



Since 1962

Vivekanand Education Society's Institute of Technology

(An Autonomous Institute Affiliated to University of Mumbai)

Department of Information Technology

A.Y. 24-25

Mobile App Development and Progressive Web App

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment.
Roll No.	48
Name	Suyam Raut
Class	D15 B
Subject	MAD & PWA
Grade:	

Swayam Raut
D15B Roll No. 48

Experiment 1: Installation and Configuration of Flutter Environment.

(A)

Install the Flutter SDK

Step 1: Visit the flutter website and navigate to the section Windows based installation.

Docs Showcase Comm

Get started with Flutter 2.2. See What's new in docs, including a list of the new Instructor-led video

Flutter documentation



Get Started	Widgets Catalog	API Docs
Set up your environment and start building	Dip into the rich set of Flutter widgets available in the SDK.	Bookmark the API reference for the Flutter framework.
Cookbook	Samples	Videos
Browse the cookbook for many easy Flutter recipes.	Check out the Flutter examples.	View the many videos on the Flutter YouTube channel.

What's new on this site

To see changes to the site since our last release, see What's new

Install

Docs > Get started > Install

Select the operating system on which you are installing Flutter:



Windows



macOS



Linux



Chrome OS

Swayam Raut
D15B Roll No. 48

Windows install

Docs > Get started > Install > Windows

System requirements



To install and run Flutter, your development environment must meet these minimum requirements:

- **Operating Systems:** Windows 7 SP1 or later (64-bit), x86-64 based.
- **Disk Space:** 1.64 GB (does not include disk space for IDE/tools).
- **Tools:** Flutter depends on these tools being available in your environment.
 - Windows PowerShell 5.0 or newer (this is pre-installed with Windows 10)
 - Git for Windows 2.x, with the **Use Git from the Windows Command Prompt** option.

If Git for Windows is already installed, make sure you can run git commands from the command prompt or PowerShell.

Step 2: Download the zip file for Flutter SDK

Get the Flutter SDK

1. Download the following installation bundle to get the latest stable release of the Flutter SDK:

flutter_windows_2.2.3-stable.zip

For other release channels, and older builds, see the [SDK releases page](#).

2. Extract the zip file and place the contained flutter in the desired installation location for the Flutter SDK (for example, C:\Users\<your-user-name>\Documents).

⚠ Warning: Do not install Flutter in a directory like C:\Program Files\ that requires elevated privileges.

If you don't want to install a fixed version of the installation bundle, you can skip steps 1 and 2. Instead, get the source code of the Flutter repo on GitHub, and change branches or tags as needed. For example:

C:\src>git clone https://github.com/flutter/flutter.git -b stable

You are now ready to run Flutter commands in the Flutter Console.

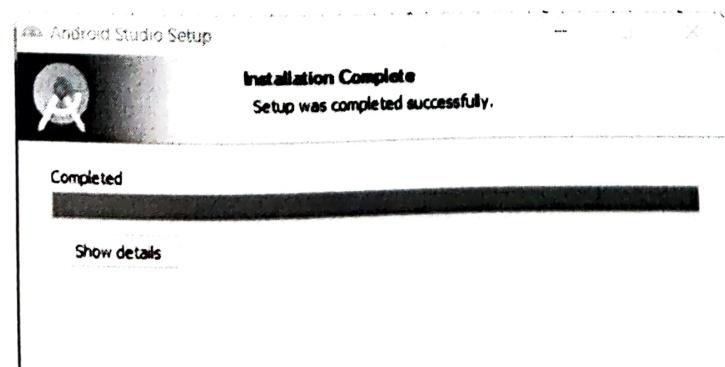
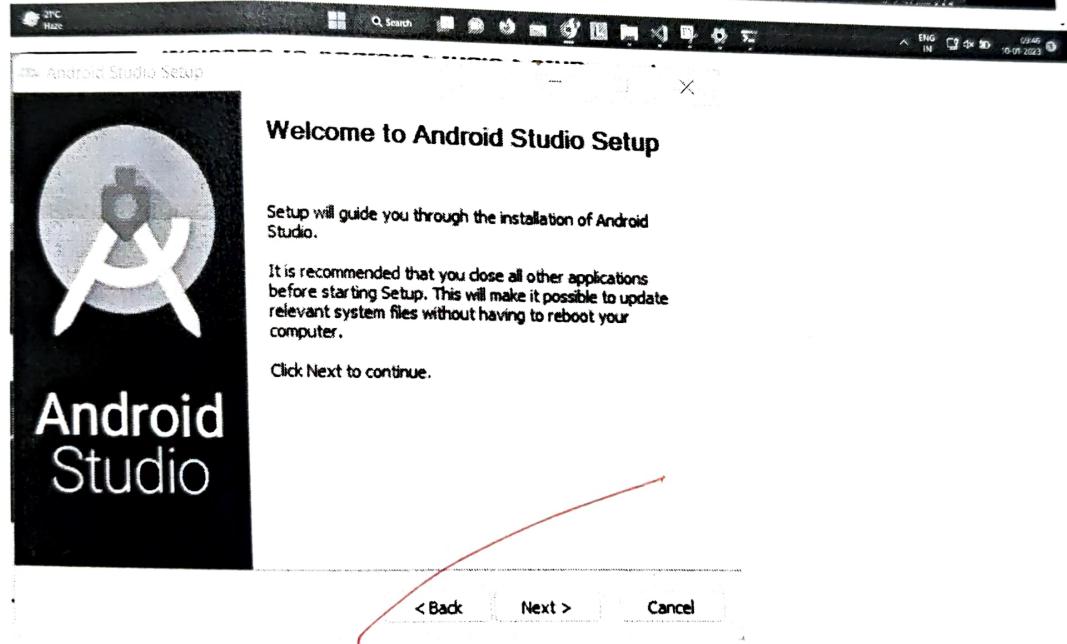
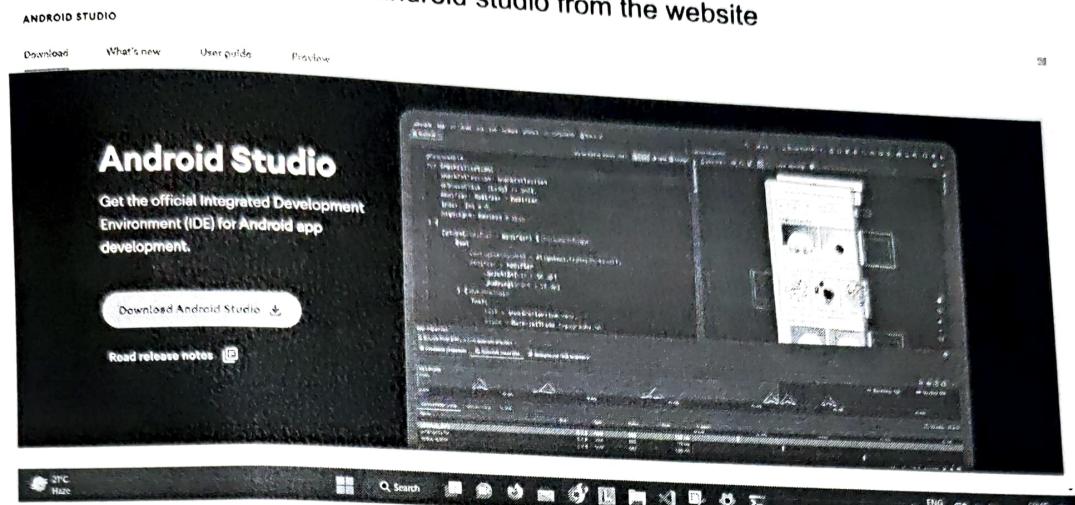
Step 3: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory.

Swayam Raut
D15B Roll No. 48

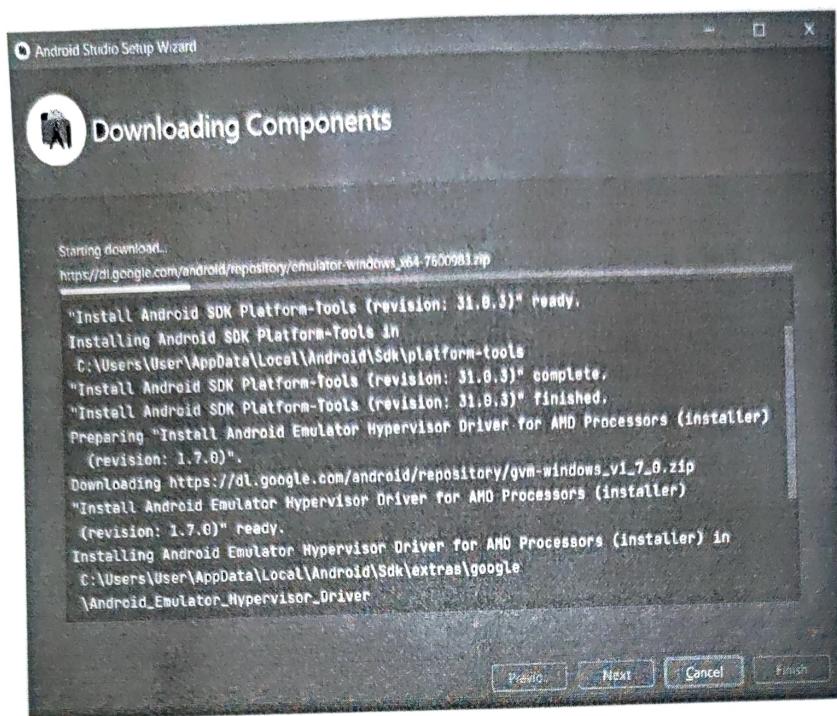


Step 4: Run the flutter doctor command

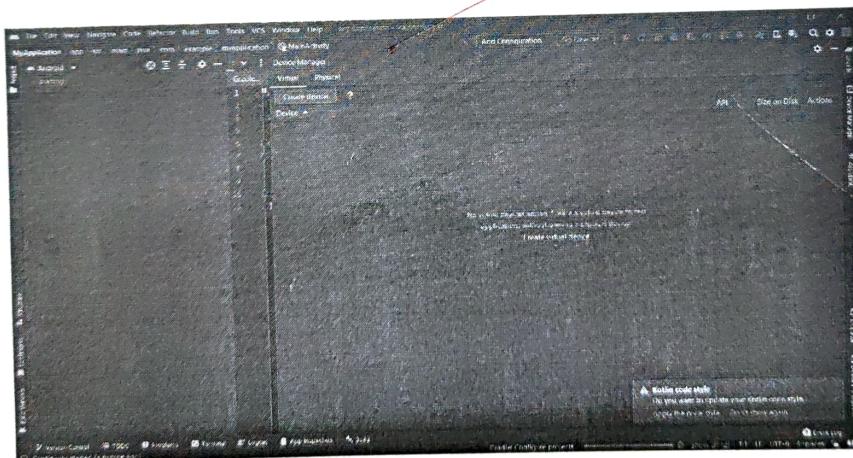
Step 6: Download and install android studio from the website



Swayam Raut
D15B Roll No. 48

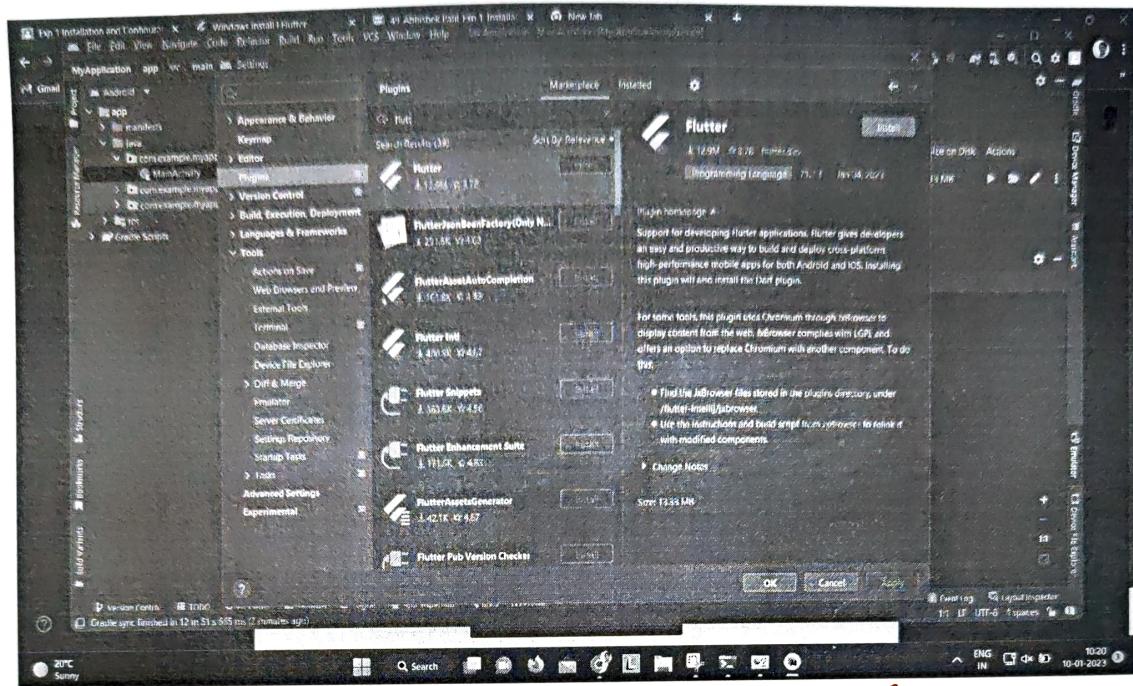


Step 8: Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.



Swayam Raut
D15B Roll No. 48

Step 10: Install the plugin for Dart and rt and Flutter



Now, we could use Android studio to build apps using Flutter.

Conclusion: In this lab experiment we learnt to configure Android Studio and make it compatible with Flutter based application development using Dart.

INo.48 Experiment - II

Jo
A*

m. Basic Flutter UI ~~design~~ design with common Widgets.

To design & implement flutter interface incorporating common widgets such as text fields, buttons, switches, sliders and other interactive elements to understand basic flutter

Procedure

1. Project Setup

- Create new project on android studio.
- Set up main.dart as an entry point.
- Configure basic Material App structure.

2. Implementation Steps:

- Created a stateful widget for the form screen.
- Implement column widget
- Add the following widget:
 - Image Widget
 - Text Field
 - Switch Widget
 - Slider Widget
 - Custom dialog of country selection
 - Check box for terms & conditions.
 - Button widget for form submissions.

3. Testing

- Tested widget rendering & layout.
- Verified component interaction.
- Checked form responsiveness.

Results :-

- Properly structured widget tree
- Working form elements
- Responsive layout with proper spa

Aim: To design a flutter app by including basic widgets.

main.dart (File that contains the UI Layout)

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: FormScreen(),
    );
  }
}

class FormScreen extends StatefulWidget {
  @override
  _FormScreenState createState() => _FormScreenState();
}

class _FormScreenState extends State<FormScreen> {
  // State variables
  String _nameText = "";
  String _passwordText = "";
  bool _notificationsEnabled = false;
  double _volumeValue = 0.0;
  String _selectedCountry = 'Select a country';
  bool _agreedToTerms = false;

  // List of countries for dropdown
  final List<String> _countries = ['USA', 'India', 'Germany', 'Japan'];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              // Image
              Image.asset('assets/icon.png',
                width: 100,
                height: 100,
              ),
              // Text Input
              TextFormField(
                decoration: InputDecoration(
                  labelText: 'Name',
                  border: OutlineInputBorder(),
                ),
                onChanged: (value) {
                  setState(() {
                    _nameText = value;
                  });
                },
              ),
              // Password Input
              TextFormField(
                decoration: InputDecoration(
                  labelText: 'Password',
                  border: OutlineInputBorder(),
                ),
                obscureText: true,
                onChanged: (value) {
                  setState(() {
                    _passwordText = value;
                  });
                },
              ),
              // Notifications Checkbox
              SwitchListTile(
                title: Text('Notifications'),
                value: _notificationsEnabled,
                onChanged: (value) {
                  setState(() {
                    _notificationsEnabled = value;
                  });
                },
              ),
              // Volume Slider
              Slider(
                value: _volumeValue,
                min: 0.0,
                max: 1.0,
                onChanged: (value) {
                  setState(() {
                    _volumeValue = value;
                  });
                },
              ),
              // Country Dropdown
              DropdownButton(
                value: _selectedCountry,
                items: _countries.map((country) {
                  return DropdownMenuItem(
                    value: country,
                    child: Text(country),
                  );
                }).toList(),
                onChanged: (value) {
                  setState(() {
                    _selectedCountry = value;
                  });
                },
              ),
              // Terms Agreement Checkbox
              CheckboxListTile(
                title: Text('I agree to the terms and conditions'),
                value: _agreedToTerms,
                onChanged: (value) {
                  setState(() {
                    _agreedToTerms = value;
                  });
                },
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

```
    setState() {
        _notificationsEnabled = value;
    });
},
],
),
).
```

SizedBox(height: 16),

```
// Volume Slider
Column(
  children: [
    Text('Volume: ${_volumeValue.toInt()}'),
    Slider(
      value: _volumeValue,
      min: 0,
      max: 100,
      onChanged: (double value) {
        setState(() {
          _volumeValue = value;
        });
      },
    ),
  ],
),
).
```

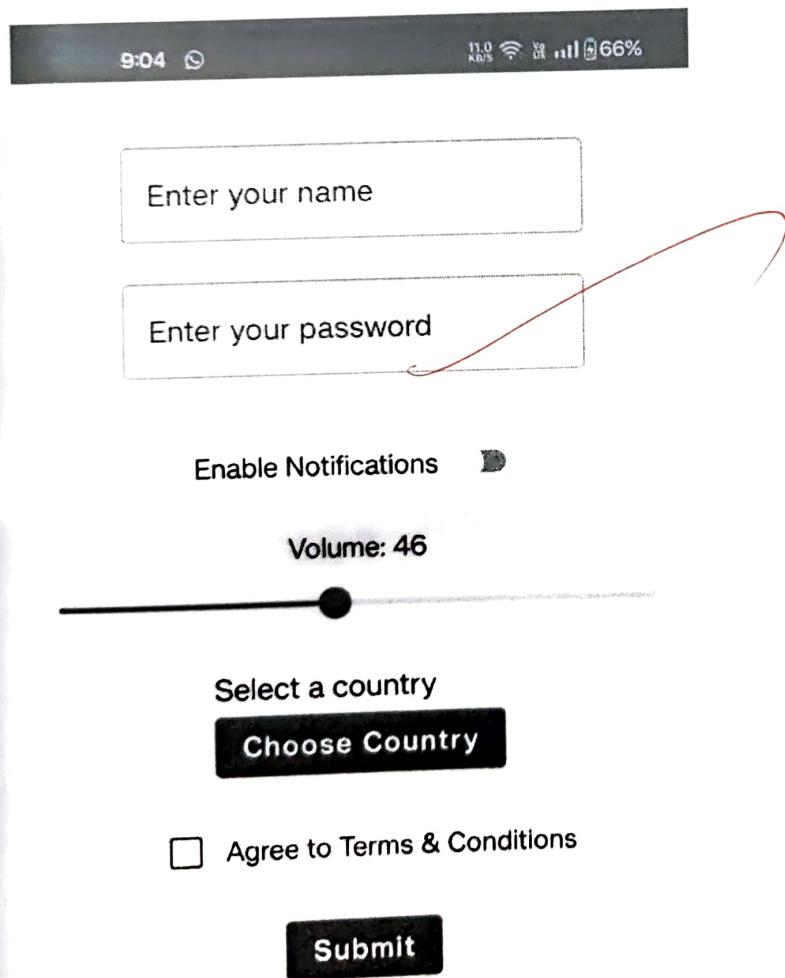
SizedBox(height: 16),

```
// Country Dropdown
Column(
  children: [
    Text(
      _selectedCountry,
      style: TextStyle(fontSize: 16, color: Colors.black),
    ),
    ElevatedButton(
      onPressed: () {
        _showCountryDialog();
      },
      child: Text('Choose Country'),
    ),
  ],
),
).
```

SizedBox(height: 16),

```
});  
    Navigator.of(context).pop();  
},  
);  
}).toList(),  
,  
);  
};  
}  
}
```

Output



The following custom components have been used:

- 1) Dropdown
- 2) Checkbox
- 3) Notification slider

Select a country

Choose Country

USA Terms & Conditions

India

Germany

Japan

Agree to Terms & Conditions

Submit

Enable Notifications 

Conclusion: In this lab experiment, we acquainted ourselves with the fundamental concepts of Dart programming language and learnt to import and position custom UI components using it.





Vivekanand Education Society's

Institute of Technology

(An Autonomous Institute Affiliated to University of Mumbai)

Department of Information Technology

A.Y. 24-25

Mobile App Development and Progressive Web App Lab

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	21 P
Name	Suyam Raut
Class	D15 B
Subject	MAD & PWA
Grade:	<i>S (A)</i>

Experiment - III

Enhanced flutter UI with Icons, Images, and custom fonts

Aim:- To enhance the basic flutter UI by incorporating ~~Material~~ Material icons, custom images, and Google fonts to create a more polished and professional user interface.

Procedure

1. Set Up & config

- Add google fonts dependency to pubspec.yaml
- Configure assets directory for images
- Setup necessary material icon imports

2. Implementation Steps

Enhanced UI components with icons:

- Added person icon to name field
- Added notification icon to switch
- Added check icon to submit button

Testing

Verified proper loading of custom fonts

Checked rich rendering and re-alignment.

Tested overall UI consistency.

Results:-

Material icons integrated through the ~~interfas~~ interface

Custom font called Hack applied ~~consistently~~ consistently

Improved visual hierarchy and UX.

Professional looking interface element.

• ~~Conclusion:~~

- ~~Basic~~ ~~Flutter~~ widgets provide foundation for building interactive UI.
- Material Design icons and custom fonts significantly enhance UI aesthetics.
- ~~Proper~~ Flutter's widget system effectively handles complex layouts.
- Proper planning of widget structure crucial for maintainable code.

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

main.dart

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

```
void main() {
  runApp(MyApp());
}
```

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: FormScreen(),
    );
  }
}
```

```
class FormScreen extends StatefulWidget {
  @override
  _FormScreenState createState() => _FormScreenState();
}
```

```
class _FormScreenState extends State<FormScreen> {
  String _nameText = "";
  String _passwordText = "";
  bool _notificationsEnabled = false;
  double _ageValue = 18.0;
  String _selectedCountry = 'Select a country';
  bool _agreedToTerms = false;

  final List<String> _countries = ['USA', 'India', 'Germany', 'Japan'];
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: SingleChildScrollView(
      child: Padding(
```

```
onChanged: (value) {
    setState(() {
        _passwordText = value;
    });
},
),
SizedBox(height: 16),

// Notifications Switch with Icon
Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
        Icon(Icons.notifications),
        Text('Enable Notifications', style: GoogleFonts.hack()),
        Switch(
            value: _notificationsEnabled,
            onChanged: (bool value) {
                setState(() {
                    _notificationsEnabled = value;
                });
            },
        ),
    ],
),
SizedBox(height: 16),

// Age Slider
Column(
    children: [
        Text(
            'Select your age: ${_ageValue.toInt()}',
            style: GoogleFonts.hack(),
        ),
        Slider(
            value: _ageValue,
            min: 18,
            max: 100,
            onChanged: (double value) {
```

Output:



Enter your name

Enter your password

Enable Notifications

Select your age: 69

India

Agree to Terms & Conditions

Here, we imported a custom font called hack.ttf
Additionally, we also relied on external components for MaterialUI library
We also added an image as a widget.





Vivekanand Education Society's Institute of Technology

(An Autonomous Institute Affiliated to University of Mumbai)

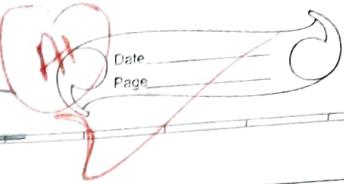
Department of Information Technology

A.Y. 24-25

Mobile App Development and Progressive Web App Lab

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget.
Roll No.	
Name	
Class	D15 B
Subject	MAD & PWA
Grade:	

MPL Experiment - IV



Aim: Create an interactive form using form widget.

Theory:

In this experiment, we create login form with toggle functionality for sign up. Login and Sign up functionality was implemented through authentication on firestore.

Key widgets used:

TextField, ElevatedButton, TextButton, and Image

Key features implemented

~~TextButton~~: Provides a way to switch between the login & registration forms.

~~Image~~: Displays the image of the logo

~~State management~~: The `_isRegister` boolean variable determines whether the form displays login or registration. `SetState` is used to build UI.

~~Firebase Integration~~: We first configure firestore for your project & replace placeholder values. The `_loginUser` & `_registerUser` functions handle the Firebase authentication process.

Swayam Raut
D15 B Roll No. 48

MAD Experiment 4

Aim: Create an interactive form using form widget.



Register

Enter your name

Email

Password

Sign Up

Login

Email

Password

Login

No account? Sign up

Already have an account? [Login](#)

Registration Page

Login Page

Both the pages are linked to Firebase and store user data in FireStore. The plugins have been configured in a manner that fetches the user data at the time of logging in.

Firebase User Data

The screenshot shows the Firebase Authentication console with a dark theme. At the top, there's a banner about Dynamic Links shutting down. Below it is a search bar and an 'Add user' button. The main area is a table with columns: Identifier, Providers, Created, Signed in, and User UID. The data shows multiple users created on 24 Feb 2025, each with a unique email address and a corresponding User UID.

Identifier	Providers	Created	Signed in	User UID
swayam@ymail.com	✉	24 Feb 2025	24 Feb 2025	67uK8MbYhbw90t7y13zq10l...
swayamraut252@gmail...	✉	24 Feb 2025	24 Feb 2025	HWkUW3qG9IMZobZc2f9F14...
swayamraut2@gmail.c...	✉	24 Feb 2025	24 Feb 2025	OH4PlqzMOSYIKngk0z5mSf5...
swayamraut3@gmail.c...	✉	24 Feb 2025	24 Feb 2025	vNjXnmTaQ0N4bpaF5JhYUN...
swayamraut@gmail.com	✉	24 Feb 2025	24 Feb 2025	OwTFSHLGKTemc4udAFir9iV...
swayamraut1@gmail.c...	✉	24 Feb 2025	24 Feb 2025	Y3o6i53BdXSVla2ZVCFNelH3...
w1@gmail.com	✉	24 Feb 2025	24 Feb 2025	SFvgWJHqL8ckryaEYV13lzW...

Conclusion: In this experiment, we learn to use various form elements from the flutter library and integrated it with Firebase.



Vivekanand Education Society's

Institute of Technology

(An Autonomous Institute Affiliated to University of Mumbai)

Department of Information Technology

A.Y. 24-25

Mobile App Development and Progressive Web App Lab

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App.
Roll No.	48
Name	Sudayam Raut
Class	D15 B
Subject	MAD & PWA
Grade:	<i>So</i> <i>At</i>

MAD Experiment - I

Page No.

A

To apply navigation, routing and gestures in flutter App based on the same code.

Theory:

In this ~~→~~ lab, ~~we~~ after implementing authentication using firebase, we created custom routes that serve ~~to~~ different pages based on the route selected and the account that the user is logged in.

~~Key~~ features for navigation, routing and gestures:

- 1) ~~Named Routes~~: Defined named routes in Material App's routes property. They makes navigation cleaner.
- 2) ~~Navigation and named Routes~~: Used Navigator.pushReplacement Named ensures
- 3) ~~Gestures Detection~~: Wrapped the ScanProduct Screen with a GestureDetector to detect taps & double taps. You can add others.
- 4) ~~Logout Navigation~~: In Main Screen, the user now navigates
- 5) ~~StreamBuilder Integration~~: The StreamBuilder in MyApp now uses named routes to navigate to appropriate screen based on the authentication state.

Conclusion:

With these changes, you have an app with navigation between screens, swiping using named routes, and gesture detection. You can expand by adding more complex navigation logic, passing data between screens using routes, and implementing more sophisticated gestures interaction.

Swayam Raut
D15B Roll No. 48

MAD Experiment 5

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



Scan Product QR Page

Login

Email:

Password:

Login

No account? Sign up



Successful login state achieved using Firebase

View Product Page

Register Complaint Page



In-app navigation achieved through built in app routes

Additionally, logout functionality was also implemented which, on clicking, exits the navigation interface and returns the user back to the login screen.

Conclusion: In this experiment, we learnt to set routes within flutter applications to help navigate across multiple pages of the app. We also learnt to navigate to various screens through authentication using firebase. Logout functionality was also implemented which provided a route back to the login interface.





Vivekanand Education Society's Institute of Technology

(An Autonomous Institute Affiliated to University of Mumbai)

Department of Information Technology

A.Y. 24-25

Mobile App Development and Progressive Web App Lab

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	28
Name	Sudayam Rauli
Class	D15 B
Subject	MAD & PWA
Grade:	 

MAD Experiment - VI

Aim: To demonstrate setting up firebase (Authentication, and firestore) in a cross-platform flutter app for iOS & android.

Theory:

Firebase simplifies backend development. ~~firebase - core~~ initializes firebase. ~~firebase - auth~~ handles user authentication (login / registration). ~~cloud - firestore~~, stores data. we use email / password authentication, and basic firebase user collection.

Implementation:

- ① Firebase Setup: Create a project on the ~~firebase~~ website.
- ② Dependencies: Add ~~firebase - core~~, ~~firebase - auth~~, ~~cloud - firestore~~ to ~~pubspec.yaml~~.
- ③ Initialization: ~~firebase.initializeApp()~~ in main.dart
- ④ Authentication: Use ~~firebaseAuth~~ for login / registration
- ⑤ Firestore: Store user data in a "users" collection
- ⑥ Navigation: Use named routes and StreamBuilder for auth state management
- ⑦ UI: TextFormField, ElevatedButton, TextButton for login / registration forms.

Necessary Considerations:

- ~~firebase console setup~~
- Error handling
- Correct Platform ~~conf~~ configuration files

Conclusion:

This experiment demonstrated firebase integration in flutter. Proper setup and error handling are crucial.

This allows for simple authentication and data storage ~~across~~ platforms.

Aim: To set up Firebase authentication for the android application.

1. firebase_core Initialization:

- await Firebase.initializeApp(); is crucial. This line initializes Firebase for your app. It's placed in the main() function *before* runApp().
- WidgetsFlutterBinding.ensureInitialized(); is also important, and is required before Firebase.initializeApp().

2. Firebase Authentication (firebase_auth):

- The code uses FirebaseAuth for user authentication.
- signInWithEmailAndPassword and createUserWithEmailAndPassword are used for login and registration.
- Error handling is improved with FirebaseAuthException catching, providing user friendly messages.

3. Cloud Firestore (cloud_firestore):

- FirebaseFirestore is used to store user data in the "users" collection.
- set() is used to write data to Firestore.

4. Platform Setup (iOS and Android):

- To make this work, you *must* follow the Firebase console setup for both iOS and Android. This involves:
 - Creating a Firebase project.
 - Adding iOS and Android apps to your project.
 - Downloading the google-services.json (Android) and GoogleService-Info.plist (iOS) files and placing them in the correct locations within your Flutter project.
 - Ensuring the correct Firebase plugins are added to your pubspec.yaml.

5. Error Handling: FirebaseAuth exceptions are now specifically handled, providing better error messages to the user.

6. Navigation: Navigation is implemented using named routes, allowing for cleaner transitions.

We need the following packages;

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
```

snapshot

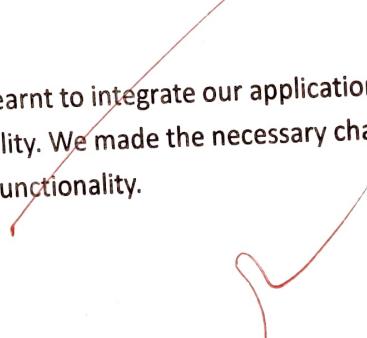
The following authentication features will stop working when Firebase Dynamic Links shuts down on 25 August 2025: email link authentication for mobile apps, as well as Cordova OAuth support for web apps.

Search by email address, phone number or user UID

Add user

Identifier	Providers	Created	Signed In	User UID
sw@gmai.com	✉️	25 Feb 2025	25 Feb 2025	FcyCyr4d25hBqtv5LocgmAQ...
hola@1234.com	✉️	25 Feb 2025	25 Feb 2025	HEsW6Alvtzlq16HiVXm5Ne0...
aayeg@gmail.com	✉️	25 Feb 2025	25 Feb 2025	ISHFIQIEzPYMQAmRBC0wMl...
swayam@ymail.com	✉️	24 Feb 2025	24 Feb 2025	67uK8MbVhbw90l7y13zq1O...
swayamraut252@gmail...	✉️	24 Feb 2025	24 Feb 2025	HWkUW3qG9iMZobZcz819F1...
swayamraut2@gmail.c...	✉️	24 Feb 2025	24 Feb 2025	OH4PlqzMOSYIKnqk0z5mSf5...
swayamraut3@gmail.c...	✉️	24 Feb 2025	24 Feb 2025	vNjXnmTaQDN4bpaF5JhYUN...
swayamraut@gmail.com	✉️	24 Feb 2025	25 Feb 2025	OwTFSHLGKTmc4udAFir9IV...

Conclusion: In this experiment, we learnt to integrate our application with firebase and implement authentication functionality. We made the necessary changes to our flutter program to implement the desired functionality.





Vivekanand Education Society's

Institute of Technology

(An Autonomous Institute Affiliated to University of Mumbai)

Department of Information Technology

A.Y. 24-25

Mobile App Development and Progressive Web App Lab

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.	43
Name	Sudayum Paul
Class	D15 B
Subject	MAD & PWA
Grade:	<i>S</i> / <i>ATI</i>

MPL

~~MPL Experiment - VII~~

(A+)

- m. Add to your home screen feature on a web application

Progressive Web Apps

PWAs are built with web technologies (HTML, CSS, JS) that offer user experience similar to native apps, allowing them to be installed on devices and work offline, while also being accessible through a web browser.

Features:

- 1) Built with HTML, ~~CSS~~, CSS, JS
- 2) App-like interface
- 3) Offline functionality
- 4) Background updates
- 5) Directly installed from the web browser.

In this experiment, to set up a basic PWA,
 we create a file called manifest.json
 where we define the basic configuration details
 of the application and include 2 files

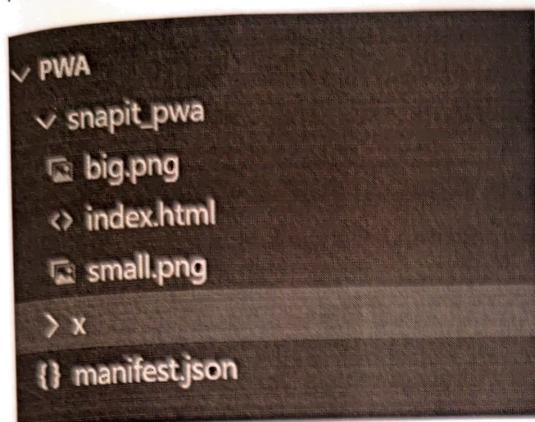
- (i) small.png
- (ii) large.png

These 2 files are mandatory as they represent
 the icon that PWA will be installed under.

Conclusion

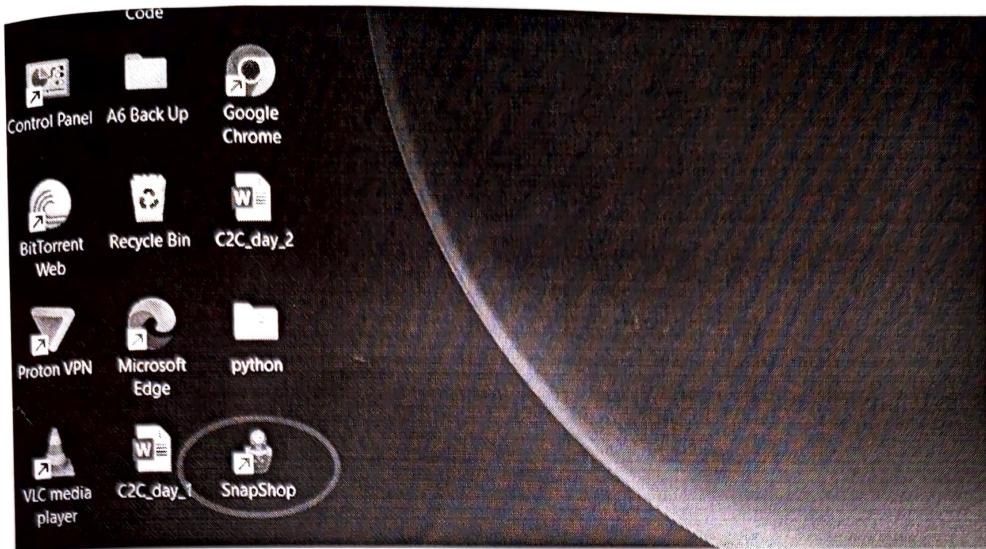
In this experiment we learnt to set up a PWA by defining manifest.json file. This PWA could be installed on visiting its address through a browser on a mobile phone.

Aim: Add to your home screen feature on a web application.



The screenshot shows a code editor with a dark theme. The file being edited is 'manifest.json'. The code defines a service worker object with various properties like name, short_name, description, start_url, display, background_color, theme_color, and icons. The icons section contains two entries: one for a small icon (192x192) and one for a big icon (512x512), both in image/png format.

```
1  {
2      "name": "SnapShop",
3      "short_name": "SnapShop",
4      "description": "A QR-based shopping app",
5      "start_url": "index.html",
6      "display": "standalone",
7      "background_color": "#f0f2f5",
8      "theme_color": "#0ea655",
9      "icons": [
10         {
11             "src": "small.png",
12             "sizes": "192x192",
13             "type": "image/png"
14         },
15         {
16             "src": "big.png",
17             "sizes": "512x512",
18             "type": "image/png"
19         }
20     ]
21 }
22 }
```



Conclusion: Through this experiment we learnt to add the 'add to my webpage' feature to our web application. This is the most fundamental step to be performed while building progressive web applications.

R



Vivekanand Education Society's

Institute of Technology

(An Autonomous Institute Affiliated to University of Mumbai)

Department of Information Technology

A.Y. 24-25

Mobile App Development and Progressive Web App Lab

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.	48
Name	Sudayam Ramta
Class	D15 B
Subject	MAD & PWA
Grade:	<i>S</i> <i>(A+)</i>

MPL Experiment - ⑧

To code and register a service worker and complete the install and activation process for SnapIt PWA.

Service Worker Life Cycle:

A service is a JavaScript script that runs ~~independently~~ in the background, separate from main web page, ~~enabling features like offline functionality, caching, and push notifications.~~ Unlike traditional scripts, service ~~workers~~ do not have direct access to the DOM but can intercept network requests and cache resources, ensuring that a PWA remains functional even when the user ~~is~~ is offline.

It consists of the following steps:

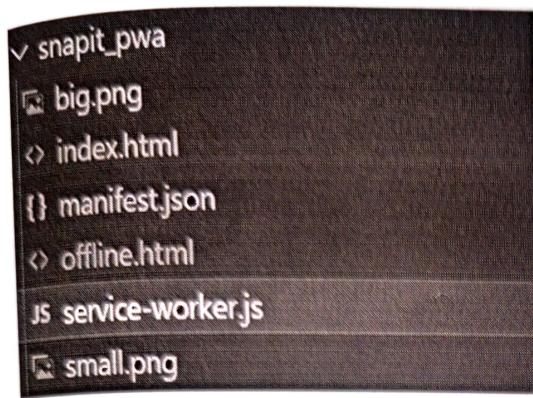
- 1) Registration
- 2) Installation
- 3) Activation

The fetch event is responsible for intercepting network requests and deciding whether to serve cached content or fetch new content from the internet. If a user is offline, the service worker serves cached file. Caching improves performance by reducing network requests.

Conclusion: By implementing a ~~service worker with~~ caching, we successfully implemented offline functionality and improved performance for the web app.

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.

Create service-worker.js



Create a cacheable file called offline.html to be displayed in the absence of an internet connection.

```
offline.html U X
snapit_pwa > offline.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Offline</title>
7      <style>
8          body {
9              font-family: Arial, sans-serif;
10             text-align: center;
11             padding: 50px;
12         }
13     </style>
14 </head>
15 <body>
16     <h1>You're Offline</h1>
17     <p>It looks like you have lost internet access. Some features may not work.</p>
18     
19 </body>
20 </html>
```

The screenshot shows the Chrome DevTools Application tab with the 'Service workers' section selected. The left sidebar lists various storage types: Manifest, Service workers, and Storage (including Local storage, Session storage, Extension storage, IndexedDB, Cookies, Private state, Interest groups, Shared storage, Cache storage, and Storage buckets). The main panel displays information about a service worker:

- Source:** service-worker.js (version 5)
- Received:** 17/3/2025, 11:32:18 pm
- Status:** #3147 activated and is running (with a Stop button)
- Push:** Test push message from DevTools (with a Push button)
- Sync:** test-tag-from-devtools (with a Sync button)
- Periodic sync:** test-tag-from-devtools (with a Periodic sync button)
- Update Cycle:** Version #3147 (Install, Wait, Activate), Update Activity, Timeline.

At the bottom, there are links for "Service workers from other origins" and "See all registrations".

The screenshot shows the configuration for the SnapShop app manifest. It includes the following sections:

- App Manifest**: Shows the file manifest.json.
- Errors and warnings**:
 - A warning: Richer PWA Install UI won't be available on desktop. Please add at least one screenshot with the form_factor set to wide.
 - A warning: Richer PWA Install UI won't be available on mobile. Please add at least one screenshot for which form_factor is not set or set to a value other than wide.
- Identity**:
 - Name: SnapShop
 - Short name: SnapShop
 - Description: A QR-based shopping app
 - Computed App ID: http://127.0.0.1:5500/snapit_pwa/index.html (with a Learn more link)
 - Note: id is not specified in the manifest, start_url is used instead. To specify an App ID that matches the current identity, set the id field to /snapit_pwa/index.html.
- Presentation**:
 - Start URL: index.html



Vivekanand Education Society's Institute of Technology

(An Autonomous Institute Affiliated to University of Mumbai)

Department of Information Technology

A.Y. 24-25

Mobile App Development and Progressive Web App Lab

Experiment No.	09
Experiment Title.	To implement Service worker events like fetch, sync and push for E-commerce PWA.
Roll No.	48
Name	Swayam Ranit
Class	D15 B
Subject	MAD & PWA
Grade:	 

~~MPL Experiment - 9~~

A.P.P.

Aim: To implement service worker events like fetch sync, and push for the SnapIt PWA.

Enhancing the service work with advanced network control and notifications.

~~But it's~~

Enhanced Fetch event

Fetch event was upgraded to follow both network-first and cache-first strategies.

- For requests that fetch dynamic content (such as API calls), the service worker tries to retrieve fresh content from the internet.
- For static resources like images and HTML files, the cache-first strategy being used.

Sync Event (for handling offline requests)

- Enables data to be stored temporarily when the device is offline.
- It ensures that no data is lost due to poor connectivity and allow user to continue using application in offline mode.

Push Event (Real-time notifications)

- Enables the service worker to listen for notifications sent from the server and display them to the user.
- Push notification enhance user engagement by delivering updates such as alerts, reminders, etc.

Conclusion:

We implemented advanced service worker events,

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

http://127.0.0.1:5500/snapit_pwa/ Network requests Update Unregister

Source service-worker.js

Received 17/3/2025, 11:32:18 pm

Status ● #3147 activated and is running **Stop**

Clients http://127.0.0.1:5500/snapit_pwa/index.html

Push **Test push message from DevTools.** **Push**

Sync **test-tag-from-devtoolsee** **Sync**

Periodic sync **test-tag-from-devtools** **Periodic sync**

Update Cycle

Version	Update Activity	Timeline
► #3147	Install	
► #3147	Wait	
► #3147	Activate	[Progress Bar]

Make the following changes to the service-worker.js

// Install Event: Cache assets

// Activate Event: Cleanup old caches

// Fetch Event: Supports both Cache-First & Network-First

// Sync Event: Retry sending data when online

// Function to send pending screenshots to the server

// Push Event: Display push notifications



Vivekanand Education Society's Institute of Technology

(An Autonomous Institute Affiliated to University of Mumbai)

Department of Information Technology

A.Y. 24-25

Mobile App Development and Progressive Web App Lab

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	46
Name	Sudayum Rath
Class	D15 B
Subject	MAD & PWA
Grade:	

- Page No. A/P
Date: 2023/07/10
- 48 MPL Experiment - 10
- To deploy the PWA on github pages, making it accessible online with live URL.
- ## Deployment Process and configuration
- Preparing the repository
Add all the files to a github repository
 - Enabling of github pages
• Go to the setting page of the repository and enable github pages
• Branch containing the PWA files was selected as the publishing source.
 - Update the PWA deployment
• Make sure that manifest.json and service-worker.js are present in the repository
 - Testing and verification
• After deployment, the PWA was tested in a browser by visiting github pages URL.
• The app was installed as a PWA on mobile device to confirm that it worked as expected.
- The app would now become live at [www.swayamraut8.github.com/snabit-pwa](https://swayamraut8.github.com/snabit-pwa)
- Conclusion: Deploying the snabit PWA to github pages successfully made the application publicly accessible. Users can now access and install the PWA directly from a web link without needing to clone the repository. This deployment ensures that

Swayam Raut
D15B Roll No. 48

MAD Experiment 10

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

The screenshot shows a GitHub repository named 'snapit_pwa'. The repository has 2 branches and 0 tags. The main branch contains several commits from user 'swayamraut8' made 3 weeks ago. The commits are:

- Made some changes by adding the icon
- Made some changes by adding the icon
- Add files via upload
- Made some changes by adding the icon
- Made some changes by adding the icon
- Made some changes by adding the icon

The repository has 2 commits and was last updated 3 weeks ago. It has 0 stars, 1 watching, and 0 forks. There are no releases published.

The screenshot shows the GitHub Pages settings for the 'snapit_pwa' repository. Under the 'General' tab, the 'Build and deployment' section is active. The 'Source' dropdown is set to 'Deploy from a branch'. The 'Branch' dropdown is set to 'main'. A red arrow points from the bottom left towards this 'main' branch selection. The 'Save' button is visible at the bottom right.

The screenshot shows the live GitHub Pages site for 'snapit_pwa'. The URL is https://swayamraut8.github.io/snapit_pwa/. The site was last deployed 3 minutes ago by user 'swayamraut8'. A red signature is written across the bottom of the page.



Vivekanand Education Society's Institute of Technology

(An Autonomous Institute Affiliated to University of Mumbai)

Department of Information Technology

A.Y. 24-25

Mobile App Development and Progressive Web App Lab

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	48
Name	Swayan Rath
Class	D15 B
Subject	MAD & PWA
Grade:	<i>S/ C</i>

MPL Experiment - 1

To use ~~google light~~ lighthouse tool to test the PWA functioning

What's google lighthouse?

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

Key features

Performance: It's an aggregation of how the page fares in aspects such as loading speed, time taken for loading for basic frames.

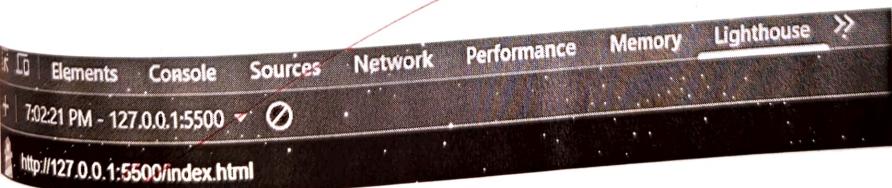
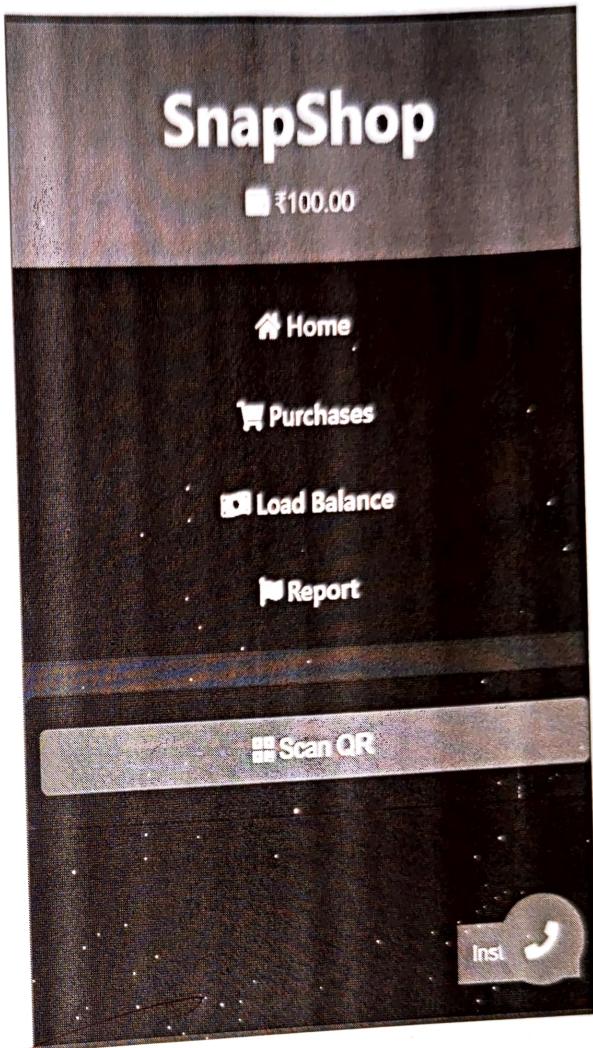
PWA Score: It's a score based on the baseline PWA checklist laid down by Google which includes Service Worker implementation.

Accessibility: It's a measure of how accessible a website is, across a plethora of accessibility features that can be implemented in a page.

Best Practices: It takes into account whether certain practices that are associated with PWAs are followed. The more the compliance, higher the score.

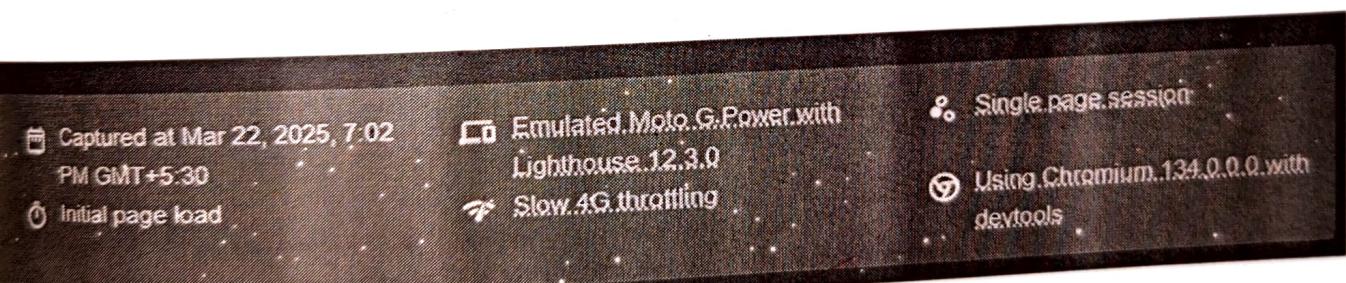
Conclusion: In this experiment, we learnt to use Google's Lighthouse to ~~check~~ judge a PWA's compliance with best responsive practices.

aim: Use Lighthouse chrome extension to analyse the PWA for performance.



DIAGNOSTICS

- Eliminate render-blocking resources — Potential savings of 970 ms
- Enable text compression — Potential savings of 18 KiB
- Page prevented back/forward cache restoration — 1 failure reason
- Minify JavaScript — Potential savings of 135 KiB
- Remove duplicate modules in JavaScript bundles — Potential savings of 16 KiB
- Reduce unused CSS — Potential savings of 21 KiB
- Reduce unused JavaScript — Potential savings of 121 KiB
- Minimize main-thread work — 2.8 s
- Avoid non-composited animations — 1 animated element found
- Initial server response time was short — Root document took 10 ms



Inclusion: Thus we successfully used google Lighthouse PWA Analysis Tool for testing the PWA on various factors such as performance, Accessibility, and SEO.

[Handwritten signature]