Vivekanand Education Society's Institute of Technology

An Autonomous Institute Affiliated to University of Mumbai
Hashu Advani Memorial Complex, Collector Colony, Chembur East, Mumbai - 400074.

Department of Information Technology

CERTIFICATE

This is to certify that **Varun Khubani** of **D15A** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Name of the Course : MAD & PWA Lab

Course Code : ITL604

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

Faculty Incharge : Mrs. Kajal Joseph.

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

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

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: Nike Shoes Clone and Plants Ecommerce PWA App **Roll No. 30**

Lab Objectives:

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

Lab Outcomes:

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

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1	17/1/24	24/1/24	10
2.	To design Flutter UI by including common widgets.	LO2	24/1/24	31/1/24	10
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, LO2, LO3	4/2/24	5/2/24	5
13.	Assignment-2	LO4, LO5, 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.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

Name: Varun Khubani
Division: D15A
Roll
No:30
Batch: B

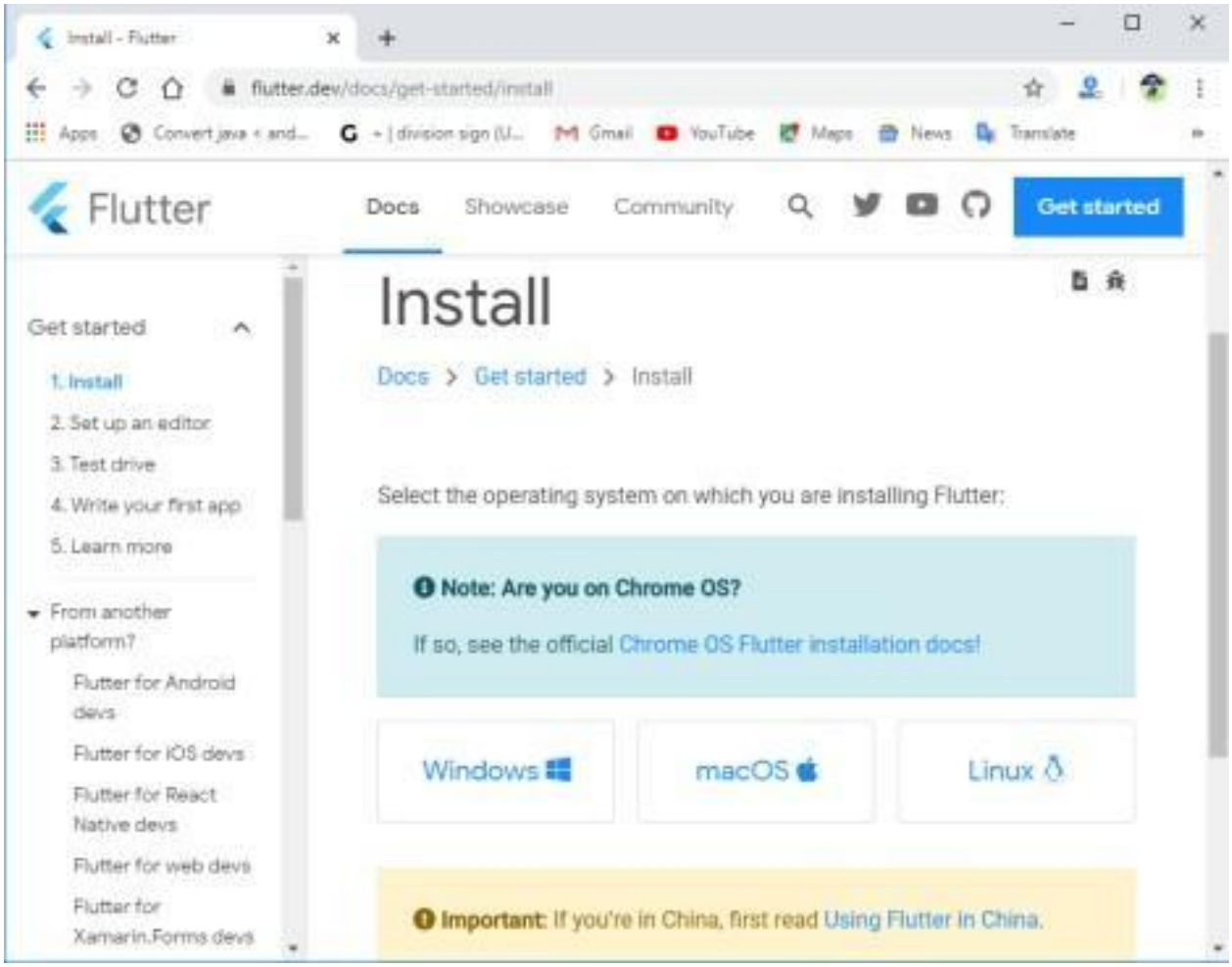
Experiment No. 1

Aim: To Install and Configure Flutter Environment

Pre Requisites:

Install the Flutter SDK

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



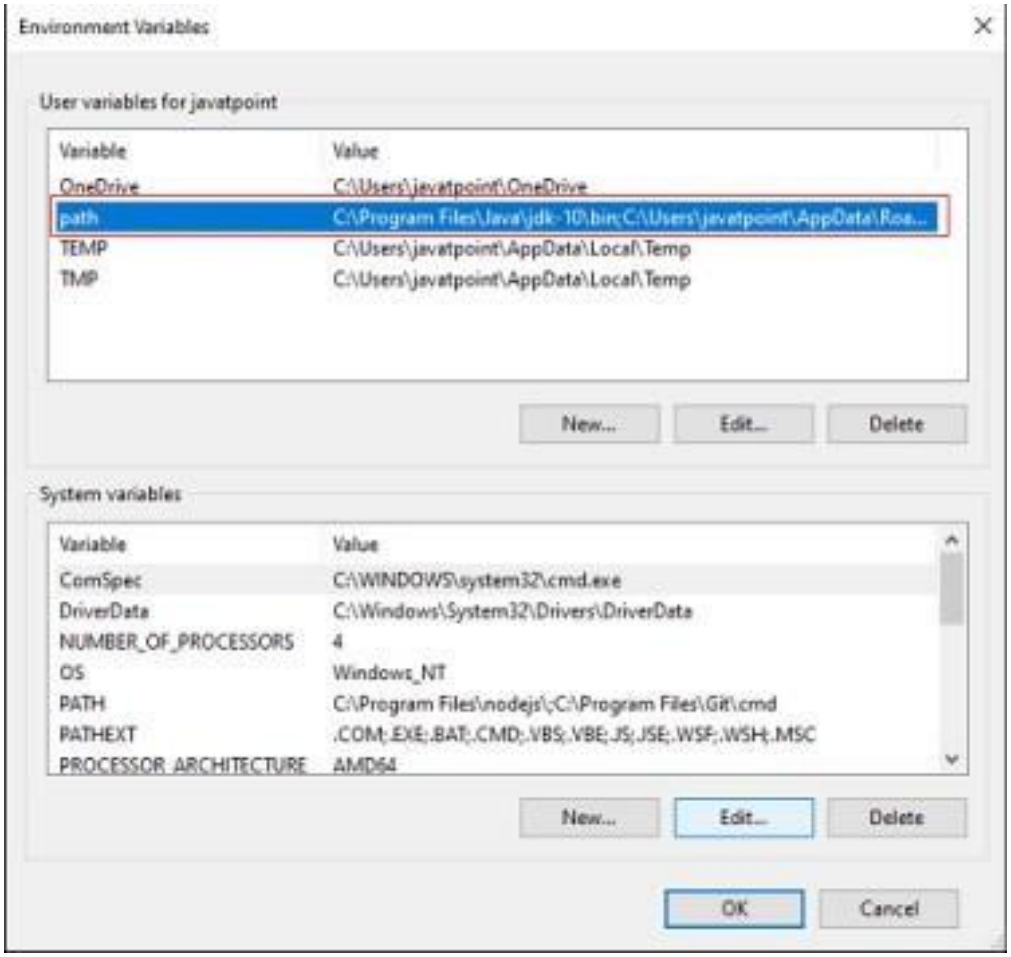
Step 2: Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will

find the download link for SDK.

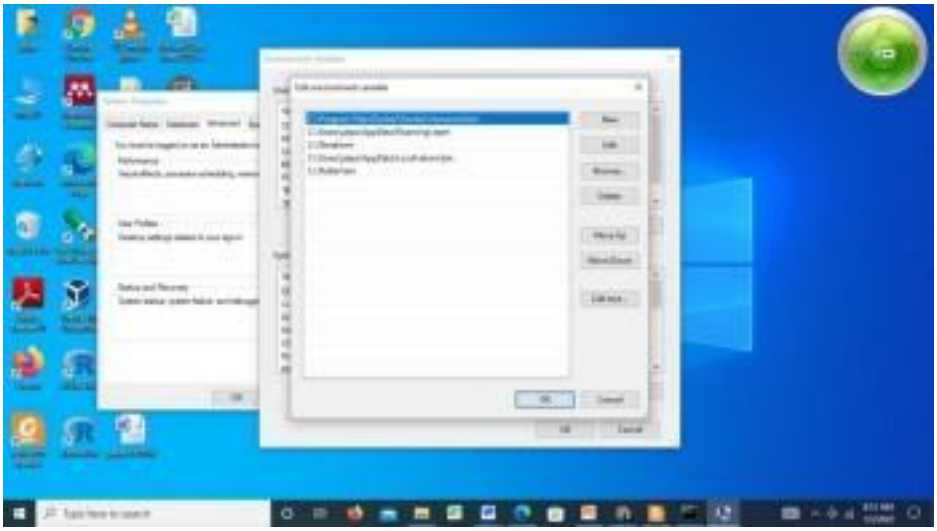
Step 3: When your download is complete, extract the **zip** file and place it in the desired installation folder or location, for example, C: /Flutter.

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

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

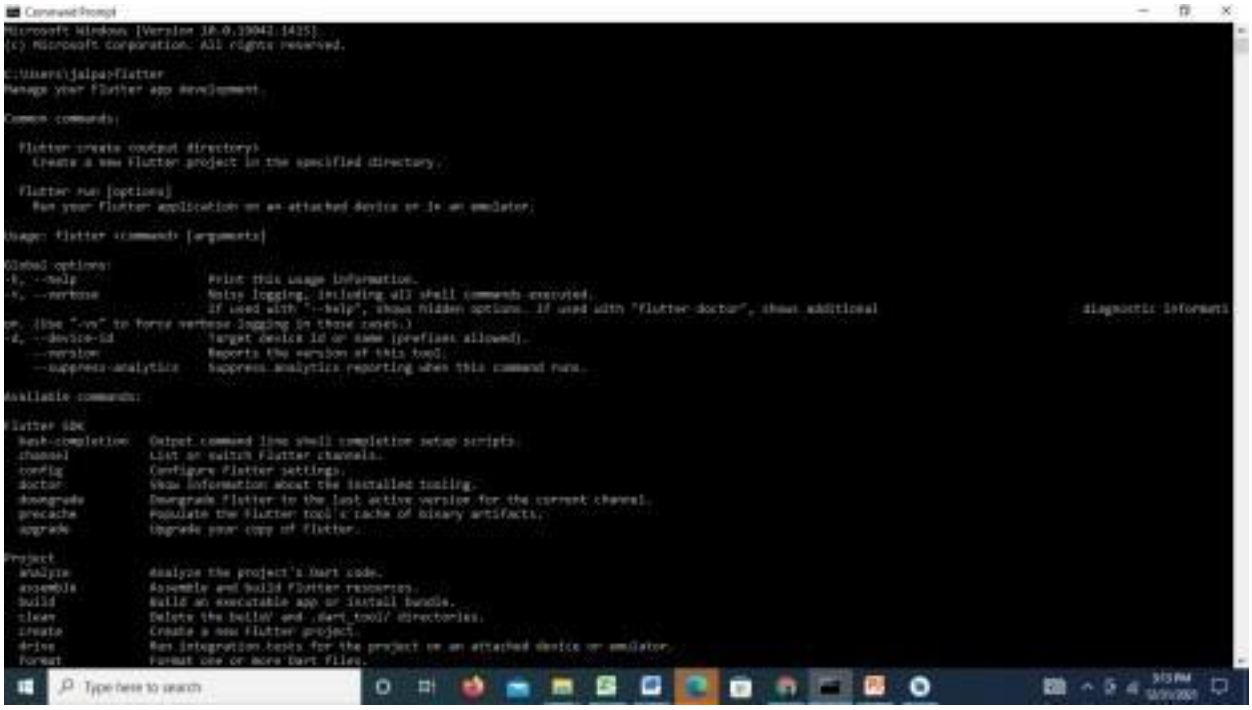


Step 4.2: Now, select path -> click on edit. The following screen appears

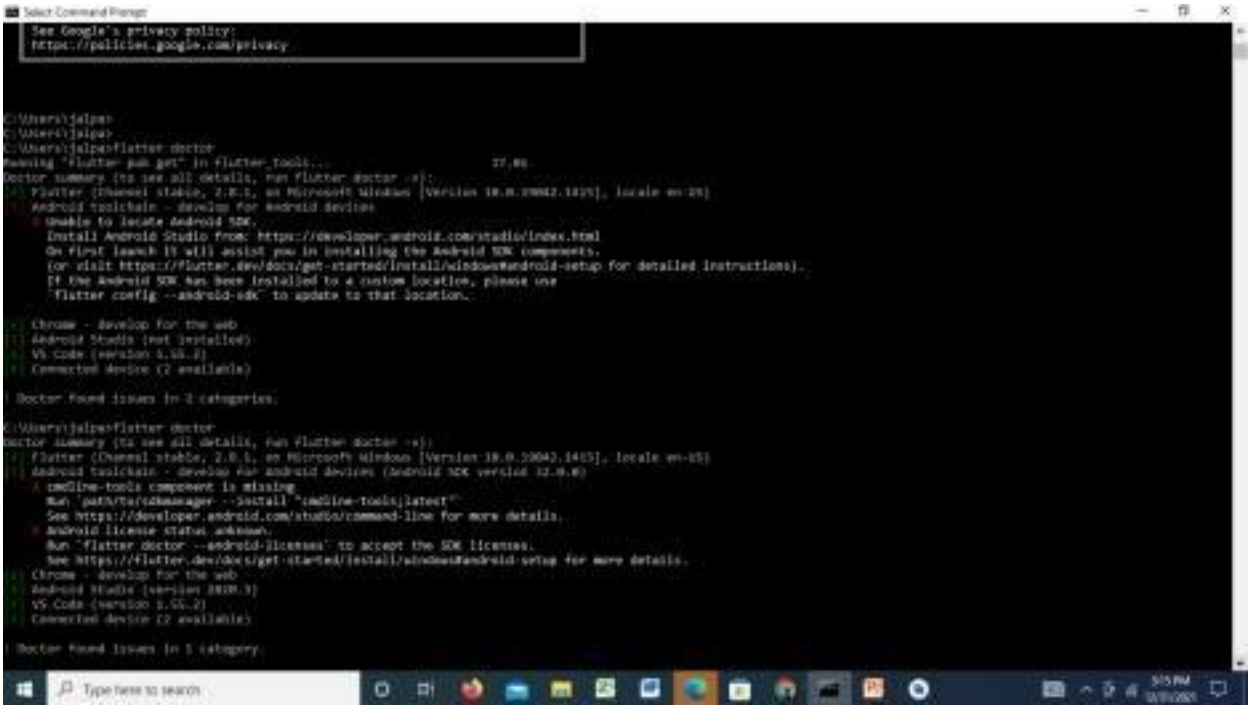


Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value - > ok -> ok -> ok.

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



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



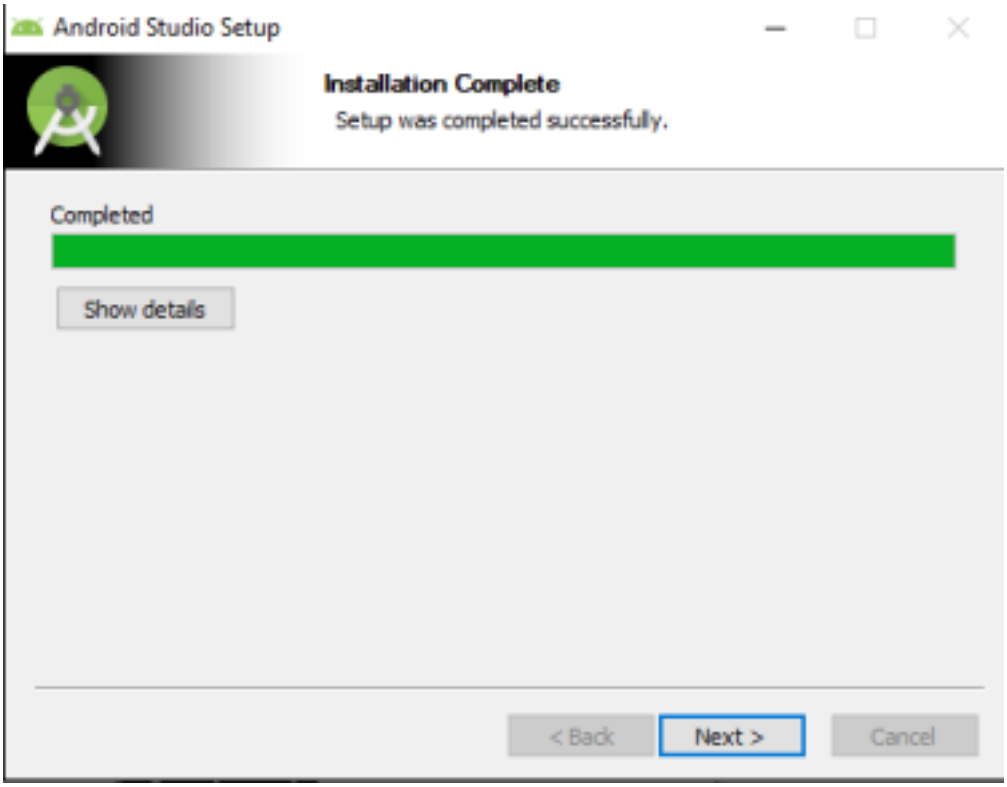
Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

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

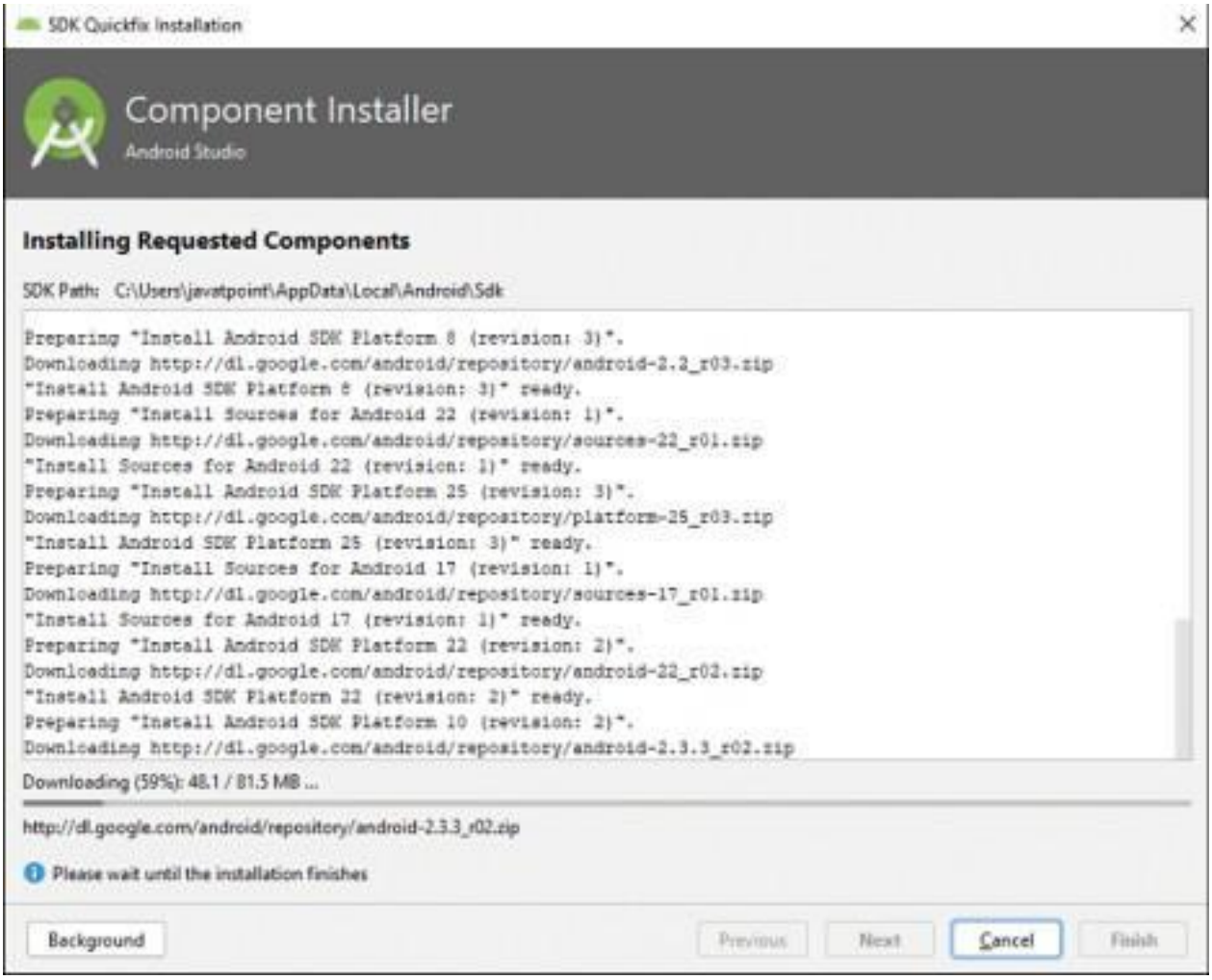
Step 7.1: Download the latest Android Studio executable or zip file from the official site.

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

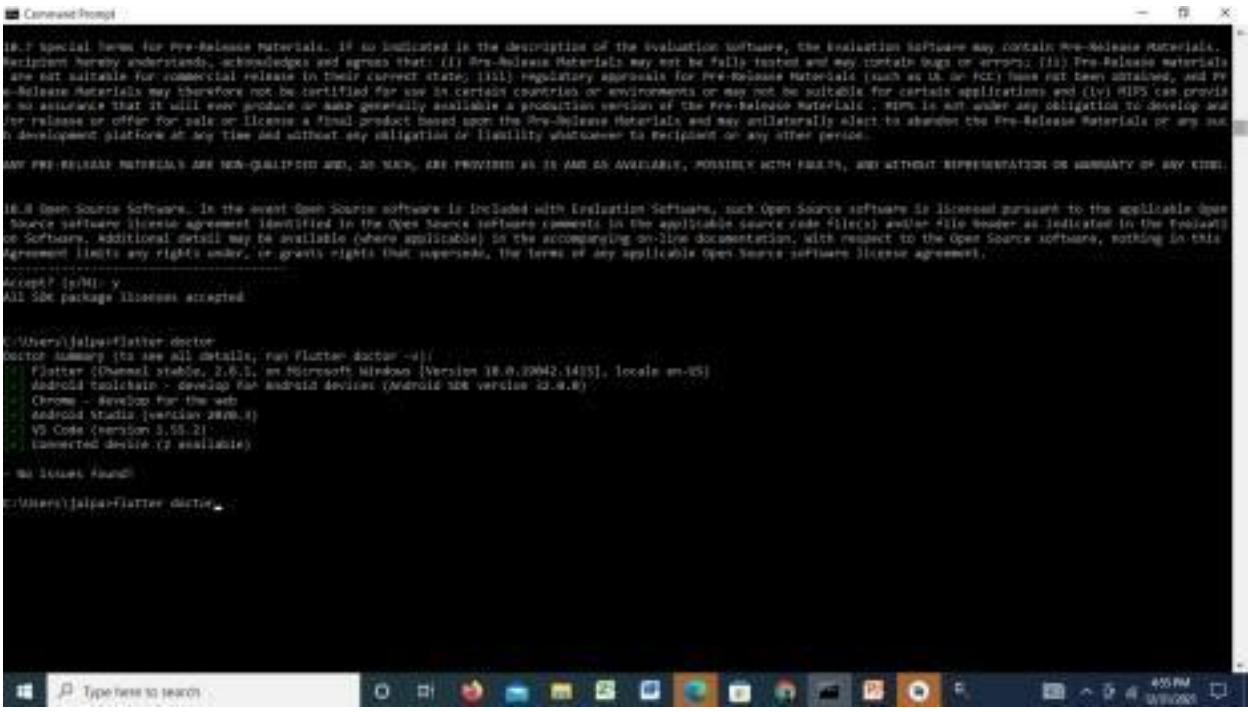
Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



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



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



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

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

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

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

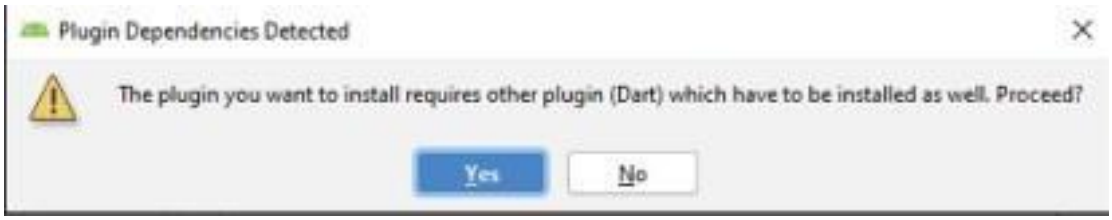
Step 8.4: Now, verify the all AVD configuration. If it is correct, click on Finish.
The following screen appears.

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

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

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

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

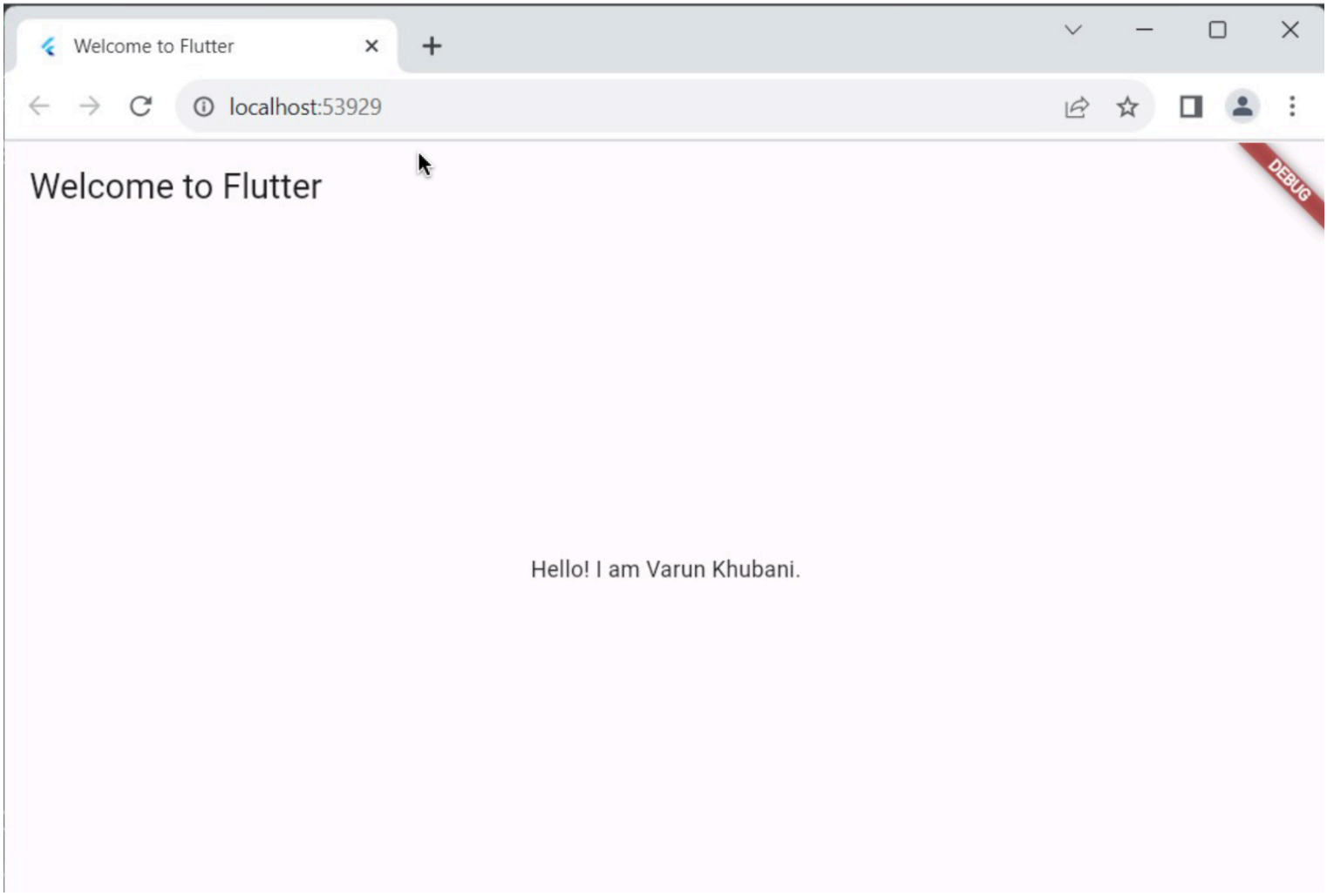


Step 9.3: Restart the Android Studio.

Code:

```
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('Varun Khubani'),
),
),
);
}
}
```

Output:



Conclusion: Hence We ran a simple program on running a simple text,on flutter, running on a virtual device

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Name: Varun Khubani
Division: D15A
Roll no: 30
Batch: B

Experiment No 2

Aim: To design flutter UI by including common widgets.

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

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

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

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

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

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

6. RaisedButton and FlatButton: These button widgets create interactive elements for users to trigger actions, with RaisedButton offering a raised appearance and FlatButton a flat design.

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

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

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

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

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

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

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

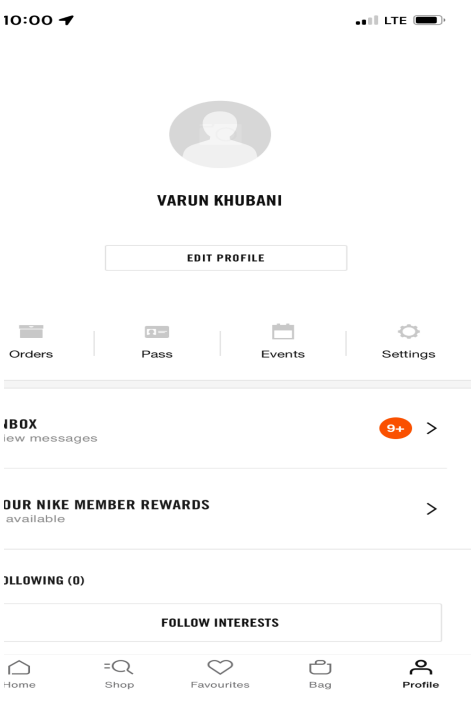
Code:

```
import 'package:Nike_Clone/core/constants/app_colors.dart'; import
'package:Nike_Clone/core/constants/constants.dart'; import
'package:Nike_Clone/core/widgets/round_icon_button.dart'; import
'package:Nike_Clone/features/chat/presentation/screens';
import 'package:flutter/material.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';

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

static const routeName = '/home';

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

class _HomeScreenState extends State<HomeScreen> with
TickerProviderStateMixin {
late final TabController _tabController;

@override
void initState() {
_tabController = TabController(length: 5, vsync: this);
    super.initState();
}
```

```
@override
void dispose() {
  _tabController.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return DefaultTabController(
    length: 5,
    child:
    Scaffold(
      backgroundColor: AppColors.greyColor,
      appBar: AppBar(
        backgroundColor: AppColors.whiteColor,
        elevation: 0,
        title: NikeShoes(), actions:
        [
          _buildSearchWidget(),
          _buildMessengerWidget(),
        ],
        bottom: TabBar(
          tabs: Constants.getHomeScreenTabs(_tabController.index),
          controller: _tabController,
          onTap: (index) {
            setState(() {});
          },
        ),
      ),
      body: TabBarView(
        controller: _tabController,
        children: Constants.screens,
      ),
    ),
  );
}

Widget _NikeShoes() => const Text( 'Nike
      Shoes',
      style: TextStyle(
        color: AppColors.blueColor,
```

```
fontSize: 30,
fontWeight: FontWeight.bold,
),
);

  Widget _buildMainPaget() => const RoundIconButton(
        icon: FontAwesomeIcons.magnifyingGlass,
);

  Widget _buildHomePaget() => InkWell(
        onTap: () {
Navigator.of(context).pushNamed(ChatsScreen.routeName);
},
child: const RoundIconButton(
icon: FontAwesomeIcons.facebookMessenger,
),
);
}
```

Output:



MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Name: Varun Khubani
Division: D15A
Roll No:
30 Batch:
B

Experiment No 3

Aim: To include images and fonts in flutter app

Theory: Certainly! Here's a simplified process for adding images in Flutter:

1. Import Libraries:
   Ensure that you have the necessary libraries imported in your Dart file. For images, you'll typically use `dart:ui` and other relevant Flutter packages.

2. Adding Local Images:
   - Place your local images in the `assets` folder.
   - Declare the images in the `pubspec.yaml` file.

3. Adding Network Images:
   - Use the `Image.network` widget for displaying images from the internet.

4. Image Widget:
   - Create an `Image` widget and provide it with an `ImageProvider`.
   - Use `AssetImage` for local images and `NetworkImage` for network images.

5. ImageProvider:
   - Understand that `AssetImage` and `NetworkImage` are subclasses of the `ImageProvider` class.
   - You can create custom `ImageProvider` if needed.

6. CachedNetworkImage (Optional):
   - If you want to cache network images, consider using the `cached_network_image` package.

7. Image Loading and Error Handling:
   - Customize the loading and error behavior using `loadingBuilder` and `errorBuilder` properties of the `Image` widget or other relevant widgets.

Remember, the actual implementation details might vary based on your specific use case and the packages you choose to use. The key is to understand the concepts of working with local and network images and the various widgets and packages available in Flutter for handling images.

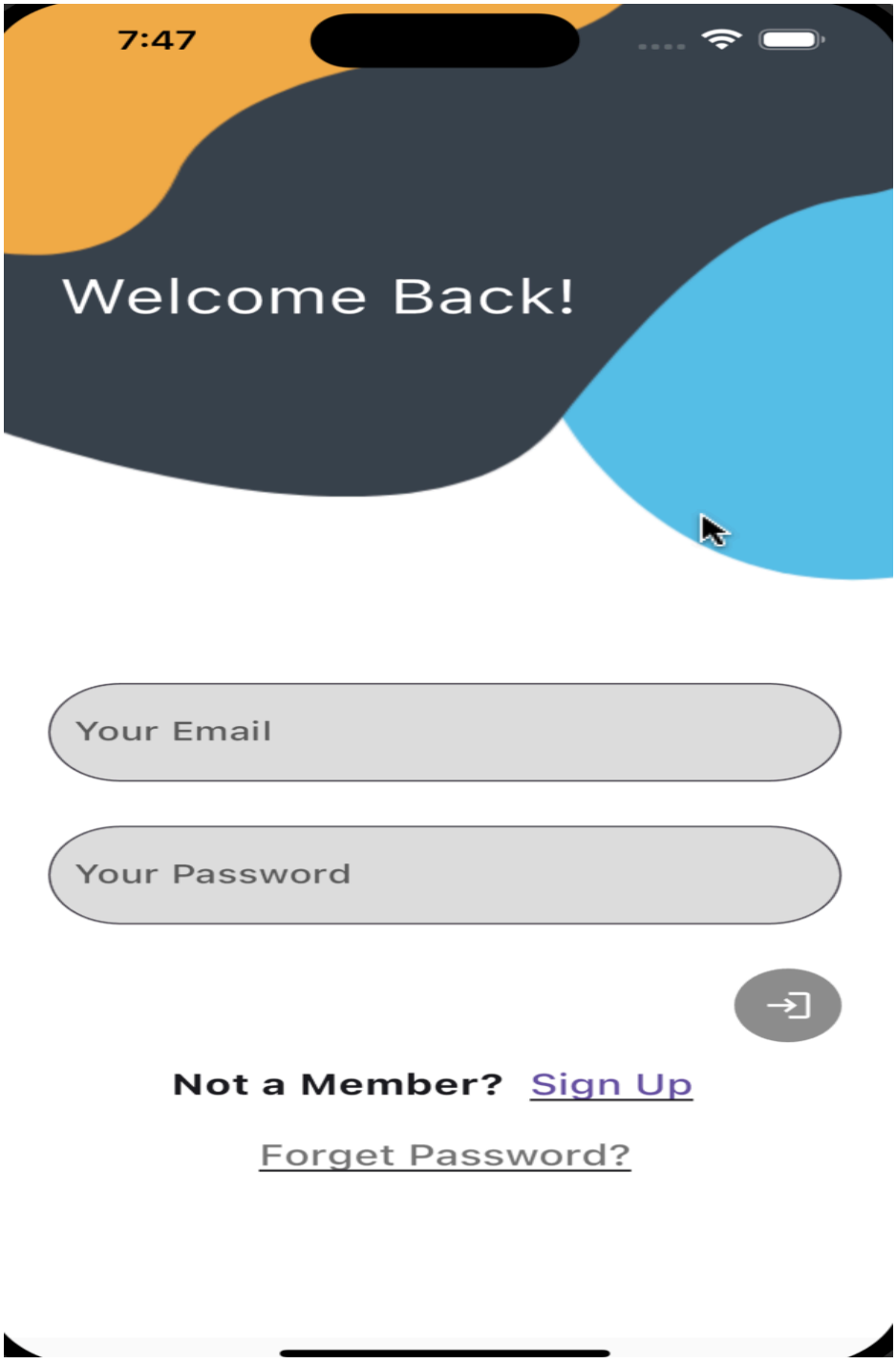
Code:

```dart
import
'package:Nike_Clone/features/auth/presentation/screens/create_account_
screee.dart';
import
'package:Nike_Clone/features/auth/providers/auth_provider.dart';
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

import '/core/constants/constants.dart';
import '/core/widgets/round_button.dart';
import '/core/widgets/round_text_field.dart';
import '/features/auth/utils/utils.dart';

final _formKey = GlobalKey<FormState>();

class LoginScreen extends ConsumerStatefulWidget {
  const LoginScreen({super.key});

  @override
  ConsumerState<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends ConsumerState<LoginScreen> {
  late final TextEditingController _emailController;
  late final TextEditingController _passwordController;

  bool isLoading = false;

  @override
  void initState() {
    _emailController = TextEditingController();
    _passwordController = TextEditingController();
    super.initState();
  }
  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
```

```
}
Future<void> login() async {
  if (_formKey.currentState!.validate()) {
    _formKey.currentState!.save();
    setState(() => isLoading = true);
    await
    ref.read(authProvider).signIn(
          email: _emailController.text,
          password: _passwordController.text,
        );
    setState(() => isLoading = false);
  }
}
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(),
    body: Padding(
      padding: Constants.defaultPadding,
      child: Column(
        mainAxisSize: MainAxisSize.max,
        mainAxisAlignment:
        MainAxisAlignment.spaceAround, children: [
          Image.asset(
            'assets/icons/main_logo.png
            ', width: 60,
          ),
          Form(
            key: _formKey,
            child: Column(
              children: [
                RoundTextField(
                  controller:
                  _emailController, hintText:
                  'Email',
                  keyboardType: TextInputType.emailAddress,
                  textInputAction: TextInputAction.next,
                  validator: validateEmail,
                ),
                const SizedBox(height: 15),
                RoundTextField(
                  controller: _passwordController,
```

```
                  hintText: 'Password',
                  keyboardType: TextInputType.visiblePassword,
                  textInputAction: TextInputAction.done,
                  isPassword: true,
                  validator: validatePassword,
                ),
                const SizedBox(height: 15),
                RoundButton(onPressed: login, label: 'Login'),
                const SizedBox(height: 15),
                const Text(
                  'Forget Password',
                  style: TextStyle(fontSize: 18),
                ),
              ],
            ),
          ),
          Column(
            children: [
              RoundButton(
                onPressed: () {
                  Navigator.of(context).pushNamed(
                    CreateAccountScreen.routeName,
                  );
                },
                label: 'Create new account',
                color: Colors.transparent,
              ),
              Image.asset(
                'assets/icons/main.png',
                height: 50,
              ),
            ],
          ),
        ],
      ),
    ),
  );
}
}
```

Output:


7:47
Welcome Back!
Your Email
Your Password
Not a Member? Sign Up
Forget Password?

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Name: Varun Khubani
Division: D15A
Roll No:
30 Batch:
B

Experiment No 4

Aim: To create interactive form using form widget

Theory: In Flutter, a "Form" is a widget that represents a container for a collection of form fields. It helps manage the state of the form and facilitates the validation and submission of user input. Here are some key concepts and theories about forms in Flutter:

1. Widget Hierarchy:
   Forms in Flutter are composed of the `Form` widget, which contains a list of `FormField` widgets. Each `FormField` represents an individual input field like text fields, checkboxes, or dropdowns.

2. Form State:
   The `Form` widget maintains the state of the form, including the current values of the form fields and their validation statuses. The form state is automatically managed by Flutter.

3. Validation:
   Forms provide built-in validation through the `validator` property of each `FormField`. Validators are functions that determine whether the input is valid. The form's overall validity is determined by the validity of all its fields.

4. Form Submission:
   Form submission is typically triggered by a button press. The `onPressed` callback of the button can call the `FormState.save()` method, which invokes the `onSaved` callback for each form field and then calls the `onFormSaved` callback.

5. GlobalKey<FormState>:
   To interact with the form state, a `GlobalKey<FormState>` is commonly used. This key allows access to the form state and is used to validate and save the form.

6. Auto-validation:
   Flutter provides automatic validation by calling the `validator` function whenever the user input changes. This allows for real-time feedback to the user about the validity of their input.

7. Form Submission Lifecycle:
   The form submission process involves validation, saving, and then handling the saved data. Developers can customize this process by providing their own logic within the `onSaved` and `onFormSaved` callbacks.

8. Focus Management:
Forms handle the focus of input fields, making it easy to navigate through the form using keyboard input or programmatically setting focus on specific fields.

9. FormKey and GlobalKey:
Using a `GlobalKey<FormState>` allows for more control over the form, such as triggering form validation or resetting the form. It is usually defined as a global key in the widget tree.

10. Form Persistence:
Form data can be persisted across different screens or app sessions by passing the data down the widget tree or using state management solutions like Provider or Riverpod.

Forms play a crucial role in user interaction, data collection, and validation in Flutter applications, providing a structured and efficient way to handle user input.

Code:

```
import 'dart:io';
import 'package:Nike_Clone/core/constants/app_colors.dart';
import 'package:Nike_Clone/core/constants/constants.dart';
import 'package:Nike_Clone/core/utils/utils.dart';
import 'package:Nike_Clone/core/widgets/pick_image_widget.dart';
import 'package:Nike_Clone/core/widgets/round_button.dart'; import
'package:Nike_Clone/core/widgets/round_text_field.dart';
importpackage:Nike_Clone/features/auth/presentation/widgets/birthday_p
icker.dart';
import'package:Nike_Clone/features/auth/presentation/widgets/gender_pi
cker.dart';
import
'package:Nike_Clone/features/auth/providers/auth_provider.dart';
 import 'package:Nike_Clone/features/auth/utils/utils.dart';
 import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
final _formKey = GlobalKey<FormState>();
class CreateAccountScreen extends ConsumerStatefulWidget {
 const CreateAccountScreen({super.key});
 static const routeName = '/create-account';
 @override
 ConsumerState<CreateAccountScreen> createState() =>
     _CreateAccountScreenState();
}
```

```dart
class _CreateAccountScreenState extends ConsumerState<CreateAccountScreen>
{
  File? image;
  DateTime? birthday;
  String gender = 'male';
  bool isLoading = false;
  late final TextEditingController _fNameController;
  late final TextEditingController _lNameController;
  late final TextEditingController _emailController;
  late final TextEditingController _passwordController;
  @override
  void initState() {
    _fNameController = TextEditingController();
    _lNameController = TextEditingController();
    _emailController = TextEditingController();
    _passwordController = TextEditingController();
    super.initState();
  }
  @override
  void dispose() {
    _fNameController.dispose();
    _lNameController.dispose();
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }
  Future<void> createAccount() async {
    if (_formKey.currentState!.validate()) {
      _formKey.currentState!.save();
      setState(() => isLoading = true);
      await ref
          .read(authProvider)
          .createAccount(
            fullName: '${_fNameController.text} ${_lNameController.text}',
            birthday: birthday ?? DateTime.now(),
            gender: gender,
            email: _emailController.text,
            password: _passwordController.text,
            image: image,
          )
```

```
          .then((credential) {
        if (!credential!.user!.emailVerified) {
          Navigator.pop(context);
        }
      }).catchError((_) {
        setState(() => isLoading = false);
      });
      setState(() => isLoading = false);
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: AppColors.realWhiteColor,
      appBar: AppBar(),
      body: SingleChildScrollView(
        child: Padding(
          padding: Constants.defaultPadding,
          child: Form(
            key: _formKey,
            child: Column(
              children: [
                GestureDetector(
                  onTap: () async {
                    image = await pickImage();
                    setState(() {});
                  },
                  child: PickImageWidget(image: image),
                ),
                const SizedBox(height: 20),
                Row(
                  children: [
                    // First Name Text Field
                    Expanded(
                      child: RoundTextField(
                        controller: _fNameController,
                        hintText: 'First name',
                        textInputAction: TextInputAction.next,
                        validator: validateName,
```

```
            ),
          ),
          const SizedBox(width: 10),
          // Last Name Text Field
          Expanded(
            child: RoundTextField(
              controller: _lNameController,
              hintText: 'Last name',
              textInputAction: TextInputAction.next,
              validator: validateName,
            ),
          ),
        ],
      ),
      const SizedBox(height: 20),
      BirthdayPicker(
        dateTime: birthday ?? DateTime.now(),
        onPressed: () async {
          birthday = await pickSimpleDate(
            context: context,
            date: birthday,
          );
          setState(() {});
        },
      ),
      const SizedBox(height: 20),
      GenderPicker(
        gender: gender,
        onChanged: (value) {
          gender = value ?? 'male';
          setState(() {});
        },
      ),
      const SizedBox(height: 20),
      // Phone number / email text field
      RoundTextField(
        controller: _emailController,
        hintText: 'Email',
        textInputAction: TextInputAction.next,
        keyboardType:
        TextInputType.emailAddress,
```

```
validator: validateEmail,
),
const SizedBox(height: 20),
// Password Text Field
                RoundTextField(
controller: _passwordController, hintText:
                  'Password',
textInputAction: TextInputAction.done, keyboardType:
                  TextInputType.visiblePassword, validator:
                  validatePassword,
isPassword: true,
),
const SizedBox(height: 20), isLoading
? const Center(child: CircularProgressIndicator())
: RoundButton(
onPressed: createAccount, label: 'Create
                        Account',
),
],
),
),
),
),
);
}
```

Output:

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Name: Varun Khubani
Division: D15A
Roll No:
30 Batch:
B

Experiment No 5

Aim: To apply routing in Flutter Application

Theory: In Flutter, routing refers to the navigation system that allows users to move between different screens or pages within an app. Flutter uses a widget-based approach for navigation, where each screen is represented by a widget. The `Navigator` class manages the stack of routes and facilitates transitions between them.

Routes are typically defined using the `MaterialPageRoute` class, providing a seamless and platform-aware transition between screens. Developers can use the `Navigator` to push new routes onto the stack or pop existing routes off it. Named routes help in easily identifying and navigating to specific screens.

Flutter also supports route arguments, allowing developers to pass data between screens. Additionally, the `Navigator` provides a flexible set of transitions, such as slide, fade, or custom animations, enhancing the user experience during navigation.

Overall, Flutter's routing system provides a structured and intuitive way to handle navigation within mobile and web applications.

Code:

```dart
import
'package:Nike_Clone/features/auth/presentation/screens/create_account_
screee.dart';
import
'package:Nike_Clone/features/auth/providers/auth_provider.dart';
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

import '/core/constants/constants.dart';
import '/core/widgets/round_button.dart';
import '/core/widgets/round_text_field.dart';
import '/features/auth/utils/utils.dart';

final _formKey = GlobalKey<FormState>();
```

login_Screen.dart

```dart
class LoginScreen extends ConsumerStatefulWidget {
  const LoginScreen({super.key});

  @override
  ConsumerState<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends ConsumerState<LoginScreen> {
  late final TextEditingController _emailController;
  late final TextEditingController _passwordController;

  bool isLoading = false;

  @override
  void initState() {
    _emailController = TextEditingController();
    _passwordController = TextEditingController();
    super.initState();
  }

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  Future<void> login() async {
    if (_formKey.currentState!.validate()) {
      _formKey.currentState!.save();
      setState(() => isLoading = true);
      await
      ref.read(authProvider).signIn(
            email: _emailController.text,
            password: _passwordController.text,
          );
      setState(() => isLoading = false);
    }
  }

  @override
```

```
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(),
    body: Padding(
      padding: Constants.defaultPadding,
      child: Column(
        mainAxisSize: MainAxisSize.max,
        mainAxisAlignment:
        MainAxisAlignment.spaceAround, children: [
          Image.asset(
            'assets/icons/fb_logo.png',
            width: 60,
          ),
          Form(
            key: _formKey,
            child: Column(
              children: [
                RoundTextField(
                  controller:
                  _emailController, hintText:
                  'Email',
                  keyboardType: TextInputType.emailAddress,
                  textInputAction: TextInputAction.next,
                  validator: validateEmail,
                ),
                const SizedBox(height: 15),
                RoundTextField(
                  controller: _passwordController,
                  hintText: 'Password',
                  keyboardType: TextInputType.visiblePassword,
                  textInputAction: TextInputAction.done,
                  isPassword: true,
                  validator: validatePassword,
                ),
                const SizedBox(height: 15),
                RoundButton(onPressed: login, label: 'Login'),
                const SizedBox(height: 15),
                const Text(
                  'Forget Password',
                  style: TextStyle(fontSize: 18),
                ),
```

```
],
),
),
            Column(
              children
              : [
                RoundButton(
                  onPressed:
                  () {
                    Navigator.of(context).pushName
                      d(
                      CreateAccountScreen.routeNam
                      e,
);
},
label: 'Create new account', color:
                    Colors.transparent,
),
                Image.asset(
                  'assets/icons/meta.png
                  ', height: 50,
),
],
),
],
),
),
);
}
}
```

```
import 'dart:io';

import 'package:facebook_clone/core/constants/app_colors.dart';
import 'package:facebook_clone/core/constants/constants.dart';
import 'package:facebook_clone/core/utils/utils.dart';
import 'package:facebook_clone/core/widgets/pick_image_widget.dart';
import 'package:facebook_clone/core/widgets/round_button.dart';
import 'package:facebook_clone/core/widgets/round_text_field.dart';
import
'package:facebook_clone/features/auth/presentation/widgets/birthday_picker
.dart';
```

Create_account_screen.dart

```dart
import
'package:Nike_Clone/features/auth/presentation/widgets/gender_picker.d
art';
import
'package:Nike_Clone/features/auth/providers/auth_provider.dart';
import 'package:Nike_Clone/features/auth/utils/utils.dart'; import
'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

final _formKey = GlobalKey<FormState>();

class CreateAccountScreen extends ConsumerStatefulWidget {
  const CreateAccountScreen({super.key});

  static const routeName = '/create-account';

  @override
  ConsumerState<CreateAccountScreen> createState() =>
      _CreateAccountScreenState();
}

class _CreateAccountScreenState extends ConsumerState<CreateAccountScreen>
{
  File? image;
  DateTime? birthday;
  String gender = 'male';
  bool isLoading = false;

  // controllers
  late final TextEditingController _fNameController;
  late final TextEditingController _lNameController;
  late final TextEditingController _emailController;
  late final TextEditingController _passwordController;

  @override
  void initState() {
    _fNameController = TextEditingController();
    _lNameController = TextEditingController();
    _emailController = TextEditingController();
    _passwordController = TextEditingController();
```

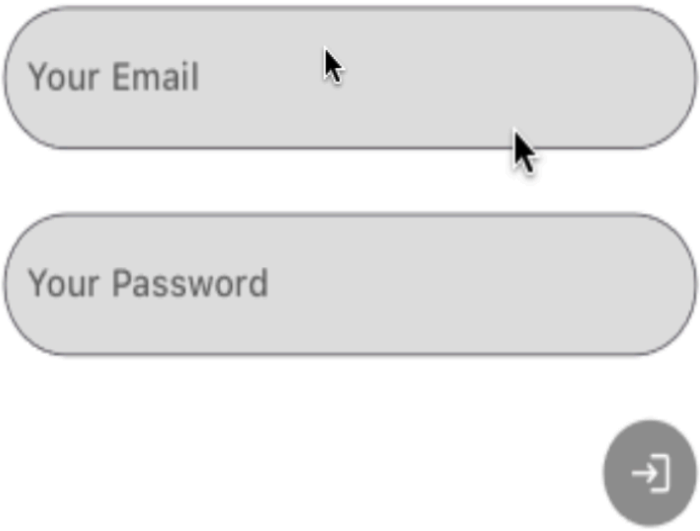
```dart
  super.initState();
}

@override
void dispose() {
  _fNameController.dispose();
  _lNameController.dispose();
  _emailController.dispose();
  _passwordController.dispose();
  super.dispose();
}

Future<void> createAccount() async {
  if (_formKey.currentState!.validate()) {
    _formKey.currentState!.save();
    setState(() => isLoading = true);
    await ref
        .read(authProvider)
        .createAccount(
          fullName: '${_fNameController.text} ${_lNameController.text}',
          birthday: birthday ?? DateTime.now(),
          gender: gender,
          email: _emailController.text,
          password: _passwordController.text,
          image: image,
        )
        .then((credential) {
      if (!credential!.user!.emailVerified) {
        Navigator.pop(context);
      }
    }).catchError((_) {
      setState(() => isLoading = false);
    });
    setState(() => isLoading = false);
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
```

```
    backgroundColor: AppColors.realWhiteColor,
    appBar: AppBar(),
    body: SingleChildScrollView(
      child: Padding(
        padding: Constants.defaultPadding,
        child: Form(
          key: _formKey,
          child: Column(
            children: [
              GestureDetector(
                onTap: () async {
                  image = await pickImage();
                  setState(() {});
                },
                child: PickImageWidget(image: image),
              ),
              const SizedBox(height: 20),
              Row(
                children: [
                  // First Name Text Field
                  Expanded(
                    child: RoundTextField(
                      controller: _fNameController,
                      hintText: 'First name',
                      textInputAction: TextInputAction.next,
                      validator: validateName,
                    ),
                  ),
                  const SizedBox(width: 10),
                  // Last Name Text Field
                  Expanded(
                    child: RoundTextField(
                      controller: _lNameController,
                      hintText: 'Last name',
                      textInputAction: TextInputAction.next,
                      validator: validateName,
                    ),
                  ),
                ],
              ),
```

```
              const SizedBox(height: 20),
              BirthdayPicker(
                dateTime: birthday ?? DateTime.now(),
                onPressed: () async {
                  birthday = await pickSimpleDate(
                    context: context,
                    date: birthday,
                  );
                  setState(() {});
                },
              ),
              const SizedBox(height: 20),
              GenderPicker(
                gender: gender,
                onChanged: (value) {
                  gender = value ?? 'male';
                  setState(() {});
                },
              ),
              const SizedBox(height: 20),
              // Phone number / email text field
              RoundTextField(
                controller: _emailController,
                hintText: 'Email',
                textInputAction: TextInputAction.next,
                keyboardType: TextInputType.emailAddress,
                validator: validateEmail,
              ),
              const SizedBox(height: 20),
              // Password Text Field
              RoundTextField(
                controller: _passwordController,
                hintText: 'Password',
                textInputAction: TextInputAction.done,
                keyboardType: TextInputType.visiblePassword,
                validator: validatePassword,
                isPassword: true,
              ),
              const SizedBox(height: 20),
              isLoading
```

```
? const Center(child: CircularProgressIndicator())
: RoundButton(
onPressed: createAccount, label: 'Create
                        Account',
),
],
),
),
),
),
);
}
}
```

Output:



MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Name: Varun Khubani
Division: D15A
Roll No:
30 Batch:
B

Experiment No 6

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.

```
// 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) {
return web;
}
    switch (defaultTargetPlatform) {
      case TargetPlatform.android:
return android;
      case TargetPlatform.iOS:
        return ios;
      case TargetPlatform.macOS:
          throw UnsupportedError(
```

Firebase API code:

```
                                        'DefaultFirebaseOptions have not
                                        been configured for macos - '
again.',                                'you can reconfigure this by
                                        running the FlutterFire CLI

                                      );
      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.',
        );
      default:
        throw UnsupportedError(
          'DefaultFirebaseOptions are not supported for this platform.',
        );
    }
  }

  static const FirebaseOptions web = FirebaseOptions(
    apiKey: 'AIzaSyBg9-swGAvdOLn4vhuIu2PM70VW5ew7wqU',
    appId: '1:466675301682:web:61da688dacd6dccf32ab2b',
    messagingSenderId: '466675301682',
    projectId: Nike_Shoes-5f3aa',
    authDomain: 'Nike_Shoes-5f3aa.firebaseapp.com',
    storageBucket: Nike_Shoes-5f3aa.appspot.com',
    measurementId: 'G-HZCQMPK1MZ',
  );

  static const FirebaseOptions android = FirebaseOptions(
    apiKey: 'AIzaSyAbozls46qv0OvzLeNxnuq-nQRpiz_7gVw',
    appId: '1:466675301682:android:40a4a7be8fb3eba732ab2b',
    messagingSenderId: '466675301682',
    projectId: 'Nike-clone-5f3aa',
    storageBucket: 'Nike-clone-5f3aa.appspot.com',
  );
```

```
  static const FirebaseOptions ios =
    FirebaseOptions( apiKey:
    'AIzaSyB1gwDnGzEv1MZ3VglmVCGgzrzKnXHabSQ',
    appId:
    '1:466675301682:ios:6283eb03b0d3702832ab2b',
messagingSenderId: '466675301682',
    projectId:
    'facebook-clone-5f3aa',
storageBucket: 'facebook-clone-5f3aa.appspot.com',
    iosBundleId: 'com.example.facebookClone',
);
}
```

Login Screen Code:

```
import
'package:Nike_Clone/features/auth/presentation/screens/create_account_
screee.dart';
import
'package:Nike_Clone/features/auth/providers/auth_provider.dart';
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

import '/core/constants/constants.dart';
import '/core/widgets/round_button.dart';
import '/core/widgets/round_text_field.dart';
import '/features/auth/utils/utils.dart';

final _formKey = GlobalKey<FormState>();

class LoginScreen extends ConsumerStatefulWidget {
  const LoginScreen({super.key});

@override
ConsumerState<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends ConsumerState<LoginScreen> {
  late final TextEditingController _emailController;
late final TextEditingController _passwordController;
```

```dart
bool isLoading = false;

@override
void initState() {
  _emailController = TextEditingController();
  _passwordController = TextEditingController();
  super.initState();
}

@override
void dispose() {
  _emailController.dispose();
  _passwordController.dispose();
  super.dispose();
}

Future<void> login() async {
  if (_formKey.currentState!.validate()) {
    _formKey.currentState!.save();
    setState(() => isLoading = true);
    await
    ref.read(authProvider).signIn(
          email: _emailController.text,
          password: _passwordController.text,
        );
    setState(() => isLoading = false);
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(),
    body: Padding(
      padding: Constants.defaultPadding,
      child: Column(
        mainAxisSize: MainAxisSize.max,
        mainAxisAlignment:
        MainAxisAlignment.spaceAround, children: [
          Image.asset(
            'assets/icons/fb_logo.png',
```

```
            width: 60,
          ),
          Form(
            key: _formKey,
            child: Column(
              children: [
                RoundTextField(
                  controller:
                  _emailController, hintText:
                  'Email',
                  keyboardType: TextInputType.emailAddress,
                  textInputAction: TextInputAction.next,
                  validator: validateEmail,
                ),
                const SizedBox(height: 15),
                RoundTextField(
                  controller: _passwordController,
                  hintText: 'Password',
                  keyboardType: TextInputType.visiblePassword,
                  textInputAction: TextInputAction.done,
                  isPassword: true,
                  validator: validatePassword,
                ),
                const SizedBox(height: 15),
                RoundButton(onPressed: login, label: 'Login'),
                const SizedBox(height: 15),
                const Text(
                  'Forget Password',
                  style: TextStyle(fontSize: 18),
                ),
              ],
            ),
          ),
          Column(
            children: [
              RoundButton(
                onPressed: () {
                  Navigator.of(context).pushNamed(
                    CreateAccountScreen.routeName,
                  );
                },
```

```
label: 'Create new account', color:
                  Colors.transparent,
),
                Image.asset(
                  'assets/icons/main.png
                  ', height: 50,
),
],
),
],
),
),
);
}
}
```

Create Account Screen Code:

```
import 'dart:io';

import 'package:Nike_Clone/core/constants/app_colors.dart';
import 'package:Nike_Clone/core/constants/constants.dart';
import 'package:Nike_Clone/core/utils/utils.dart';
import 'package:Nike_Clone/core/widgets/pick_image_widget.dart';
import 'package:Nike_Clone/core/widgets/round_button.dart'; import
'package:Nike_Clone/core/widgets/round_text_field.dart'; import
'package:Nike_Clone/features/auth/presentation/widgets/birthday_picker
.dart';
import
'package:Nike_Clone/features/auth/presentation/widgets/gender_picker.d
art';
import
'package:Nike_Clone/features/auth/providers/auth_provider.dart';
import 'package:Nike_Clone/features/auth/utils/utils.dart'; import
'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

final _formKey = GlobalKey<FormState>();
```

```dart
class CreateAccountScreen extends ConsumerStatefulWidget {
  const CreateAccountScreen({super.key});

  static const routeName = '/create-account';

  @override
  ConsumerState<CreateAccountScreen> createState() =>
      _CreateAccountScreenState();
}

class _CreateAccountScreenState extends ConsumerState<CreateAccountScreen>
{
  File? image;
  DateTime? birthday;
  String gender = 'male';
  bool isLoading = false;

  // controllers
  late final TextEditingController _fNameController;
  late final TextEditingController _lNameController;
  late final TextEditingController _emailController;
  late final TextEditingController _passwordController;

  @override
  void initState() {
    _fNameController = TextEditingController();
    _lNameController = TextEditingController();
    _emailController = TextEditingController();
    _passwordController = TextEditingController();
    super.initState();
  }

  @override
  void dispose() {
    _fNameController.dispose();
    _lNameController.dispose();
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }
```

```dart
Future<void> createAccount() async {
  if (_formKey.currentState!.validate()) {
    _formKey.currentState!.save();
    setState(() => isLoading = true);
    await ref
        .read(authProvider)
        .createAccount(
          fullName: '${_fNameController.text} ${_lNameController.text}',
          birthday: birthday ?? DateTime.now(),
          gender: gender,
          email: _emailController.text,
          password: _passwordController.text,
          image: image,
        )
        .then((credential) {
      if (!credential!.user!.emailVerified) {
        Navigator.pop(context);
      }
    }).catchError((_) {
      setState(() => isLoading = false);
    });
    setState(() => isLoading = false);
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: AppColors.realWhiteColor,
    appBar: AppBar(),
    body: SingleChildScrollView(
      child: Padding(
        padding: Constants.defaultPadding,
        child: Form(
          key: _formKey,
          child: Column(
            children: [
              GestureDetector(
                onTap: () async {
```

```
                image = await pickImage();
                setState(() {});
              },
              child: PickImageWidget(image: image),
            ),
            const SizedBox(height: 20),
            Row(
              children: [
                // First Name Text Field
                Expanded(
                  child: RoundTextField(
                    controller: _fNameController,
                    hintText: 'First name',
                    textInputAction: TextInputAction.next,
                    validator: validateName,
                  ),
                ),
                const SizedBox(width: 10),
                // Last Name Text Field
                Expanded(
                  child: RoundTextField(
                    controller: _lNameController,
                    hintText: 'Last name',
                    textInputAction: TextInputAction.next,
                    validator: validateName,
                  ),
                ),
              ],
            ),
            const SizedBox(height: 20),
            BirthdayPicker(
              dateTime: birthday ?? DateTime.now(),
              onPressed: () async {
                birthday = await pickSimpleDate(
                  context: context,
                  date: birthday,
                );
                setState(() {});
              },
            ),
```

```
              const SizedBox(height: 20),
              GenderPicker(
                gender: gender,
                onChanged: (value) {
                  gender = value ?? 'male';
                  setState(() {});
                },
              ),
              const SizedBox(height: 20),
              // Phone number / email text field
              RoundTextField(
                controller: _emailController,
                hintText: 'Email',
                textInputAction: TextInputAction.next,
                keyboardType: TextInputType.emailAddress,
                validator: validateEmail,
              ),
              const SizedBox(height: 20),
              // Password Text Field
              RoundTextField(
                controller: _passwordController,
                hintText: 'Password',
                textInputAction: TextInputAction.done,
                keyboardType: TextInputType.visiblePassword,
                validator: validatePassword,
                isPassword: true,
              ),
              const SizedBox(height: 20),
              isLoading
                  ? const Center(child: CircularProgressIndicator())
                  : RoundButton(
                      onPressed: createAccount,
                      label: 'Create Account',
                    ),
            ],
          ),
        ),
      ),
    ),
  );
```

```
}
}
```

Code to HomePage:

```
import 'dart:async';

import 'package:cloud_firestore/cloud_firestore.dart';
import
'package:Nike_Clone/core/constants/firebaes_collection_names.dart';
import 'package:Nike_Clone/core/constants/firebase_field_names.dart';
import 'package:Nike_Clone/features/posts/models/post.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

final getAllPostsProvider =
StreamProvider.autoDispose<Iterable<Post>>((ref) {
final controller = StreamController<Iterable<Post>>();

final sub = FirebaseFirestore.instance
.collection(FirebaseCollectionNames.posts)
.orderBy(FirebaseFieldNames.datePublished, descending: true)
.snapshots()
.listen((snapshot) {
    final posts = snapshot.docs.map(
      (postData) => Post.fromMap(
postData.data(),
),
);
controller.sink.add(posts);
});

  ref.onDispose(() {
    sub.cancel();
    controller.close();
});

return controller.stream;
});
```

CartPage:

```
import 'package:Nike_Clone/core/constants/app_colors.dart';
import 'package:Nike_Clone/core/screens/loader.dart'; import
'package:Nike_Clone/core/utils/utils.dart';
import
'package:Nike_Clone/features/chat/presentation/widgets/chats_user_info
.dart';
import
'package:Nike_Clone/features/chat/presentation/widgets/messages_list.d
art';
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

import
'package:Nike_Clone/features/chat/providers/chat_provider.dart';
import 'package:Nike_Clone/font_awesome_flutter.dart'; import
'package:image_picker/image_picker.dart';

class ChatScreen extends ConsumerStatefulWidget
  { const ChatScreen({
    super.key,
    required this.userId,
  });

  final String userId;
  static const routeName = '/home_screen';
  @override
  ConsumerState<ConsumerStatefulWidget> createState() =>
_ChatScreenState();
}

class _ChatScreenState extends ConsumerState<ChatScreen> {
  late final TextEditingController messageController;
  late final String chatroomId;

  @override
  void initState() {
    messageController = TextEditingController();
```

```
    super.initState();
  }

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

  @override
  Widget build(BuildContext context) {
    return FutureBuilder(
      future: ref.watch(chatProvider).createChatroom(userId:
widget.userId),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return const Loader();
        }
        chatroomId = snapshot.data ?? 'No customer Id';
        return Scaffold(
          backgroundColor: AppColors.realWhiteColor,
          appBar: AppBar(
            leading: IconButton(
              onPressed: Navigator.of(context).pop,
              icon: const Icon(
                Icons.arrow_back_ios,
                color: AppColors.messengerBlue,
              ),
            ),
            titleSpacing: 0,
            title: ChatUserInfo(
              userId: widget.userId,
            ),
          ),
          body: Column(
            children: [
              Expanded(
                child: MessagesList(
```

```
                chatroomId: chatroomId,
              ),
            ),
            const Divider(),
            _buildMessageInput(),
          ],
        ),
      );
    },
  );
}

// Chat Text Field
Widget _buildMessageInput() {
  return Container(
    padding: const EdgeInsets.all(8.0),
    child: Row(
      children: [
        IconButton(
          icon: const Icon(
            Icons.image,
            color: AppColors.messengerDarkGrey,
          ),
          onPressed: () async {
            final image = await pickImage();
            if (image == null) return;
            await ref.read(chatProvider).sendFileMessage(
                  file: image,
                  chatroomId: chatroomId,
                  receiverId: widget.userId,
                  messageType: 'image',
                );
          },
        ),
        IconButton(
          icon: const Icon(
            FontAwesomeIcons.video,
            color: AppColors.messengerDarkGrey,
            size: 20,
          ),
```

```
            onPressed: () async {
              final video = await pickVideo();
              if (video == null) return;
              await ref.read(chatProvider).sendFileMessage(
                    file: video,
                    chatroomId: chatroomId,
                    receiverId: widget.userId,
                    messageType: 'video',
                  );
            },
          ),
          // Text Field
          Expanded(
            child: Container(
              height: 40,
              decoration: BoxDecoration(
                color: AppColors.messengerGrey,
                borderRadius: BorderRadius.circular(15),
              ),
              child: TextField(
                controller: messageController,
                decoration: const InputDecoration(
                  hintText: 'Aa',
                  hintStyle: TextStyle(),
                  border: InputBorder.none,
                  contentPadding: EdgeInsets.only(
                    left: 20,
                    bottom: 10,
                  ),
                ),
                textInputAction: TextInputAction.done,
              ),
            ),
          ),
          IconButton(
            icon: const Icon(
              Icons.send,
              color: AppColors.messengerBlue,
            ),
            onPressed: () async {
```

```
// Add functionality to handle send button press await
              ref.read(RenderImage).sendErrorMeassage(
message: messageController.text, chatroomId:
                    chatroomId, receiverId:
                    widget.userId,
);
RenderImage.clear();
},
),
],
),
);
}
}
```

Create OrderSummary:

```
import 'dart:io';

import 'package:Nike_Clone/core/constants/app_colors.dart';
import 'package:Nike_Clone/core/constants/constants.dart';
import 'package:Nike_Clone/core/utils/utils.dart';
import 'package:Nike_Clone/core/widgets/round_button.dart';
import
'package:Nike_Clone/features/posts/presentation/widgets/image_video_vi
ew.dart';
import
'package:Nike_Clone/features/posts/presentation/widgets/profile_info.d
art';
import
'package:Nike_Clone/features/posts/providers/posts_provider.dart';
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

class CreatePostScreen extends ConsumerStatefulWidget {
  const CreatePostScreen({super.key});

  static const routeName = '/create-post';
```

```
  @override
  ConsumerState<CreatePostScreen> createState() =>
_CreatePostScreenState();
}

class _CreatePostScreenState extends ConsumerState<CreatePostScreen> {
  late final TextEditingController _postController;
  File? file;
  String fileType = 'image';
  bool isLoading = false;

  @override
  void initState() {
    _postController = TextEditingController();
    super.initState();
  }

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        actions: [
          TextButton(
            onPressed: makePost,
            child: const Text('Post'),
          ),
        ],
      ),
      body: SingleChildScrollView(
        child: Padding(
          padding: Constants.defaultPadding,
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
```

```
            children: [
              const ProfileInfo(),
              // post text field
              TextField(
                controller: _postController,
                decoration: const InputDecoration(
                  border: InputBorder.none,
                  hintText: 'What\'s on your
                  mind?', hintStyle: TextStyle(
                    fontSize: 18,
                    color: AppColors.darkGreyColor,
                  ),
                ),
                keyboardType: TextInputType.multiline,
                minLines: 1,
                maxLines: 10,
              ),
              const SizedBox(height: 20),
              file != null
                  ? ImageVideoView(
                      file: file!,
                      fileType: fileType,
                    )
                  : PickFileWidget(
                      pickImage: () async {
                        fileType = 'image';
                        file = await pickImage();
                        setState(() {});
                      },
                      pickVideo: () async {
                        fileType = 'video';
                        file = await pickVideo();
                        setState(() {});
                      },
                    ),
              const SizedBox(height: 20),
              isLoading
                  ? const Center(
                      child: CircularProgressIndicator(),
                    )
```

```
                  : RoundButton(
                      onPressed: makePost,
                      label: 'Post',
                    ),
            ],
          ),
        ),
      ),
    );
  }

  Future<void> makePost() async {
    setState(() => isLoading = true);
    await ref
        .read(postsProvider)
        .makePost(
          content: _postController.text,
          file: file!,
          postType: fileType,
        )
        .then((value) {
      Navigator.of(context).pop();
    }).catchError((_) {
      setState(() => isLoading = false);
    });
    setState(() => isLoading = false);
  }
}

class PickFileWidget extends StatelessWidget {
  const PickFileWidget({
    super.key,
    required this.pickImage,
    required this.pickVideo,
  });

  final VoidCallback pickImage;
  final VoidCallback pickVideo;

  @override
```

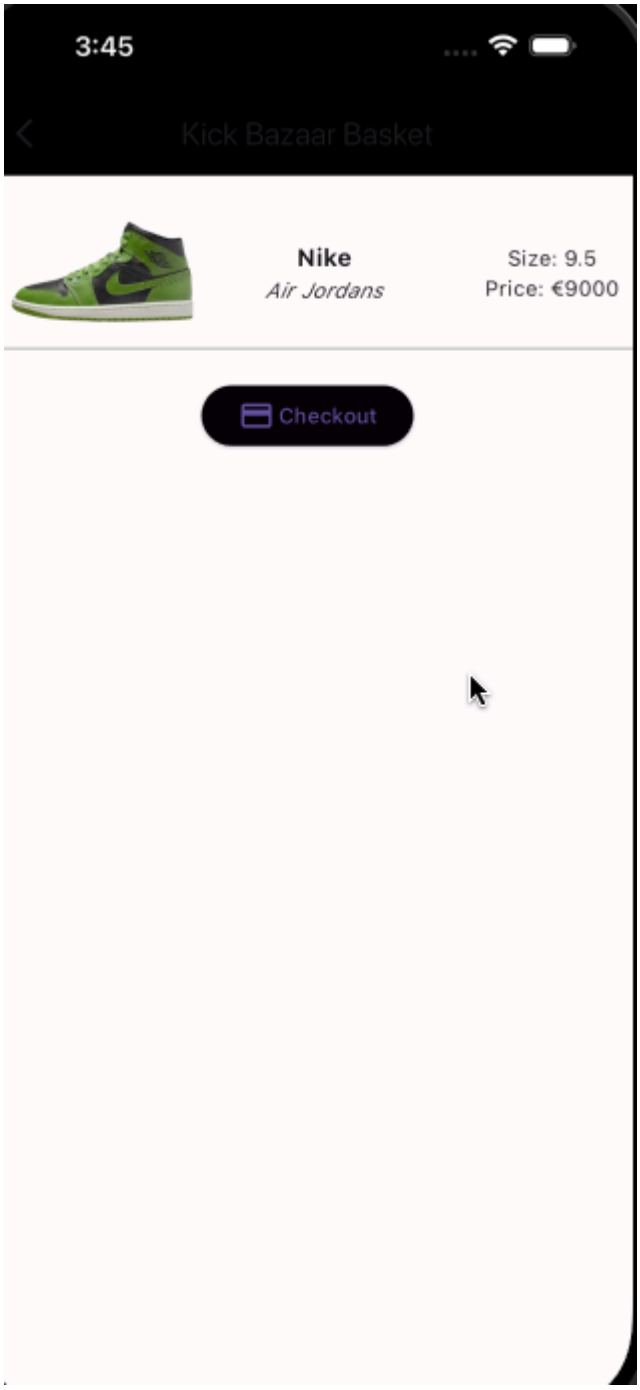
```
  Widget build(BuildContext context)
    { return Column(
      children: [
        TextButto
        n(
onPressed: pickImage,
child: const Text('Pick Image'),
),
const Divider(),
        TextButton(
onPressed: pickVideo,
child: const Text('Pick Video'),
),
],
);
}
```

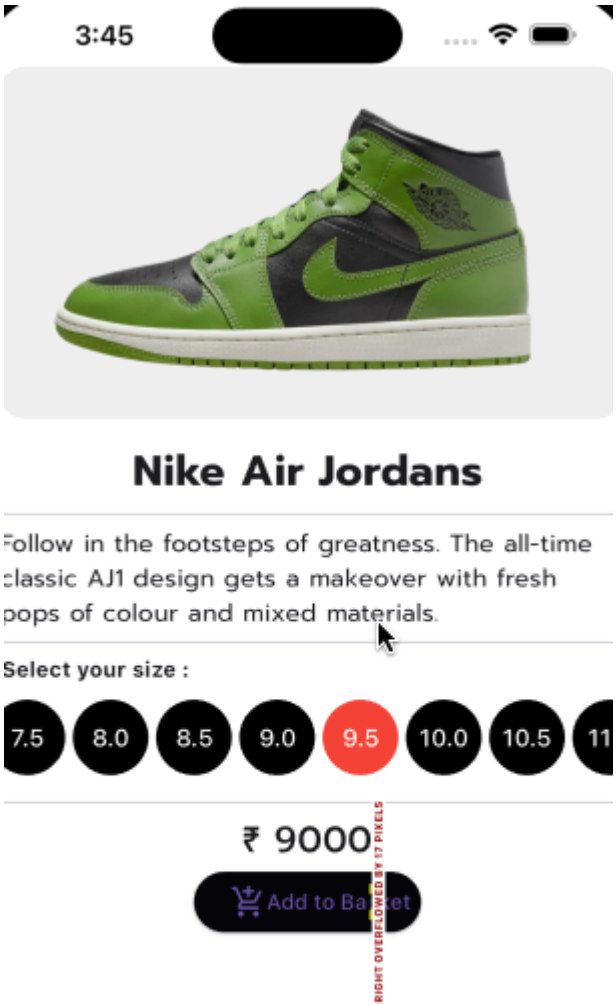
Get allOrders

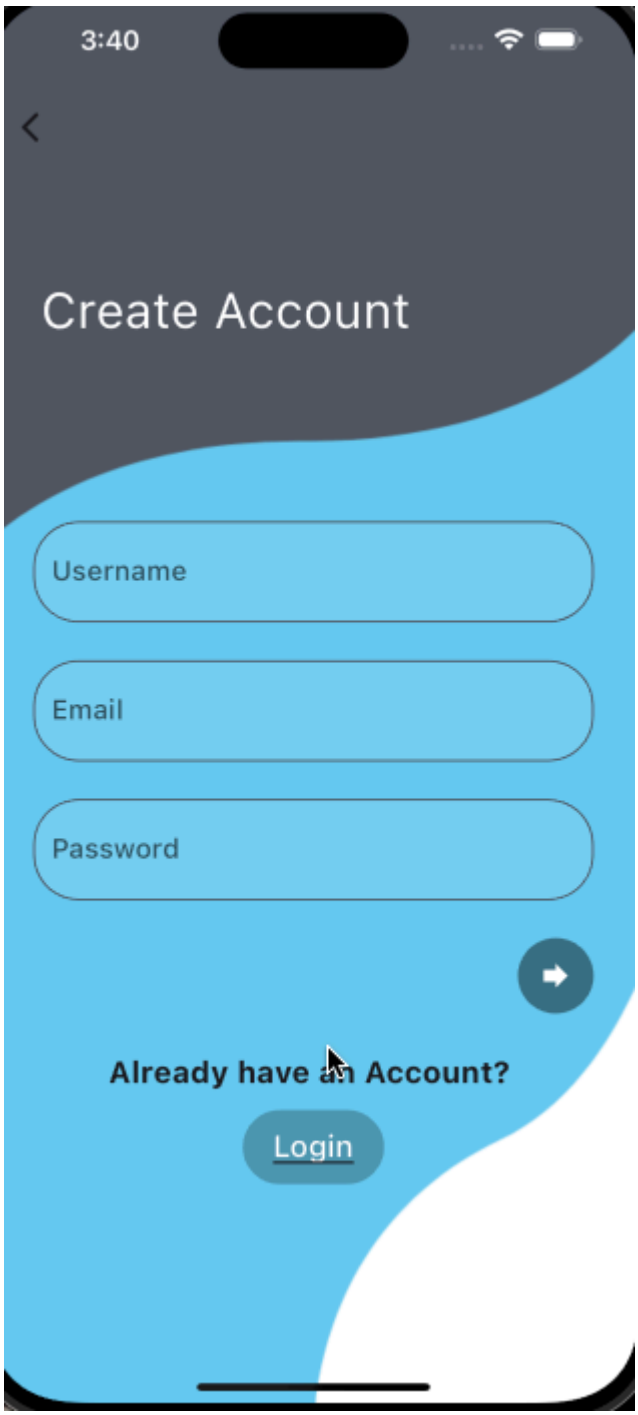
```
import 'package:Nike_Clone/core/constants/firebase_field_names.dart';
import 'package:flutter/foundation.dart' show immutable;

@immutable
class Comment {
final String commentId;
  final String authorId;
  final String postId;
  final String text;
final DateTime createdAt;
  final List<String> likes;

const Comment({
required this.commentId,
    required this.authorId,
    required this.postId,
    required this.text,
    required this.createdAt,
    required this.likes,
});
```

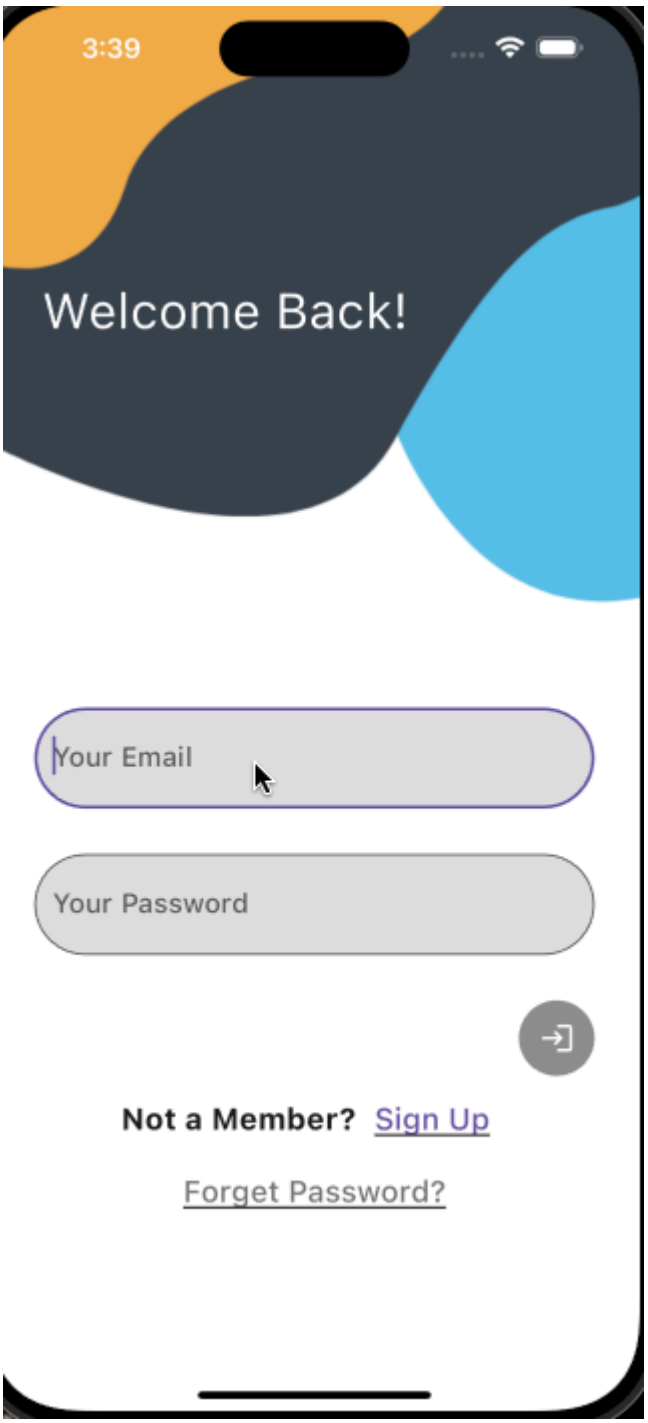
```
  Map<String, dynamic> toMap()
    { return <String,
    dynamic>{
FirebaseFieldNames.commentId: commentId,
      FirebaseFieldNames.authorId: authorId,
      FirebaseFieldNames.postId: postId,
      FirebaseFieldNames.text: text,
FirebaseFieldNames.createdAt: createdAt.millisecondsSinceEpoch,
      FirebaseFieldNames.likes: likes,
};
}

  factory Comment.fromMap(Map<String, dynamic> map)
    { return Comment(
commentId: map[FirebaseFieldNames.commentId] ?? '',
      authorId: map[FirebaseFieldNames.authorId] ?? '',
      postId: map[FirebaseFieldNames.postId] ?? '',
text: map[FirebaseFieldNames.text] ?? '',
      createdAt:
      DateTime.fromMillisecondsSinceEpoch(
map[FirebaseFieldNames.createdAt] ?? 0,
),
      likes: List<String>.from(
        (map[FirebaseFieldNames.likes] ??
        []),
),
);
}
}
```

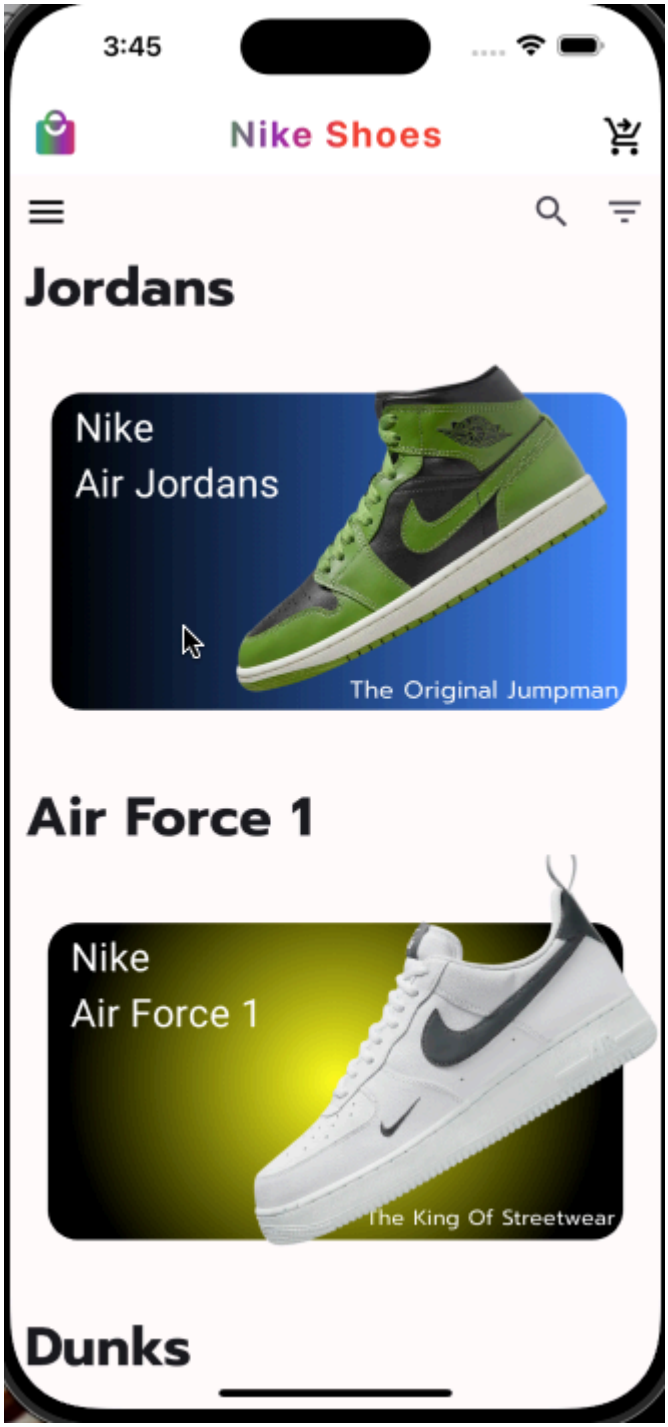
Output:


3:45
Kick Bazaar Basket
Nike
Air Jordans
Size: 9.5
Price: €9000
Checkout


3:45
Nike Air Jordans
Follow in the footsteps of greatness. The all-time classic AJ1 design gets a makeover with fresh pops of colour and mixed materials.
Select your size :
7.5
8.0
8.5
9.0
9.5
10.0
10.5
11
₹ 9000
Add to Basket
RIGHT OVERFLOWED BY 17 PIXELS


3:40
Create Account
Username
Email
Password
Already have an Account?
Login


3:39
Welcome Back!
Your Email
Your Password
Not a Member? Sign Up
Forget Password?


3:45
Nike Shoes
Jordans
Nike
Air Jordans
The Original Jumpman
Air Force 1
Nike
Air Force 1
The King Of Streetwear
Dunks

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.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

Name: Varun Khubani
Division: D15A
Roll No:
30 Batch:
B

Experiment No 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.

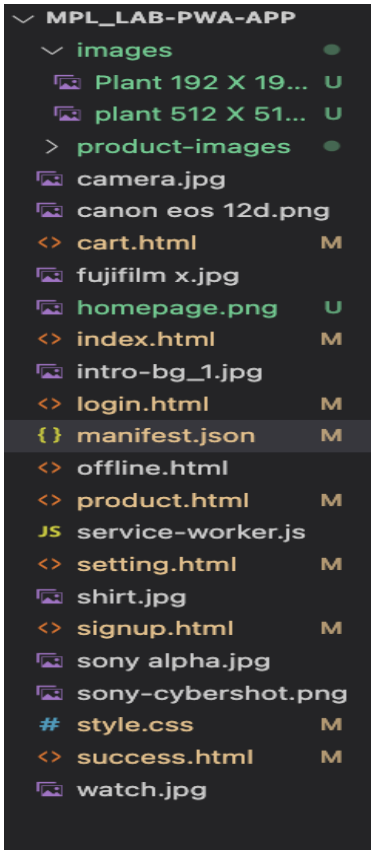
Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
Installable — They allow the user to “save” the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to face all the steps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

```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"> 300
</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"> 130
</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>
```

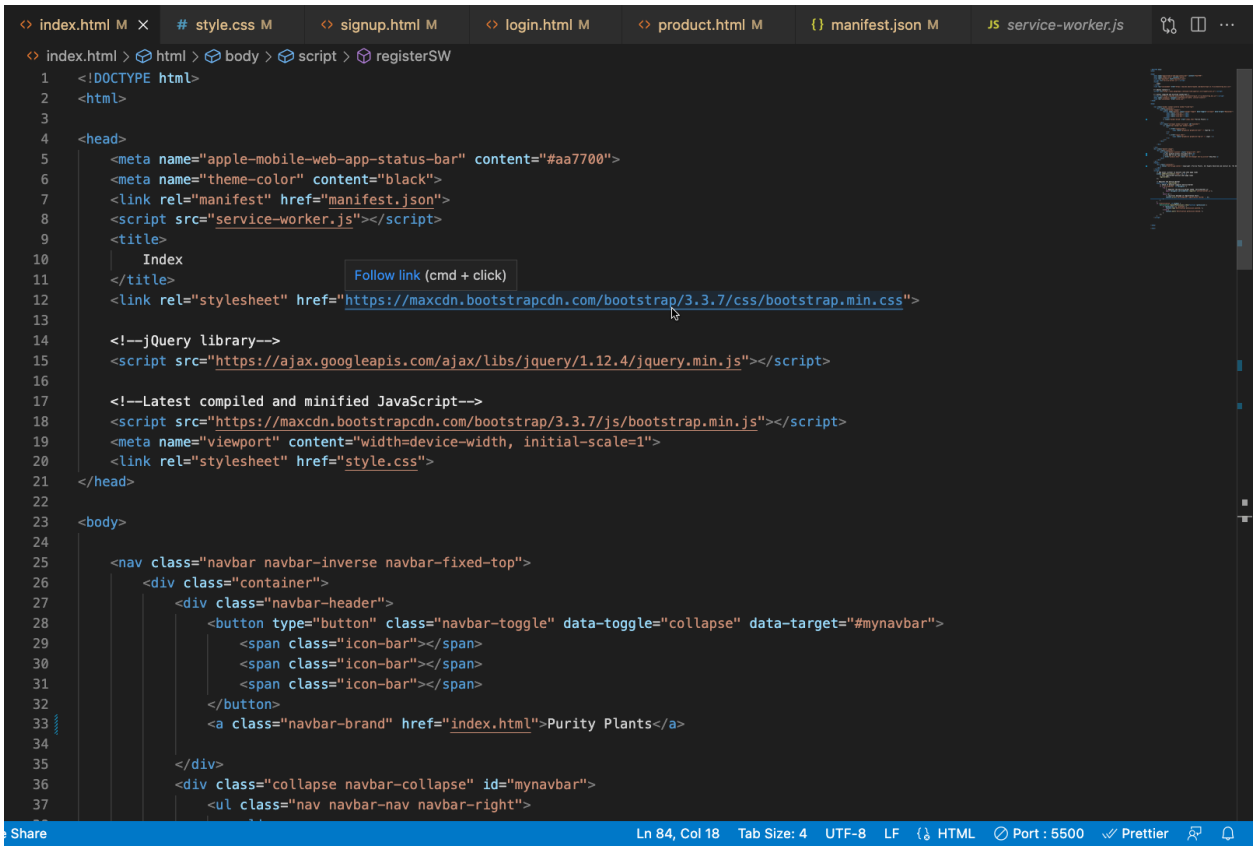
```
            <div class="row text-center">

            <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">309 </div>

            </div>

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

            <div class="  col-md-3 col-sm-6  thumbnail " >
            <img src="/product-images/11.webp">
              <div class="caption">
            <h2>Night-blooming Cereus:</h2>
            <p>Fragrant nocturnal blooms.</p>
            </div>
            <div class=" btn btn-primary  btn-block btn-md btn-success">249 </div>
            </div>

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

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

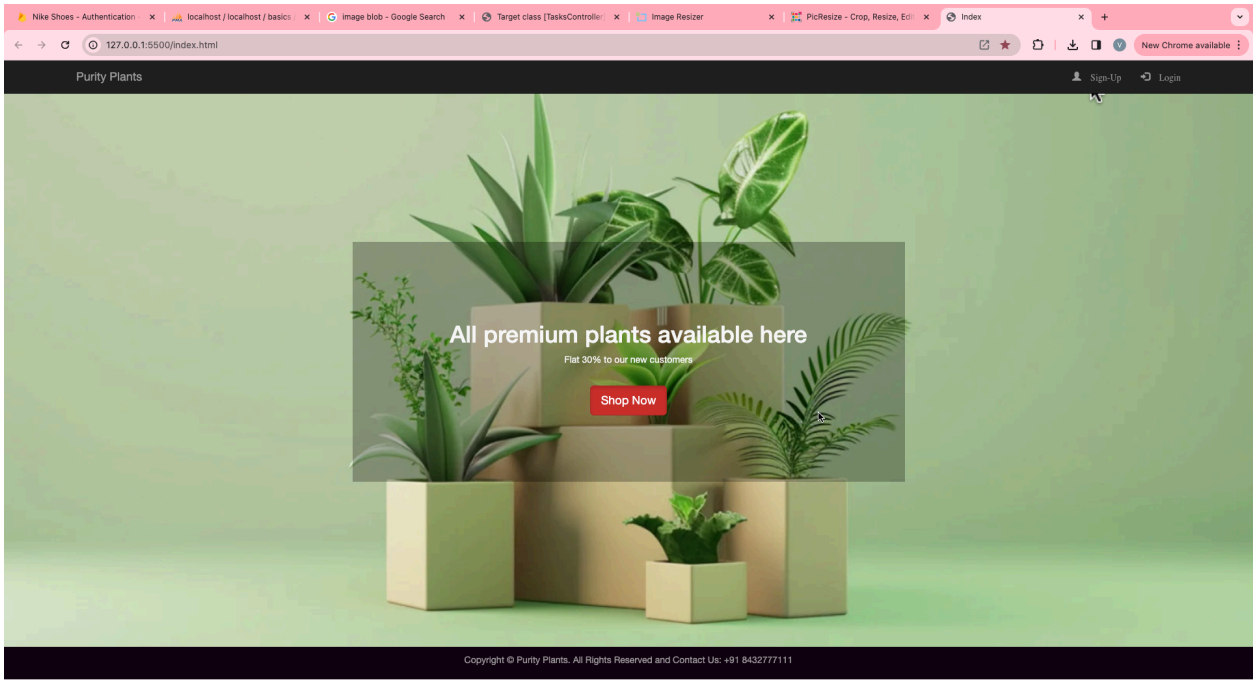
```
                                </div>
                    </footer>




        </body>
    </html>
```

Style.css

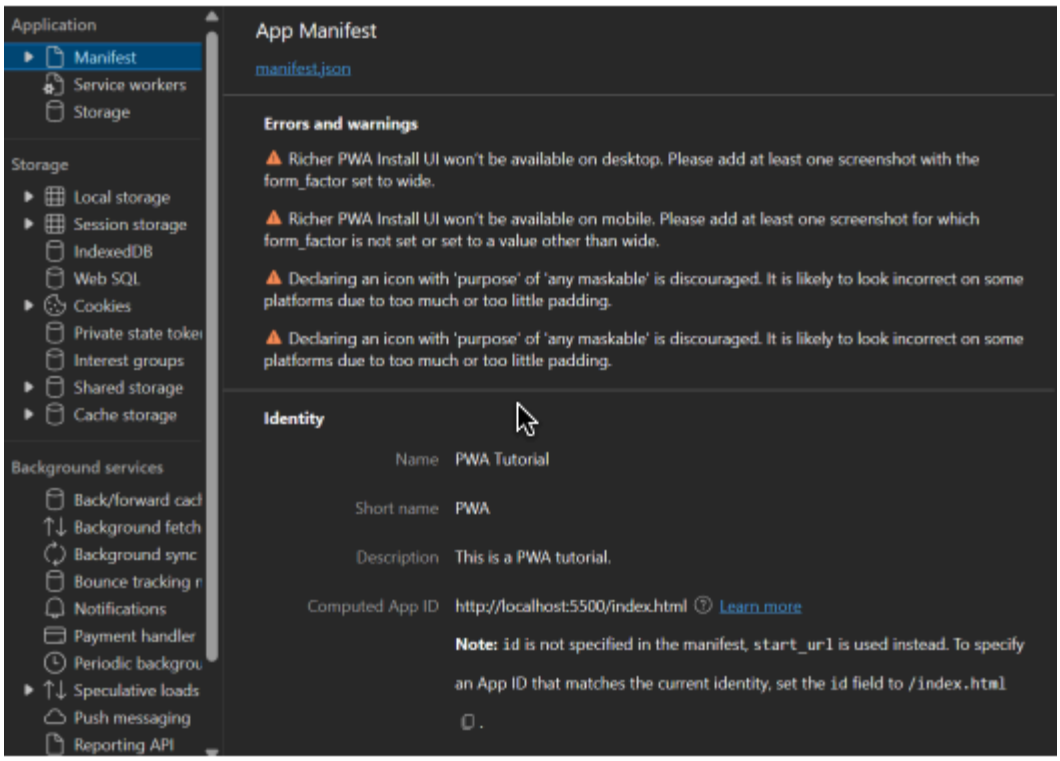
```
.banner-image
{padding-top: 75px;
padding-bottom: 50px;
text-align: center;
color: #f8f8f8;
background: url(homepage.png) no-repeat center center;
background-size: cover;
}
.banner-content{
 position: relative;
 padding-top: 6%;
padding-bottom: 6%;

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

}
```

Starting the Server
:

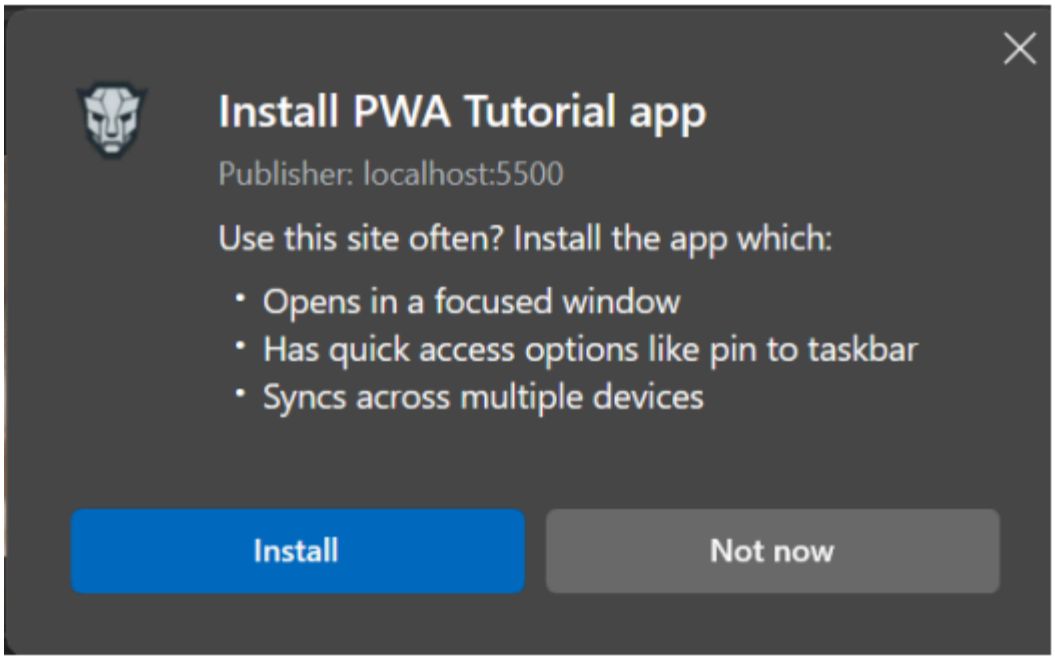


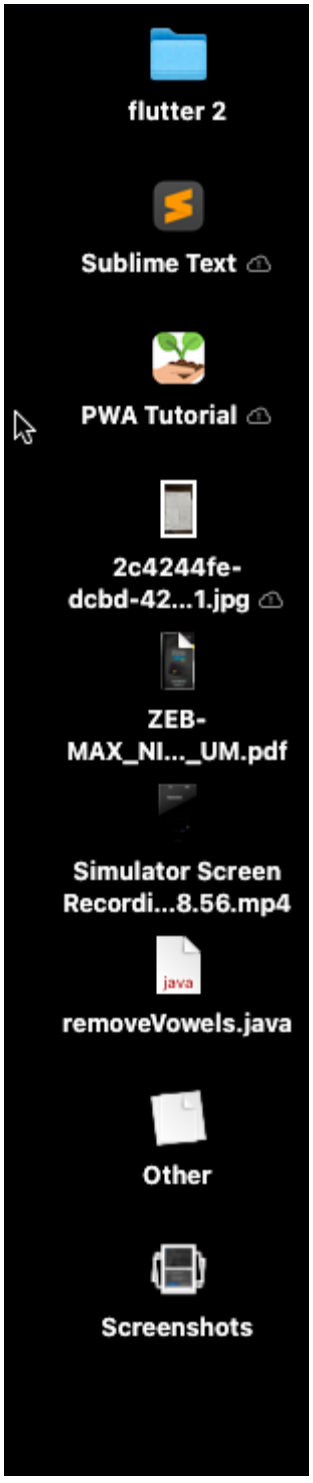


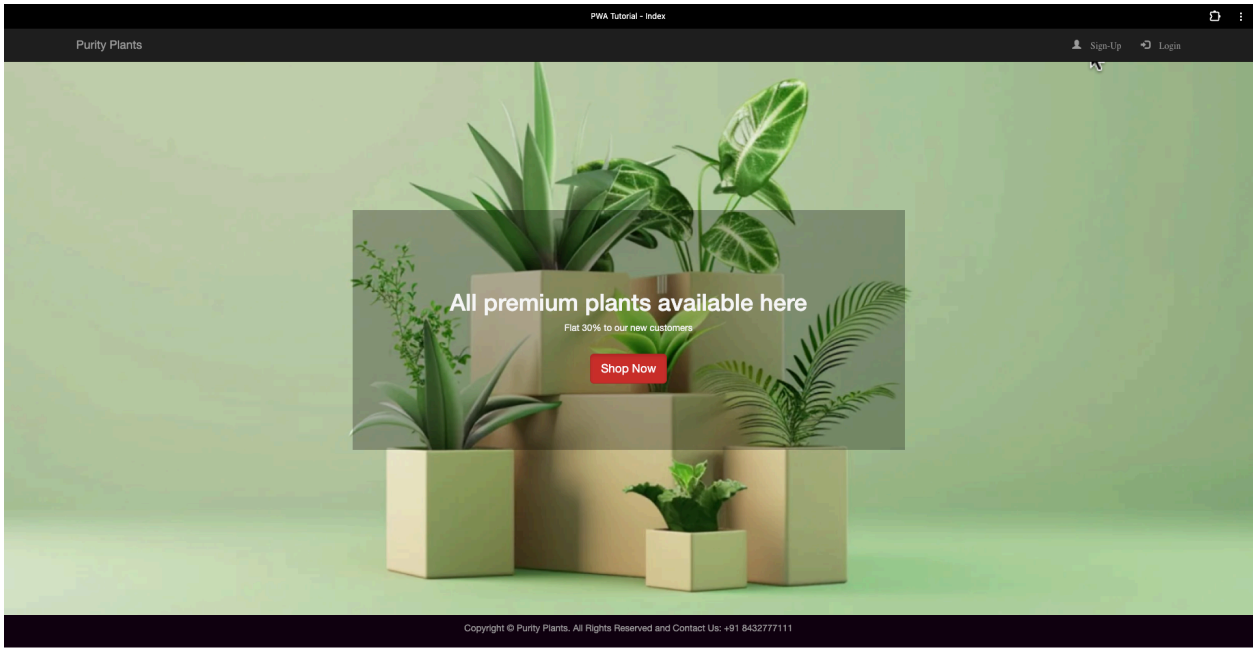
Now go to developer options -> Application->Manifest



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




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



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

MAD & PWA Lab
Journal

Experiment No.	08
Experiment Title.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA
Roll No.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Name: Varun Khubani
Division: D15A
Roll
No:30
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.

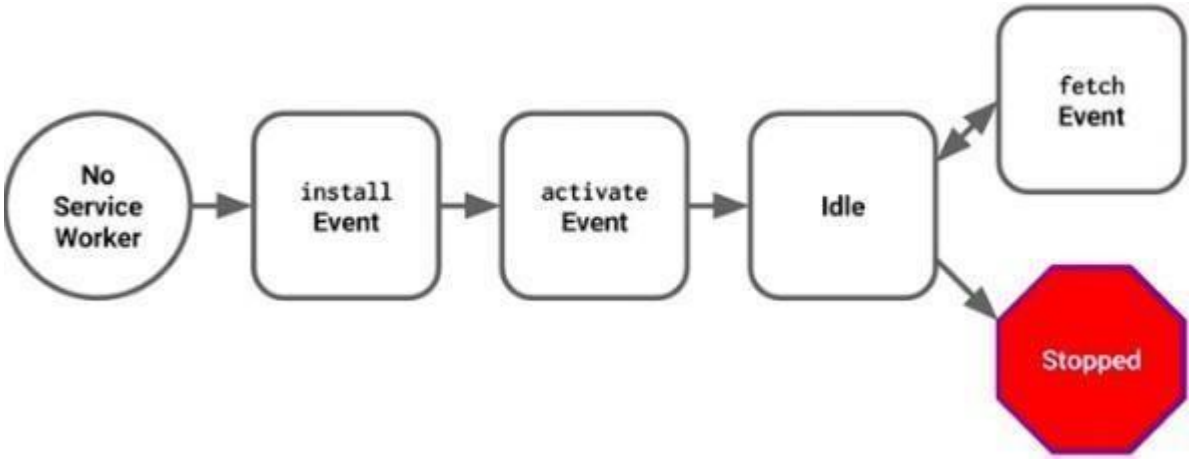
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

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

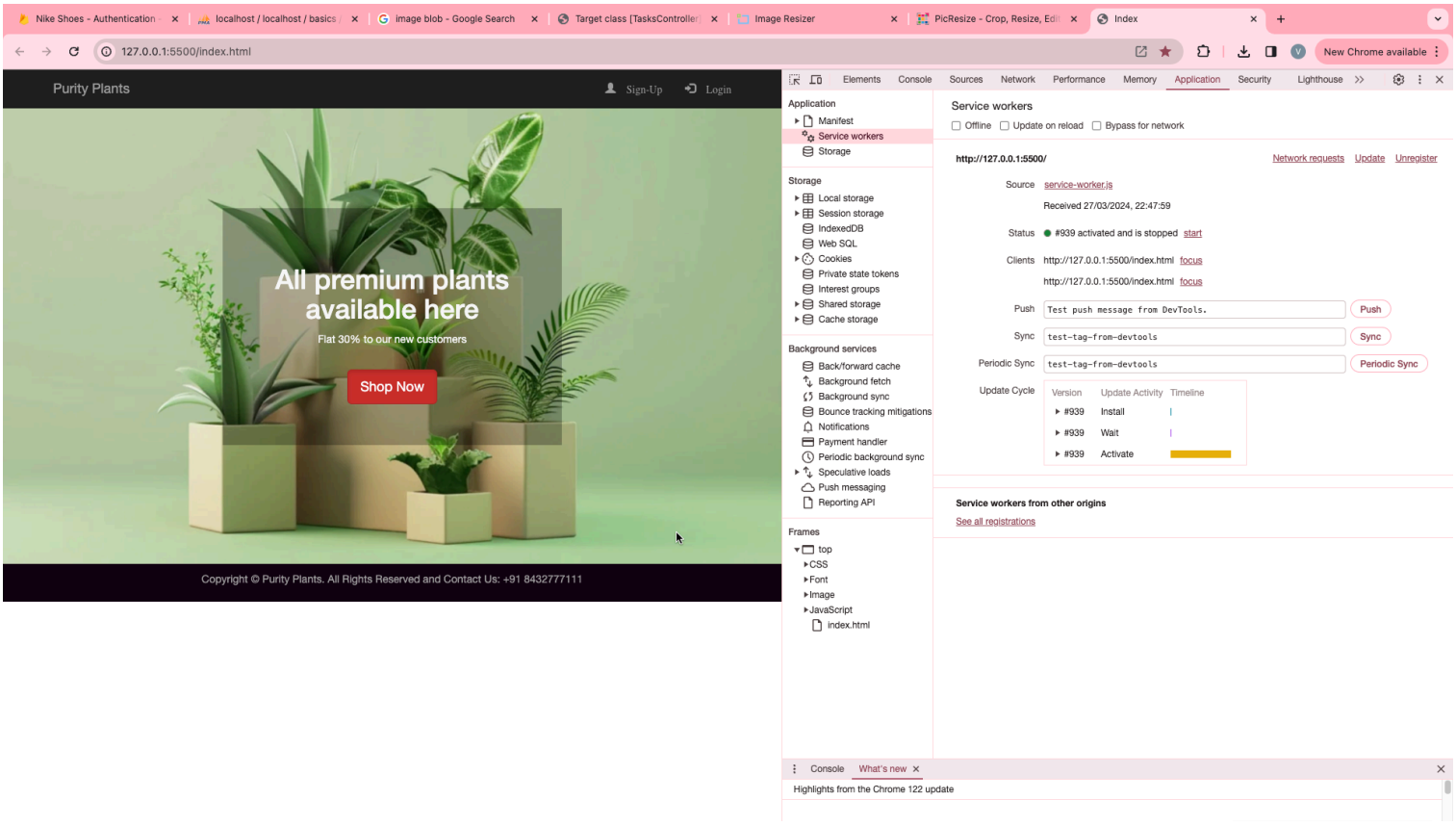
```
self.addEventListener('install', function(event) {
 event.waitUntil(
   caches.open(CACHE_NAME)
     .then(function(cache) {
       console.log('Opened cache');
       return cache.addAll(urlsToCache)
         .catch(function(error) {
           console.error('Cache.addAll error:', error);
     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

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.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Name: Varun Khubani
Division: D15A
Roll
No:30
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.

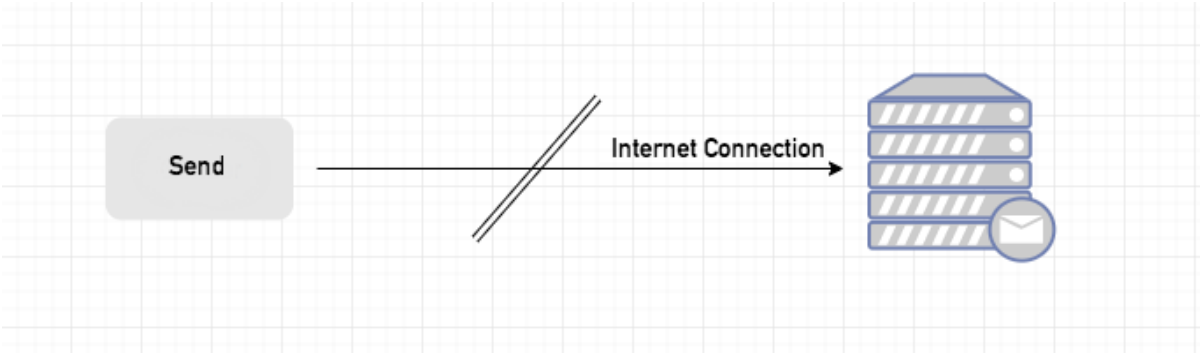
```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);

    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

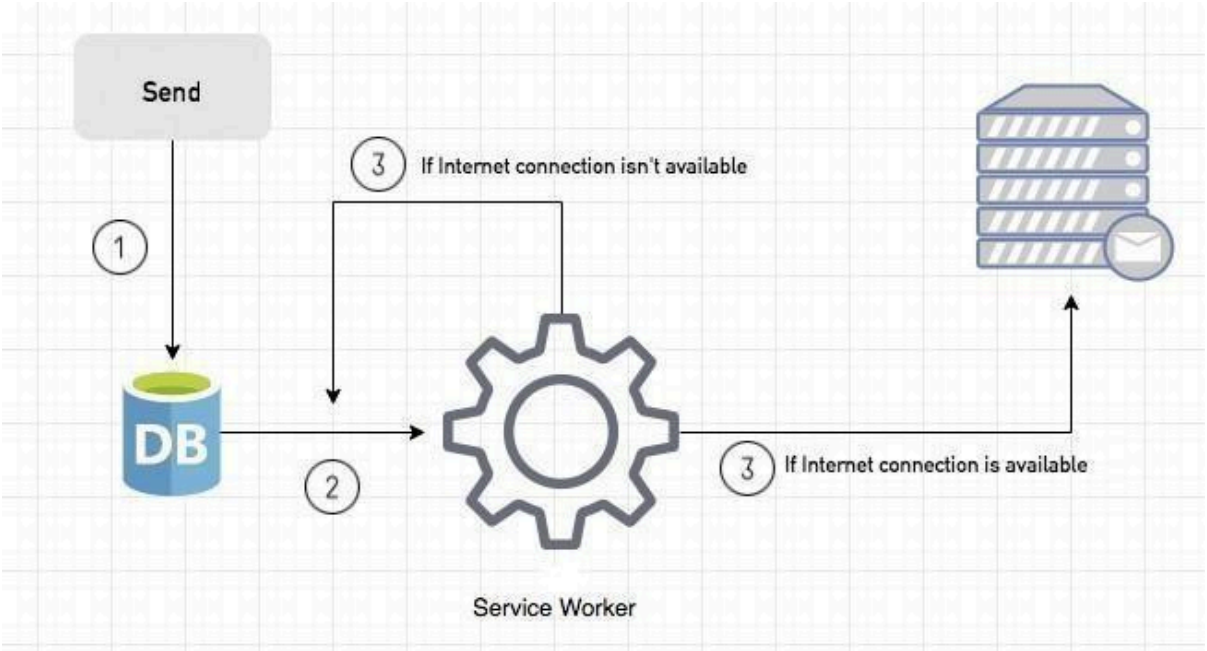
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are 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.

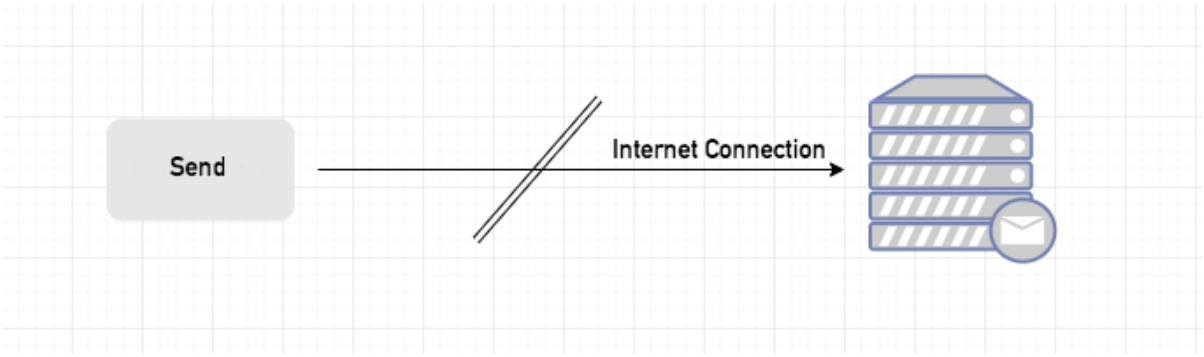
```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);
```

Sync Event

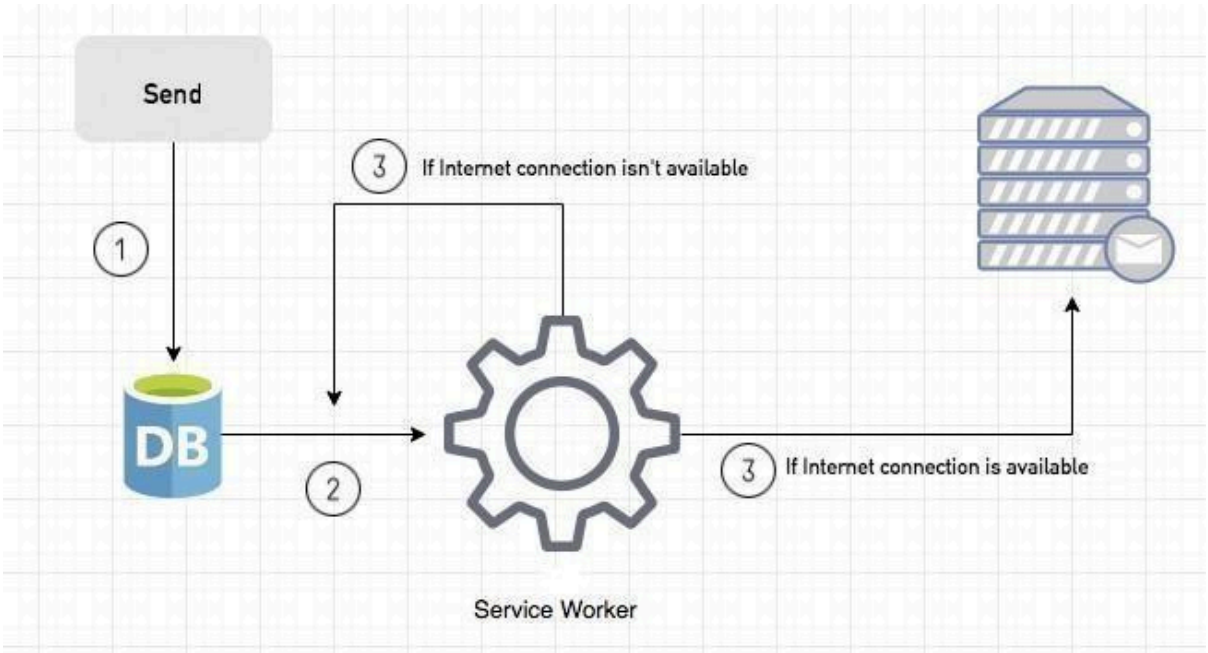
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are 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.

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.');
              }
          });
      }
```

This code sends notification permission to your Device , and click on allow to send push notification

service-worker.js

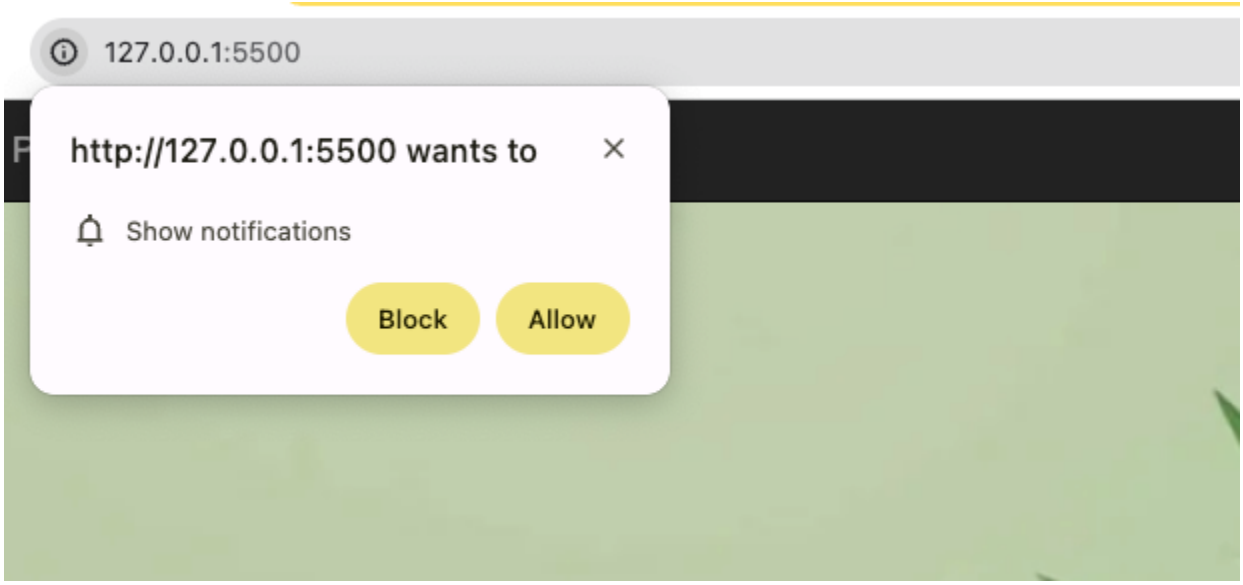
```
// service-worker.js


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);
         });
     })
```

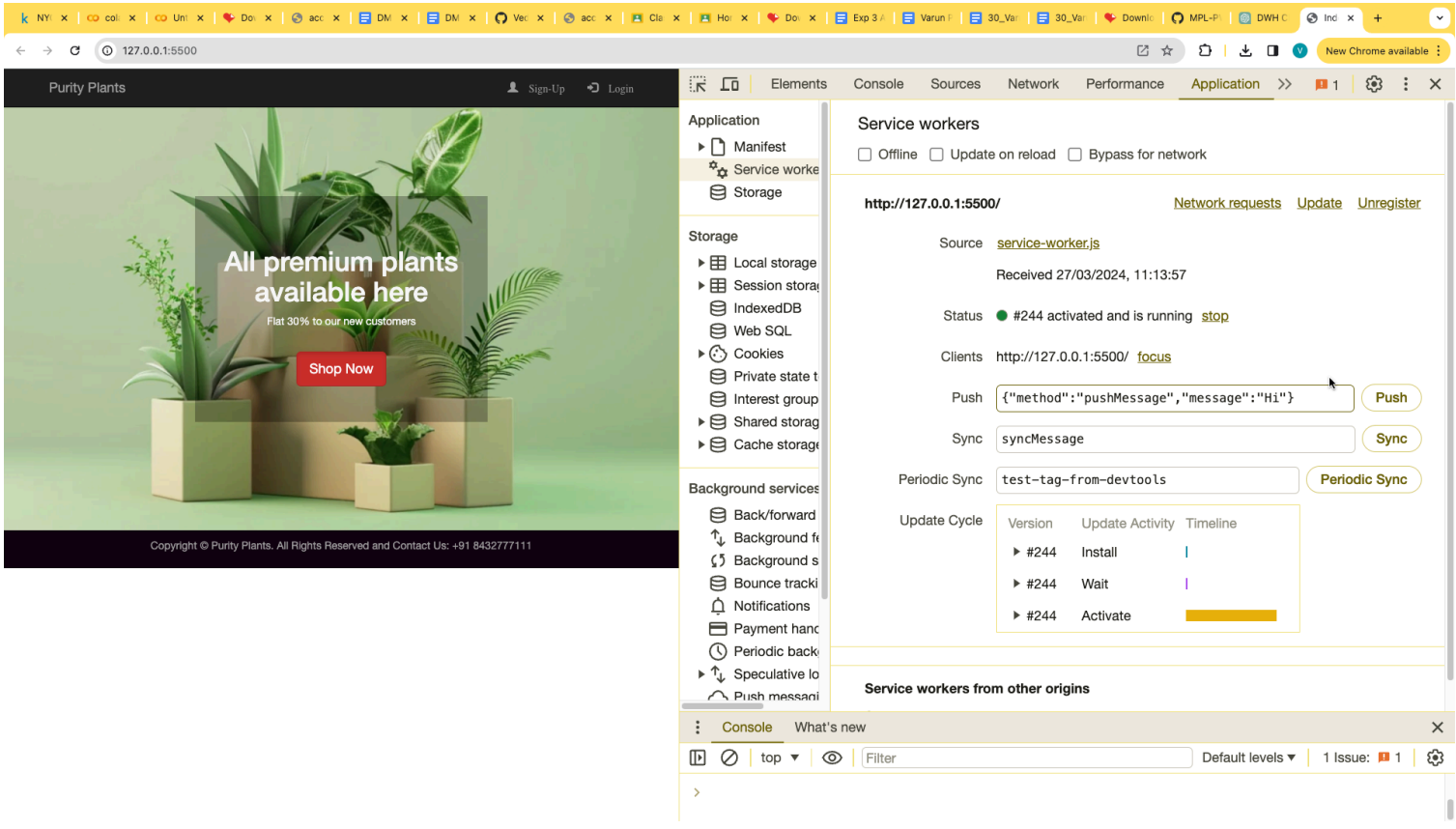
```
 );
});

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") {
```

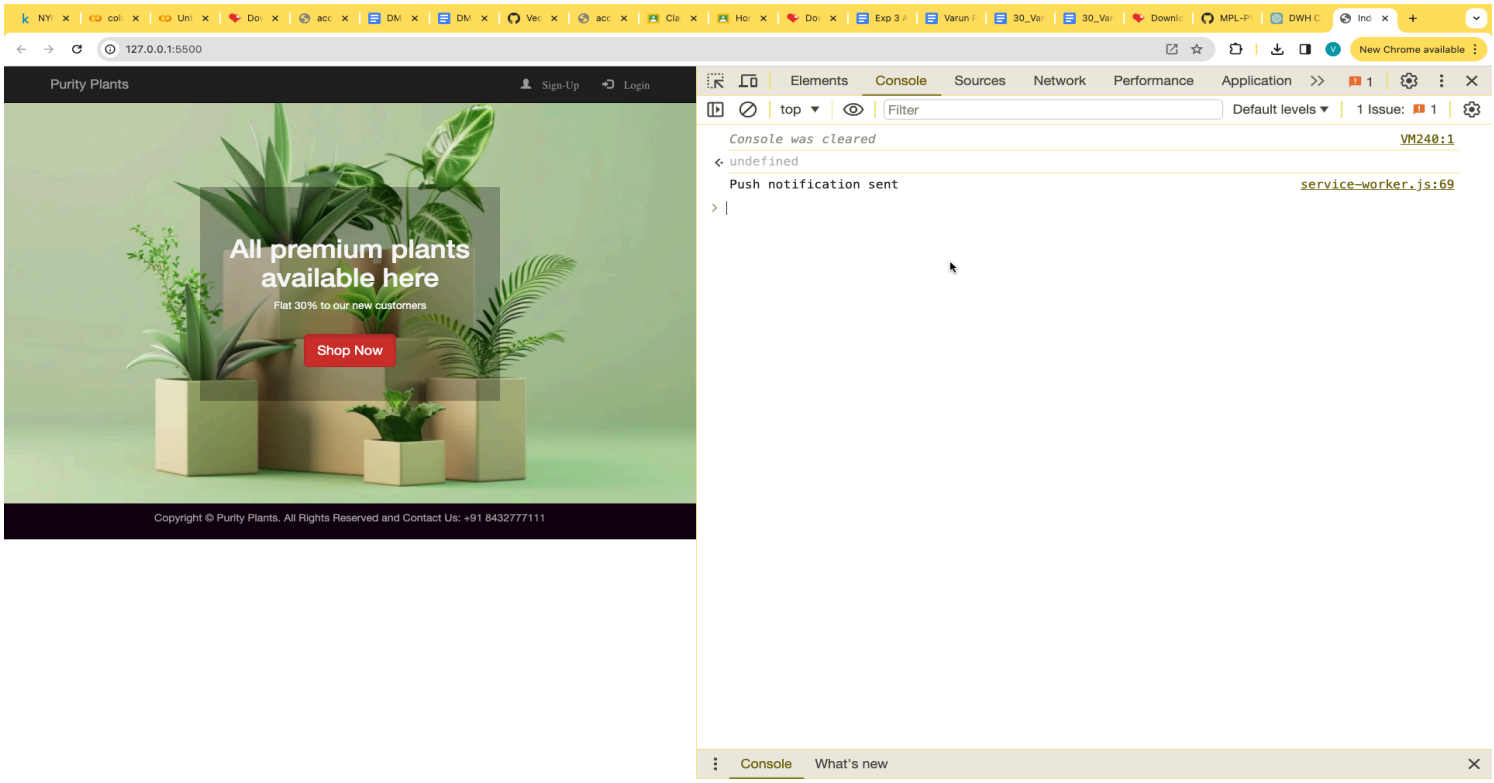
```
      console.log("Push notification sent");
      self.registration.showNotification("Ecommerce website", { body:
data.message });
    }
  } catch (error) {
    console.error("Error parsing push data:", error);
  }
 }
});

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

Output:


127.0.0.1:5500
http://127.0.0.1:5500 wants to
Show notifications
Block
Allow


127.0.0.1:5500
New Chrome available
Purity Plants
Sign-Up
Login
All premium plants available here
Flat 30% to our new customers
Shop Now
Copyright © Purity Plants. All Rights Reserved and Contact Us: +91 8432777111
Elements
Console
Sources
Network
Performance
Application
Application
Manifest
Service worke
Storage
Storage
Local storage
Session storag
IndexedDB
Web SQL
Cookies
Private state t
Interest group
Shared storag
Cache storage
Background services
Back/forward
Background f
Background s
Bounce tracki
Notifications
Payment hanc
Periodic back
Speculative lo
Push messagi
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5500/
Network requests
Update
Unregister
Source
service-worker.js
Received 27/03/2024, 11:13:57
Status
#244 activated and is running
stop
Clients
http://127.0.0.1:5500/
focus
Push
{"method":"pushMessage","message":"Hi"}
Push
Sync
syncMessage
Sync
Periodic Sync
test-tag-from-devtools
Periodic Sync
Update Cycle
Version
Update Activity
Timeline
#244
Install
#244
Wait
#244
Activate
Service workers from other origins
Console
What's new
top
Filter
Default levels
1 Issue:
1



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.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Name: Varun Khubani
Division:D15A
Roll
No:30
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

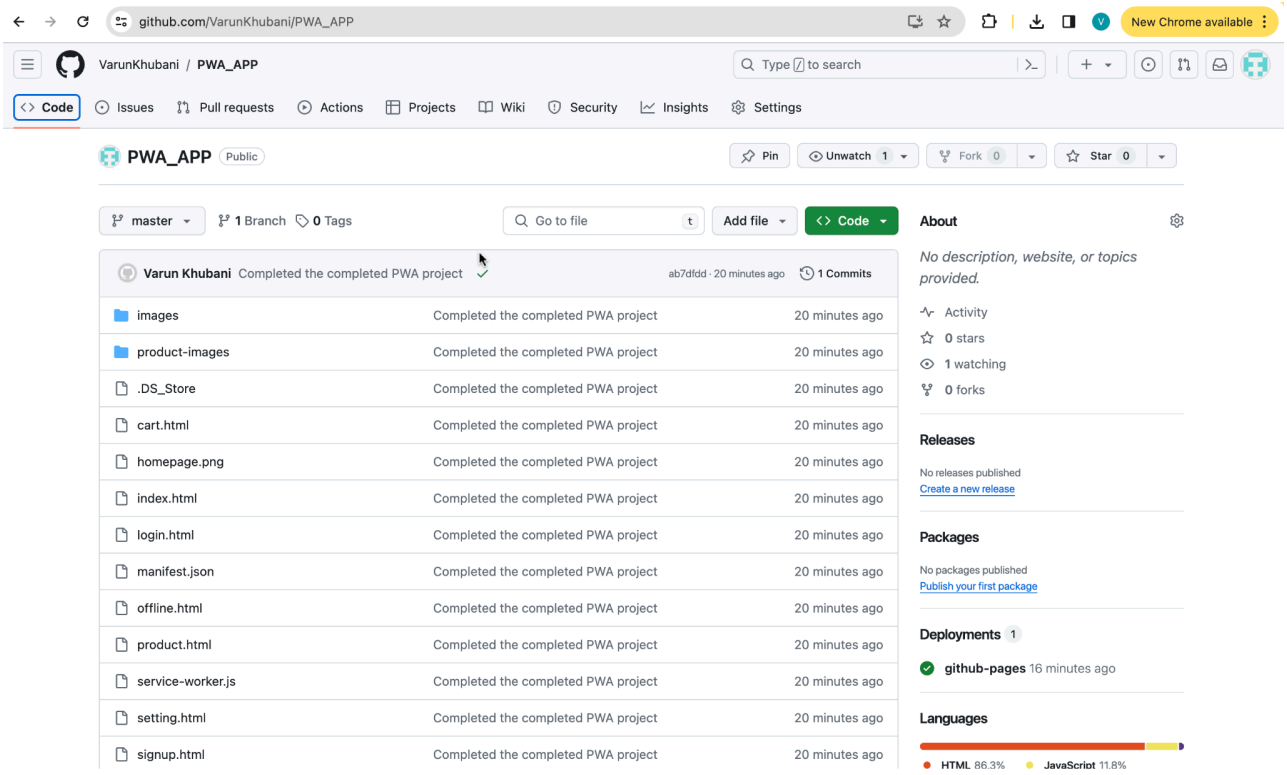
1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5.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

Cons

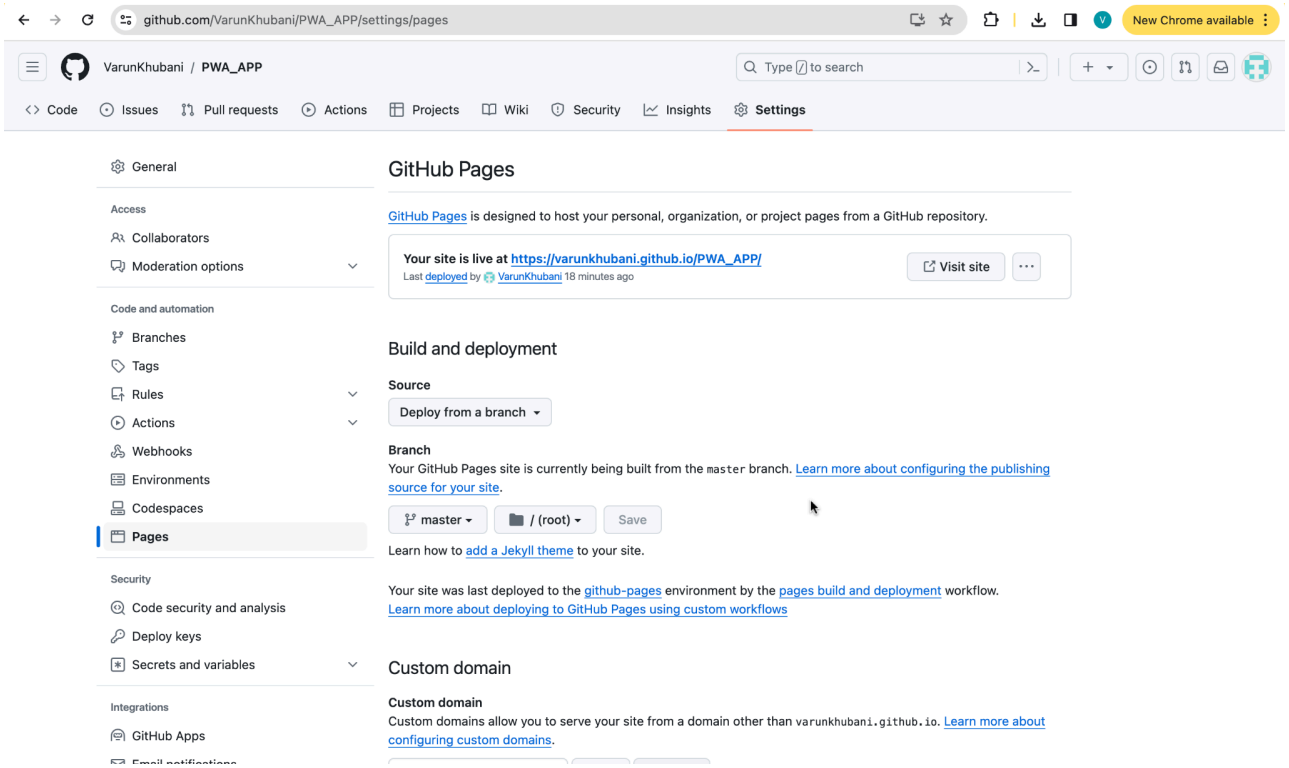
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

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

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

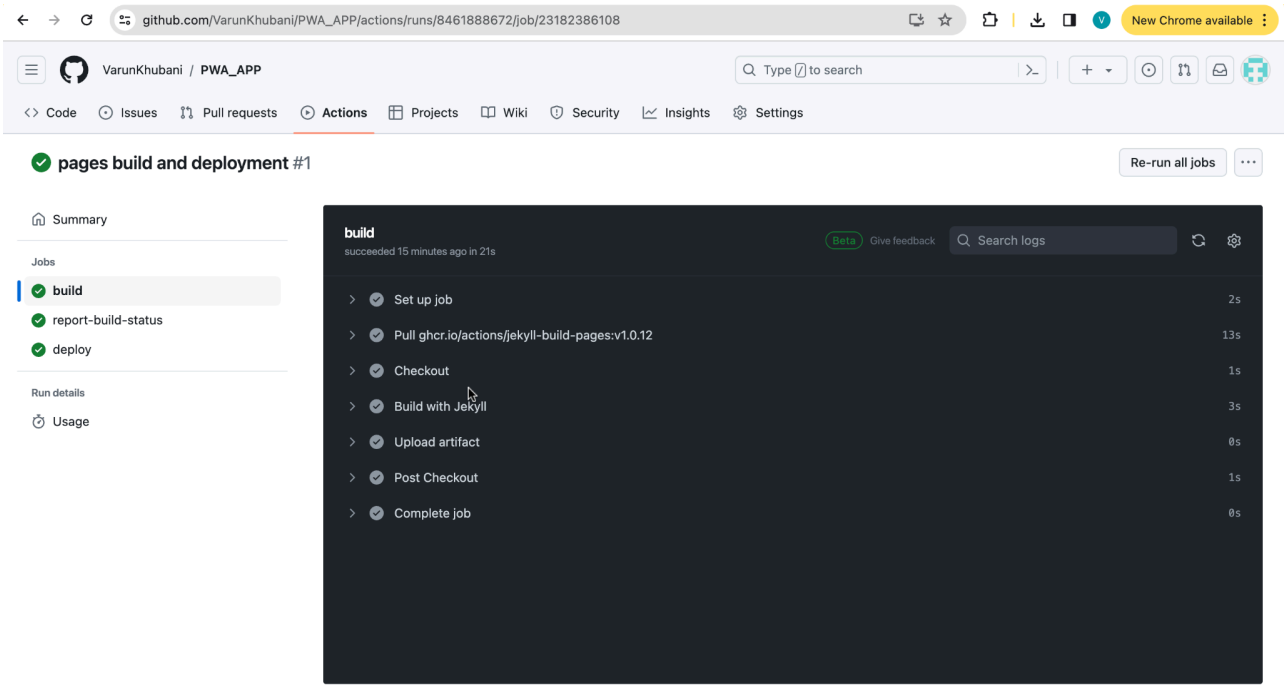


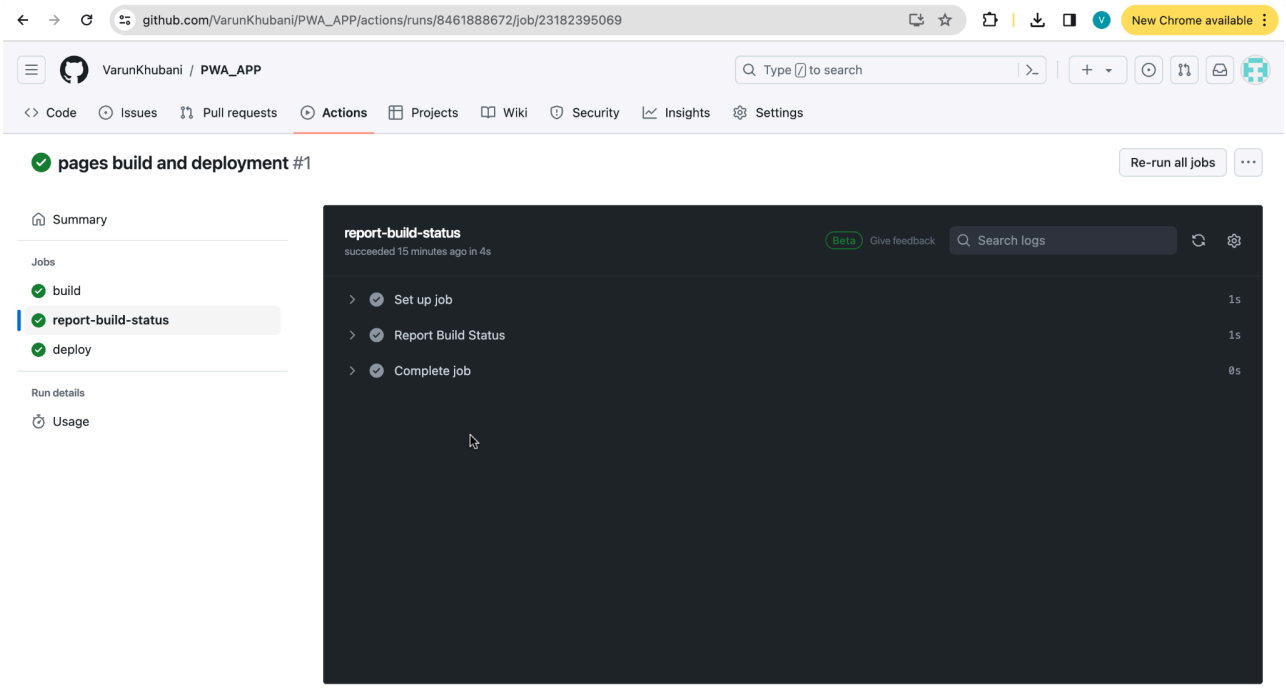
Step 2: Go to settings -> pages and choose your root directory and save it

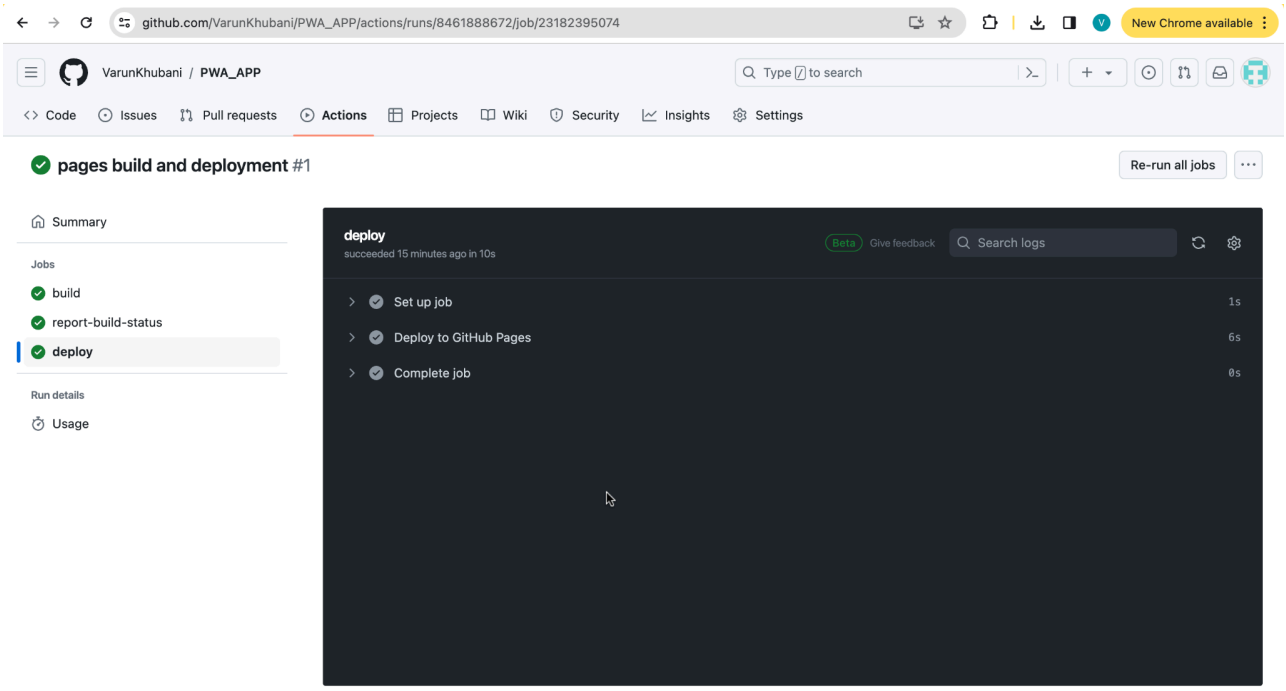


Step 3: Now go to your Code and you will see a small circle near your recent commit(Mine is finished deploying so i am getting a tick-mark sign)

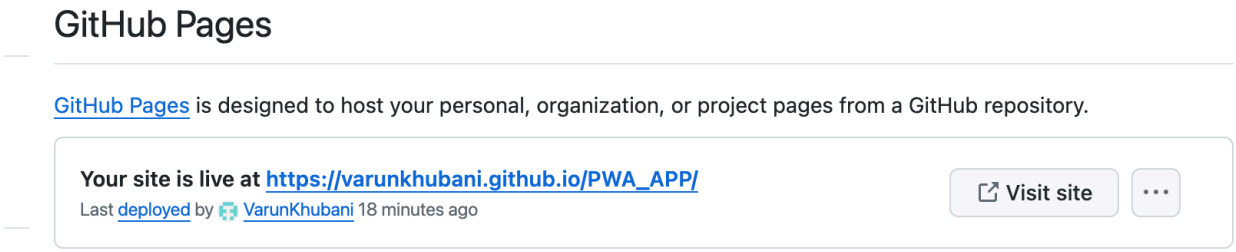
On clicking Logs of all the deployment is shown for convenience

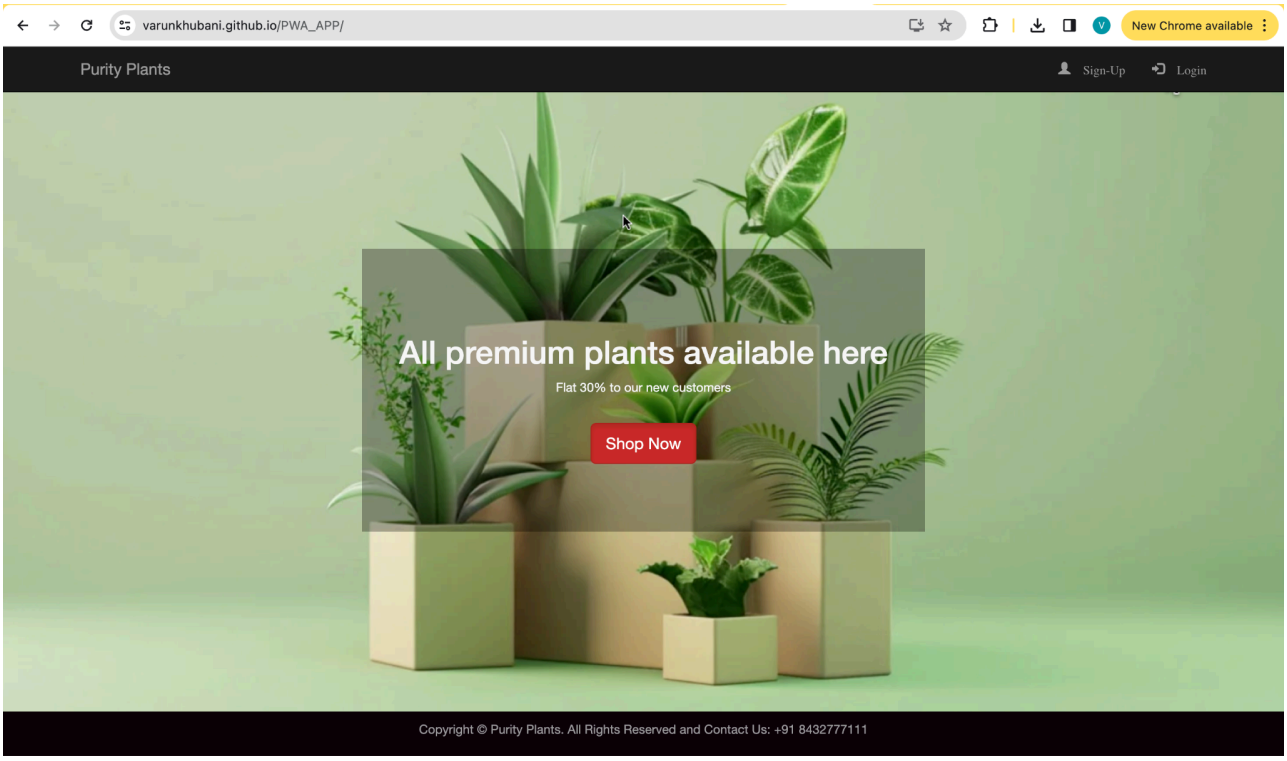

github.com/VarunKhubani/PWA_APP/actions/runs/8461888672/job/23182386108
New Chrome available
VarunKhubani / PWA_APP
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
pages build and deployment #1
Re-run all jobs
Summary
Jobs
build
report-build-status
deploy
Run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


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



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





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

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

Name: Varun Khubani
Division: D15A
Roll
No:30
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

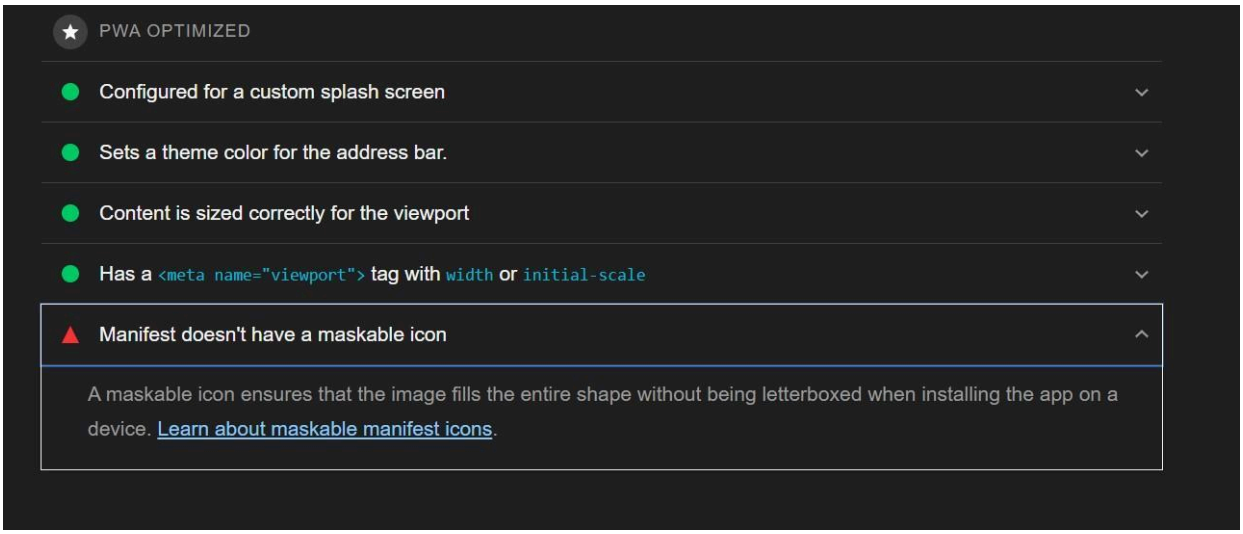
site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th percentile and so on.

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:

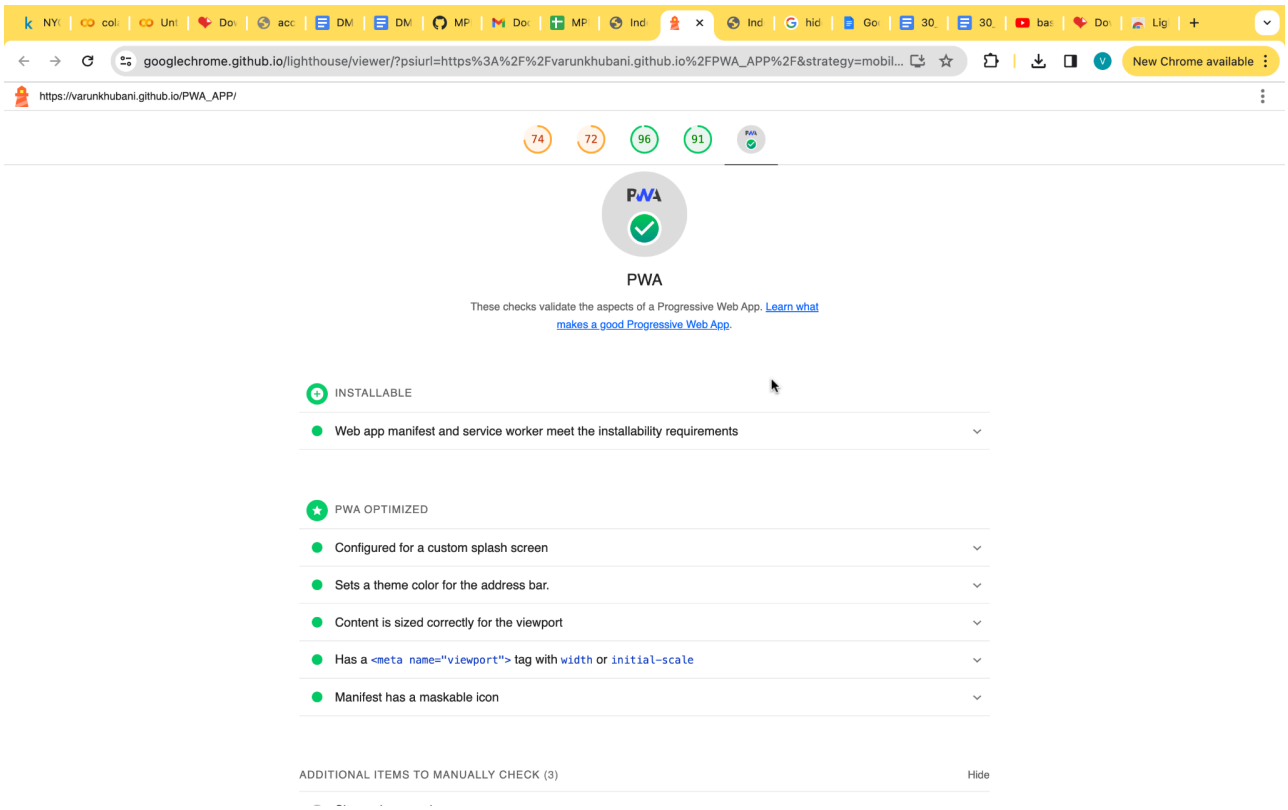
Before



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.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Name:- Varun Khubani
Div:- D15A
Roll:- 30
MAD PWD Lab Assignment-1

Q1) Flutter Overview

Flutter is an open source UI software development toolkit created by Google for building natively compiled application for mobile, web & desktop for single codebase

Key Features of Flutter

A) Single codebase:- Flutter offers a single codebase that can be used to create applications for multiple platforms

B) HOT Reload:- Developers can see the changes made in the code instantly reflected in the app

C) High Performance:- Flutter uses the SKIA graphics engine to render visuals contributing to smooth animation and a responsive render experience.

D) Access to native Features: Flutter provides seamless access to native features and API's following developers to integrate device functionalities without compromising on performance.

Q2) Widget Tree & Composition

Widget Tree is a hierarchical structure UI elements represented by widgets, widgets are the building blocks of flutter applic and the widget tree organize them in a parent child relationship widget column involves combing

Commonly widgets include.
A) Container
B) Stack
C) List view
D) Text
E) Image

Q3) Stack Management in Flutter

State Management in Flutter is crucial to handle changes in application data UI, Proper state Management ensures that the UI stays in sync

while underlying data contributes to a scalable & maintainable codebase

SetState :- used for small to medium size application where state is localized and doesn't need to be shared between multiple widgets.

Provider :- well suited for medium to large size applications.

Riverpod :- Suitable for projects where you want to take advantage for advanced features

Q4) Firebase integration in Flutter

Integration Process

Step 1 :- Create a Firebase Project in Firebase website

Step 2 :- Add Firebase dependencies & flutter fire packages

Step 3 :- Initialize Firebase in your flutter app by calling Firebase.initializeApp in main function.

Step 4 :- Use Firebase services based on your requirement like Firebase Database, Authentication, storage etc.

Benefits of Firebase

A) Real Time NO SQL Database
B) Authentication
C) Cloud storage
D) Hashing.

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.	30
Name	Varun Khubani
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	

Varun Khubani
Roll: 30
Camlin Page

PWA - Assignment - 2 Div: D15A

Q1) Define Progressive web App (PWA) & explain its significane in Modern web development

→ A progressive web App (PWA) is atype of web applications that uses modern web capabilities to deliver an app like-experience to users PWAs are built using standard web technologies. Key characteristics of PWA are:-

1) Progressive enhancement - They are built using progressive enhancement principle meaning they provide basic experience for all users.
2) Responsive design - PWA's are designed to feel & behave like native mobile apps. They include multiple features such as animations
3) Reliability - PWA's are designed to reliable even when users are offline or has slow or variable internet connection.
4) Fast performance
5) Discoverability
6) security
7) cross-platform compatibility.

The significance of PWA's in modern development lies in their ability to bridge the gap b/w web & native app experience. By combining the reach & accesibility of the web and native functionality & engagement.

Control 3) Good control 3) less control over
over overall element appearance
layout on extreme sizes

Q3) Describe the lifecycle of services workers including registration, installation and activation phase

→ Service workers are crucial components of progressive web Appl (PWAs) that enable features such as offline support, push notifications & background

1). Registration - 1st phase of lifecycle occurs in the main Javascript file of web applications.

2) Registration is done using 'navigator service. worker' involves serial key registration & installation.

Installation - Once servile worker registered browser downloads and starts installation process of service During installation the service worker script is executed and the install event is fired.
Installation process is critical for setting up the service workers initial cache & preparing it to take control & preparing it to take.
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
- Developers can use the 'activate' event handler to manage cache cleanup restoring & others activation relate tasks.
Servile remains active untill it is explicitly unregistered or updated
Its important to note that service workers run seperately from main web page & operate it on the different level.

Q.2) Responsive web design approach that ensure a website adopts it layout & context to seamlessly function and display well across various screen size from desktops to smartphone and other devices too

Importance for PWA:-

- Foundation for APP like Experience PWP ensures proper layout & navigation across devices, mimicking the adaptability of native apps.

Accessibility:- A responsive design make PWA accessible to wider audience using diverse devices

Feature	Responsive	fluid	adaptive
layout	flexible adopts to any size of the screen	continuously adjust based on relative units less efforts for basic layout	uses multiple fixed with layout

Development effort	moderate	less effort any very basic layout	more effort to create multiple layout

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

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