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

CERTIFICATE

This is to certify that **Ninad Rao** 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: **Kajal Jewani**
Signature:

Head of Department
Name: **Dr. Shalu Chopra**
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 Analyze 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: **Ninad Rao**

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

Since 1962

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.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	L01
Grade:	

EXPERIMENT NO. 1
MAD and PWA Lab

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).

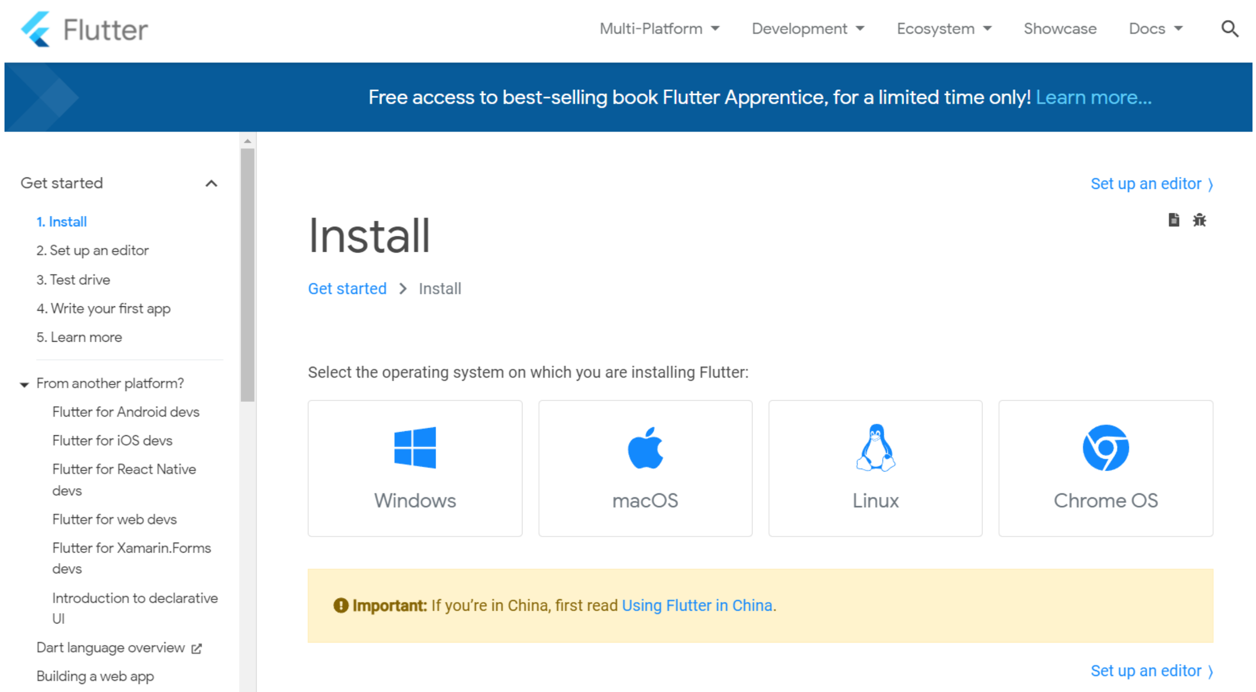
Flutter consists of two important parts:

- 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.

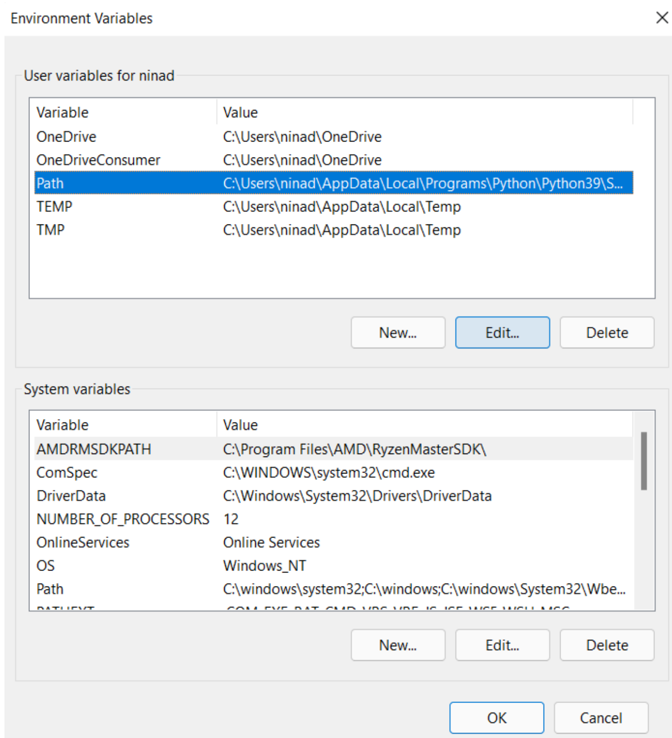


2. Next, to download the latest Flutter SDK, click on the **Windows icon**. Here, you will find the download link for SDK.

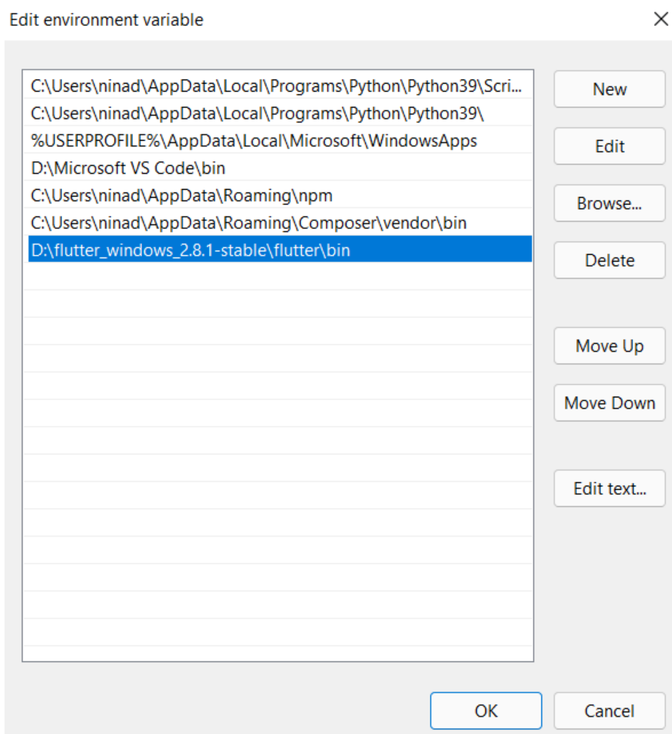
3. When your download is complete, extract the **zip file** and place it in the desired installation folder or location, for example C:/Flutter.

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**. You will get the following screen.



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:\Users\ninad>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.
```

```
                Welcome to Flutter! - https://flutter.dev

The Flutter tool uses Google Analytics to anonymously report feature usage
statistics and basic crash reports. This data is used to help improve
Flutter tools over time.

Flutter tool analytics are not sent on the very first run. To disable
reporting, type 'flutter config --no-analytics'. To display the current
setting, type 'flutter config'. If you opt out of analytics, an opt-out
event will be sent, and then no further information will be sent by the
Flutter tool.

By downloading the Flutter SDK, you agree to the Google Terms of Service.
Note: The Google Privacy Policy describes how data is handled in this
service.

Moreover, Flutter includes the Dart SDK, which may send usage metrics and
crash reports to Google.

Read about data we send with crash reports:
https://flutter.dev/docs/reference/crash-reporting

See Google's privacy policy:
https://policies.google.com/privacy
```

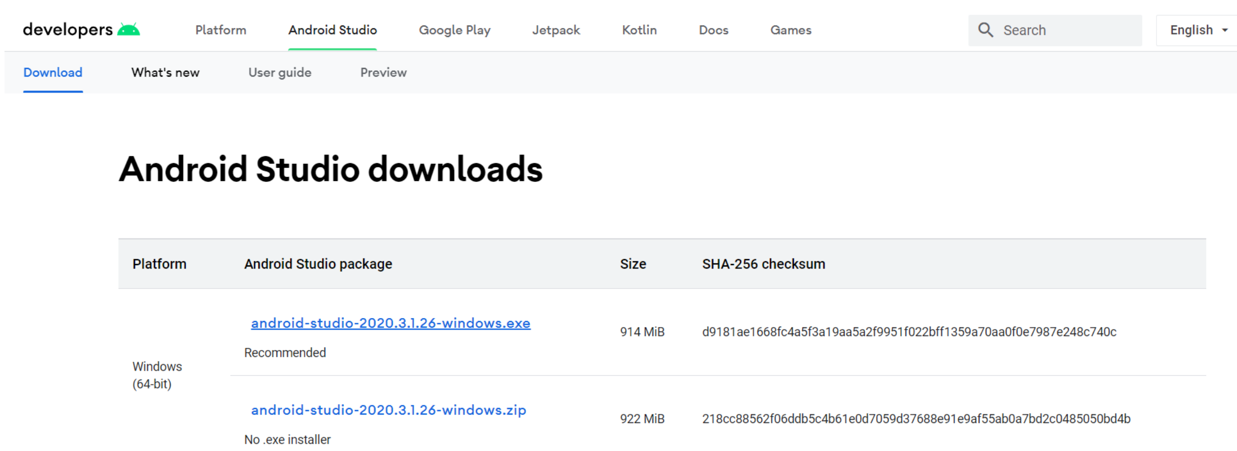
Now, run the **$ flutter doctor** command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation.

```
C:\Users\ninad>flutter doctor
Running "flutter pub get" in flutter_tools...                    12.6s
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.8.1, on Microsoft Windows [Version 10.0.22000.434], locale en-IN)
[x] Android toolchain - develop for Android devices
    x 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)
[√] 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.

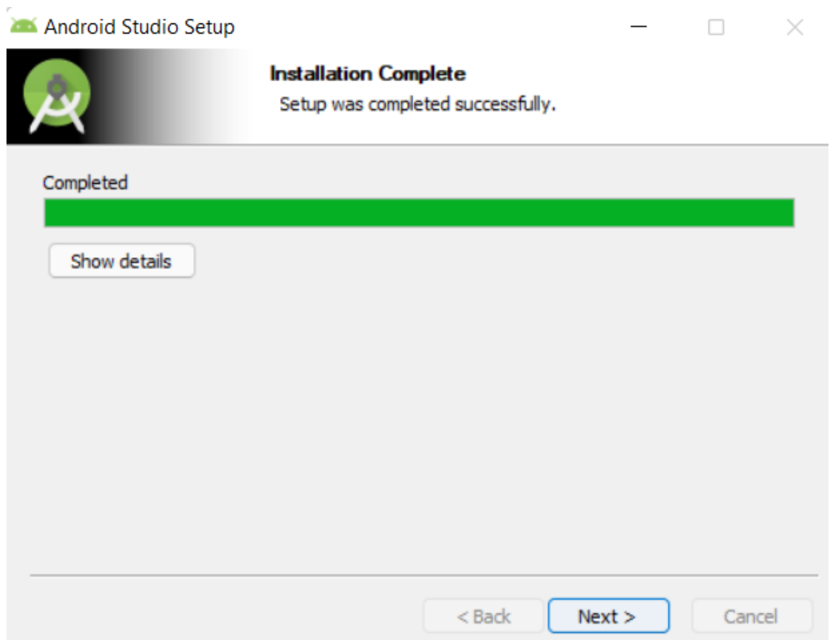
7. Install the **Android SDK**. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps:

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

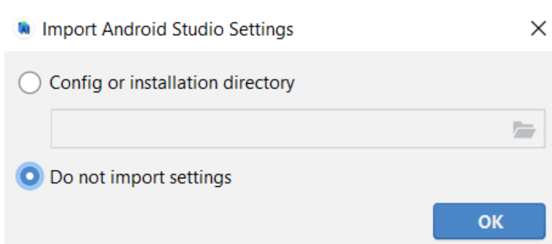


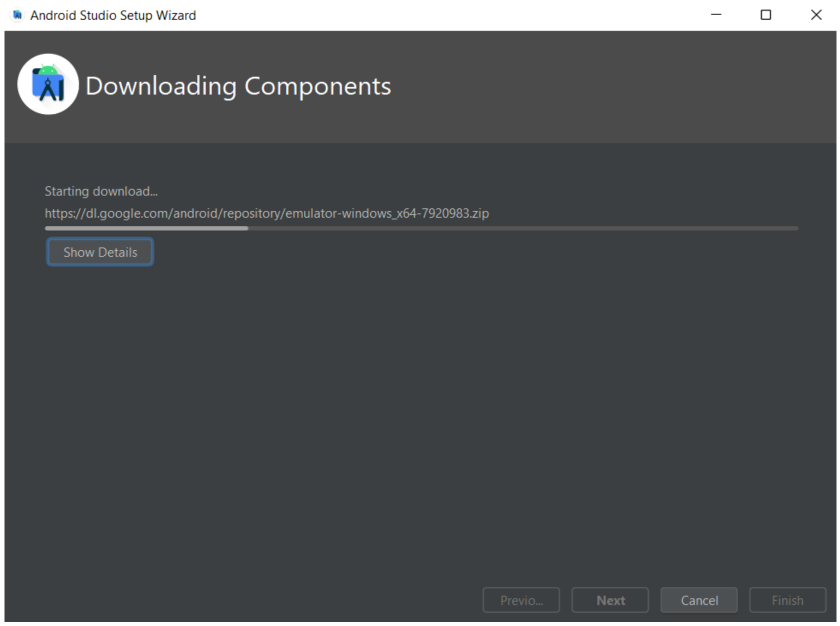
    b. When the download is complete, open the .exe file and run it. You will get the following dialog box.

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

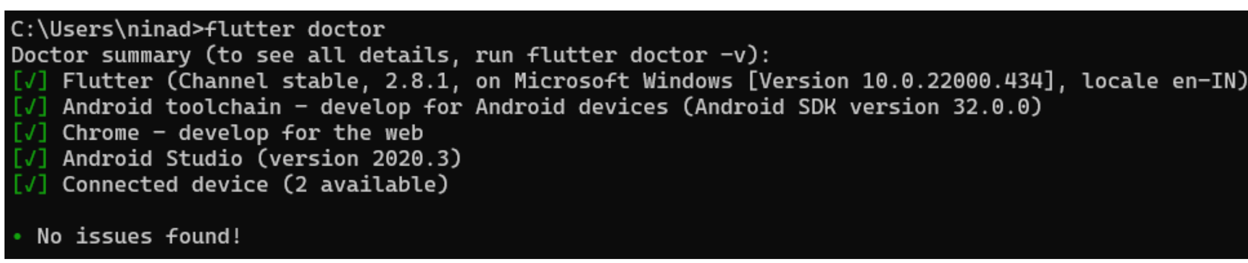


d. In the above screen, **click Next** -> **Finish**. Once the Finish button is clicked, you need to choose the '**Don't import Settings option**' and click **OK**. It will start the Android Studio.





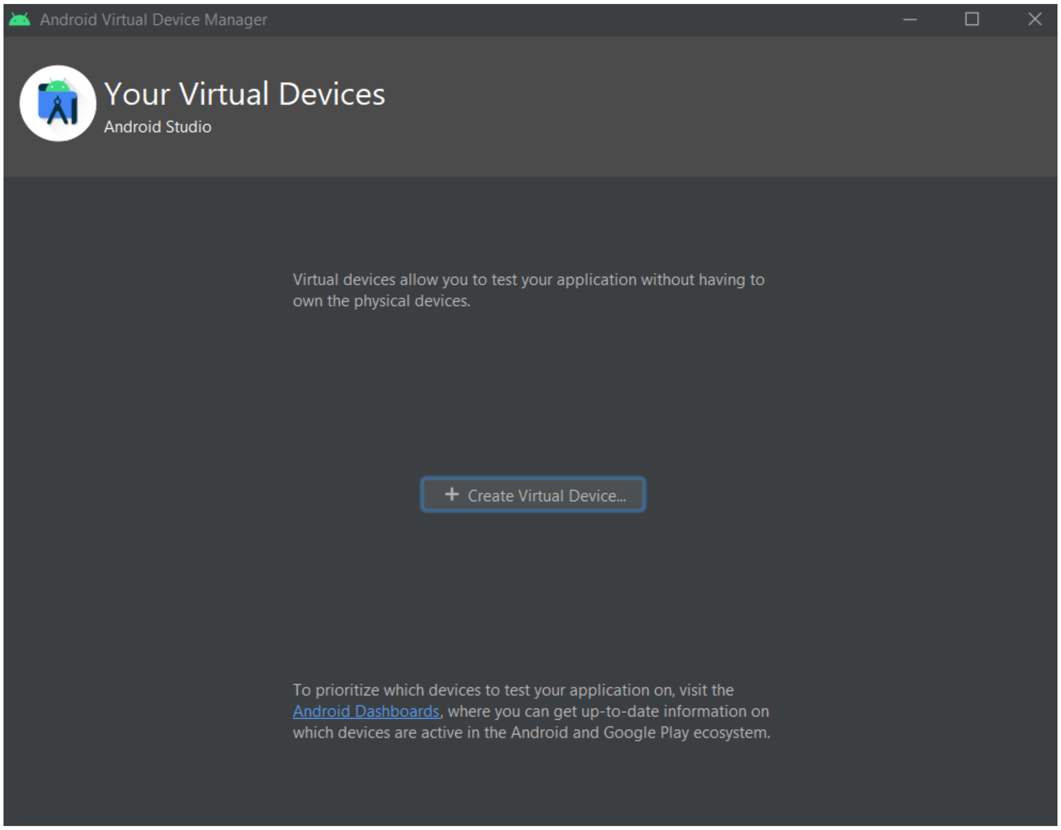
e. Run the **$ flutter doctor** command and Run **flutter doctor --android-licenses** command.



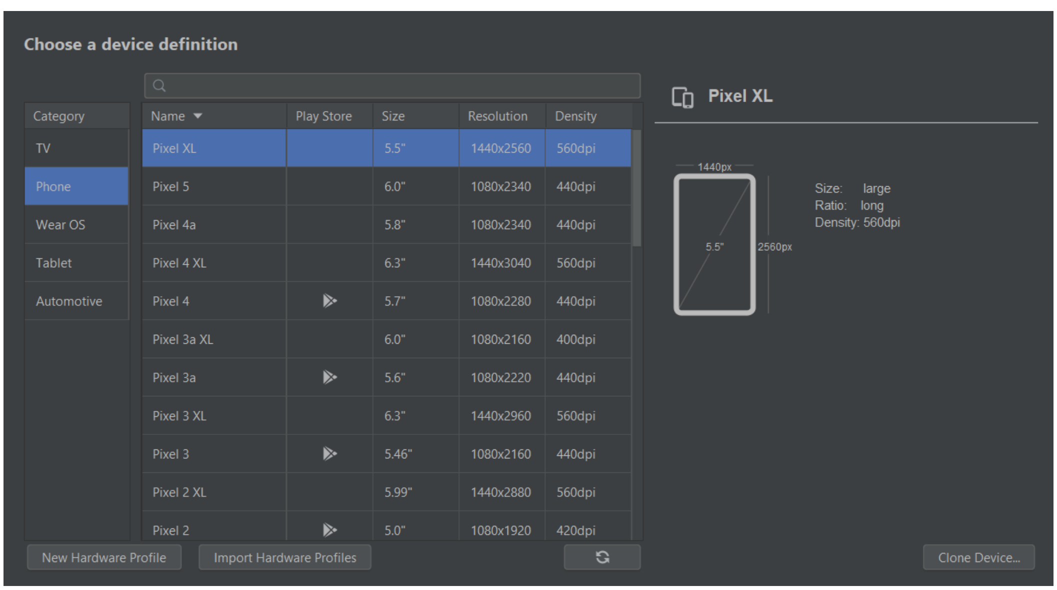
```
C:\Users\ninad>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.8.1, on Microsoft Windows [Version 10.0.22000.434], locale en-IN)
[√] Android toolchain - develop for Android devices (Android SDK version 32.0.0)
[√] Chrome - develop for the web
[√] Android Studio (version 2020.3)
[√] 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.

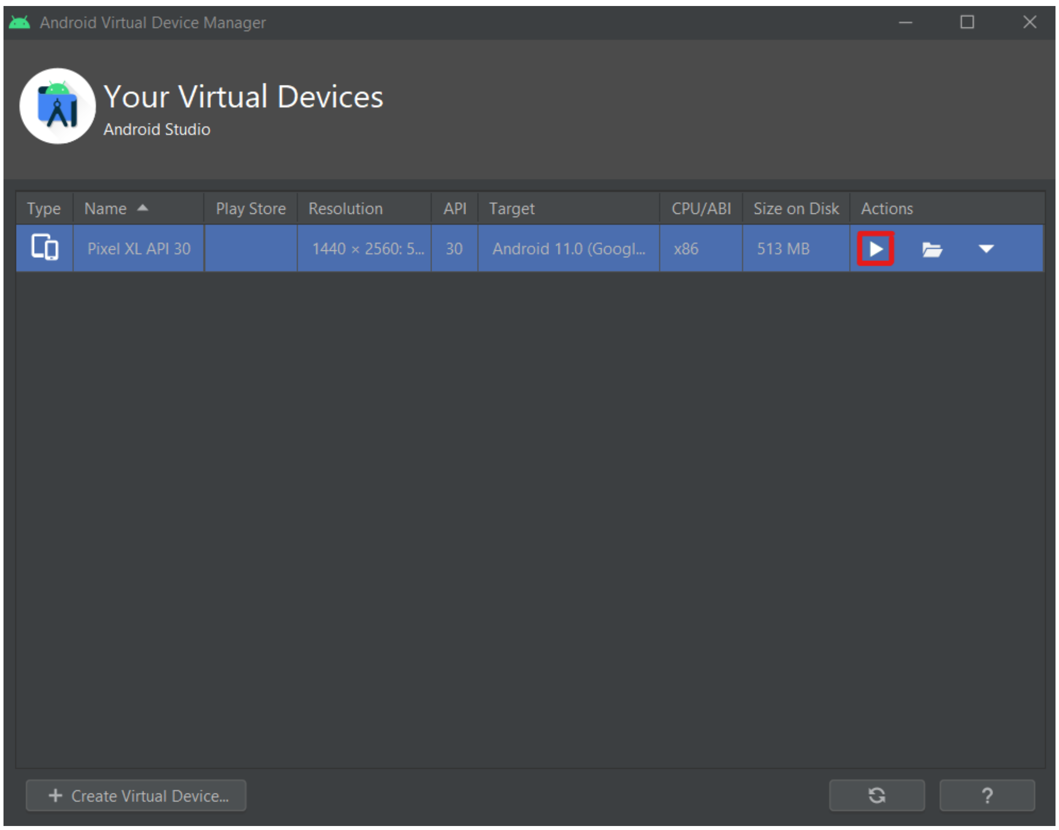


b. Choose your device definition and click on **Next**.



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.

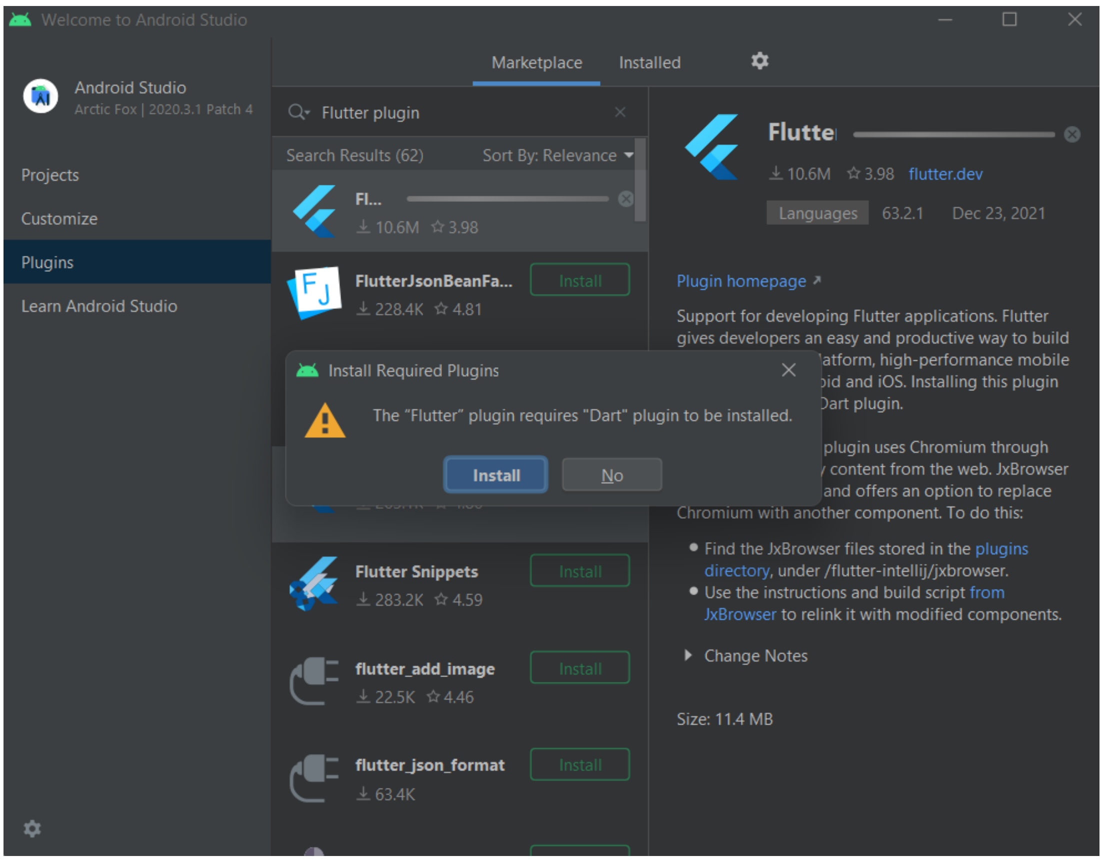


e. Last, click on the icon pointed into the **red color rectangle**. The Android emulator displayed as shown below screen.

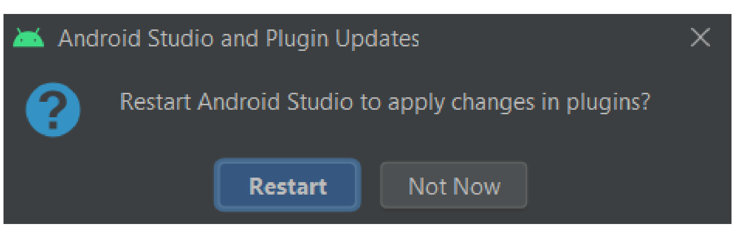
9. Now, install the **Flutter** and **Dart plugin** for building Flutter applications in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.

    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)

V.E.S. Since 1962

Department of Information Technology

MAD & PWA Lab Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	L02
Grade:	

Vivekanand Education Society's
Institute of Technology
(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

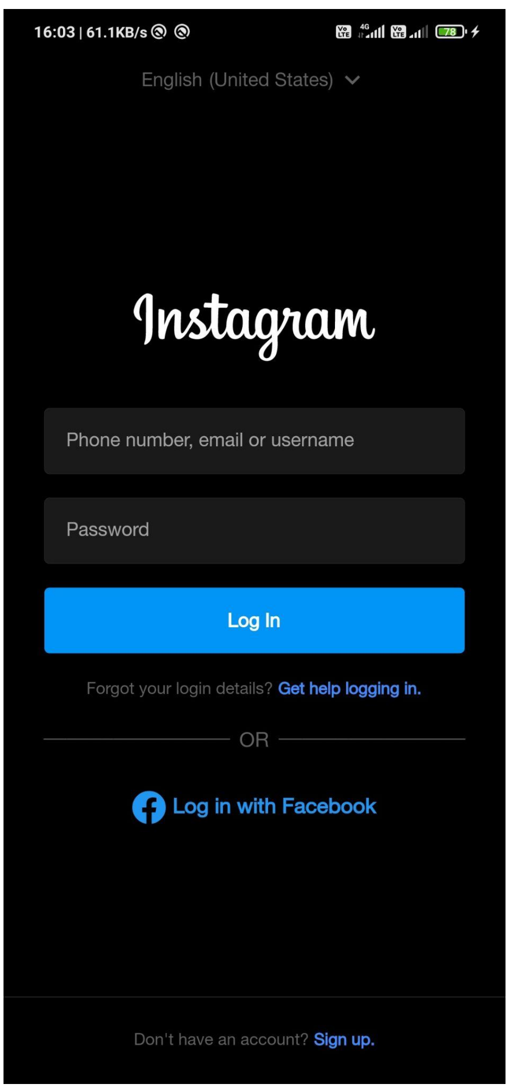
Since 1962

EXPERIMENT NO. 2
MAD and PWA Lab

Aim: To design Flutter UI by including common widgets.

Theory:

Flutter widgets are built using a modern framework that takes inspiration from React. 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 differs 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.



Flutter comes with a suite of powerful widgets, of which the following are used making the **Login Screen** of our application (Instagram) :

- Stateful
- Scaffold
- Column
- SizedBox
- Row
- Text
- Container
- TextField
- Progress Bar
- Center
- Inkwell

A **Stateful widget** is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely. They are useful when the part of the user interface you are describing can change dynamically, e.g. due to having an internal clock-driven state, or depending on some system state. Following is an example of the widget I have used in the Login Screen.

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

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

class _LoginScreenState extends State<LoginScreen> {
 @override
 Widget build(BuildContext context) {
  return Container(

  );
 }
}
```

The examples of the StatefulWidget are Checkbox, Radio, Slider, InkWell, Form, and TextField.

The **Scaffold** is a widget in Flutter used to implement the basic material design visual layout structure. It is quick enough to create a general-purpose mobile application and contains almost everything we need to create a functional and responsive Flutter app. This widget is able to occupy the whole device screen. Following is an example of the widget I have used in the Login Screen.

```
return Scaffold(
    resizeToAvoidBottomInset: false,
    body: SingleChildScrollView(
```

```
    child: Column(
      children: [],
    ),
  ),
);
```

The following are the properties and description of the Scaffold widget class.

Property	Description
appBar	It is a horizontal bar that is mainly displayed at the top of the Scaffold widget.
body	It is the other primary and required property of this widget, which will display the main content in the Scaffold.
drawer	It is a slider panel that is displayed at the side of the body.
floatingActionButton	It is a button displayed at the bottom right corner and floating above the body.
backgroundColor	This property is used to set the background color of the whole Scaffold widget.
primary	It is used to tell whether the Scaffold will be displayed at the top of the screen or not.
persistentFooterButton	It is a list of buttons that are displayed at the bottom of the Scaffold widget.
bottomNavigationBar	This property is like a menu that displays a navigation bar at the bottom of the Scaffold.
endDrawer	It is similar to a drawer property, but they are displayed at the right side of the screen by default.
resizeToAvoidBottomInset	Scaffold's floating widgets adjust their size themselves to avoid the onscreen keyboard.
floatingActionButtonLocation	By default, it is positioned at the bottom right corner of the screen.

The **Column widget** arranges its children in a vertical direction on the screen. In other words, it will expect a vertical array of children widgets. A column widget does not appear scrollable because it displays the widgets within the visible view. Flutter column widget has several other properties like mainAxisSize, textDirection, verticalDirection, etc. These are **mainAxisAlignment** and **crossAxisAlignment** properties.

Property	Description
start	It will place the children from the starting of the main axis.
end	It will place the children at the end of the main axis.
center	It will place the children in the middle of the main axis.
spaceBetween	It will place the free space between the children evenly.
spaceAround	It will place the free space between the children evenly and half of that space before and after the first and last children widget.
spaceEvenly	It will place the free space between the children evenly and before and after the first and last children widget.

We can also control how a column widget aligns its children using the property **mainAxisAlignment** and **crossAxisAlignment**. The column's cross-axis will run horizontally, and the main axis will run vertically. Following is an example of the widget I have used in the Login Screen.

```
child: Column(
        children: [],
),
```

The **SizedBox widget** is a box with a specified size. If given a child, this widget forces it to have a specific width and/or height. These values will be ignored if this widget's parent does not permit them.

In the Login Screen, I have used the SizedBox as a vertical space between the widgets which are placed in the screen. Following is an example of the widget I have used in the Login Screen.

```
SizedBox(
        height: 44,
),
```

The following are the properties and description of the SizedBox widget class.

Property	Description
width	If non-null, requires the child to have exactly this width.
height	If non-null, requires the child to have exactly this height.

The **Row widget** arranges its children in a horizontal direction on the screen. A row widget does not appear scrollable because it displays the widgets within the visible view. Flutter row widget has several other properties like mainAxisSize, textDirection, verticalDirection, etc. These are **mainAxisAlignment** and **crossAxisAlignment** properties.

Property	Description
start	It will place the children from the starting of the main axis.
end	It will place the children at the end of the main axis.
center	It will place the children in the middle of the main axis.
spaceBetween	It will place the free space between the children evenly.
spaceAround	It will place the free space between the children evenly and half of that space before and after the first and last children widget.
spaceEvenly	It will place the free space between the children evenly and before and after the first and last children widget.

We can control how a row widget aligns its children based on our choice using the property crossAxisAlignment and mainAxisAlignment. The row's cross-axis will run vertically, and the main axis will run horizontally. Following is an example of the widget I have used in the Login Screen.

```
Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: const <Widget>[
        Text("English",
                style: TextStyle(color: Color(0xff616161), fontSize: 16.0)),
        SizedBox(
                width: 5.0,
        ),
        Text("(United States)",
                style: TextStyle(color: Color(0xff616161), fontSize: 16.0)),
        SizedBox(
                width: 2.0,
        ),
        Icon(
                Icons.expand_more,
                color: Color(0xff616161),
        ),
        ],
),
```

I have used the Row widget for every widget placement in the screen like the TextField, Button etc. A **Text widget** in Flutter that allows us to display a string of text with a single line in our application. If we do not specify any styling to the text widget, it will use the closest DefaultTextStyle class style. Following is an example of the widget I have used in the Login Screen.

```
Text("Log in with Facebook",
        style: TextStyle(
                color: Colors.blue,
        fontWeight: FontWeight.bold,
        fontSize: 18,
```

```
        ),
),
```

The following are the essential properties of the Text widget used in our application :

Property	Description
fontWeight	It determines the thickness of the text.
fontSize	It determines the size of the text.
fontFamily	It is used to specify the typeface for the font.
fontStyle	It is used to style the font either in bold or italic form.
Color	It is used to determine the color of the text.

I have used numerous Text widgets in the Login Screen such as “English (United States)”, “Log In”, “Forgot your login details? Get help logging in.”, “Don’t have an account? Sign up.” etc.

The **Container widget** in Flutter is a parent widget that can contain multiple child widgets and manage them efficiently through width, height, padding, background color, etc. It is a widget that combines common painting, positioning, and sizing of the child widgets. It is also a class to store one or more widgets and position them on the screen according to our needs. Following is an example of the widget I have used in the Login Screen.

```
Container(
        child: const Text(
        "Don't have an account? ",
        style: TextStyle(
                color: Color(0xff616161),
                ),
        ),
        padding: const EdgeInsets.symmetric(
                vertical: 8,
        ),
```

```
),
```

The following are the essential properties of the Container widget :

Property	Description
child	This property is used to store the child widget of the container.
color	This property is used to set the background color of the text.
height and width	This property is used to set the container's height and width according to our needs.
margin	This property is used to surround the empty space around the container.
padding	This property is used to set the distance between the border of the container (all four directions) and its child widget.
alignment	This property is used to set the position of the child within the container.
decoration	This property allows the developer to add decoration on the widget.
transform	The transform property allows developers to rotate the container.
constraints	This property is used when we want to add additional constraints to the child.

I have used numerous Container widgets in the Login Screen to keep Text widget, Inkwell widget, Icon widget, TextField widget etc.

A **TextField widget** is an input element which holds the alphanumeric data, such as name, password, address, etc. It is a GUI control element that enables the user to enter text information using a programmable code. It can be of a single-line text field or multiple-line text field.

TextField in Flutter is the most commonly used text input widget that allows users to collect inputs from the keyboard into an app. We can use the TextField widget in building forms, sending messages, creating search experiences, and many more. By default, Flutter decorated the TextField with an underline.

```
TextField(
```

```
    cursorColor: Colors.white,
    cursorWidth: 1,
    controller: textEditingController,
    decoration: InputDecoration(
        hintText: hintText,
        border: inputBorder,
        focusedBorder: inputBorder,
        enabledBorder: inputBorder,
        filled: true,
        contentPadding: const EdgeInsets.all(17),
    ),
    keyboardType: textInputType,
    obscureText: isPass,
);
```

The following are the essential properties of the TextField widget used in our application :

Property	**Description**
cursorColor	It is used to show the color of the cursor that indicates the current location of the text insertion point in the field.
cursorWidth	It is used to show how thick the cursor will be.
controller	It is a popular method to retrieve text field value using TextEditingController. It will be attached to the TextField widget and then listen to change and control the widget's text value.
decoration	It is used to show the decoration around TextField.
hintText	It is used to show the hint text inside TextField.
keyboardType	It is used to show the type of keyboard to use for editing the text.
obscureText	It is used to hide the text being edited. When this is set to true, all the characters in the text field are replaced by *.

I have used the TextField widget for the input of Email ID and Password fields in the Login Screen. For the password field, I have kept the obscureText true to hide the text which is to be entered.

A **Progress Bar widget** is a graphical control element used to show the progress of a task such as downloading, uploading, installation, file transfer, etc. I have used **CircularProgressIndicator** for Log In button when clicked.

It is a widget, which spins to indicate the waiting process in your application. It shows the progress of a task in a circular shape. It also displays the progress bar in two ways :

1. A **determinate** progress bar is used when we want to show the progress of ongoing tasks such as the percentage of downloading or uploading files, etc. We can show the progress by specifying the value between 0.0 and 1.0.
2. An **indeterminate** progress bar is used when we do not want to know the percentage of an ongoing process. By default, CircularProgressIndicator shows the indeterminate progress bar.

```
child: _isLoading
        ? const Center(
                child: CircularProgressIndicator(
        color: primaryColor,
        ),
        )
        : const Text('Log In',
        style: TextStyle(
                fontSize: 16.0, fontWeight: FontWeight.bold)),
```

Over here, I have used the indeterminate progress bar, which will end once the user logs in and the screen changes from Login Screen to Home Screen.

The **Center widget** will center its child within itself. This widget will be as big as possible if its dimensions are constrained and **widthFactor** and **heightFactor** are null. If a dimension is unconstrained and the corresponding size factor is null then the widget will match its child's size in that dimension.

```
Center(
        child: CircularProgressIndicator(
        color: primaryColor,
),
```

The following are the properties of the Center widget :

Property	Description
alignment	It is used to show how to align the child.
widthFactor	If non-null, sets its width to the child's width multiplied by this factor.
heightFactor	If non-null, sets its height to the child's width multiplied by this factor.

I have used the Center widget to center the progress bar used in the Log In button of the Login Screen.

Buttons are the graphical control element that provides a user to trigger an event such as taking actions, making choices, searching things, and many more. They can be placed anywhere in our UI like dialogs, forms, cards, toolbars, etc.

InkWell widget button is a material design concept, which is used for touch response. This widget comes under the Material widget where the ink reactions are actually painted. It creates the app UI interactive by adding gesture feedback. It is mainly used for adding a splash ripple effect.

```
child: InkWell(
        onTap: loginUser,
        child: Container(
                child: _isLoading
                ? const Center(
                child: CircularProgressIndicator(
                        color: primaryColor,
```

```
                    ),
                    )
                    : const Text('Log In',
                    style: TextStyle(
                              fontSize: 16.0, fontWeight: FontWeight.bold)),
                    width: double.infinity,
                    alignment: Alignment.center,
                    padding: const EdgeInsets.symmetric(vertical: 17),
                    decoration: const ShapeDecoration(
                    shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.all(Radius.circular(4)),
          ),
          color: blueColor),
          ),
),
```

I have used the Inkwell widget to make the Log In button in the Login Screen.

Conclusion: Hence, we understood how to design Flutter UI by including the widgets that have been used in making the **Login Screen** of our application (Instagram).

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

MAD & PWA Lab Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	L02
Grade:	

EXPERIMENT NO. 3
MAD and PWA Lab

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

Theory:

An **Icon** is a graphic image representing an application or any specific entity containing meaning for the user. It can be selectable and unselectable. Flutter provides an **Icon widget** to create icons in our applications. We can create icons in Flutter, either using inbuilt icons or with the custom icons. Flutter provides the list of all icons in the Icons class.

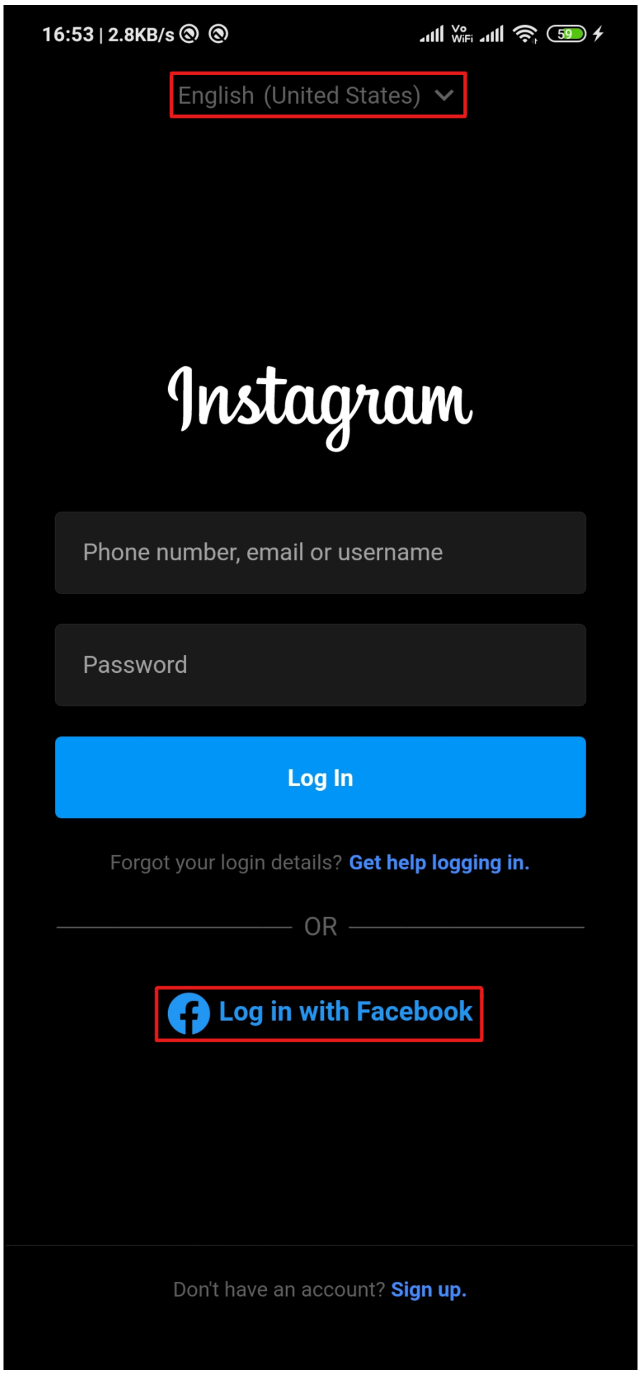
To use this class, make sure you set **uses-material-design: true** in your project's **pubspec.yaml** file in the flutter section. This ensures that the Material Icons font is included in your application. This font is used to display the icons.

Property	Description
icon	It is used to specify the icon name to display in the application. Generally, Flutter uses material design icons that are symbols for common actions and items.
color	It is used to specify the color of the icon.
size	It is used to specify the size of the icon in pixels. Usually, icons have equal height and width.
textDirection	It is used to specify to which direction the icon will be rendered.

In the following **Login Screen** of the application (Instagram), I have used two Icons.

```
Icon(
        Icons.expand_more,                    // Icon on the top of the screen
    color: Color(0xff616161),
),
Icon(
        FontAwesomeIcons.facebook,                    // Icon below the OR text
```

```
        color: Colors.blue,
        size: 27,
),
```



The first **icon** selected is expand_more which is a downward arrow icon. The expand_more icon name is selected from the list of the icons that Flutter provides.

The **color** selected is a hex code of 0xff616161 which is a light grey color.

The second **icon** selected is facebook which is a facebook logo. The facebook icon is selected from the **FontAwesomeIcons** list which is imported from the package font_awesome_flutter which is installed separately.

The **color** selected is blue which is from the colors.dart file provided by the Flutter.

The **size** provided to the icon is 27 pixels.

Displaying **Images** is the fundamental concept of most of the mobile apps. Flutter has an **Image widget** that allows displaying different types of images in the mobile application.

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).

How to display the image in Flutter:
To display an image in Flutter, do the following steps :

1. First, we need to create a new folder inside the root of the Flutter project and name it assets. We can also give it any other name if you want.
2. Next, inside this folder, add an image manually.
3. Update the pubspec.yaml file. Suppose the image name is image1.jpg, then pubspec.yaml file will be:
```
assets:
  - assets/image1.jpg
```

Displaying images from the internet or network is very simple. Flutter provides a built-in method **Image.network** to work with images from a URL. The Image.network method also allows you to use some optional properties, such as height, width, color, fit, and many more.

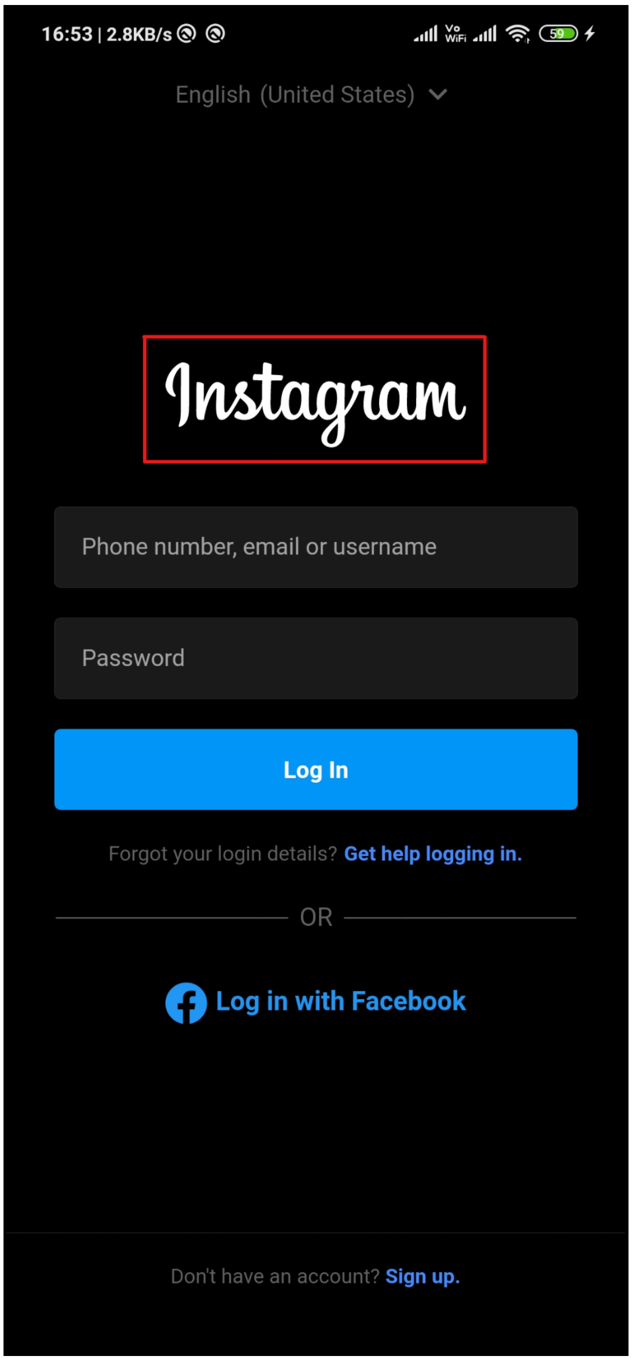
When we display an image, it simply pops onto the screen as they are loaded. It does not assume usefulness between the users. We can display an image Fade-In Images to overcome this issue.

The Image uses a **FadeInImage widget** that shows a placeholder image while the target image is loading, then fades in the new image when it loads. The FadeInImage can work with various types of images, such as local assets, in-memory, or images from the internet.

In the following **Login Screen** of the application (Instagram), I have used a SVG image to display. It is similar to the **Image** class.

```
SvgPicture.asset(
        'assets/ic_instagram.svg',
```

```
        color: primaryColor,
        height: 54,
),
```



First I have used a **SVG image** to display in the center of the screen. Therefore I have made a folder called **assets** and then added this image in that folder.

Next, I updated the **pubspec.yaml** file where I uncommented the line of assets and added the following line :

```
# To add assets to your application, add an assets section, like this:
 assets:
   - assets/ic_instagram.svg
```

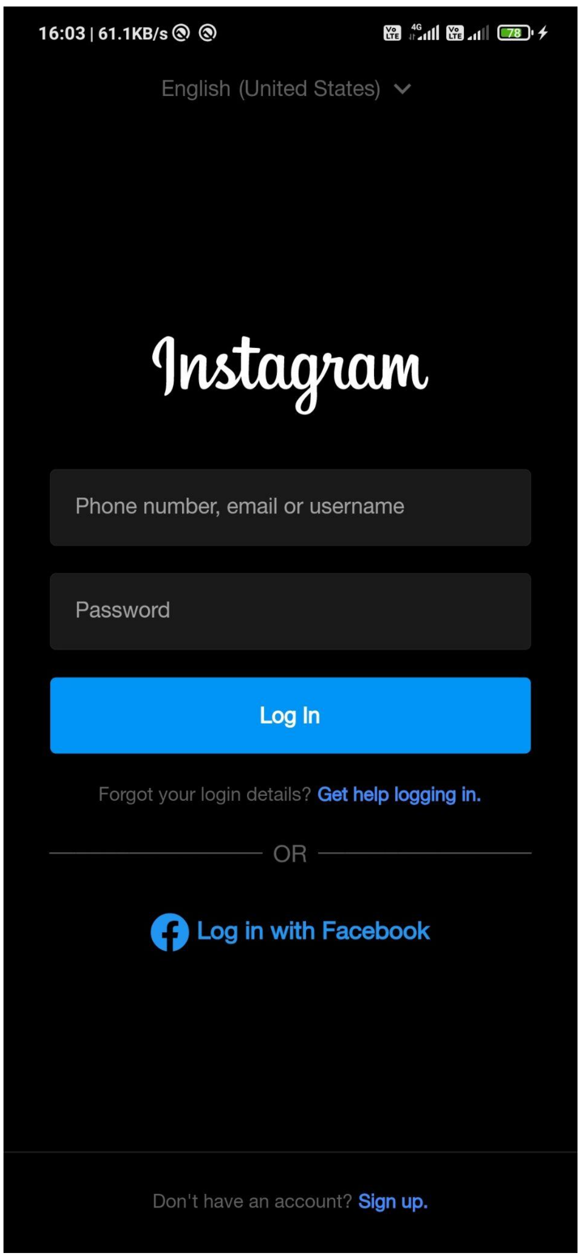
After this, I used the class **SvgPicture.asset()**. This widget will parse SVG data into a Picture using a PictureProvider. To use this widget, we need to install and import the package flutter_svg in the login screen dart file.

The **color** selected is **primaryColor** which is called from the colors.dart file which is provided by the Flutter. So primaryColor is white color.

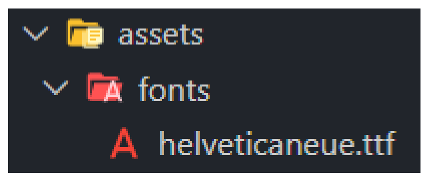
The **height** selected is 54 pixels. The **width** is kept by default as the image has.

Although Android and iOS offer high quality system **fonts**, one of the most common requests from designers is for custom fonts. Flutter works with **custom fonts** and you can apply a custom font across an entire app or to individual widgets.

In the following **Login Screen** of the application (Instagram), I have used Helvetica Neue font everywhere.



To work with a font, **import the font files** into the project. It's common practice to put font files in a fonts or assets folder at the root of a Flutter project.



Flutter supports the following font formats :

- .ttf
- .otf

Once you've identified a font, tell Flutter where to find it. You can do this by including a **font definition** in the **pubspec.yaml** file.

```
fonts:
  - family: Helvetica
    fonts:
      - asset: assets/fonts/helveticaneue.ttf
```

You have two options for how to apply fonts to text :

1. Default font for everything
2. Font only within specific widgets

To use a **font as the default**, set the fontFamily property as part of the app's theme. The value provided to fontFamily must match the family name declared in the pubspec.yaml.

```
theme: ThemeData(
        fontFamily: 'Helvetica',
        brightness: Brightness.dark,
        scaffoldBackgroundColor: mobileBackgroundColor,
),
```

If you want to apply the **font to a specific widget**, such as a Text widget, provide a TextStyle to the widget.

```
child: Text(
        'This font is Helvetica Neue',
        style: TextStyle(
        fontFamily: 'Helvetica',
        ),
),
```

I have used the font as default for my application.

Conclusion: Hence, we understood how to include icons, images, fonts that have been used in making the Login Screen of our application (Instagram).

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.	To create an interactive Form using form widget
Roll No.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	L02
Grade:	

EXPERIMENT NO. 4
MAD and PWA Lab

Aim: To create an interactive Form using a form widget.

Theory:

Apps often require users to enter information into a **Text Field**. For example, you might require users to log in with an Email Address and Password combination.

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.

First, create a Form. The **Form widget** acts as a container for grouping and validating multiple form fields. When creating the form, provide a **GlobalKey**. This uniquely identifies the Form, and allows validation of the form in a later step.

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

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

 @override
 Widget build(BuildContext context) {
  final _formKey = GlobalKey<FormState>();

  return Form(
   key: _formKey,
   child: TextFormField(
   ),
  );
 }
}
```

Although the Form is in place, it doesn't have a way for users to enter text. That's the job of a TextFormField. The **TextFormField widget** renders a material design text field and can display validation errors when they occur.

Validate the input by providing a **validator()** function to the TextFormField. If the user's input isn't valid, the validator function returns a String containing an error message. If there are no errors, the validator must return null. For this example, create a validator that ensures the TextFormField isn't empty. If it is empty, return a friendly error message.

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

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

 @override
 Widget build(BuildContext context) {
  final _formKey = GlobalKey<FormState>();

  return Form(
   key: _formKey,
   child: TextFormField(
    validator: (value) {
     if (value == null || value.isEmpty) {
      return 'Please enter the respective details';
     }
     return null;
    },
   ),
  );
 }
}
```

Now that you have a form with a text field, provide a **Button** that the user can tap to submit the information.

When the user attempts to submit the form, check if the form is valid. If it is, display a success message. If it isn't (the text field has no content) display the error message.

First of all, I have created a separate class for **TextInputField** which can be reused whenever there is a form and the user has to enter the details. These are the properties I have used while creating the Input Field.

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

class TextFieldInput extends StatelessWidget {
 final TextEditingController textEditingController;
 final bool isPass;
 final String hintText;
 final TextInputType textInputType;
 const TextFieldInput({
  Key? key,
  required this.textEditingController,
  this.isPass = false,
  required this.hintText,
  required this.textInputType,
 }) : super(key: key);

 @override
 Widget build(BuildContext context) {
  final inputBorder = OutlineInputBorder(
   borderSide: Divider.createBorderSide(context),
  );

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

```
  return Form(
    key: _formKey,
    child: TextFormField(
      validator: (String? value) {
        if (value == null || value.isEmpty) {
          return 'Please enter the respective details';
        }
        return null;
      },
      cursorColor: Colors.white,
      cursorWidth: 1,
      controller: textEditingController,
      decoration: InputDecoration(
        hintText: hintText,
        border: inputBorder,
        focusedBorder: inputBorder,
        enabledBorder: inputBorder,
        filled: true,
        contentPadding: const EdgeInsets.all(
          17,
        ),
      ),
      keyboardType: textInputType,
      obscureText: isPass,
    ),
  );
 }
}
```

The essential properties I have used are :

Property	Description
cursorColor	It shows the color of the cursor. The cursor indicates the current location of the text insertion point in the field.
cursorWidth	It shows how thick the cursor will be. It defaults to 2.0 pixels.
controller	It controls the text being edited. If null, this widget will create its own TextEditingController.
decoration	It controls the BoxDecoration of the box behind the text input. It defaults to having a rounded rectangle grey border and can be null to have no box decoration.
hintText	It is displayed on top of the child of InputDecorator when the input isEmpty and either labelText is null or the input has the focus.
border	It is the shape of the border to draw around the decoration's container.
focusedBorder	It is the border to display when the InputDecorator has the focus and is not showing an error.
enabledBorder	It is the border to display when the InputDecorator is enabled and is not showing an error.
filled	If it is true, the decoration's container is filled with fillColor.
contentPadding	It is the padding for the input decoration's container. The decoration's container is the area which is filled if filled is true and bordered per the border.
keyboardType	It shows the type of keyboard to use for editing the text. It defaults to TextInputType.text if maxLines is one and TextInputType.multiline otherwise.
obscureText	It is used to hide the text being edited (e.g., for passwords). When this is set to true, all the characters in the text field are replaced by * .

I have used the **TextFieldInput** class in the **Login Screen** and **Sign Up Screen**. The code in Login Screen is as follows :

```
// Text field input for email
Container(
        padding: const EdgeInsets.symmetric(horizontal: 32),
        child: TextFieldInput(
        hintText: 'Phone number, email or username',
        textInputType: TextInputType.emailAddress,
        textEditingController: _emailController,
        ),
),

const SizedBox(
        height: 19,
),

// Text field input for password
Container(
        padding: const EdgeInsets.symmetric(horizontal: 32),
        child: TextFieldInput(
        hintText: 'Password',
        textInputType: TextInputType.text,
        textEditingController: _passwordController,
        isPass: true,
        ),
),

const SizedBox(
        height: 19,
),
```

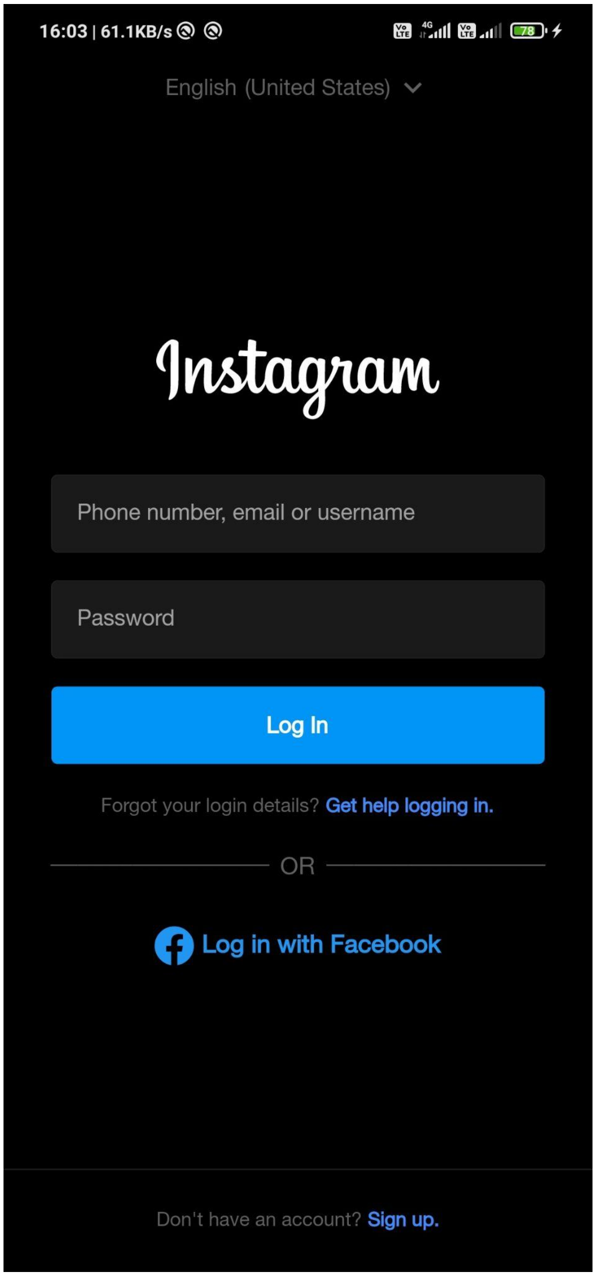
Now, adding the validation part to the form. To validate the form, use the **_formKey** created in Step 1. You can use the **_formKey.currentState()** method to access the **FormState**, which is automatically created by Flutter when building a Form.

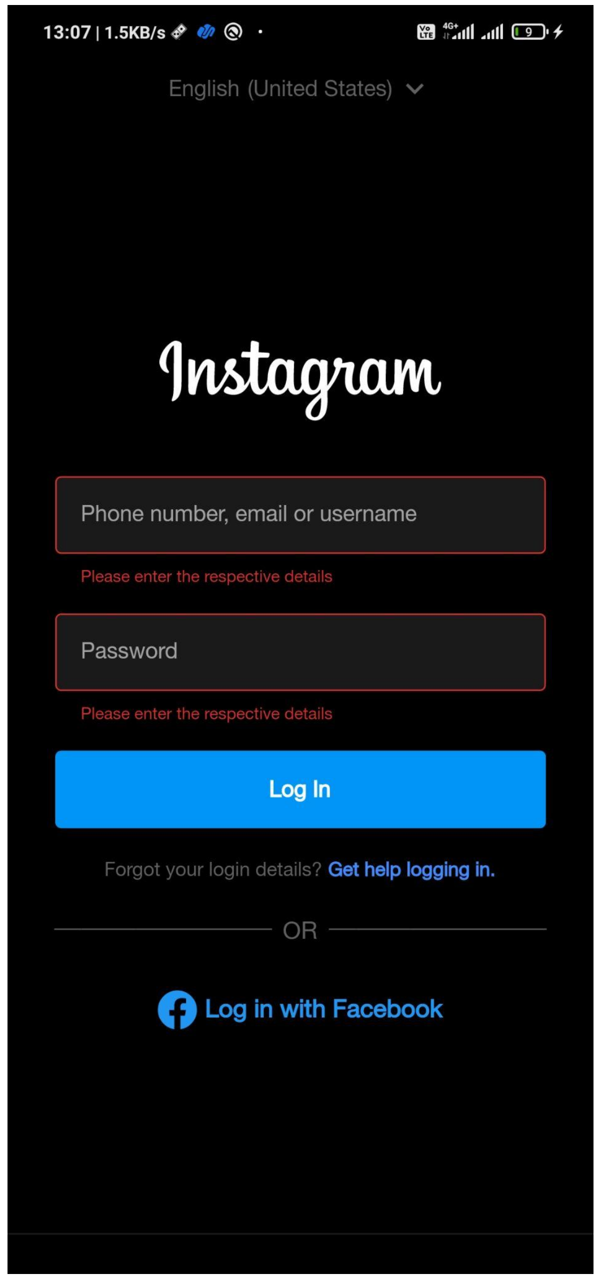
The **FormState** class contains the **validate()** method. When the validate() method is called, it runs the **validator()** function for each text field in the form. If everything looks good, the validate() method returns true. If any text field contains errors, the validate() method rebuilds the form to display any error messages and returns false.

```
Form(
    key: _formKey,
    child: Column(
    children: [
        // Text field input for email
        Container(
        padding: const EdgeInsets.symmetric(
        horizontal: 32,
        ),
        child: TextFieldInput(
        hintText: 'Phone number, email or username',
        textInputType: TextInputType.emailAddress,
        textEditingController: _emailController,
        ),
    ),

    const SizedBox(
        height: 19,
    ),

    // Text field input for password
    Container(
        padding: const EdgeInsets.symmetric(
```

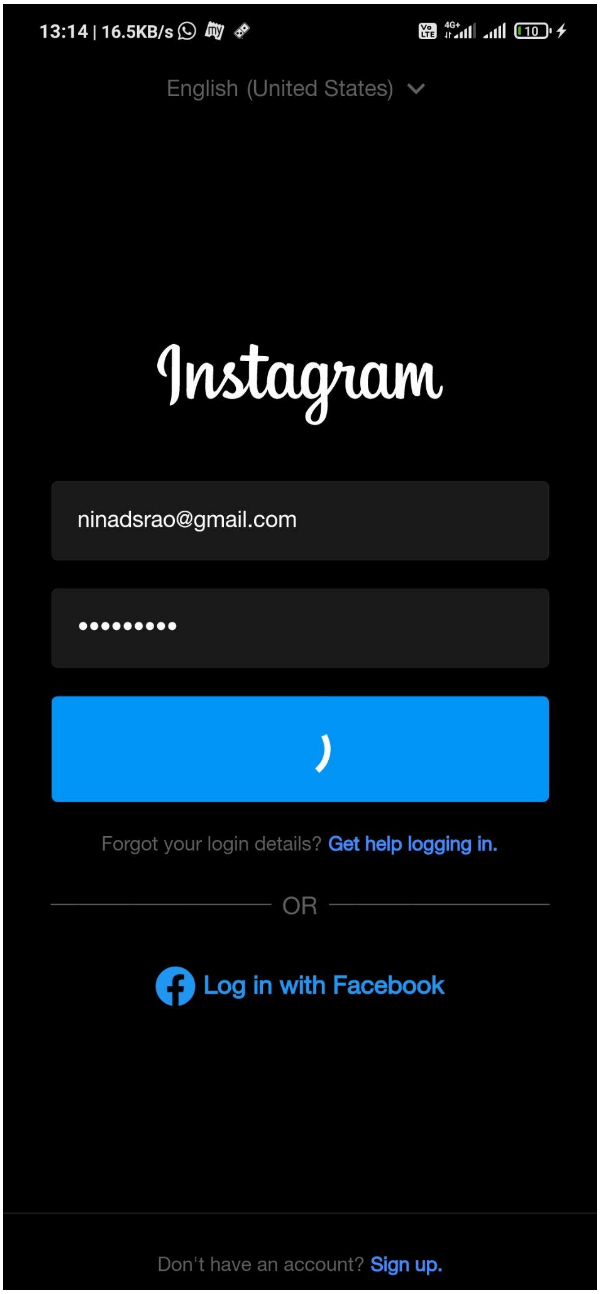
```
                horizontal: 32,
            ),
            child: TextFieldInput(
            hintText: 'Password',
            textInputType: TextInputType.text,
            textEditingController: _passwordController,
            isPass: true,
            ),
    ),

    const SizedBox(
            height: 19,
    ),

            // Button Login
    Container(
            padding: const EdgeInsets.symmetric(
            horizontal: 32,
            ),
            child: InkWell(
            onTap: () async {
                if (_formKey.currentState!.validate()) {
                    loginUser;
                }
            },
            child: Container(
                child: _isLoading
                    ? const Center(
                    child: CircularProgressIndicator(
                        color: primaryColor,
                    ),
```

```
                              )
                                        : const Text(
                                        'Log In',
                                        style: TextStyle(
                                                  fontSize: 16.0,
                                                  fontWeight: FontWeight.bold,
                                        ),
                              ),
                              width: double.infinity,
                              alignment: Alignment.center,
                              padding: const EdgeInsets.symmetric(
                                        vertical: 17,
                              ),
                              decoration: const ShapeDecoration(
                                        shape: RoundedRectangleBorder(
                                        borderRadius: BorderRadius.all(
                                                  Radius.circular(4),
                                        ),
                                        ),
                                        color: blueColor),
                              ),
                    ),
                    ),
          ],
          ),
),
```





The above screens are the **Login Screen** of my application. The first screen is the normal Login Screen where it has two input fields for Email Address and Password. The second screen appears like this when the user doesn't enter the details or inputs in the **TextField** of Email Address or Password. This is done by validating the fields using the global form key.



Once the user enters the correct details, the Log In button changes to the **CircularProgressIndicator** which is used as a widget over here. After that, the user is logged in.

Conclusion: Hence, we understood how to create an interactive Form using a form widget that has been used in making the Login Screen of our application (Instagram).

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

MAD & PWA Lab Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	L02
Grade:	

EXPERIMENT NO. 5
MAD and PWA Lab

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, my app can have a screen that contains a login button. When the user taps on that button, immediately it will display the home screen of the app.

Flutter has an imperative routing mechanism, the Navigator widget, and a more idiomatic declarative routing mechanism (which is similar to build methods as used with widgets), the Router widget. The two systems can be used together (indeed, the declarative system is built using the imperative system). Typically, small applications are served well by just using the Navigator API, via the MaterialApp constructor's MaterialApp.routes property.

In any mobile app, navigating to different pages defines the workflow of the application, and the way to handle the navigation is known as Routing. Flutter provides a basic routing class MaterialPageRoute and two methods **Navigator.push()** and **Navigator.pop()** that shows how to navigate between two routes. The following steps are required to start navigation in your application.

1. First, you need to create two routes.
2. Then, navigate to one route from another route by using the Navigator.push() method.
3. Finally, navigate to the first route by using the Navigator.pop() method.

Create two routes

Here, we are going to create two routes for navigation. In both routes, we have created only a single button. When we tap the button on the Login Screen, it will navigate to the Home Screen. Again, when we tap the button on the Home Screen, it will return to the Login Screen.

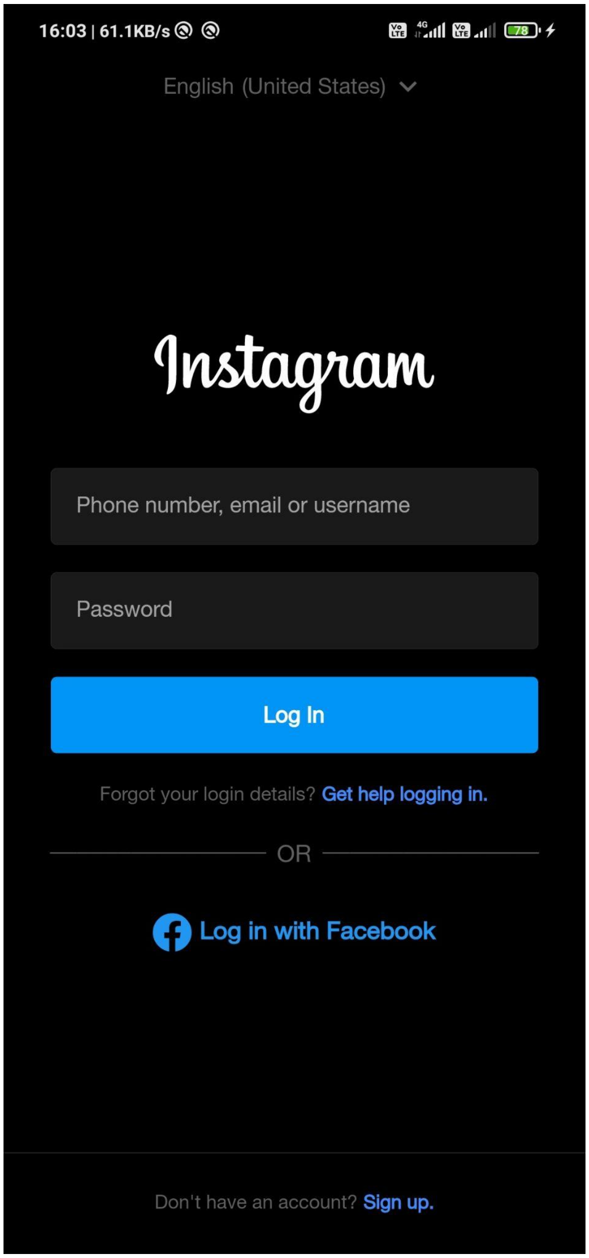
The **Navigator.push()** method is used to navigate/switch to a new route/page/screen. Here, the push() method adds a page/route on the stack and then manages it by using the Navigator. Again we use the MaterialPageRoute class that allows transition between the routes using a platform-specific animation.

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

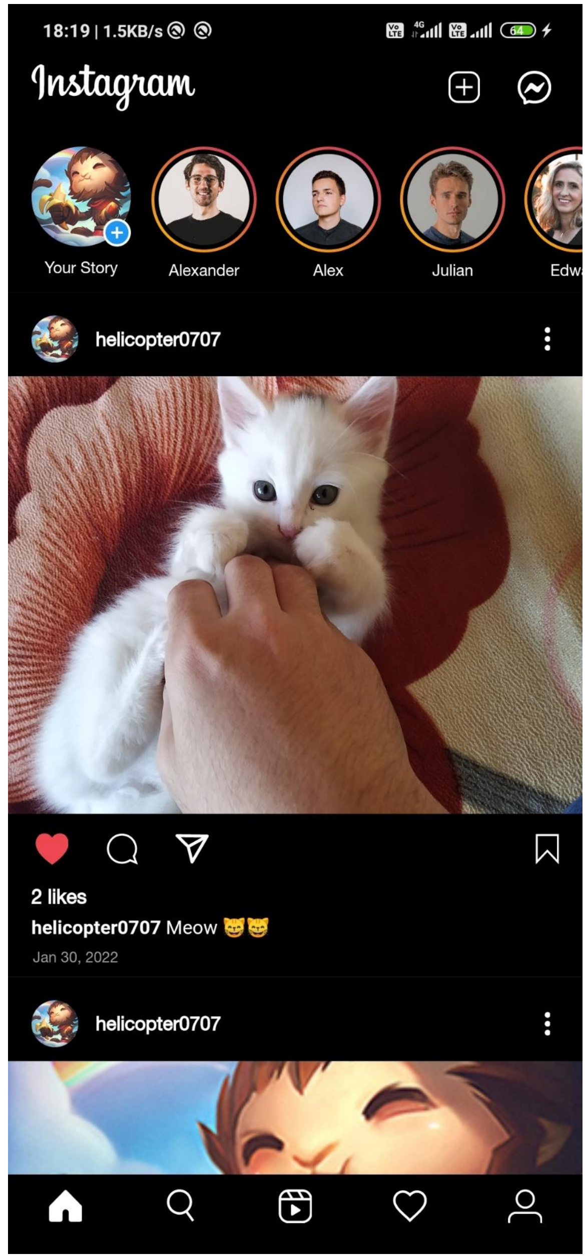
Now, we need to use the **Navigator.pop()** method to close the second route and return to the first route. The pop() method allows us to remove the current route from the stack, which is managed by the Navigator. To implement a return to the original route, we need to update the onPressed() callback method in the HomeScreen route widget.

```
onPressed: () {
        Navigator.pop(
                context,
                MaterialPageRoute(
                builder: (context) => LoginScreen()),
        );
},
```

So, on clicking the **Log In** button in the Login Screen, we redirect to the Home Screen. And after clicking on the **Sign Out** button in the Home Screen, we go back to the Login Screen.



16:03 | 61.1KB/s
English (United States)
Instagram
Phone number, email or username
Password
Log In
Forgot your login details? Get help logging in.
OR
Log in with Facebook
Don't have an account? Sign up.



18:19 | 1.5KB/s
Instagram
Your Story
Alexander
Alex
Julian
helicopter0707
2 likes
helicopter0707 Meow
Jan 30, 2022
helicopter0707

Navigation with Named Routes

We can work with named routes by using the **Navigator.pushNamed()** function. This function takes two required arguments (build context and string) and one optional argument. Also, we know about the MaterialPageRoute, which is responsible for page transition. If we do not use this, then it is difficult to change the page.

The following steps are necessary, which demonstrate how to use named routes.

1. **First, we need to create two screens**

Here, we are going to create two routes for navigation. In both routes, we have created only a single button. When we tap the button on the Login Screen, it will navigate to the Home Screen. Again, when we tap the button on the Home Screen, it will return to the Login Screen.

2. **Define the routes**

In this step, we have to define the routes. The MaterialApp constructor is responsible for defining the initial route and other routes themselves. Here the initial route tells the start of the page and routes property defines the available named routes and widgets.

```
MaterialApp(
    debugShowCheckedModeBanner: false,
    title: 'Instagram Clone',
    theme: ThemeData(
      fontFamily: 'Helvetica',
      brightness: Brightness.dark,
      scaffoldBackgroundColor: mobileBackgroundColor,
    ),
    initialRoute: '/',
    routes: {
      // When navigating to the "/" route, build the Login Screen widget.
      '/': (context) => LoginScreen(),
```

```
        // When navigating to the "/home" route, build the Home Screen widget.
        '/home': (context) => LoginScreen(),
      },
),
```

3. **Navigate to the second screen using the Navigator.pushNamed() function**

In this step, we need to call the Navigator.pushNamed() method for navigation. For this, we need to update an onPressed() callback in the build method of Login Screen like below code snippets.

```
onPressed: () {
        Navigator.pushNamed(
                context,
                '/home',
        );
},
```

4. **Use a Navigator.pop() function to return to the first screen**

It is the final step, where we will use the Navigator.pop() method to return to the Login Screen from Home Screen.

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

The output is the same as above.

Gesture Detector

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. Some of the examples of gestures are:

1. When the mobile screen is locked, you slide your finger across the screen to unlock it
2. Tapping a button on your mobile screen
3. Tapping and holding an app icon on a touch-based device to drag it across screens

Flutter provides a widget that gives excellent support for all types of gestures by using the **GestureDetector** widget. The GestureWidget is non-visual widgets, which is primarily used for detecting the user's gesture. The basic idea of the gesture detector is a stateless widget that contains parameters in its constructor for different touch events.

The GestureDetector widget decides which gesture is going to recognize based on which of its callbacks are non-null. Let us learn how we can use these gestures in our application with a simple onTap() event and determine how the GestureDetector processes this. I have used the GestureDetector widget for Login Button and Sign Up button and many more.

1. Login Button

```
Container(
        padding: const EdgeInsets.symmetric(
                horizontal: 32,
        ),
        child: GestureDetector(
        onTap: loginUser,
        child: Container(
        child: _isLoading
        ? const Center(
                child: CircularProgressIndicator(
                        color: primaryColor,
                ),
```

```
                )
          : const Text(
                'Log In',
                style: TextStyle(
                fontSize: 16.0,
                fontWeight: FontWeight.bold,
                ),
          ),
          width: double.infinity,
          alignment: Alignment.center,
          padding: const EdgeInsets.symmetric(
                vertical: 17,
          ),
          decoration: const ShapeDecoration(
                shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.all(
                      Radius.circular(4),
                ),
                ),
                color: blueColor),
          ),
          ),
),
```

2. Asking for sign up text button

```
GestureDetector(
          onTap: () {
                Navigator.push(
                context,
                MaterialPageRoute(
                      builder: (context) => PhoneOrEmail()),
```

```
        );
        },
    child: Container(
    child: const Text(
        "Sign up.",
        style: TextStyle(
        fontWeight: FontWeight.bold,
        color: Colors.blueAccent,
        ),
    ),
    padding: const EdgeInsets.symmetric(
        vertical: 8,
    ),
    ),
),
```

Conclusion: Hence, we understood how to apply navigation, routing and gestures in the Login Screen, Sign Up screen and Home Screen of my Flutter Application.

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.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	L03
Grade:	

EXPERIMENT NO. 6
MAD and PWA Lab

Aim: To Connect Flutter UI with firebase database

Theory:

Google **Firebase** is a Google-backed application development software that enables developers to develop iOS, Android and Web apps. Firebase provides tools for tracking analytics, reporting and fixing app crashes, creating marketing and product experiments.

Firebase offers a number of services, including:

1. **Analytics** – Google Analytics for Firebase offers free, unlimited reporting on as many as 500 separate events. Analytics presents data about user behavior in iOS and Android apps, enabling better decision-making about improving performance and app marketing.

2. **Authentication** – Firebase Authentication makes it easy for developers to build secure authentication systems and enhances the sign-in and onboarding experience for users. This feature offers a complete identity solution, supporting email and password accounts, phone auth, as well as Google, Facebook, GitHub, Twitter login and more.

3. **Cloud messaging** – Firebase Cloud Messaging (FCM) is a cross-platform messaging tool that lets companies reliably receive and deliver messages on iOS, Android and the web at no cost.

4. **Realtime database** – the Firebase Realtime Database is a cloud-hosted NoSQL database that enables data to be stored and synced between users in real time. The data is synced across all clients in real time and is still available when an app goes offline.

5. **Crashlytics** – Firebase Crashlytics is a real-time crash reporter that helps developers track, prioritize and fix stability issues that reduce the quality of their apps. With crashlytics, developers spend less time organizing and troubleshooting crashes and more time building features for their apps.

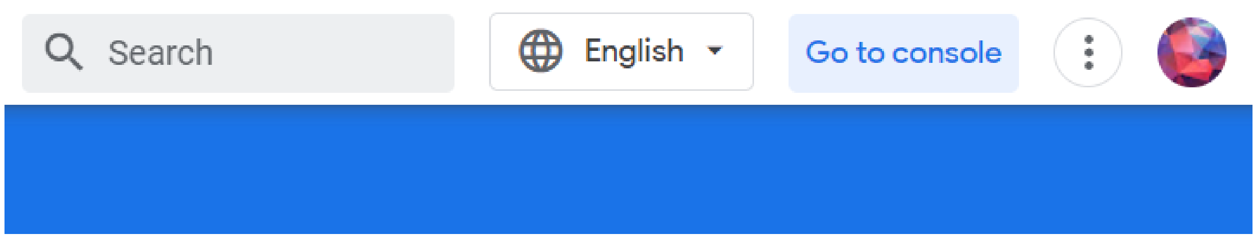
6. **Performance** – Firebase Performance Monitoring service gives developers insight into the performance characteristics of their iOS and Android apps to help them determine where and when the performance of their apps can be improved.

7. **Test lab** – Firebase Test Lab is a cloud-based app-testing infrastructure. With one operation, developers can test their iOS or Android apps across a variety of devices and device configurations. They can see the results, including videos, screenshots and logs, in the Firebase console.

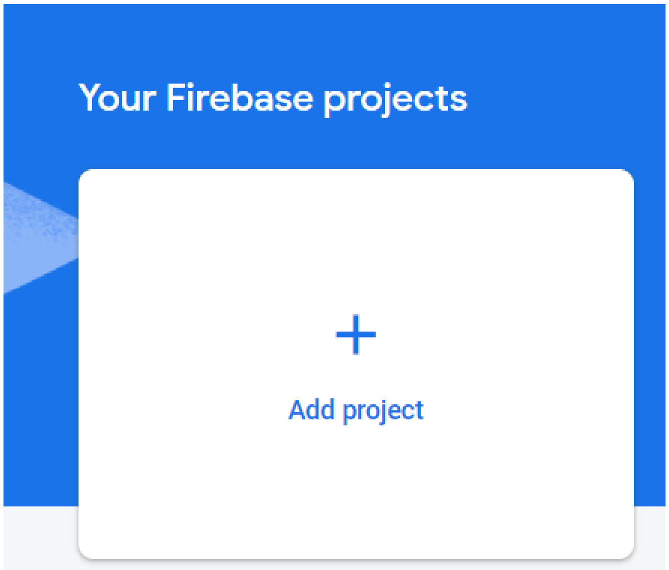
Creating a Firebase Project

Add Firebase to your existing Google Cloud project:

1. Log in to the Firebase console, then click Add project.
2. Select your existing Google Cloud project from the dropdown menu, then click Continue.
3. (Optional) Enable Google Analytics for your project, then follow the prompts to select or create a Google Analytics account.
4. Click Add Firebase.

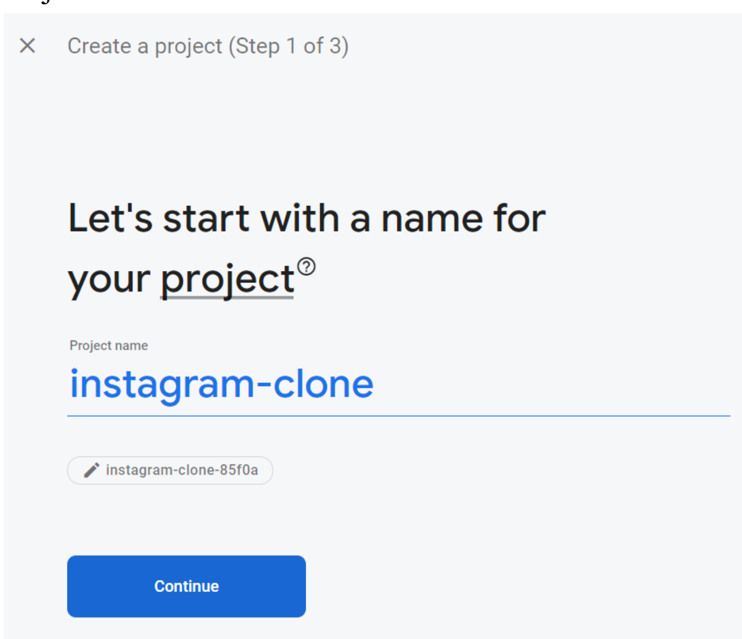


Now open the Firebase console and click Create New Project.

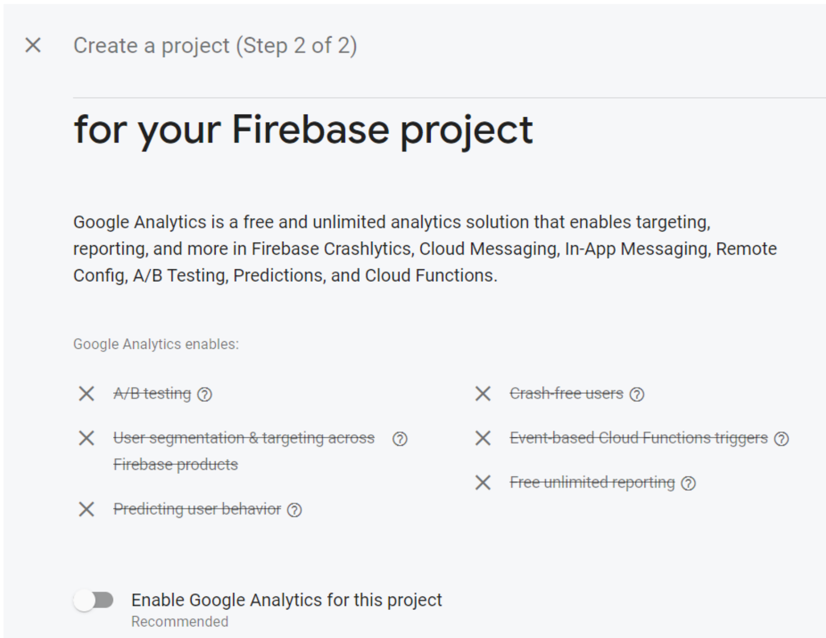


Enter the following information and click on Create Project:

1. Project name

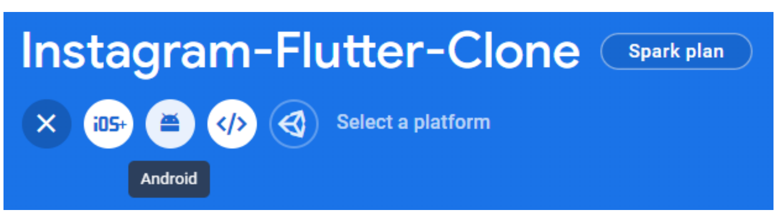


2. Disable the Google Analytics for the project, we do not need this now unless we deploy the app.

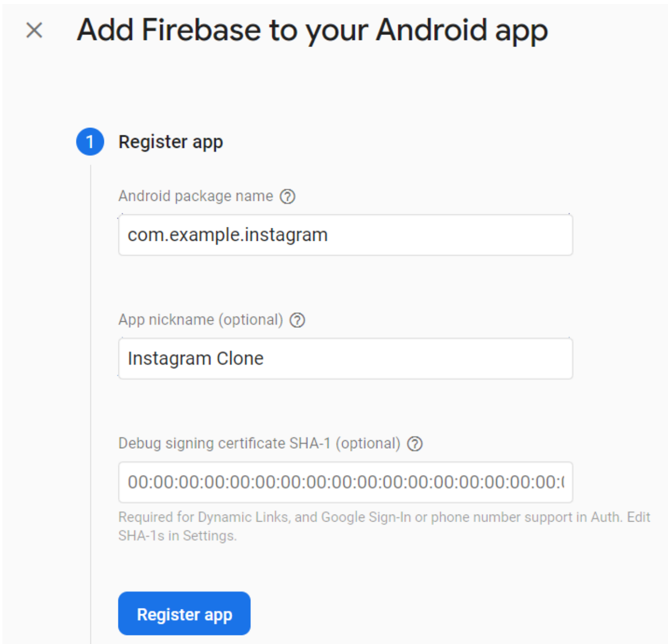


Setting up the Android App/IOS App

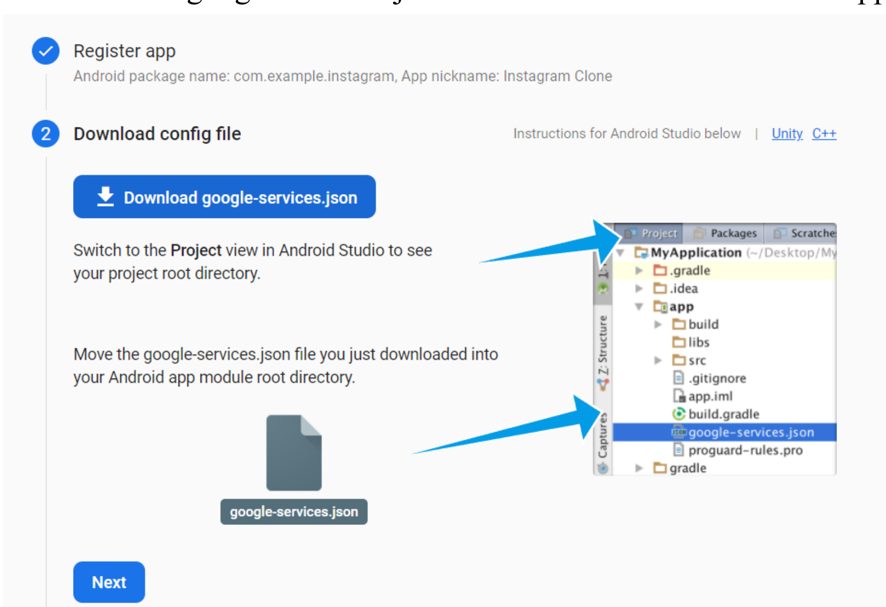
Now add an Android app if you are running your app on Android or else add an IOS app. I will be adding the Android app.



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



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



3. Add Firebase SDK to the project.

3 Add Firebase SDK

Instructions for Gradle | Unity C++

The Google services plugin for Gradle loads the `google-services.json` file you just downloaded. Modify your build.gradle files to use the plugin.

Project-level build.gradle (`<project>/build.gradle`):

```
buildscript {
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
  }
  dependencies {
    ...
    // Add this line
    classpath 'com.google.gms:google-services:4.3.10'
  }
}

allprojects {
  ...
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
    ...
  }
}
```

App-level build.gradle (`<project>/<app-module>/build.gradle`):

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
  // Import the Firebase BoM
  implementation platform('com.google.firebase:firebase-bom:29.1.0')

  // Add the dependencies for the desired Firebase products
  // https://firebase.google.com/docs/android/setup#available-libraries
}
```

By using the Firebase Android BoM, your app will always use compatible Firebase library versions. Learn more

Finally, press "Sync now" in the bar that appears in the IDE:

Gradle files have changed sir Sync now

Previous Next

Installing the FlutterFire libraries

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

```yaml
dependencies:
  cloud_firestore: ^3.1.6
  cupertino_icons: ^1.0.2
  firebase_auth: ^3.3.5
  firebase_core: ^1.11.0
  firebase_storage: ^10.2.5
  flutter:
    sdk: flutter
  flutter_staggered_grid_view: ^0.6.1
  flutter_svg: ^1.0.2
  font_awesome_flutter: any
  image_picker: ^0.8.4+4
  intl: ^0.17.0
  provider: ^6.0.2
  uuid: ^3.0.5
```

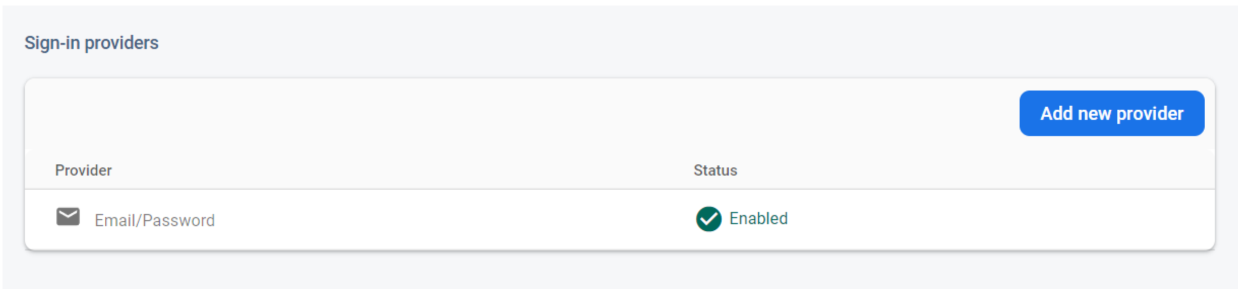
Initializing Firebase in the code

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

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

Making a simple Realtime Database call

For Login and Sign Up, I have used Email/Password provider in Authentication of the Firebase project.



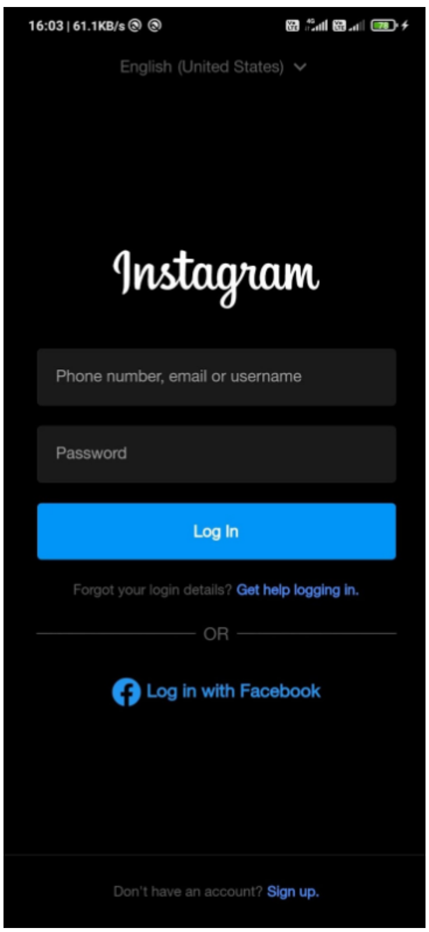
For user authentication, I have made a separate **auth_methods.dart** file where I have added the functions to get user details, login and register.

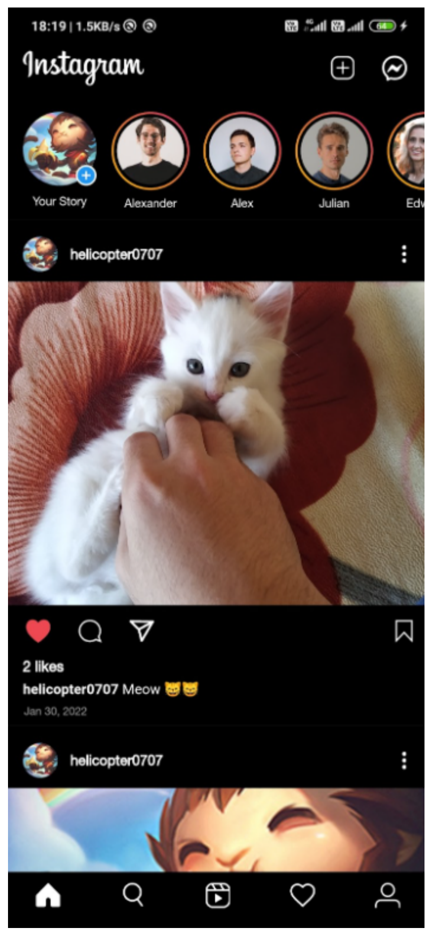
```
import 'dart:typed_data';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:instagram_clone/models/user.dart' as model;
import 'package:instagram_clone/resources/storage_methods.dart';

class AuthMethods {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  Future<model.User> getUserDetails() async {
    User currentUser = _auth.currentUser!;
    DocumentSnapshot snap =
        await _firestore.collection('users').doc(currentUser.uid).get();
    return model.User.fromSnap(snap);
  }

  // Sign up the user
```

```
Future<String> signUpUser({
  required String email,
  required String fullName,
  required String password,
  required String username,
  required String bio,
  // required int phoneNo,
  required Uint8List file,
}) async {
  String res = "Some error occurred";
  try {
    if (fullName.isNotEmpty ||
        email.isNotEmpty ||
        password.isNotEmpty ||
        username.isNotEmpty ||
        bio.isNotEmpty) {
      // Register user
      UserCredential cred = await _auth.createUserWithEmailAndPassword(
        email: email,
        password: password,
      );

      String photoUrl = await StorageMethods().uploadImage(
        'profilePics',
        file,
        false,
      );

      model.User user = model.User(
        username: username,
        uid: cred.user!.uid,
```

```
        fullName: fullName,
        email: email,
        bio: bio,
        followers: [],
        following: [],
        photoUrl: photoUrl,
      );

      // Add user to our database
      await _firestore.collection('users').doc(cred.user!.uid).set(
            user.toJson(),
          );
      res = "Success";
    }
  } catch (err) {
    res = err.toString();
  }
  return res;
}

// Login user
Future<String> loginUser({
  required String email,
  required String password,
}) async {
  String res = "Some error occurred";
  try {
    if (email.isNotEmpty || password.isNotEmpty) {
      await _auth.signInWithEmailAndPassword(
        email: email,
        password: password,
```

```
    );
    res = "Success";
   }
  } catch (err) {
   res = err.toString();
  }
  return res;
 }
}
```





Conclusion: Hence, we understood how to connect Flutter UI with firebase database.

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 in a Web app manifest file to enable "add to homescreen feature".
Roll No.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	L04
Grade:	

EXPERIMENT NO. 7
MAD and PWA Lab

Aim: To learn about Progressive Web Apps and how they are different from standard websites.

Theory:

PWA

PWA stands for a progressive web app. This is an app built from the web technologies we all know and love, like HTML, CSS, and JavaScript, but with a feel and functionality that rivals an actual native app. Thanks to a couple of smart additions, you can turn almost any website into a progressive web app. This means that you can build a PWA rather quickly, in regards to a native app that's pretty difficult to develop. Plus, you can offer all the features of native apps, like push notifications, offline support, and much more.

Difference Between Websites and PWA

Websites	PWA
1. Need to be accessed by URLs and/or search engines in a browser. 2. Offer lesser engagement options since the only interaction is on the website. 3. Cannot be accessed offline. 4. Not as good as PWAs in the context of metrics.	1. Can be directly accessed just like a native application. 2. Offers more engagement options via popups, push notifications, etc. 3. Static and cached data can be accessed offline. 4. Generally perform better with metrics.

Benefits of PWA

- You don't have to go through the process to get into different app stores
- You can build PWAs with common web technologies
- They are often cheaper to build
- Since you're turning your site into an app, you'll have fewer code-bases to maintain
- PWAs are responsive and work with many different screen sizes
- PWAs are smooth, fast and lightweight
- No need to hand off big chunks of money to Google and Apple
- They work offline, unlike your regular site

- PWAs are discoverable via search engines (which have a much larger audience than app stores. Plus, if you want you can still get your PWAs distributed via app stores)
- You can use push notifications to re-engage users
- Installing a PWA can lead to higher engagement

Disadvantages of PWA

- **No Access to App Stores** - The app needs to be installed from its proprietary website.
- **Limited Functionality** - Support for Hardware functionality like Bluetooth, WiFi, NFC etc. is always doubtful since these are always supported better in native apps.
- **Performance** - Since the app essentially runs on a browser client, it'll never be as fast as a native application.

Code to enable "Add to Home Screen" option

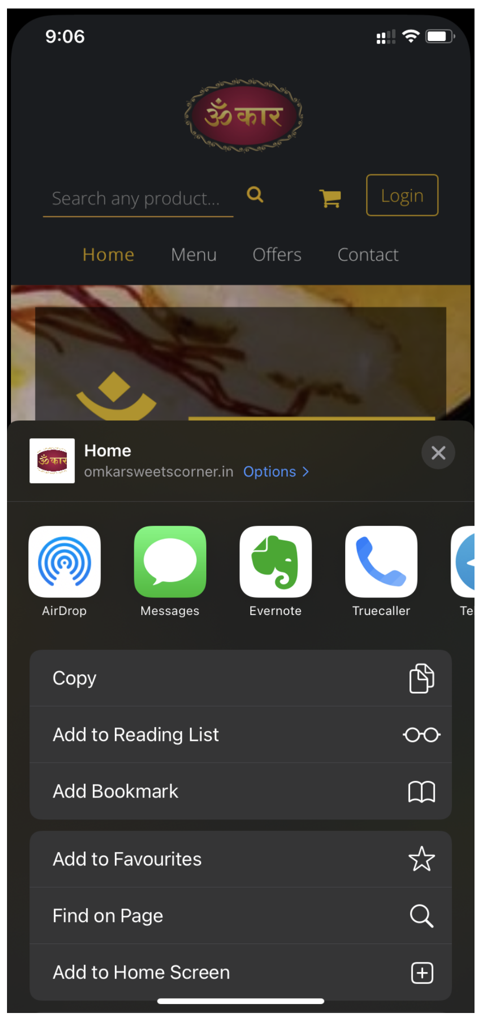
manifest.json:

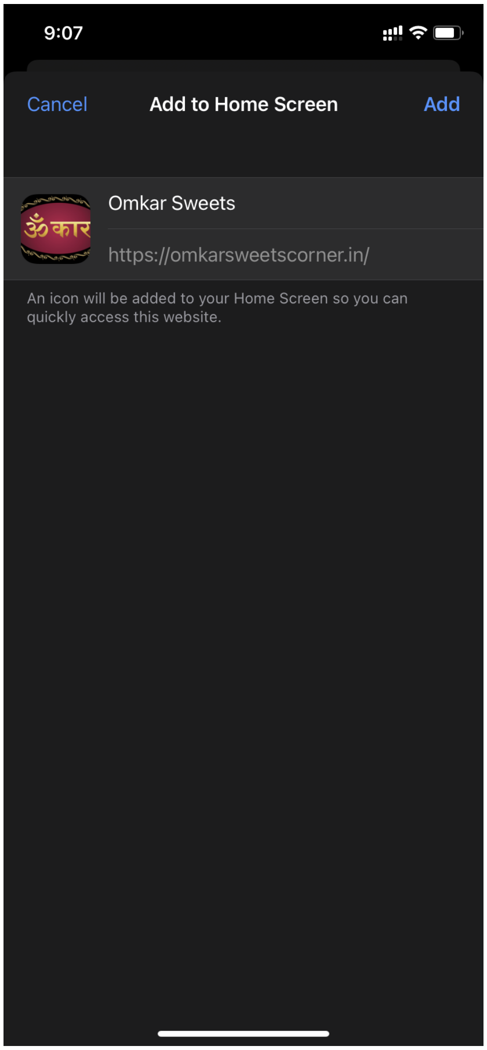
```
{
  "name": "Omkar Sweets Corner",
  "short_name": "Omkar Sweets",
  "start_url": "/",
  "background_color": "#212529",
  "description": "Your one stop shop for Exclusive Dry fruit Sweets, Mawa Mithai, Bengali Sweets, Top Quality Dry fruits, Wide range of Farsan, Imported Chocolates, Fancy Cakes and Lip Smacking Chat",
  "display": "fullscreen",
  "theme_color": "#212529",
  "icons": [
    {
      "src": "windows11/SmallTile.scale-100.png",
      "sizes": "71x71"
    },
    {
      "src": "windows11/SmallTile.scale-125.png",
      "sizes": "89x89"
    },
```

```
{
  "src": "windows11/SmallTile.scale-150.png",
  "sizes": "107x107"
},
{
  "src": "windows11/Square44x44Logo.targetsize-24.png",
  "sizes": "24x24"
},
{
  "src": "android/android-launchericon-512-512.png",
  "sizes": "512x512"
},
{
  "src": "android/android-launchericon-192-192.png",
  "sizes": "192x192"
},
{
  "src": "android/android-launchericon-144-144.png",
  "sizes": "144x144"
},
{
  "src": "android/android-launchericon-96-96.png",
  "sizes": "96x96"
},
{
  "src": "android/android-launchericon-72-72.png",
  "sizes": "72x72"
},
{
  "src": "android/android-launchericon-48-48.png",
  "sizes": "48x48"
},
{
  "src": "ios/16.png",
  "sizes": "16x16"
},
```

```
    {
      "src": "ios/20.png",
      "sizes": "20x20"
    },
    {
      "src": "ios/29.png",
      "sizes": "29x29"
    },
  ]
}
```

Output





9:07
Cancel
Add to Home Screen
Add
Omkar Sweets
https://omkarsweetscorner.in/
An icon will be added to your Home Screen so you can quickly access this website.

9:08
ॐकार
Search any product...
Login
Home
Menu
Offers
Contact
ॐकार
Omkar Sweets Corner
आम्ही नाती, परंपरा आणि गोडवा जपतो
Offering a range of sweet delicacies to please your sweet tooth along with zestful savouries!
Best Sellers
New Arrivals

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

MAD & PWA Lab Journal

Experiment No.	08
Experiment Title.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA
Roll No.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	L05
Grade:	

EXPERIMENT NO. 8
MAD and PWA Lab

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

What can we do with Service Workers?

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

- You can **Cache**

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

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

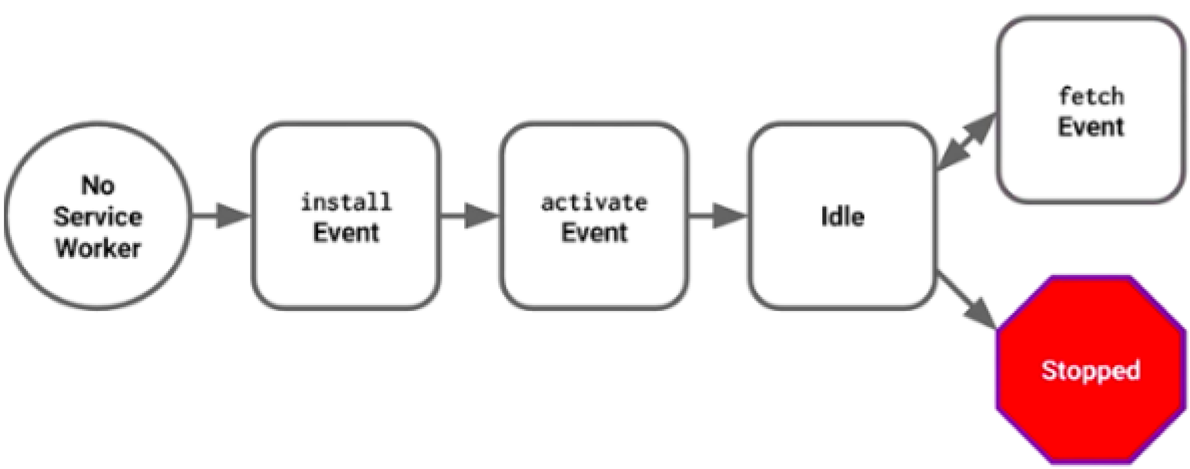
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can't we do with Service Workers?

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

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

Service Worker Cycle



A service worker goes through three steps in its lifecycle:

- Registration
- Installation
- Activation

Registration

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

main.js

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

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

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

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

main.js

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

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

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

main.js

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

Installation

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

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

service-worker.js

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

Activation

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

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

service-worker.js

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

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

Code

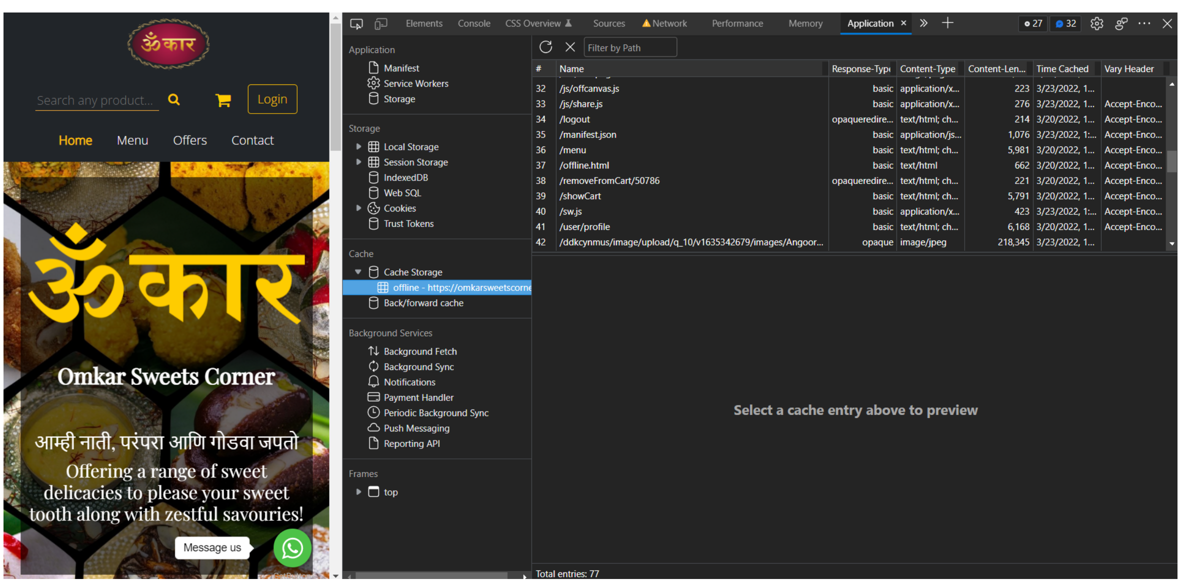
sw.js

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

```
    event.waitUntil(preLoad());
});

var filesToCache = [
    '/',
    '/courses',
    '/news',
];

var preLoad = function () {
    return caches.open("offline").then(function (cache) {
        // caching index and important routes
        return cache.addAll(filesToCache);
    });
};

self.addEventListener("fetch", function (event) {
    event.respondWith(checkResponse(event.request).catch(function () {
        return returnFromCache(event.request);
    }));
    event.waitUntil(addToCache(event.request));
});

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

var addToCache = function (request) {
```

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

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

Output Screenshot



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, sync and push for E-commerce PWA
Roll No.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	L05
Grade:	

EXPERIMENT NO. 9
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 "offline first" web applications with Cache API.

Things to note about Service Worker:

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

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

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

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

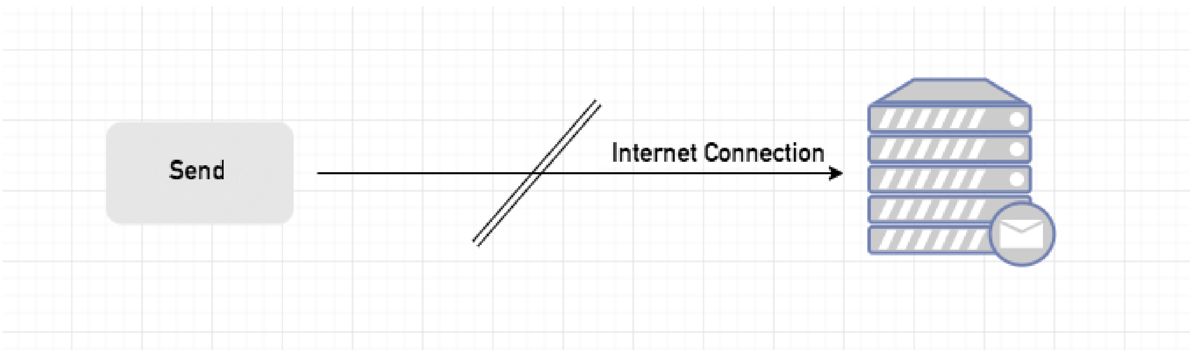
- **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed successfully, the new response will be cached and returned. But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is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

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

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

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

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

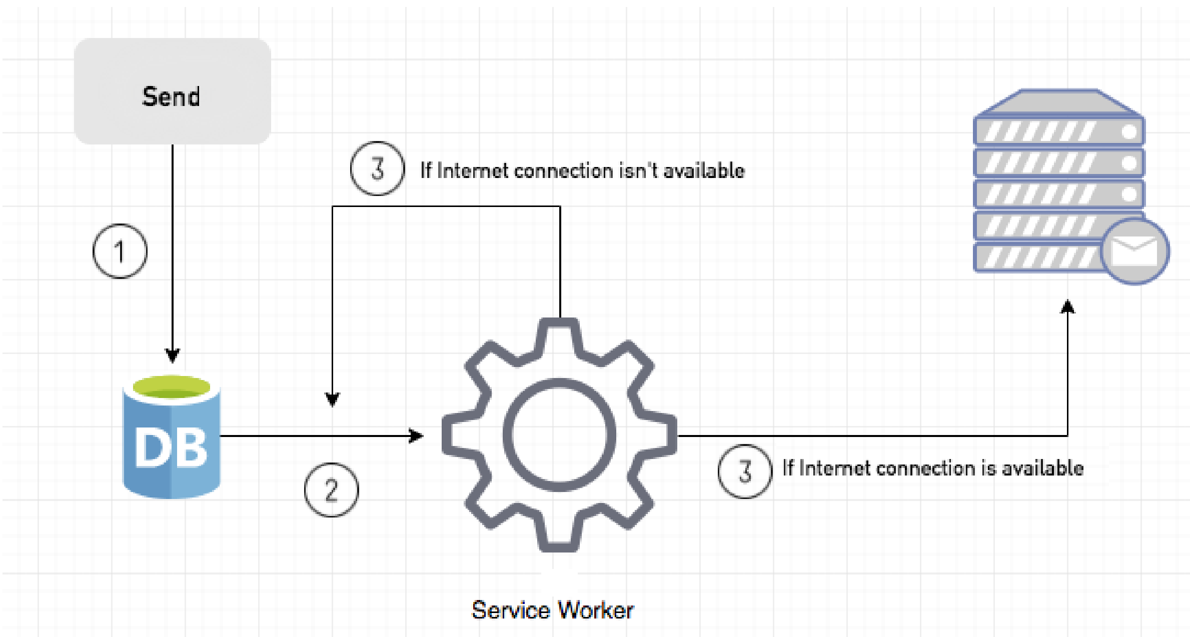
Sync Event

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

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



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



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

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

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

Event Listener for Background Sync Registration

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

Event Listener for sw.js

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

Push Event

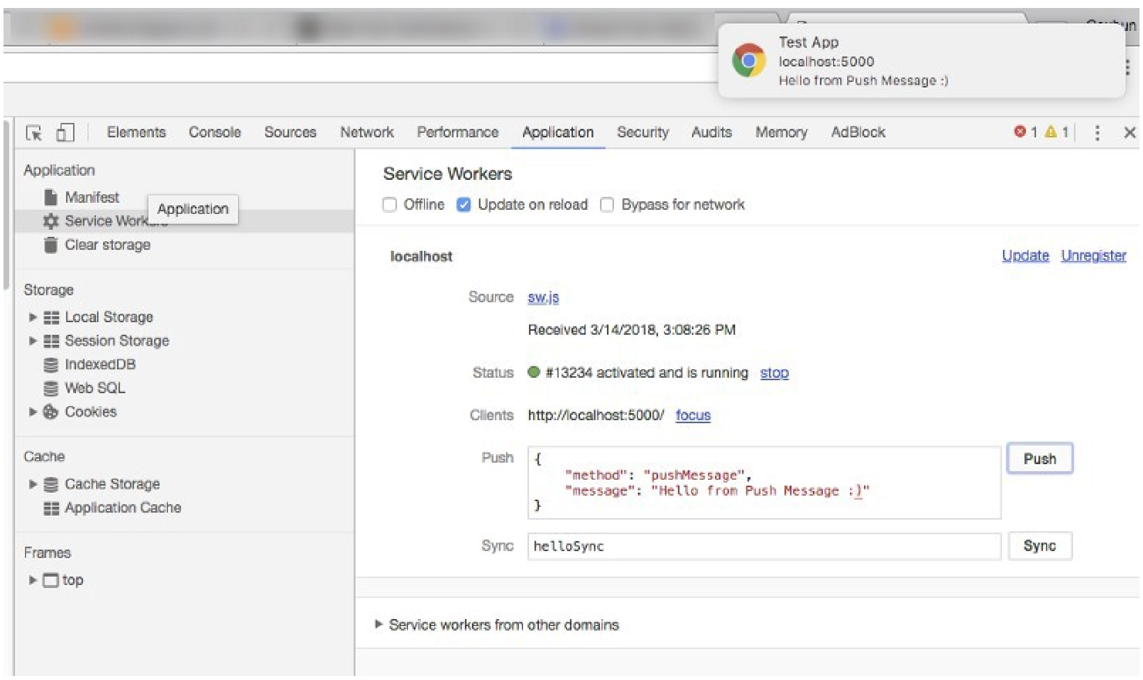
This is the event that handles push notifications that are received from the server. You can apply any method with received data.

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

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

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

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



Code:

sw.js

```
self.addEventListener("install", function (event) {
    event.waitUntil(preLoad());
```

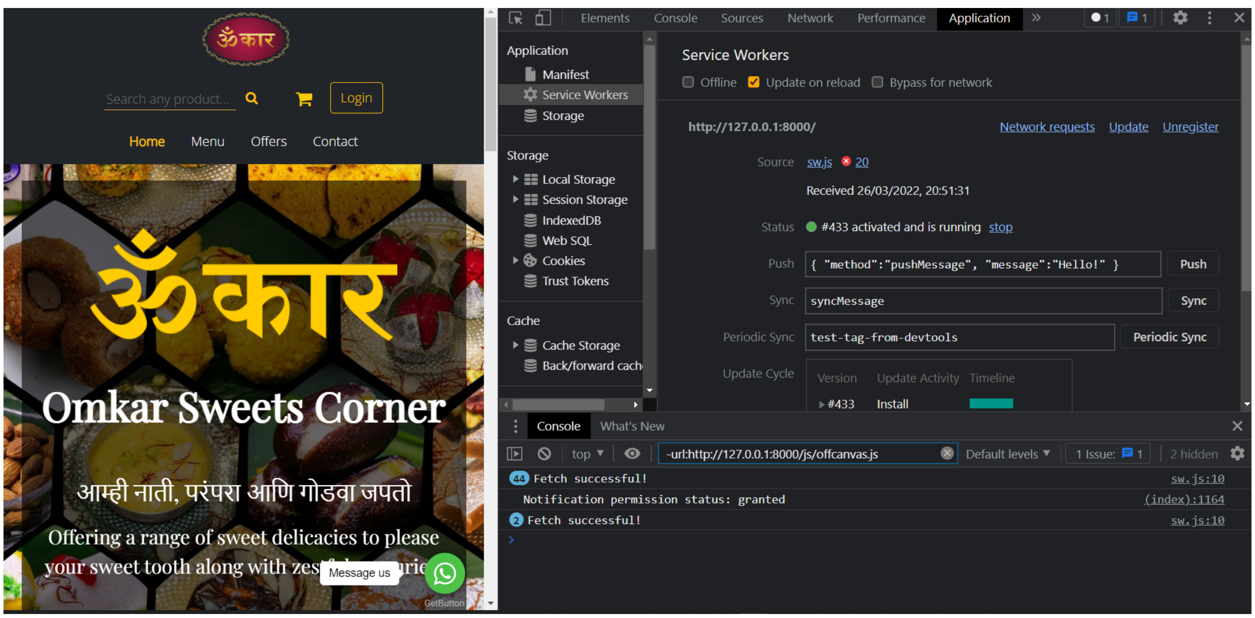
```
});

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

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

self.addEventListener('push', function (event) {
    if (event && event.data) {
        var data = event.data.json();
        if (data.method == "pushMessage") {
            console.log("Push notification sent");
            event.waitUntil(self.registration.showNotification("Omkar Sweets Corner", {
                body: data.message
            }))
        }
    }
})


var filesToCache = [
    '/',
    '/menu',
    '/contactUs',
    '/offline.html',
];
```

```
var preLoad = function () {
  return caches.open("offline").then(function (cache) {
    // caching index and important routes
    return cache.addAll(filesToCache);
  });
};

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

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

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

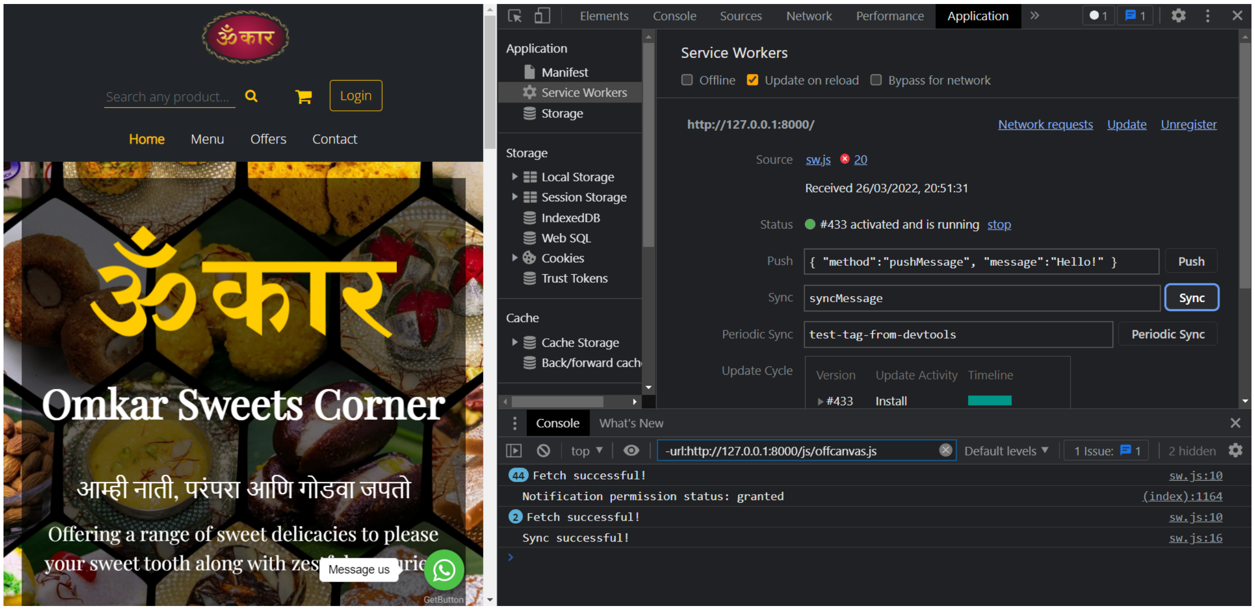
```
        });
    });
};
```

Output:

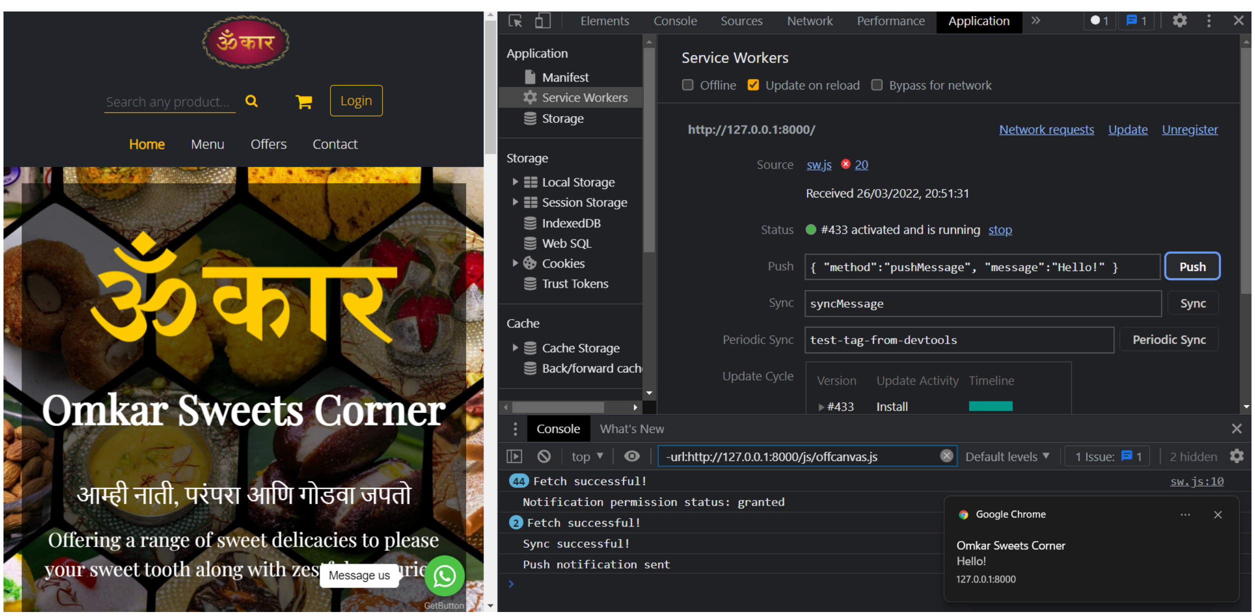
Fetch event



Sync event



Push event



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

<u>MAD & PWA Lab Journal</u>

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	L05
Grade:	

EXPERIMENT NO. 10
MAD and 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 favoring this over Firebase:
1. Free to use
2. Right out of github
3. Quick to set up

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

Advantages

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

Disadvantages

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

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

- 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.
- Firebase apps can be written entirely with client-side code, update in real-time out-of-the-box, interoperate well with existing services, scale automatically, and provide strong data security.
- Data Accessibility- Data is stored as JSON in Firebase. Every piece of data has its own URL which can be used in Firebase's client libraries and as a REST endpoint. These URLs can also be entered into a browser to view the data and watch it update in real-time.

Reasons for favoring over GitHub Pages:
1. Realtime backend made easy
2. Fast and responsive

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

Advantages

- Hosted by Google.
- Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
- A real-time database will be available to you, which can store 1 GB of data.

- You'll also have access to a blob store, which can store another 1 GB of data.
- Support for HTTPS. A free certificate will be provisioned for your custom domain within 24 hours.

Disadvantages

- 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.
- Command-line interface only.
- No in-built support for any static site generator.

Link to our GitHub repository:
https://omkarsweetscorner.github.io

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

MAD & PWA Lab Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	L06
Grade:	

EXPERIMENT NO. 11
MAD and PWA Lab

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

Theory:

Google Lighthouse
Lighthouse is an open-source, automated tool for improving the quality of web pages. You can run it against any web page, public or requiring authentication. It has audits for performance, accessibility, progressive web apps, SEO and more.

You can run Lighthouse in Chrome DevTools, from the command line, or as a Node module. You give Lighthouse a URL to audit, it runs a series of audits against the page, and then it generates a report on how well the page did. From there, use the failing audits as indicators on how to improve the page. Each audit has a reference doc explaining why the audit is important, as well as how to fix it.

Features of Lighthouse
Google Lighthouse gives a breakdown of your site into the accompanying metrics. Here is a brief explanation of each of the aforementioned metrics:

1. **Performance**
Performance is generally viewed as the most valuable metric given by the Google Lighthouse tool. Like the PageSpeed Insights, the Performance area of the Lighthouse report contains a few helpful metrics you can use to advance your site to climb Google's rankings. The Performance segment of the Lighthouse report joins the Opportunities, Field Data, Lab Data, and Diagnostics metrics of the PageSpeed Insights tool. A great example is the opportunities metric as it flags three types of render-blocking URL's namely stylesheets, scripts, and HTML imports. This merged perspective on performance metrics gives an exact and valuable analysis of your site's performance and any progressions you should make to expand your site's exhibition.

2. **Accessibility**
The first of the new regions of Google Lighthouse is the Accessibility metric. Basically what this metric does is feature potential chances to improve the availability and client experience of your mobile app or website. Following the accessibility improvement report will guarantee that your

clients can without much of a stretch explore and utilize your site. Just as guaranteeing that you have the most obvious opportunity with regards to positioning better on web search engines.

3. **Best Practices**

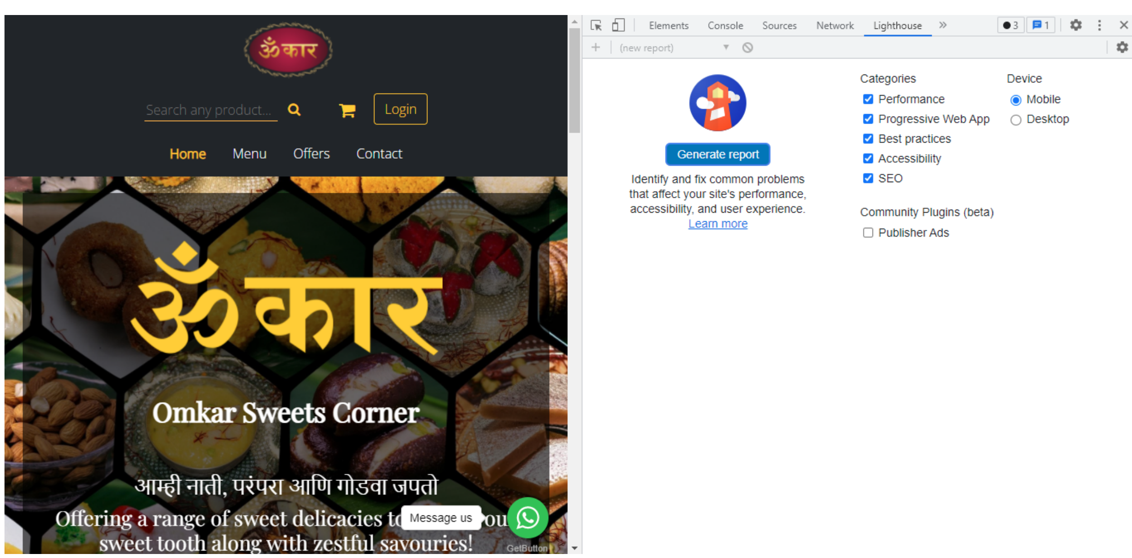
Another segment new to Google's analysis tools is the Best Practices metric. This region of the Lighthouse report doesn't carefully give execution related data. However, it will give you recommendations which can improve both your exhibition and client experience, particularly for mobile sites.

4. **SEO**

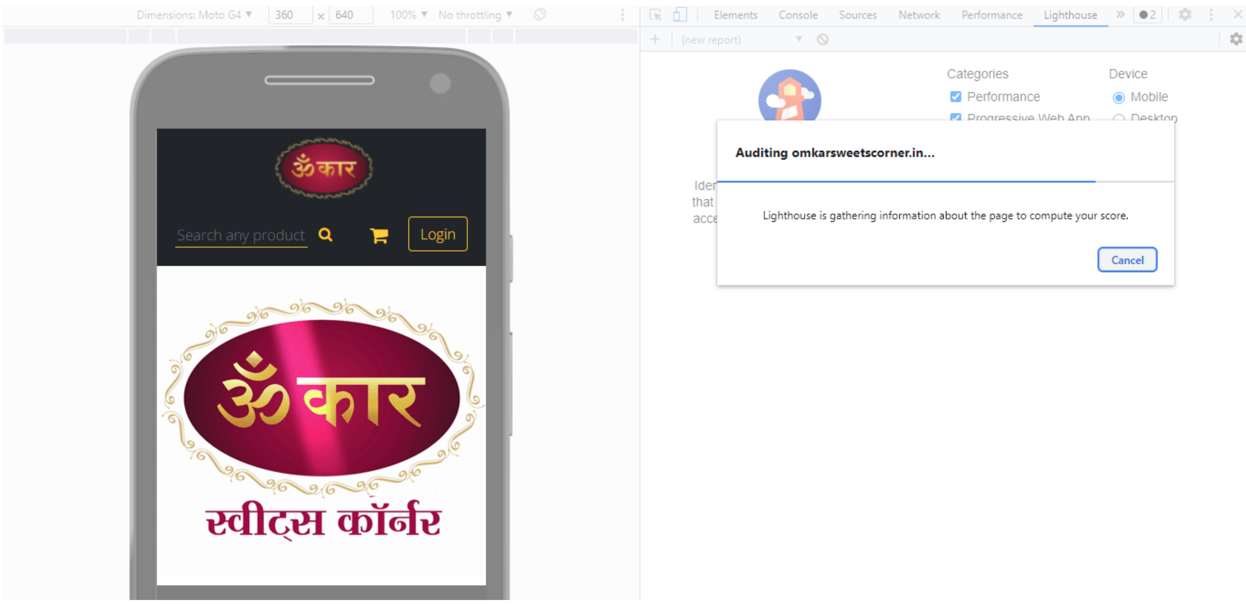
The latest and most dynamic of the highlights in Google's Lighthouse instrument is the SEO metric. PageSpeed Insights doesn't offer this tool. This is why most web designers and SEO specialists prefer to utilize Google Lighthouse to analyze a website. The SEO metric gives fundamental tools to examine your page's streamlining for search engine results rankings. While there are numerous more factors which Lighthouse doesn't consider or quantify, the most essential focuses are secured.

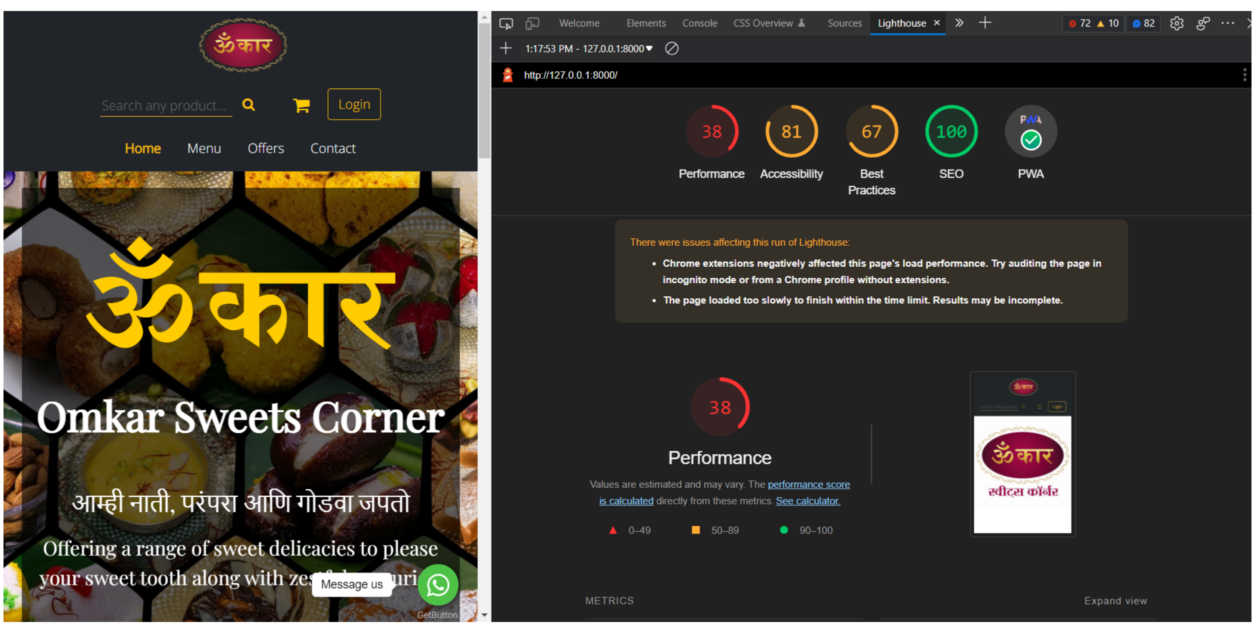
5. **Progressive Web Applications**

The Progressive Web App area is another of Google's most up to date execution measurements incorporated into its Lighthouse tool. While the meaning of a Progressive Web App (PWA) hasn't been especially clear, Google's documentation expresses that there are a few key variables which make a site a PWA. A great feature of this metric is registering service workers which allow you to enable push notifications on your web app



For Mobile Device

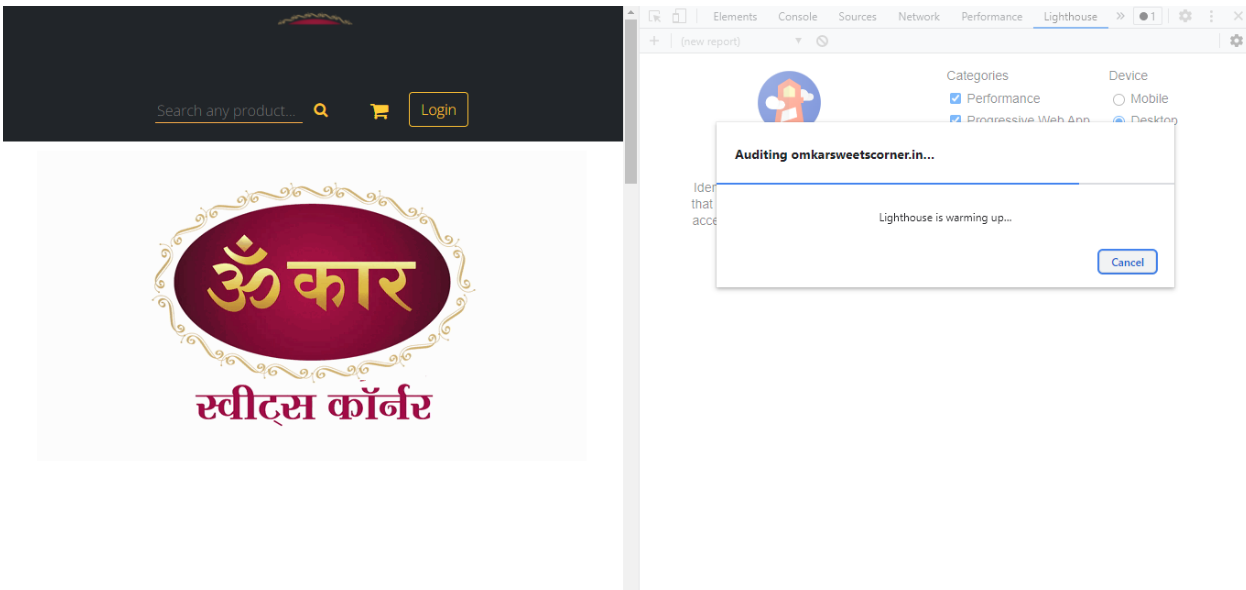


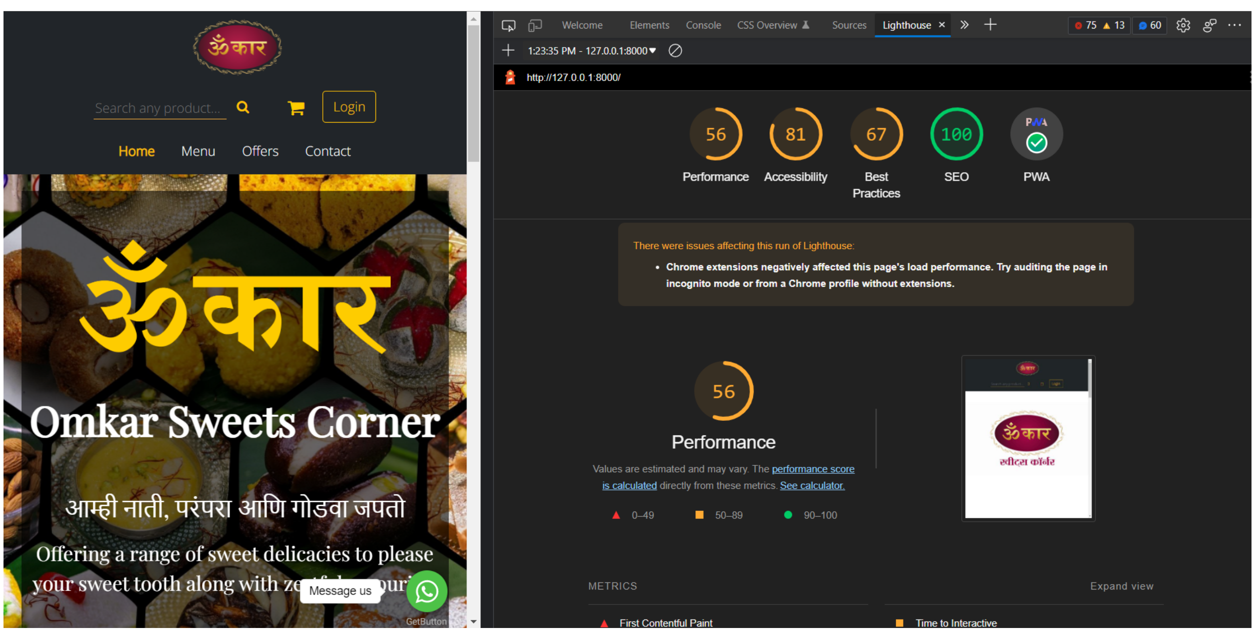


Vivekanand Education Society's
Institute of Technology
(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

V.E.S.
Since 1962

For Desktop Device





http://127.0.0.1:8000/

38	81	67	100	PWA
Performance	Accessibility	Best Practices	SEO	PWA

There were issues affecting this run of Lighthouse:

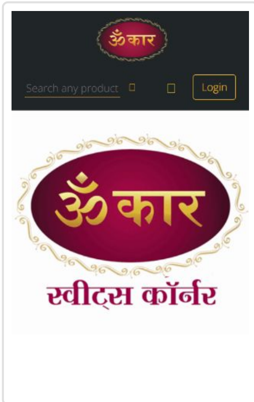
- **Chrome extensions negatively affected this page's load performance. Try auditing the page in incognito mode or from a Chrome profile without extensions.**
- **The page loaded too slowly to finish within the time limit. Results may be incomplete.**

38

Performance

Values are estimated and may vary. The [performance score is calculated](#) directly from these metrics. [See calculator.](#)

▲ 0–49 50–89 90–100



METRICS Expand view

▲ First Contentful Paint
3.6 s

Time to Interactive
6.1 s

▲ Speed Index
7.2 s

▲ Total Blocking Time
810 ms

▲ Largest Contentful Paint
56.0 s

Cumulative Layout Shift
0.002

View Original Trace

View Treemap

Show audits relevant to: All FCP TBT LCP CLS

OPPORTUNITIES

Opportunity	Estimated Savings
▲ Properly size images	46.05 s
▲ Serve images in next-gen formats	30.75 s
▲ Use video formats for animated content	27 s
▲ Efficiently encode images	22.65 s
▲ Eliminate render-blocking resources	2.47 s
Enable text compression	0.75 s
Reduce unused JavaScript	0.22 s

These suggestions can help your page load faster. They don't [directly affect](#) the Performance score.

DIAGNOSTICS

- ▲ Avoid enormous network payloads — Total size was 16,006 KiB
- ▲ Serve static assets with an efficient cache policy — 9 resources found
- ▲ Ensure text remains visible during webfont load
- ▲ Does not use passive listeners to improve scrolling performance
- ▲ Image elements do not have explicit `width` and `height`
- ▲ Minimize main-thread work — 9.1 s
- Reduce JavaScript execution time — 2.6 s
- ○ Avoid chaining critical requests — 18 chains found

- Keep request counts low and transfer sizes small — 53 requests • 16,006 KiB
- Largest Contentful Paint element — 1 element found
- Avoid large layout shifts — 1 element found
- Avoid long main-thread tasks — 17 long tasks found
- Avoid non-composited animations — 9 animated elements found

More information about the performance of your application. These numbers don't directly affect the Performance score.

PASSED AUDITS (20) Show

81

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Only a subset of accessibility issues can be automatically detected so manual testing is also encouraged.

NAMES AND LABELS

- Buttons do not have an accessible name
- Form elements do not have associated labels
- Links do not have a discernible name

These are opportunities to improve the semantics of the controls in your application. This may enhance the experience for users of assistive technology, like a screen reader.

CONTRAST

- Background and foreground colors do not have a sufficient contrast ratio.

These are opportunities to improve the legibility of your content.

NAVIGATION

- `[id]` attributes on active, focusable elements are not unique

Heading elements are not in a sequentially-descending order

These are opportunities to improve keyboard navigation in your application.

ADDITIONAL ITEMS TO MANUALLY CHECK (10) Show

These items address areas which an automated testing tool cannot cover. Learn more in our guide on conducting an accessibility review.

PASSED AUDITS (17) Show

NOT APPLICABLE (21) Show

67

Best Practices

TRUST AND SAFETY

- Does not use HTTPS — 1 insecure request found
- Requests the notification permission on page load
- Ensure CSP is effective against XSS attacks

GENERAL

- Browser errors were logged to the console
- Issues were logged in the `Issues` panel in Chrome Devtools
- Detected JavaScript libraries

PASSED AUDITS (9) Show

NOT APPLICABLE (1) Show

100

SEO

These checks ensure that your page is following basic search engine optimization advice. There are many additional factors Lighthouse does not score here that may affect your search ranking, including performance on Core Web Vitals. Learn more.

ADDITIONAL ITEMS TO MANUALLY CHECK (1) Show

Run these additional validators on your site to check additional SEO best practices.

PASSED AUDITS (14) Show

PWA

These checks validate the aspects of a Progressive Web App. Learn more.

INSTALLABLE

- Web app manifest and service worker meet the installability requirements

PWA OPTIMIZED

- Registers a service worker that controls page and `start_url`
- Configured for a custom splash screen
- Sets a theme color for the address bar.
- Content is sized correctly for the viewport
- Has a `<meta name="viewport">` tag with `width` or `initial-scale`
- Provides a valid `apple-touch-icon`
- Manifest has a maskable icon

ADDITIONAL ITEMS TO MANUALLY CHECK (3) Show

These checks are required by the baseline PWA Checklist but are not automatically checked by Lighthouse. They do not affect your score but it's important that you verify them manually.

- Captured at Apr 6, 2022, 1:17 PM GMT+5:30
- Emulated Moto G4 with Lighthouse 9.4.0
- Single page load
- Initial page load
- Slow 4G throttling
- Using Chromium 100.0.4896.60 with devtools

Generated by **Lighthouse** 9.4.0 | File an issue http://127.0.0.1:8000/

56 Performance

81 Accessibility

67 Best Practices

100 SEO

PWA

There were issues affecting this run of Lighthouse:

- **Chrome extensions negatively affected this page's load performance. Try auditing the page in incognito mode or from a Chrome profile without extensions.**

56

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

0–49 50–89 90–100

METRICS Expand view

First Contentful Paint 2.6 s	Time to Interactive 2.6 s
Speed Index 5.2 s	Total Blocking Time 0 ms
Largest Contentful Paint 7.2 s	Cumulative Layout Shift 0.003

View Original Trace

View Treemap

Show audits relevant to: All FCP TBT LCP CLS

OPPORTUNITIES

Opportunity	Estimated Savings
▲ Properly size images	8.8 s
▲ Serve images in next-gen formats	5.12 s
▲ Use video formats for animated content	4.52 s
▲ Efficiently encode images	3.64 s
▲ Eliminate render-blocking resources	2.31 s
▲ Preconnect to required origins	0.91 s
▲ Reduce initial server response time	0.72 s
Enable text compression	0.2 s

These suggestions can help your page load faster. They don't directly affect the Performance score.

DIAGNOSTICS

- ▲ Avoid enormous network payloads — Total size was 21,653 KiB
- ▲ Serve static assets with an efficient cache policy — 24 resources found
- ▲ Ensure text remains visible during webfont load
- ▲ Does not use passive listeners to improve scrolling performance
- ▲ Image elements do not have explicit `width` and `height`
- ▲ Minimize main-thread work — 4.3 s
- ○ Avoid chaining critical requests — 18 chains found
- ○ Keep request counts low and transfer sizes small — 54 requests • 21,653 KiB
- ○ Largest Contentful Paint element — 1 element found
- ○ Avoid large layout shifts — 4 elements found
- ○ Avoid long main-thread tasks — 11 long tasks found

Avoid non-composited animations — 1 animated element found

More information about the performance of your application. These numbers don't directly affect the Performance score.

PASSED AUDITS (20) Show

81

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Only a subset of accessibility issues can be automatically detected so manual testing is also encouraged.

NAMES AND LABELS

- Buttons do not have an accessible name
- Form elements do not have associated labels
- Links do not have a discernible name

These are opportunities to improve the semantics of the controls in your application. This may enhance the experience for users of assistive technology, like a screen reader.

CONTRAST

- Background and foreground colors do not have a sufficient contrast ratio.

These are opportunities to improve the legibility of your content.

NAVIGATION

- `[id]` attributes on active, focusable elements are not unique
- Heading elements are not in a sequentially-descending order

These are opportunities to improve keyboard navigation in your application.

ADDITIONAL ITEMS TO MANUALLY CHECK (10) Show

These items address areas which an automated testing tool cannot cover. Learn more in our guide on [conducting an accessibility review](#).

PASSED AUDITS (17) Show

NOT APPLICABLE (21) Show

67

Best Practices

TRUST AND SAFETY

- ▲ Does not use HTTPS — 1 insecure request found
- ▲ Requests the notification permission on page load
- ○ Ensure CSP is effective against XSS attacks

GENERAL

- ▲ Browser errors were logged to the console
- ▲ Issues were logged in the `Issues` panel in Chrome Devtools
- ○ Detected JavaScript libraries

PASSED AUDITS (9) Show

NOT APPLICABLE (1) Show

100

SEO

These checks ensure that your page is following basic search engine optimization advice. There are many additional factors Lighthouse does not score here that may affect your search ranking, including performance on [Core Web Vitals](). [Learn more]().

ADDITIONAL ITEMS TO MANUALLY CHECK (1) Show

Run these additional validators on your site to check additional SEO best practices.

PASSED AUDITS (12) Show

NOT APPLICABLE (2) Show

PWA

These checks validate the aspects of a Progressive Web App. [Learn more]().

INSTALLABLE

- Web app manifest and service worker meet the installability requirements

PWA OPTIMIZED

- Registers a service worker that controls page and start_url
- Configured for a custom splash screen
- Sets a theme color for the address bar.
- Content is sized correctly for the viewport
- Has a <meta name="viewport"> tag with width or initial-scale
- Provides a valid apple-touch-icon
- Manifest has a maskable icon

ADDITIONAL ITEMS TO MANUALLY CHECK (3) Show

These checks are required by the baseline PWA Checklist but are not automatically checked by Lighthouse. They do not affect your score but it's important that you verify them manually.

- Captured at Apr 6, 2022, 1:23 PM GMT+5:30
- Initial page load
- Emulated Desktop with Lighthouse 9.4.0
- Custom throttling
- Single page load
- Using Chromium 100.0.4896.60 with devtools

Generated by **Lighthouse** 9.4.0 | File an issue

Since 1962

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

Assignment No.	01
Assignment Title.	Select and submit 5 to 10 screenshots of the Application showcasing features you will be working on. Write a description in around 500 words of the same.
Roll No.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1, LO2, LO3
Grade:	

ASSIGNMENT NO. 1
MAD and PWA Lab

Aim: You are suggested to select features from the Application which you will be developing. The features should comprise of

1. Common widgets
2. Should include icons, images, charts etc.
3. Should have an interactive Form
4. Should apply navigation, routing and gestures
5. Should connect with FireBase database

You may select and submit 5 to 10 screenshots of the Application showcasing features you will be working on. Write a description in around 500 words of the same.

The application chosed is **Instagram**. Instagram from Facebook (now **Meta**) allows you to create and share your photos, stories, and videos with the friends and followers you care about. This app helps to connect with friends, share what you're up to, or see what's new from others all over the world.

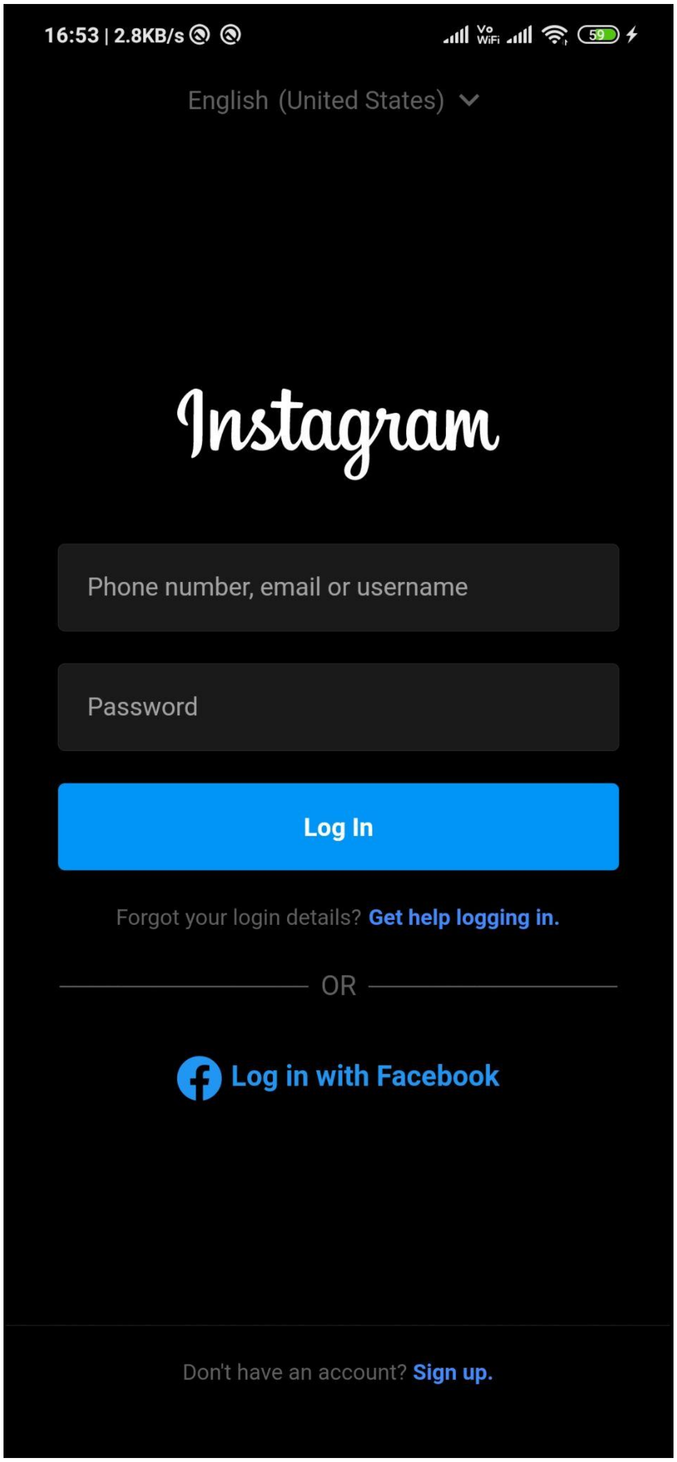
Firebase will be used for :

1. Storing the registration details which will be :
   - Full Name
   - Email
   - Password
   - Phone Number
   - Username
   - Bio
   - Profile Image

   The Email/Phone Number and Password will be used while logging into the application
2. Storing the posts uploaded by the user along with :
   - Likes on the post
   - Comments on the post
3. Storing the count of followers and following of the user

Interactive Form will be used in LogIn and Register part of the application. Also later onwards, will be implemented in editing the profile details in the Profile Screen.

Login Screen



This is the login screen. Here the widgets will be used for inserting :

Instagram Logo (SVG File)

Text field inputs for entering :
Phone number/Email/Username
Password

Log In button to log in to the home screen
Hyperlink for the "Sign up." which will redirect to the custom Sign up screen

Facebook Icon

Dropdown for choosing the language

Widgets which will be used here are :

1. Scaffold
2. SingleChildScrollView
3. Column
4. SizedBox
5. Row
6. Icon
7. SvgPicture
8. Container
9. Text
10. Inkwell

Home Screen

This is the home screen. Here the widgets will be used for :

Instagram Logo (SVG File)
Icon for Adding Post and Messenger Icon
Circle Image avatar for the Stories section
Each post has the Avatar of the user, Username and more options Icon

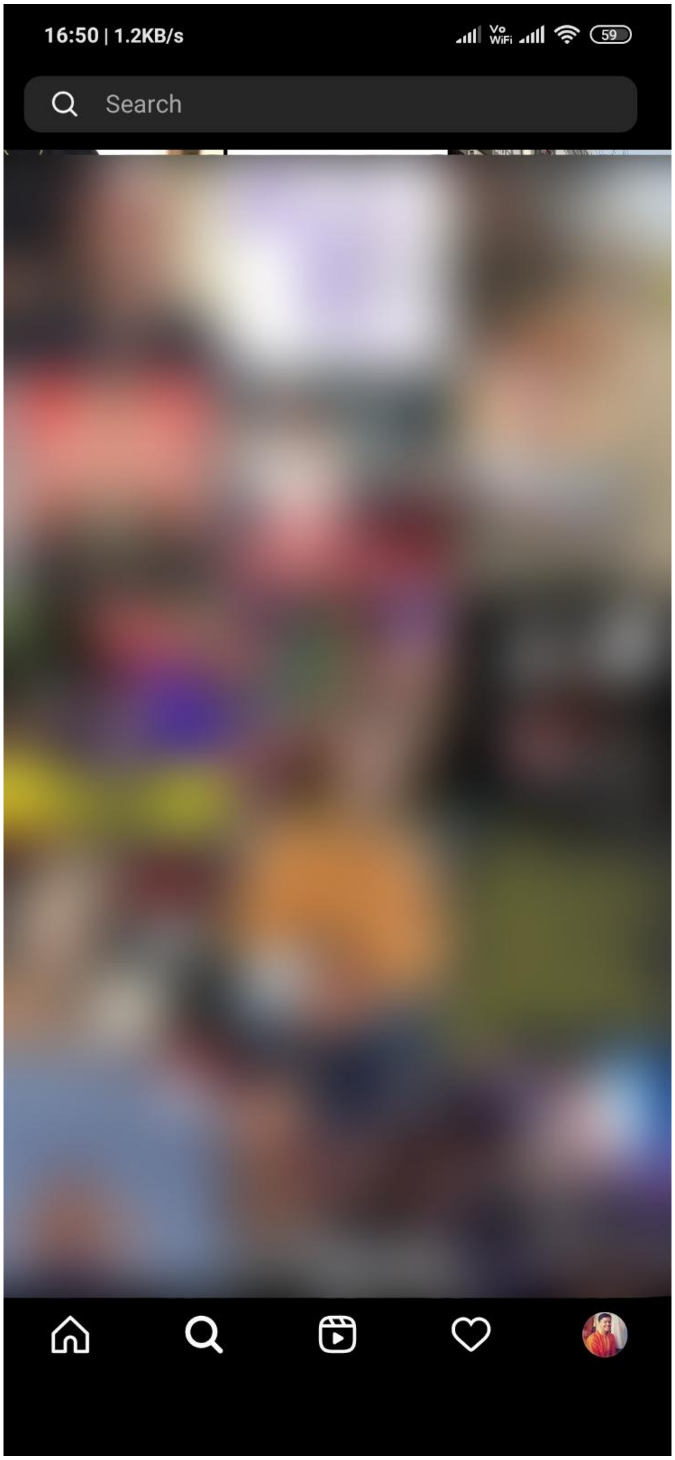
The post will have :
Like Icon to like the post
Comment Icon to comment on the post
Send Icon
Bookmark Icon

Bottom Navigation bar will consist of :
Home Icon to go to home screen
Search Icon to go to search screen
Reels Icon
Like Icon to see who has liked the posts
Profile Avatar Icon to go to profile screen

Widgets which will be used here are :

1. Scaffold
2. SingleChildScrollView
3. CircleAvatar
4. SizedBox
5. Row
6. Icon
7. SvgPicture
8. Container
9. Text
10. BottomNavigationBar

Search Screen (Posts)



This is the search screen to view all the posts together. Here the widgets will be used for :

Search bar with an Search Icon in the top
Positioning of the images of the posts on the screen

Comment Icon to comment on the post
Send Icon
Bookmark Icon

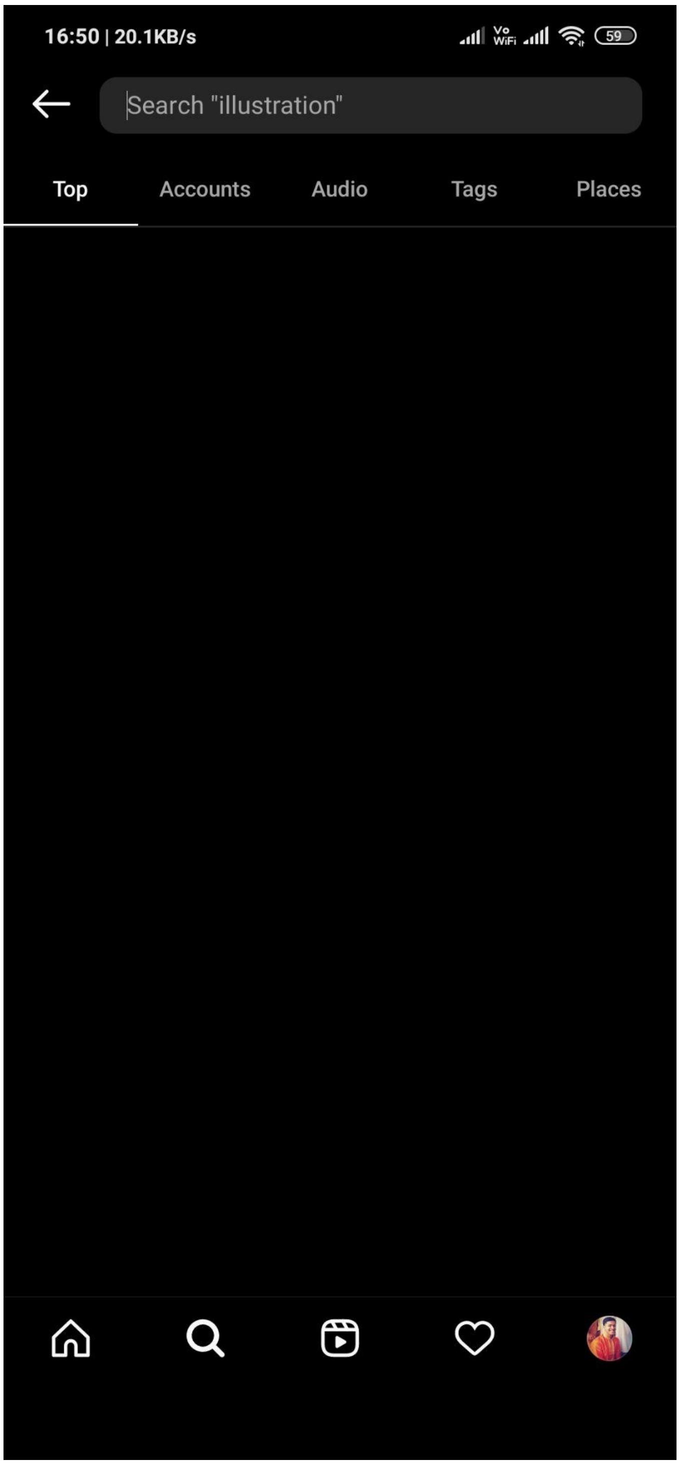
A row of description, no. of the likes it has and the comments section

Bottom Navigation bar will consist of :
Home Icon to go to home screen
Search Icon to go to search screen
Reels Icon
Like Icon to see who has liked the posts
Profile Avatar Icon to go to profile screen

Widgets which will be used here are :

1. Scaffold
2. SingleChildScrollView
3. Column
4. SizedBox
5. Row
6. Icon
7. SvgPicture
8. Container
9. Text
10. BottomNavigationBar

Search Screen (Users)



This is the search screen to search for users. Here the widgets will be used for :

The search bar will be used to search for users

A Left Sided Arrow Icon will show up when clicked on the search bar

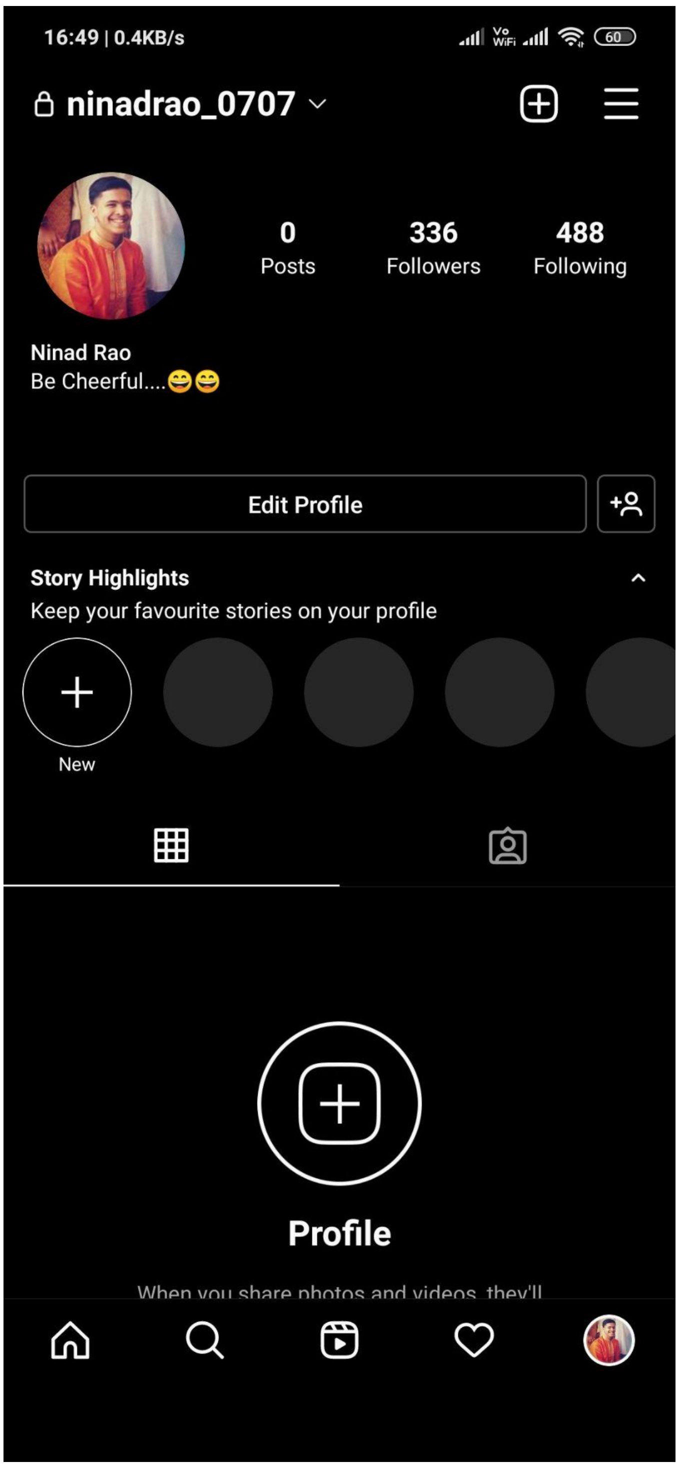
When searched for a user, it will redirect to the profile screen of the respective user
On seeing the profile screen of the user, there will be a button to follow/unfollow the user

Bottom Navigation bar will consist of :
Home Icon to go to home screen
Search Icon to go to search screen
Reels Icon
Like Icon to see who has liked the posts
Profile Avatar Icon to go to profile screen

Widgets which will be used here are :

1. Scaffold
2. InputDecoration
3. Column
4. SizedBox
5. Row
6. Icon
7. Form
8. ListView
9. AppBar
10. BottomNavigationBar

Profile Screen



This is the profile screen. Here the widgets will be used for :

A Lock Icon and a Downward Arrow Icon
Icon for Adding Post and More Options Icon
Profile Avatar Image of the user will be there

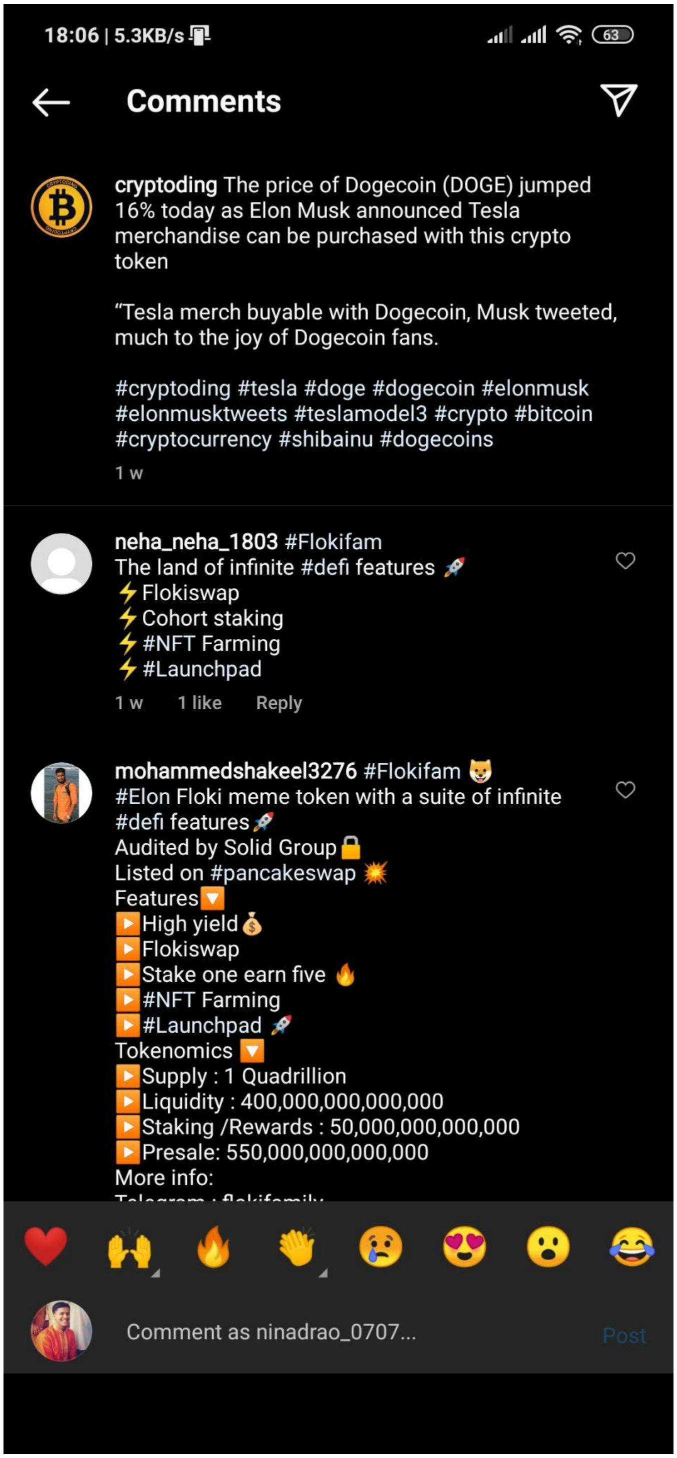
Edit Profile button will be there below the user details which will edit the profile details :
Profile Picture
Full Name
Description/Bio

Nav Tabs of 2 icons :
Post Icon which will have the user's posts
Tagged photos Icon
If there are no posts, there will be a Icon as shown with Profile and a short description written below it

Widgets which will be used here are :

1. Scaffold
2. InputDecoration
3. Column
4. SizedBox
5. Row
6. Icon
7. Form
8. BottomNavigationBar
9. Inkwell

Comments Screen



This is the comments screen. Here the widgets will be used for :

A Left Arrow Icon which will redirect to the home screen and a Send Icon on the right

Below, there will be 3 columns in which :
One column will have the user profile avatar Icon.
Second column will have the comments along with the description of the post.
Third column will have the Like Icon
In the bottom, there will be a row of emojis which will be used while entering a comment.

In the navigation bar, there will be :
User Profile avatar Icon
Text field to enter the comment
Hyperlink to post the comment

Widgets which will be used here are :

1. Scaffold
2. AppBar
3. Column
4. SizedBox
5. Row
6. Icon
7. ListView
8. CircleAvatar
9. Inkwell
10. Expanded

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

Assignment No.	02
Assignment Title.	To study the requirement for progressive web application for E-commerce using the concept of service worker, Web app Manifest and framework tools.
Roll No.	58
Name	Ninad Rao
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4, LO5, LO6
Grade:	

Vivekanand Education Society's
Institute of Technology
(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Since 1962

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 App

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 a 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, businesses 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. Capitalizing 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 present 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:

Webpack - Webpack-pwa-framework

Webpack is a static module bundler for modern JavaScript applications. Without such bundlers, running JS files 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.

Lighthouse - Lighthouse 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.