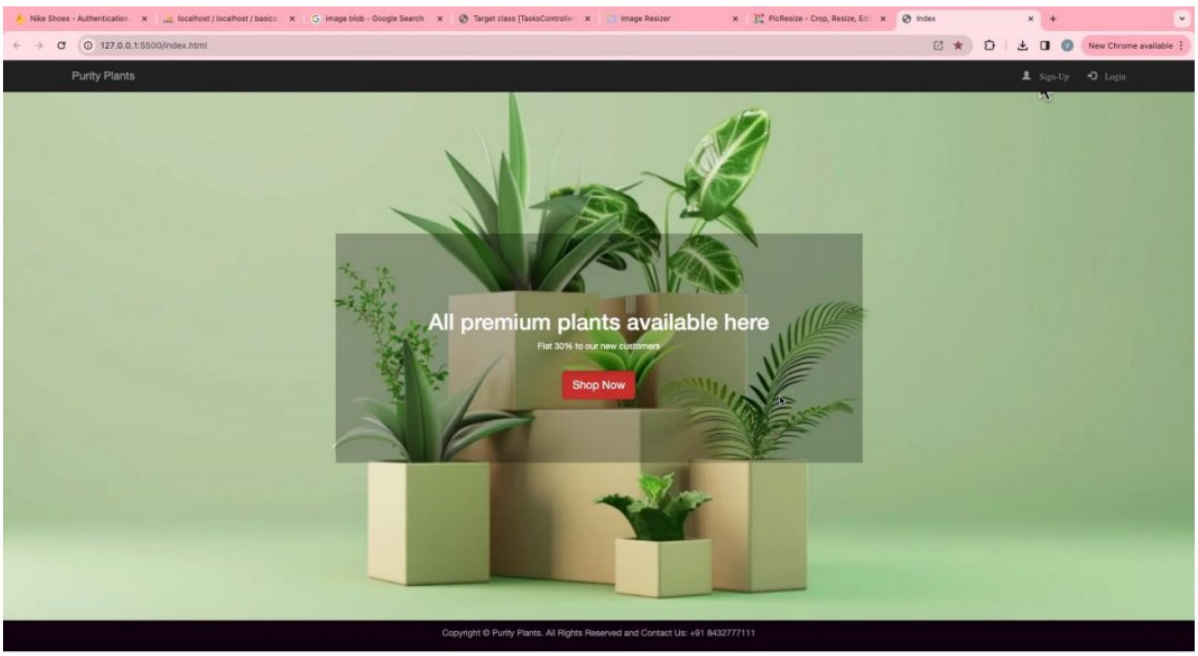
```
.banner-image
{padding-top: 75px;
padding-bottom: 50px; text-
align: center;
color: #f8f8f8;
background: url(homepage.png) no-repeat center center;
background-size: cover;
}
```

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


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


}
```

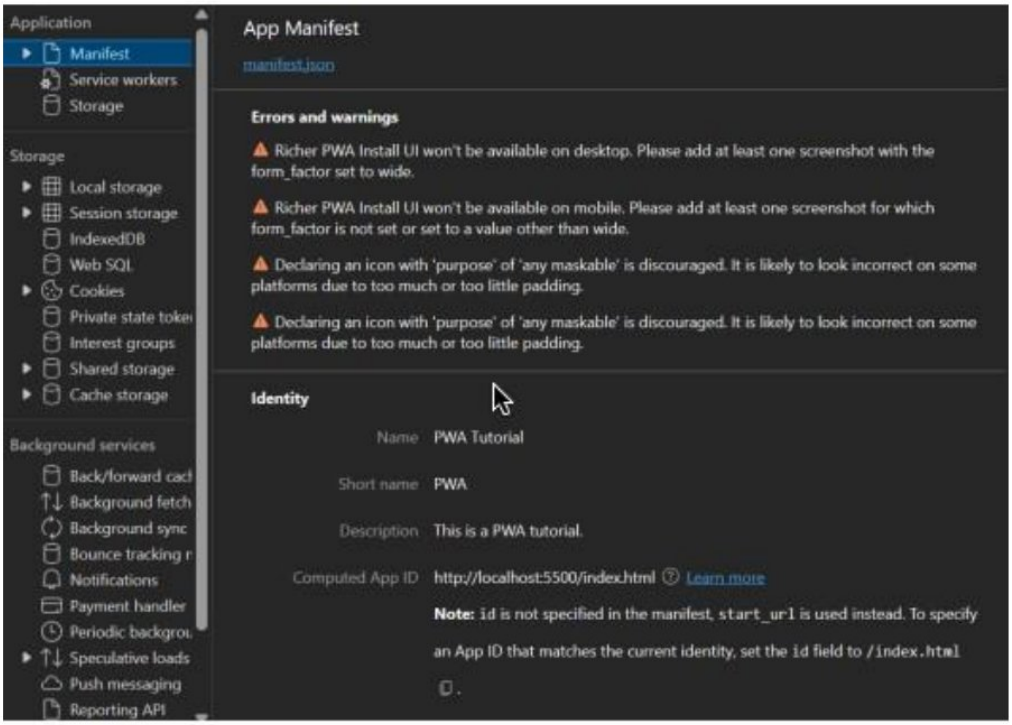
Starting the Server

:

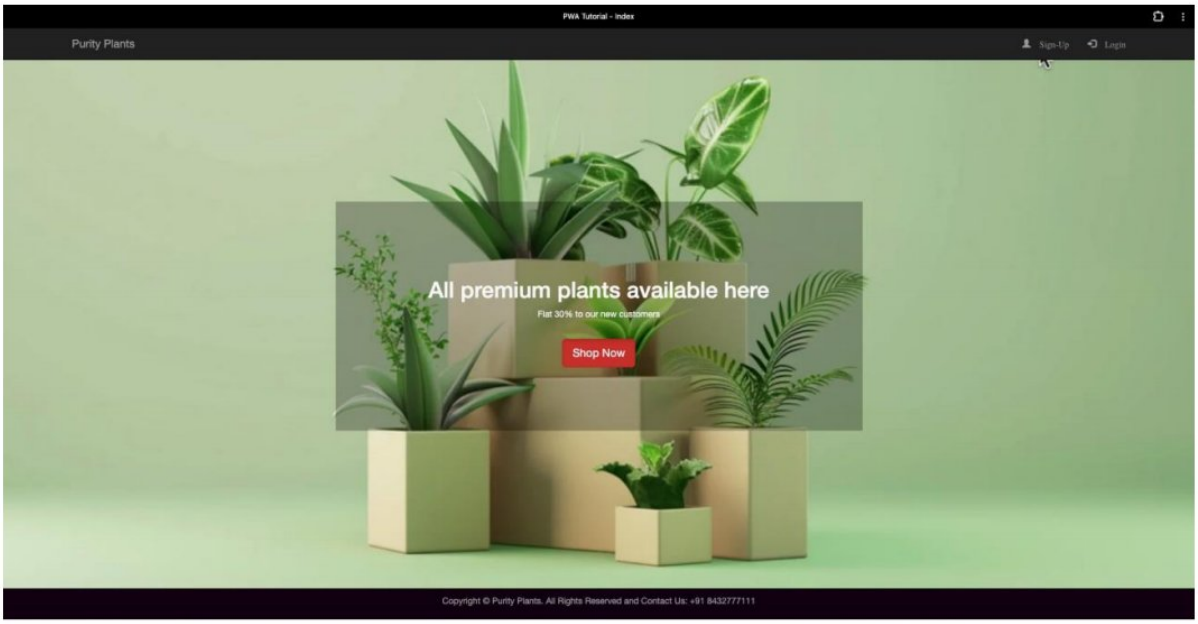




Now go to developer options -> Application->Manifest



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



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

MAD & PWA Lab
Journal

Experiment No.	08
Experiment Title.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA
Roll No.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Name: Sonali Makhijani
Division: D15A
RollNo: 34
Batch: B

Experiment No 8

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

Service Worker

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

Things to note about Service Worker:

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

What can we do with Service Workers?

- You can dominate **Network Traffic**

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

- You can **Cache**

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

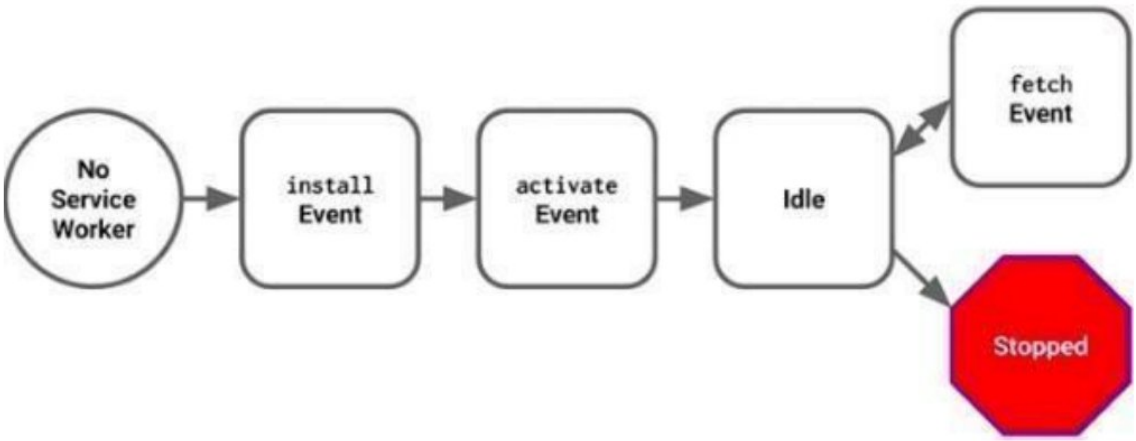
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can't we do with Service Workers?

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

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

Service Worker Cycle



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

- Registration
- Installation
- Activation

Registration

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

main.js

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

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

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

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

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

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

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

main.js

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

Installation

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

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

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

Activation

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

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

service-worker.js

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

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

Code in service-worker.js

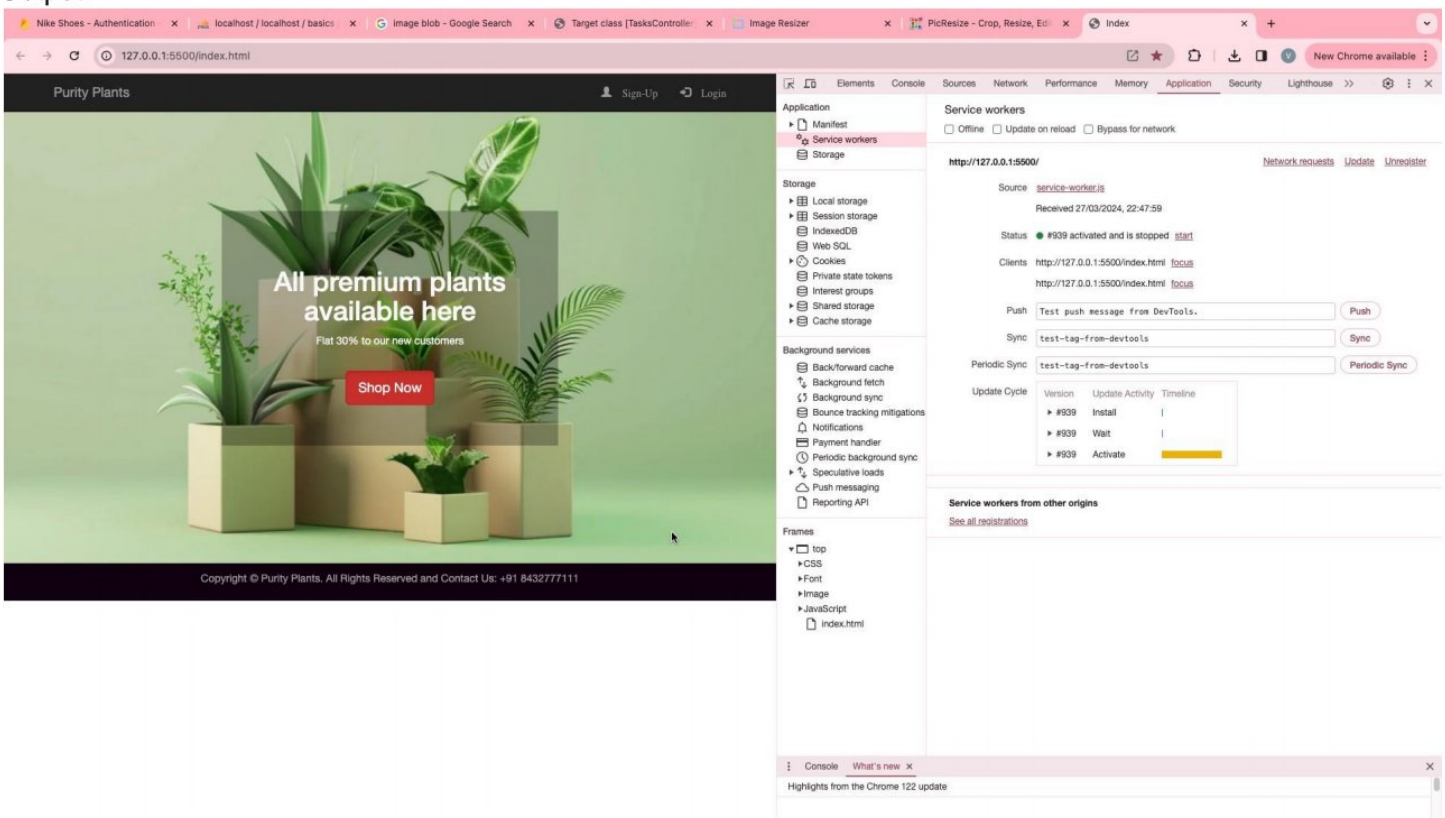
```
self.addEventListener('install', function(event) {

 event.waitUntil(
   caches.open(CACHE_NAME)

     .then(function(cache) {

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

         .catch(function(error) {

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

     })
 );

});



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

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

     return Promise.all(

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

           return caches.delete(cacheName);

         }
       })

     );

   })
 );

});
```

Code in index.html

```
<script>

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

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

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

Output:



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

MAD & PWA Lab
Journal

Experiment No.	09
Experiment Title.	To implement Service worker events like fetch, sync and push for E-commerce PWA
Roll No.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Name: Sonali Makhijani
Division: D15A
Roll No:34
Batch : B

Experiment No 9

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

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

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

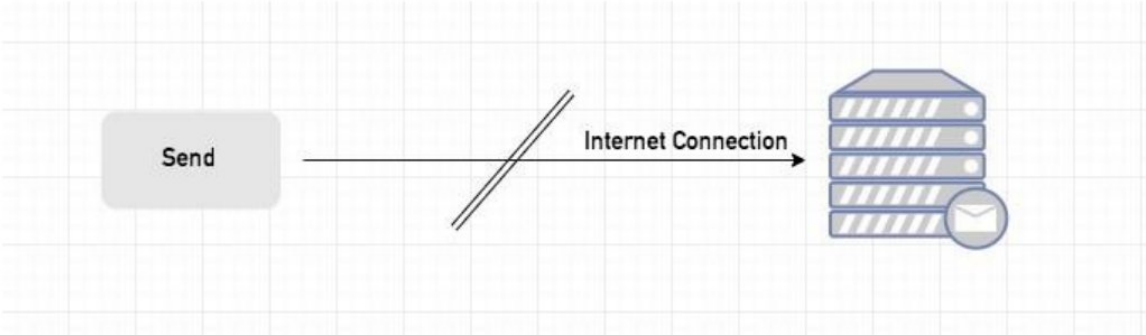
But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

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

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

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

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

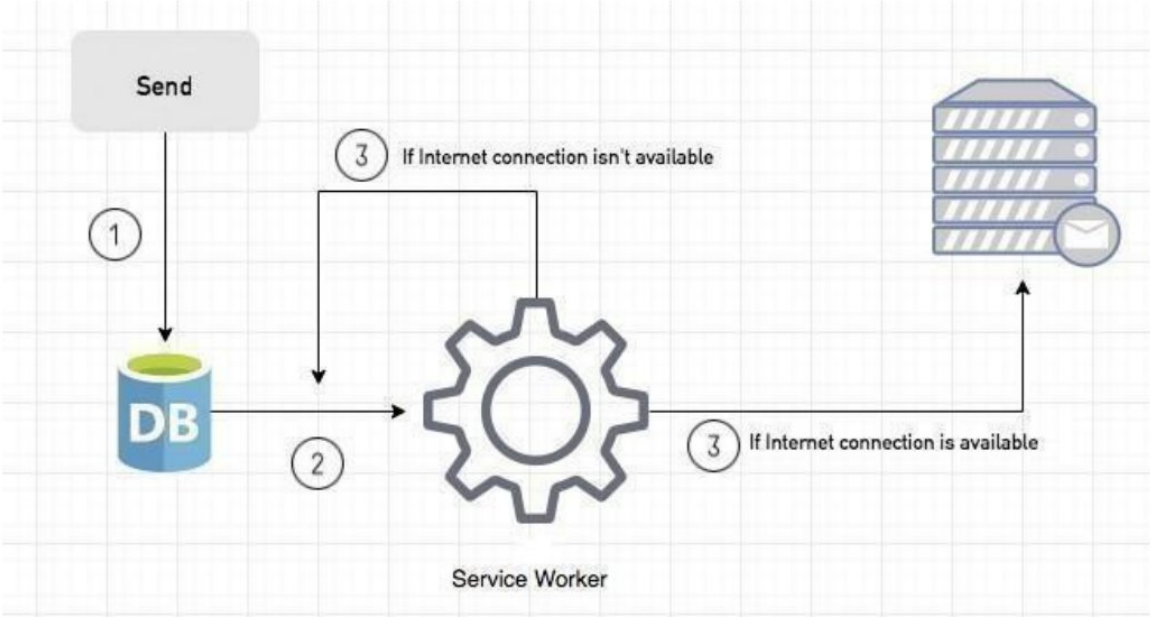
Sync Event

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

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



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



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

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

Event Listener for Background Sync Registration

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

Event Listener for sw.js

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

Push Event

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

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

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

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

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

Theory:
Service Worker

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

Things to note about Service Worker:

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

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

Fetch Event

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

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

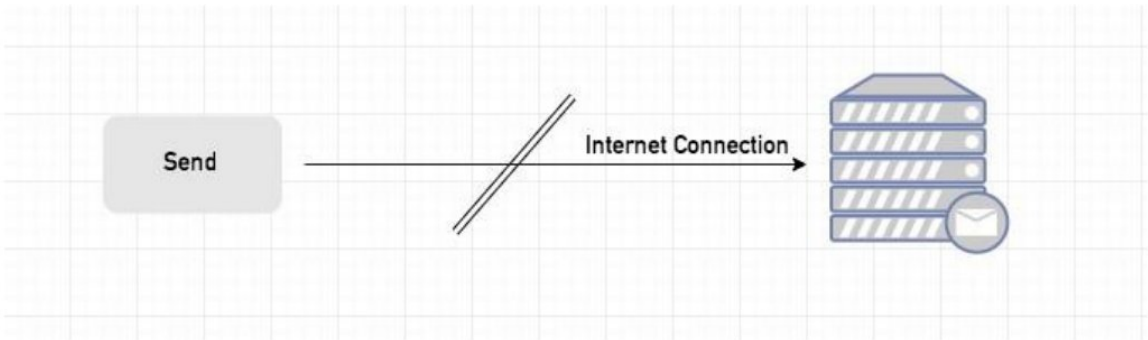
- **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
- **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed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

Sync Event

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

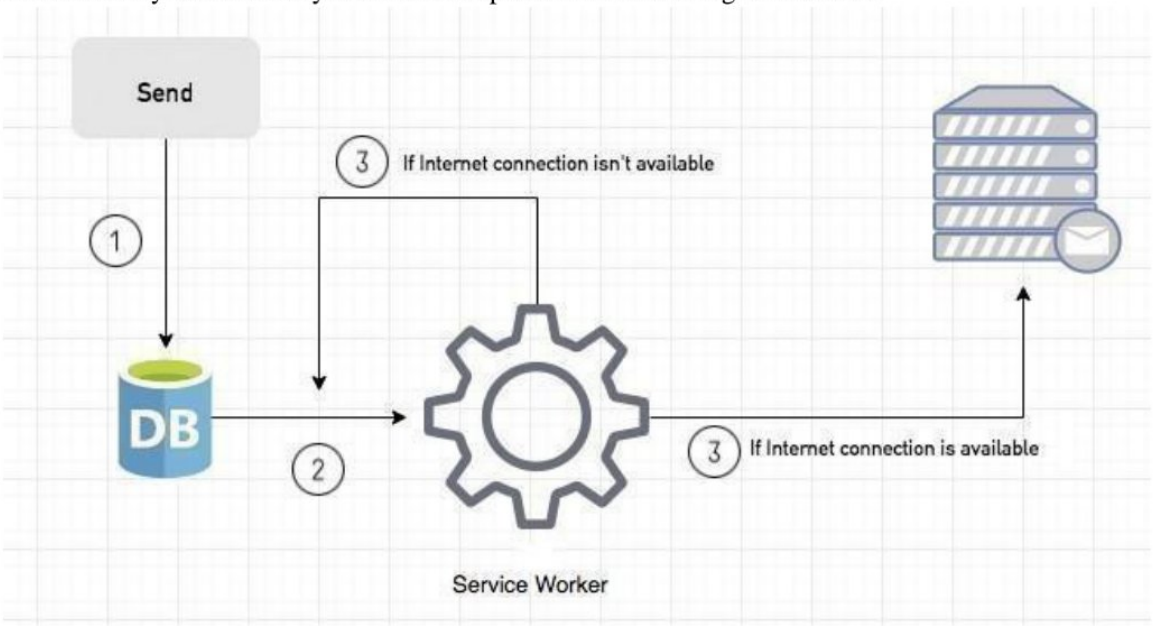
Here is a job for the Background Sync.

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




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

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



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

Event Listener for Background Sync Registration

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

Event Listener for sw.js

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

Push Event

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

We can check in the following example.

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

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

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

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

Code:

In index.html

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

```
// service-worker.js



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

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



self.addEventListener('install', function(event) {
 event.waitUntil(
   caches.open(CACHE_NAME)
     .then(function(cache) {
       console.log('Opened cache');
       return cache.addAll(urlsToCache)
         .catch(function(error) {
           console.error('Cache.addAll error:', error);
         });
     })
```

```
            });
        }
```

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

```
 );
});


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


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


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

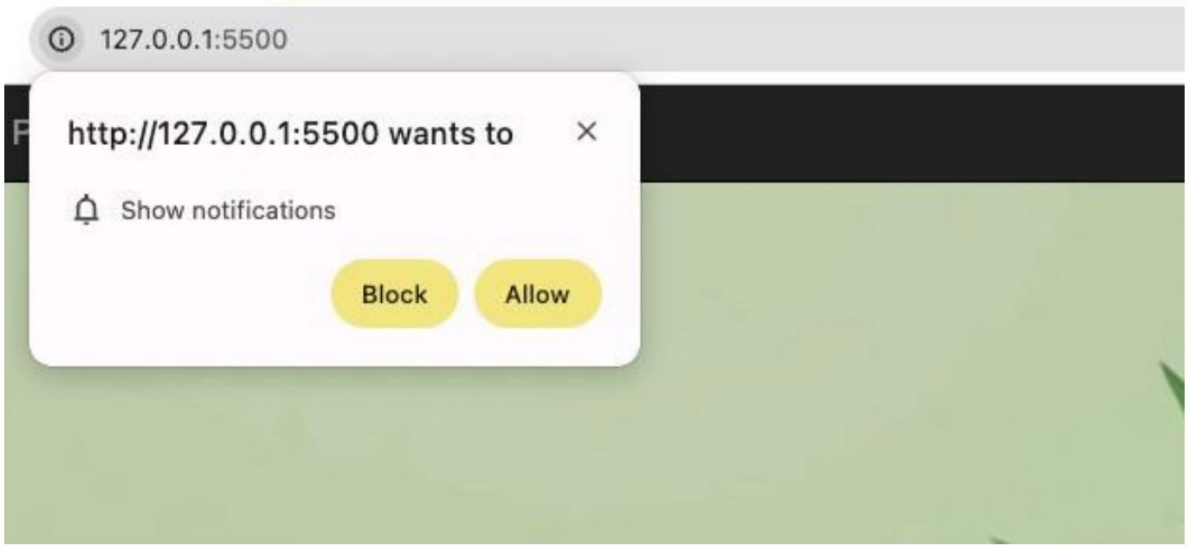
```
// Push event listener
self.addEventListener("push", function (event) {
 if (event && event.data) {
   try {
     var data = event.data.json();
     if (data && data.method === "pushMessage") {
```

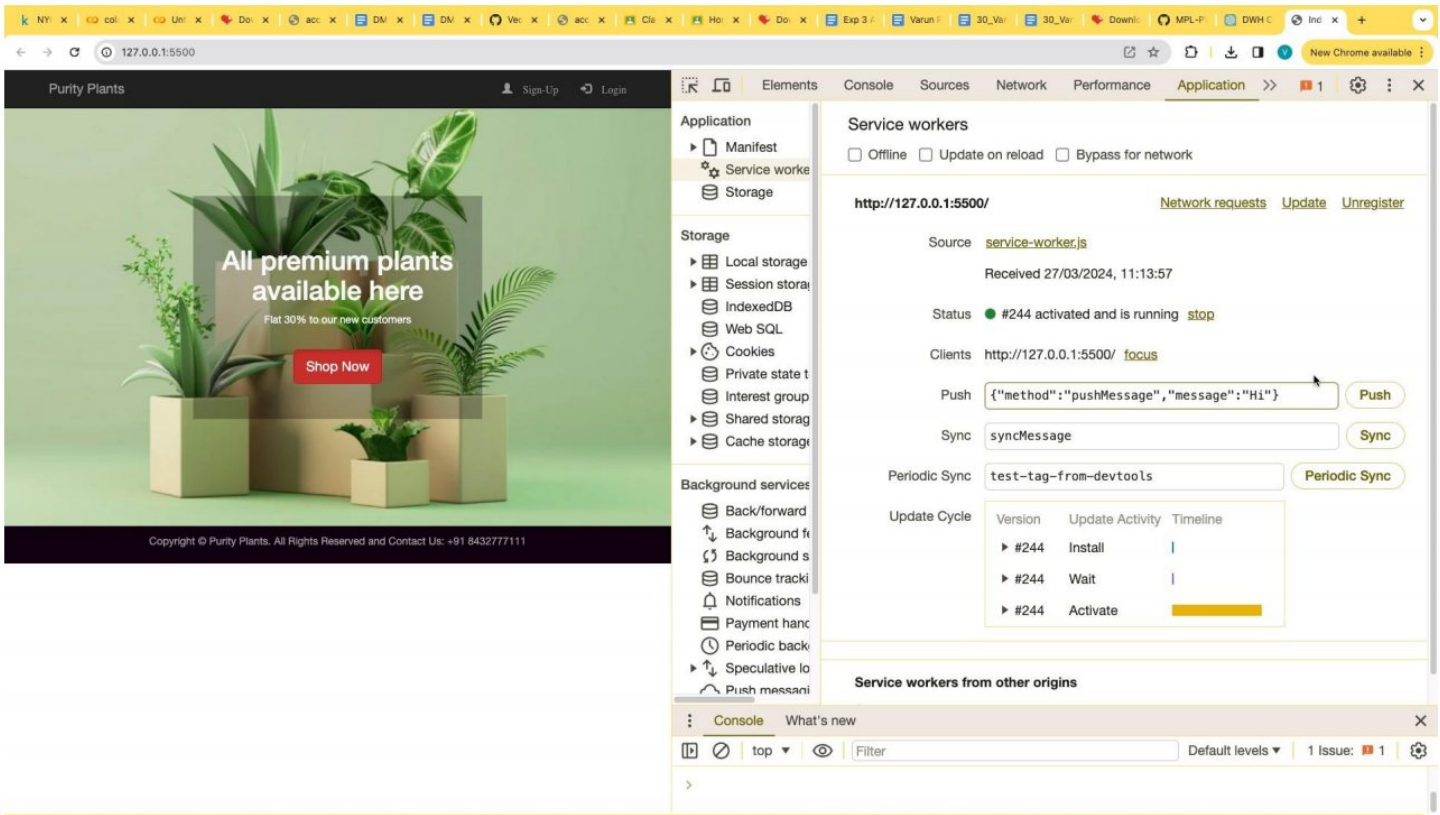
```
      console.log("Push notification sent");
      self.registration.showNotification("Ecommerce website", { body:
data.message });
    }
  } catch (error) {
    console.error("Error parsing push data:", error);
  }
 }
});


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


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

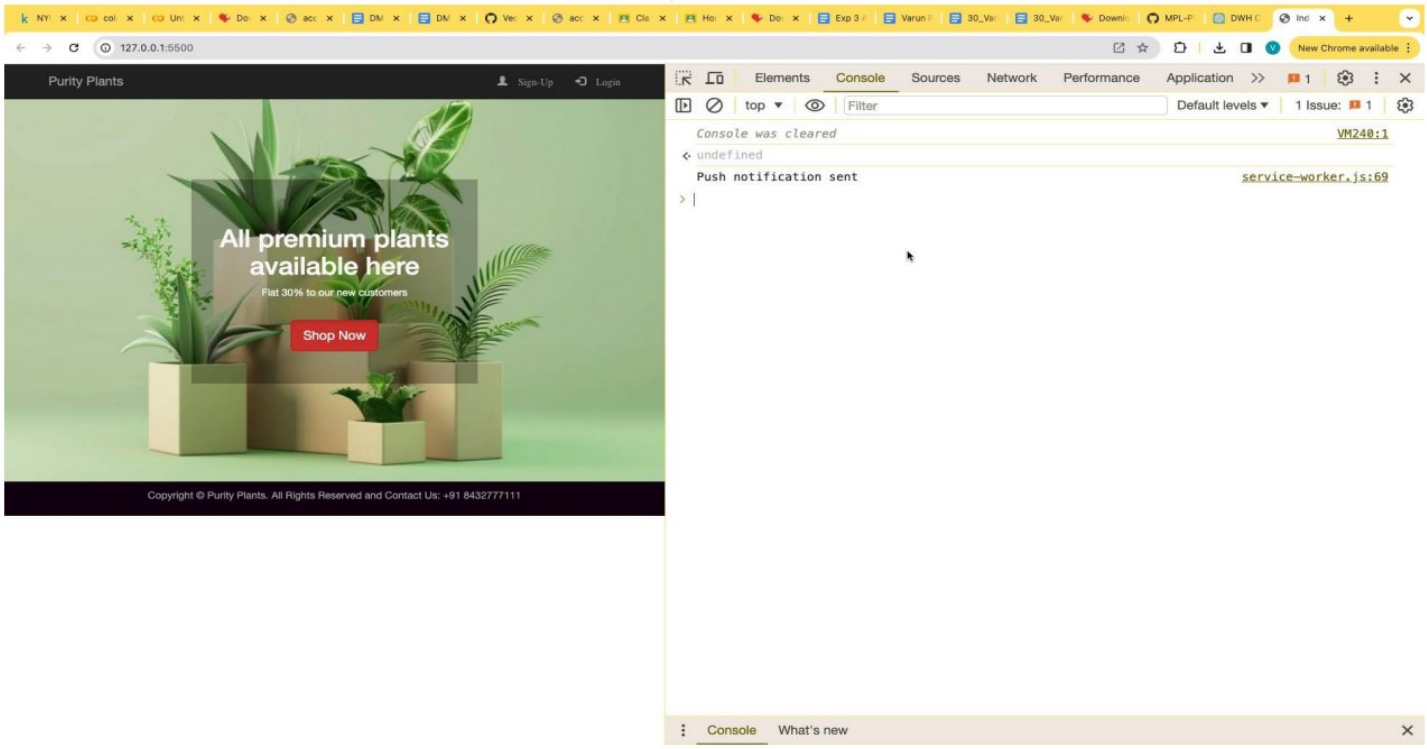
```
      });
  });
};
```

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


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

Output:







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

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Name: Sonali Makhijani

Div: D15A

Roll No: 34

Batch B

Experiment No 10

Aim:

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

Theory:

GitHub Pages

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

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

Pros

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

Cons

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

Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

Pros

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

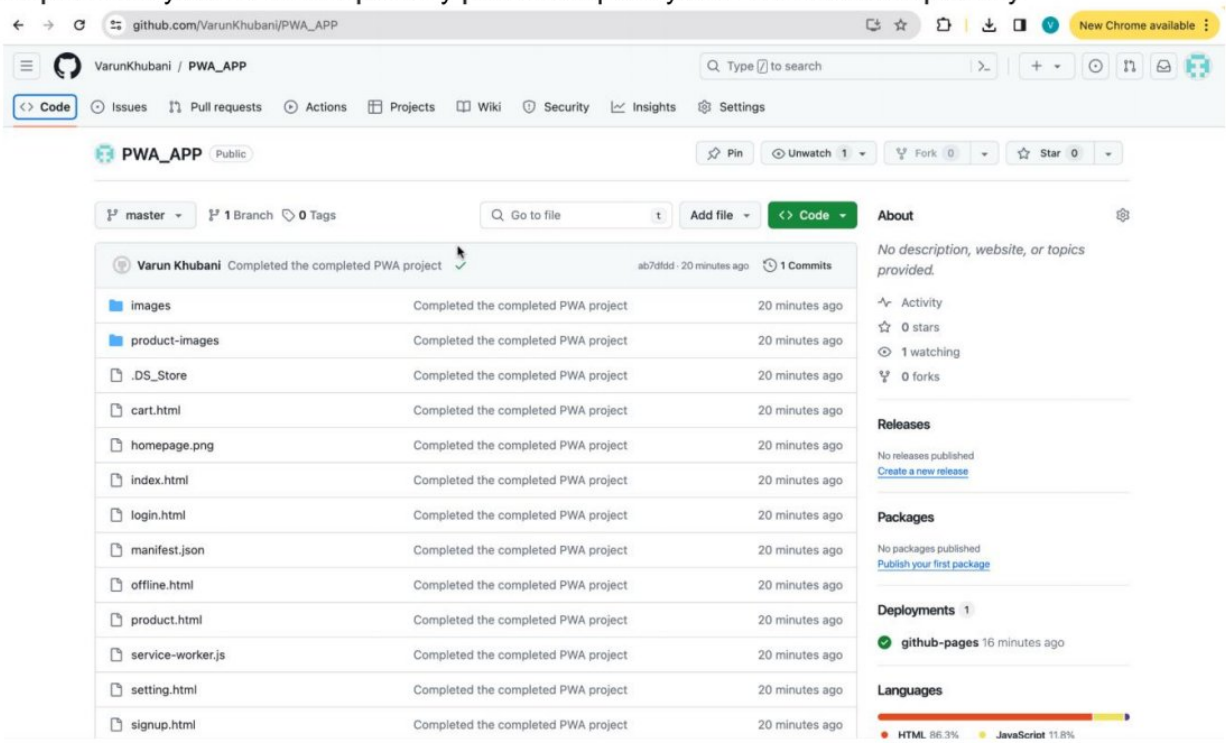
5. Support for HTTPS. A free certificate will be provisioned for your custom domain within 24 hours.

Cons

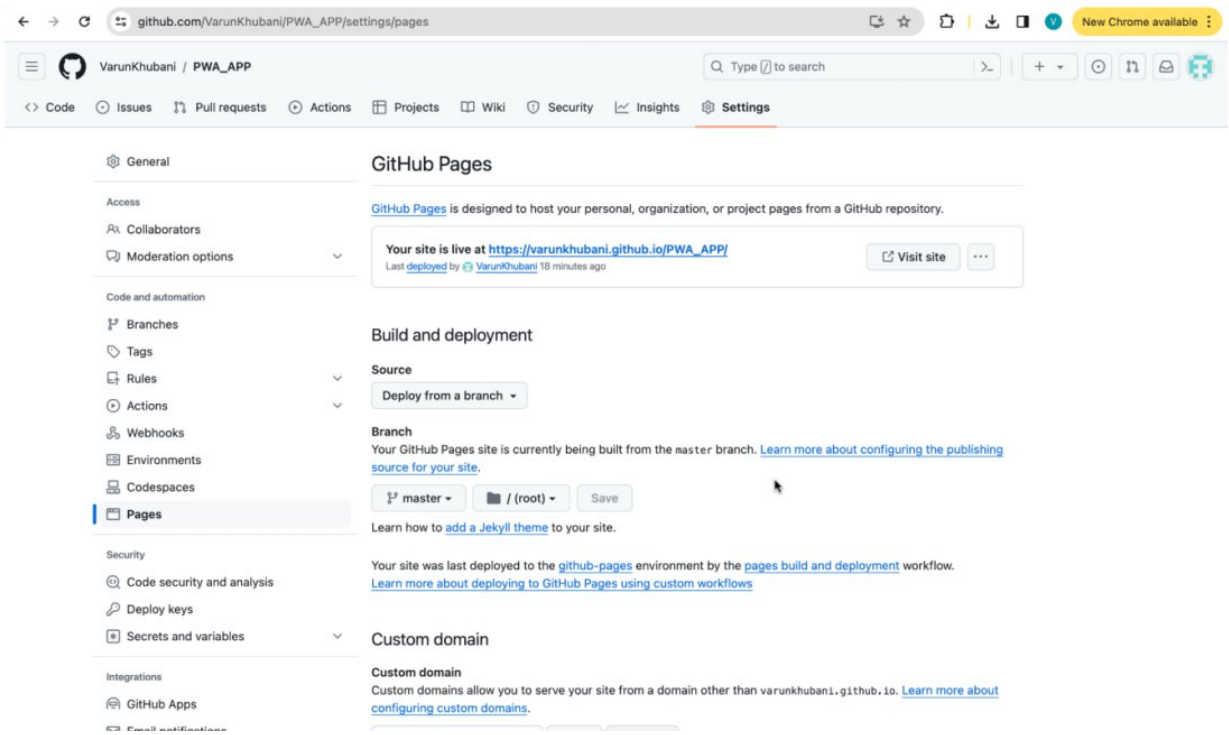
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

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

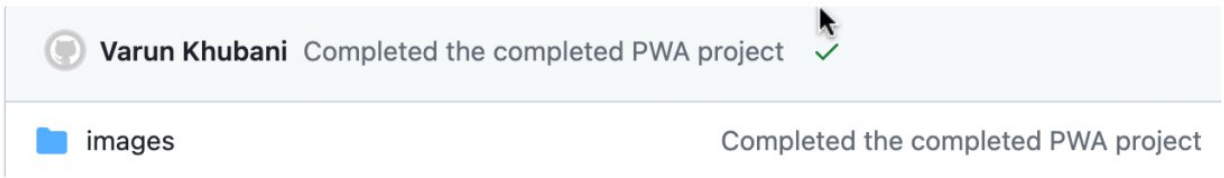
Step1: Make your GitHub Repository public and push your PWA into the repository



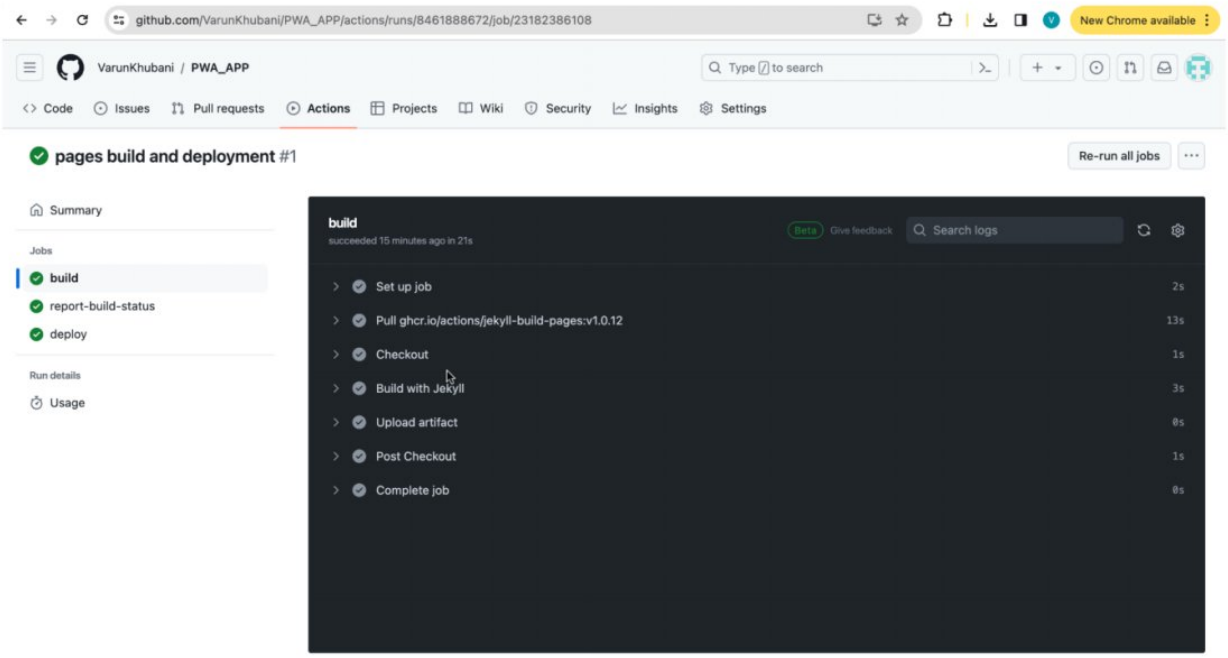
Step 2: Go to settings -> pages and choose your root directory and save it

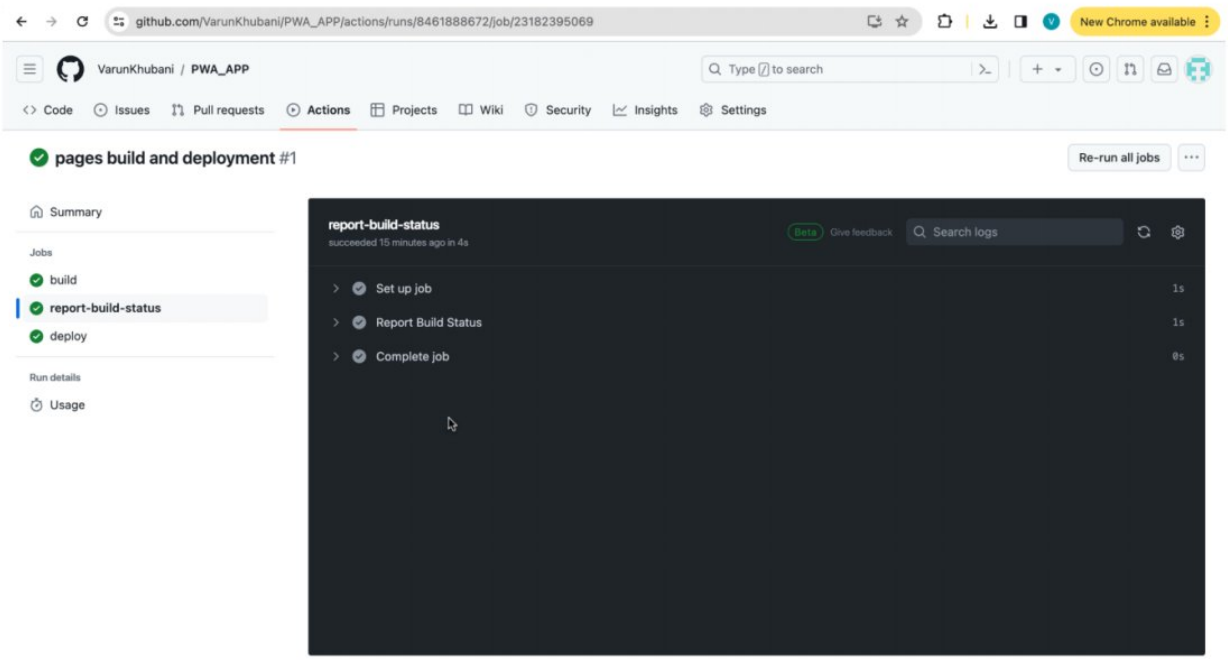


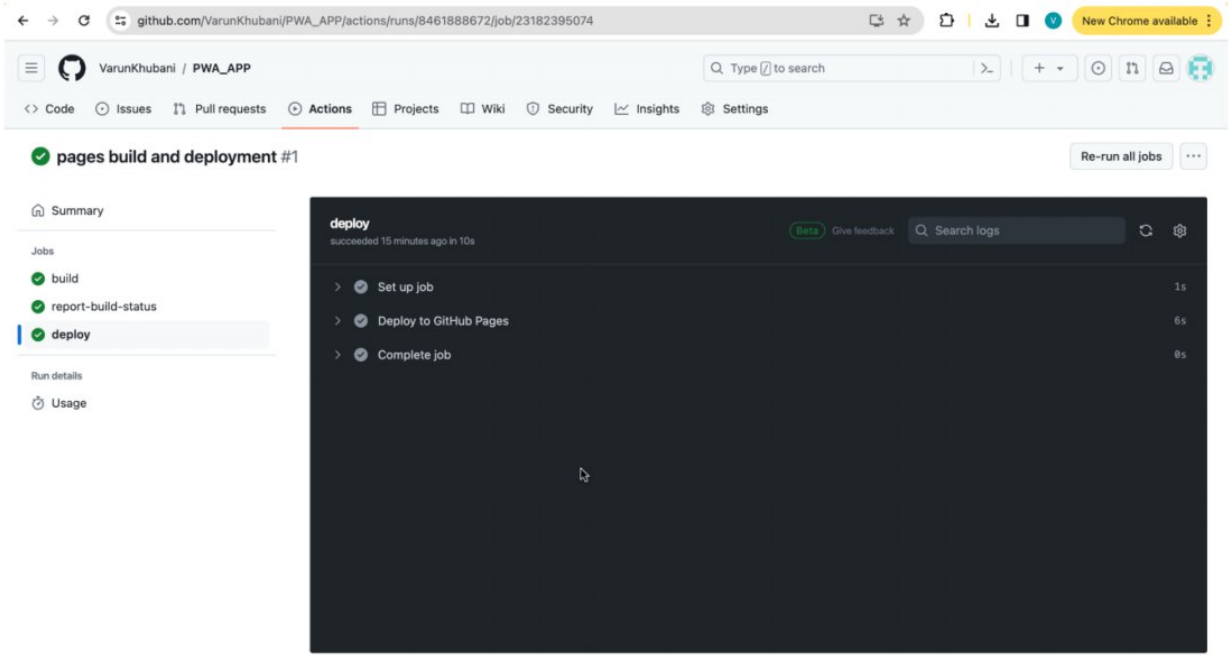
Step 3: Now go to your Code and you will see a small circle near your recent commit(Mine is finished deploying so i am getting a tick-mark sign)



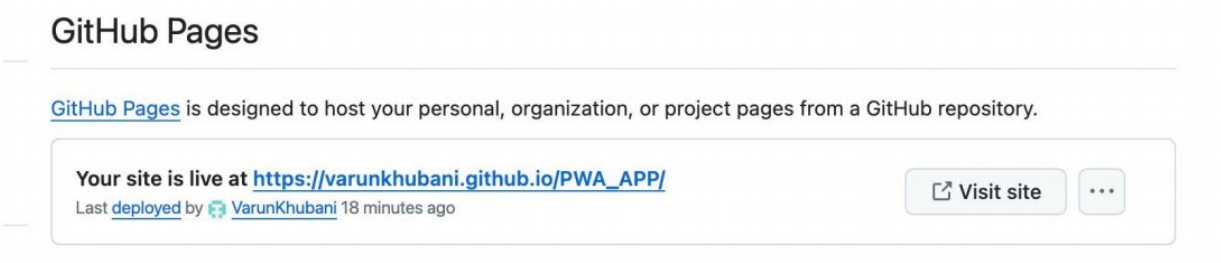
On clicking Logs of all the deployment is shown for convenience

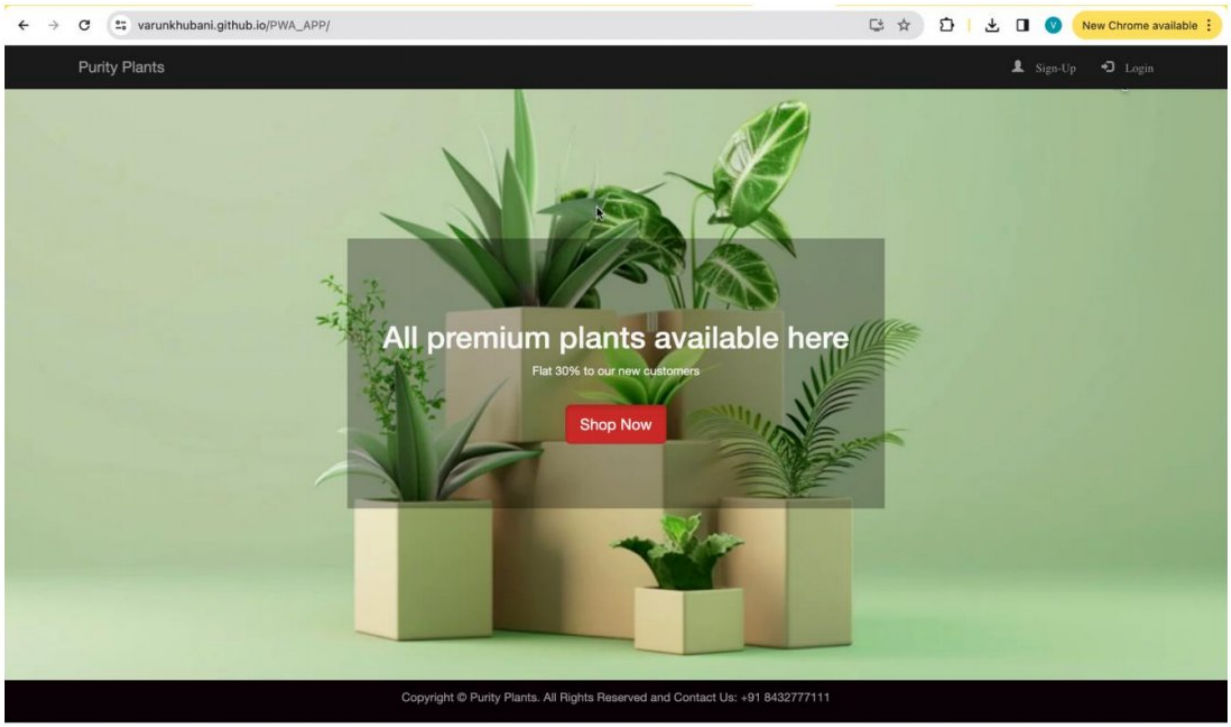






Step4: Go to Settings -> Pages again , and you will see the pages has been deployed and a link is given





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

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

Name: Sonali Makhijani
Division: D15A
Roll No:34
Batch: B

Experiment No 11

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

Theory :

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

Google Lighthouse :

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

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

Key Features and Audit Metrics

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

1. **Performance:** This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the

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

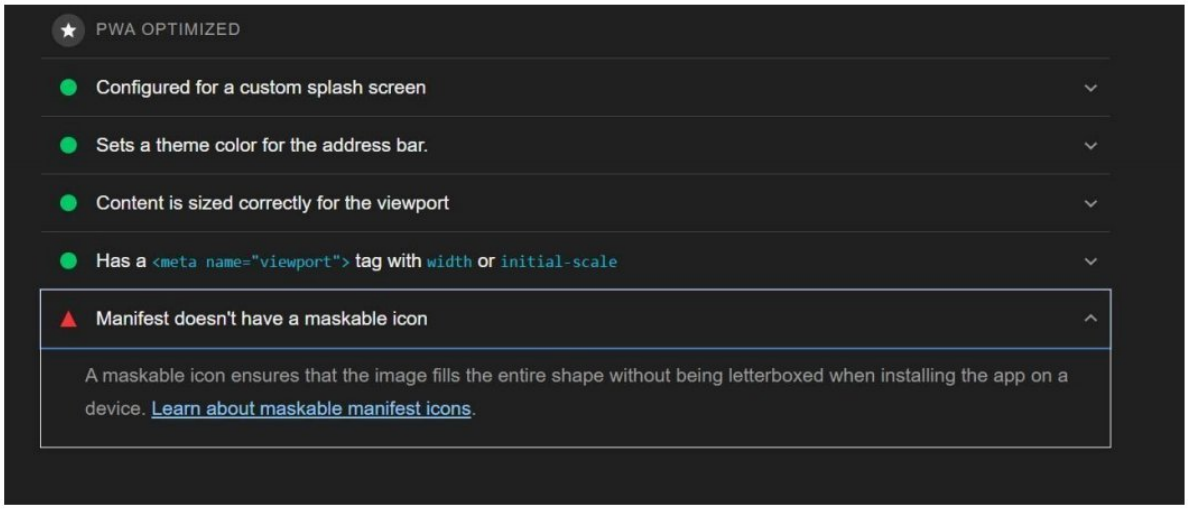
2. **PWA Score (Mobile):** Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards the PWA paradigm, where the objective is to make the application behave as close as possible to native mobile applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled environments, etc.
3. **Accessibility:** As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.
4. **Best Practices:** As any developer would know, there are a number of practices that have been deemed 'best' based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS

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

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

Output:

Before



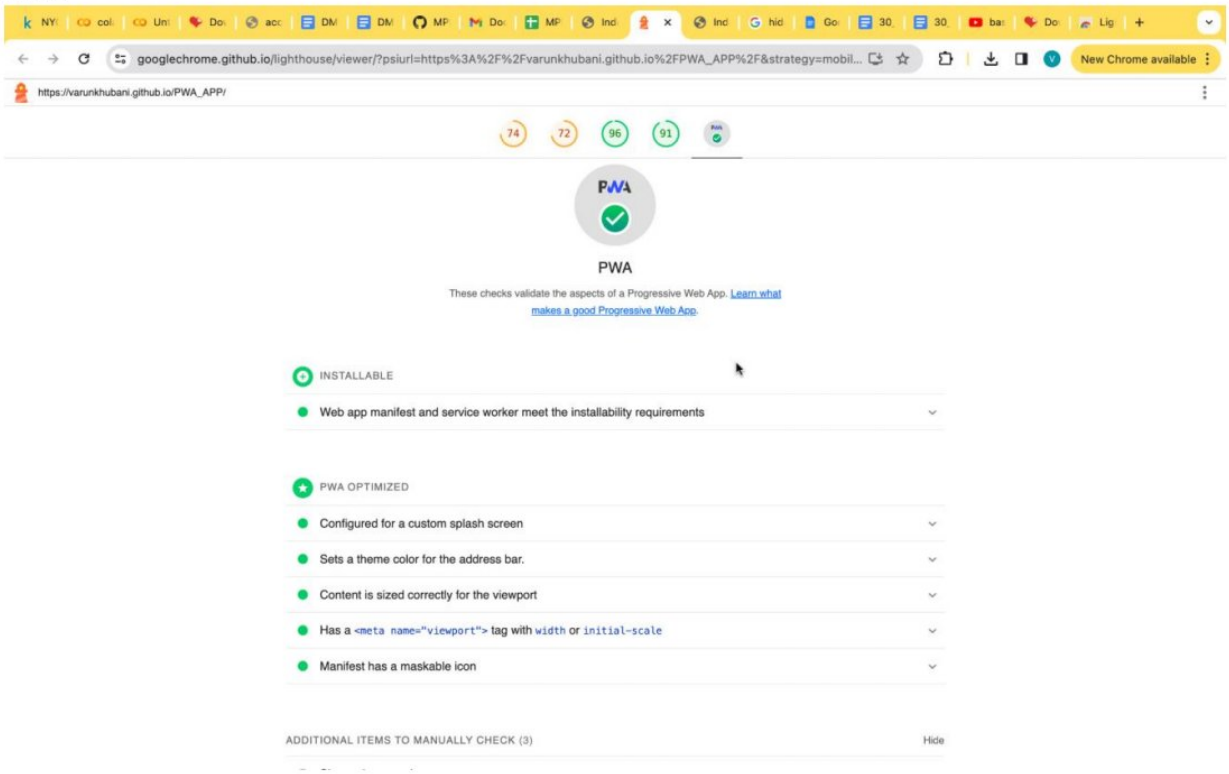
We encountered an issue here , it says “Manifest does not have a maskable icon”

Changes made to the code:

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

```
]
}
```

After:



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

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	5

Sonali Makhijani
34 D15-A

MAD Assignment-1

Q.1 - Flutter Overview:

Flutter is an open source UI software development ToolKit created by Google for building natively compiled application for mobile, web, and develop from single codebase

- Key features of Flutter :-

(a) Single Codebase - Flutter offers a single codebase that can be used to create applications for multiple platforms reducing development time & effort

(b) Hot Reload - Developers can instantly see the effect of code changes with hot reload, enhancing productivity and speeding up the debugging process.

(c) Rich Widgets - Flutter provides a wide range of customizable widgets, allowing developers to create expressive and visually appealing user interfaces.

(d) Performance - Flutter compiles to native ARM code, providing high-performance apps with smooth animations and quick load times.

(e) Access to native features - Flutter provides seamless access to native features and APIs following developers to integrate device specific functionalities without compromising on performance.

- How it is different from Traditional Approach:-

(1) development efficiency - Traditional app development often involves writing and maintaining multiple codebases, which is more time consuming & costly

(2) <u>productivity tools</u> - features like hot reload are not typically available in traditional development environments which can slow down development process.

Q.2 - widget tree and composition :-
widget Tree is a hierarchial structure of UI element represented by widgets ; widgets are the building blocks of Flutter applications and the widget tree organizes them in a parent child relationship widget column involves combining simple widgets to create more complex UIs

- Commonly used widgets :-

1) Scaffold : provides the basic material design layout structure.
2) AppBar : A tool bar typically used at the top of the top screen.
3) Text : displays text with various styling options.
4) Row/column : organizes children widgets in horizontal or vertical manner.
5) container : A versatile widget for styling, positioning or sizing its child
6) Listview : Displays a scrollbar list of widgets
7) Stack : layers widgets on top of each other.

Q.3 - State Management in Flutter :

State Management in flutter is a critical aspect of web development as it deals with managing the data that changes overtime in your app and how those changes affect the UI.
It provides advantages like UI consistency, code Maintainability, performance optimization, scalability and many more.

- Compare and contrast different state Management approaches in Flutter :

Features	SetState	Provider	Riverpod
Description	Built-in feature flutter for managing state	State management library that allows data sharing accross widgets	A more flexible and powerful evolution of provider
complexity	Simple and straight forward for beginners	Moderate complexity requires understanding of provider concepts	highest complexity due to its flexibility and advanced features
Scalability	Not suitable for large apps or extensive state management	good scalability and maintainability for larger applications	Excellent scalability suitable for complex and large applications

performance	can lead to inefficient in larger apps due to frequent widget rebuilds	optimizes performance by prioritizing unnecessary widgets	High optimized state management minimizing unnecessary widgets.

- Scenarios for each approach:-

- Set State :- Best for simple or demo apps where you're experimenting with Flutter or when the state is confined to a simple widget.

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

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

Q.4 - Firebase Integration in Flutter :-

Steps in process of Integrating firebase with flutter :-

1) Setup firebase project : create a firebase project and register your flutter app in the firebase console.
2) Configure flutter app : Add firebase dependencies in 'pubsec.yaml' and initialize firebase in the main.dart using 'firebase.initializeApp()'.
3) Using firebase services : Include firebase services in your flutter app like Authentication ('firebase_auth') ; firebase ('cloud_firestore') and realtime database ('firebase_database').
4) Authentication Example : Sign in with Email and password using 'FirebaseAuth.Instance.SignInWith EmailAndPassword'.
5) Firebase Example : Read data from firestore using 'FirebaseFirestore.instance.collection('user').doc('user.get()'.

- Benefits of using Firebase as Backend Solution :

1) Real-time data type :
Firebase provides create-time data synchronisation allowing updates to be immediately reflected in all connected clients.

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

- Real-time data Synchronisation in Firebase

Firebase achieves real-time data Synchronisation through web sockets. when data changes on the server, firbase sends updates to the connected clients over web sockets ensuring that the clients wear data sync with the server.

MAD & PWA Lab
Journal

Experiment No.	Assignment-2
Assignment 2 Questions	1. Define Progressive Web App (PWA) and explain its significance in modern web development.Discuss the key characteristics that differentiate PWAs from traditional mobile apps 2. Define responsive web design and explain its importance in the context of Progressive Web Apps.Compare and contrast responsive, fluid, and adaptive web design approaches. 3. Describe the lifecycle of Service Workers, including registration, installation, and activation phases. 4. Explain the use of IndexedDB in the Service Worker for data storage.
Roll No.	34
Name	Sonali Makhijani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	4

Sonali Makhijani
D15-A 34

MAD - PWA Assignment - 2

Q 1) Define Progressive Web App (PWA) and explain its significance that differentiate PWAs from traditional mobile app.

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

Key characteristics of Progressive web App are :-

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

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

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

④ Fast performance

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

⑥ Security :- PWAs are served over HTTPs to ensure security

⑦ cross - platform compatibility

The significance of PWAs is modern web development lies in their ability to bridge the gap between web and native app native app experiences By combining the reach and accessibility of the web with functionality and engagement of native apps. They allow developers to create fast, reliable and engaging experience that work seamlessly across a wide range of devices and platforms, helping to improve user engagement, retention and conversion rates. PWAs are easier and more cost effective to develop and maintain compared to native apps making them an attractive option for business and developers looking to reach a broad audience with their applications.

(Q2) Define responsive web design and explain it's Importance in the context of progressive web apps. Compare and contrast responsive, fluid and adaptive web design approaches.

→ Responsive web design is a web page design approach that ensure a website adapts the layout and content to seamlessly function and display well across various screens sizes, from desktops to smart phones.

Importance for PWAs :-

- foundation for App - like Experience : RWD ensures proper layout and navigation across devices, mimicking the adaptability of native apps.
- Accessibility :- A responsive design make PWA accessible to wider audience using diverse devices
- SEO benefits - Responsive design is a positive ranking factor in search engine algo, boosting the discoverability of PWAs.

① Responsive web design v/s fluid v/s Adaptive.

feature	Responsive	fluid	Adaptive
layout	flexible, adapts to any screen size	continously adjusts based on relative units	uses multiple fixed width layouts.
Development efforts	Moderate	less effort for basic layout	More effects to create Multiple layout
control	Good control over overall layout	less control over element appearance on extreme size.	High control over layout for each device category
Suitability for PWA	Ideal foundation due to it's versatility	can be used but may require adjustment for optimal experience	less common for PWA due to development overhead

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

→ Service workers are crucial components of progressive web apps (PWAs) that enable features such as offline support, push notifications and background synchronizations. The lifecycle of service worker involves several key phases:

1) Registration :- 1st phase of lifecycle of service worker involves several steps in the main JS file of app
- Registration is done using the `navigator.service.worker.register()` method, which takes the path to service worker file as a parameter.

2) Installation :- Once service worker registered. browser downloads and starts installations process of service worker script.
- During installation, the service worker script is executed, and the `install` event is fired. Inside `install` event handler. developers can define what resources should be cached for offline access using techniques like caching strategies or pre caching
- Installations process is critical for setting up the service worker's initial cache and preparing it to take control of web application.

3) Activation : - After installation phase, Activation phase it occurs when service worker is ready to take control of the web application
- During activation, 'activate' event is fired, giving service worker an opportunity to clean up any old caches or perform other tasks neccessary for applications' functionality
- Developers can use the 'activate' event handler to manage cache cleanup versioning and other activation related tasks.
- Service remains active until it is explicitly unregistered or updated.

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

-> IndexedDB lets you store and retrieve objects that are indexed with a key, any objects supported by the structured clone algorithm can be stored.
IndexedDB is a low-level API for client-side storage of significant amounts of structured data, including files & blobs.