Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

CERTIFICATE

This is to certify that **Pushkar Mavale** 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 **2021-2022.**

Subject Teacher
Name:
Signature:

Head of Department
Name:
Signature:

External Examiner
Signature

Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Since 1962

Department of Information Technology

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

Year/Sem/Class : D15A

Faculty Incharge : Mrs. Kajal Jewani.

Lab Teachers : Mrs. Kajal Jewani.

Email : kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Lab Objectives:

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

Lab Outcomes:

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

Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Index

Name of Student: Pushkar Mavale

Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

MAD & PWA Lab Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	45
Name	Pushkar Mavale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	
Grade:	

Name: Pushkar Mavale **Class / Roll No:** D15A / 45

MAD and PWA Lab

Experiment No: 01

To Install and Configure Flutter Environment

Aim: Installation and Configuration of Flutter Environment.

Theory:

Flutter is a free and open-source mobile UI framework created by Google and released in May 2017. In a few words, it allows you to create a native mobile application with only one codebase. This means that you can use one programming language and one codebase to create two different apps (for iOS and Android).

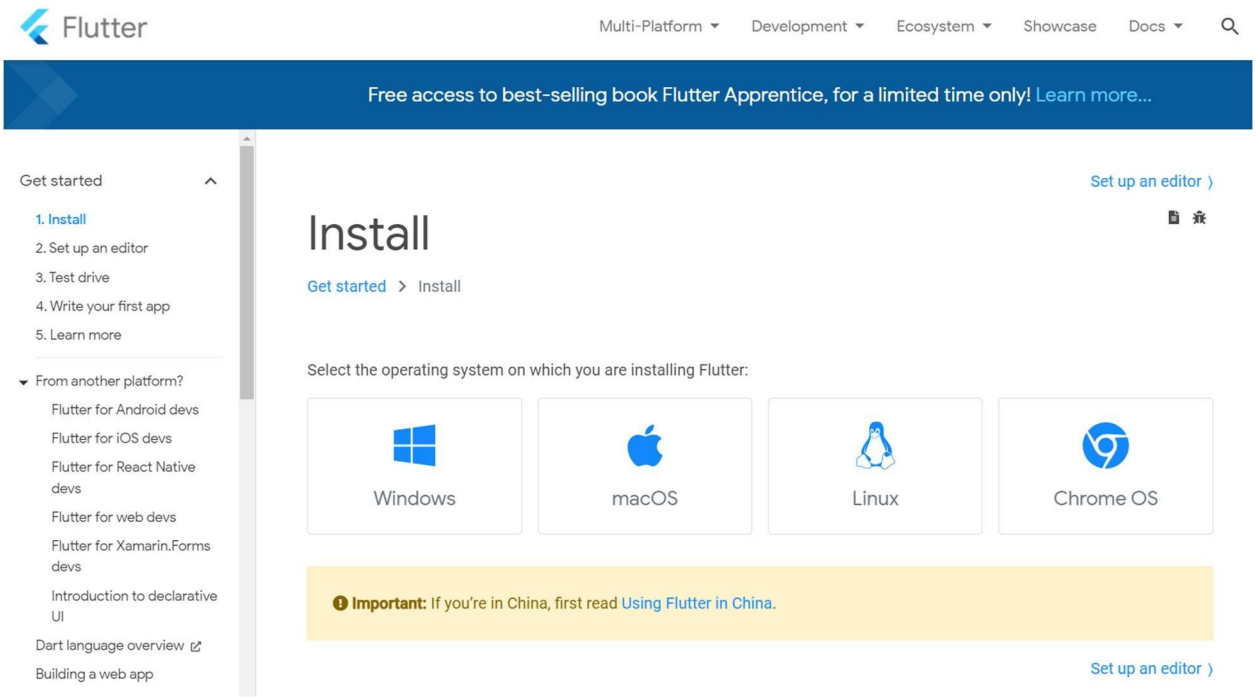
Flutter consists of two important parts:

- An SDK (Software Development Kit): A collection of tools that are going to help you develop your applications. This includes tools to compile your code into native machine code (code for iOS and Android).
- A Framework (UI Library based on widgets): A collection of reusable UI elements (buttons, text inputs, sliders, and so on) that you can personalize for your own needs.

To develop with Flutter, you will use a programming language called Dart. The language was created by Google in October 2011, but it has improved a lot over these past years. Dart focuses on front-end development, and you can use it to create mobile and web applications.

Steps:

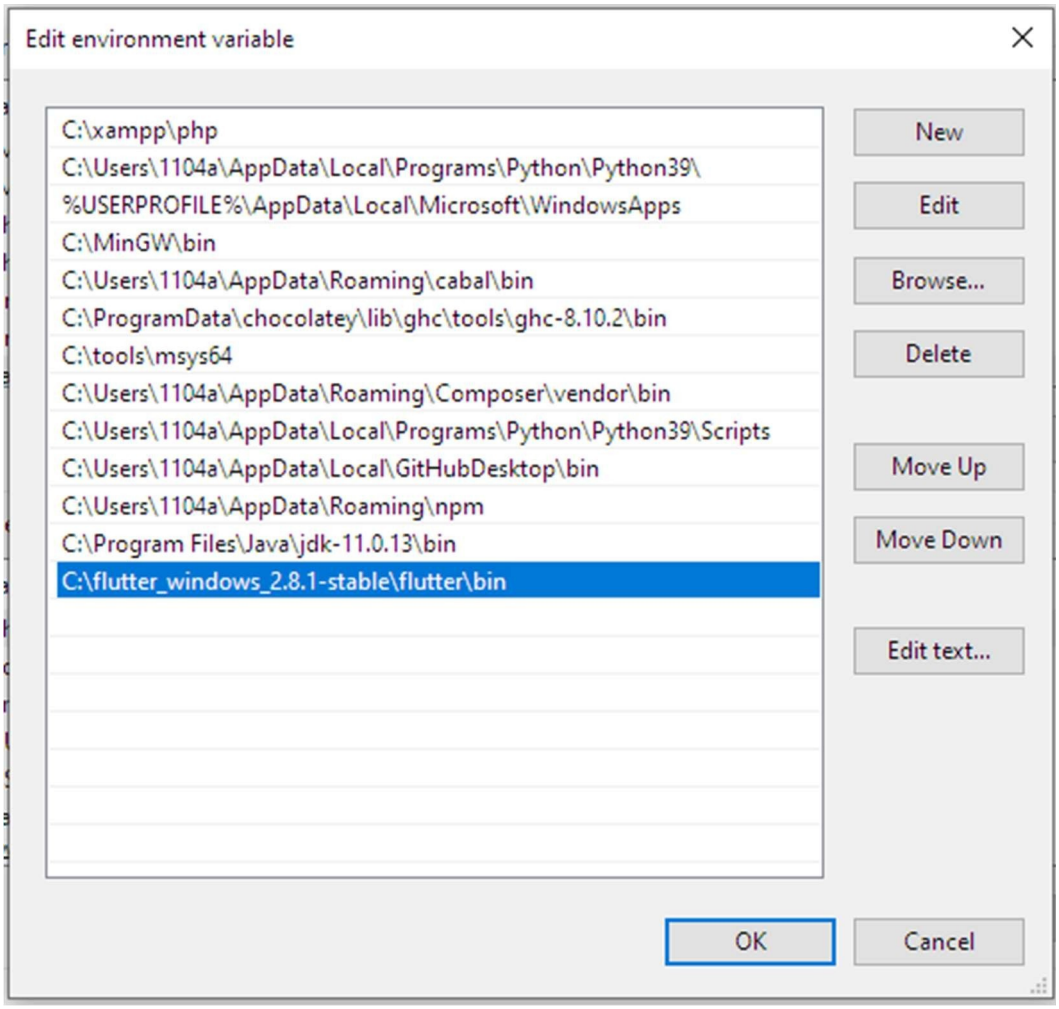
1. Download the installation bundle of the Flutter Software Development Kit for windows. To download **Flutter SDK**, go to its official website https://docs.flutter.dev/get-started/install, you will get the following screen.



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

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

4. To run the Flutter command in the regular windows console, you need to update the **system path** to include the flutter bin directory. The following steps are required to do this:
    a. Go to **My Computer properties** -> **Advanced tab** -> **Environment variables**.
    b. Now, **select path** -> **click on edit**. The following screen appears:



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

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

```
C:/■/system32
19:39:05 ❯ flutter
Manage your Flutter app development.

Common commands:

  flutter create <output directory>
    Create a new Flutter project in the specified directory.

  flutter run [options]
    Run your Flutter application on an attached device or in an emulator.

Usage: flutter <command> [arguments]

Global options:
-h, --help                  Print this usage information.
-v, --verbose               Noisy logging, including all shell commands executed.
                            If used with "--help", shows hidden options. If used with "flutter doctor", shows additional diagnostic information. (Use "-vv" to force
                            verbose logging in those cases.)
-d, --device-id             Target device id or name (prefixes allowed).
    --version               Reports the version of this tool.
    --suppress-analytics    Suppress analytics reporting when this command runs.

Available commands:

Flutter SDK
  bash-completion   Output command line shell completion setup scripts.
  channel           List or switch Flutter channels.
  config            Configure Flutter settings.
  doctor            Show information about the installed tooling.
  downgrade         Downgrade Flutter to the last active version for the current channel.
  precache          Populate the Flutter tool's cache of binary artifacts.
  upgrade           Upgrade your copy of Flutter.

Project
  analyze           Analyze the project's Dart code.
  assemble          Assemble and build Flutter resources.
  build             Build an executable app or install bundle.
  clean             Delete the build/ and .dart_tool/ directories.
  create            Create a new Flutter project.
  drive             Run integration tests for the project on an attached device or emulator.
  format            Format one or more Dart files.
  gen-l10n          Generate localizations for the current project.
  pub               Commands for managing Flutter packages.
  run               Run your Flutter app on an attached device.
  test              Run Flutter unit tests for the current project.

Tools & Devices
  attach            Attach to a running app.
  custom-devices    List, reset, add and delete custom devices.
  devices           List all connected devices.
```

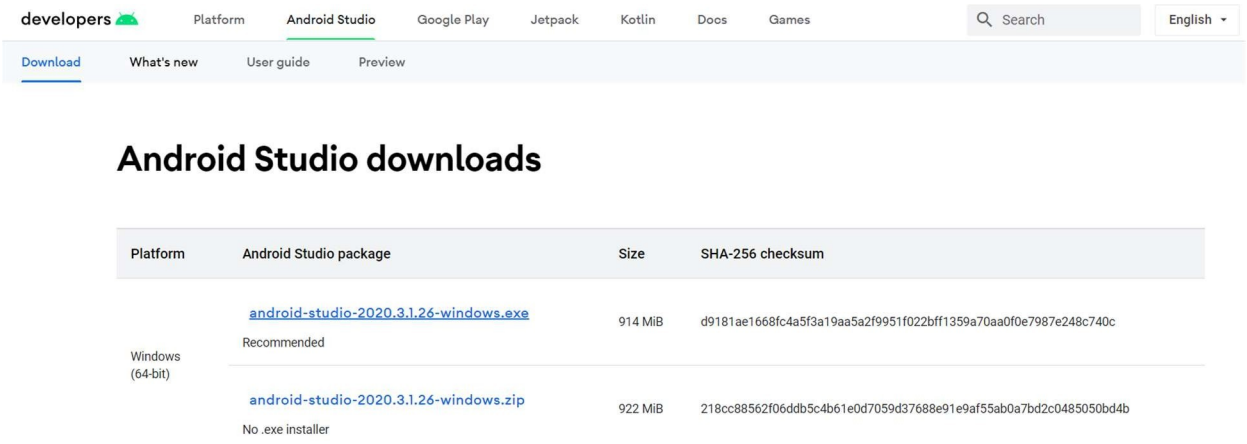
Now, run the **$ flutter doctor** command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation.

```
C:/■/system32
19:40:57 ❯ flutter doctor
Running "flutter pub get" in flutter_tools...                      12.3s
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.8.1, on Microsoft Windows [Version 10.0.19044.1466], locale en-IN)
[x] Android toolchain - develop for Android devices
    ✗ Unable to locate Android SDK.
      Install Android Studio from: https://developer.android.com/studio/index.html
      On first launch it will assist you in installing the Android SDK components.
      (or visit https://flutter.dev/docs/get-started/install/windows#android-setup for detailed instructions).
      If the Android SDK has been installed to a custom location, please use
      `flutter config --android-sdk` to update to that location.

[√] Chrome - develop for the web
[!] Android Studio (not installed)
[√] VS Code, 64-bit edition (version 1.63.2)
[√] Connected device (2 available)

! Doctor found issues in 2 categories.
```

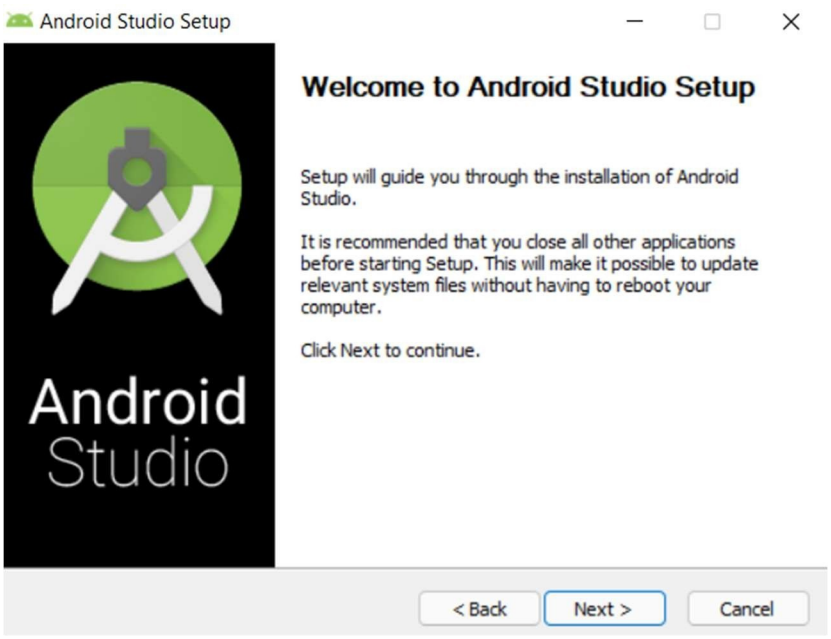
6. When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the **details of all missing tools**, which are required to run Flutter as well as the **development tools that are available** but not connected with the device.

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:

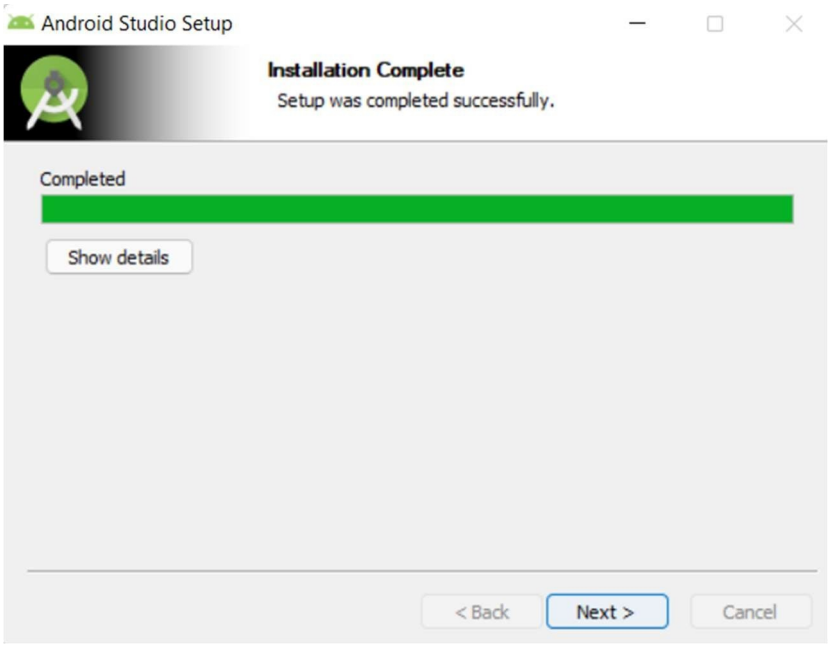
   a. Download the latest Android Studio executable or zip file from the official site.



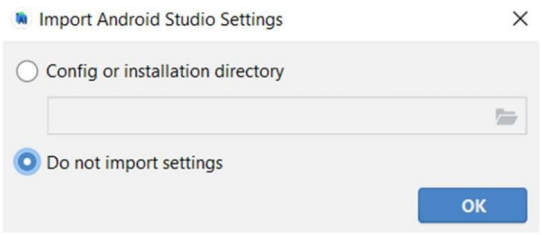
   b. When the download is complete, open the .exe file and run it. You will get the following dialog box.

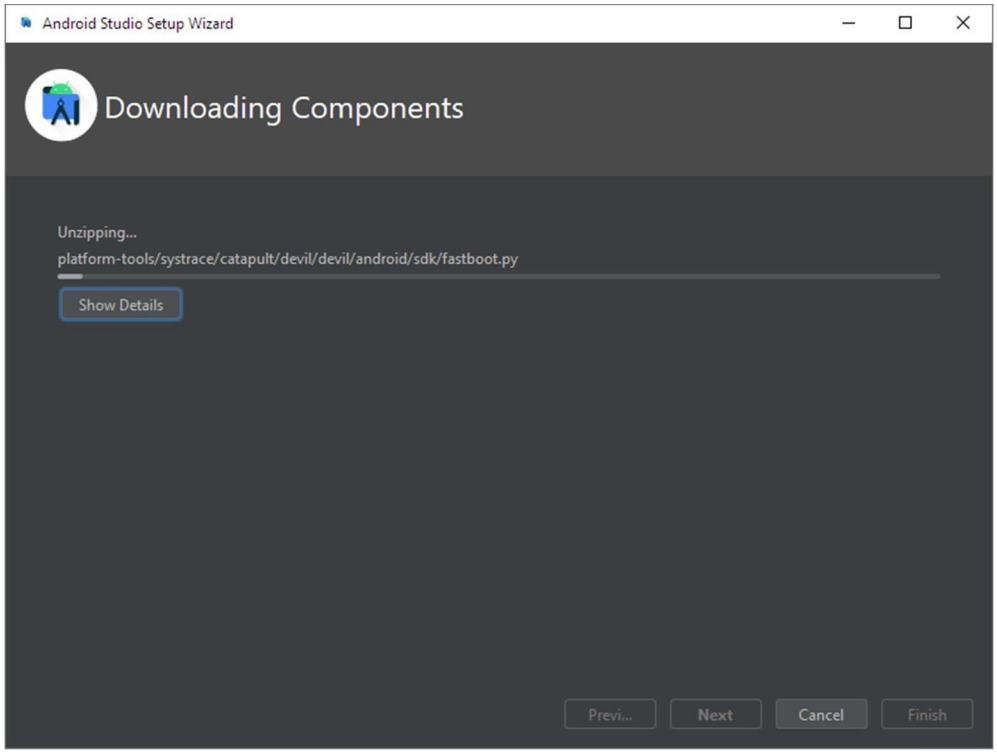


   c. Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



d. 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.



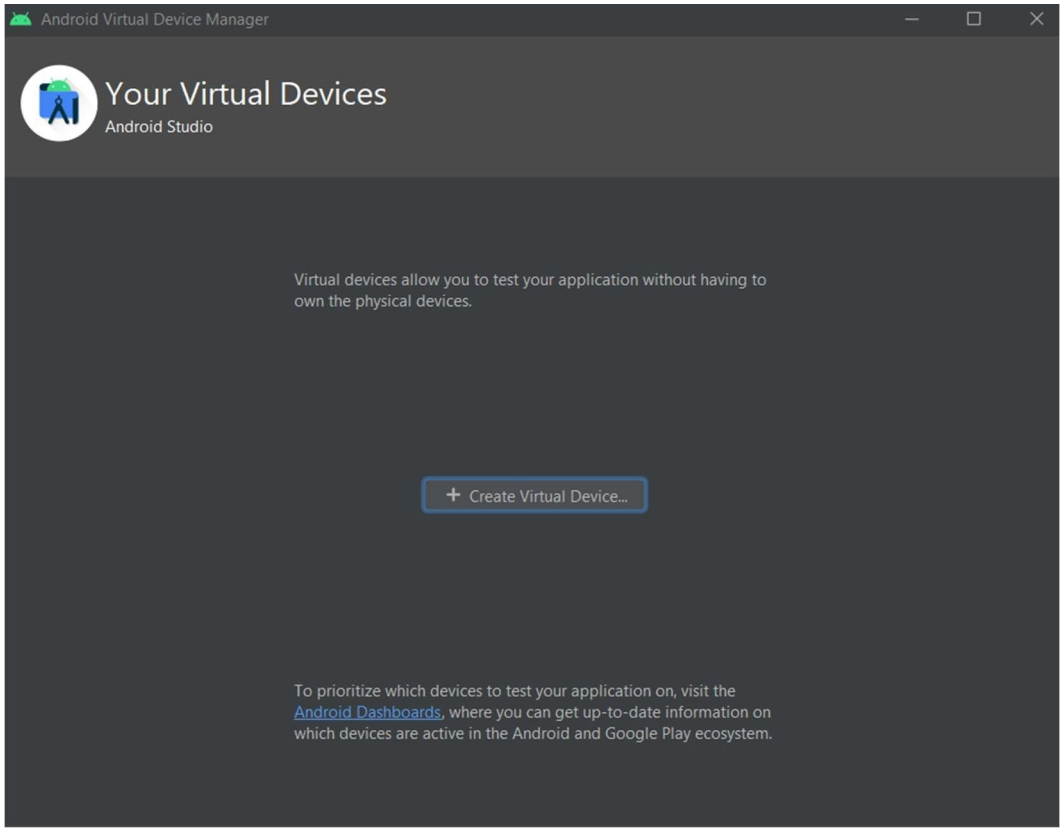


e. Run the **$ flutter doctor** command and Run **flutter doctor --android-licenses** command.

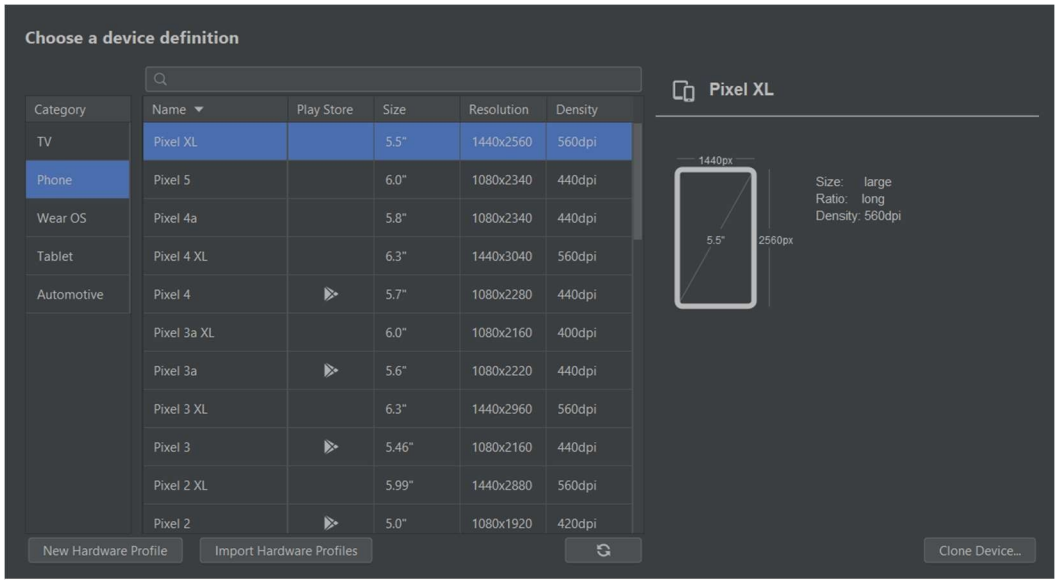
```
C:\Users\himan>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.8.1, on Microsoft Windows [Version 10.0.19043.1466], locale en-US)
[√] Android toolchain - develop for Android devices (Android SDK version 32.0.0)
[√] Chrome - develop for the web
[√] Android Studio (version 4.1)
[√] VS Code (version 1.63.0)
[√] Connected device (2 available)

• No issues found!
```

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

a. 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.

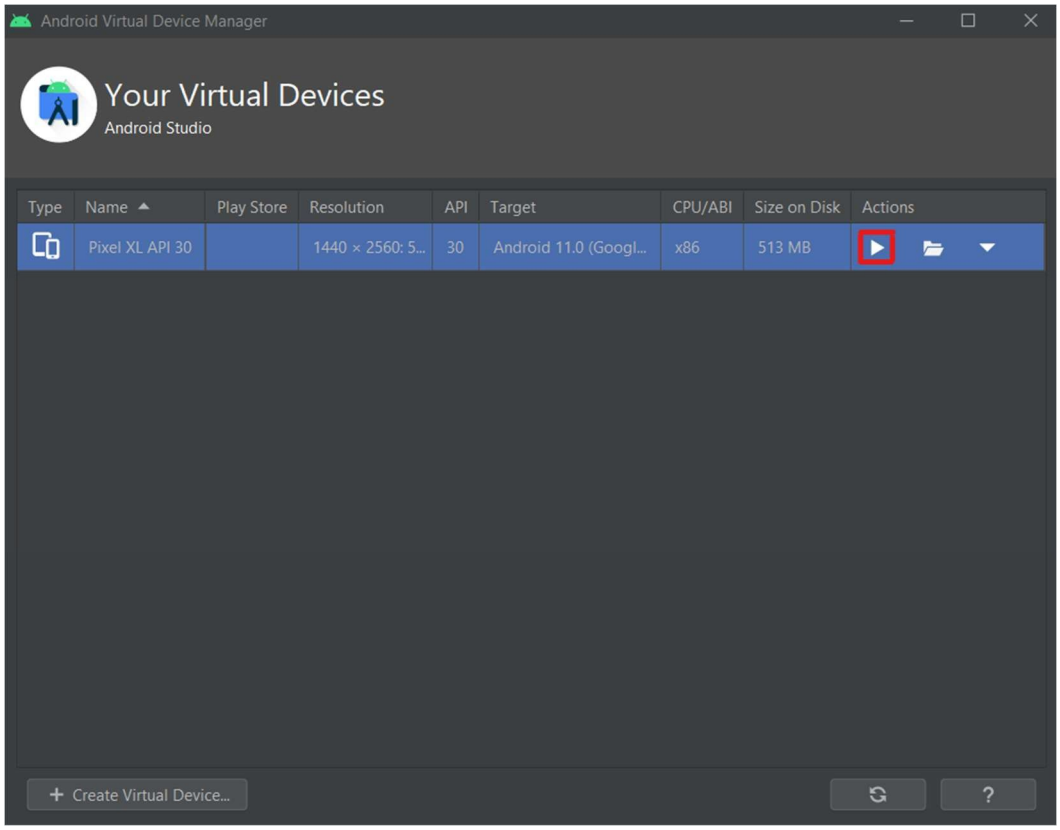


b. Choose your device definition and click on **Next**.

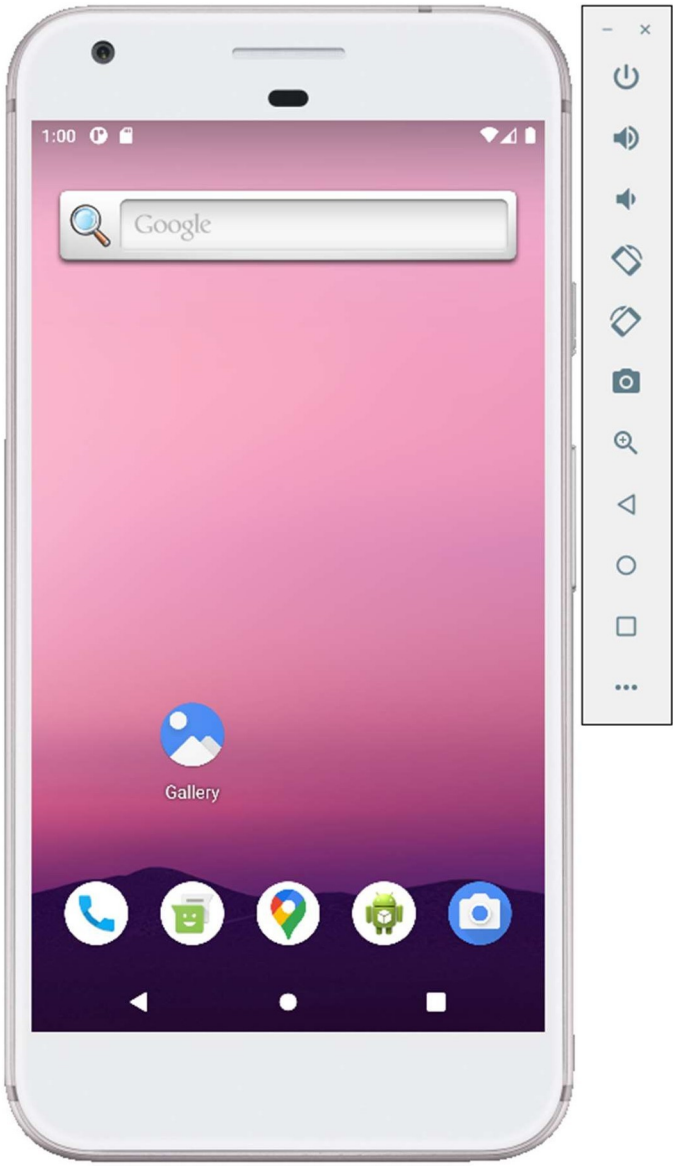


c. Select the system image for the latest Android version and click on **Next**.

d. Now, verify the all AVD configuration. If it is correct, click on **Finish**. The following screen appears.



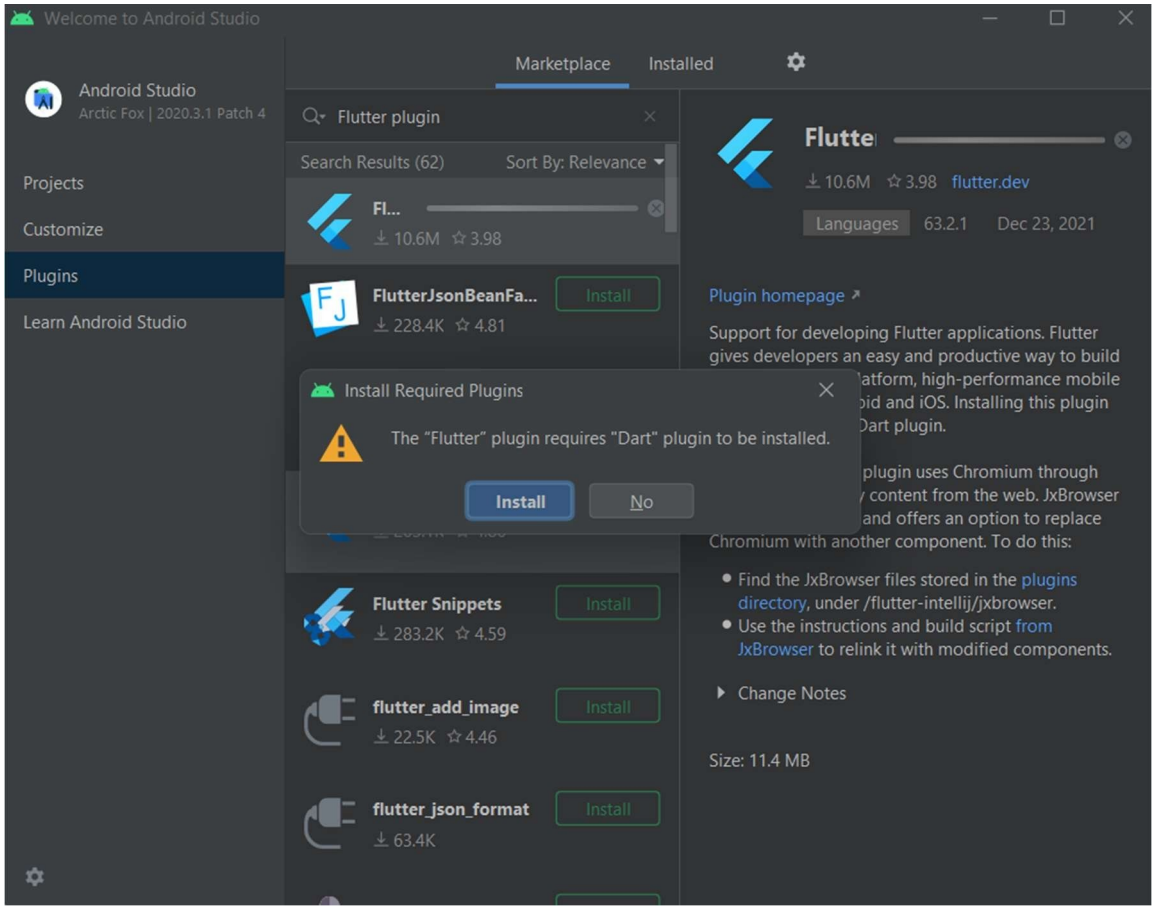
e. Last, click on the icon pointed into the **red color rectangle**. The Android emulator displayed as shown below screen.



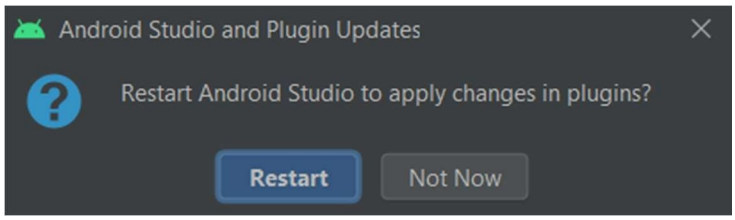
9. Now, install the **Flutter** and **Dart plugin** for building Flutter applications in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.

    a. Open the Android Studio and then go to **File** -> **Settings** -> **Plugins**.

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



c. Restart the Android Studio.



Conclusion:

Hence, we understood how to install and configure the Flutter environment by installing the Flutter SDK, installing and setting up Android Studio and in the end creating and adding a virtual device to the Android Studio.

Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

MAD & PWA Lab Journal

Experiment No.	02
Experiment	To design flutter UI by including common widget.
Roll No.	45
Name	Pushkar Mavale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	
Grade:	

Name: Pushkar Mavale **Class/Roll No**. D15A/45

MAD and PWA Lab

Experiment - 02

To design Flutter UI by including common widgets.

Aim: To design Flutter UI using common widgets.

Theory:

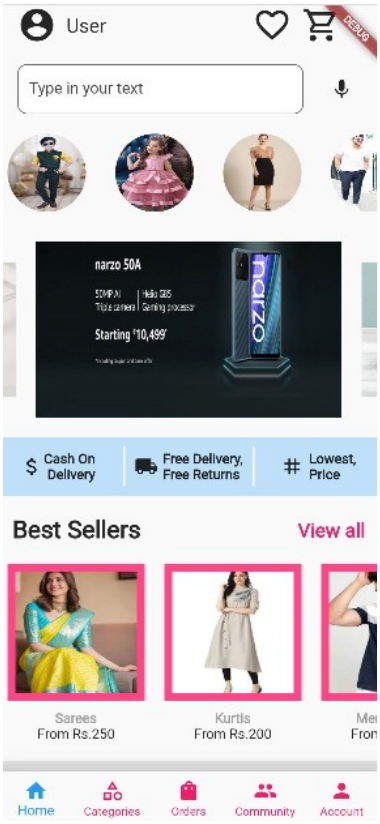
Widgets

Flutter widgets are built using a modern framework that takes inspiration from React. The central idea is that you build your UI out of widgets. Widgets describe what their view should look like given their current configuration and state. When a widget's state changes, the widget rebuilds its description, which the framework diffs against the previous description in order to determine the minimal changes needed in the underlying render tree to transition from one state to the next.

When you made any alteration in the code, the widget rebuilds its description by calculating the difference between the previous and current widget to determine the minimal changes for rendering in the UI of the app.

Widgets are nested with each other to build the app. It means the root of your app is itself a widget, and all the way down is a widget also. For example, a widget can display something, can define design, can handle interaction, etc.

The Screen:



Widgets Used:

1) BottomNavigationBar:

It is a material widget that's displayed at the bottom of an app for selecting among a small number of views, typically between three and five.

The bottom navigation bar consists of multiple items in the form of text labels, icons, or both, laid out on top of a piece of material. It provides quick navigation between the top-level views of an app. For larger screens, side navigation may be a better fit.

Syntax:

```
BottomNavigationBar(
    items: [
      BottomNavigationBarItem(icon:Icon(Icons.home),label:"Home"),
      BottomNavigationBarItem(icon:Icon(Icons.category_outlined),label:"Categories"),
      BottomNavigationBarItem(icon:Icon(Icons.shopping_bag),label: "Orders"),
      BottomNavigationBarItem(icon:Icon(Icons.people),label: "Community"),
      BottomNavigationBarItem(icon:Icon(Icons.person),label: "Account"),
    ],
    unselectedItemColor: Colors.pink,
    type: BottomNavigationBarType.fixed,
)
```

2) SafeArea:

It is a widget that insets its child with sufficient padding to avoid intrusions by the operating system. It comes in handy when there is borderline content that can be masked by the notch or the chin. It will also indent the child by the amount necessary to avoid The Notch on the iPhone X, or other similar creative physical features of the display.

Syntax:

```
SafeArea(
        child: <Widget>
      ),
```

3) TextField:

A material design text field.

A text field lets the user enter text, either with a hardware keyboard or with an on-screen keyboard.

The text field calls the onChanged callback whenever the user changes the text in the field. If the user indicates that they are done typing in the field (e.g., by pressing a button on the soft keyboard), the text field calls the onSubmitted callback.

Syntax:

```
         TextField(
           decoration: InputDecoration(
           border: OutlineInputBorder(
                       borderRadius: BorderRadius.circular(10.0),
                    ),
           filled: true,
           hintStyle: TextStyle(color: Colors.grey[800]),
           hintText: "Type in your text",
           fillColor: Colors.white70),
        )
```

4) ListView:

ListView is the most commonly used scrolling widget. It displays its children one after another in the scroll direction. In the cross axis, the children are required to fill the ListView.

If non-null, the itemExtent forces the children to have the given extent in the scroll direction.

If non-null, the prototypeItem forces the children to have the same extent as the given widget in the scroll direction.

Specifying an itemExtent or a prototypeItem is more efficient than letting the children determine their own extent because the scrolling machinery can make use of the foreknowledge of the children's extent to save work, for example when the scroll position changes drastically.

Syntax:

```
ListView(
  padding: const EdgeInsets.all(8),
  children: <Widget>[
    Container(




     height: 50,
      color: Colors.amber[600],
      child: const Center(child: Text('Entry A')),
    ),
    Container( he
      ight: 50,
      color: Colors.amber[500],
      child: const Center(child: Text('Entry B')),
    ),
  ],
),
```

5) Wrap:

A widget that displays its children in multiple horizontal or vertical runs.

A Wrap lays out each child and attempts to place the child adjacent to the previous child in the main axis, given by direction, leaving spacing space in between. If there is not enough space to fit the child, Wrap creates a new run adjacent to the existing children in the cross axis.

After all the children have been allocated to runs, the children within the runs are positioned according to the alignment in the main axis and according to the crossAxisAlignment in the cross axis.

Syntax:

```
Wrap(
  spacing: 8.0, // gap between adjacent chips
  runSpacing: 4.0, // gap between lines
  children: <Widget>[
    Chip(
```

```
      avatar: CircleAvatar(backgroundColor: Colors.blue.shade900, child:
 const Text('AH')),
      label: const Text('Hamilton'),
    ),
    Chip(
      avatar: CircleAvatar(backgroundColor: Colors.blue.shade900, child:
 const Text('ML')),
      label: const Text('Lafayette'),
    ),
  ],
 ),
```

Full Code:

```
import 'dart:developer';
import 'dart:html';

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:mdi/mdi.dart';

import 'package:carousel_slider/carousel_slider.dart';

void main() {
  runApp(MaterialApp(
    home: Home()
  ),
  );
}

class Home extends StatelessWidget {
  // const Home({Key? key}) : super(key: key);
  List<String>
list1=['boy.jpg','child.jpg','formal.jpg','men.jpg','women.jpg'];
  List<String> pro=['co.jpg','fur.jpg','books.jpg','foot.jpg'];
  List<Best> be=[Best("Sarees",250,"saree.jpg"),Best("Kurtis",
200,"kurti.jpg"),Best("Mens wear", 149,"men_wear.jpg"),Best("Kitchen", 30,
"kitchen.jpg")];
  Widget im(String s){
    return  Container(
            margin: EdgeInsets.only(top: 20,right: 20,left: 10),
            height: 80,
            width: 80,
            // color: Colors.blue,

            decoration: BoxDecoration(
              image: DecorationImage(
                image: AssetImage(s),
                fit: BoxFit.fill,
              ),
              // color: Colors.red,
```

```
            shape: BoxShape.circle,
            // borderRadius: BorderRadius.circular),
          ),
  );
}
Widget Product(String s){
  return Container(
          decoration: BoxDecoration(
            image: DecorationImage(
                image:AssetImage(s),
                fit: BoxFit.fill,
            )
          ),
          // color: Colors.red,
        );
}
Widget bestSellers(Best s){
  return  Column(
    children: [
      Container(
          margin: EdgeInsets.only(top: 20,right: 10,left: 10),
          height: 140,
          width: 140,
          // color: Colors.blue,

          decoration: BoxDecoration(
            image: DecorationImage(
              image: AssetImage(s.image),
              fit: BoxFit.fill,
            ),
            border: Border.all(
                color: Colors.pinkAccent,
                width: 8,
            )
            // color: Colors.red,
            // borderRadius: BorderRadius.circular),
          ),
        ),
        SizedBox(
          height: 10,
        ),
        Container(
          child:Column(
                children: [
                  Text(
                    s.name,
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Colors.grey,
                    ),
                  ),
                  Text(
                    "From Rs.${s.price}",
                  )
          ]
          )
```

```
        )
      ]
  );
}
@override
Widget build(BuildContext context) {
  return Scaffold(
    body:
    Column(
      children: <Widget>[
        Row(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: <Widget>[
            Expanded(
              flex: 1,
              child: Row(
                children: <Widget>[
                  Container(
                      child: Icon(
                        Icons.account_circle_sharp,
                        size: 40,
                      ),
                      alignment: Alignment.centerLeft,
                      margin: new EdgeInsets.fromLTRB(20, 0, 10, 0)
                  ),
                  Container(
                    child: Text(
                      "User",
                      style: TextStyle(fontSize: 20),
                    ),
                  ),
                ],
              ),
            ),
            Expanded(
              flex: 1,
              child: Container(
                width: double.infinity,
              ),
            ),
            Expanded(
              flex: 1,
              child: Container(
                child: Row(
                  children: <Widget>[
                    Container(
                      child: Icon(
                        Icons.favorite_border_outlined,
                        size: 40,
                      ),
                      margin: new EdgeInsets.only(right: 10),
                    ),
                    Container(
                      child: Icon(
                        Icons.shopping_cart_outlined,
                        size: 40,
                      ),
```

```
            )
          ],
        ),
      ),
    ),
  ],
),
Container(
  height: 20,
),
Container(
  child: Row(
    children: [
      Expanded(
        flex: 4,
        child: Container(
          child: TextField(
            decoration: InputDecoration(
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(10.0),
                ),
                filled: true,
                hintStyle: TextStyle(color: Colors.grey[800]),
                hintText: "Type in your text",
                fillColor: Colors.white70),
          ),
          margin: new EdgeInsets.only(left: 20),
        ),
      ),
      Expanded(
        flex: 1,
        child: Container(
          child: Icon(
            Icons.mic,
          ),
        ),

      ),

    ],
  ),
),
SingleChildScrollView(
  scrollDirection: Axis.horizontal,
  child: Row(
    children: list1.map((e) => im(e)).toList(),
  ),
),
Container(
  child: CarouselSlider(
    items: pro.map((e) => Product(e)).toList(),
    options: CarouselOptions(
      height: 180.0,
      enlargeCenterPage: true,
      autoPlay: true,
      aspectRatio: 16 / 9,
      autoPlayCurve: Curves.fastOutSlowIn,
```

```
      enableInfiniteScroll: true,
      autoPlayAnimationDuration: Duration(milliseconds: 300),
      viewportFraction: 0.8,
    ),
  ),
  margin: EdgeInsets.only(top: 30),
),
SizedBox(
  height: 20,
),
Container(
  child: Row(
    children: [
      Expanded(
        flex: 1,
          child: Container(
              height: 60,
              child: Row(
                children: [
                    Icon(
                      Icons.attach_money_outlined
                    ),
                    Text(
                      "Cash On \n Delivery",
                      style: TextStyle(
                        fontWeight: FontWeight.bold,
                      ),
                    )
                ],
                mainAxisAlignment: MainAxisAlignment.center,
              ),
            color: Colors.blue[100],
          )
      ),
      Container(
        width: 2,
        height: 40,
        color: Colors.white,
        margin: EdgeInsets.fromLTRB(2, 0, 2, 0),
      ),
      Expanded(
          flex: 1,
          child: Container(
            height: 60,
            child: Row(
              children: [
                Container(
                  child: Icon(
                      Icons.local_shipping,
                  ),
                  margin: EdgeInsets.only(right: 5),
                ),
                Text(
                  "Free Delivery,\nFree Returns",
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                  ),
```

```
                ),
              ],
              mainAxisAlignment: MainAxisAlignment.center,
            ),
            color: Colors.blue[100],
          )
      ),
      Container(
        width: 2,
        height: 40,
        color: Colors.white,
        margin: EdgeInsets.fromLTRB(2, 0, 2, 0),
      ),
      Expanded(
          flex: 1,
          child: Container(
            height: 60,
            child: Row(
              children: [
                Container(
                  child: Icon(
                    Icons.tag_sharp,
                  ),
                  margin: EdgeInsets.only(right: 5),
                ),
                Text(
                  "Lowest,\nPrice",
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ],
              mainAxisAlignment: MainAxisAlignment.center,
            ),
            color: Colors.blue[100],
          )
      ),
    ],
  ),
  color: Colors.blue[100],
),
SizedBox(
  height: 20,
),
Container(
  child: Row(
    children: [
        Expanded(
            flex:1,
            child: Text(
                "Best Sellers",
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                  fontSize: 25,
                ),
            )
        ),
```

```
                  Expanded(
                      flex:1,
                      child: Text(
                          "View all",
                          style: TextStyle(
                            fontWeight: FontWeight.bold,
                            fontSize: 20,
                            color: Colors.pink,
                          ),
                        textAlign:TextAlign.end,
                      )
                  ),
                ],
              ),
              margin: EdgeInsets.fromLTRB(15, 0, 15, 0)
            ),
            SingleChildScrollView(
              scrollDirection: Axis.horizontal,
              child: Row(
                children: be.map((e) => bestSellers(e)).toList(),
              ),
            ),
            SizedBox(
              height: 20,
            ),
            Container(
              color: Colors.grey[300],
              height: 10,
            ),
            BottomNavigationBar(
              items: [
                BottomNavigationBarItem(icon:Icon(Icons.home),label:"Home"),

BottomNavigationBarItem(icon:Icon(Icons.category_outlined),label:"Categories"),
                BottomNavigationBarItem(icon:Icon(Icons.shopping_bag),label:
"Orders"),
                BottomNavigationBarItem(icon:Icon(Icons.people),label:
"Community"),
                BottomNavigationBarItem(icon:Icon(Icons.person),label:
"Account"),
              ],
              unselectedItemColor: Colors.pink,
              type: BottomNavigationBarType.fixed,
            )
          ],
        ),
      );
  }
}


class Best{
    late String name;
    late int price;
    late String image;
    Best(String name,int price,String image){
      this.name=name;
```

```
      this.price=price;
      this.image=image;
    }
}
```

Conclusion:

Thus, we learned how to use Widgets to work with UI in Flutter.

Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

MAD & PWA Lab Journal

Experiment No.	03
Experiment	To int Title.
Roll No.	45
Name	Pushkar Mavale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	
Grade:	

MAD and PWA

Lab Experiment

No: 03

Images, Icons and Fonts in Flutter

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

Theory:

Adding assets and images

Flutter apps can include both code and assets (sometimes called resources). An asset is a file that is bundled and deployed with your app and is accessible at runtime. Common types of assets include static data (for example, JSON files), configuration files, icons, and images (JPEG, WebP, GIF, animated WebP/GIF, PNG, BMP, and WBMP).

Specifying assets

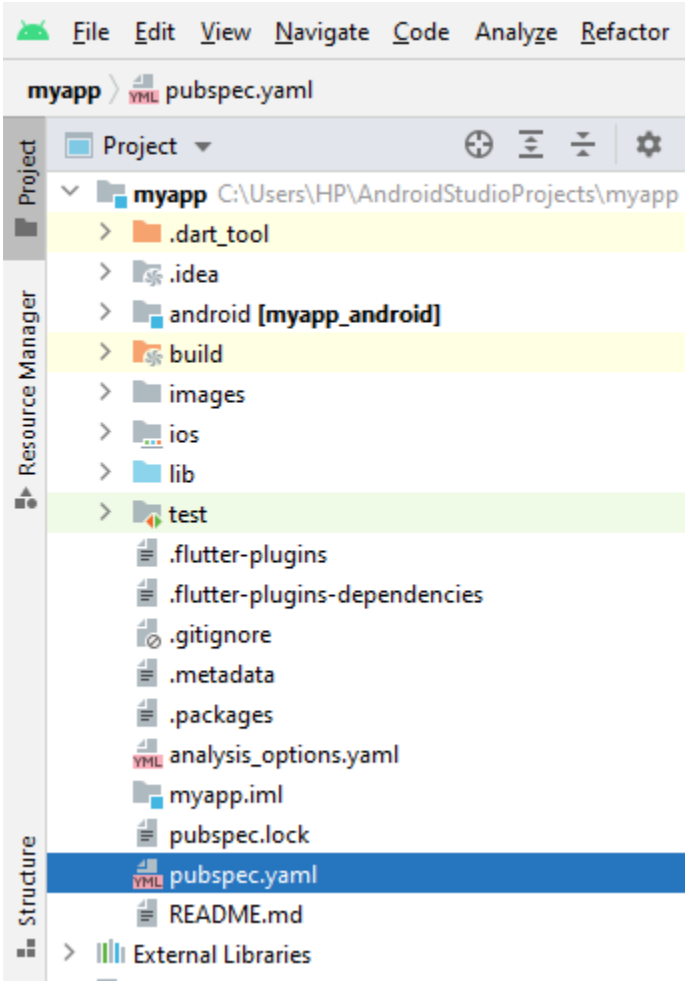
Flutter uses the pubspec.yaml file, located at the root of your project, to identify assets required by an app.

Here is an example:

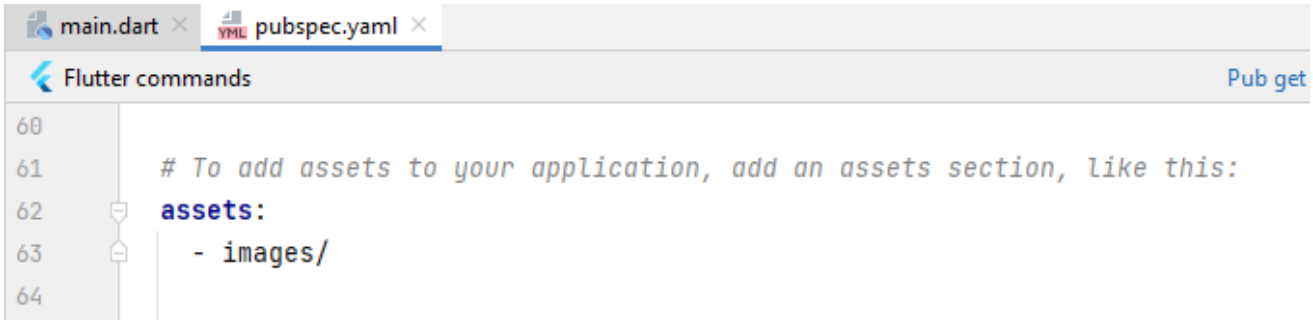
```
flutter:
  assets:
    - directory/
    - directory/subdirectory/
```

Steps:

1. Open your Flutter Project. Open the pubspec.yaml file.

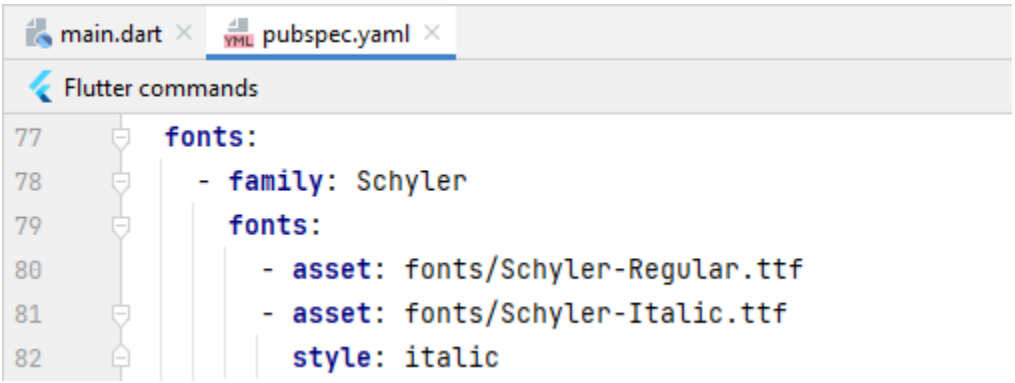


2. Under the flutter section, create a new section called assets. Under this section, include the directories in which you're storing the images and fonts.



3. Download your fonts' .ttf files from Google Fonts or DaFont.

4. In pubspec.yaml, right under assets, create another section for fonts. Similarly, list all the ttf files which are in use.



5. Use these assets in your code. Fonts can be used as strings.
   Eg. fontFamily: 'Schyler',

   Images can be loaded as AssetImage.
   Eg. AssetImage('images/image1.png')

Full Code:

```
import 'dart:developer';
import 'dart:html';

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:mdi/mdi.dart';

import 'package:carousel_slider/carousel_slider.dart';

void main() {
  runApp(MaterialApp(
    home: Home()
  ),
  );
}

class Home extends StatelessWidget {
  // const Home({Key? key}) : super(key: key);
  List<String>
list1=['boy.jpg','child.jpg','formal.jpg','men.jpg','women.jpg'];
  List<String> pro=['co.jpg','fur.jpg','books.jpg','foot.jpg'];
  List<Best> be=[Best("Sarees",250,"saree.jpg"),Best("Kurtis",
200,"kurti.jpg"),Best("Mens wear", 149,"men_wear.jpg"),Best("Kitchen", 30,
"kitchen.jpg")];
  Widget im(String s){
    return  Container(
            margin: EdgeInsets.only(top: 20,right: 20,left: 10),
            height: 80,
            width: 80,
```

```
          // color: Colors.blue,

          decoration: BoxDecoration(
            image: DecorationImage(
              image: AssetImage(s),
              fit: BoxFit.fill,
            ),
            // color: Colors.red,
            shape: BoxShape.circle,
            // borderRadius: BorderRadius.circular),
          ),
  );
}
Widget Product(String s){
  return Container(
          decoration: BoxDecoration(
            image: DecorationImage(
                image:AssetImage(s),
                fit: BoxFit.fill,
            )
          ),
          // color: Colors.red,
        );
}
Widget bestSellers(Best s){
  return  Column(
    children: [
      Container(
          margin: EdgeInsets.only(top: 20,right: 10,left: 10),
          height: 140,
          width: 140,
          // color: Colors.blue,

          decoration: BoxDecoration(
            image: DecorationImage(
              image: AssetImage(s.image),
              fit: BoxFit.fill,
            ),
            border: Border.all(
                color: Colors.pinkAccent,
                width: 8,
            )
            // color: Colors.red,
            // borderRadius: BorderRadius.circular),
          ),
        ),
        SizedBox(
          height: 10,
        ),
        Container(
          child:Column(
                children: [
                  Text(
                    s.name,
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Colors.grey,
```

```
                    ),
                  ),
                  Text(
                    "From Rs.${s.price}",
                  )
              ]
            )

          )
        ]
  );
}
@override
Widget build(BuildContext context) {
  return Scaffold(
    body:
    Column(
      children: <Widget>[
        Row(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: <Widget>[
            Expanded(
              flex: 1,
              child: Row(
                children: <Widget>[
                  Container(
                      child: Icon(
                        Icons.account_circle_sharp,
                        size: 40,
                      ),
                      alignment: Alignment.centerLeft,
                      margin: new EdgeInsets.fromLTRB(20, 0, 10, 0)
                  ),
                  Container(
                    child: Text(
                      "User",
                      style: TextStyle(fontSize: 20),
                    ),
                  ),
                ],
              ),
            ),
            Expanded(
              flex: 1,
              child: Container(
                width: double.infinity,
              ),
            ),
            Expanded(
              flex: 1,
              child: Container(
                child: Row(
                  children: <Widget>[
                    Container(
                      child: Icon(
                        Icons.favorite_border_outlined,
                        size: 40,
```

```
              ),
              margin: new EdgeInsets.only(right: 10),
            ),
            Container(
              child: Icon(
                Icons.shopping_cart_outlined,
                size: 40,
              ),
            )
          ],
        ),
      ),
    ),
  ],
),
Container(
  height: 20,
),
Container(
  child: Row(
    children: [
      Expanded(
        flex: 4,
        child: Container(
          child: TextField(
            decoration: InputDecoration(
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(10.0),
                ),
                filled: true,
                hintStyle: TextStyle(color: Colors.grey[800]),
                hintText: "Type in your text",
                fillColor: Colors.white70),
          ),
          margin: new EdgeInsets.only(left: 20),
        ),
      ),
      Expanded(
        flex: 1,
        child: Container(
          child: Icon(
            Icons.mic,
          ),
        ),

      ),

    ],
  ),
),
SingleChildScrollView(
  scrollDirection: Axis.horizontal,
  child: Row(
    children: list1.map((e) => im(e)).toList(),
  ),
),
Container(
```

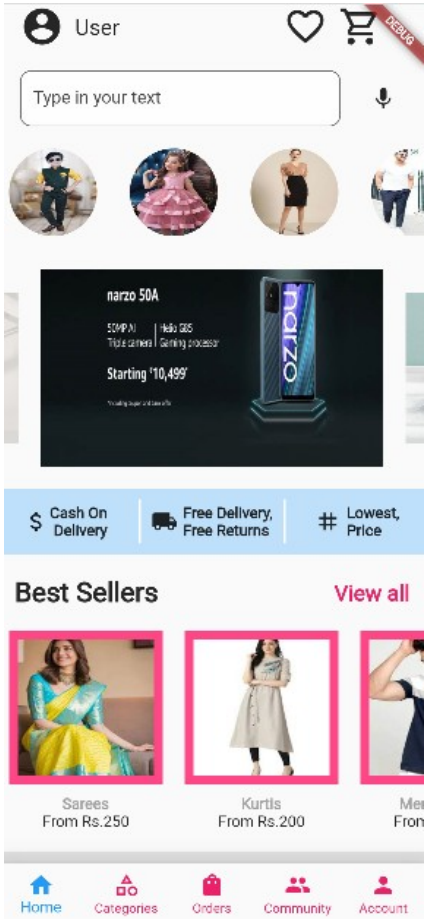
```
  child: CarouselSlider(
    items: pro.map((e) => Product(e)).toList(),
    options: CarouselOptions(
      height: 180.0,
      enlargeCenterPage: true,
      autoPlay: true,
      aspectRatio: 16 / 9,
      autoPlayCurve: Curves.fastOutSlowIn,
      enableInfiniteScroll: true,
      autoPlayAnimationDuration: Duration(milliseconds: 300),
      viewportFraction: 0.8,
    ),
  ),
  margin: EdgeInsets.only(top: 30),
),
SizedBox(
  height: 20,
),
Container(
  child: Row(
    children: [
      Expanded(
        flex: 1,
          child: Container(
              height: 60,
              child: Row(
                children: [
                    Icon(
                      Icons.attach_money_outlined
                    ),
                    Text(
                      "Cash On \n Delivery",
                      style: TextStyle(
                        fontWeight: FontWeight.bold,
                      ),
                    )
                ],
                mainAxisAlignment: MainAxisAlignment.center,
              ),
            color: Colors.blue[100],
          )
      ),
      Container(
        width: 2,
        height: 40,
        color: Colors.white,
        margin: EdgeInsets.fromLTRB(2, 0, 2, 0),
      ),
      Expanded(
          flex: 1,
          child: Container(
            height: 60,
            child: Row(
              children: [
                Container(
                  child: Icon(
                      Icons.local_shipping,
```

```
                ),
                margin: EdgeInsets.only(right: 5),
              ),
              Text(
                "Free Delivery,\nFree Returns",
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                ),
              ),
            ],
            mainAxisAlignment: MainAxisAlignment.center,
          ),
          color: Colors.blue[100],
        )
      ),
      Container(
        width: 2,
        height: 40,
        color: Colors.white,
        margin: EdgeInsets.fromLTRB(2, 0, 2, 0),
      ),
      Expanded(
          flex: 1,
          child: Container(
            height: 60,
            child: Row(
              children: [
                Container(
                  child: Icon(
                    Icons.tag_sharp,
                  ),
                  margin: EdgeInsets.only(right: 5),
                ),
                Text(
                  "Lowest,\nPrice",
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ],
              mainAxisAlignment: MainAxisAlignment.center,
            ),
            color: Colors.blue[100],
          )
      ),
    ],
  ),
  color: Colors.blue[100],
),
SizedBox(
  height: 20,
),
Container(
  child: Row(
    children: [
        Expanded(
           flex:1,
```

```
                    child: Text(
                        "Best Sellers",
                        style: TextStyle(
                          fontWeight: FontWeight.bold,
                          fontSize: 25,
                        ),
                    )
                ),
                Expanded(
                    flex:1,
                    child: Text(
                        "View all",
                        style: TextStyle(
                          fontWeight: FontWeight.bold,
                          fontSize: 20,
                          color: Colors.pink,
                        ),
                      textAlign:TextAlign.end,
                    )
                ),
              ],
            ),
            margin: EdgeInsets.fromLTRB(15, 0, 15, 0)
          ),
          SingleChildScrollView(
            scrollDirection: Axis.horizontal,
            child: Row(
              children: be.map((e) => bestSellers(e)).toList(),
            ),
          ),
          SizedBox(
            height: 20,
          ),
          Container(
            color: Colors.grey[300],
            height: 10,
          ),
          BottomNavigationBar(
            items: [
              BottomNavigationBarItem(icon:Icon(Icons.home),label:"Home"),

BottomNavigationBarItem(icon:Icon(Icons.category_outlined),label:"Categories")
,
              BottomNavigationBarItem(icon:Icon(Icons.shopping_bag),label:
"Orders"),
              BottomNavigationBarItem(icon:Icon(Icons.people),label:
"Community"),
              BottomNavigationBarItem(icon:Icon(Icons.person),label:
"Account"),
            ],
            unselectedItemColor: Colors.pink,
            type: BottomNavigationBarType.fixed,
          )
        ],
      ),
    );
  }
```

```
}


class Best{
    late String name;
    late int price;
    late String image;
    Best(String name,int price,String image){
      this.name=name;
      this.price=price;
      this.image=image;
    }
}
```

The Screen-



Conclusion:

Thus, we learned how to use Icons, Images and Fonts in our Flutter App.

Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

MAD & PWA Lab Journal

Experiment No.	04
Experiment Title.	Interactive forms in flutter
Roll No.	45
Name	Pushkar Mavale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	
Grade:	

Name: Pushkar Mavale **Class/Roll No.** D15A/45

MAD and PWA Lab

Experiment No: 04

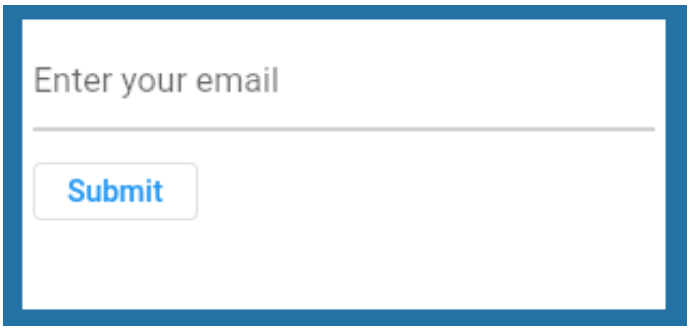
Interactive Forms in Flutter

Aim: To create an interactive Form using the Form widget

Theory:

Apps often require users to enter information into a text field. To make apps secure and easy to use, check whether the information the user has provided is valid. If the user has correctly filled out the form, process the information. If the user submits incorrect information, display a friendly error message letting them know what went wrong.

1) Form Widget



Form is an optional container for grouping together multiple form field widgets (e.g. TextField widgets). Each form field should be wrapped in a FormField widget, with the Form widget as a common ancestor of all of those. Call methods on FormState to save, reset, or validate each FormField that is a descendant of this Form. To obtain the FormState, you may use *Form.of* with a context whose ancestor is the Form, or pass a GlobalKey to the Form constructor and call GlobalKey.currentState.

Eg:

```
Form(
      key:  _formKey,
      child: TextFormField(
            decoration: const InputDecoration(
              hintText: 'Enter your email',
            ),
        ),
    );
```

2) FormField Widget

It is a single form field. It maintains the current state of the form field so that updates and validation errors are visually reflected in the UI.

When used inside a Form, you can use methods on FormState to query or manipulate the form data as a whole. For example, calling FormState.save will invoke each FormField's onSaved callback in turn.

3) TextFormField Widget

It is a FormField that contains a TextField. This is a convenience widget that wraps a TextField widget in a FormField.

A Form ancestor is not required. The Form simply makes it easier to save, reset, or validate multiple fields at once. To use without a Form, pass a GlobalKey to the constructor and use GlobalKey.currentState to save or reset the form field.

When a controller is specified, its TextEditingController.text defines the initialValue. If this FormField is part of a scrolling container that lazily constructs its children, like a ListView or a CustomScrollView, then a controller should be specified. The controller's lifetime should be managed by a stateful widget ancestor of the scrolling container.

Eg:

```
TextFormField(
  decoration: const InputDecoration(
    icon: Icon(Icons.person),
    hintText: 'What do people call you?',
    labelText: 'Name *',
  ),
  onSaved: (String? value) {
    // This optional block of code can be used to run
    // code when the user saves the form.
  },
  validator: (String? value) {
    return (value != null && value.contains('@')) ? 'Do not use the @
char.' : null;
  },
)
```

Full Code:

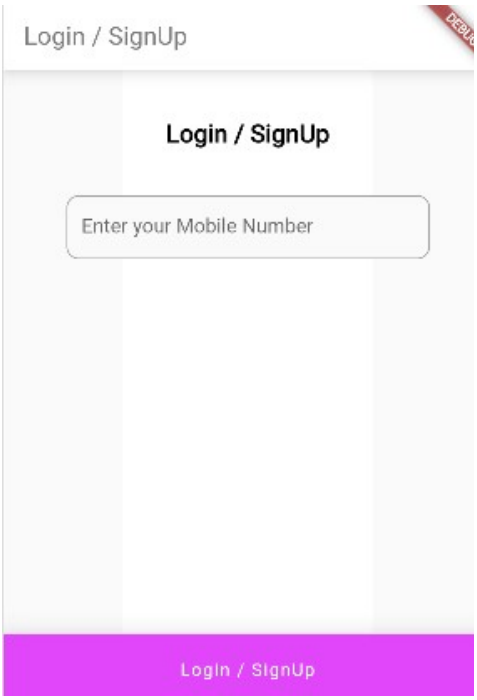
```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:fluttertoast/fluttertoast.dart';
import 'package:carousel_slider/carousel_slider.dart';
import 'dart:math' as math;

void main() {
  runApp(MaterialApp(
    home: Login(),
  ));
}


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

  @override
  _LoginState createState() => _LoginState();
}

class _LoginState extends State<Login> {

  final _formKey = GlobalKey<FormState>();

  final usernameController = TextEditingController();
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.white,
        title: Text('Login / SignUp',
          style: TextStyle(
            color: Colors.grey[600],
          ),
        ),
      ),
      body: Center(
          child: Form(
            key: _formKey,
            child: Column(
              children: <Widget>[
                Container(
                  child: Center(
                    child: Text(
                      'Login / SignUp',




                      style: TextStyle(
                        color: Colors.black,
                        fontWeight: FontWeight.bold,
                        fontSize: 20.0,
                      ),
                    ),
                ),
                color: Colors.white,
                height: 100,
                width: 200,
              ),
```

```
                Container(
                  margin: EdgeInsets.symmetric(horizontal: 50.0,vertical: 0.0),
                  child: TextFormField(

                    validator: (value) {
                      if (value == null || value.isEmpty) {
                        return 'Number Required';
                      }
                      return null;
                    },
                    controller: usernameController,
                    enableSuggestions: false,
                    autocorrect: false,
                    decoration: InputDecoration(
                      hintText: 'Enter your Mobile Number',
                      border: OutlineInputBorder(
                          borderRadius: BorderRadius.circular(10.0),
                          borderSide: const BorderSide(color: Colors.grey)),
                    ),
                  ),
                ),

                Container(
                  child: Center(
                    child: Text(
                      '',
                      style: TextStyle(
                        color: Colors.black,
                        fontWeight: FontWeight.bold,
                        fontSize: 20.0,
                      ),
                    ),
                  ),
                  color: Colors.white,
                  height: 300,




                  width: 200,
                ),

                Row(
                  crossAxisAlignment: CrossAxisAlignment.center,
                  children: [
                    Expanded(
                      child: FloatingActionButton(
                        backgroundColor: Colors.purpleAccent,
                        onPressed: () {
                          FocusManager.instance.primaryFocus?.unfocus();
                          if (_formKey.currentState!.validate()) {
                            if (usernameController.text != '1234567890') {
                              Fluttertoast.showToast(
                                msg: "Invalid Mobile Number",
                                toastLength: Toast.LENGTH_SHORT,
                                gravity: ToastGravity.CENTER,
                                timeInSecForIosWeb: 1,
                              );
                              return;
                            }
                            else{
                              ScaffoldMessenger.of(context).showSnackBar(
                                const SnackBar(content: Text('Signing you
in..')),
```

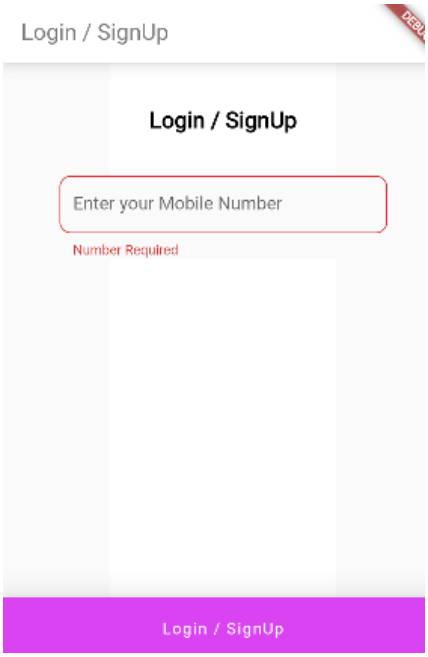
```
                                );
                                // Navigator.push(
                                //     context,
                                //     MaterialPageRoute(builder: (context) =>
SearchScreen())
                                // );
                              }
                            }
                            else {

                            }
                          },
                          child: Text('Login / SignUp'),
                          shape: BeveledRectangleBorder(
                            borderRadius: BorderRadius.zero,
                          ),
                        ),
                      ),
                    ],
                  ),

                ],
              ),
            )),




    );
  }
}
```

Output-

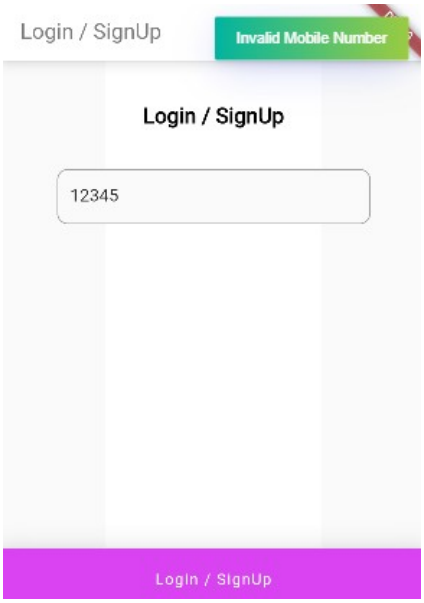
1) Login Screen:



On providing No Input:



On providing Invalid Credentials:



Conclusion:

Thus, we learned how to use Form Widget in Flutter and use FormFields with the validator to create interactive forms in Flutter.

Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

MAD & PWA Lab Journal

Experiment No.	05
Experiment Title.	To apply navigation,routing and gestures
Roll No.	45
Name	Pushkar Mavale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	
Grade:	

Name: Pushkar Mavale **Division / Roll No.** D15A / 45

MAD and PWA Lab

Experiment No: 05

Meesho App

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

Theory:

Navigation and routing are some of the core concepts of all mobile applications, which allows the user to move between different pages. We know that every mobile application contains several screens for displaying different types of information. For example, an app can have a screen that contains various products. When the user taps on that product, immediately it will display detailed information about that product.

Gestures are an interesting feature in Flutter that allows us to interact with the mobile app (or any touch-based device). Generally, gestures define any physical action or movement of a user in the intention of specific control of the mobile device.

Navigation and Routing:

Most apps contain several screens for displaying different types of information. For example, an app might have a screen that displays products. When the user taps the image of a product, a new screen displays details about the product.

The next few sections show how to navigate between two routes, using these steps:

1. **Create two routes.**

   First, create two routes to work with. Since this is a basic example, each route contains only a single button. Tapping the button on the first route navigates to the second route. Tapping the button on the second route returns to the first route.

2. **Navigate to the second route using Navigator.push().**

   To switch to a new route, use the Navigator.push() method. The push() method adds a Route to the stack of routes managed by the Navigator. Where does the Route come from? You can create your own, or use a MaterialPageRoute, which is useful because it transitions to the new route using a platform-specific animation. In the build() method of the FirstRoute widget, update the onPressed() callback

```
onPressed: () {
  Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => const
SecondRoute()),
  );
}
```

3. **Return to the first route using Navigator.pop().**

   How do you close the second route and return to the first? By using the Navigator.pop() method. The pop() method removes the current Route from the stack of routes managed by the Navigator.

   To implement a return to the original route, update the onPressed() callback in the SecondRoute widget:

```
onPressed: () {
  Navigator.pop(context);
}
```

Properties:

Name	Data type	Description
initialRoute	String?	The name of the first route to show
observers	List<NavigatorObserv	A list of observers for this

	er>	navigator.
onGenerateInitialRo utes	RouteListFactory	Called when the widget is created to generate the initial list of Route objects if initialRoute is not null.
onGenerateRoute	RouteFactory?	Called to generate a route for a given RouteSettings.
onPopPage	PopPageCallback?	Called when pop is invoked but the current Route corresponds to a Page found in the pages list.
pages	List<Page>	The list of pages with which to populate the history.
requestFocus	bool	Whether or not the navigator and it's new topmost route should request focus when the new route is pushed onto the navigator.
transitionDelegate	TransitionDelegate	The delegate used for deciding how routes transition in or off the screen during the pages updates.

Gestures:

It is the second layer that represents semantic actions such as tap, drag, and scale, which are recognized from multiple individual pointer events. It is also able to dispatch multiple events corresponding to gesture lifecycle like drag start, drag update, and drag end. Some of the popularly used gesture are listed below:

Tap: It means touching the surface of the screen from the fingertip for a short time and then releasing them. This gesture contains the following events:

1. onTapDown
2. onTapUp
3. onTap
4. onTapCancel

Double Tap: It is similar to a Tap gesture, but you need to tap twice in a short time. This gesture contains the following events:

1. onDoubleTap

Drag: It allows us to touch the surface of the screen with a fingertip and move it from one location to another location and then release them. Flutter categories the drag into two types:

1. Horizontal Drag: This gesture allows the pointer to move in a horizontal direction. It contains the following events:

    1. onHorizontalDragStart
    2. onHorizontalDragUpdate
    3. onHorizontalDragEnd

2. Vertical Drag: This gesture allows the pointer to move in a vertical direction. It contains the following events:
    1. onVerticalDragStart
    2. onVerticalDragStart
    3. onVerticalDragStart

Long Press: It means touching the surface of the screen at a particular location for a long time. This gesture contains the following events:

1. onLongPress

Pan: It means touching the surface of the screen with a fingertip, which can move in any direction without releasing the fingertip. This gesture contains the following events:

1. onPanStart
2. onPanUpdate
3. onPanEnd

Full Code:

```
import 'dart:developer';
import 'dart:html';

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:meesho/pages/account.dart';
import 'package:mdi/mdi.dart';

import 'package:carousel_slider/carousel_slider.dart';

void main() {
  runApp(MaterialApp(
    home: Home(),
    routes: {
      '/account':(context)=> Account(),
    },
  ),
  );
}

class Home extends StatelessWidget {
  // const Home({Key? key}) : super(key: key);
  List<String> list1=['boy.jpg','child.jpg','formal.jpg','men.jpg','women.jpg'];
  List<String> pro=['co.jpg','fur.jpg','books.jpg','foot.jpg'];
  List<Best> be=[Best("Sarees",250,"saree.jpg"),Best("Kurtis", 200,"kurti.jpg"),Best("Mens
wear", 149,"men_wear.jpg"),Best("Kitchen", 30, "kitchen.jpg")];
  Widget im(String s){
    return  Container(
          margin: EdgeInsets.only(top: 20,right: 20,left: 10),
          height: 80,
          width: 80,
          // color: Colors.blue,

          decoration: BoxDecoration(
            image: DecorationImage(
              image: AssetImage(s),
              fit: BoxFit.fill,
            ),
            // color: Colors.red,
            shape: BoxShape.circle,
            // borderRadius: BorderRadius.circular),
          ),
    );
  }
  Widget Product(String s){
    return Container(
          decoration: BoxDecoration(
            image: DecorationImage(
                image:AssetImage(s),
```

```
            fit: BoxFit.fill,
          )
        ),
        // color: Colors.red,
      );
  }
  Widget bestSellers(Best s){
    return  Column(
     children: [
       Container(
         margin: EdgeInsets.only(top: 20,right: 10,left: 10),
         height: 140,
         width: 140,
         // color: Colors.blue,

         decoration: BoxDecoration(
           image: DecorationImage(
             image: AssetImage(s.image),
             fit: BoxFit.fill,
           ),
           border: Border.all(
             color: Colors.pinkAccent,
             width: 8,
           )
           // color: Colors.red,
           // borderRadius: BorderRadius.circular),
         ),
       ),
       SizedBox(
         height: 10,
       ),
       Container(
         child:Column(
           children: [
             Text(
               s.name,
               style: TextStyle(
                 fontWeight: FontWeight.bold,
                 color: Colors.grey,
               ),
             ),
             Text(
               "From Rs.${s.price}",
             )
           ],
         )

       )
     ],
    );
  }
  @override
  Widget build(BuildContext context) {
```

```
  return Scaffold(
    body:
    Column(
      children: <Widget>[
        Row(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: <Widget>[
            Expanded(
              flex: 1,
              child: Row(
                children: <Widget>[
                  Container(
                    child: Icon(
                      Icons.account_circle_sharp,
                      size: 40,
                    ),
                    alignment: Alignment.centerLeft,
                    margin: new EdgeInsets.fromLTRB(20, 0, 10, 0)
                ),
                  Container(
                    child: Text(
                      "User",
                      style: TextStyle(fontSize: 20),
                    ),
                ),
              ],
            ),
          ),
          Expanded(
            flex: 1,
            child: Container(
              width: double.infinity,
            ),
          ),
          Expanded(
            flex: 1,
            child: Container(
              child: Row(
                children: <Widget>[
                  Container(
                    child: Icon(
                      Icons.favorite_border_outlined,
                      size: 40,
                    ),
                    margin: new EdgeInsets.only(right: 10),
                ),
                  Container(
                    child: Icon(
                      Icons.shopping_cart_outlined,
                      size: 40,
                    ),
                )
              ],
```

```
          ),
        ),
      ),
    ],
  ),
),
Container(
  height: 20,
),
Container(
  child: Row(
    children: [
      Expanded(
        flex: 4,
        child: Container(
          child: TextField(
            decoration: InputDecoration(
              border: OutlineInputBorder(
                borderRadius: BorderRadius.circular(10.0),
              ),
              filled: true,
              hintStyle: TextStyle(color: Colors.grey[800]),
              hintText: "Type in your text",
              fillColor: Colors.white70),
          ),
          margin: new EdgeInsets.only(left: 20),
        ),
      ),
      Expanded(
        flex: 1,
        child: Container(
          child: Icon(
            Icons.mic,
          ),
        ),
      ),

    ),

  ],
),
),
),
SingleChildScrollView(
  scrollDirection: Axis.horizontal,
  child: Row(
    children: list1.map((e) => im(e)).toList(),
  ),
),
),
Container(
  child: CarouselSlider(
    items: pro.map((e) => Product(e)).toList(),
    options: CarouselOptions(
      height: 180.0,
      enlargeCenterPage: true,
      autoPlay: true,
```

```
          aspectRatio: 16 / 9,
          autoPlayCurve: Curves.fastOutSlowIn,
          enableInfiniteScroll: true,
          autoPlayAnimationDuration: Duration(milliseconds: 300),
          viewportFraction: 0.8,
        ),
      ),
      margin: EdgeInsets.only(top: 30),
    ),
    SizedBox(
      height: 20,
    ),
    Container(
      child: Row(
        children: [
          Expanded(
            flex: 1,
            child: Container(
              height: 60,
              child: Row(
                children: [
                  Icon(
                    Icons.attach_money_outlined
                  ),
                  Text(
                    "Cash On \n Delivery",
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                    ),
                  )
                ],
                mainAxisAlignment: MainAxisAlignment.center,
              ),
              color: Colors.blue[100],
            )
          ),
          Container(
            width: 2,
            height: 40,
            color: Colors.white,
            margin: EdgeInsets.fromLTRB(2, 0, 2, 0),
          ),
          Expanded(
            flex: 1,
            child: Container(
              height: 60,
              child: Row(
                children: [
                  Container(
                    child: Icon(
                      Icons.local_shipping,
                    ),
                    margin: EdgeInsets.only(right: 5),
```

```
                ),
                Text(
                  "Free Delivery,\nFree Returns",
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ],
              mainAxisAlignment: MainAxisAlignment.center,
            ),
            color: Colors.blue[100],
          )
        ),
        Container(
          width: 2,
          height: 40,
          color: Colors.white,
          margin: EdgeInsets.fromLTRB(2, 0, 2, 0),
        ),
        Expanded(
          flex: 1,
          child: Container(
            height: 60,
            child: Row(
              children: [
                Container(
                  child: Icon(
                    Icons.tag_sharp,
                  ),
                  margin: EdgeInsets.only(right: 5),
                ),
                Text(
                  "Lowest,\nPrice",
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ],
              mainAxisAlignment: MainAxisAlignment.center,
            ),
            color: Colors.blue[100],
          )
        ),
      ],
    ),
    color: Colors.blue[100],
  ),
  SizedBox(
    height: 20,
  ),
  Container(
    child: Row(
      children: [
```

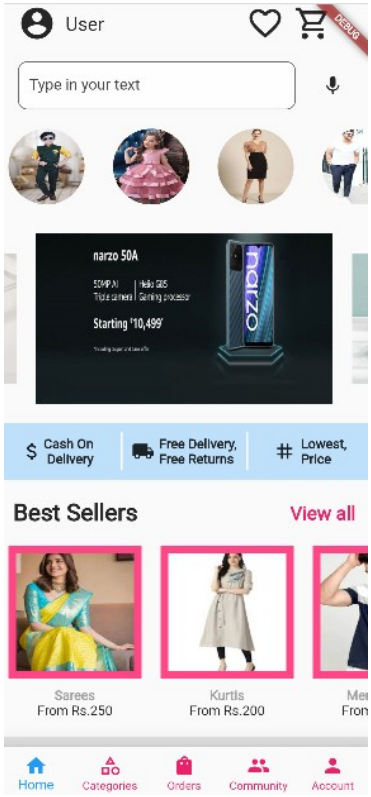
```
            Expanded(
              flex:1,
              child: Text(
                "Best Sellers",
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                  fontSize: 25,
                ),
              )
            ),
            Expanded(
              flex:1,
              child: Text(
                "View all",
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                  fontSize: 20,
                  color: Colors.pink,
                ),
                textAlign:TextAlign.end,
              )
            ),
          ],
        ),
        margin: EdgeInsets.fromLTRB(15, 0, 15, 0)
      ),
      SingleChildScrollView(
        scrollDirection: Axis.horizontal,
        child: Row(
          children: be.map((e) => bestSellers(e)).toList(),
        ),
      ),
      SizedBox(
        height: 20,
      ),
      Container(
        color: Colors.grey[300],
        height: 10,
      ),
      bottom(context),
    ],
  ),
 );
 }
}


class Best{
  late String name;
  late int price;
  late String image;
  Best(String name,int price,String image){
    this.name=name;
```

```
    this.price=price;
    this.image=image;
  }
}


Widget bottom(BuildContext context){
  return BottomNavigationBar(
    items: [
      BottomNavigationBarItem(icon:Icon(Icons.home),label:"Home"),
      BottomNavigationBarItem(icon:Icon(Icons.category_outlined),label:"Categories"),
      BottomNavigationBarItem(icon:Icon(Icons.shopping_bag),label: "Orders"),
      BottomNavigationBarItem(icon:Icon(Icons.people),label: "Community"),
      BottomNavigationBarItem(icon:Icon(Icons.person),label: "Account"),
    ],
    onTap: (value) {
      if(value==4){
        Navigator.pushNamed(context,'/account');
      }
      if(value==0){
        Navigator.pushNamed(context, '/');
      }
    },
    unselectedItemColor: Colors.pink,
    type: BottomNavigationBarType.fixed,
  );

}


void main(){
 runApp(MaterialApp(
    home: Account(),
 ));
}



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

 @override
  _AccountState createState() => _AccountState();
}

class _AccountState extends State<Account> {

 List<String> account=[
    "My Followed Shops","My Bank Details","My Shared Products","My Payments",
    "Refer & Earn","Spins","Enter Referral Code","Meesho Credits","Help",
   "Business Logo","Become a Supplier"
 ];
```

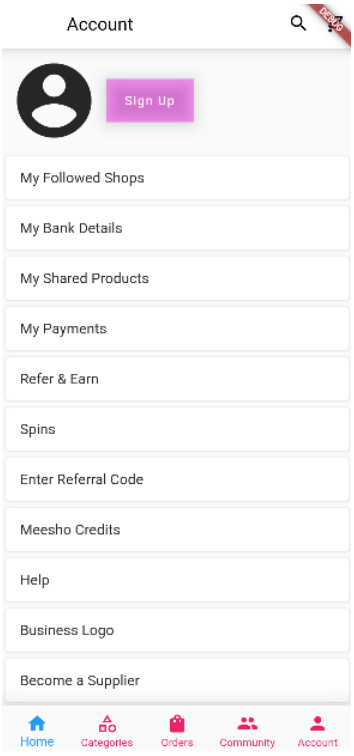
```
 final _formKey = GlobalKey<FormState>();

 final usernameController = TextEditingController();
 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title:Text("Account",style: TextStyle(
        color: Colors.black,
      ),),
      backgroundColor: Colors.white,
      actions: [
        IconButton(
          onPressed: (){},
          icon:Icon(Icons.search),
          color: Colors.black,
        ),
        IconButton(
          onPressed: (){},
          icon:Icon(Icons.shopping_cart_outlined),
          color: Colors.black,
        ),

      ],
    ),
    body: Column(
      children: [
        Container(
          child: Row(
          children: [
            IconButton(
              onPressed:() {},
              icon: Icon(Icons.account_circle_rounded),
              iconSize: 100,
            ),
            FloatingActionButton.extended(
              onPressed: (){},
              label: Text("Sign Up"),
              backgroundColor: Color.fromARGB(100, 253, 1, 240),
              shape: BeveledRectangleBorder(
                borderRadius: BorderRadius.zero
              ),
            ),
          ],
        ),
      ),
        Expanded(
          child:ListView.builder(
            itemCount: account.length,
            itemBuilder: (context,index){
              return Card(
              child: ListTile(
                onTap:(){},
```

```
                    title: Text(account[index]),
                  ),
                );
              }
            ),
          ),
          bottom(context),
        ],
      )
    );
  }
}
```

The Screen:



On clicking on **Account icon** placed in **Bottom Navigation Bar**, he will be displayed his **Account** Section



Conclusion:

Hence, we have learned to apply navigation, routing and gestures in our Flutter App

Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

MAD & PWA Lab Journal

Experiment No.	06
Experiment Title.	To connect Flutter UI with Firebase Database
Roll No.	45
Name	Pushkar Mavale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	
Grade:	

Name: Pushkar Mavale Class/Roll No. D15A/45

MAD and PWA

Lab Experiment

No: 06

To Connect Flutter UI with Firebase Database

Aim: To connect Flutter UI with Firebase.

Theory:

What is Firebase?

Firebase is a Backend-as-a-Service (Baas). It provides developers with a variety of tools and services to help them develop quality apps, grow their user base and earn profit. It is built on Google's infrastructure.

Firebase is categorized as a NoSQL database program, which stores data in JSON-like documents.

Key Features of Firebase:

1. Authentication
2. Realtime Database
3. Hosting
4. Serverless Functions
5. Application Setup for Android, iOS and Web
6. Notifications
7. Storage

Use Cases:

Use cases

Firebase use cases include:

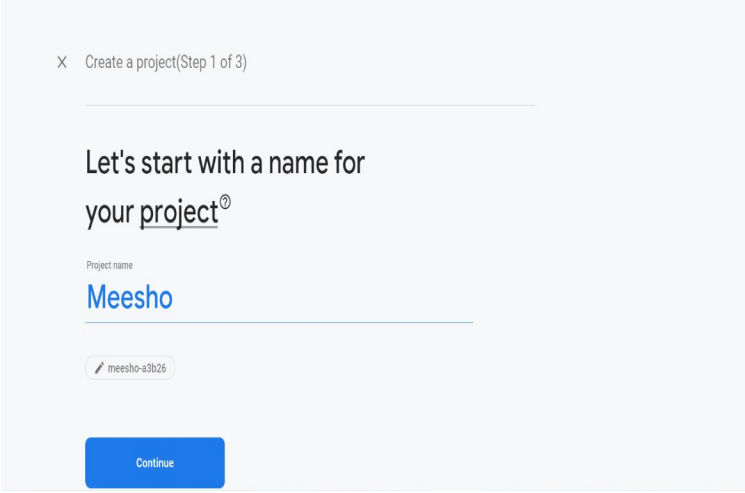
- Create onboarding flows – developers can give users a quick, intuitive sign-in process using Firebase Authentication. They allow users to sign into their apps via their Google, Twitter, Facebook or GitHub accounts in less than five minutes. Developers can also track each step of their onboarding flows to enhance the user experience.

developers can use Google Analytics for Firebase to log events at each step of their onboarding flows, create funnels to determine where users are dropping off and use remote configuration to make changes to their apps to see how those changes affect conversions.

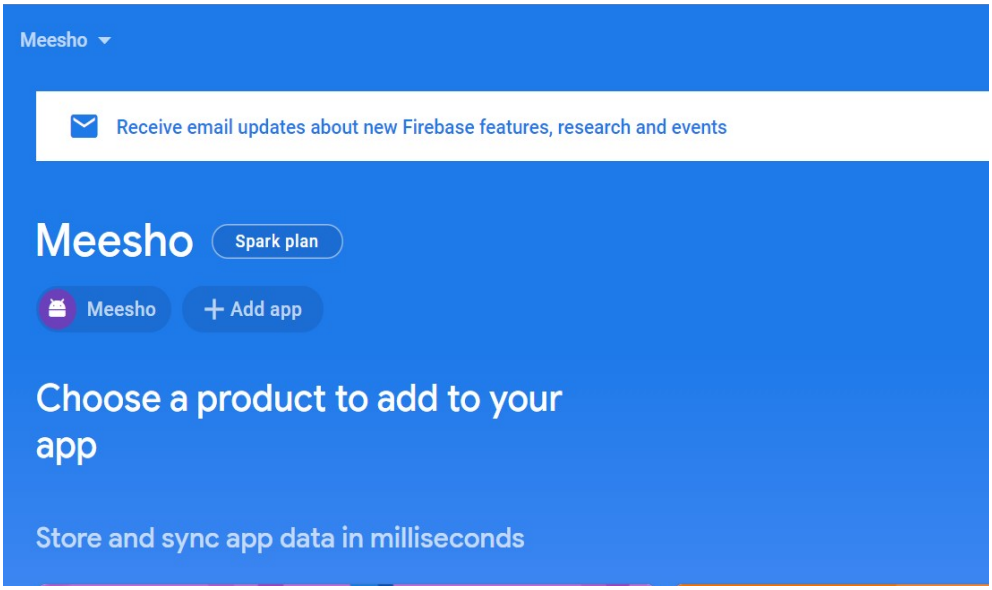
- Customize a “welcome back” screen – developers can use personalization to give every user the best experience by customizing the initial screen based on a user’s preferences, usage history, location or language. Developers can define audiences based, in part, on user behaviours and show targeted content to each audience.
- Progressively roll out new features – developers can launch new features with minimal risk by first testing those features on a few users to see how they work and how users respond. Then, when developers are satisfied, they can roll out their apps to the rest of their users.

Steps to integrate Firebase with a Flutter Project

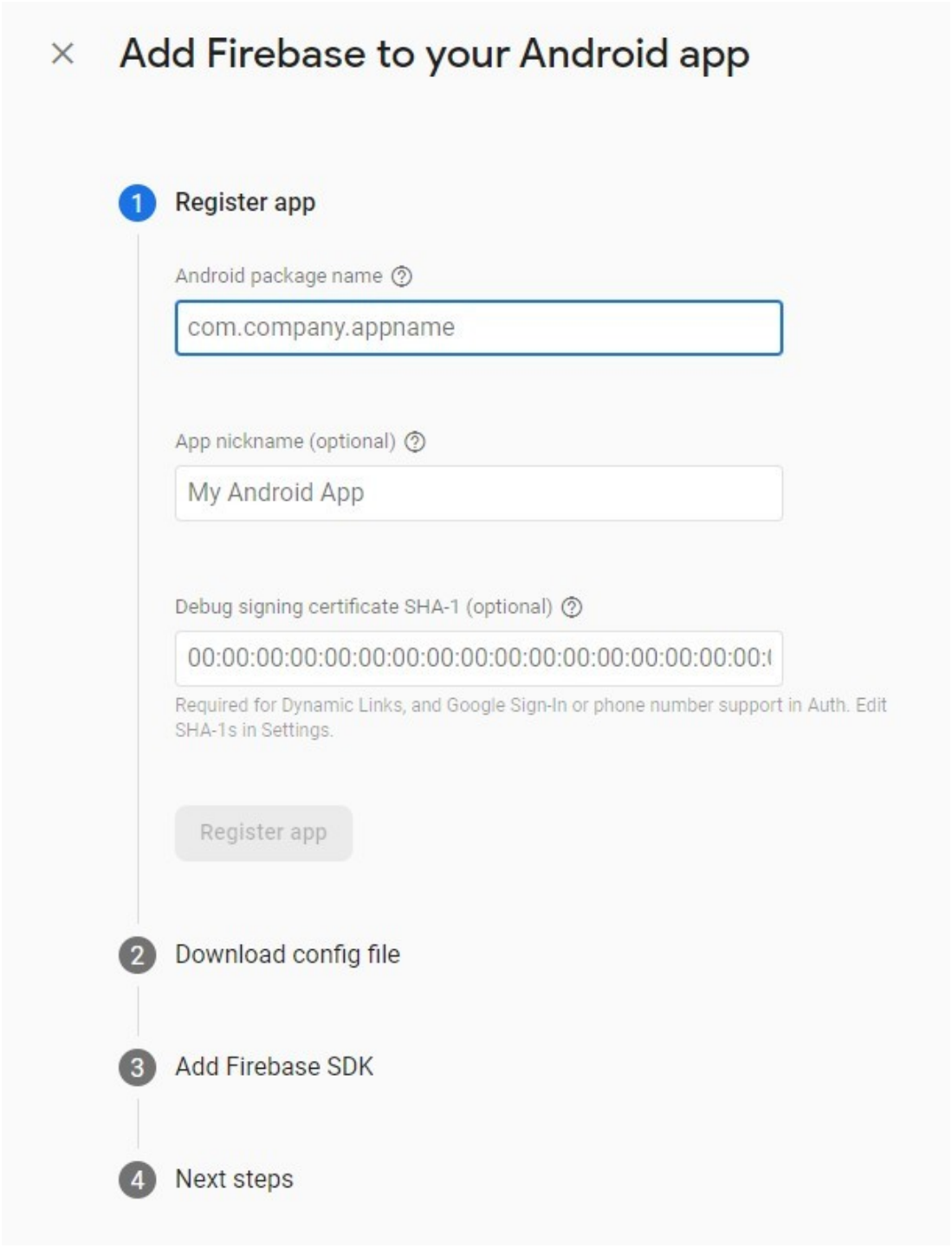
1. Create a Firebase Project



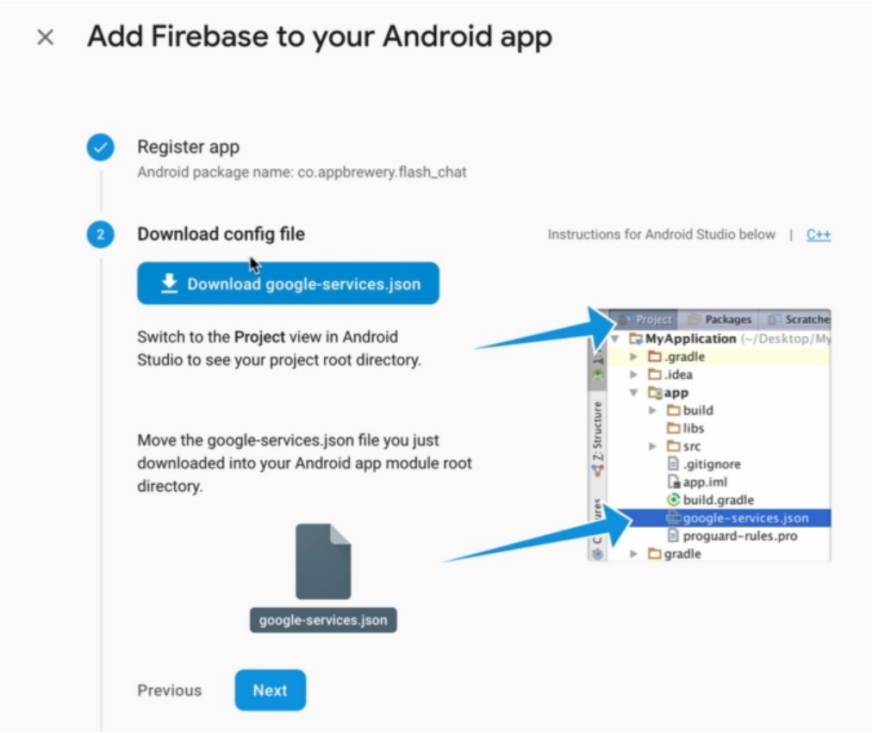
2. Add an Android App (iOS App if you're running your app on iOS)



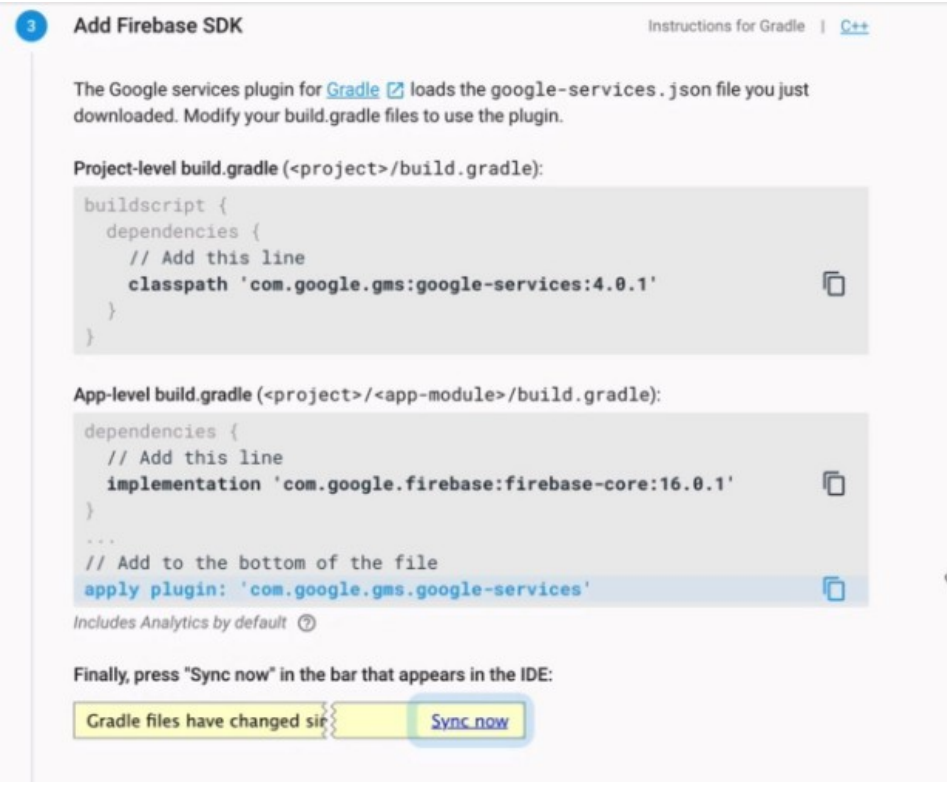
3. Register your Android App using the package name in the app-level build.gradle file and a random app nickname.



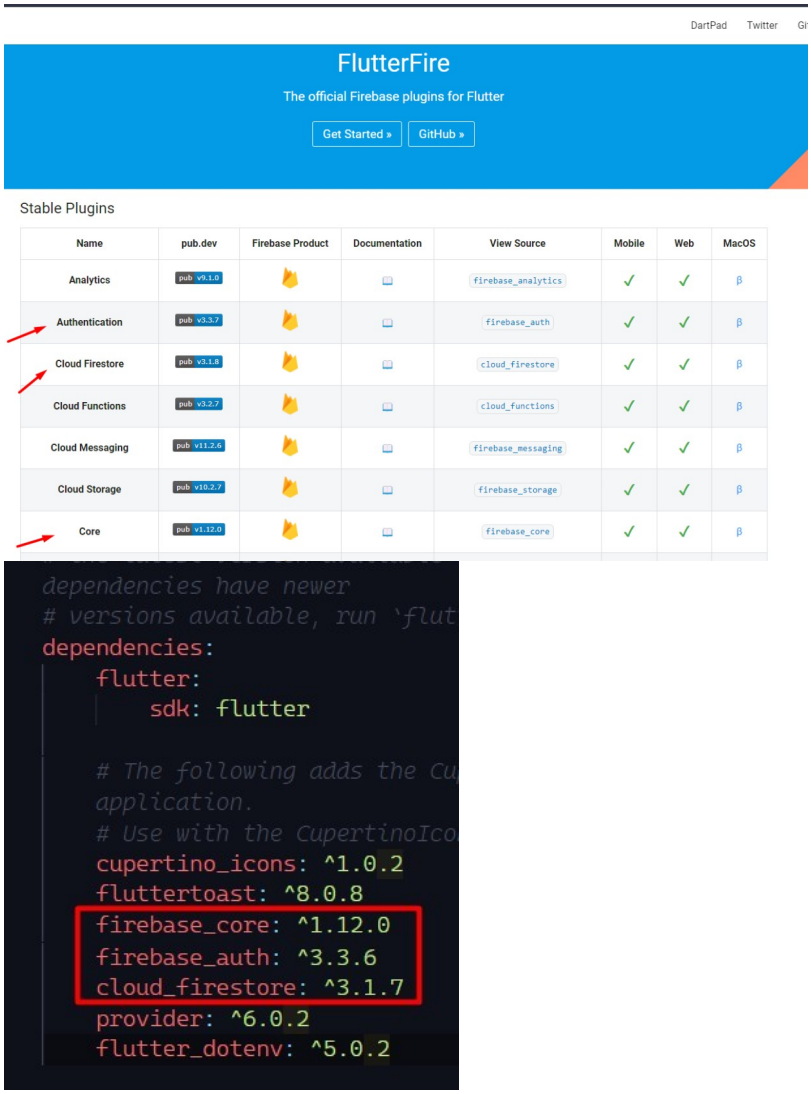
4. Download the google-services.json file and store it in the android/app directory.



5. Add Firebase SDK to the project.



6. Add the following packages from pub.dev to your pubspec.yaml file



7. Once done installing these packages, update the *main.dart* to initialize firebase when the app starts.

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

Conclusion:

Thus, we learned how to integrate Firebase with our Flutter Projects.

Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

MAD & PWA Lab Journal

Experiment No.	07
Experiment Title.	To write meta data of your eCommerce PWA
Roll No.	45
Name	Pushkar Mavale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	
Grade:	

Name: Pushkar Mavale Class: D15A Roll No: 45

Experiment No. 7

To write meta data of your Ecommerce PWA

Aim: To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to home Screen 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:

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

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

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

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

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

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

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

8. **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.

9. **Linkable** — Easily shared via URL without complex installations.

10. **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:

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

Name: Pushkar Mavale Class: D15A Roll No: 45

Code:

```
{
    "name": "Open PWA",
    "short_name": "Open PWA",
    "start_url": "./",
    "scope": ".",
    "display": "standalone",
    "orientation": "portrait",
    "background_color": "#fff",
    "theme_color": "#640700",
    "newBranch":"true",
    "icons": [
      {
        "src": "assets/icons/icon.png",
        "type": "image/png",
        "sizes": "48x48"
      },
      {
        "src": "assets/icons/icon.png",
        "type": "image/png",
        "sizes": "96x96"
      },
      {
        "src": "assets/icons/icon.png",
        "sizes": "128x128",
        "type": "image/png"
      },
      {
        "src": "assets/icons/icon.png",
        "sizes": "144x144",
        "type": "image/png"
      },
      {
        "src": "assets/icons/icon.png",
        "type": "image/png",
        "sizes": "192x192"
      },
      {
        "src": "assets/icons/icon.png",
        "type": "image/png",
```

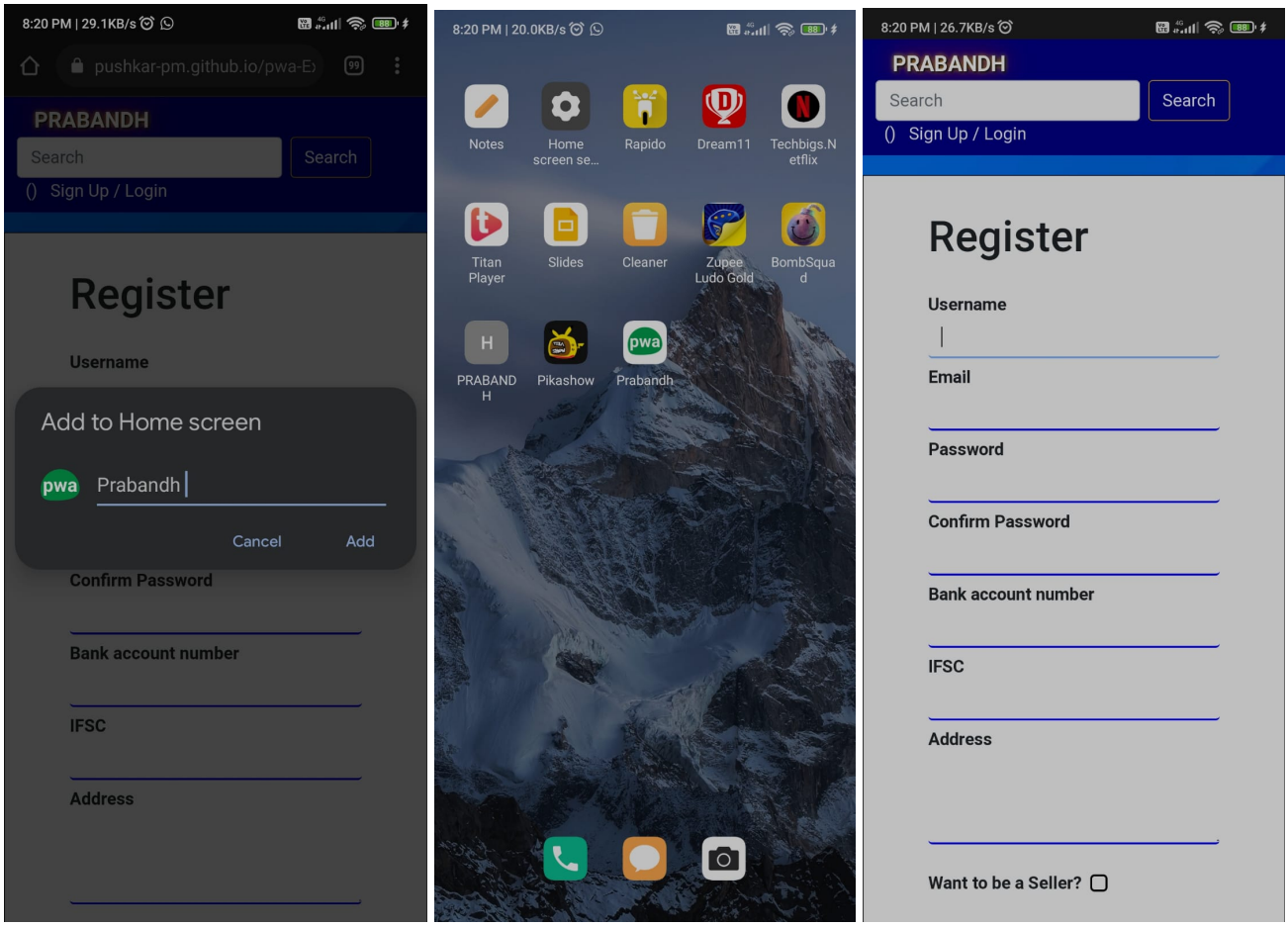
Name: Pushkar Mavale Class: D15A Roll No: 45

```
        "sizes": "256x256"
      },
    ]
}
```

Add the link tag to link to the manifest.json file

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

Output:



Conclusion:

Hence, we learnt how to write a metadata of our E-commerce website PWA in a Web App Manifest File to enable add to home screen feature.

Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

MAD & PWA Lab Journal

Experiment No.	08
Experiment Title.	To code and register a service worker
Roll No.	45
Name	Pushkar Mavale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	
Grade:	

Name: Pushkar Mavale Class: D15A Roll No: 45

Experiment No. 8
To code and register a service worker

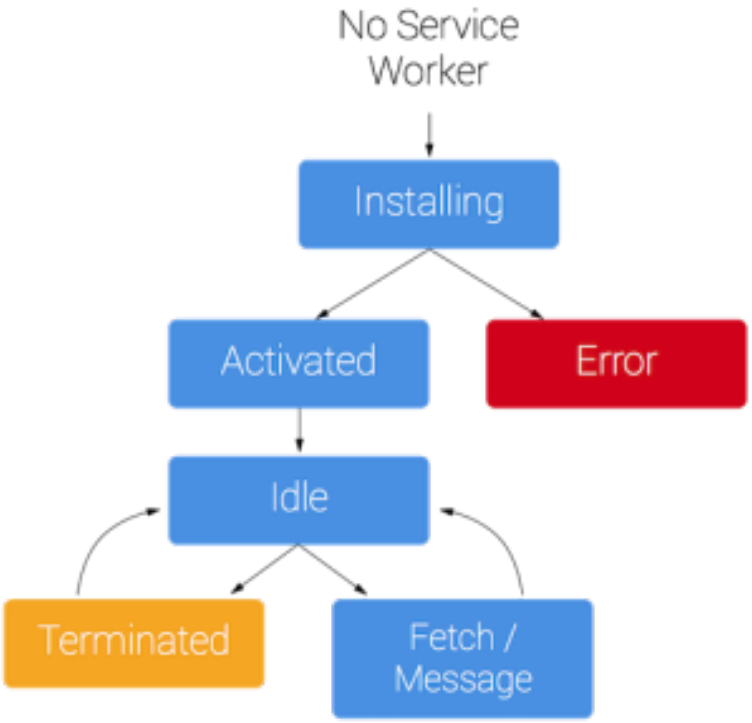
Aim: To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA

Theory:

Service Worker 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.

Installation of service worker:

Life Cycle of service worker:



Registering service worker:

Before a service worker takes control of your page, it must be registered for your PWA. That means the first time a user comes to your PWA, network requests will go directly to your server because the service worker is not yet in control of your pages.

After checking if the browser supports the Service Worker API, your PWA can register a service worker. When loaded, the service worker sets up shop between your PWA and the network, intercepting requests and serving the corresponding responses.

```
if ('serviceWorker' in navigator) { navigator.serviceWorker.register('/js/service-worker.js')
.then((registration) => { console.log("service worker registered")
})
.catch((err) => { console.log("sw registraration "+ err) }); }
```

Installing service worker:

In this phase, pre-fetch operations are typically executed. Their goal is to ensure target assets are downloaded and made already available in the cache for the SW. These assets are commonly static files (e.g., js, css) that represent the *core shell* of our application. That means only the minimum files and styles should be available immediately to the user, even when offline.

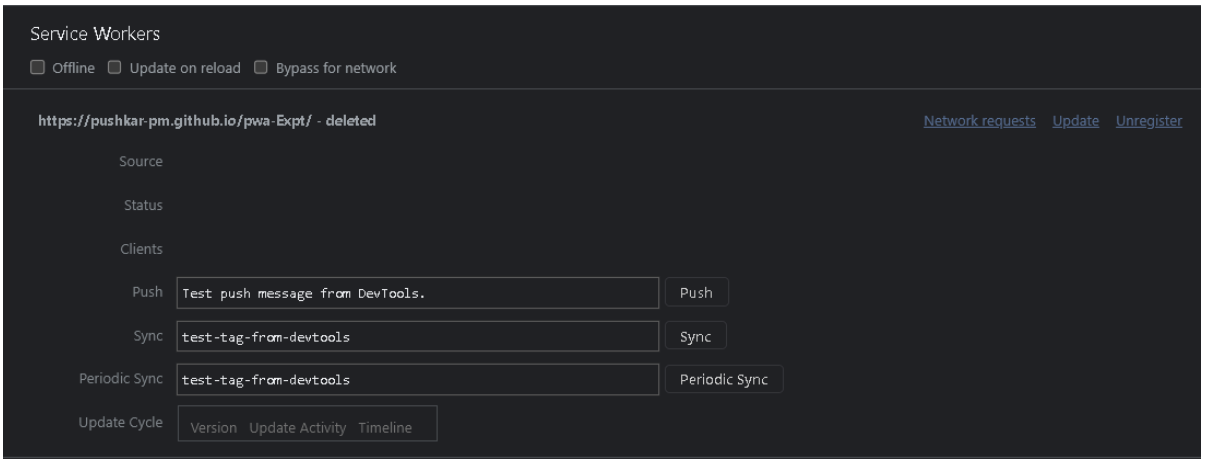
```
var urlsToCache = [
'/',
'/css/app.css',
'/js/app.js',
        'https://fonts.googleapis.com/css2?family=Lato:wght@300;400;700&display=s w ap',
         'https://unpkg.com/tailwindcss@^1.0/dist/tailwind.min.css',
         'https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/fon
t awesome.min.css',
        'https://maxst.icons8.com/vue-static/landings/line-awesome/lin e
awesome/1.3.0/css/line-awesome.min.css'
];
self.addEventListener('install', (event) => {
console.log("service worker installted") event.waitUntil(
caches.open('static')
.then((cache) => { return cache.addAll(urlsToCache);
})
```

Verify if a service worker is registered

To verify if a service worker is registered, use developer tools in your favorite browser.

In Firefox and Chromium-based browsers (Microsoft Edge, Google Chrome, or Samsung Internet):

1. Open developer tools, then click the Application tab.
2. In the left pane, select Service Workers.
3. Check that the service worker's script URL appears with the status "Activated". (You'll learn what this status means in the lifecycle section in this chapter). On Firefox the status can be "Running" or "Stopped".



Updating a service worker

Service workers get updated when the browser detects that the service worker currently controlling the client and the new (from your server) version of the same file are byte-different.

After a successful installation, the new service worker will wait to activate until the existing (old) service worker no longer controls any clients. This state is called "waiting", and it's how the browser ensures that only one version of your service worker is running at a time. Refreshing a page or reopening the PWA won’t make the new service worker take control. The user

needs to close or navigate away from all tabs and windows using the current service worker and then navigate back. Only then will the new service worker to take control. Visit this service worker lifecycle article for more information.

Code:

Register:

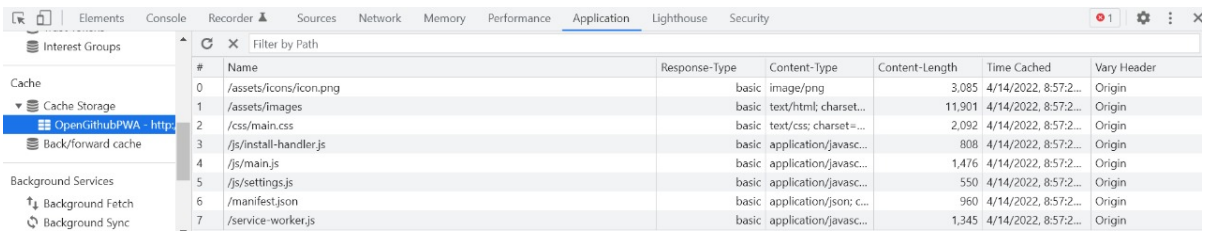
```
<script
type="text/javascript">
                    if ('serviceWorker' in navigator) {
                      navigator.serviceWorker
                        .register(`service-worker.js`)
                        .then(function () {
                          console.log("Service Worker Registered");
                        });
                    }
                  </script>
```

Service-worker code:

```
let cacheName =
"OpenGithubPWA";
                    let filesToCache = [
                      "service-worker.js",
                      "js/main.js",
                      "js/install-handler.js",
                      "js/settings.js",
                      "css/main.css",
                      "assets/icons/icon.png",
                      "assets/images",
                      "manifest.json"
                    ];

                    self.addEventListener("install", function (event) {
                      event.waitUntil(caches.open(cacheName).then((cache) => {
                        console.log('installed successfully')
                        return cache.addAll(filesToCache);
                      }));
                    });


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

```
`                 if (event.request.url.includes('clean-cache')) {
                   caches.delete(cacheName);
                   console.log('Cache cleared')
                  }


                  event.respondWith(caches.match(event.request).then(function
                (response) {
                   if (response) {
                    console.log('served form cache')
                   } else {
                    console.log('Not serving from cache ', event.request.url)
                   }
                   return response || fetch(event.request);
                  })
                  );
                });


                self.addEventListener('activate', function (e) {
                 e.waitUntil(
                  caches.keys().then(function (keyList) {
                   return Promise.all(keyList.map(function (key) {
                    if (key !== cacheName) {
                     console.log('service worker: Removing old cache', key);
                     return caches.delete(key);
                    }
                   }));
                  })
                 );
                 return self.clients.claim();
                });
```

Output:



Conclusion:

I have learn from this experiment how we can create service worker and store the cache that will improve the application to work in offline environment

Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

MAD & PWA Lab Journal

Experiment No.	09
Experiment Title.	To implement service worker events like fetch
Roll No.	45
Name	Pushkar Mavale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	
Grade:	

Name: Pushkar Mavale Class: D15A Roll No: 45

Experiment No: 09
MAD and PWA Lab

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 “offine frst” 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 Indexed DB 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 requests and current location’s origin are the same (Static content is requested.), this is called “cache First” but if you request a targeted external URL, this is called “network First”.

- **Cache First** - 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.

- **Network First** - 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.

  Code:

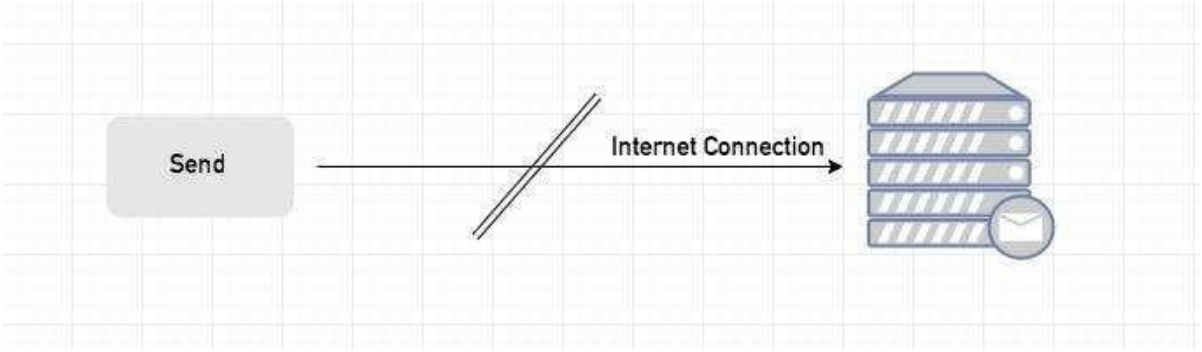
```
var cacheName = 'geeks-cache-v1';


// Call Fetch Event

self.addEventListener('fetch', e => {

console.log('Service Worker: Fetching');

e.respondWith(

fetch(e.request)

.then(res => {

// The response is a stream and in order the browser

// to consume the response and in the same time the

// cache consuming the response it needs to be // cloned in order to have two streams.

const resClone = res.clone();

// Open cache

caches.open(cacheName)
```

```
.then(cache => {

// Add response to cache

cache.put(e.request, resClone);

});

return res;

}).catch(

err => caches.match(e.request)

.then(res => res)

)

    ); });
```

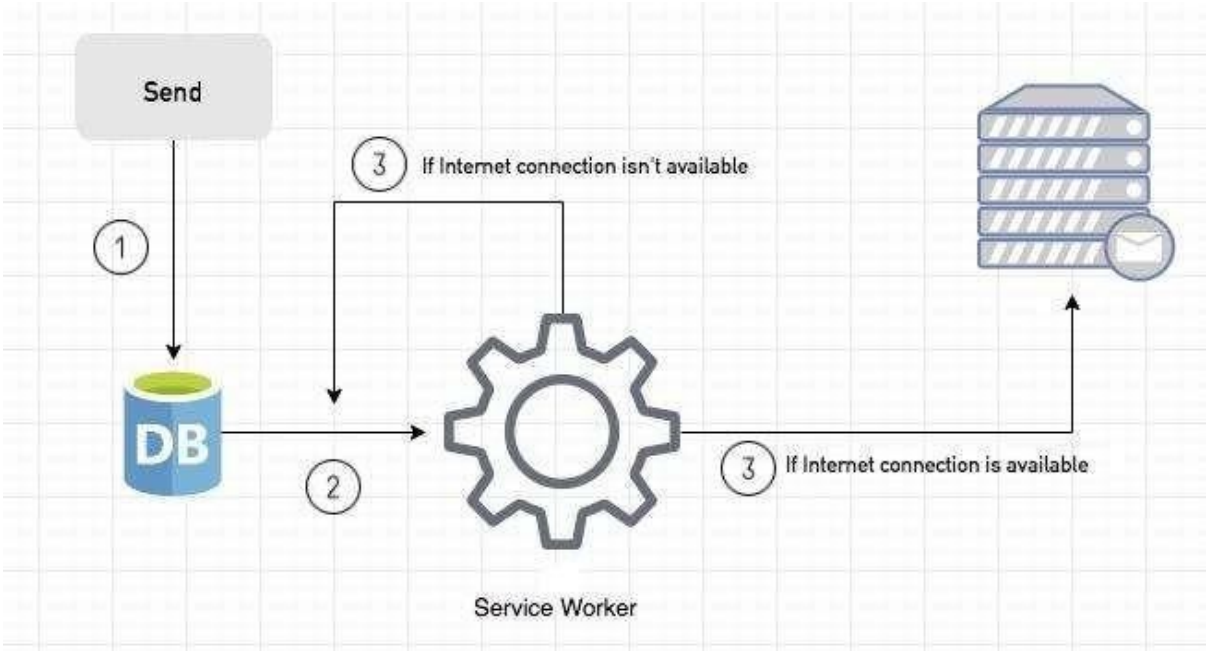
Sync Event

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

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



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



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

Push Event

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

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 fle. 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
“push Message”, we open the information notification with the “message” property
You can use Application Tab from Chrome Developer Tools for testing push notification.

Code

```
let cacheName = "OpenGithubPWA";//       any unique name


let filesToCache = [
  "service-worker.js",
  "js/main.js",
  "js/install-handler.js",
  "js/settings.js",
  "css/main.css",
  "assets/icons/icon.png",
  "assets/images",
  "manifest.json"
  // add your assets here
  // !          donot add config.json here !
];


self.addEventListener("install", function (event) {
  event.waitUntil(caches.open(cacheName).then((cache) => {
    console.log('installed successfully')
    return cache.addAll(filesToCache);
  }));
});
```

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

  if (event.request.url.includes('clean-cache')) {
    caches.delete(cacheName);
    console.log('Cache cleared')
  }

  event.respondWith(caches.match(event.request).then(function (response) {
    if (response) {
      console.log('served form cache')
    } else {
      console.log('Not serving from cache ', event.request.url)
    }
    return response || fetch(event.request);
  })
  );
});

self.addEventListener('activate', function (e) {
  e.waitUntil(
    caches.keys().then(function (keyList) {
      return Promise.all(keyList.map(function (key) {
        if (key !== cacheName) {
          console.log('service worker: Removing old cache', key);
          return caches.delete(key);
        }
```

```
      }));
    })
  );
  return self.clients.claim();
});
```

Main.js:

```
window.addEventListener('load',main)
function main(){

   vaildateCacheIfOnline()
   .then(_=>{

   })

}


function vaildateCacheIfOnline(){

   return new Promise((resolve,reject)=>{

      fetch(`config.json?cacheBust=${new Date().getTime()}`)
      .then(response => { return response.json() })
      .then(config => {
```

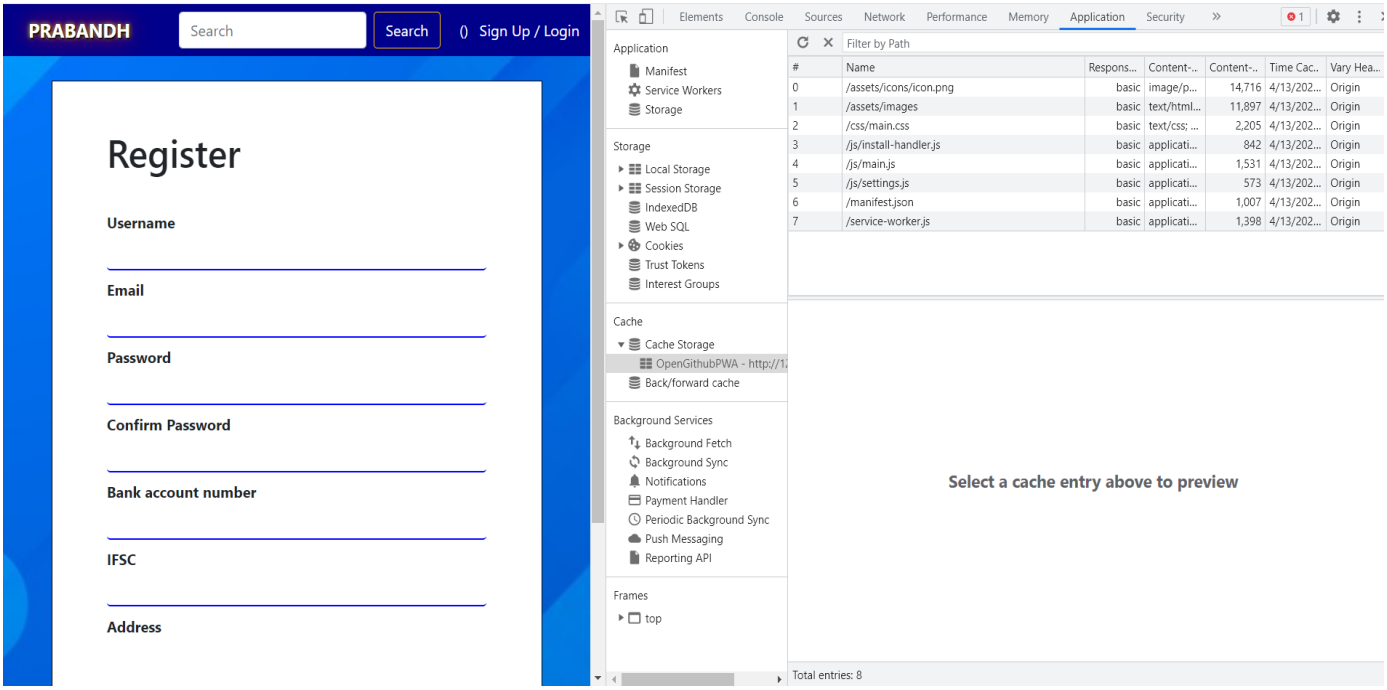
```
    let installedVersion = Settings.getVersion()
    if ( installedVersion== 0) {
       Settings.setVersion(config.version)
       document.querySelector('#version').innerHTML= `version ${config.version}`;
       return resolve();
    }
    else if (installedVersion != config.version) {
       console.log('Cache Version mismatch')
       fetch(`config.json?clean-cache=true&cacheBust=${new Date().getTime()}`).then(_ => {
          //actually cleans the cache
          Settings.setVersion(config.version);
          window.location.reload();

          return resolve();  // unnecessary
       });

    }else{
       // already updated
       console.log('Cache Updated')
       document.querySelector('#version').innerHTML= `version ${installedVersion}`;
       return resolve();
    }
 }).catch(err=>{
    console.log(err);
    return resolve();
    //handle offline here
```

```
        })
    })


}
```

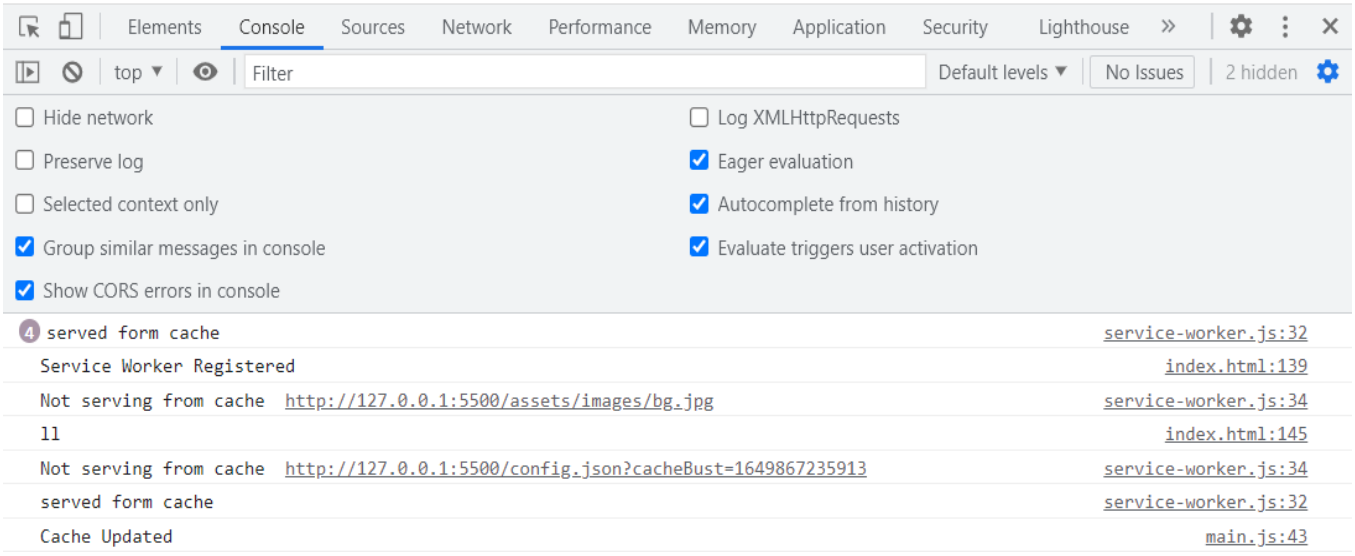
OUTPUT :

Fetch

Name: Pushkar Mavale Class: D15A Roll No: 45

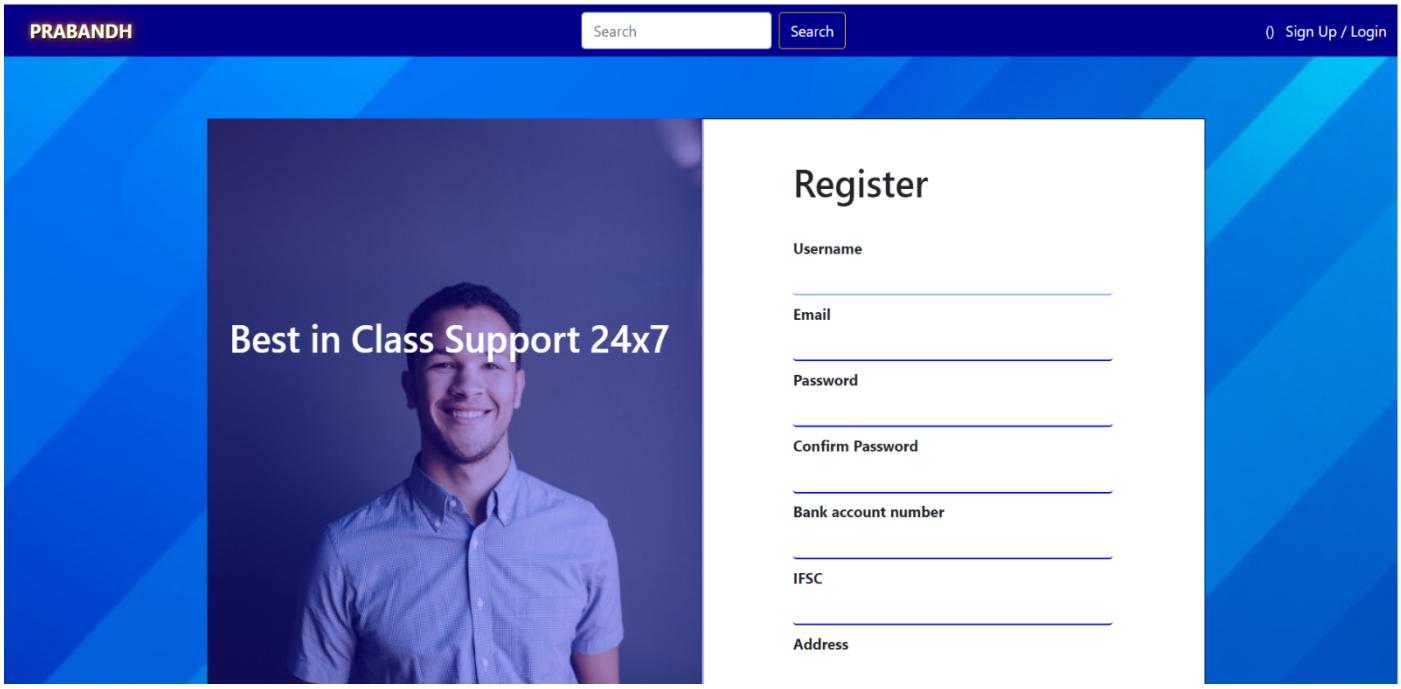


Sync



Push

Name: Pushkar Mavale Class: D15A Roll No: 45



Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

MAD & PWA Lab Journal

Experiment No.	10
Experiment Title.	Deploy ECommerce PWA to pages
Roll No.	45
Name	Pushkar Mavale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	
Grade:	

Name: Pushkar Mavale Class: D15A Roll No: 45

Experiment No. 10

MAD & PWA Lab

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 favouring 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 setup. 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 a record in the site's DNS configuration, and you are done. Cons

5. The code of your website will be public, unless you pay for a private repository.

6. Currently, there is no support for HTTPS for custom domains. It's probably coming soon though.

7. 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-timeout-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 favouring GitHub Pages:

1. Real-time backend made easy

2. Fast and responsive

Instacart, 9GAG, and Twitch are some of the popular companies that use Firebase

Name: Pushkar Mavale Class: D15A Roll No: 45

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

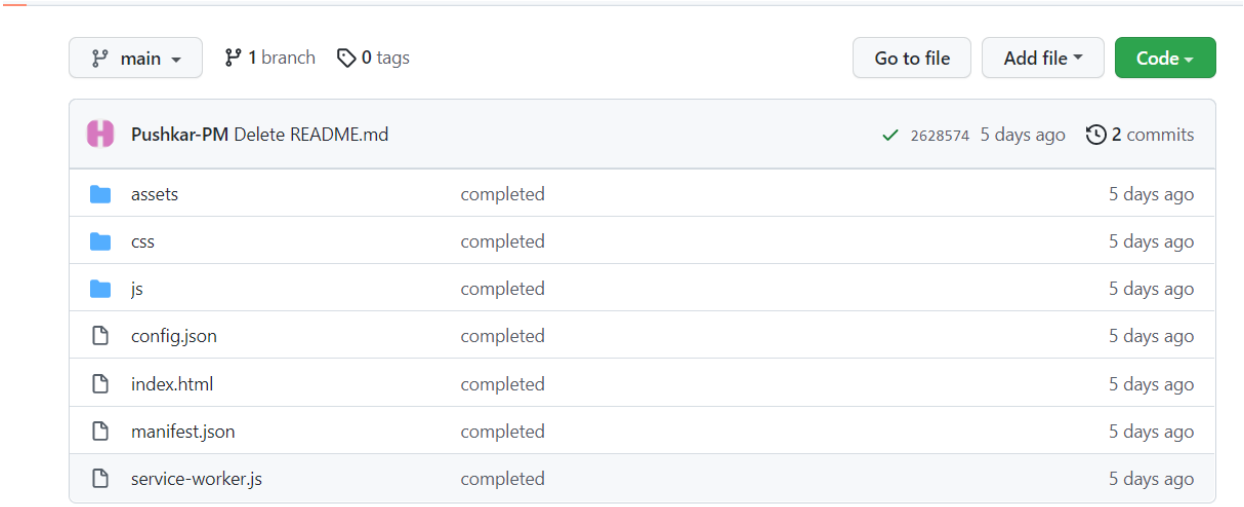
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem if you use a CDN or AMP.
2. A command-line interface only.
3. No in-built support for any static site generator.

GitHub Link:

https://github.com/gauravmandal1/ves_gram

Select your GitHub Repository, then go to Settings, click on Pages. In Custom Domain give the URL of your hosted site and click on save.

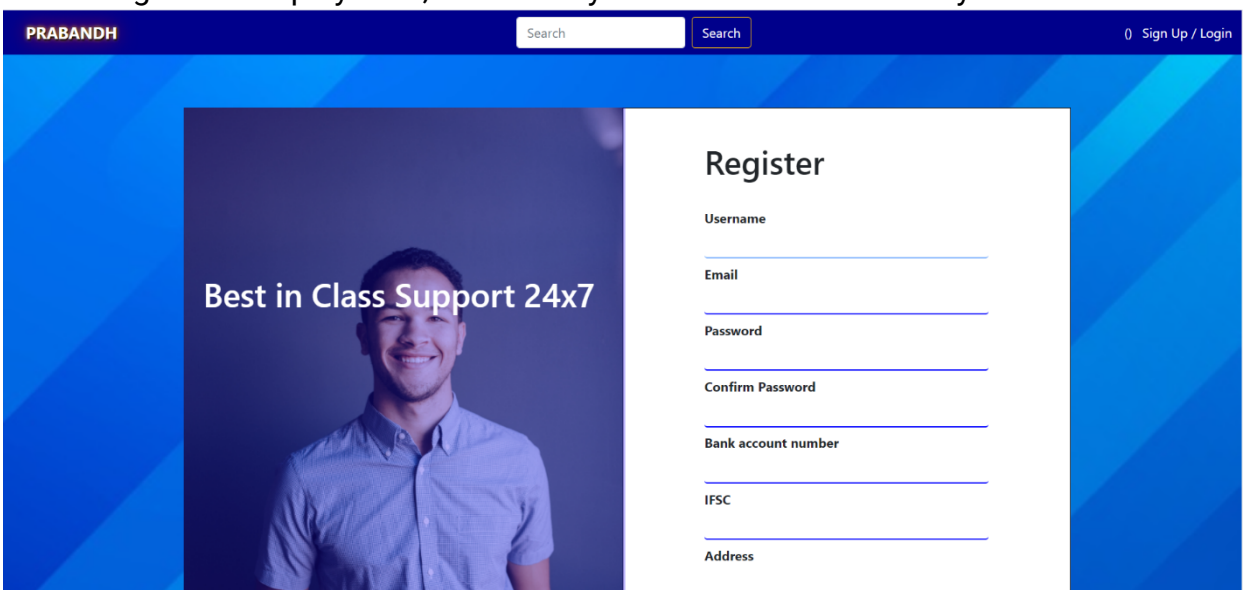
Name: Pushkar Mavale Class: D15A Roll No: 45



It will create a CNAME file in your repository.

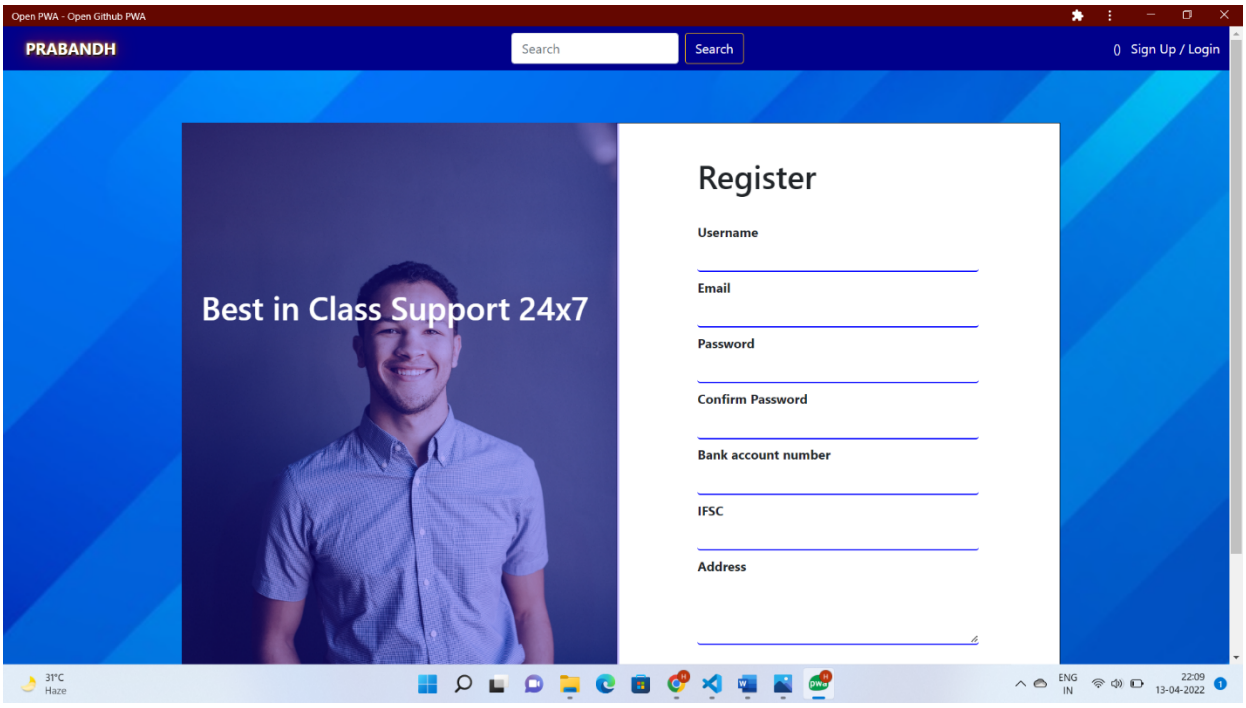
Your Github-pages Environment is ready.

On clicking on the deployment, it will take you to the hosted URL for your website.



Name: Pushkar Mavale Class: D15A Roll No: 45

On installing the PWA, the shortcut will be created on clicking the shortcut, it will open the PWA.



Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

MAD & PWA Lab Journal

Experiment No.	11
Experiment Title.	Using Google lighthouse for PWA analysis
Roll No.	45
Name	Pushkar Mavale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	
Grade:	

Name: Pushkar Mavale Class: D15A Roll No: 45

Exp 11: Lighthouse PWA analysis tool

Aim: to use google lighthouse tool PWA analysis tool to test PWA functioning

Theory:

What is a Lighthouse?

Lighthouse is an extraordinary tool that is constantly evolving and upgrading its offered set of features. We can use it to audit any web page, even if the page requires authentication.

There are different audits categories we can choose from:

- Performance
- Accessibility
- SEO
- Best practices
- Progressive web apps

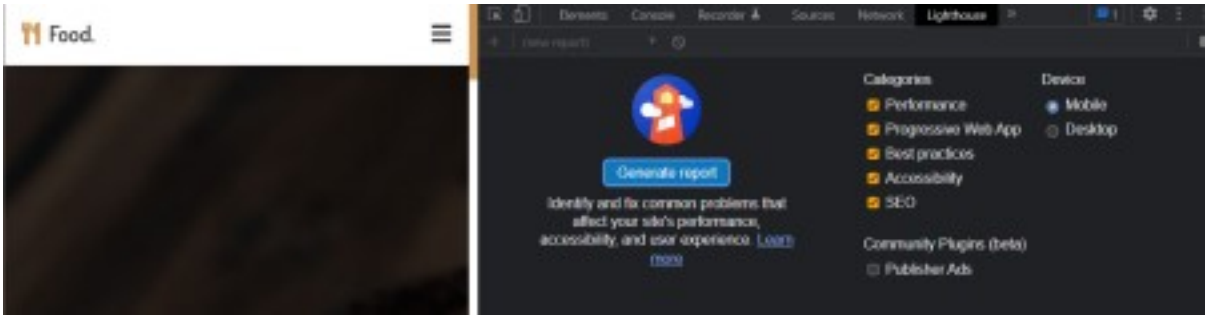
It is also possible to run Lighthouse in different contexts. The simplest one is from Chrome Dev Tools, but it is also possible from the command line, or as a Node module.

Once the page analysis process is finished, the output becomes a series of audits, passed or not, providing warnings and suggestions based on best practices in order to improve our web app under different aspects. Each suggestion point comes with a reference to Google documentation about a topic, so that we can learn more details about it.

In this lesson, we will describe Lighthouse from the Chrome Dev Tools. In my opinion, this is the easiest way to analyse a PWA during and after we developed it.

Lighthouse interface

Pressing the F12 key in Chrome opens the Dev Tools:



We can find Lighthouse under the **Audits** tab. Aside from the categories listed before, we can also choose whether we want to run the audits against a mobile or desktop device. If we choose mobile, the tool uses the built in Chrome device emulator to run the audits throttled to a fast 3G network and 4x CPU slowdown.

Lighthouse audits

If we click on the “Generate report” button, Lighthouse starts running audits for the chosen categories. By default, all are selected, but you can decide to activate only a few of them to save time.

In case we have some Chrome extensions, we receive a warning after the analysis is finished as these directly affect the Lighthouse estimations.

The generated report has a summary header with a score up to 100 for each category (a value of 0 means that an error occurred).
This immediately gives an overview about how good our web application performed against the Lighthouse checks.

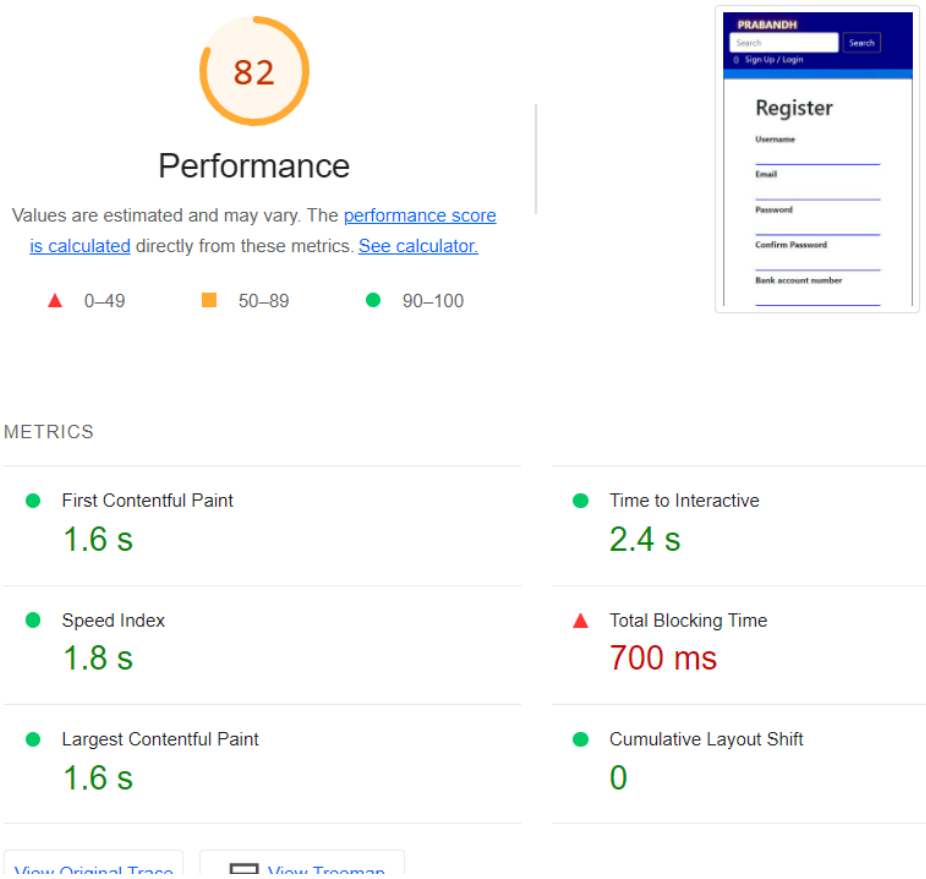
For each category, we can also drill down the details if the audits passed.

Lighthouse PWA category

If we implement everything properly in our project, the Lighthouse output should look like the screenshot above, where the audits are categorized into three groups: "Fast and reliable," "Installable," and "PWA Optimized."

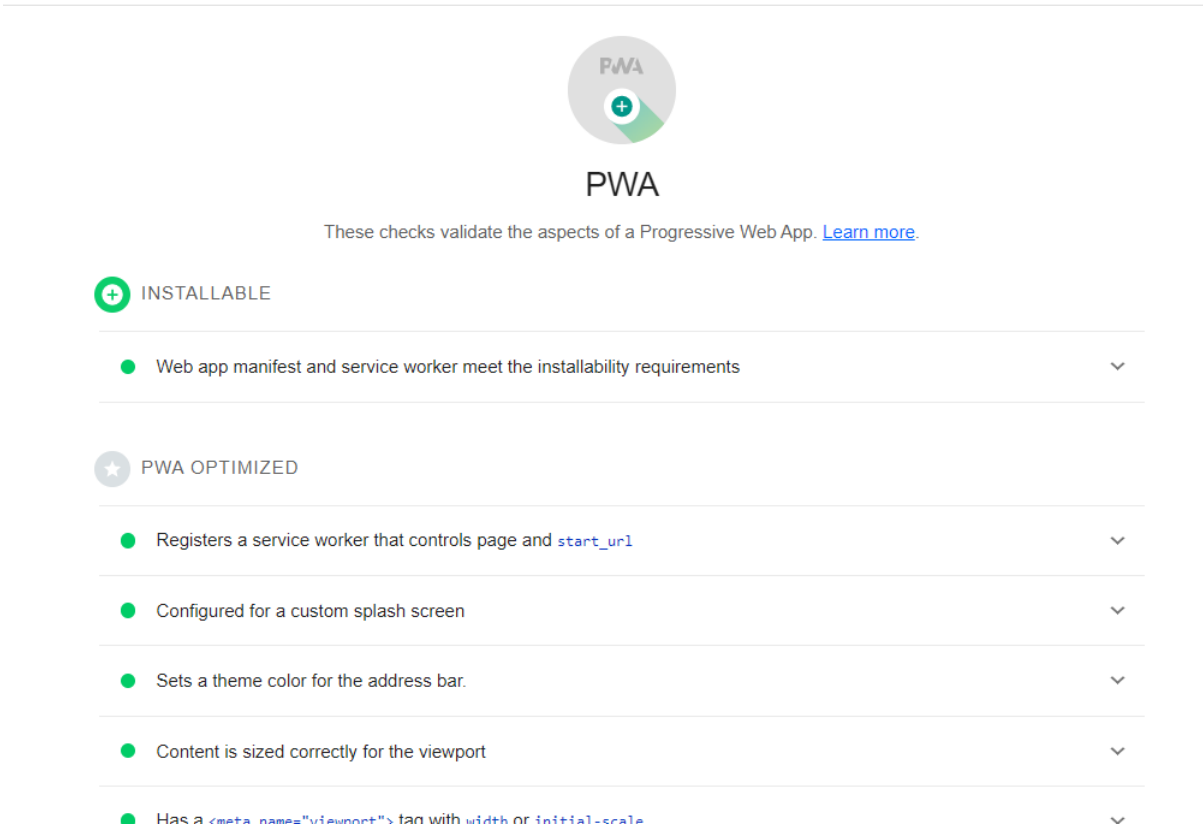
Name: Pushkar Mavale Class: D15A Roll No: 45

This grouping structure helps us better understand the effect of the audit on our application.



Some aspects are easy to achieve. For example, using a HTTPS or a theme colour is relatively simple, but others can be less obvious to achieve. Therefore, having an audit tool like Lighthouse can help us avoid missing an important aspect, which can have a serious impact on the performance or functioning of our PWA.

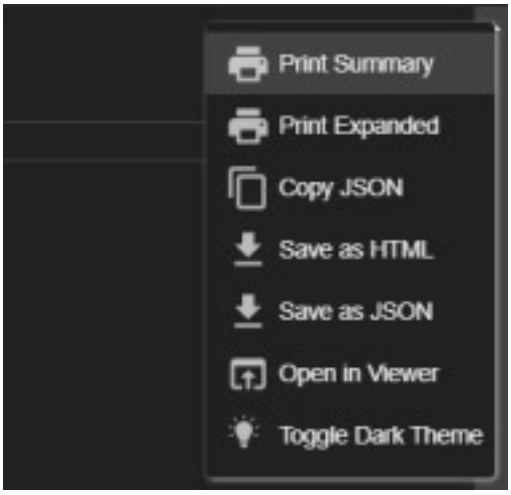
Name: Pushkar Mavale Class: D15A Roll No: 45



Export audits output

Once we run the tests, we can export the output to keep as a reference and compare the results after we edit our project.

Accessing the menu on the Audits tab of Chrome DevTools, we have the option to save the audits results in different formats.



Name: Pushkar Mavale Class: D15A Roll No: 45

Conclusion :

In this lesson you learned about another important instrument to add to your tool belt. I would suggest using Lighthouse not only at the end of implementing the PWA, but also at intermediate stages, as it can provide valuable insights to implement or at least plan while the development is ongoing.

MAD and PWA Lab

Assignment - 01: Part 1

Aim: Picking and choosing features of an application to develop

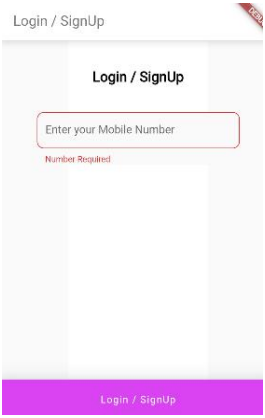
Description:

I've picked Meesho as the application to develop for the following assignments in the future.

Meesho is a startup with a basic idea of to provide various items at cheapest rate.. Meesho has its mobile application in both Android and in iOS platforms, it connects the sellers with the customers.

Following are the Screens:

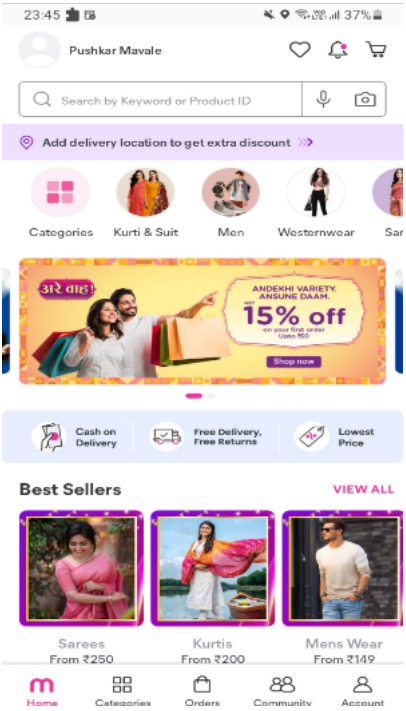
1. Login / SignUp Page:



When we open the Meesho App, a login page will open up. Here we have to enter our phone number first and then they will send a verification code on the phone number. After that, the code will be verified and login will be successful.

We will have widgets like TextKey for keyboard, DropDownButton is used for selecting the respective country. Elevated Button is used for the Login / SignUp button. The whole system is a column widget. We might be using the firebase database for storing the phone number provided by the user and for further authentication purposes.

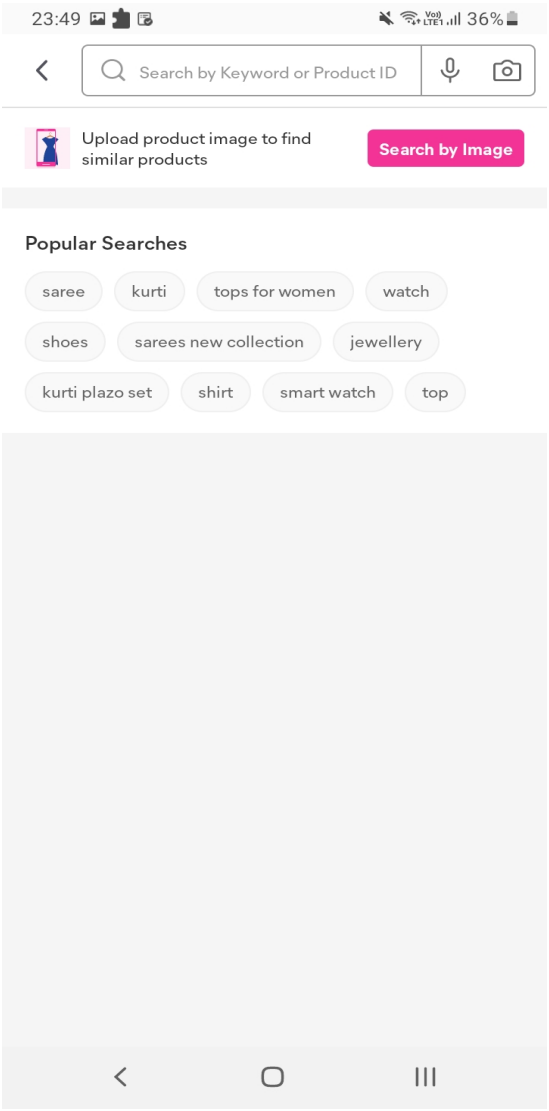
2. Home Page:



After successful Login, we will be directed to this Home Page which will include all the associated Home Services which will be offered by them. This includes a Search bar which is of the type of TextField Widget. Also, it shows Services in the Carousal Form which is of the type of Carousal Slider Widget.

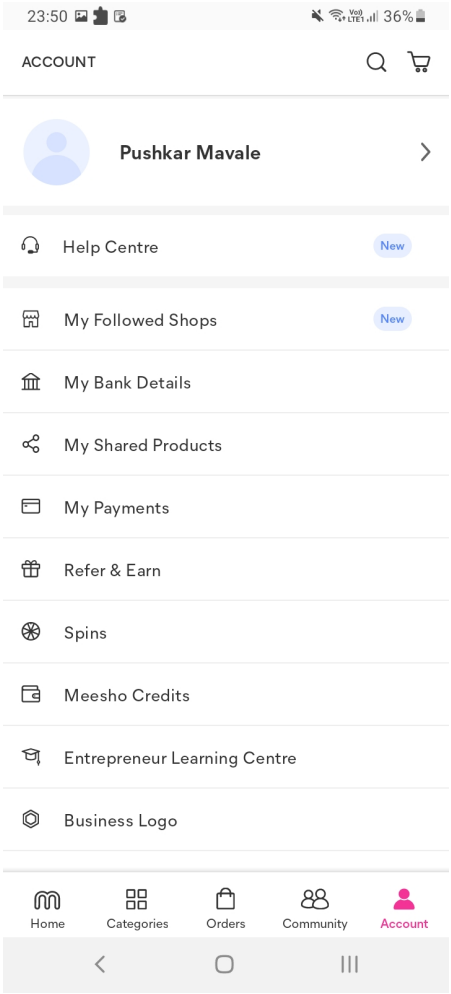
There are also some of Services offered by them which are further classified as per services which are placed in Column widget wrapped by an Expanded widget. Also there is a Bottom Navigation bar which includes list of icons for navigating from one page of app to another.

3.Search Page:



This is the search page where we can search for different services as per our convenience . It will show the things matching our interests. Here TextField is used for Search input.

4. Profile page



Here, User will get a list of available services This list will be formed using the ListView widget which will include Container as its childrens in it.

ASSIGNMENT NO: 2
MAD and PWA Lab

Aim: To study the requirement for progressive web application for E-commerce using the concept of service worker, Web app Manifest and framework tools.

Progressive Web ADD

At its core, a Progressive Web App (PWA) is a web application that uses the latest web capabilities to deliver a native app-like experience to users. It is the technology that will bring consistency between web and native apps and replace both with a single instance.

Progressive Web Apps are:
1. **Reliable.** They load instantly, regardless of the network state.They eliminate dependence on a network connection, ensuring an instant and reliable user experience, even with bad or nonexistent internet connection.
2. **Incredibly fast.** If your site is slow, customers leave, making you miss out on potential sales. PWAs load extremely fast and respond quickly to user interactions. Improved website performance positively affects conversion, user experience, and retention rates.
3. **Highly engaging.** They provide a true native app experience, including web capabilities such as the option to send Push Notifications and appear on the homescreen.

Importance of PWA:

1. **Better user adoption by installing on home screen:** Progressive Web Apps provide a highly engaging user experience, with the same capabilities as native apps. Progresssive Web Apps can be installed on the home screen, making them directly accessible to users.

In a typical month, the average amount of apps downloaded by a smartphone user is close to zero. People don't download apps.

With a Progressive Web App, 'downloading' is as simple as visiting a web link and accepting a prompt. From a user perspective, the website will transform into a 'real' app - so not a browser shortcut - by being placed on the homescreen. This achieves the same result as downloading from an app store, but without the hassle of going through the app store process. Additionally, after the PWA is installed on the homescreen, no further

updates are required. PWA's are automatically updated in the background, assuring your customers always get to see the most recent content.

2. **Boosting user engagement with push notifications:** Similar to a native app, PWA offers the capability to send Push Notifications to mobile devices. With a very intuitive way, business can reach their audience where they are the most: directly on their mobile device, creating a highly engaging user experience.

These small messages have a big impact: they grab attention, always. Compared to traditional marketing, such as emails, Push Notifications have the big advantage that they appear directly on a persons' phone; they stand out from the crowd and most importantly, a business does not need to make a lot of effort to collect email addresses or other information before they can engage with their audience.

3. **Reduced development costs, faster innovation & continuous delivery:** Businesses with an omnichannel, multiplatform strategy are currently building, maintaining and promoting up to 4 systems: their (responsive) website, their Android app, their iOS apps and in some cases also Windows 10 native apps.
A Progressive Web App completely eliminates the need for development, maintenance and marketing of any platform other than the PWA website. This provides a unique opportunity to serve all channels from one platform, which is built, maintained and serviced by one team. It reduces costs and time-to-market drastically, while still serving the same user experience and capabilities to all customers, and keeping the multi-channel strategy.

4. **Excellent instant performance.**
Internet users are always demanding more speed, especially with the massive rise of mobile-first and even mobile-online usage. Research shows that 53% of visitors leave a website after just 3 seconds page load and conversion rates decrease with 21,8% on each 1 second delay in page speed.

Progressive Web Apps are by design built for extremely smooth and fast user experiences. Capitalising on the speed benefits of the web and 'native-style' client-side caching, it outperforms both on page loads. PWAs can load within just 1 second, creating a true 'instant' speed experience, engaging the user right from the start.

Additionally, Progressive Web Apps are using significantly less disk space - on both the business' server and the user's device - providing benefits such as faster loading times, less data usage and less required storage space.

5. **PWAs can improve your SEO.**

Because a PWA is web-based, everything is discoverable by search engines. Any content within a PWA can be linked to, shared and ranked by Google and Bing. This contrasts native apps, which don't allow to be crawled by search engine robots. By breaking down the barriers between web and native, users can be directly linked to the most sophisticated digital products, on any device, at any time.

Super fast loading times, reduced bouncing rates, minimum data usage and highly engaging experiences are by default boosters for SEO rankings. With PWA offering advantages like these out-of-the-box, it's logical to reason that using a PWA will automatically result in higher SEO rankings. If done right, this is absolutely true. Google will rank well-designed PWA's higher than any other regular website.

However, technical SEO is very important. It's key to make sure the technology of Progressive Web Apps is served in the right way. Progressive Web Apps make use of the flexibility of JavaScript, which means Google sees the single published page as a JavaScript site. Google crawls PWA sites just like it would crawl an AJAX or JavaScript site, but the ability of search engines to correctly discover, crawl, and accurately index content — which is heavily reliant on JavaScript — has historically been poor.

Luckily, there is no need to worry. Developers building PWAs need to be aware of how to optimize the site to ensure the page gets indexed appropriately, which can easily be done with techniques such as server side rendering — providing PWAs the right architecture to get the top SEO rankings they deserve.

6. **Increased conversion rates.**

It stands to reason that a better user experience results in higher conversion rates. If friction and frustrations are removed from an online store, it becomes easier for customers to achieve their goals - and therefore the business' goals.

PWAs presents huge opportunities to deliver better experiences, in every aspect. Everything about PWAs is faster, leaner, slicker and more efficient, and it's felt by the user. From instant background loading onto their home screen to full screen browsing, instant page loads, and minimal usage of device space. The user has a much better experience, completes their tasks, and will happily return again in the future — whether or not triggered by a highly engaging push notification.

Technical components of PWA

1. Web App Manifest:

The web app manifest is the first component of the PWA. It is a simple JSON file that controls a user's application. Usually, it is named "manifest.json". It is the most important component for the presence of PWA. When you first connect PWA to a network, a mobile browser reads the "manifest.json" file and stores it locally in cache memory.
When there is no network access in PWA, the mobile browser uses the application's cache memory to run the PWA program while offline.

The "manifest.json" file helps the PWA to give a look of a native app. With the help of the manifest.json file, the PWA developer can control how the application is presented to the user mobile screen. The PWA developer can also set a theme for the mobile's splash-screen and the application's address bar.

The "manifest.json" file allows the PWA developer to search for a centralized location for the metadata of the web application. The JSON file defines the links to icons and icon sizes, the full and abbreviated name of an app, types, background color, theme, locations, and other visual details that are required for an app-like experience.

2. Service Worker

A service worker is a web worker. It is a JavaScript file that runs aside from the mobile browser. In other words, it is another technical component that promotes the functionality of PWA. The service worker retrieves the resources from the cache memory and delivers the messages.

It is independent of the application to which they are connected, and has many consequences:

- The service worker does not block the synchronized XHR, so it cannot be used in local storage (It is designed to be completely asynchronous).
- It can receive information from the server even when the application is not running. It shows notifications in the PWA application without opening in the mobile browser.
- It cannot directly access the DOM. Therefore, the PostMessage and Message Event Listener method is used to communicate with the webpage. The PostMessage method is used to send data, and the message event listener is used to receive data.

Service worker lifecycle

The service worker lifecycle is the most complex part of the PWA. There are three stages

in the lifetime of a service worker:

- Registration
- Installation
- Activation

Framework tools:

1. **Webpack webpack-pwa-framework**

   Webpack is a static module bundler for modern JavaScript applications. Without such bundlers, running JS fdes in a browser may cause issues with loading too many scripts in large .js files. This dev tool is recommended to solve the scaling problem for complex PWA storefronts. Webpack Module Bundler aimed at building and managing dependency graphs between CSS elements, stylesheets, fonts, images, and other JavaScript assets. The graphs give developers better control over assets processing, ease code-splitting, eliminate dead assets, and reduce the server calls.

   Among the drawbacks of Webpack are unfriendly documentation, painful source code, and steep learning curve.

2. **Lighthouse : - lighthouse pwa-pwa-framework**

   Once your PWA is created, you want to measure its performance, SEO, accessibility, and other important benchmarks. For this purpose, Google provides an awesome analytic tool - Lighthouse, that offers a set of metrics to test the app to test the application and guide you in creating PWAs with an immersive app-like experience for your visitors.

   The tool significantly reduces the need to carry out manual testing to audit your web application for PWA features. Lighthouse enables developers to automate a series of tests against a given URL and fleetingly generate an exhaustive report which gives them a clue about how well the PWA is performing, as well as what bugs it has. The recorded data can be used to improve the overall performance of the newly created web application.

   Lighthouse can either be run from the Chrome DevTools (from its Audits tab) or automated through the command line, as a Node module. Alternatively, it can be installed and run as a Chrome extension.