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

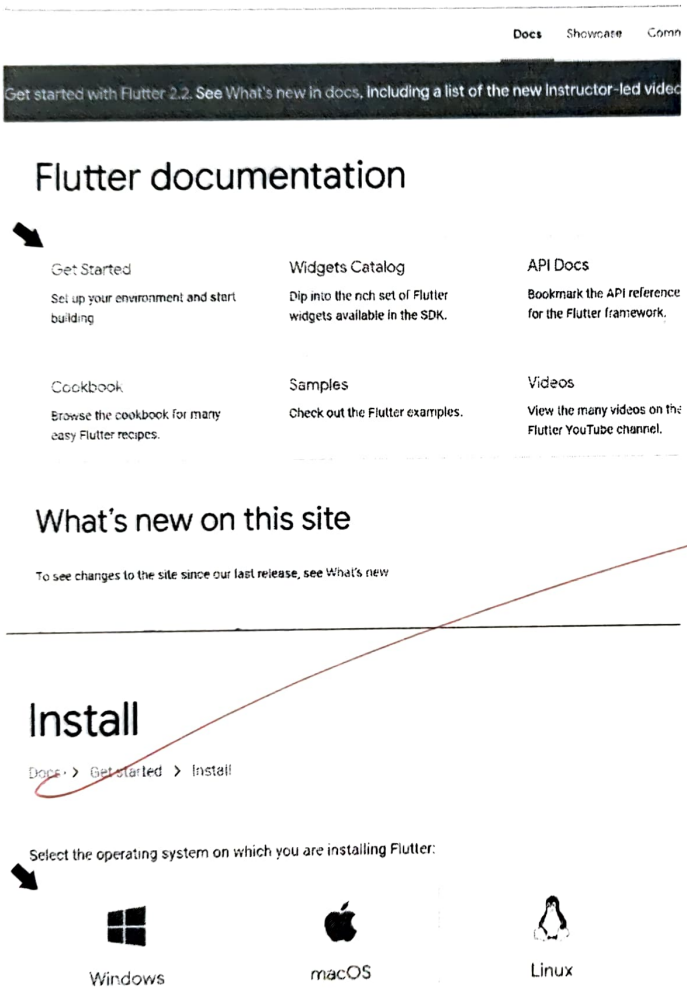
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment.
Roll No.	48
Name	Suayan Raut
Class	D15 B
Subject	MAD & PWA
Grade:	

Swayam Raut
D15B Roll No. 48

Experiment 1: Installation and Configuration of Flutter Environment.

Install the Flutter SDK

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



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.

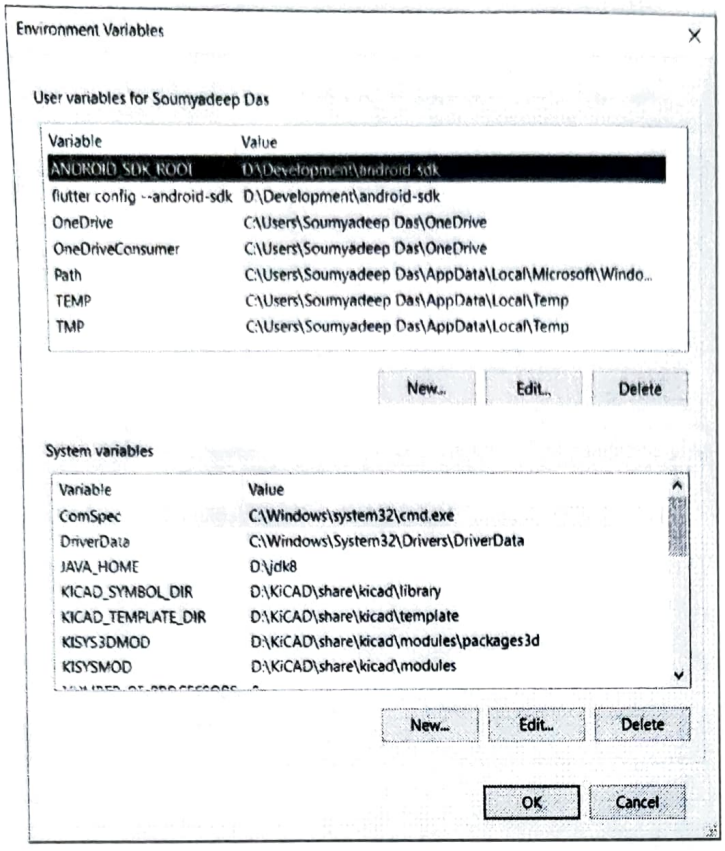
If you don't want to install a fixed version of the installation bundle, you can skip steps 1 and 2. Instead, get the source code fr the Flutter repo on GitHub, and change branches or tags as needed. For example:

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

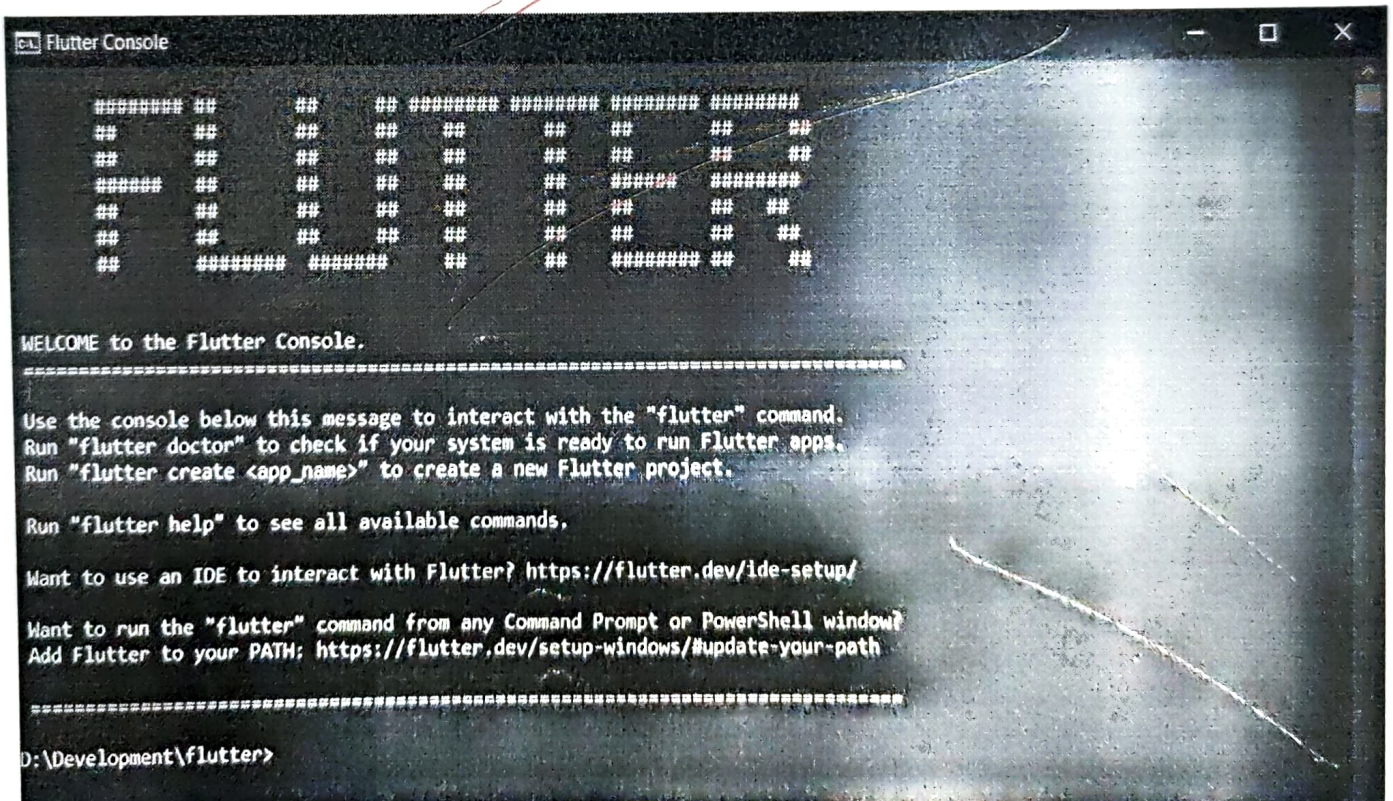
You are now ready to run Flutter commands in the Flutter Console.

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

Swayam Raut
D15B Roll No. 48

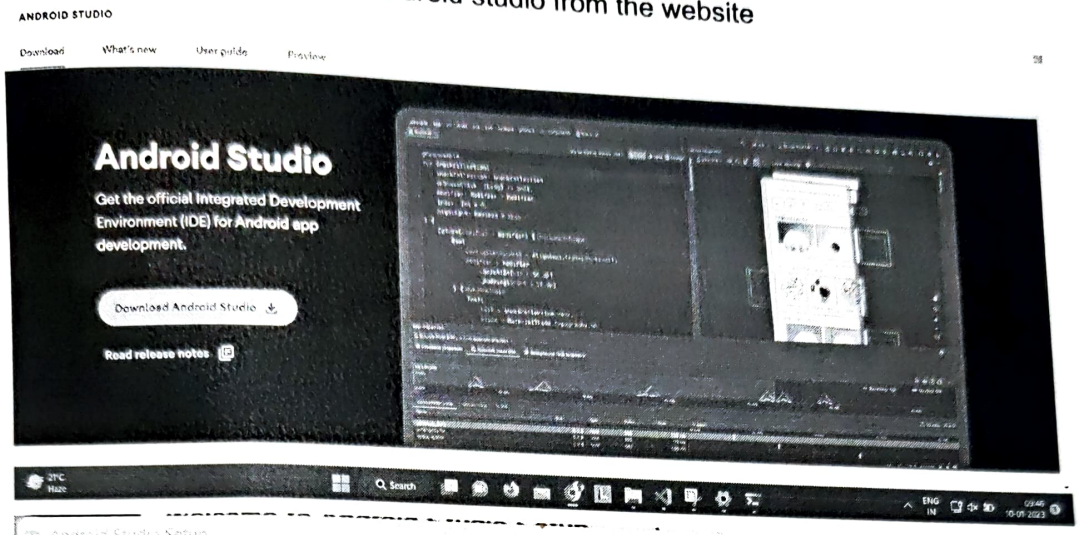


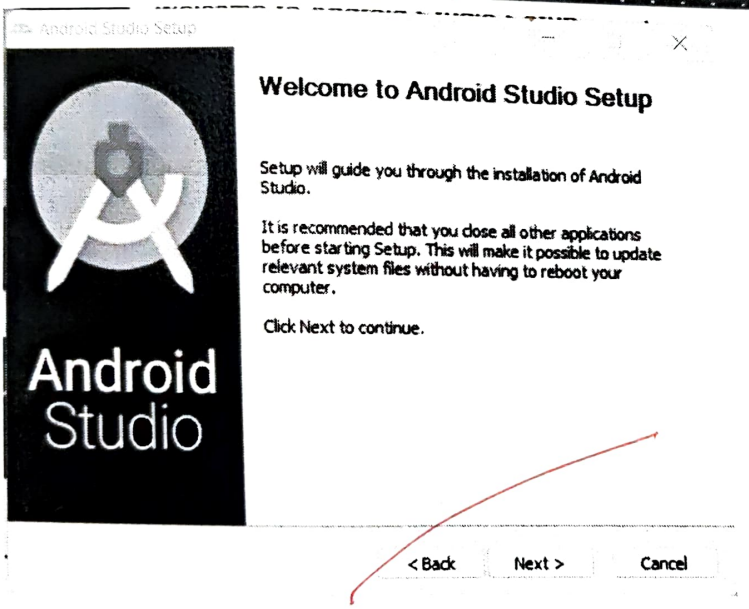
Step 4: Run the flutter doctor command

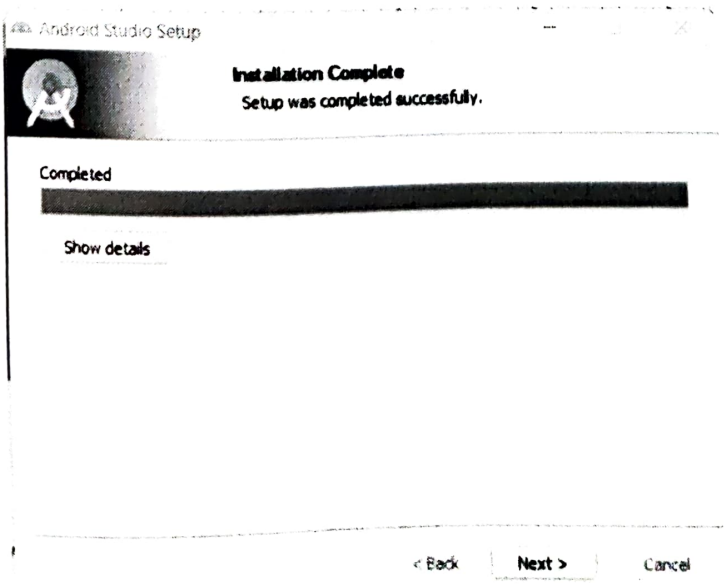


Swayam Raut
D15B Roll No. 48

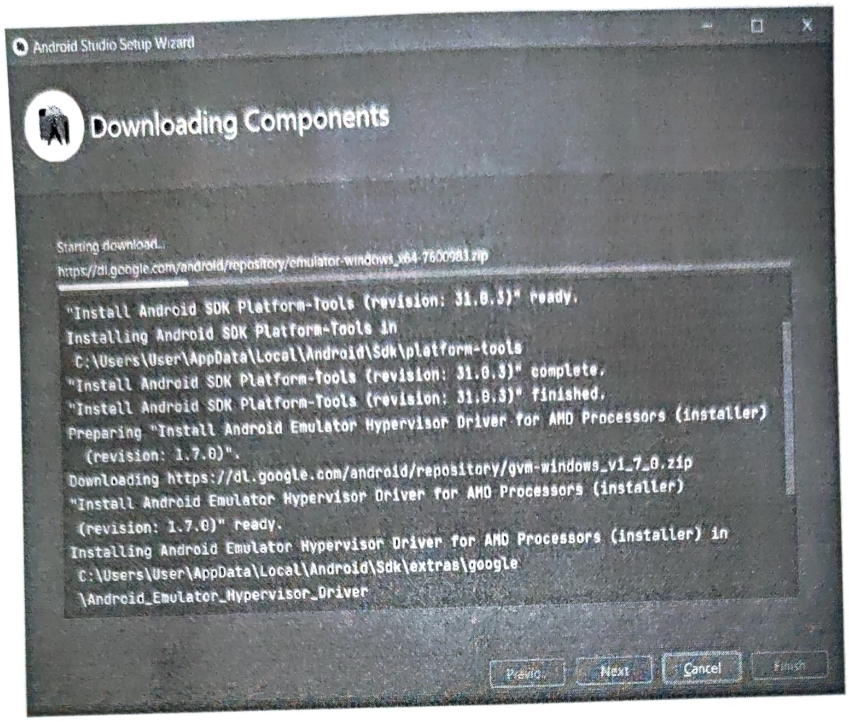
Step 6: Download and install android studio from the website



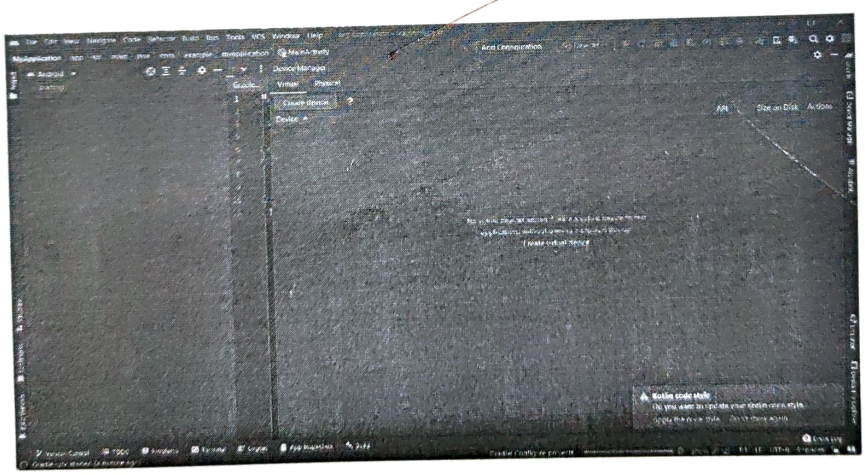




Swayam Raut
D15B Roll No. 48

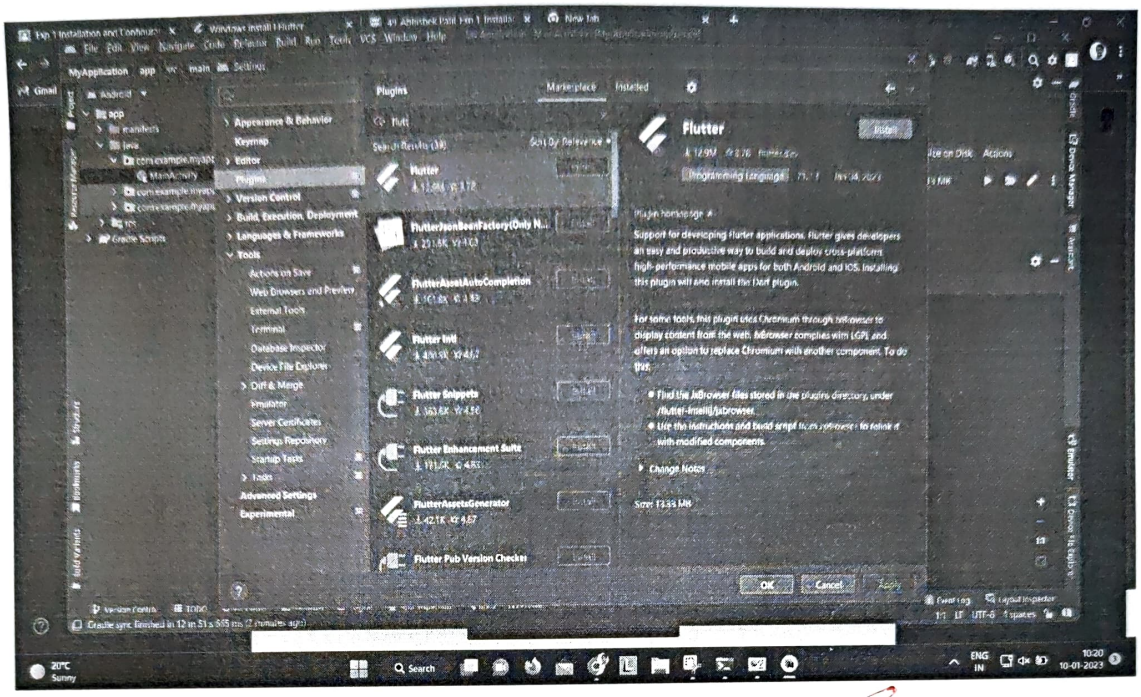


Step 8: Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.



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.

ayam Raut
B
N0.48

MPL Lab

Experiment - II

A+

m: Basic Flutter UI 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 interactivity
- checked form responsiveness.

Results :-

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

Swayam Raut
D15 B Roll No. 48 MPL Experiment 2

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

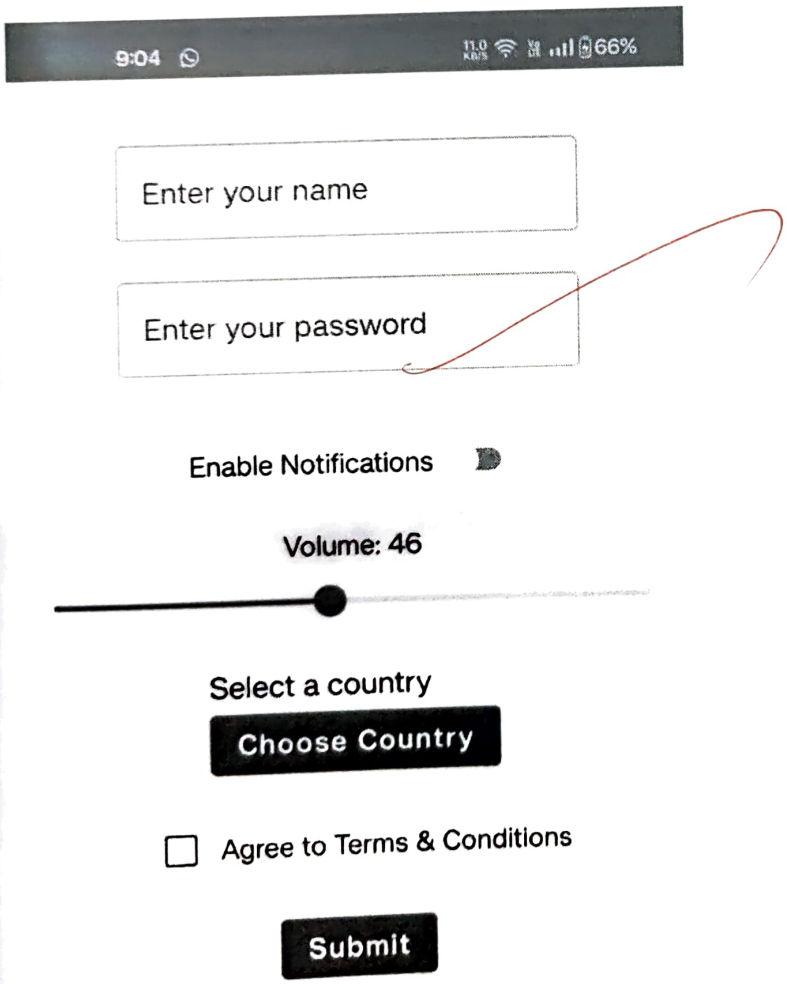
main.dart (File that contains the UI Layout)

```
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(
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              // Image
              Image.asset(
```

```
        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



Swayam Raut
D15 B Roll No. 48 MPL Experiment 2

The following custom components have been used:

1) Dropdown
2) Checkbox
3) Notification slider

Conclusion: In this lab experiment, we acquainted ourselves wi 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	Swayam Raut
Class	D15 B
Subject	MAD & PWA
Grade:	A1

Yam Rault

NO. 48

Experiment - III

A+

Enhanced flutter UI with Icons, Images, and custom fonts

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

Procedure

1. Setup & config
   - Add google fonts dependencies 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 icon rendering and re-alignment.
Tested overall UI consistency.

Results:-
Material icons integrated through the interface
Custom font called Hack applied consistently
Improved visual hierarchy and UX.
Professional looking interface element.

Page No.
Date

Conclusions:

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

Swayam Raut
D15B

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

main.dart

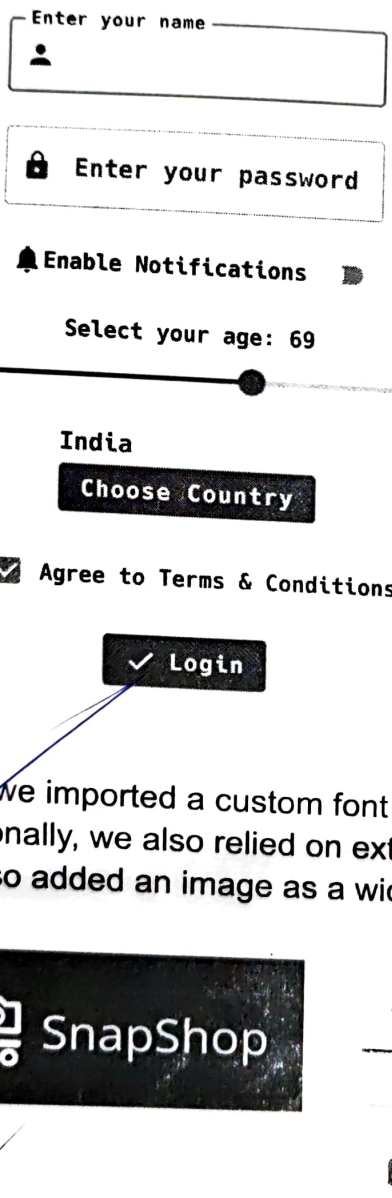
```
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';

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) {
```

Swayam Raut
D15B

MPL Experiment 3

Output:



Here, we imported a custom font called hack.tff
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:	

m Raut
Roll No. 48

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 implement through authentication on firebase.

Key widgets used:
TextField, ElevatedButton, TextButton, and Image

Key features implemented

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

Image: Displays the image of the logo

StateManagmt: The _isRegister boolean variable whether the form displays login or registration. SetState is used to build UI.

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

Teacher's Sign.:

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

Already have an account? Login

Login

Email

Password

Login

No account? Sign up

Registration Page

Login Page

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

Swayam Raut
D15 B Roll No. 48

MAD Experiment 4

Firebase User Data

napshop ▾ Authentication

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
swayam@ymail.com	✉	24 Feb 2025	24 Feb 2025	67uK8MbYhlbw90i7y13zq1Ol...
swayamraut252@gmail...	✉	24 Feb 2025	24 Feb 2025	HWkUW3qG9iMZobZcz8f9Ff4...
swayamraut2@gmail.c...	✉	24 Feb 2025	24 Feb 2025	OH4PlqzMOSYlKnqk0z5mSf5...
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	SFvgWJHqL8ckryaEYV13IzW...

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	Swayam Raut
Class	D15 B
Subject	MAD & PWA
Grade:	A+

pam Raut
5 D
No. 48 MAD Experiment - V

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

Theory:
In this ~~Lab~~ Lab, ~~base~~ 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 with

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

1) Named Routes: Defined named routes in Material App's routes property. This makes navigation cleaner.
2) Navigation and named Routes: Used Navigator.pushReplacementNamed ensures
3) Gesture Detection: Wrapped the ScanProductScreen with a GestureDetector to detect taps & double taps. You can add other
4) Logout Navigation: In MainScreen, the logout 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, routing using named routes, and gesture detection. You can expand on this by adding more complex navigation logic, passing data between screens using routes, and implementing more sophisticated gesture interactions.

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
swayam@ymail.com

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	Swayam Raut
Class	D15 B
Subject	MAD & PWA
Grade:	A+

am Raut
Roll No. 48

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()
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: TextField, ElevatedButton, TextButton for login/registration forms.

Necessary Considerations:
- Firebase console setup
- Error handling
- Correct Platform configuration files

Conclusion:
This experiment demonstrated firebase integration in flutter. Proper setups and error handling are crucial. This allows for simple authentication and data storage across platforms

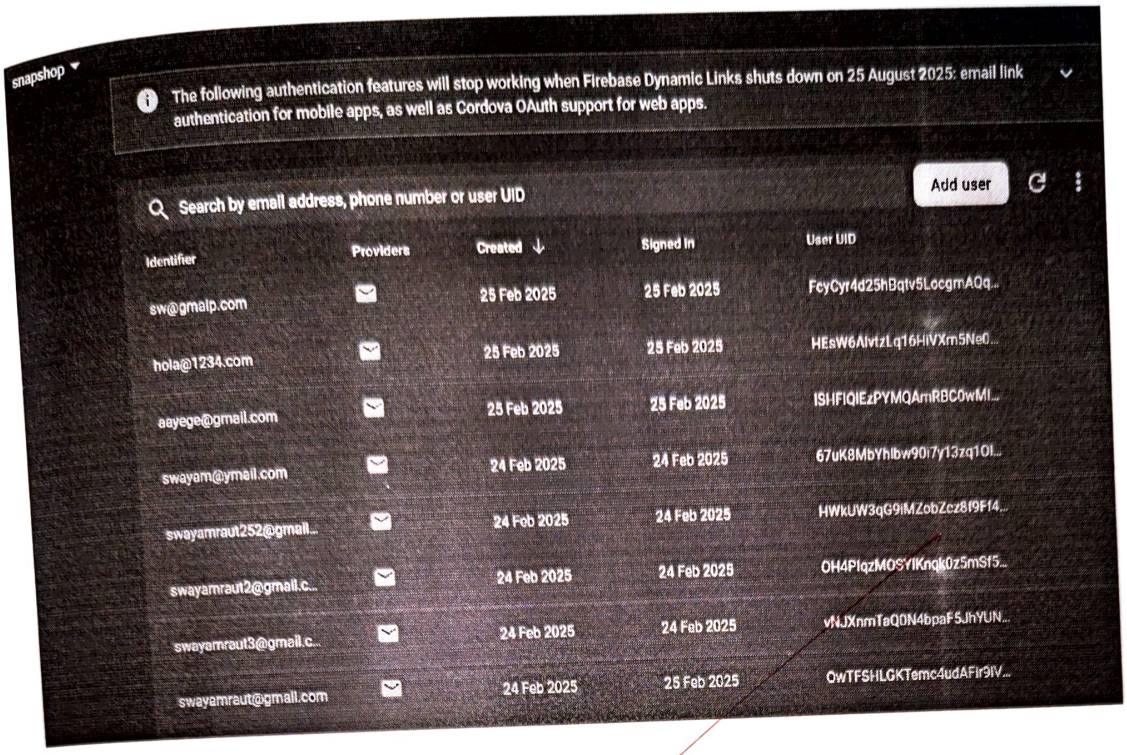
Teacher's Sign.:

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';
```



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.	42
Name	Suvyam Patil
Class	D15 B
Subject	MAD & PWA
Grade:	AH

MPL
~~MAD~~ Experiment - VII (AH)

Sayam Raut
D15B
Roll No. 48

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

Progressive Web Apps
PWA's are built with web technologies (HTML, CSS, JS) that offer a 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 browsers.

In this experiment, to set up a basic ~~web~~ 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.

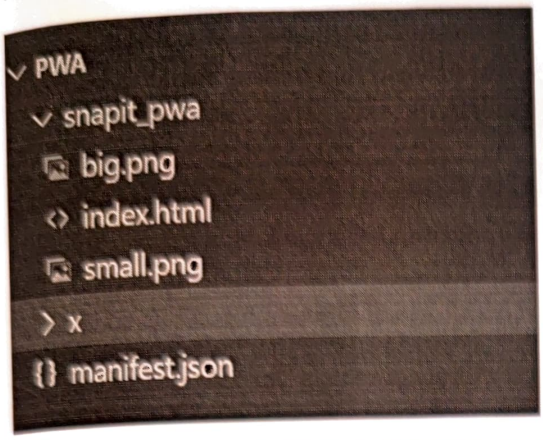
Conclusion
In this experiment we learnt to set up a PWA by define manifest.json file. This PWA could be installed on visiting its address through a browser on a mobile phone.

Teacher's Sign.: _____

Swayam Raut
D15B Roll No. 48

MAD Experiment 7

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



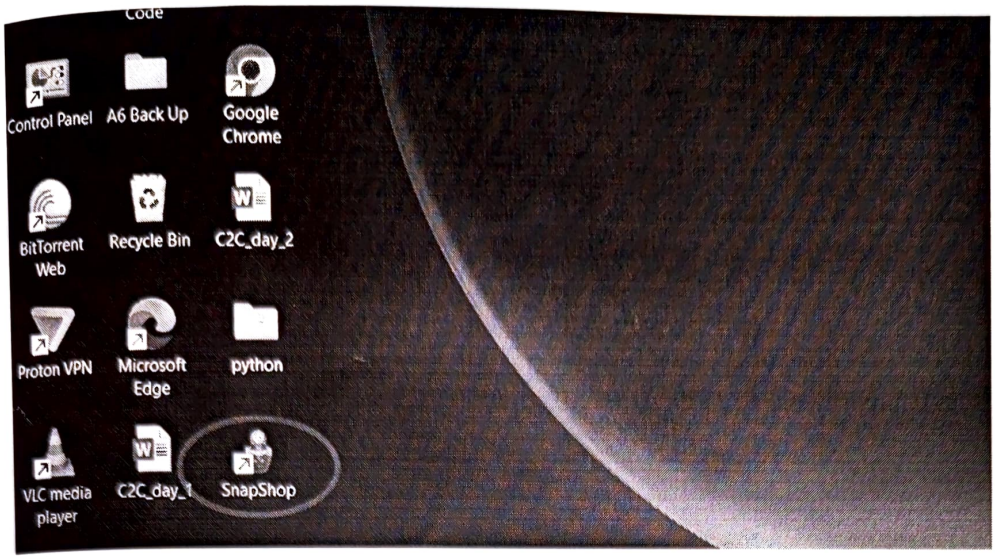
index.html manifest.json

manifest.json > [ ] icons

```
{
    "name": "SnapShop",
    "short_name": "SnapShop",
    "description": "A QR-based shopping app",
    "start_url": "index.html",
    "display": "standalone",
    "background_color": "#f0f2f5",
    "theme_color": "#0ea655",
    "icons": [
      {
        "src": "small.png",
        "sizes": "192x192",
        "type": "image/png"
      },
      {
        "src": "big.png",
        "sizes": "512x512",
        "type": "image/png"
      }
    ]
}
```

Swayam Raut
D15B Roll No. 48

MAD Experiment 7



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.

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	Swayam Rana
Class	D15 B
Subject	MAD & PWA
Grade:	AH

Yam Raut
B Roll No. 48

MPL Experiment - 8

Aim: 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 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.

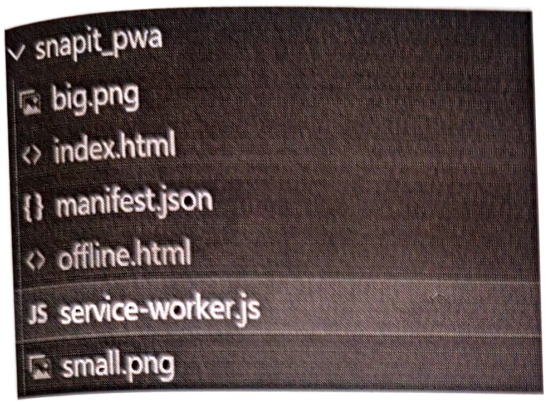
Conclusion: By implementing a service worker with caching, we successfully implemented offline functionality and improved performance for the web app.

Swayam Raut
D15B Roll No. 48

MAD Experiment 8

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

Create service-worker.js



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

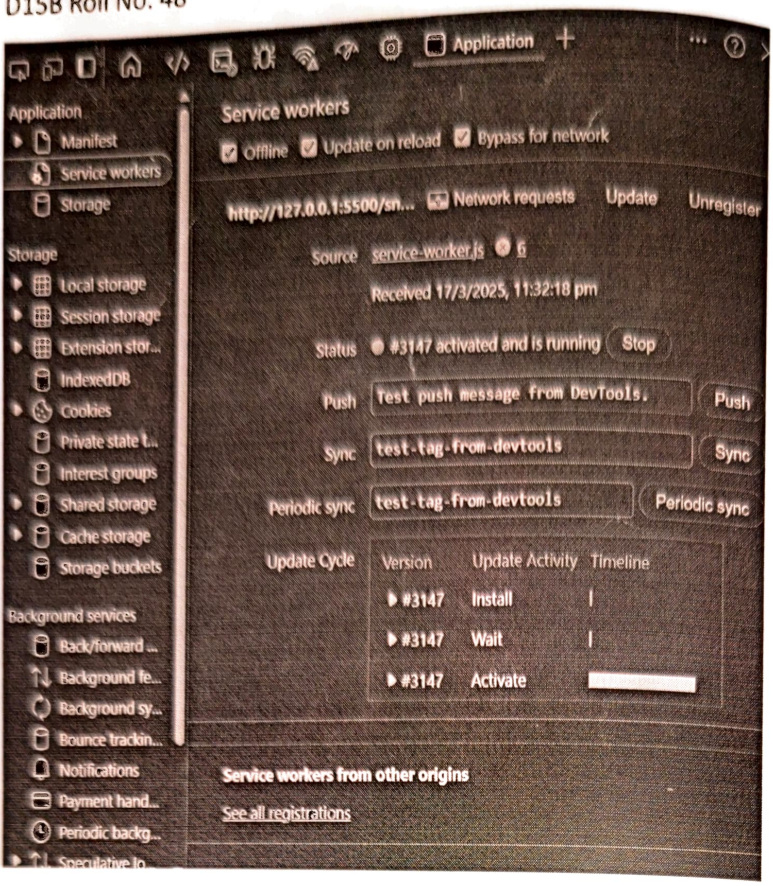
offline.html U

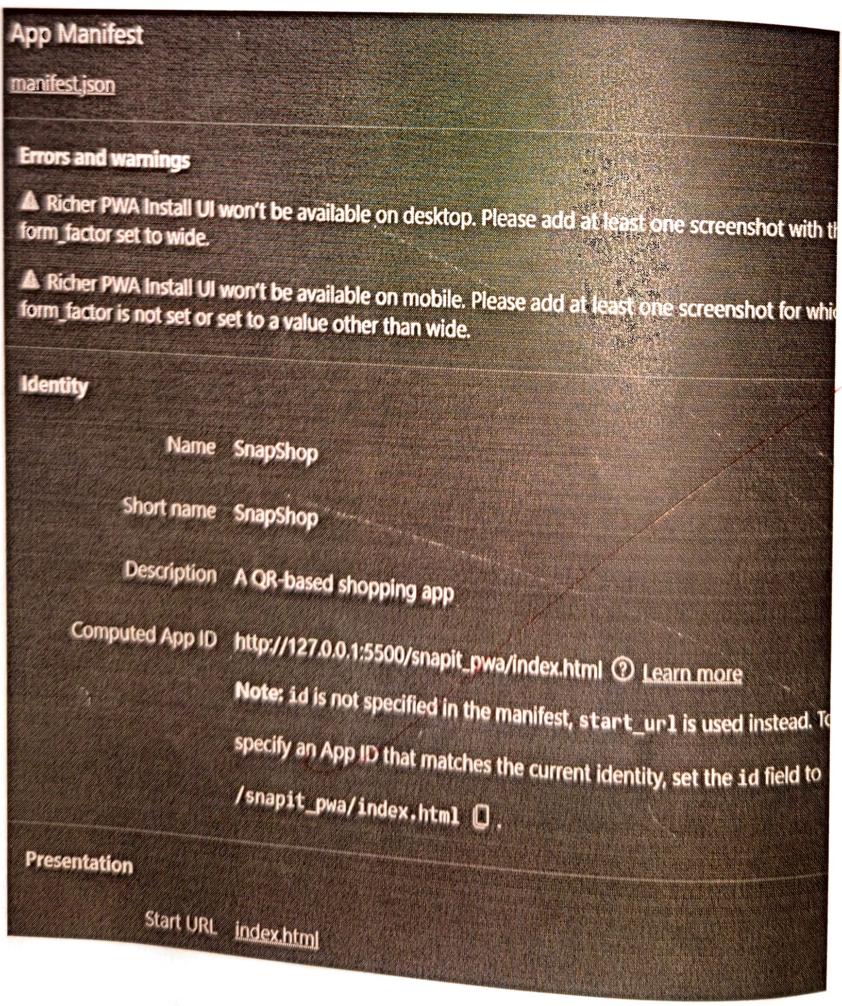
snapit_pwa > offline.html > ...

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Offline</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            padding: 50px;
        }
    </style>
</head>
<body>
    <h1>You're Offline</h1>
    <p>It looks like you have lost internet access. Some features may not work.</p>
    <img src="big.png" alt="SnapIt Logo" width="100">
</body>
</html>
```

Swayam Raut
D15B Roll No. 48

MAD Experiment 8





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 Raut
Class	D15 B
Subject	MAD & PWA
Grade:	AH

Roll No. 48

MPL Experiment - 9

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:

1) 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 retrive fresh content from the internet.
- For static resources like images and HTML files, the cache-first strategy being used.

2) Sync Even (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 users to continue using application in offline mode.

3) Push even (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.

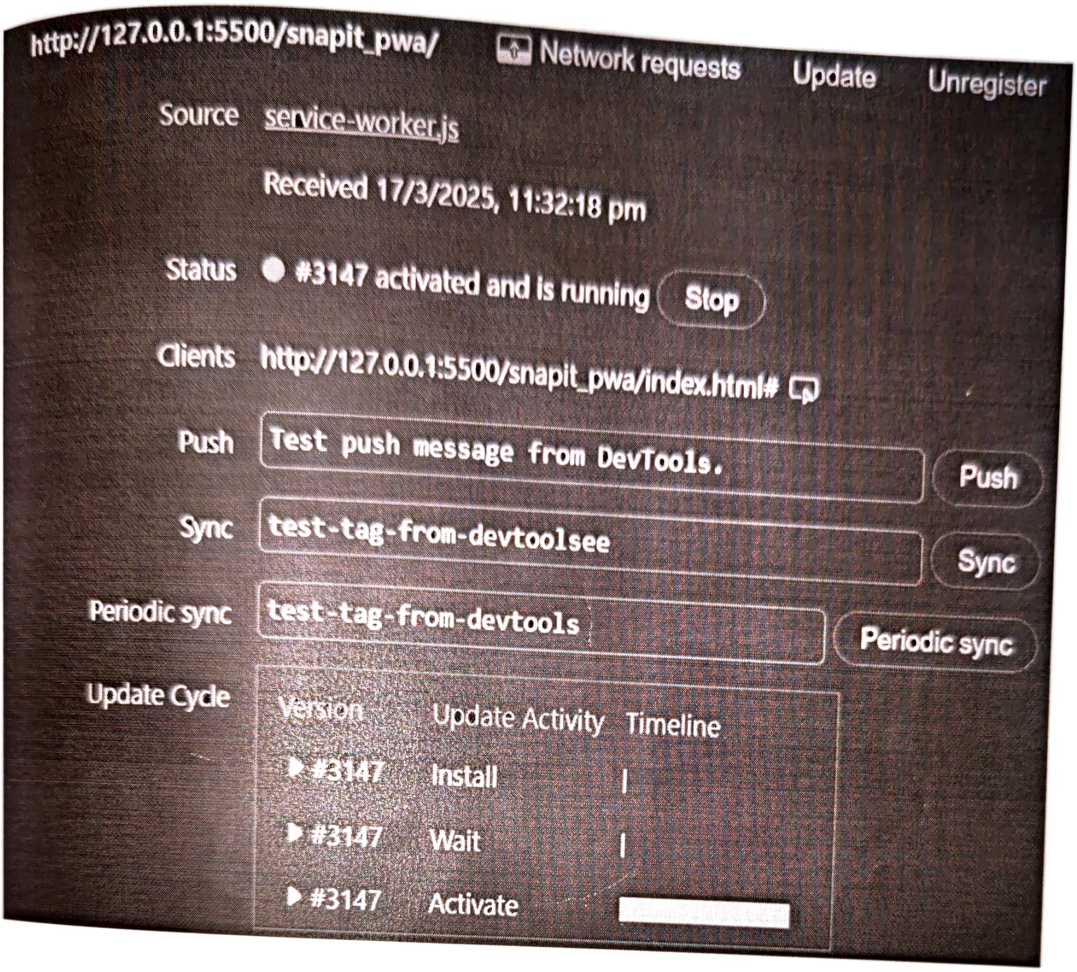
Conclusion:
We implemented advanced service worker events,

Teacher's Sign.: _____

Swayam Raut
D15B Roll No. 48

MAD Experiment 9

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



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.	48
Name	Swayam Rath
Class	D15 B
Subject	MAD & PWA
Grade:	

Swayam Raut
B
548

MPL Experiment - 10

Aim: To deploy the PWA on github pages, making it accessible online with live URL.

..

Deployment Process and configuration

1) Preparing the repositiory
- Add all the files to a github repository

2) Enabling 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.

3) Update the PWA deployment
- Make sure that manifest.json and and service worker.js are present in the repository

4) 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.

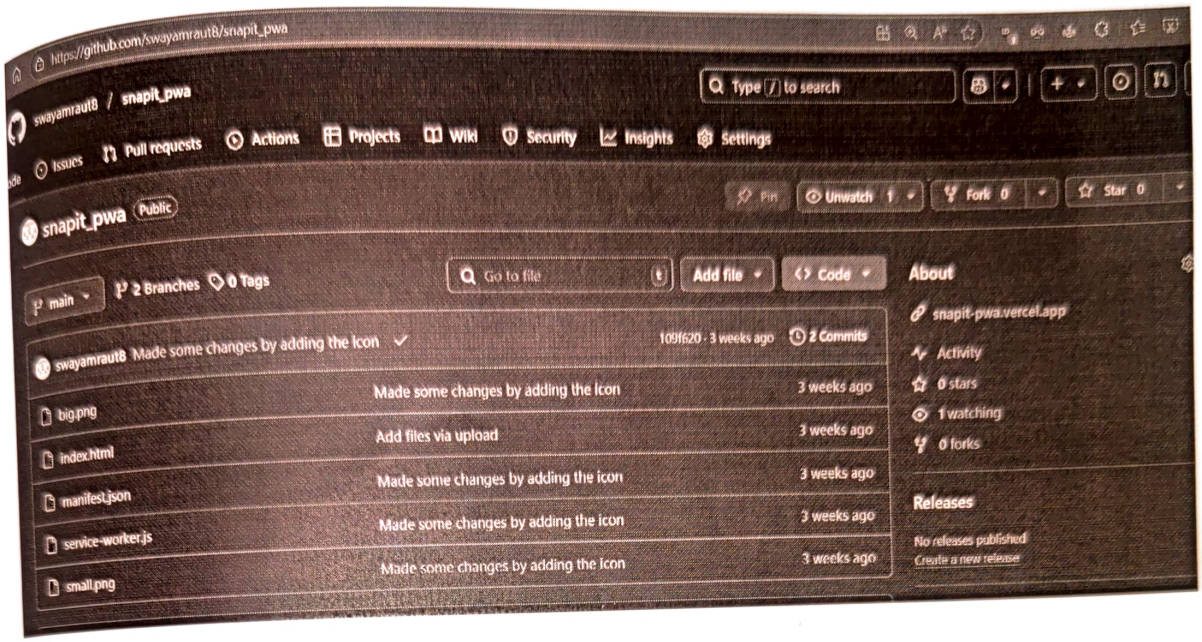
The url would now become live at
www.swayamraut8.github.com/snapit-pwa

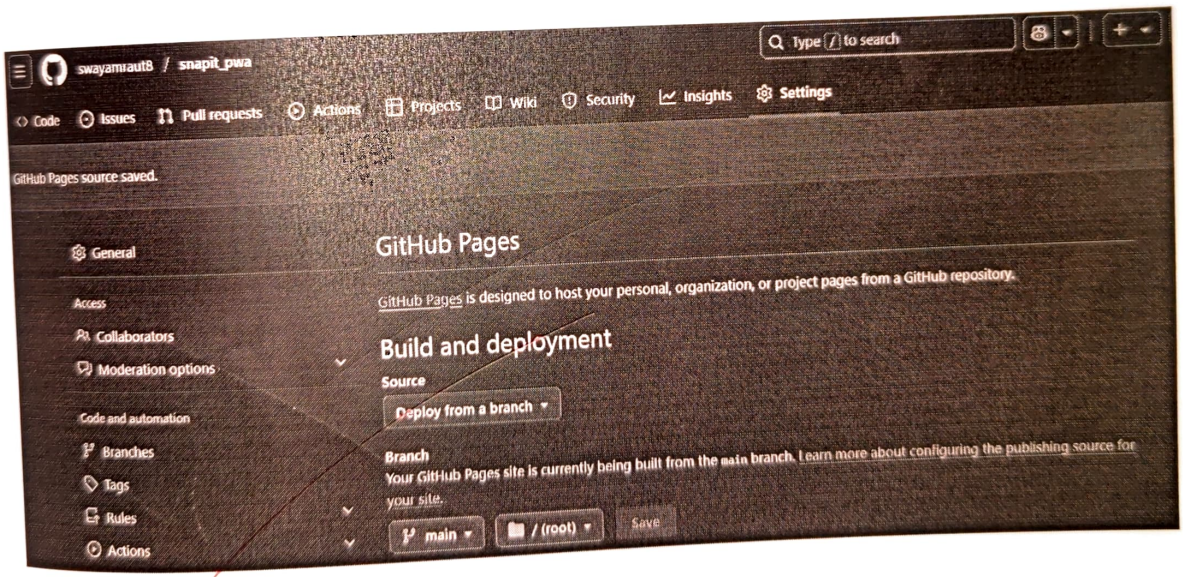
Conclusion: Deploying the snapit PWA to github pages successfully made the application publicly accessible. Users can now access and install the PWA directly from a web link without needing to clone the repository. This deployment ensures that

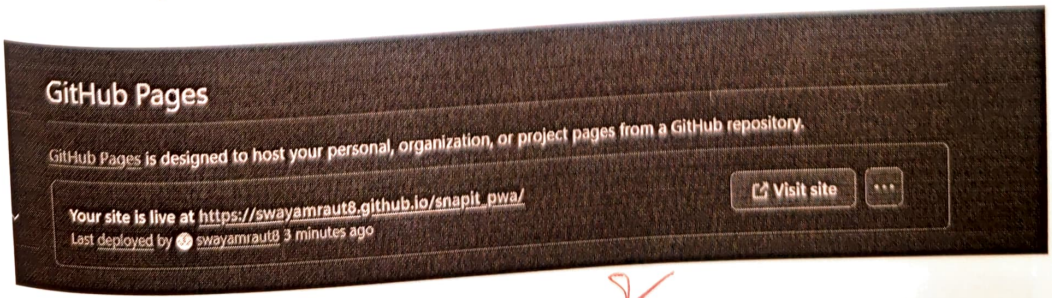
Swayam Raut
D15B Roll No. 48

MAD Experiment 10

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







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	Sravan Rao
Class	D15 B
Subject	MAD & PWA
Grade:	A+

MPL Experiment - (1)

Page No.
Date

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

What's google lighthouse ?

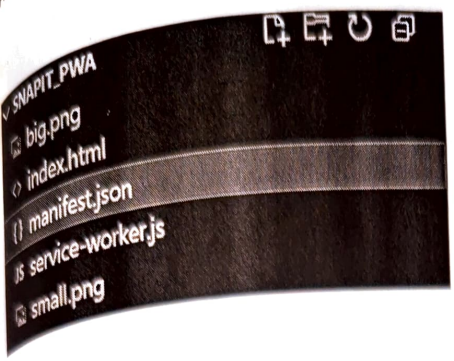
It's a tool that lets you audit your web application based on a number of parameters including performance, based on a number of metrics, mobile compatibility. Progressive web App (PWA) implementation, etc. All you have to do is run it on a page or pass it a URL, sit back for a couple of minutes and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

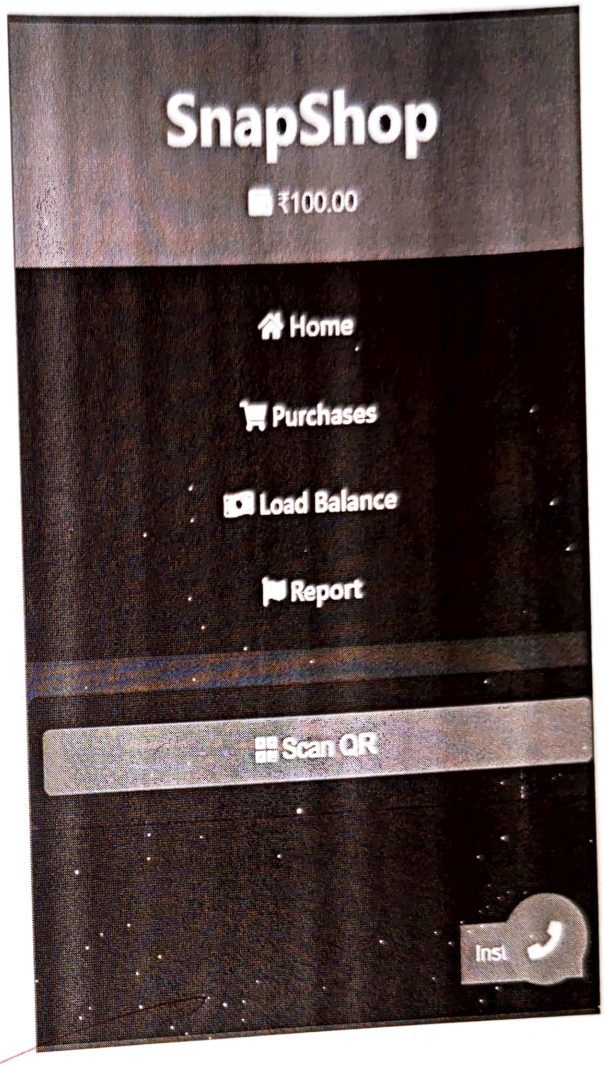
Key features

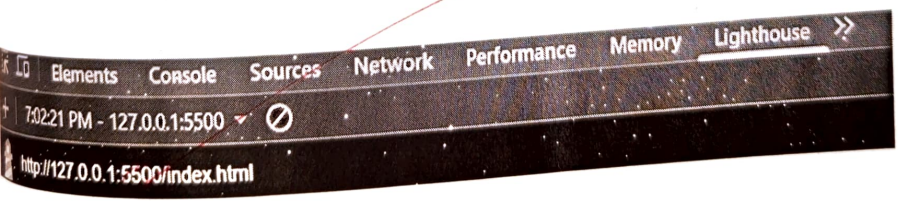
1) Performance: It's an aggregation of how the page fared in aspects such as loading speed, time taken for loading for basic frames
2) PWA score: It's a score based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation
3) Accessibility: It's a measure of how accessible a website is across a plethora of accessibility features that can be implemented in a page.
4) Best Practices: It takes into account whether certain practices that are associated with PWAs are followed. Greater the compliance, higher the score.

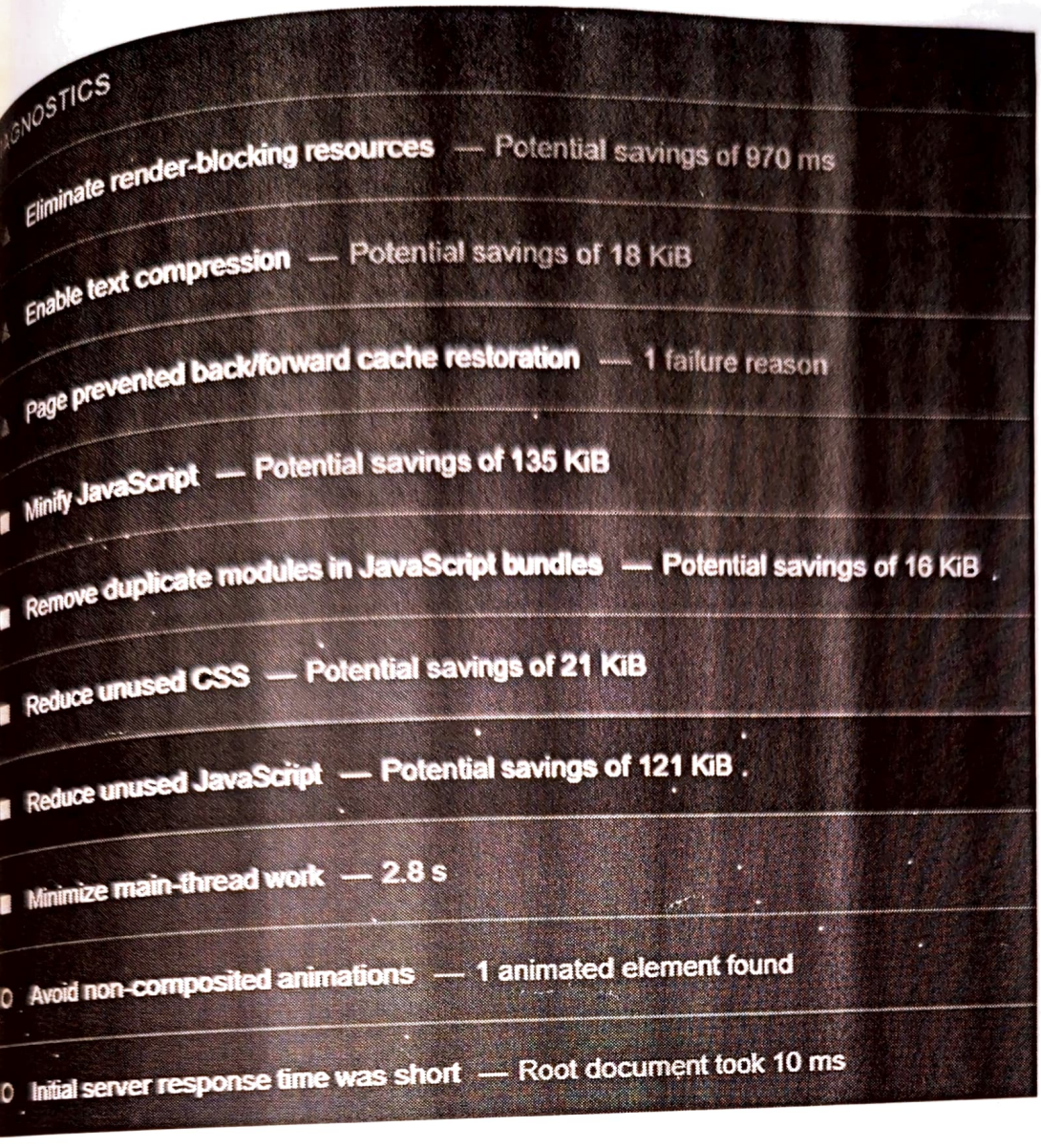
Conclusion: In this experiment, we learnt to use Google's Lighthouse to judge a PWA's compliance with best responsive practices.

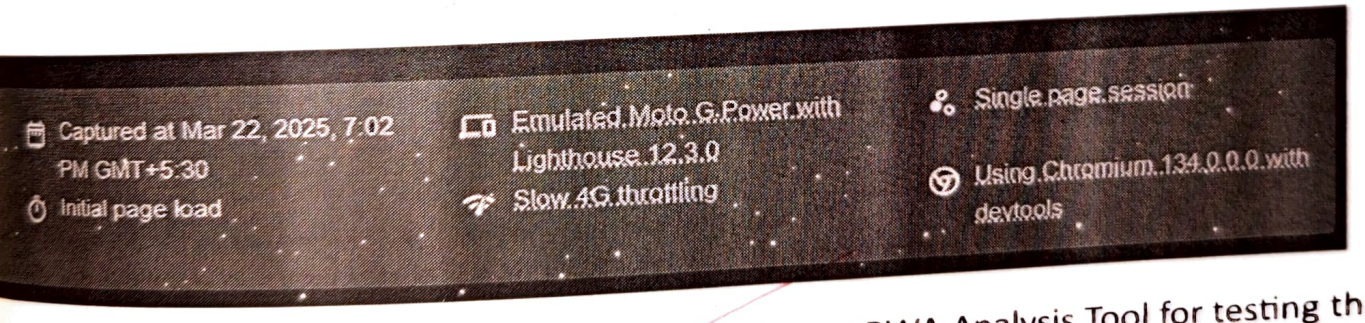
...: Use Lighthouse chrome extension to analyse the PWA for performance.











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