**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 **Abhishek Fatate** 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](mailto:kajal.jewani@ves.ac.in)

**Programme Outcomes**: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

**Program specific Outcomes**

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

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

Lab Objectives:

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

**Lab Outcomes:**

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

**Index**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr.**  **No** | **Experiment Title** | **LO** | **DOP** | **DOS** | **Grade** |
| **1.** | To install and configure the Flutter Environment | LO1 | 16/1 | 23/1 | 15 |
| **2.** | To design Flutter UI by including common widgets. | LO2 | 23/1 | 30/1 | 15 |
| **3.** | To include icons, images, fonts in Flutter app | LO2 | 30/1 | 6/2 | 15 |
| **4.** | To create an interactive Form using form widget | LO2 | 6/2 | 13/2 | 14 |
| **5.** | To apply navigation, routing and gestures in Flutter App | LO2 | 13/2 | 20/2 | 14 |
| **6.** | To Connect Flutter UI with fireBase database | LO3 | 20/2 | 5/3 | 13 |
| **7.** | To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”. | LO4 | 5/3 | 12/3 | 12 |
| **8.** | To code and register a service worker, and complete the  install and activation process for a new service worker for the E-commerce PWA | LO5 | 12/3 | 19/3 | 12 |
| **9.** | To implement Service worker events like fetch, sync and  push for E-commerce PWA | LO5 | 19/3 | 26/3 | 12 |
| **10.** | To study and implement deployment of Ecommerce PWA  to GitHub Pages. | LO5 | 26/3 | 2/4 | 13 |
| **11.** | To use google Lighthouse PWA Analysis Tool to test the PWA functioning. | LO6 | 5/3 | 12/3 | 13 |
| **12.** | Assignment-1 | LO1,LO2  ,LO3 | 2/2 | 5/2 | 5 |
| **13.** | Assignment-2 | LO4,LO5  ,LO6 | 19/3 | 21/3 | 4 |

# **MAD & PWA Lab Journal**

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

**EXP 1**

**Installation and Configuration of Flutter Environment.**

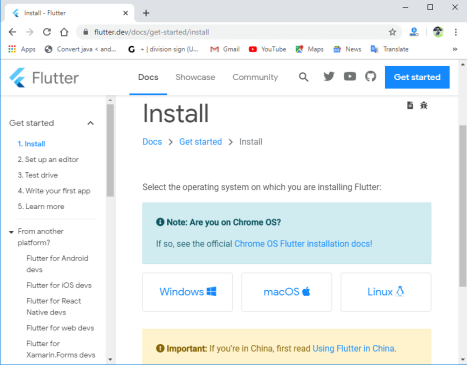
**Abhishek Fatate**

**Roll No. 15**

**Batch A**

**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.

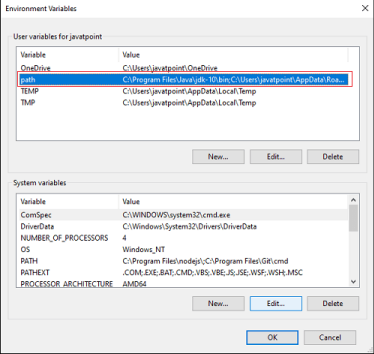
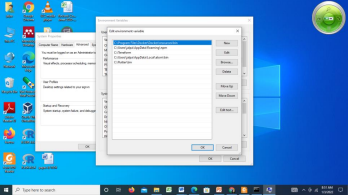


**Step 2:** Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will find the download link for SDK.

**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.

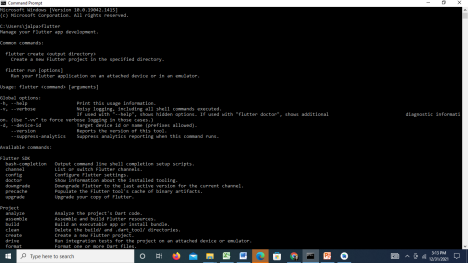
**Step 4:** To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:

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

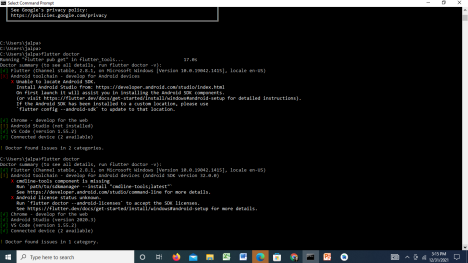
**Step 4.2:** Now, select path -> click on edit. The following screen appears

**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.

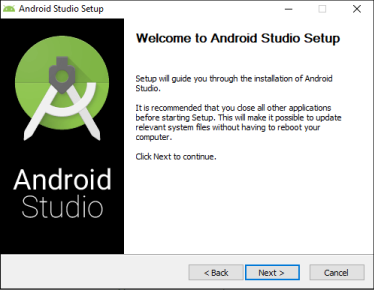


**Step 6:** When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

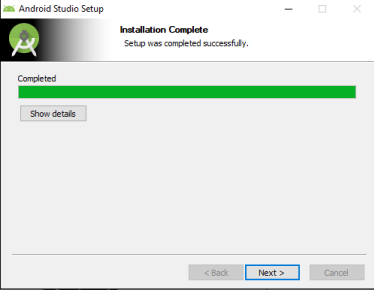
**Step 7:** Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.

**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.

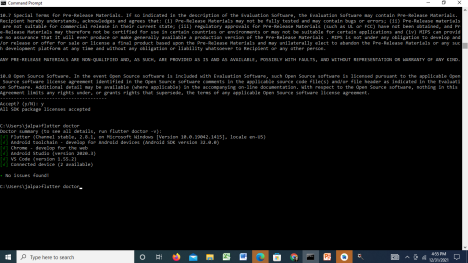


**Step 7.3:** Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



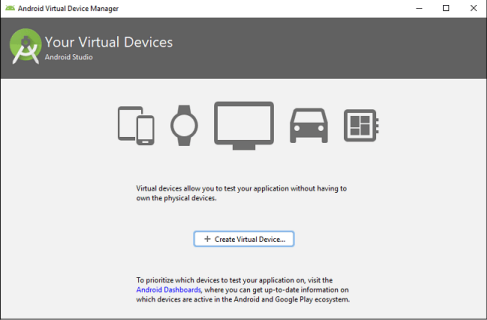
**Step 7.4:** In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to choose the 'Don't import Settings option’ and click OK. It will start the Android Studio.

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



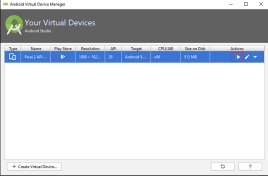
**Step 8:** Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.

**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.



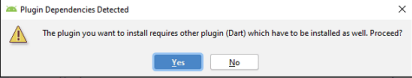
**Step 8.5:** Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.



**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('Hello Abhishek'),

),

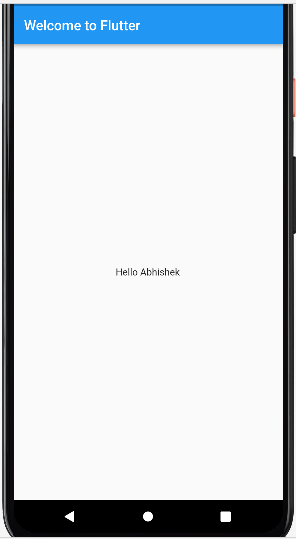
),

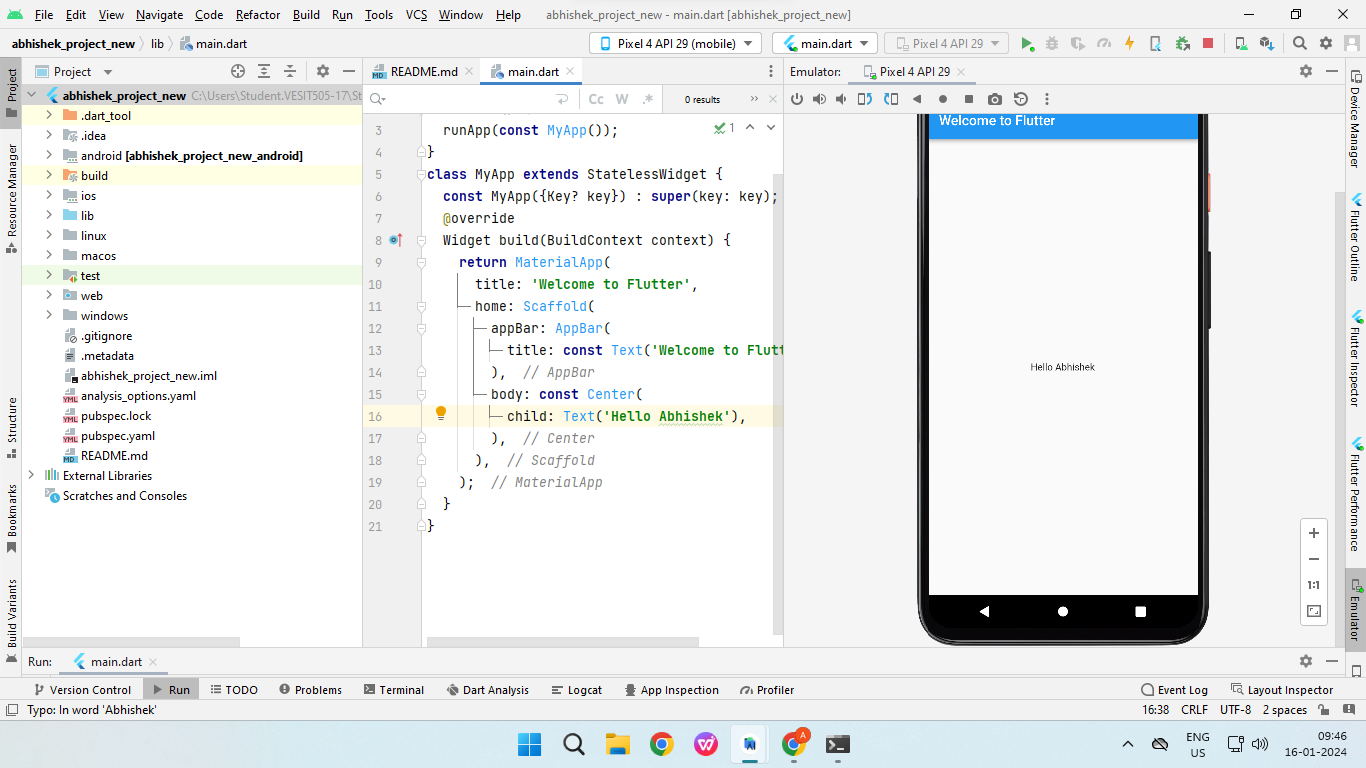
);

}

}

Output:-





# **MAD & PWA Lab Journal**

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

### **Name: Abhishek Fatate roll no.15 MAD PWA EXP 2**

**Aim:** To design Flutter UI by including common widgets.

# Theory:

We can split the Flutter widget into two categories:

1. Visible (Output and Input)
2. Invisible (Layout and Control)

1. Visible widget

The visible widgets are related to the user input and output data. Some of the important types of

this widget are:

## Text

A Text widget holds some text to display on the screen. We can align the text widget by using

textAlign property, and style property allow the customization of Text that includes font, font

weight, font style, letter spacing, color, and many more.

## Button

This widget allows you to perform some action on click. Flutter does not allow you to use the

Button widget directly; instead, it uses a type of buttons like a FlatButton and a RaisedButton.

## Image

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

which

are given below:

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

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

o file: It loads images from the system folder.

o memory: It load image from memory.

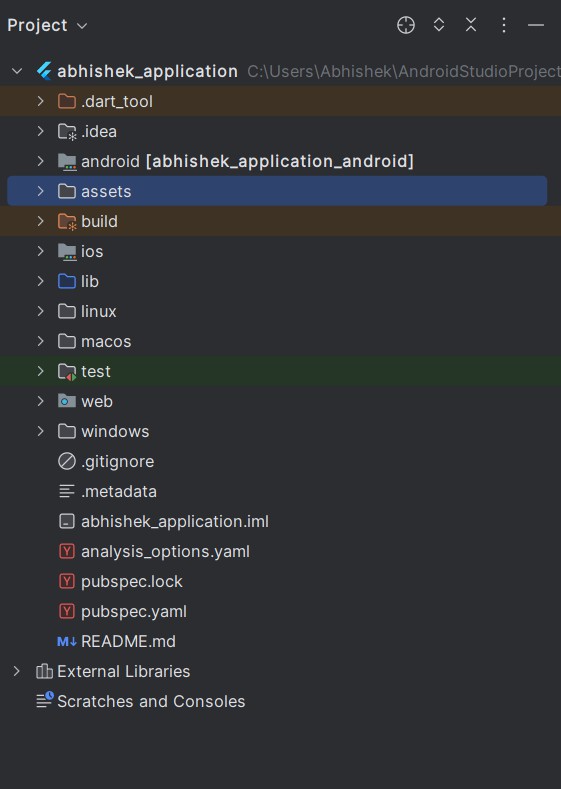
o network: It loads images from the network.

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

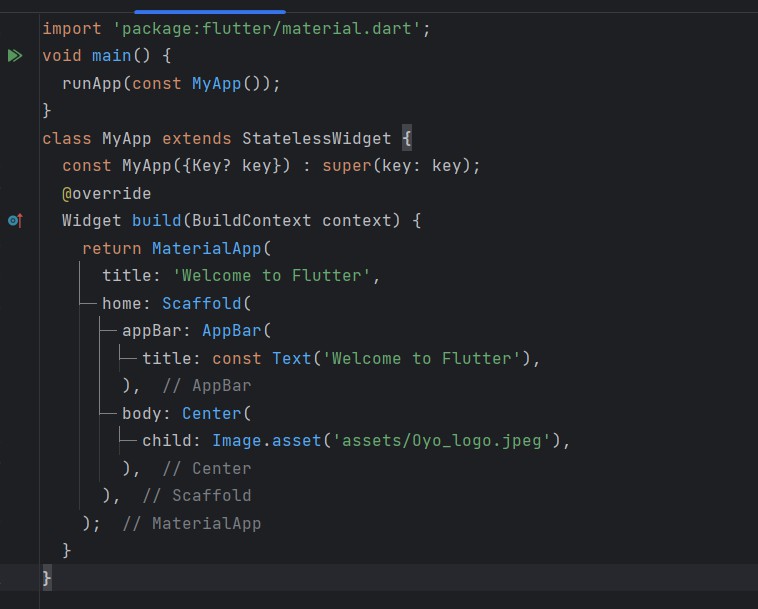
assets:

- assets/comp.jpg

## File structure:



**Code:**



# Output:



### Conclusion:

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

# **MAD & PWA Lab Journal**

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

EXP-3

Name- Abhishek Fatate

Div: D15A Roll no.- 15

AIM-: To include icons, images, fonts in Flutter app

THEORY: 2This widget holds the image which can fetch it from multiple sources

like from the asset folder or directly from the URL. It provides many constructors

for loading image, which are given below:

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

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

o file: It loads images from the system folder.

o memory: It load image from memory.

o network: It loads images from the network.

To add an image in the project, you need first to create an assets folder where you

keep your images and then add the below line in pubspec.yaml file.

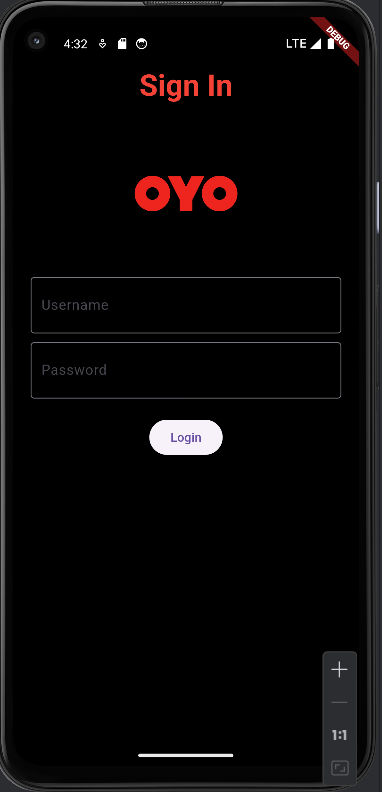
Assets:

-assets/comp.jpeg

Code:

import 'package:flutter/material.dart';  
import 'homepage.dart'; // Import the HomePage widget  
  
class SignInScreen extends StatelessWidget {  
 const SignInScreen({Key? key}) : super(key: key);  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text('Sign In'),  
 centerTitle: true,  
 ),  
 body: Padding(  
 padding: const EdgeInsets.all(20.0),  
 child: Column(  
 crossAxisAlignment: CrossAxisAlignment.center,  
 children: [  
 Image.asset(  
 'assets/Oyo\_logo.jpeg',  
 width: 150,  
 height: 150,  
 ),  
 SizedBox(height: 20),  
 TextField(  
 decoration: InputDecoration(  
 labelText: 'Username',  
 border: OutlineInputBorder(),  
 ),  
 ),  
 SizedBox(height: 10),  
 TextField(  
 decoration: InputDecoration(  
 labelText: 'Password',  
 border: OutlineInputBorder(),  
 ),  
 obscureText: true, // Hides the password input  
 ),  
 SizedBox(height: 20),  
 ElevatedButton(  
 onPressed: () {  
 // Navigate to the HomePage  
 Navigator.*pushReplacement*(  
 context,  
 MaterialPageRoute(builder: (context) => HomePage()),  
 );  
 },  
 child: Text('Login'),  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
}

Output:



# **MAD & PWA Lab Journal**

|  |  |
| --- | --- |
| Experiment No. | 04 |
| Experiment Title. | To create an interactive Form using form widget |
| Roll No. | 15 |
| Name | Abhishek Fatate |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation |
| Grade: | 14 |

MAD LAB EXP 4

Name- Abhishek Fatate

Div-D15A Roll-15

AIM : To create an interactive Form using a form widget

Code:

import 'package:flutter/material.dart';

import 'homepage.dart'; // Import the HomePage widget

class SignInScreen extends StatelessWidget {

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

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: const Text('Sign In'),

centerTitle: true,

),

body: Padding(

padding: const EdgeInsets.all(20.0),

child: Form(

child: Column(

crossAxisAlignment: CrossAxisAlignment.center,

children: [

Image.asset(

'assets/Oyo\_logo.jpeg',

width: 150,

height: 150,

),

SizedBox(height: 20),

TextFormField(

decoration: InputDecoration(

labelText: 'Username',

border: OutlineInputBorder(),

),

),

SizedBox(height: 10),

TextFormField(

decoration: InputDecoration(

labelText: 'Password',

border: OutlineInputBorder(),

),

obscureText: true, // Hides the password input

),

SizedBox(height: 20),

ElevatedButton(

onPressed: () {

// Implement form submission logic here

// For demonstration purposes, navigate to HomePage

Navigator.pushReplacement(

context,

MaterialPageRoute(builder: (context) => HomePage()),

);

},

child: Text('Login'),

),

],

),

),

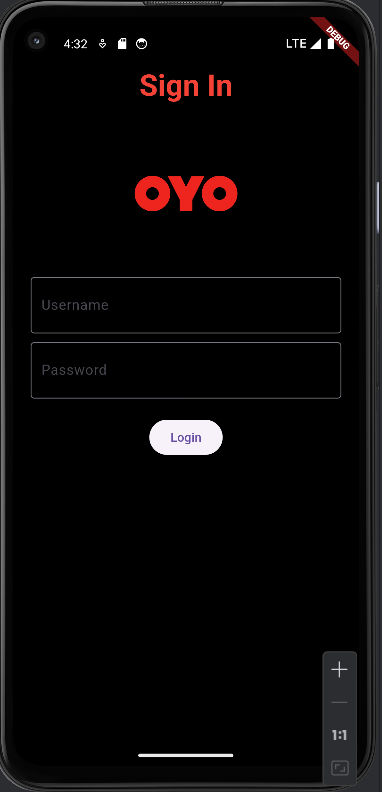
),

);

}

}

Output:



# **MAD & PWA Lab Journal**

|  |  |
| --- | --- |
| Experiment No. | 05 |
| Experiment Title. | To apply navigation, routing and gestures in Flutter App |
| Roll No. | 15 |
| Name | Abhishek Fatate |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation |
| Grade: | 14 |

MAD Lab Exp 5

Name: Abhishek Fatate Roll No. 15

Div: D15A Batch: A

Aim:

The aim of this experiment is to create an interactive form using Flutter's form widget. The form will include various form fields such as text fields, dropdown menus, and checkboxes, along

with validation logic. Additionally, a button will be added to validate and submit the form data.

Theory:

Gesture:

In Flutter, gestures are user interactions with the application, such as taps, drags, and scrolls. The GestureDetector widget is used to detect various gestures and provides callbacks to handle these interactions. Here's a brief overview:

GestureDetector Widget: The GestureDetector widget wraps its child and detects various gestures applied to it. It provides properties like onTap, onDoubleTap, onLongPress, onPanUpdate, etc., to handle different types of gestures.

Gesture Detection Process: When a user interacts with the screen, the GestureDetector widget detects the gesture based on the user's input and invokes the appropriate callback function,

allowing developers to respond to the gesture accordingly. Navigation:

Navigation in Flutter refers to moving between different screens or routes within the application. Flutter's navigation system is built around the Navigator class, which manages a stack of routes. Here's how it works:

Routes: In Flutter, each screen or page is called a route. Routes are pushed onto and popped off the Navigator's stack to navigate between screens.

Navigator.push(): To navigate from one route to another, you use the Navigator.push() method. This method adds a new route to the stack, displaying the new screen on top of the current one.

Navigator.pop(): To return to the previous route, you use the Navigator.pop() method. This removes the top route from the stack, returning to the previous screen.

Steps to Implement Navigation and Gestures:

Create Two Routes: Define two separate screens or widgets to represent the two routes in your application.

Navigate to Second Route: Use Navigator.push() to navigate from the first route to the second route when a specific gesture or action is detected.

Return to First Route: Implement a mechanism, such as a button press or gesture detection, to trigger Navigator.pop() and return from the second route to the first route.

By combining gestures and navigation, you can create dynamic and interactive Flutter

applications that provide a seamless user experience for navigating between different screens and responding to user input through gestures.

# Code in main.dart:

import 'package:abhishek\_application/firebase\_options.dart';

import 'package:firebase\_core/firebase\_core.dart';

import 'package:flutter/material.dart';

import 'LoginScreen.dart';

import 'splash\_screen.dart';

void main() async {

WidgetsFlutterBinding.ensureInitialized();

await Firebase.initializeApp(

options: DefaultFirebaseOptions.currentPlatform,

);

runApp(const MyApp());

}

class MyApp extends StatelessWidget {

const MyApp({Key? key}) : super(key: key);

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Oyo',

theme: ThemeData(

scaffoldBackgroundColor: Colors.black,

appBarTheme: AppBarTheme(

backgroundColor: Colors.black,

titleTextStyle: const TextStyle(

color: Colors.red,

fontSize: 34.0,

fontWeight: FontWeight.bold,

),

),

),

home: const SplashScreen(), // Display SplashScreen initially

);

}

}

# Code in Home\_screen.dart

import 'package:flutter/material.dart';

import 'HotelDetailsPage.dart';

import 'SearchPage.dart'; // Import the SearchPage.dart file

import 'BookingsPage.dart'; // Import the BookingsPage.dart

import 'NeedHelpPage.dart'; // Import the NeedHelpPage.dart

class HomePage extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

body: CustomScrollView(

slivers: <Widget>[

SliverAppBar(

title: Text('Home'),

centerTitle: true,

floating: true,

snap: true,

elevation: 0,

flexibleSpace: LayoutBuilder(

builder: (BuildContext context, BoxConstraints constraints) {

return constraints.biggest.height > 150

? Image.asset(

'assets/Oyo\_logo.jpeg',

fit: BoxFit.cover,

)

: SizedBox.shrink();

},

),

),

SliverToBoxAdapter(

child: SingleChildScrollView(

scrollDirection: Axis.horizontal,

child: Row(

children: [

CityButton(

cityName: 'Mumbai',

onPressed: () {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => CityPage(cityName: 'Mumbai'),

),

);

},

),

CityButton(

cityName: 'Pune',

onPressed: () {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => CityPage(cityName: 'Pune'),

),

);

},

),

CityButton(

cityName: 'Bangalore',

onPressed: () {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) =>

CityPage(cityName: 'Bangalore'),

),

);

},

),

CityButton(

cityName: 'Delhi',

onPressed: () {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => CityPage(cityName: 'Delhi'),

),

);

},

),

CityButton(

cityName: 'Chennai',

onPressed: () {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => CityPage(cityName: 'Chennai'),

),

);

},

),

CityButton(

cityName: 'Hyderabad',

onPressed: () {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) =>

CityPage(cityName: 'Hyderabad'),

),

);

},

),

],

),

),

),

SliverList(

delegate: SliverChildBuilderDelegate(

(BuildContext context, int index) {

return HotelInfo(

hotelName: 'Hotel ${index + 1}',

distance: '${(index + 1) \* 2} km from current location',

hasRestaurant: index.isEven,

imagePath: 'assets/hotel${index % 3 + 1}.jpg',

hotelIndex: index,

);

},

childCount: 9, // Add more if needed

),

),

],

),

bottomNavigationBar: BottomNavigationBar(

selectedItemColor: Colors.black,

unselectedItemColor: Colors.black,

items: [

BottomNavigationBarItem(

icon: Icon(Icons.home),

label: 'Home',

),

BottomNavigationBarItem(

icon: Icon(Icons.search),

label: 'Search',

),

BottomNavigationBarItem(

icon: Icon(Icons.bookmark),

label: 'Bookings',

),

BottomNavigationBarItem(

icon: Icon(Icons.help),

label: 'Need Help',

),

],

onTap: (int index) {

if (index == 1) {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => SearchPage(),

),

);

} else if (index == 2) {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => BookingsPage()

),

);

} else if (index == 3) {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => NeedHelpPage(),

),

);

}

},

),

);

}

}

class CityButton extends StatelessWidget {

final String cityName;

final VoidCallback onPressed;

const CityButton({

required this.cityName,

required this.onPressed,

});

@override

Widget build(BuildContext context) {

return Padding(

padding: const EdgeInsets.all(8.0),

child: ElevatedButton(

onPressed: onPressed,

child: Text(cityName),

),

);

}

}

class HotelInfo extends StatelessWidget {

final String hotelName;

final String distance;

final bool hasRestaurant;

final String imagePath;

final int hotelIndex;

const HotelInfo({

required this.hotelName,

required this.distance,

required this.hasRestaurant,

required this.imagePath,

required this.hotelIndex,

});

@override

Widget build(BuildContext context) {

double screenWidth = MediaQuery.of(context).size.width;

return Padding(

padding: const EdgeInsets.all(8.0),

child: GestureDetector(

onTap: () {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => HotelDetailsPage(

hotelName: hotelName,

distance: distance,

hasRestaurant: hasRestaurant,

imagePath: imagePath,

hotelIndex: hotelIndex,

),

),

);

},

child: Column(

crossAxisAlignment: CrossAxisAlignment.center,

children: [

ClipRRect(

borderRadius: BorderRadius.circular(10),

child: Image.asset(

imagePath,

width: screenWidth,

height: screenWidth \* 0.75, // Adjust height as needed

fit: BoxFit.cover,

),

),

SizedBox(height: 10),

Text(

'Hotel ${hotelIndex + 1}', // Display hotel number

style: TextStyle(

fontSize: 16,

fontWeight: FontWeight.bold,

color: Colors.red,

),

),

SizedBox(height: 5),

Text(

distance,

style: TextStyle(

color: Colors.red, // Set the color to red

),

),

SizedBox(height: 5),

Text(

hasRestaurant ? 'Restaurant available' : 'No restaurant',

style: TextStyle(

color: hasRestaurant ? Colors.green : Colors.red,

fontWeight: FontWeight.bold,

),

),

],

),

),

);

}

}

// Create separate pages for each city

class CityPage extends StatelessWidget {

final String cityName;

CityPage({required this.cityName});

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text(cityName),

centerTitle: true,

),

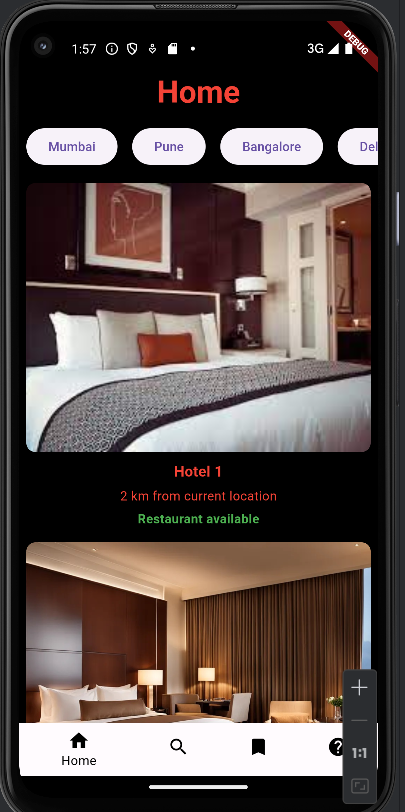
body: HomePage(),

);

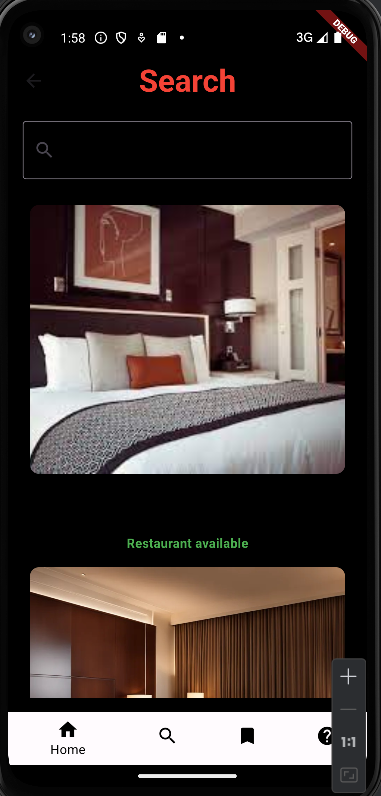
}

}}

Home screen:



Switched to search screen:



# **MAD & PWA Lab Journal**

|  |  |
| --- | --- |
| Experiment No. | 06 |
| Experiment Title. | To Connect Flutter UI with fireBase database |
| Roll No. | 15 |
| Name | Abhishek Fatate |
| 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: | 13 |

**MAD and PWA Lab**

**Experiment - 6**

**Name: Abhishek Fatate Class: D15A Roll no:15**

**Aim: To Connect Flutter UI with FireBase database**

**Theory:**

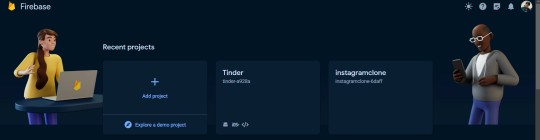
**Prerequisites**

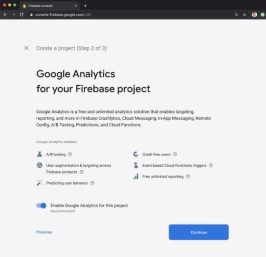
To complete this tutorial, you will need:

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

# Create a Firebase Project:

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





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

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

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

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

com.example.flutterfirebaseexample

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

android/app/build.gradle

...

defaultConfig {

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

...

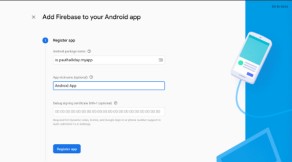
}

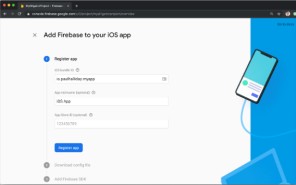
...

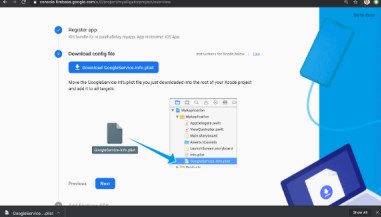
**Downloading the Conﬁg File**

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

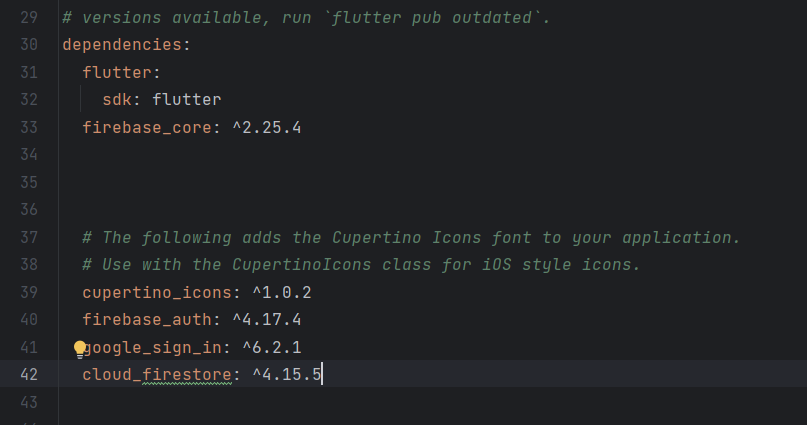
Select Download google-services.json from this page:







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

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

# Code:

Signup.dart

import 'package:flutter/material.dart';

import 'package:firebase\_auth/firebase\_auth.dart';

import 'package:google\_sign\_in/google\_sign\_in.dart';

import 'homepage.dart'; // Import the HomePage widget

class LoginScreen extends StatelessWidget {

final FirebaseAuth \_auth = FirebaseAuth.instance;

final GoogleSignIn googleSignIn = GoogleSignIn();

Future<void> \_signInWithGoogle(BuildContext context) async {

try {

final GoogleSignInAccount? googleUser = await googleSignIn.signIn();

if (googleUser != null) {

final GoogleSignInAuthentication googleAuth = await googleUser.authentication;

final AuthCredential credential = GoogleAuthProvider.credential(

accessToken: googleAuth.accessToken,

idToken: googleAuth.idToken,

);

final UserCredential userCredential = await \_auth.signInWithCredential(credential);

// Navigate to the HomePage after successful sign-in

Navigator.pushReplacement(

context,

MaterialPageRoute(builder: (context) => HomePage()),

);

}

} catch (e) {

print('Error signing in with Google: $e');

// Handle error

}

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: const Text('Sign In'),

centerTitle: true,

),

body: Padding(

padding: const EdgeInsets.all(20.0),

child: Column(

crossAxisAlignment: CrossAxisAlignment.center,

children: [

// Your login form fields here

ElevatedButton(

onPressed: () => \_signInWithGoogle(context),

child: Text('Sign In with Google'),

),

],

),

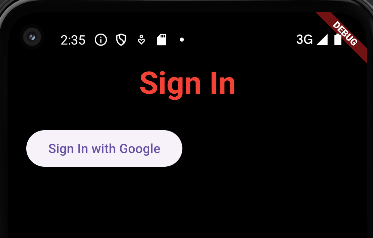
),

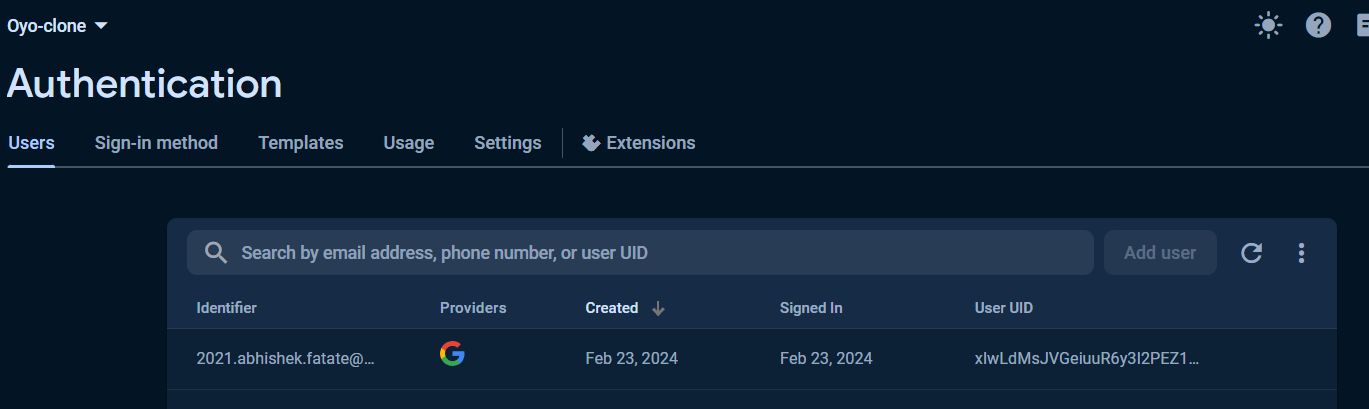
);

}

}

# Output:





**Conclusion**:

In this experiment, we have successfully connected firebase database and authenticated using google signin.

# **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. | 15 |
| Name | Abhishek Fatate |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO4: Understand various PWA frameworks and their requirements |
| Grade: | 12 |

**PWA Lab Exp 7**

Name: Abhishek Fatate Roll No. 15

Div: D15A Batch: A

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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.).

Hosted app link : [e-commerce-website-seven-chi.vercel.app](https://e-commerce-website-seven-chi.vercel.app/)

Code:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>E-Commerce Website</title>

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

<meta name="theme-color" content="#4285f4">

<link rel="apple-touch-icon" href="">

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

</head>

<body>

<header>

<h1>E-Commerce Website</h1>

<nav>

<ul>

<li><a href="#">Home</a></li>

<li><a href="#">Shop</a></li>

<li><a href="#">About</a></li>

<li><a href="#">Contact</a></li>

</ul>

</nav>

</header>

<main>

<section class="products">

<!-- Product cards will be dynamically generated here -->

</section>

</main>

<footer>

<p>&copy; 2024 E-Commerce Website. All rights reserved.</p>

</footer>

<script src="script.js"></script>

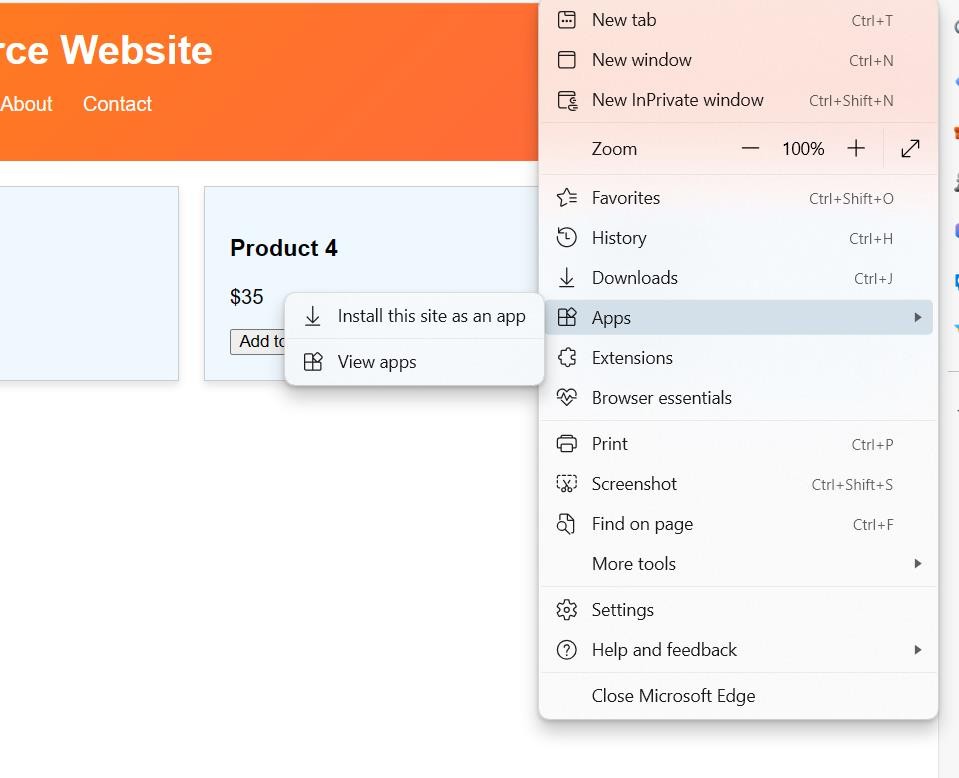
</body>

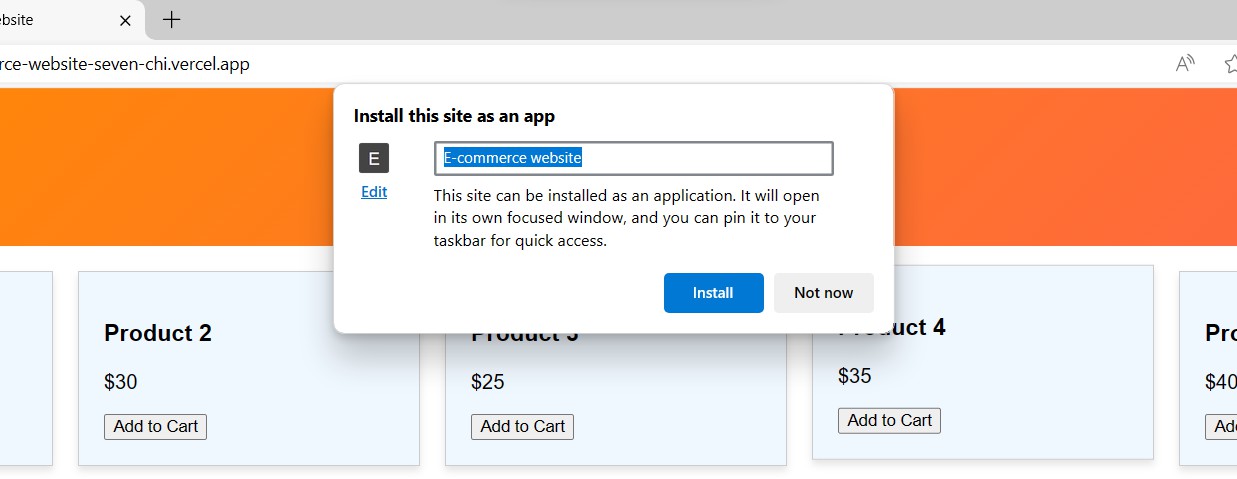
</html>

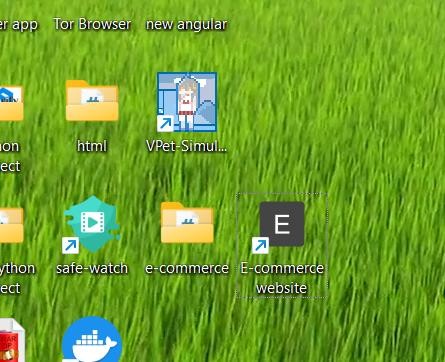
— Open this on browser

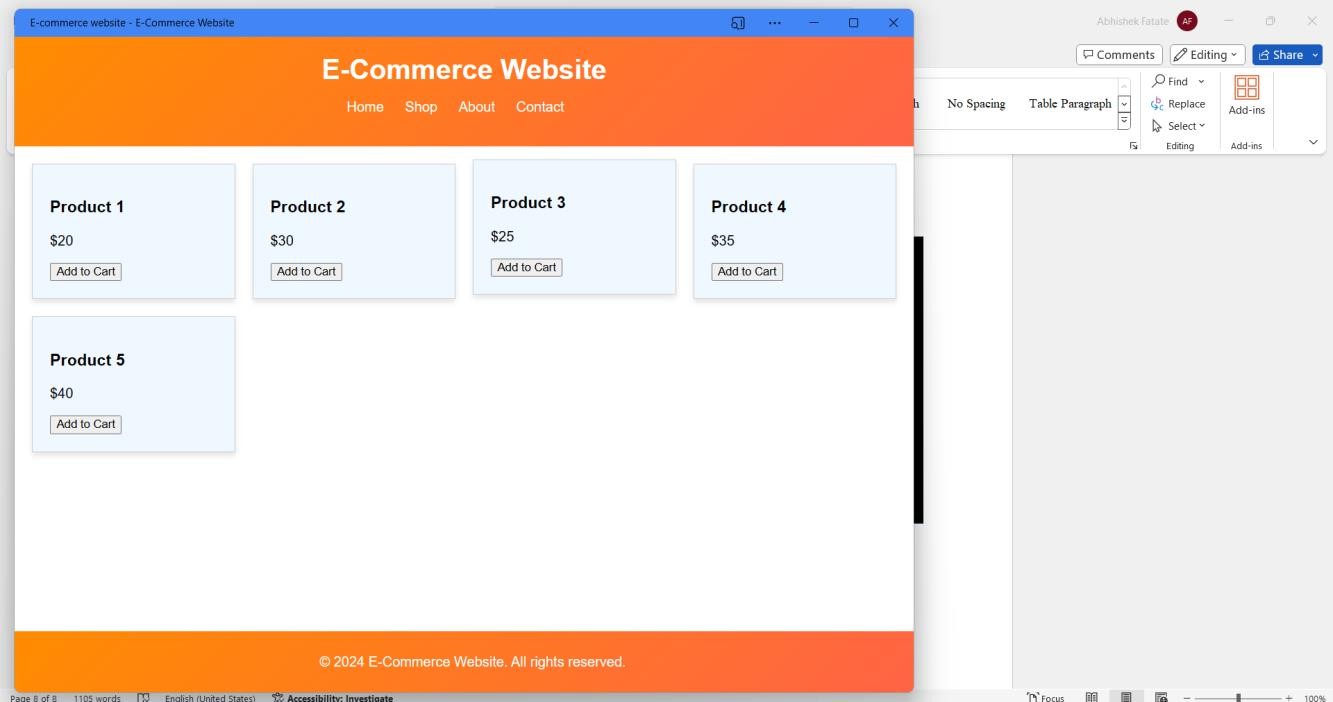


Click on 3 dots, and then applications, and then install this site as an app.









Conclusion: In this experiment, we successfully created a basic progressive application of our website and installed it on 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. | 15 |
| Name | Abhishek Fatate |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO5: Design and Develop a responsive User Interface by applying PWA Design techniques |
| Grade: | 12 |

PWA Lab Exp 8

Name: Abhishek Fatate Roll No. 15

Div: D15A Batch: A

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

**Theory**:

Service Worker

Service Worker is a script that works on browser background without user interaction independently. Also, It resembles a proxy that works on the user side.

With this script, you can track network traffic of the page, manage push notifications and develop “offline first” web applications with Cache API.

Things to note about Service Worker:

A service worker is a programmable network proxy that lets you control how network requests from your page are handled.

Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.

The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.

What can we do with Service Workers? You can dominate Network Traffic

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

You can also send a true response too.

You can Cache

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

You can manage Push Notifications

You can manage push notifications with Service Worker and show any information message to the user.

You can Continue

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

What can’t we do with Service Workers? You can’t access the Window

You can’t access the window, therefore, You can’t manipulate DOM elements. But,

you can communicate to the window through post Message and manage processes that you want.

You can’t work it on 80 Port

Service Worker just can work on HTTPS protocol. But you can work on localhost during 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.

Sample Code with Output

**index.html**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>E-Commerce Website</title>**

**<link rel="stylesheet" href="styles.css">**

**<meta name="theme-color" content="#4285f4">**

**<link rel="apple-touch-icon" href="">**

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

**</head>**

**<body>**

**<header>**

**<h1>E-Commerce Website</h1>**

**<nav>**

**<ul>**

**<li><a href="#">Home</a></li>**

**<li><a href="#">Shop</a></li>**

**<li><a href="#">About</a></li>**

**<li><a href="#">Contact</a></li>**

**</ul>**

**</nav>**

**</header>**

**<main>**

**<section class="products">**

**<!-- Product cards will be dynamically generated here -->**

**</section>**

**</main>**

**<footer>**

**<p>&copy; 2024 E-Commerce Website. All rights reserved.</p>**

**</footer>**

**<script src="script.js"></script>**

**</body>**

**</html>**

**app.js**

**if ('serviceWorker' in navigator) { window.addEventListener('load', () => {**

**navigator.serviceWorker.register('/service-worker.js')**

**.then(registration => {**

**console.log('Service Worker registered with scope:', registration.scope);**

**})**

**.catch(error => {**

**console.error('Service Worker registration failed:', error);**

**});**

**});**

**}**

**service-worker.js**

**// service-worker.js**

**const cacheName = 'ecommerce-pwa-v1'; const assetsToCache = [**

**'/',**

**'/index.html', '/main.css',**

**'/app.js'**

**// Add more files and assets here as needed**

**];**

**self.addEventListener('install', event => { event.waitUntil(**

**caches.open(cacheName)**

**.then(cache => {**

**return cache.addAll(assetsToCache);**

**})**

**);**

**});**

**self.addEventListener('activate', event => {**

**event.waitUntil( caches.keys().then(cacheNames => {**

**return Promise.all( cacheNames.filter(name => {**

**return name !== cacheName;**

**}).map(name => {**

**return caches.delete(name);**

**})**

**);**

**})**

**);**

**});**

Steps for Execution

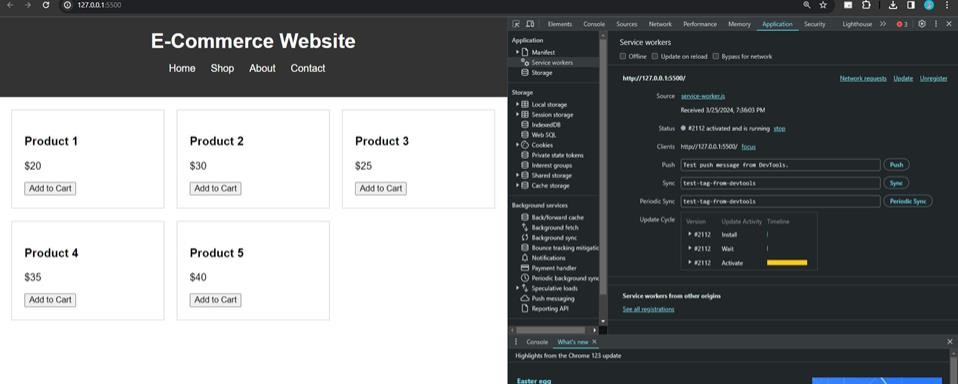
Create a folder and put all 4 files main.css , service-worker.js, app.js, index.html open visual studio

install extension Live server

open folder in visual studio open index.html on bottom right corner click go Live

it will open html page in browser

go to developer tools and take following screenshots



# **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. | 15 |
| Name | Abhishek Fatate |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO5: Design and Develop a responsive User Interface by applying PWA Design techniques |
| Grade: | 12 |

PWA Lab Exp 9

Name: Abhishek Fatate Roll No. 15

Div: D15A Batch: A

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

# Theory**:**

**Service Worker**

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

Things to note about Service Worker:

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

# Fetch Event

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

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

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

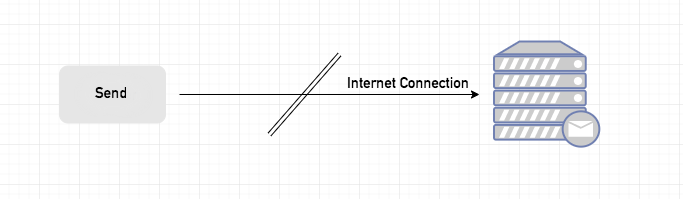


# Sync Event

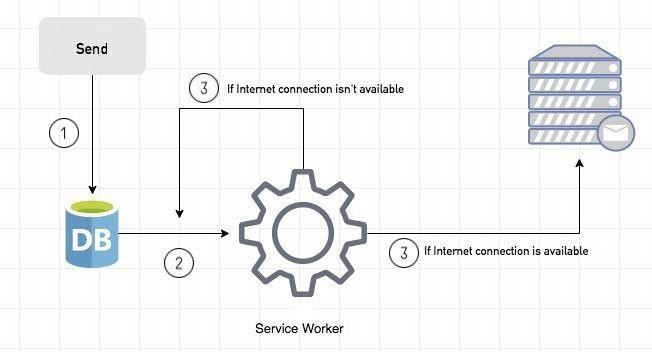
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are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

Here is a job for the Background Sync.

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



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



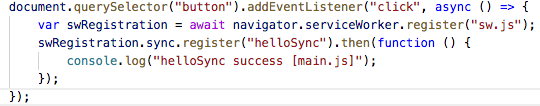
1. When we click the “send” button, email content will be saved to IndexedDB.
2. Background Sync registration.
3. **If the Internet connection is available**, all email content will be read and sent to Mail

Server.

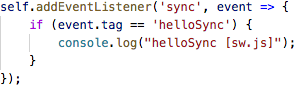
**If the Internet connection is unavailable**, the service worker waits until the connection is available even though the window is closed. When it is available, email content will be sent to Mail Server.

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

Event Listener for Background Sync Registration



Event Listener for 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.

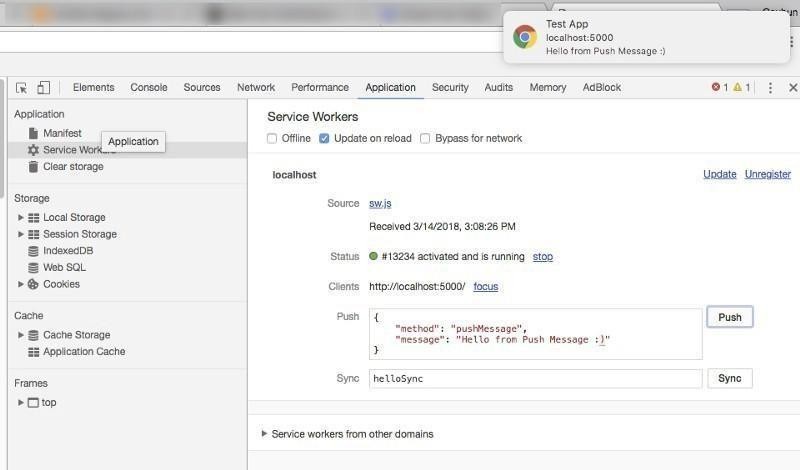
We can check in the following example.

“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.

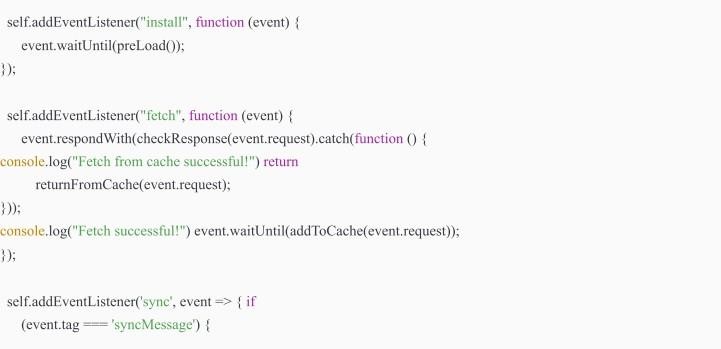
In the following code block is in sw.js file. You can handle push notifications with this event. In this example, I kept it simple. We send an object that has “method” and “message” properties. If the method value is “pushMessage”, we open the information notification with the “message” property.



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



**Cod e**: sw.js



console.log("Sync successful!")

}

});

self.addEventListener('push', function (event) { if (event && event.data) {

var data = event.data.json();

if (data.method == "pushMessage")

{ console.log("Push notification sent");

event.waitUntil(self.registration.showNotification("Omkar Sweets Corner", { body: data.message

}))

}

}

})

var filesToCache

= [ '/',

'/menu', '/contactUs', '/offline.htm l',

];

var preLoad = function () {

return caches.open("offline").then(function (cache) {

*// caching index and important routes*

return cache.addAll(filesToCache);

});

};

var checkResponse = function (request) { return new Promise(function (fulfill, reject) {

fetch(request).then(function (response) { if (response.status

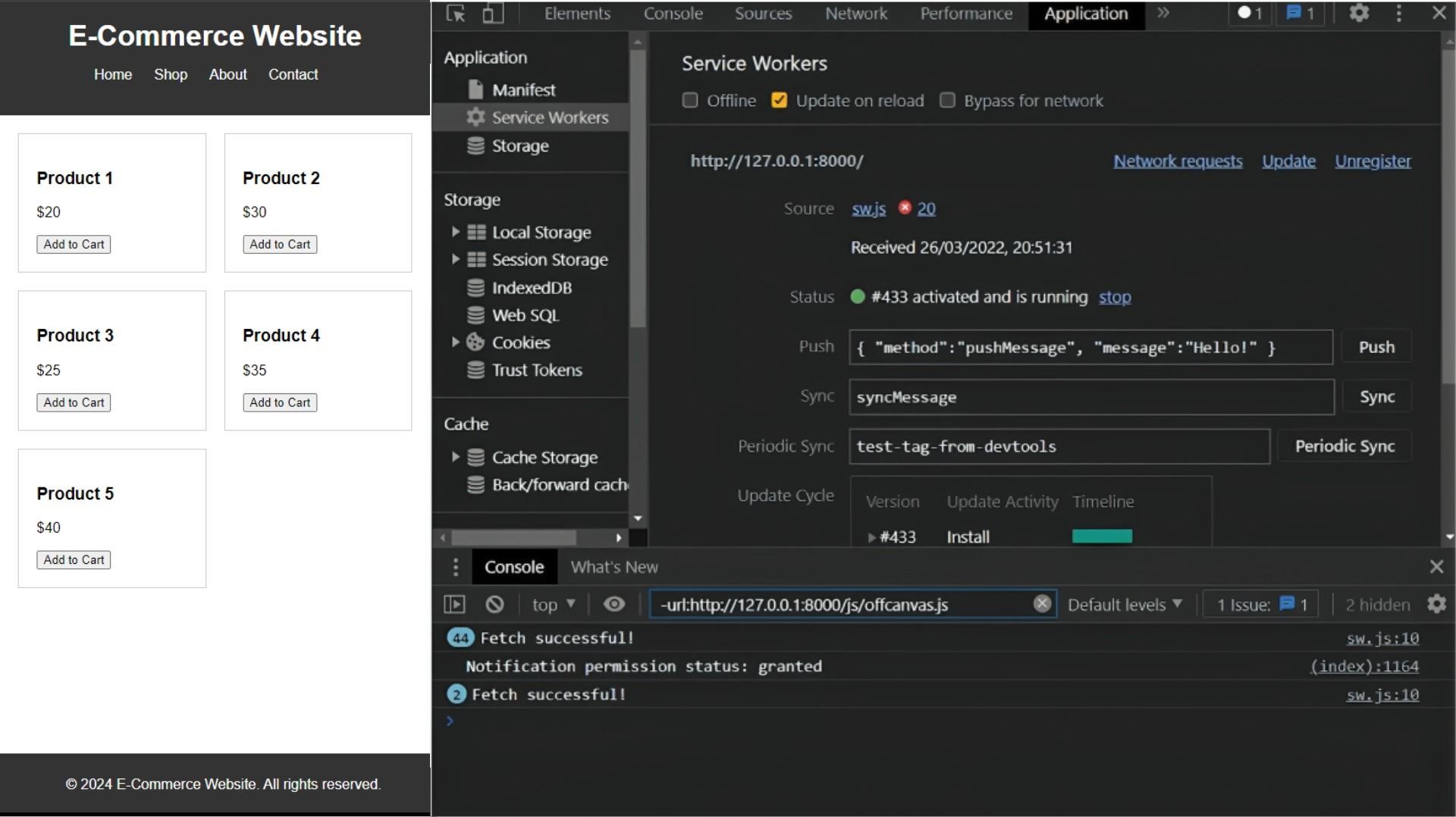
!== 404) {

fulfill(response);

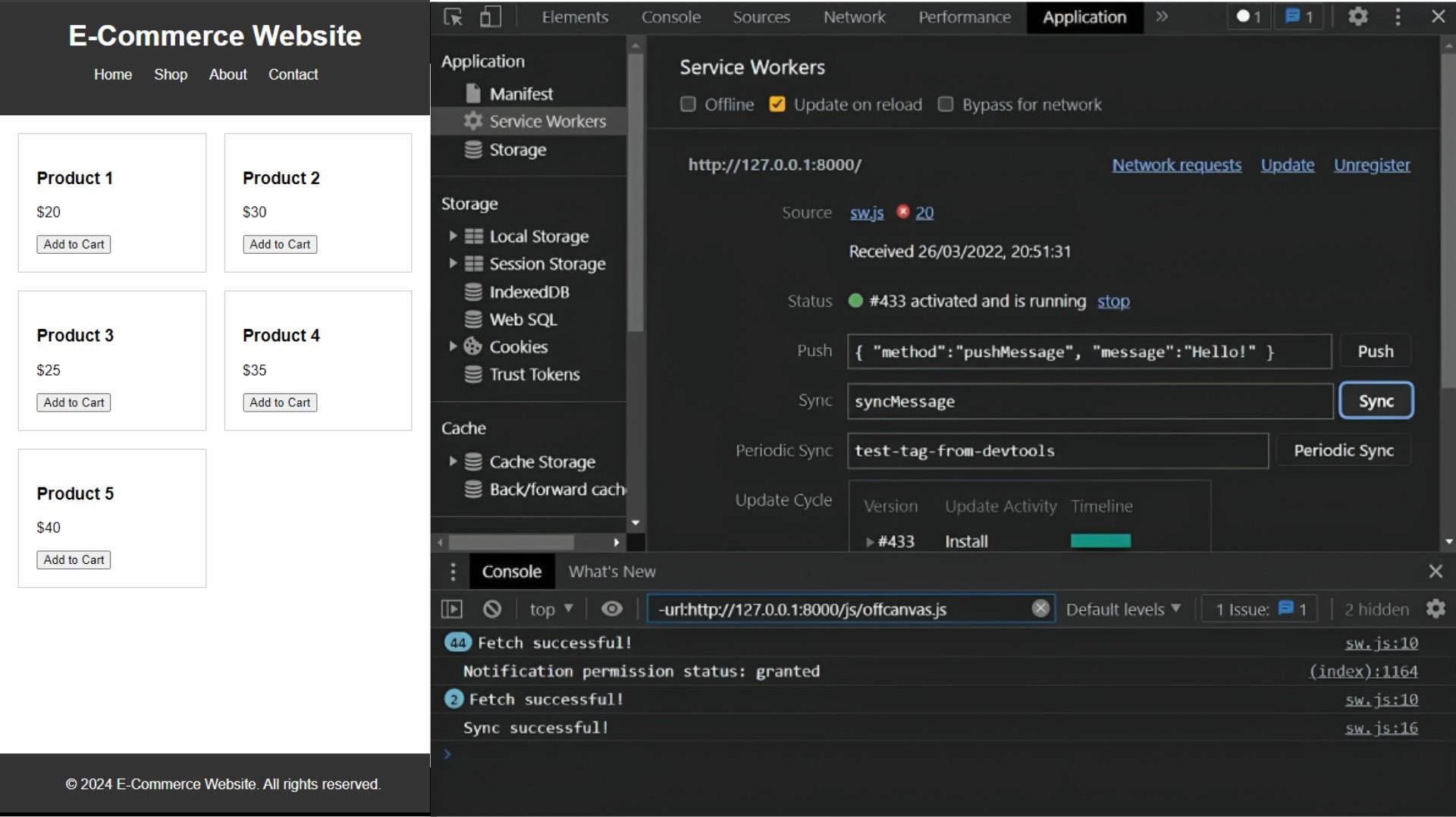
} else { reject();



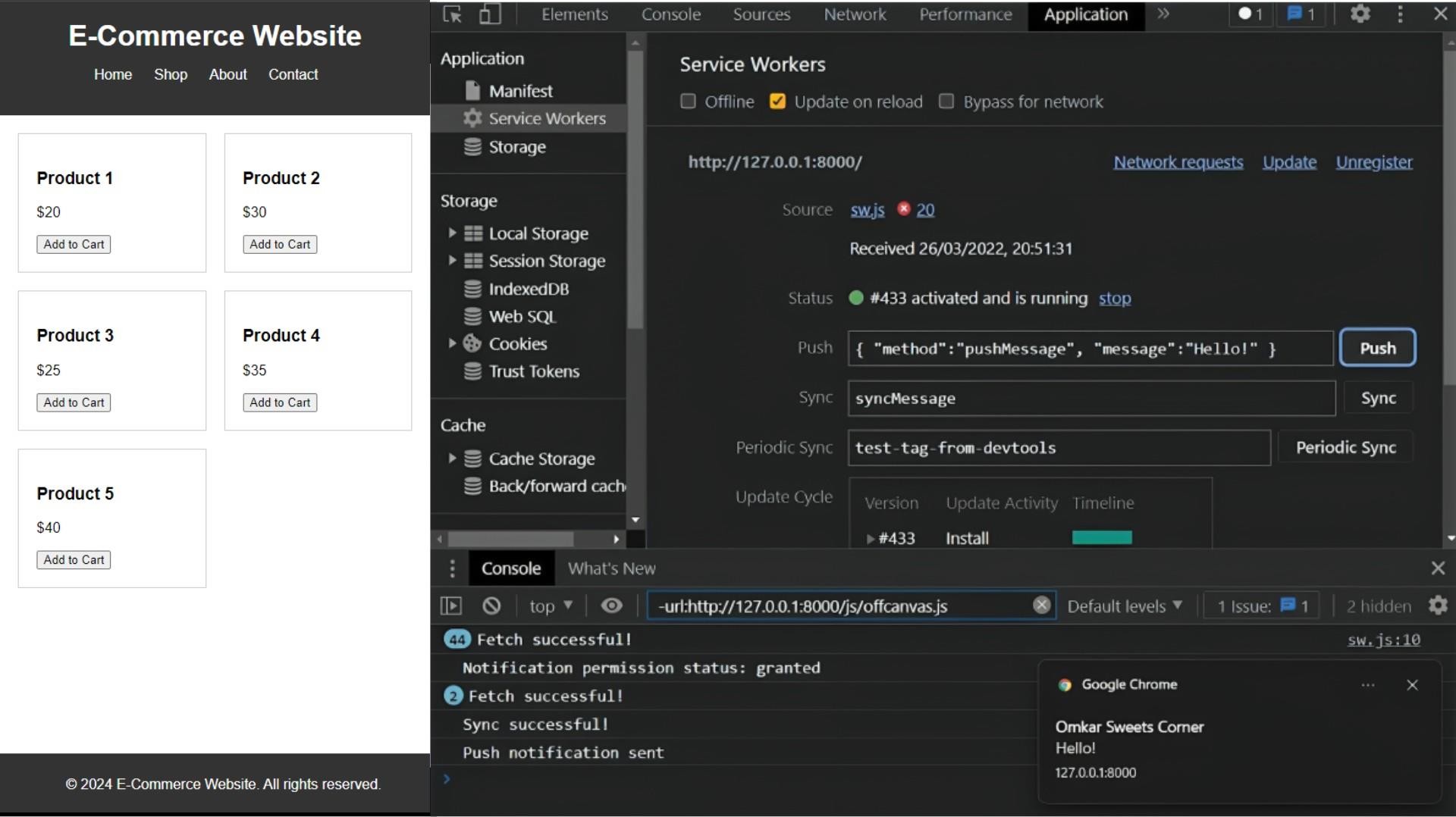
**Output**: Fetch Event



Sync event



Push event



# **MAD & PWA Lab Journal**

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

# PWA Lab Exp 10

Name: Abhishek Fatate Roll No. 15

Div: D15A Batch: A

**Aim**:

To study and implement deployment of website 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

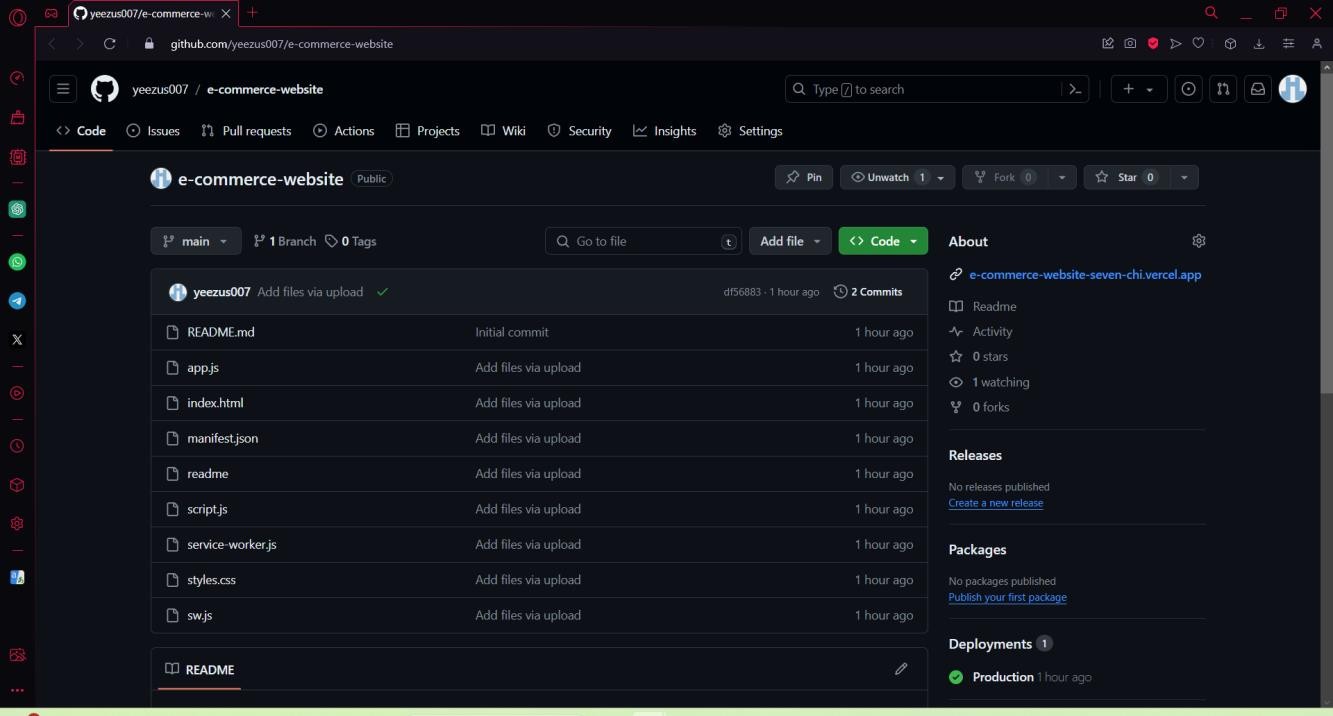
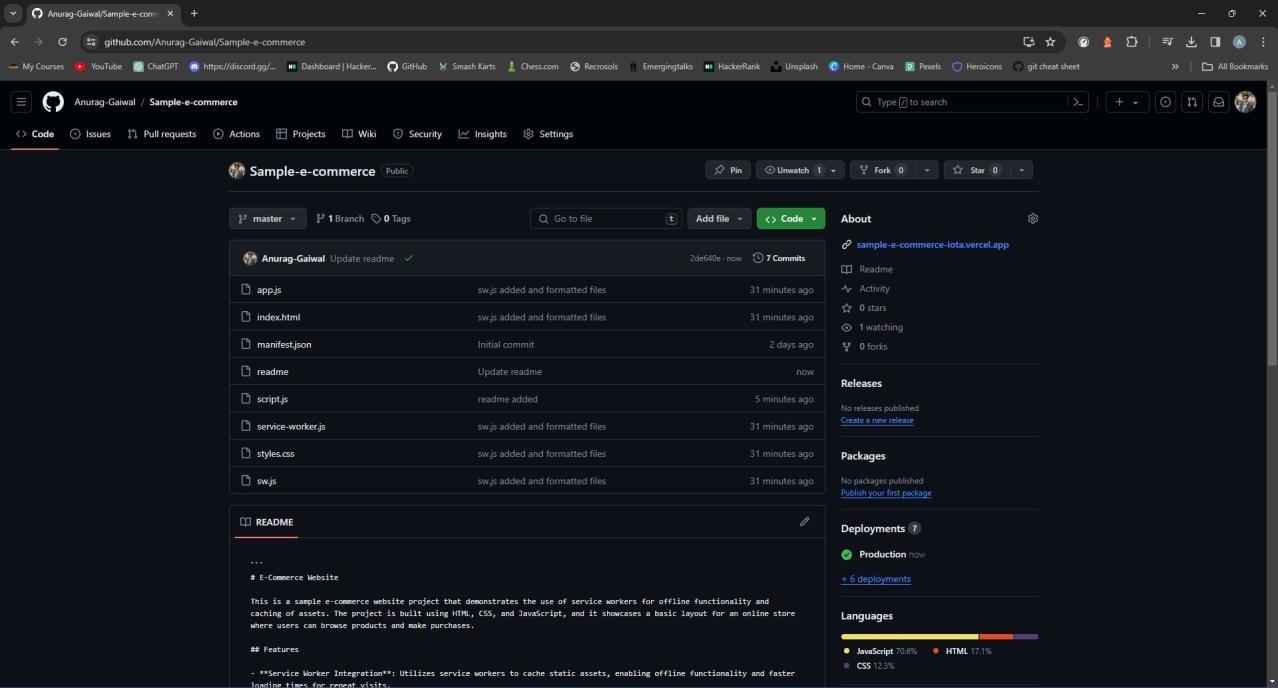
1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You’ll also have access to a blob store, which can store another 1 GB of data.
5. 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 to our GitHub repository:** [**https://github.com/yeezus007/e-commerce-website**](https://github.com/yeezus007/e-commerce-website)

**Github Screenshot:**



# **MAD & PWA Lab Journal**

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

PWA Lab Exp 11

Name: Abhishek Fatate Roll No. 15

Div: D15A Batch: A

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

Reference : https://[www.semrush.com/blog/google-lighthouse/](http://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:

Performance: This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th percentile and so on.

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

Accessibility: As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the ‘aria-’ attributes like aria-required, audio captions, button names,

etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.

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

Code:

Index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>E-Commerce Website</title>

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

<meta name="theme-color" content="#4285f4">

<link rel="apple-touch-icon" href="">

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

</head>

<body>

<header>

<h1>E-Commerce Website</h1>

<nav>

<ul>

<li><a href="#">Home</a></li>

<li><a href="#">Shop</a></li>

<li><a href="#">About</a></li>

<li><a href="#">Contact</a></li>

</ul>

</nav>

</header>

<main>

<section class="products">

<!-- Product cards will be dynamically generated here -->

</section>

</main>

<footer>

<p>&copy; 2024 E-Commerce Website. All rights reserved.</p>

</footer>

<script src="script.js"></script>

</body>

</html>

Styles.css:

body {

font-family: Arial, sans-serif; margin: 0;

padding: 0;

}

header {

background-color: #333; color: #fff;

padding: 20px; text-align: center;

}

header h1 {

margin: 0;

}

nav ul {

list-style-type: none; padding: 0;

}

nav ul li {

display: inline; margin-right: 20px;

}

nav ul li a {

color: #fff;

text-decoration: none;

}

main {

padding: 20px;

}

.products {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); grid-gap: 20px;

}

.product {

border: 1px solid #ccc; padding: 20px;

}

footer {

background-color: #333; color: #fff;

text-align: center; padding: 10px; position: fixed; bottom: 0;

width: 100%;

}

script.js:

// Simulated data for products const products = [

{ name: "Product 1", price: 20 },

{ name: "Product 2", price: 30 },

{ name: "Product 3", price: 25 },

{ name: "Product 4", price: 35 },

{ name: "Product 5", price: 40 }

];

// Function to generate product cards function generateProductCards() {

const productsSection = document.querySelector('.products');

products.forEach(product => {

const productCard = document.createElement('div'); productCard.classList.add('product'); productCard.innerHTML = `

<h3>${product.name}</h3>

<p>$${product.price}</p>

<button>Add to Cart</button>

`; productsSection.appendChild(productCard);

});

}

// Call the function to generate product cards generateProductCards();

Manifest.json:

{

"name": "E-commerce website", "short\_name": "E-commerce", "start\_url": "index.html", "scope": "./", "theme\_color": "#ffd31d", "background\_color": "#333", "display": "standalone", "icons": [

{

"src": "icon-1.png",

"sizes": "192x192",

"type": "image/png", "purpose": "any maskable"

},

{

"src": "icon-2.png",

"sizes": "512x512",

"type": "image/png", "purpose": "any maskable"

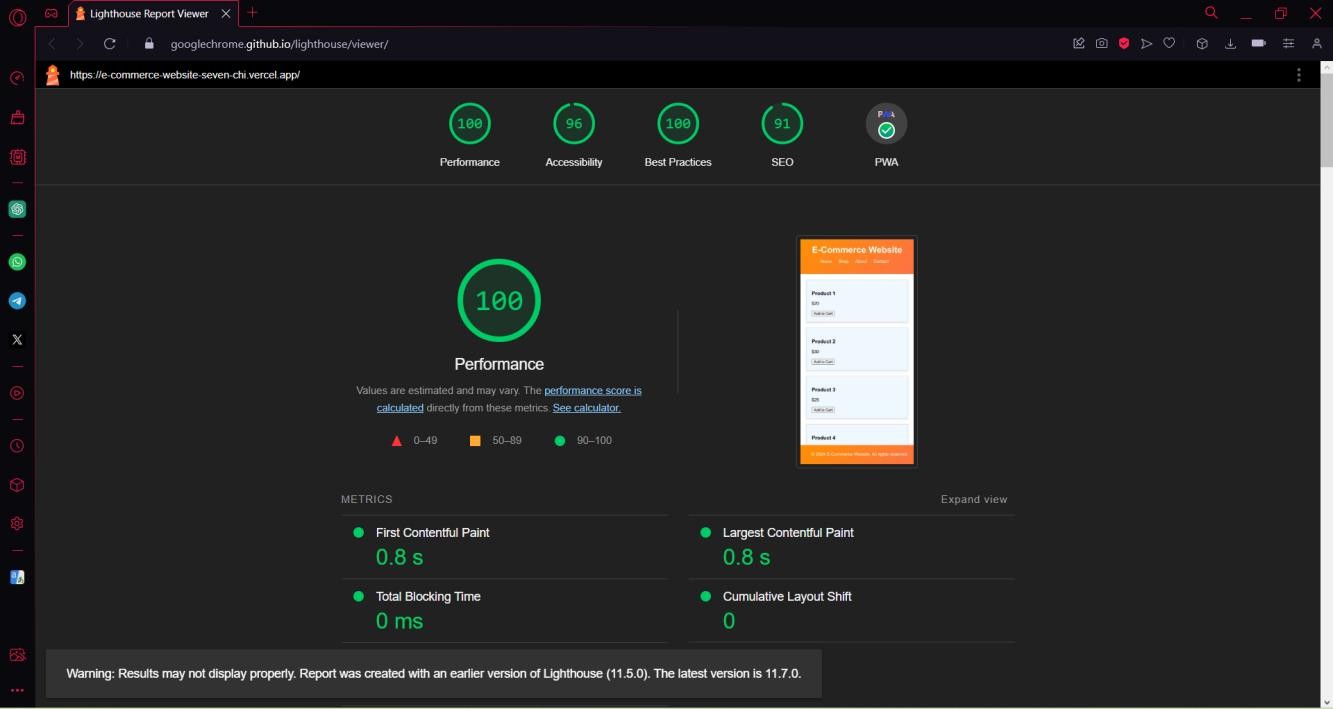
}

]

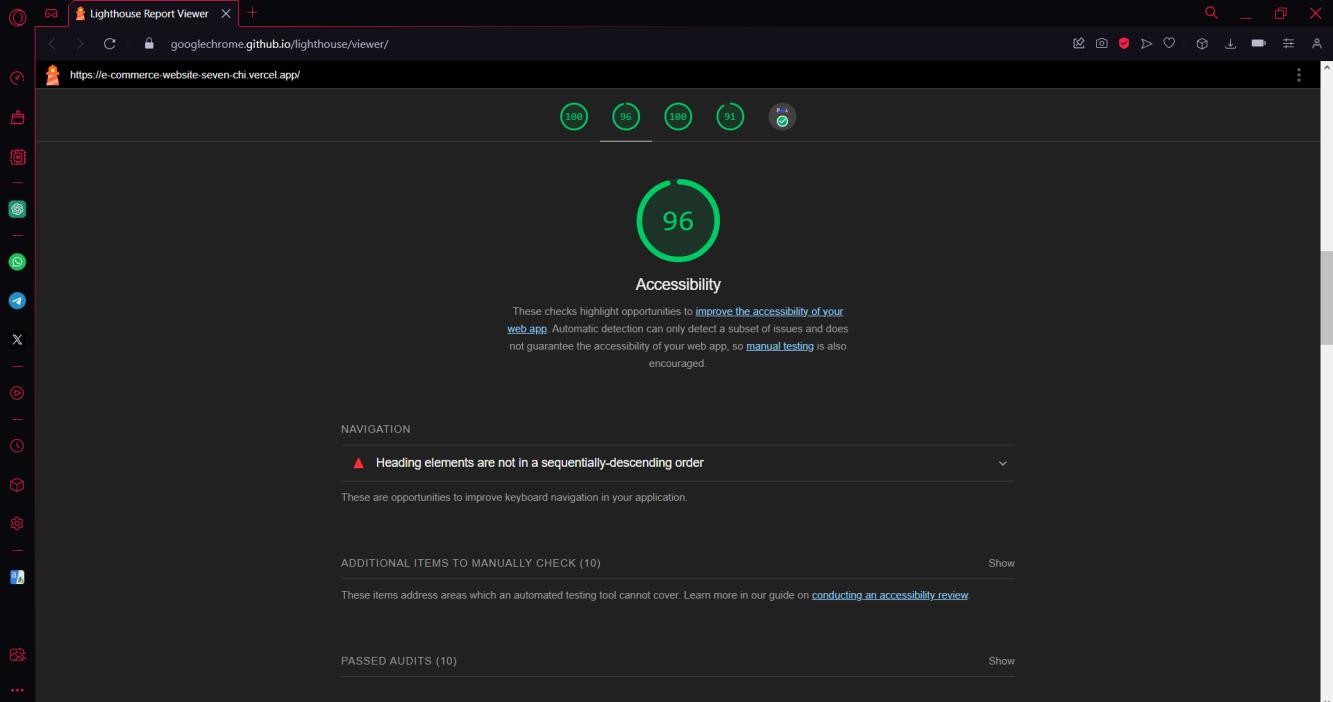
}

Output:

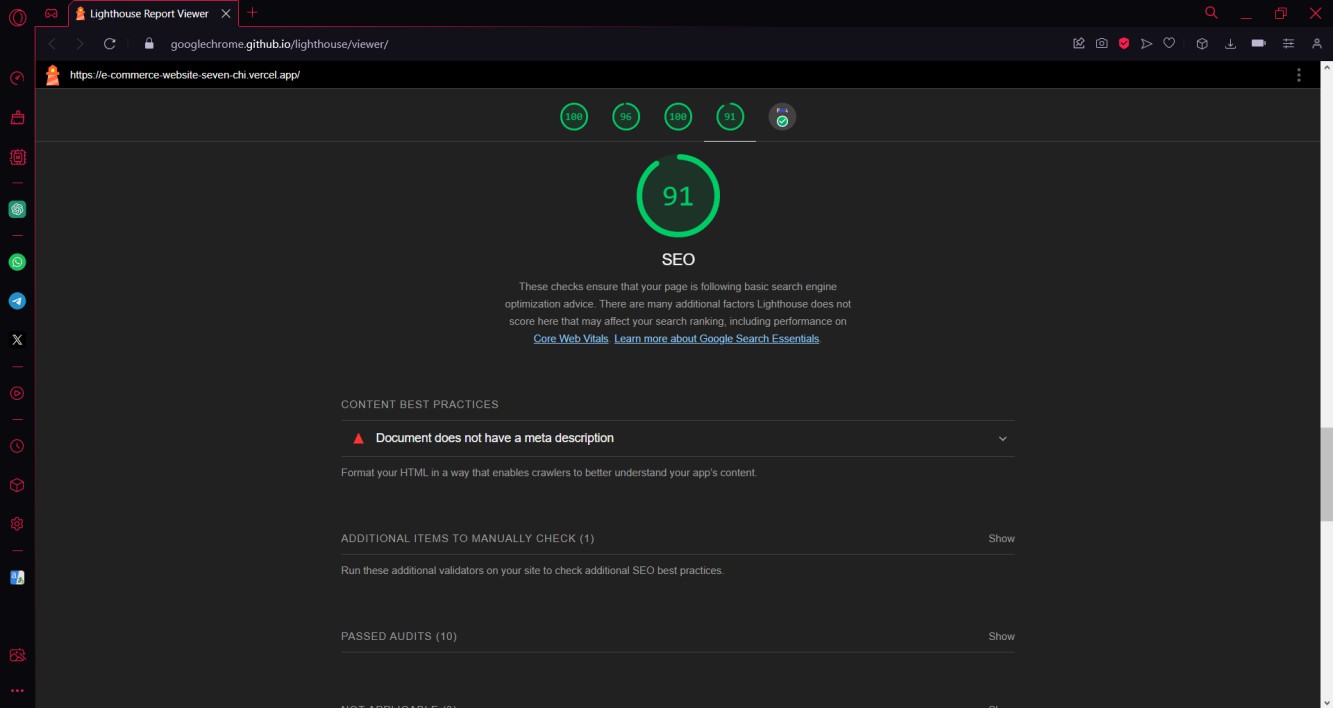
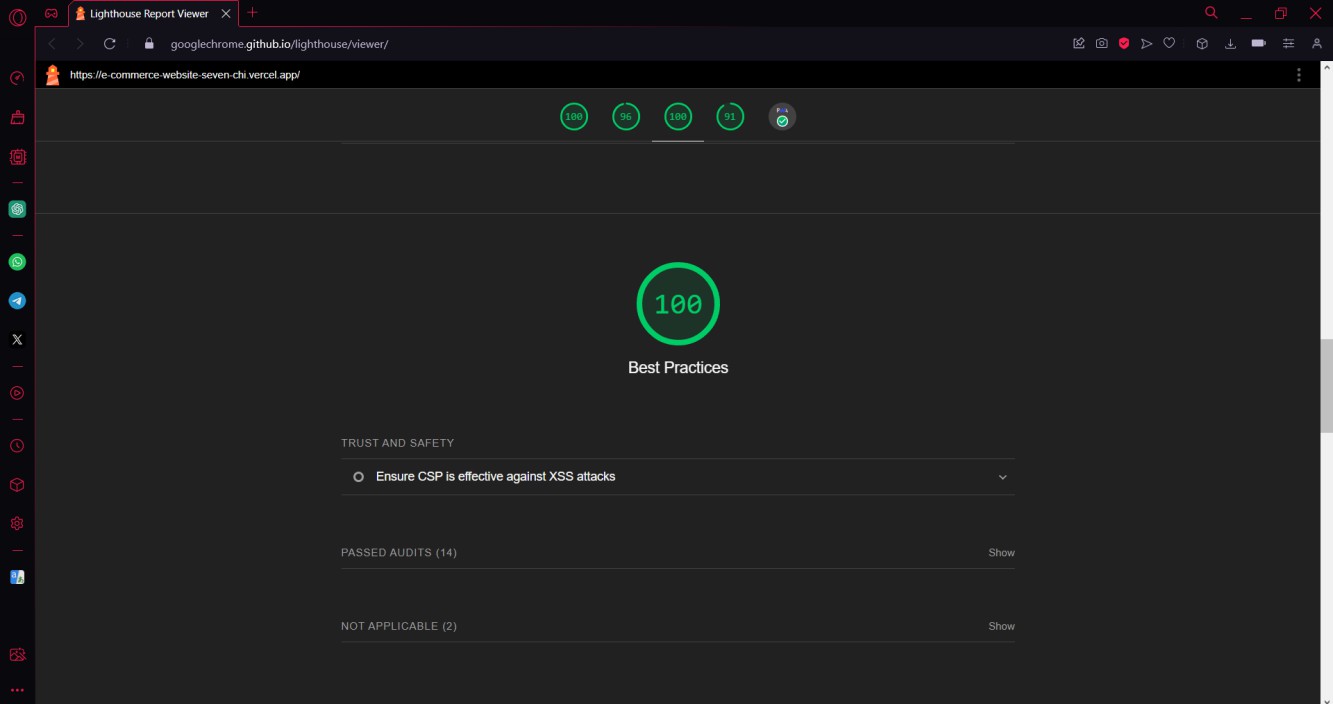
Performance -



Accessibility -

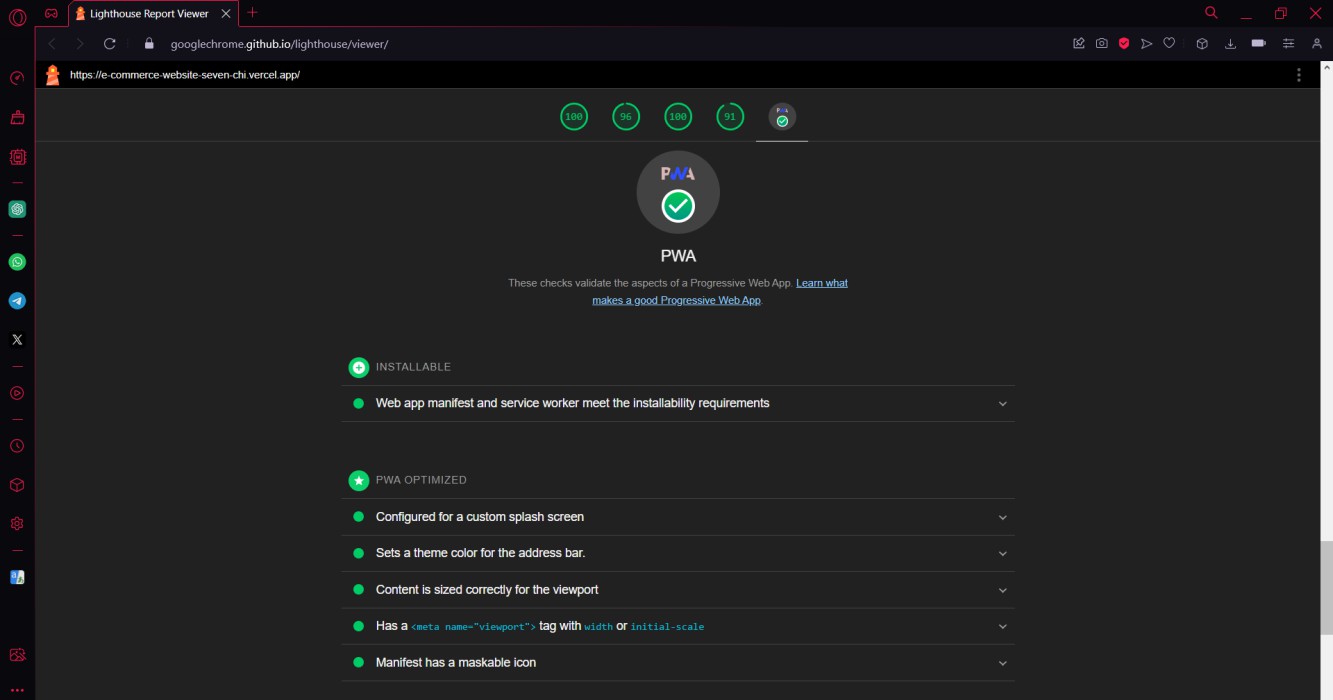


Best Practices :



SEO :

PWA :

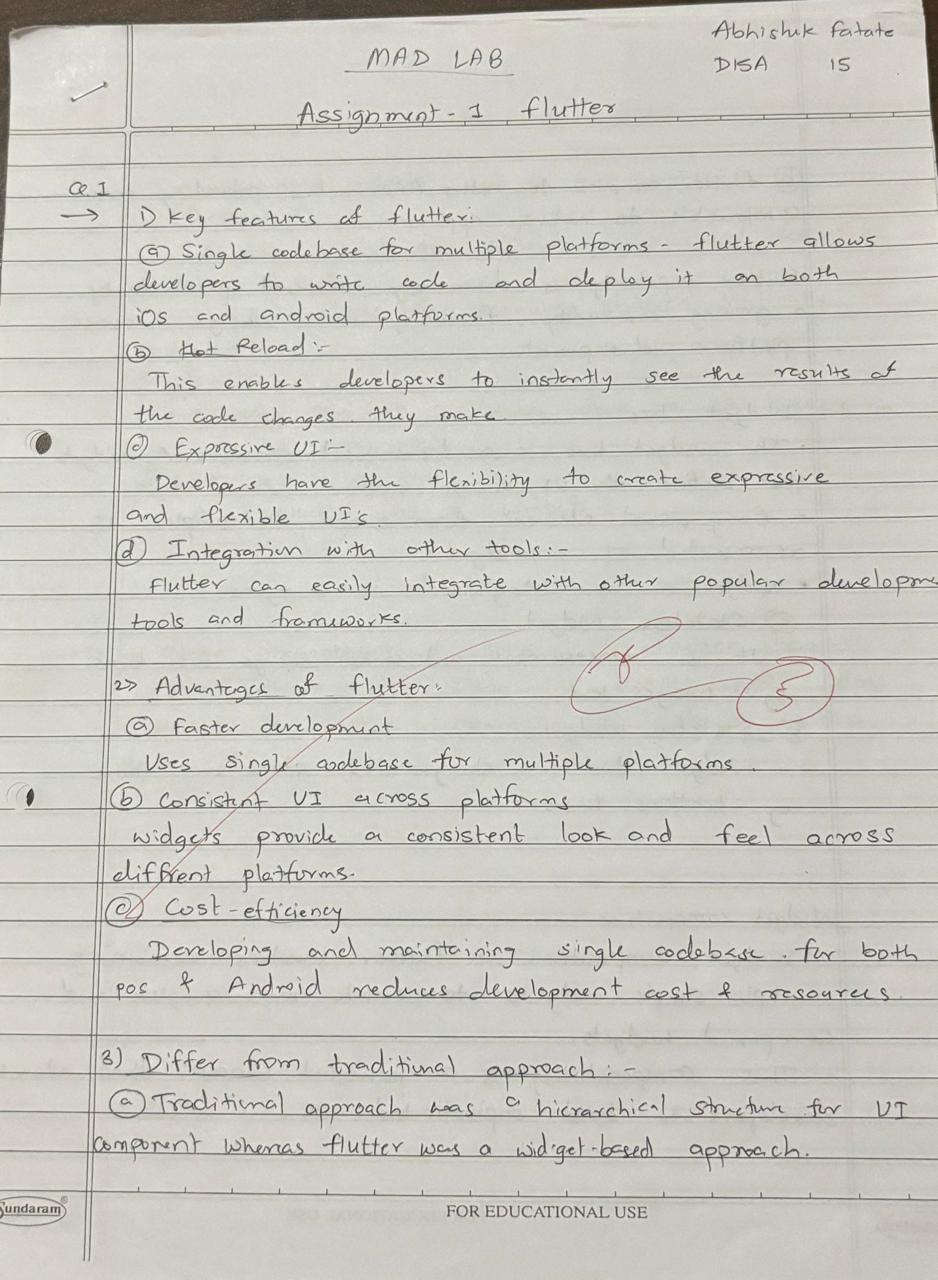


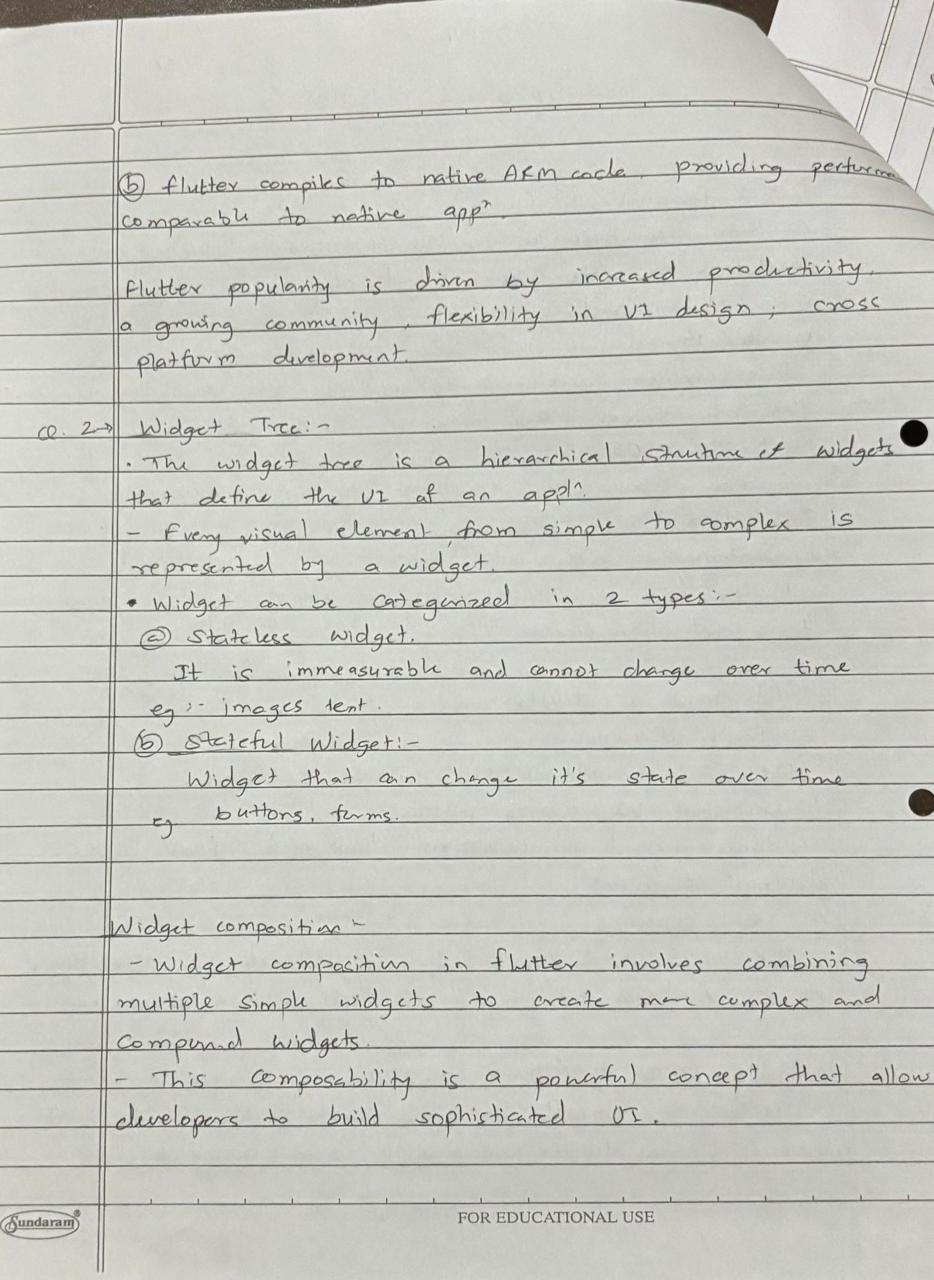
Hosted Website : [https://e-commerce-website-seven-chi.vercel.app](https://sample-e-commerce-iota.vercel.app/)

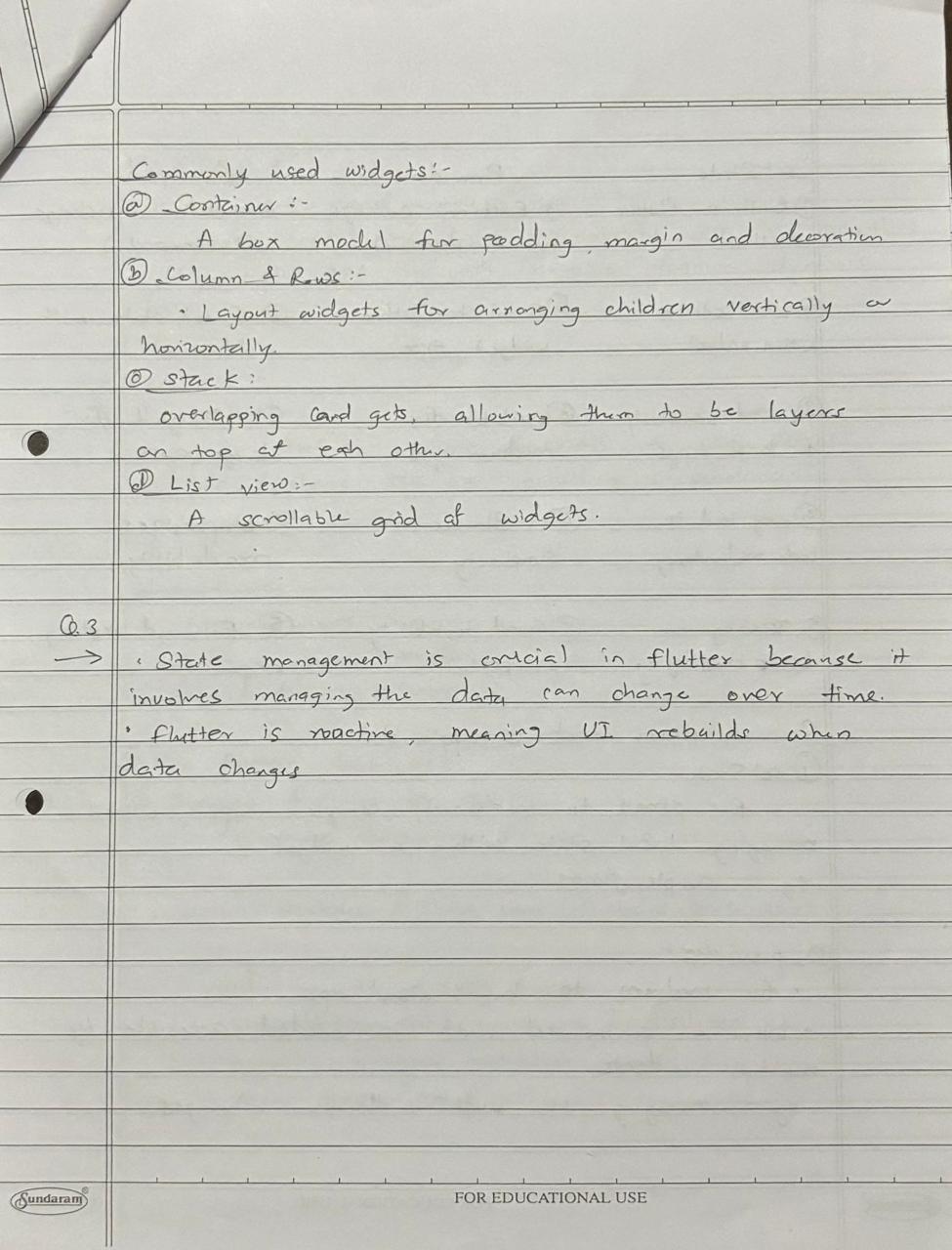
Conclusion : Thus we successfully used google Lighthouse PWA Analysis Tool for testing the PWA functioning.

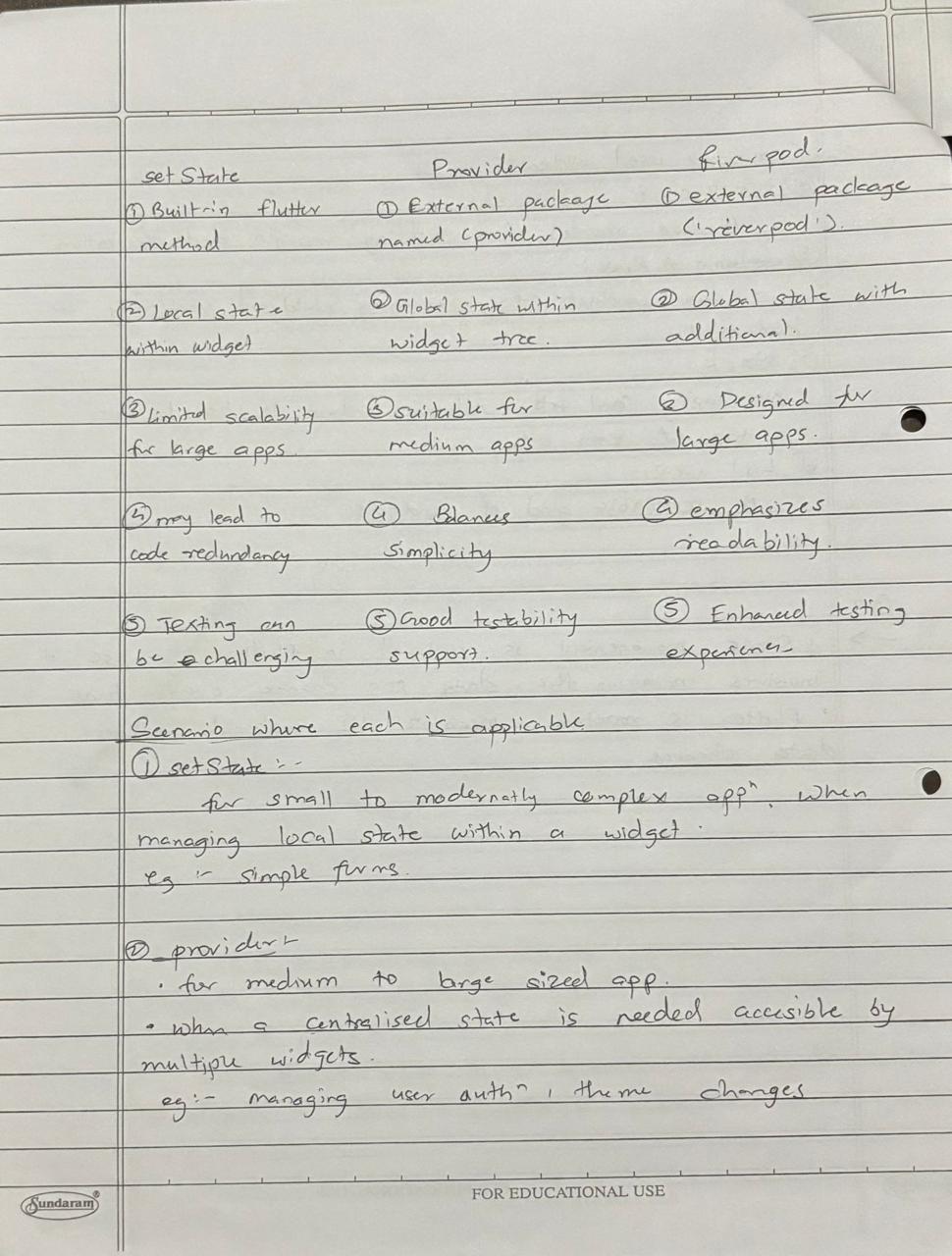
# **MAD & PWA Lab Journal**

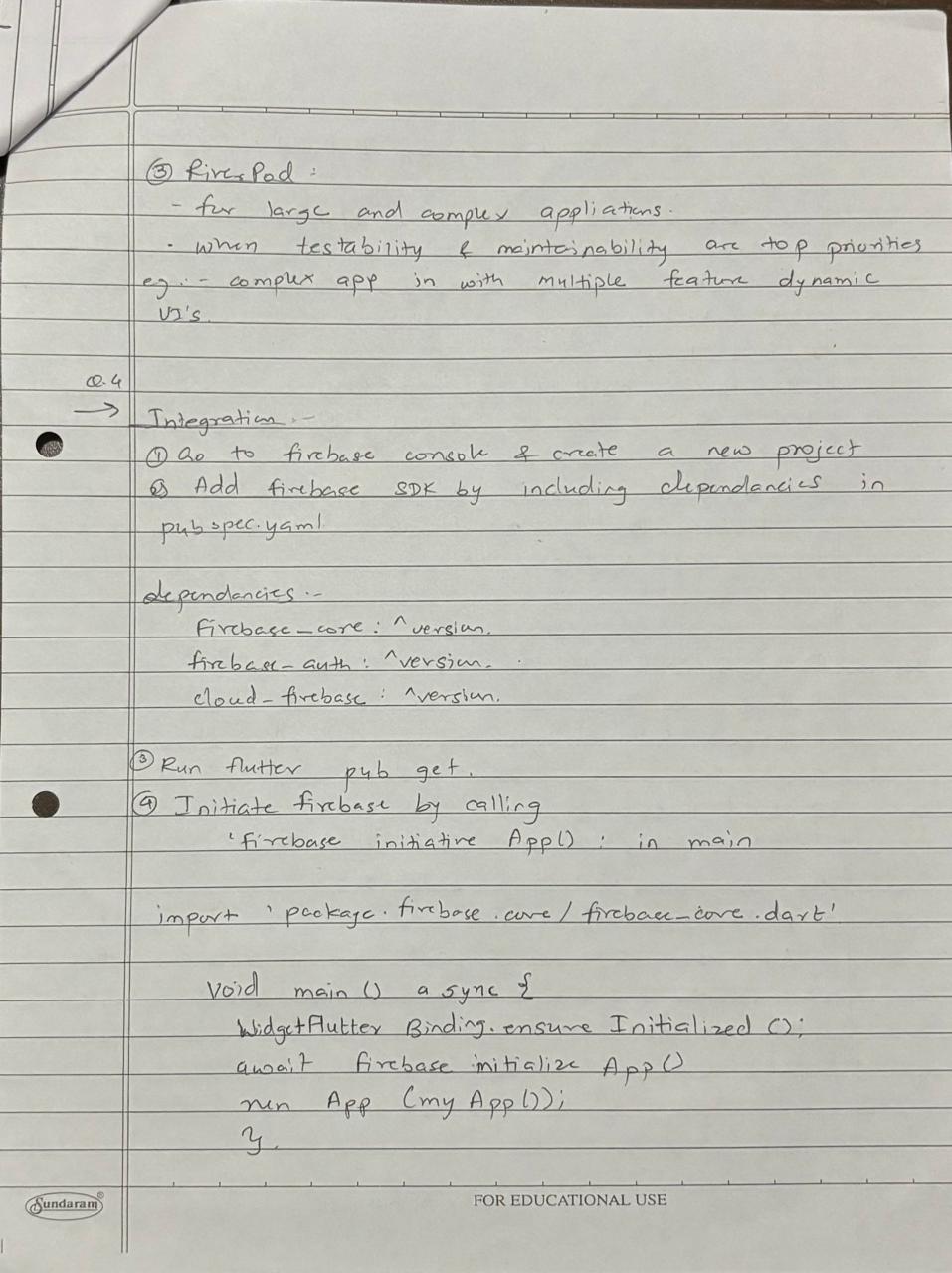
|  |  |
| --- | --- |
| Experiment No. | Assignment-1 |
| Assignment 1 Questions | 1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved. |
| Roll No. | 15 |
| Name | Abhishek Fatate |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO1: Understand cross platform mobile application development using Flutter framework  LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation  LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS |
| Grade: | 5 |

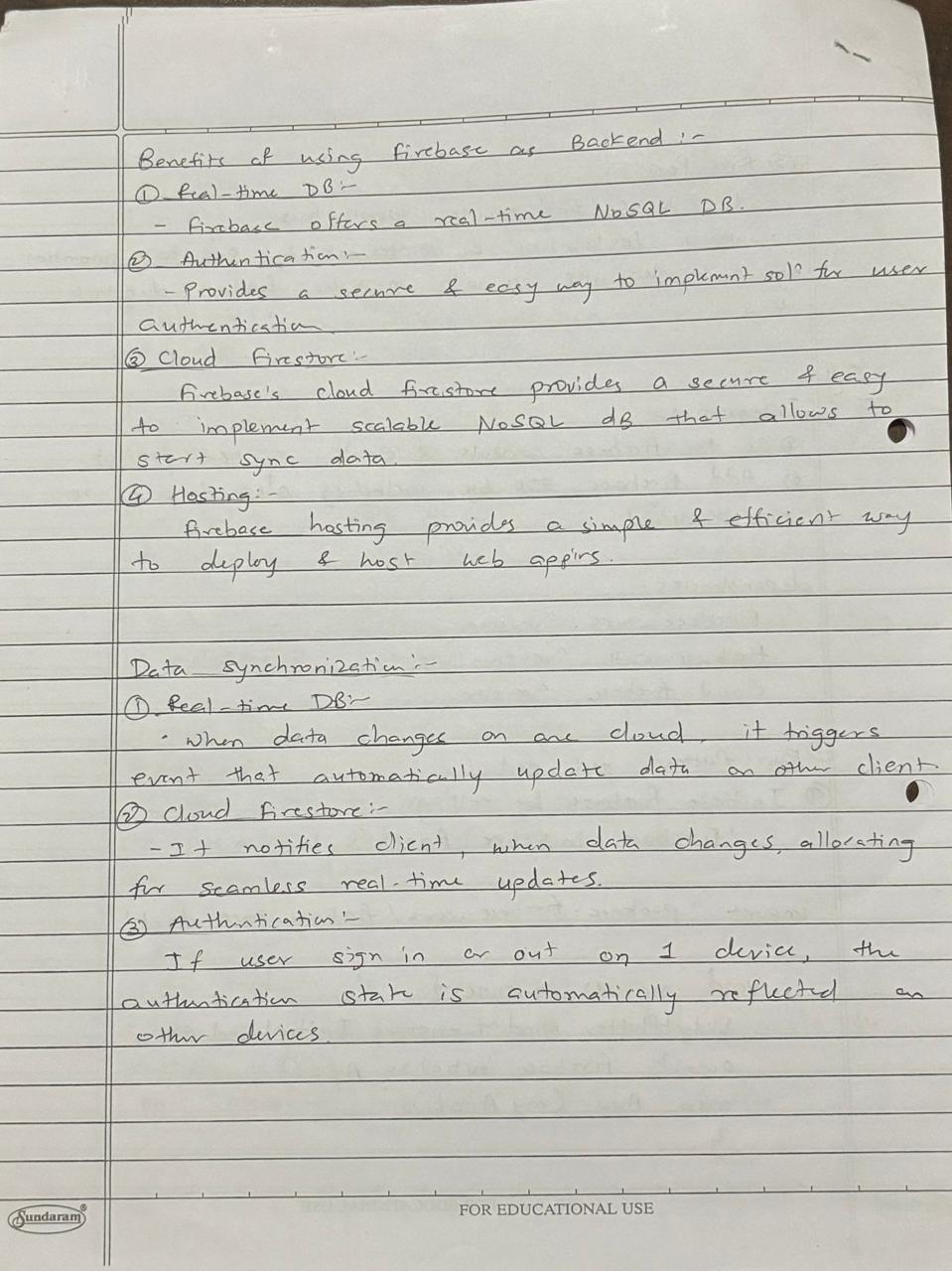














|  |  |
| --- | --- |
| Experiment No. | Assignment-2 |
| Assignment 2 Questions | 1. Define Progressive Web App (PWA) and explain its significance in modern web development.Discuss the key characteristics that differentiate PWAs from traditional mobile apps 2. Define responsive web design and explain 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. | 15 |
| Name | Abhishek Fatate |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques  LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions |
| Grade: | 4 |

