Vivekanand Education Society's Institute of Technology

An Autonomous Institute Affiliated to University of Mumbai

Hashu Advani Memorial Complex, Collector Colony, Chembur East, Mumbai - 400074.

Department of Information Technology

CERTIFICATE

This is to certify that Swaraj Patil of **D15A/D15B** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2024-2025.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Name of the Course : MAD & PWA Lab

Course Code : ITL604

Project Title: **Roll No.**

Year/Sem/Class **:** D15A/D15B **A.Y.:** 24-25

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers **:** Mrs. Kajal Joseph.

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.

Project Title: **Roll No.**

Lab Objectives:

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

Lab Outcomes:

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

Project Title: **Roll No.**

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
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			

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

Project Title: **Roll No.**

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Project Title: **Roll No.**

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.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

Project Title: **Roll No.**

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.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Project Title: **Roll No.**

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.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Project Title: **Roll No.**

MAD & PWA Lab
Journal

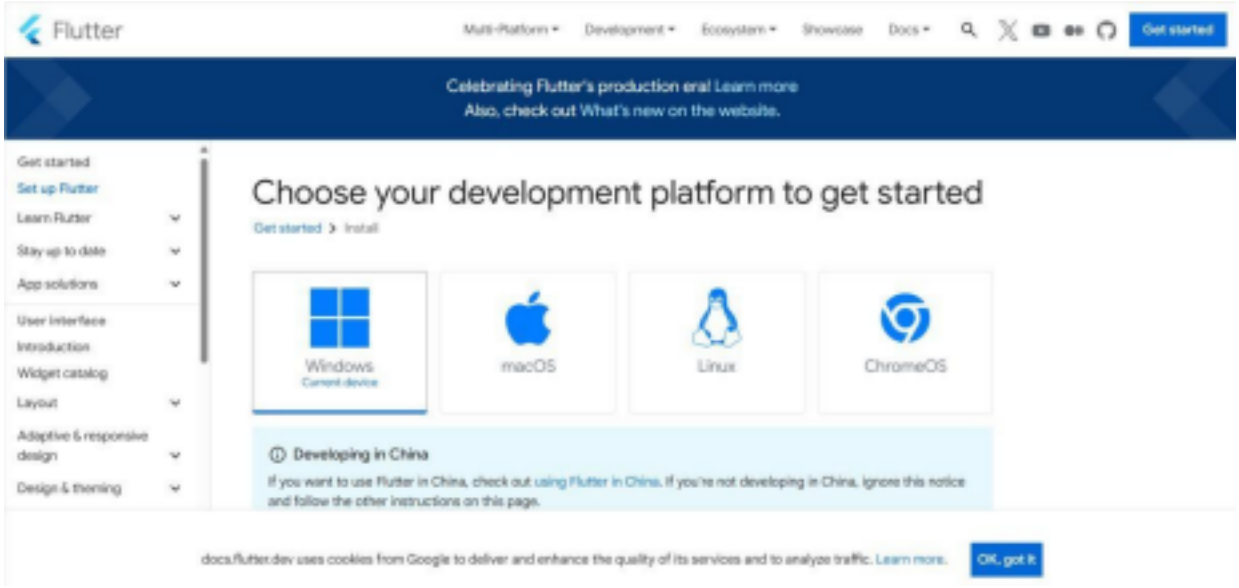
Experiment No.	Assignment-2
Assignment 2 Questions	1. Define Progressive Web App (PWA) and explain its significance in modern web development.Discuss the key characteristics that differentiate PWAs from traditional mobile apps 2. Define responsive web design and explain its importance in the context of Progressive Web Apps.Compare and contrast responsive, fluid, and adaptive web design approaches. 3. Describe the lifecycle of Service Workers, including registration, installation, and activation phases. 4. Explain the use of IndexedDB in the Service Worker for data storage.
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	

EXPERIMENT NO: - 01

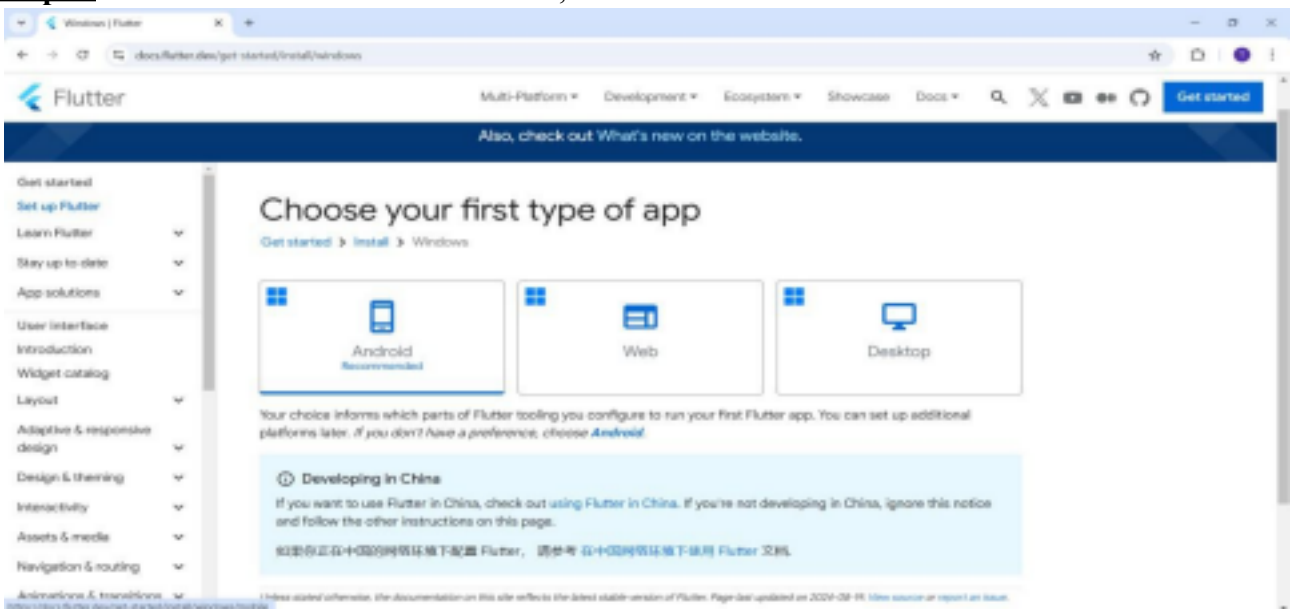
Name:- Swaraj Patil **Class:-** D15A **Roll:No: -** 35 **AIM: -** Installation and Configuration of

Flutter Environment.

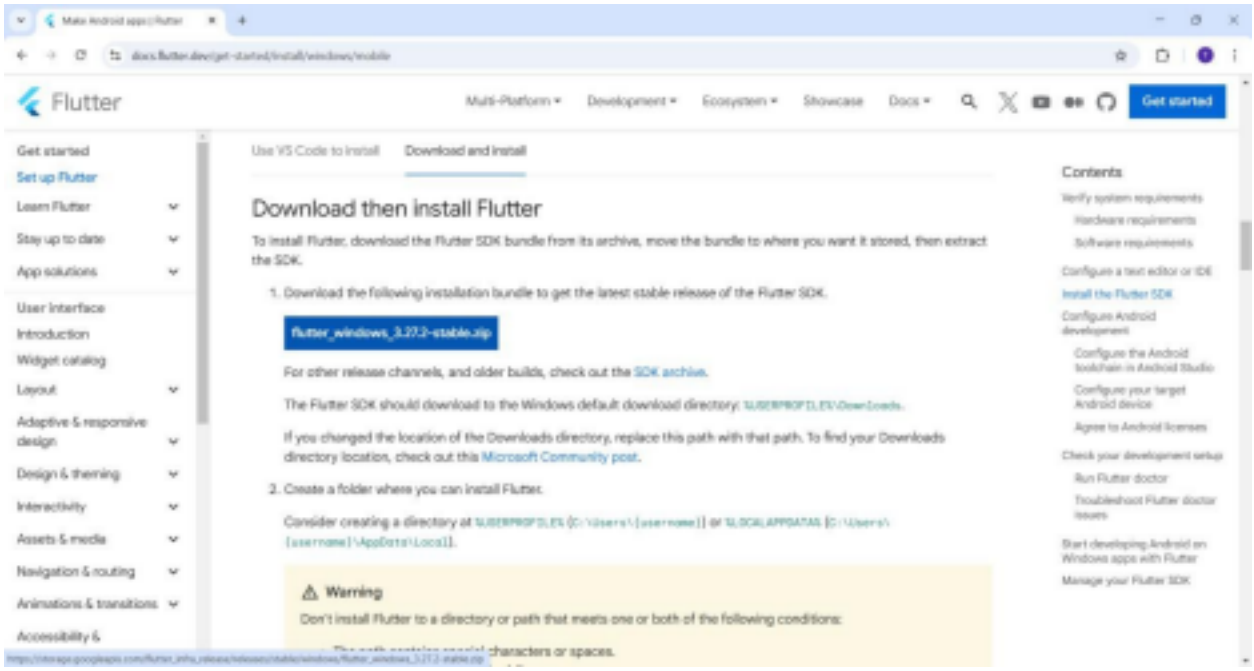
Step 1: Go to the official Flutter website: https://docs.flutter.dev/get-started/install



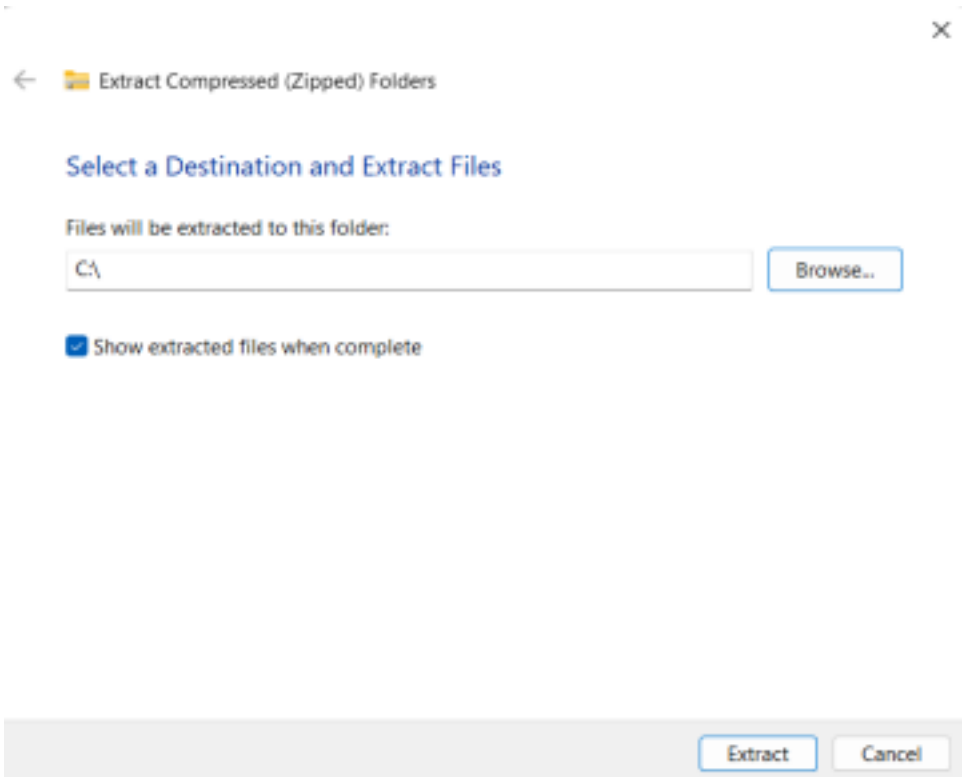
Step 2: To download the latest Flutter SDK, click on the Windows icon > Android



Step 3: For Windows, download the stable release (a .zip file).



Step 4: Extract the ZIP file to a folder (e.g., C:\flutter).

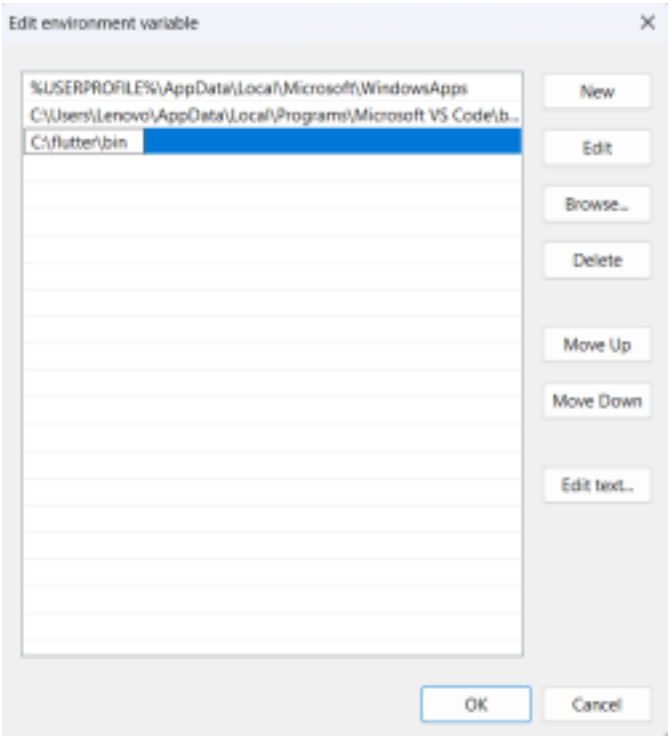


Step 5 :- Add Flutter to System PATH

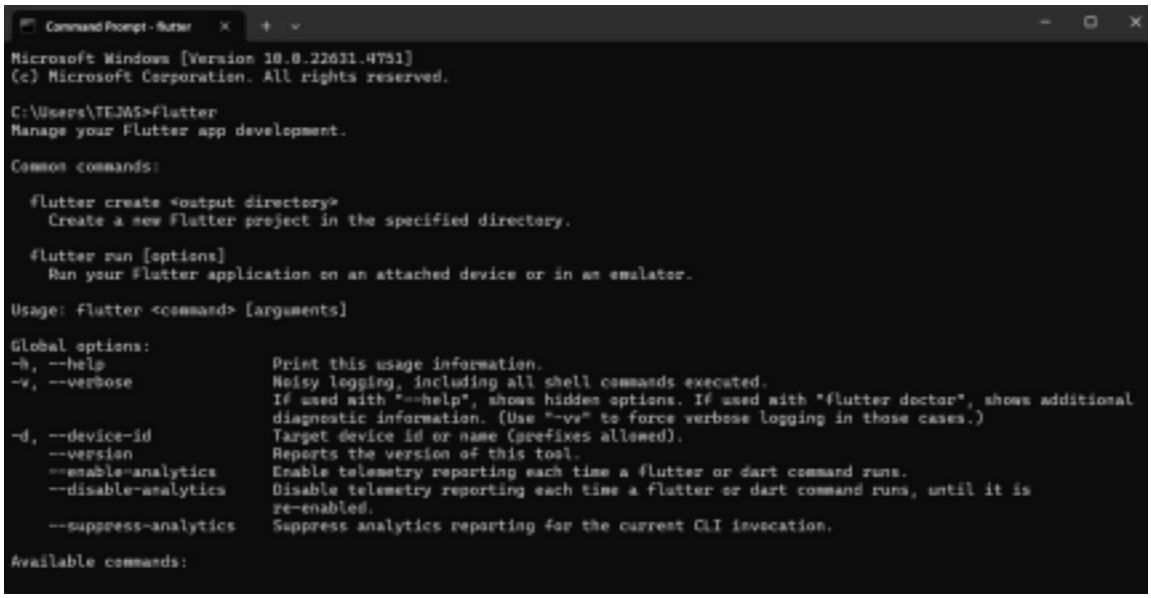
Right-click on the Start Menu > System > Advanced system settings > Environment Variables.

Under System Variables, find Path and click Edit.

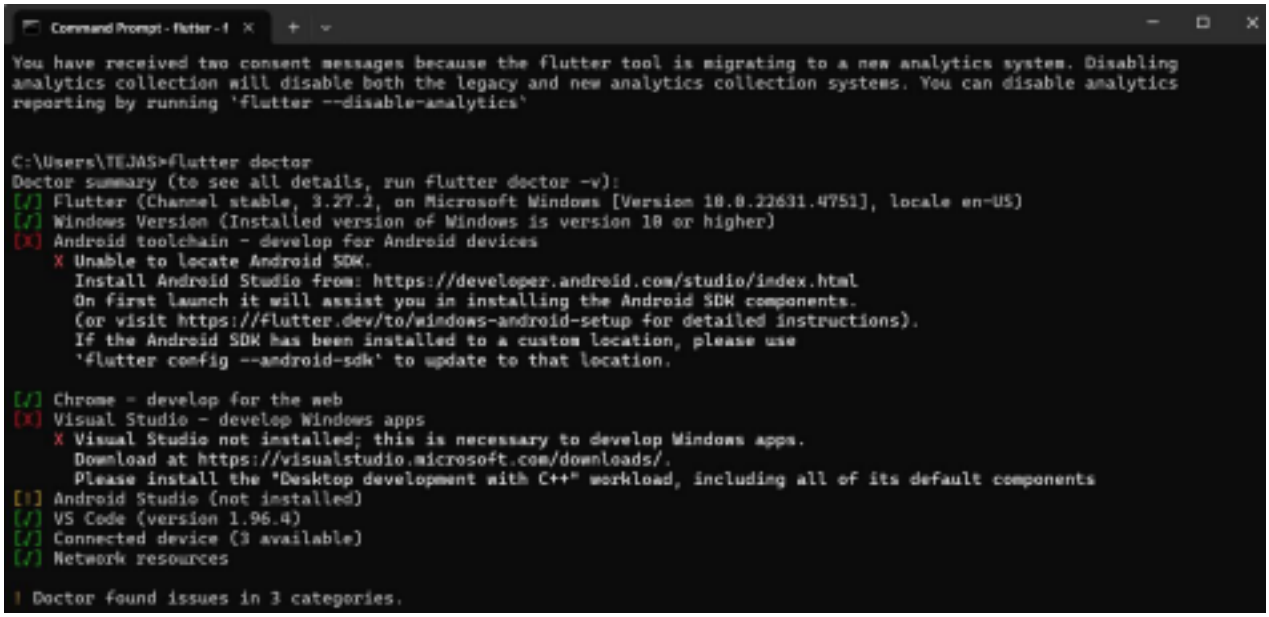
Add the full path to the flutter/bin directory (e.g., C:\flutter\bin).



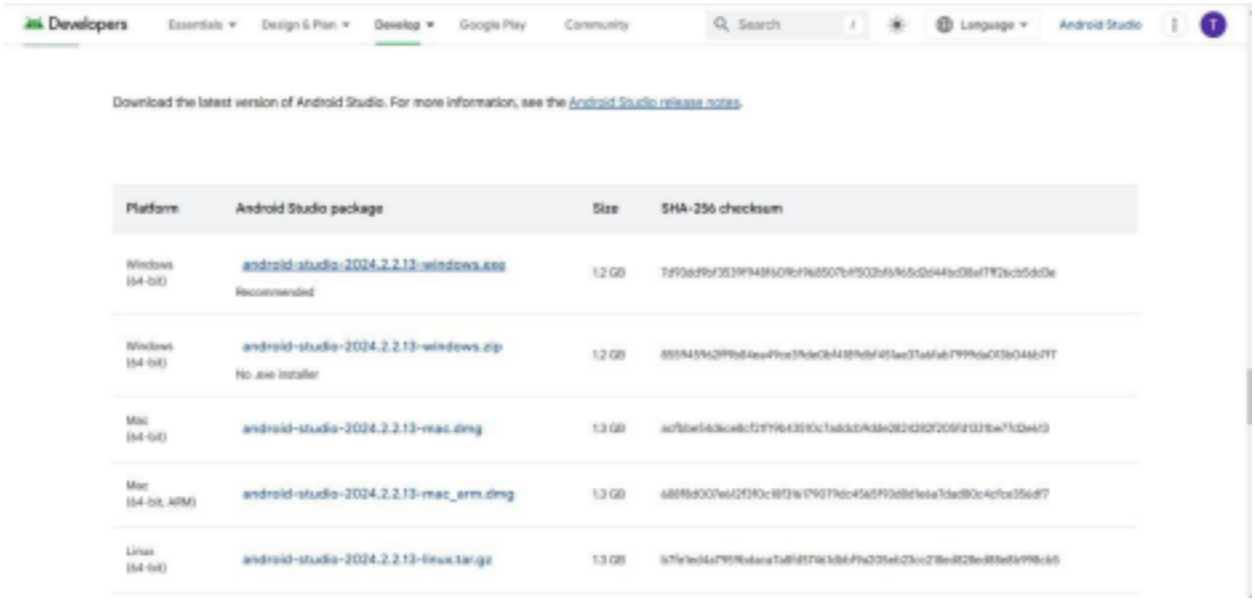
Step 6 : - Now, run the $ flutter command in command prompt.



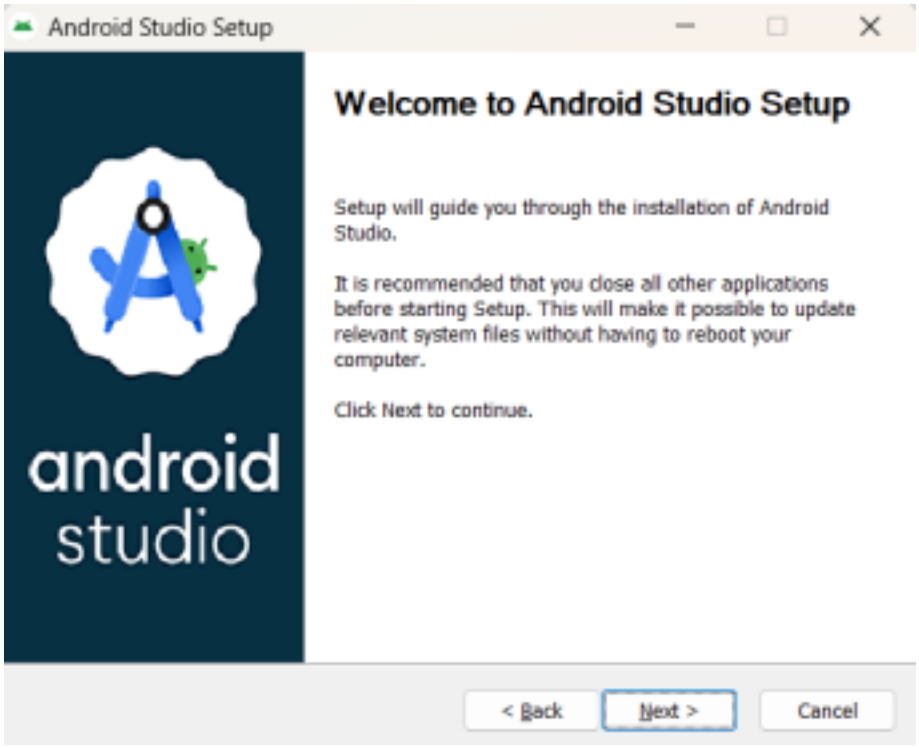
Step 7:- Run the $ flutter doctor command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation



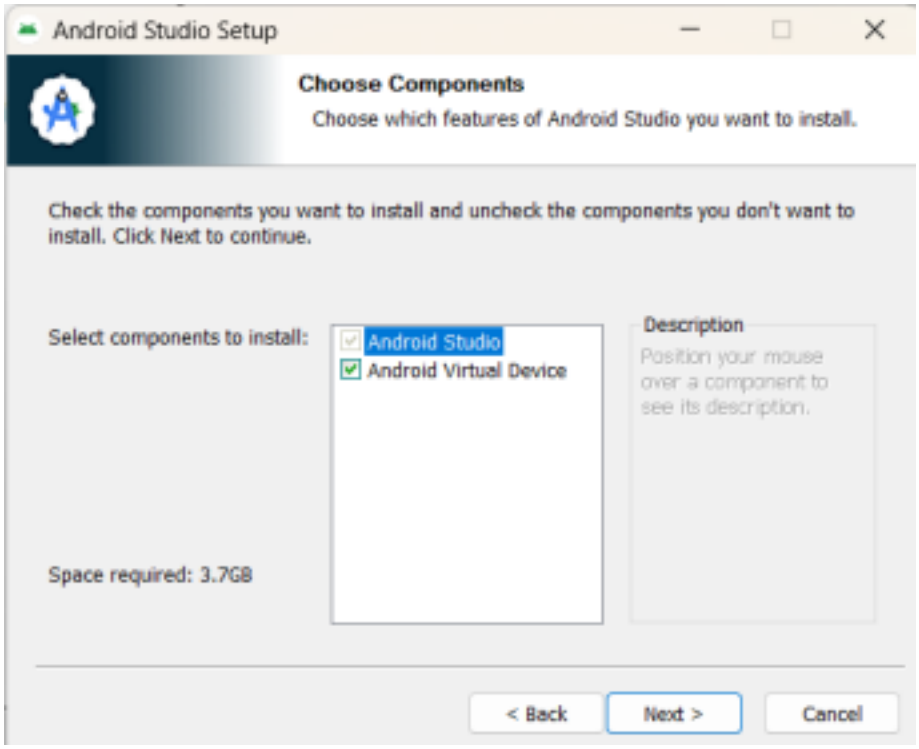
Step 8 : - Go to Android Studio and download the installer.



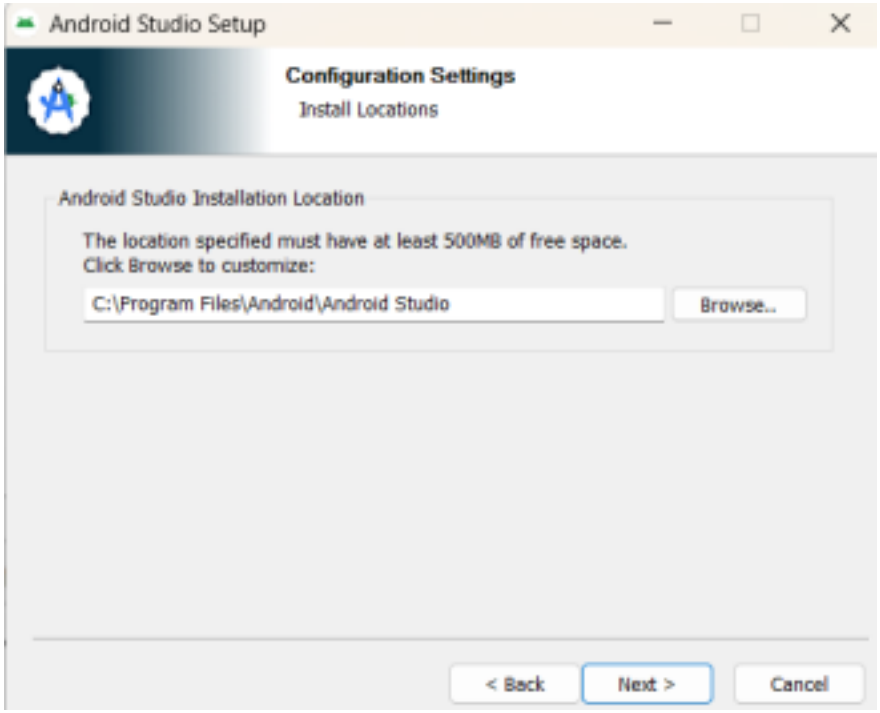
Step 8.1: - When the download is complete, open the .exe file and run it. You will get the following dialog box



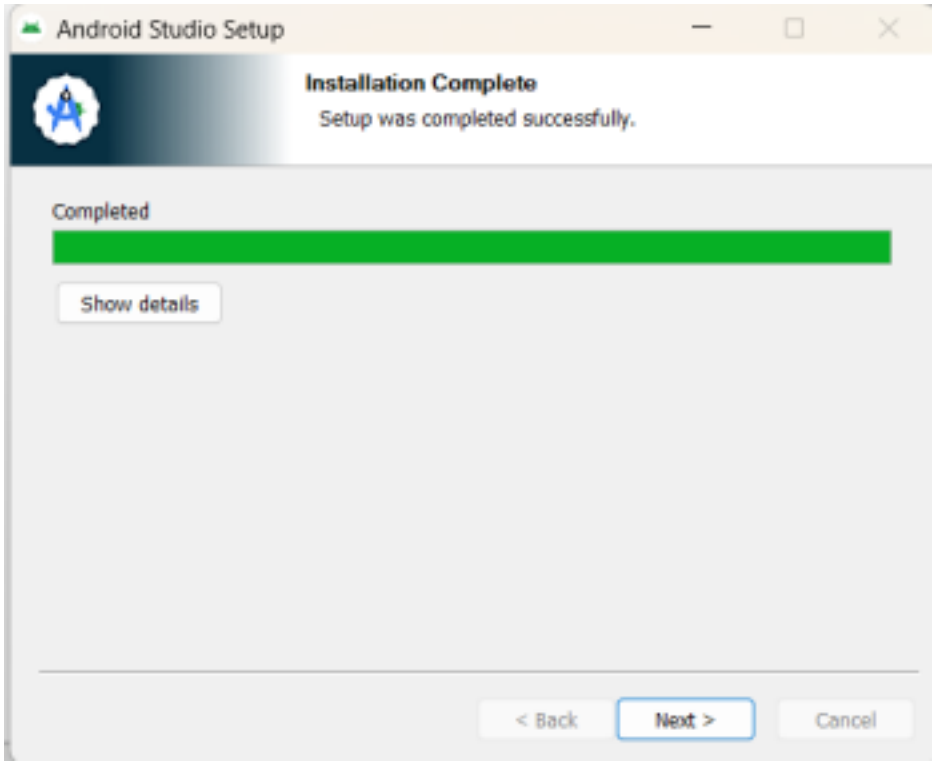
Step 8.2: - Select all the Checkboxes and Click on 'Next' Button.

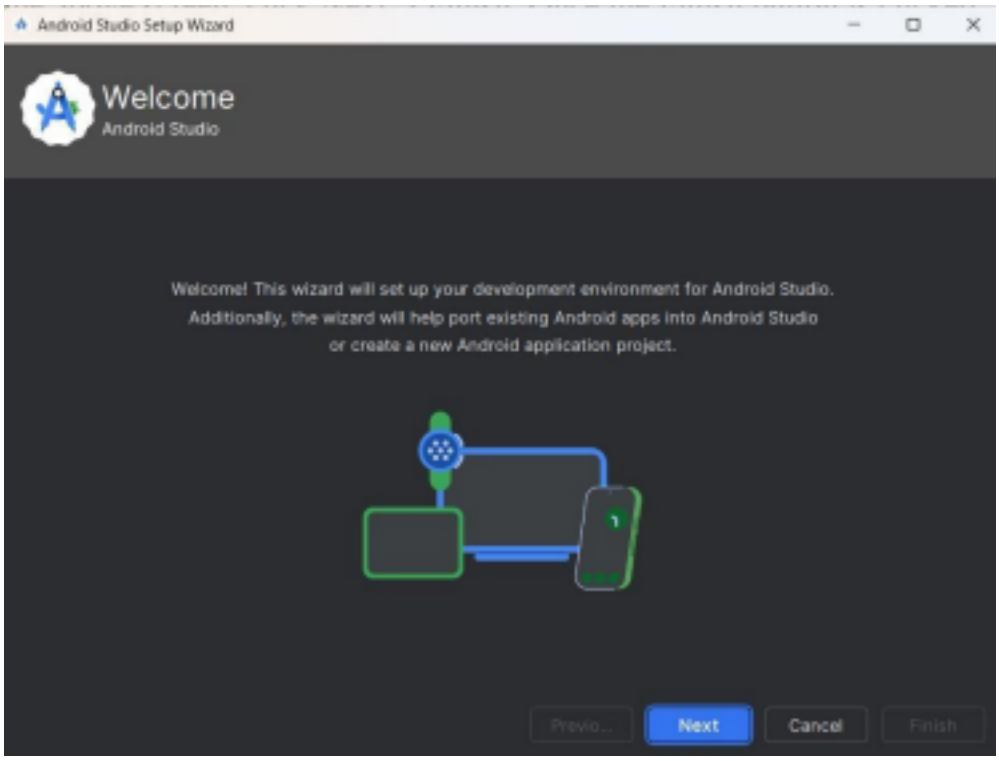


Step 8.3: - Change the destination as per your convenience and click on 'Next'

Button. 

Step 8.4: - Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.

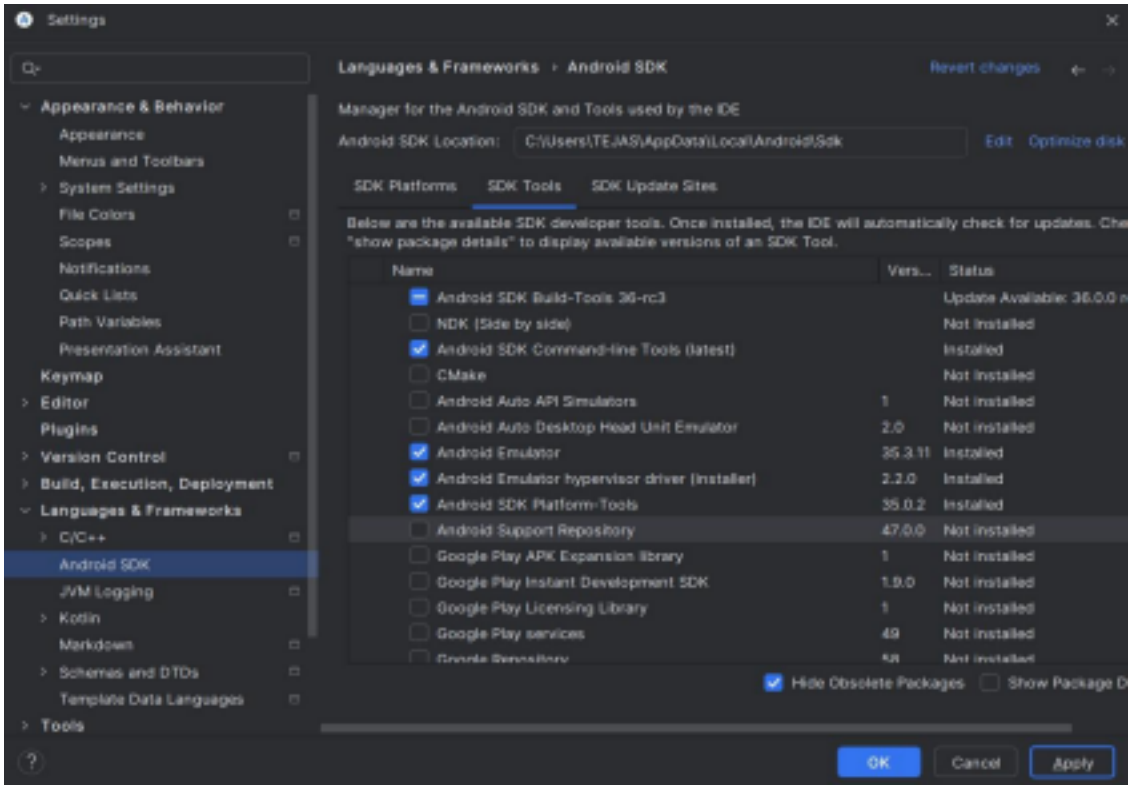




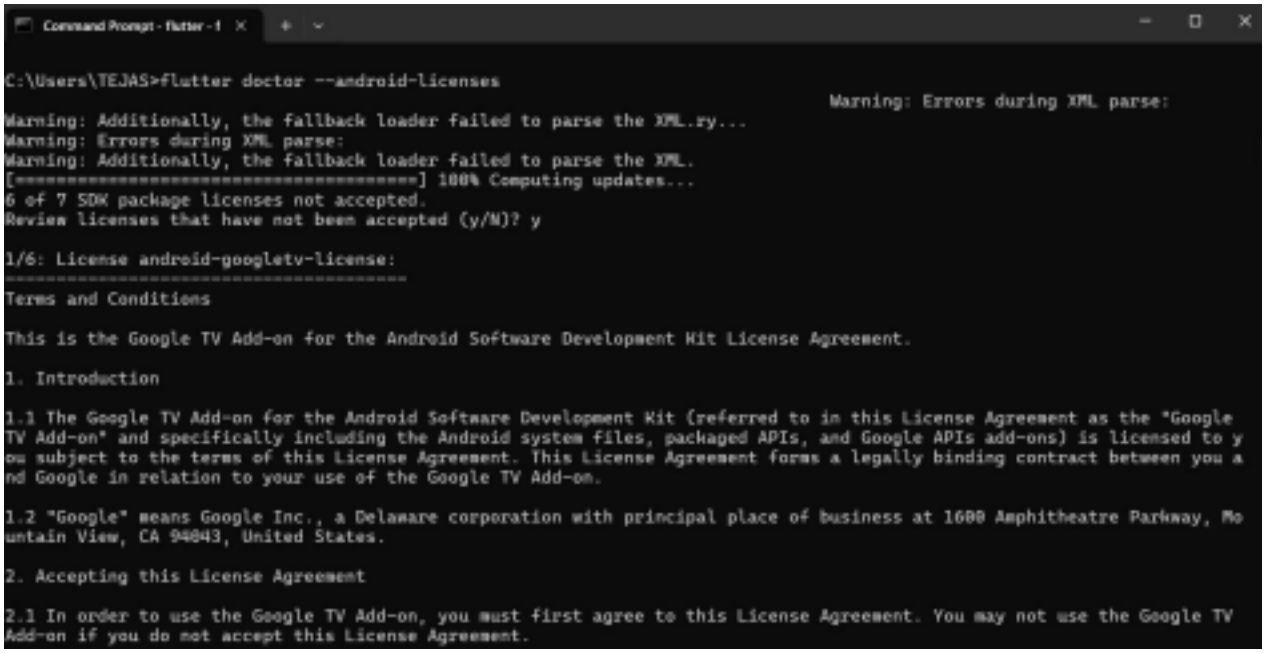
Step 8.5: -

Go to Preferences > Appearance & Behavior > System Settings > Android SDK.

Select the SDK Tools tab and check Android SDK Command-line Tools and Install it.



Step 9: - Open a terminal and run the following command



Step 10: - Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application

Step 10.1: - Open Android Studio and go to Tools > AVD Manager. Create a new virtual device.

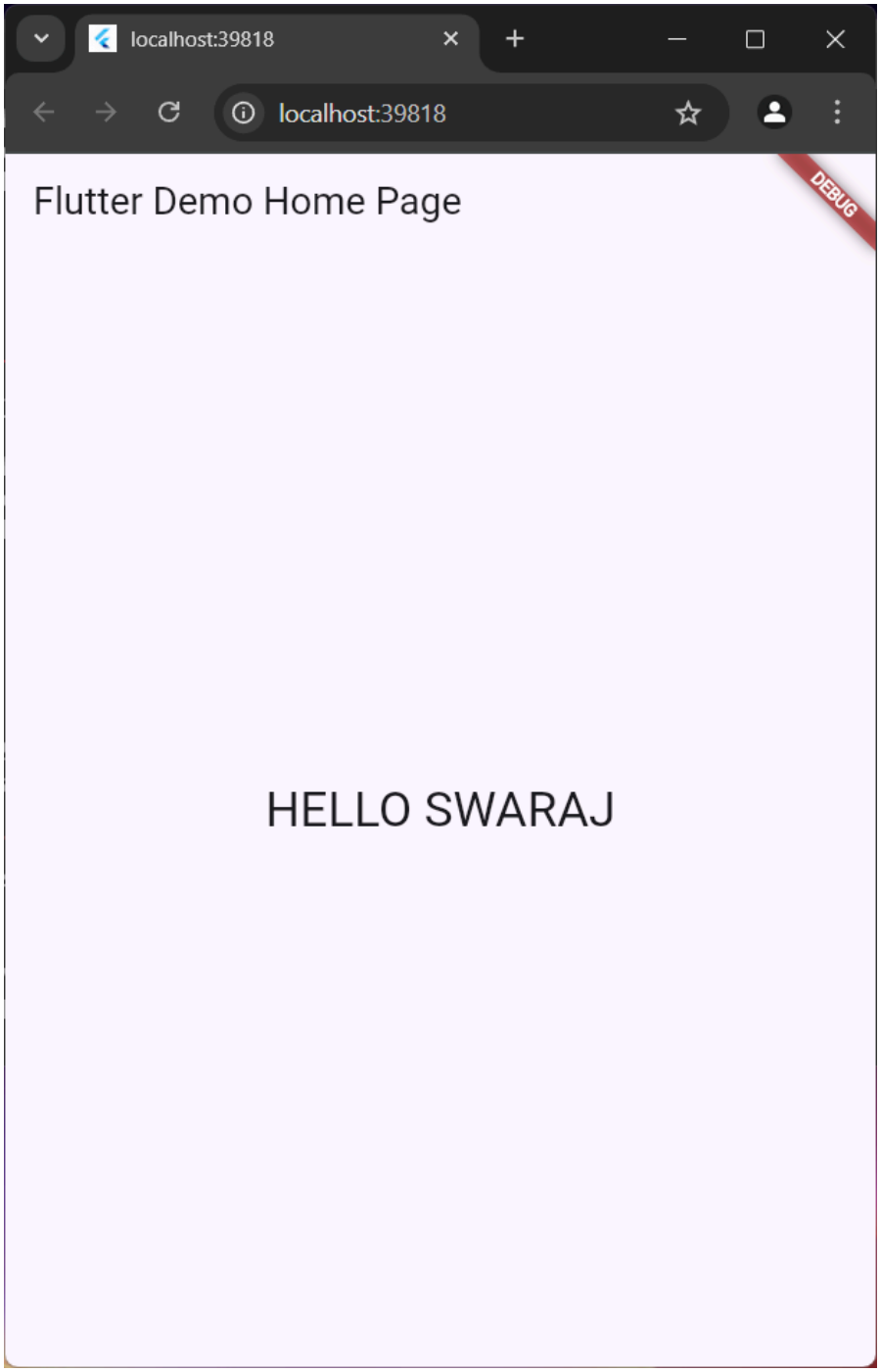
Step 10.2: - Click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen

Step 11: - Now, install Flutter and Dart plugin for building Flutter application in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself

Step 11.1: - Open the Android Studio and then go to File->Settings->Plugins. Now, search the Flutter plugin. If found, select Flutter plugin and click install

Step 11.2: - Restart the Android Studio

Step 12: - Go to File > New Project > Create Flutter Project, then select the project name and location, and click Next to proceed.


localhost:39818
localhost:39818
Flutter Demo Home Page
DEBUG
HELLO SWARAJ

EXPERIMENT NO: - 02

Name:- Swaraj Patil **Class:-** D15A **Roll:No: -** 39 **AIM: -** To design Flutter UI by including

common widgets.

Theory: -

https://github.com/Swap0-4/MPL-LAB/blob/main/MPL%20EXPERIMENT%20NO

%2002.pdf

Each element on the screen of the Flutter app is a widget. The view of the screen completely depends upon the choice and sequence of the widgets used to build the apps. And the structure of the code of apps is a tree of widgets.

When you made any alteration in the code, the widget rebuilds its description by calculating the difference of previous and current widget to determine the minimal changes for rendering in UI of the app.Widgets are nested with each other to build the app. It means the root of your app is itself a widget, and all the way down is a widget also. For example, a widget can display something, can define design, can handle interaction, etc.

The single child layout widget is a type of widget, which can have only **one child widget** inside the parent layout widget. These widgets can also contain special layout functionality. Flutter provides us many single child widgets to make the app UI attractive. If we use these widgets appropriately, it can save our time and makes the app code more readable.

The multiple child widgets are a type of widget, which contains **more than one child widget**, and the layout of these widgets are **unique**. For example, Row widget laying out of its child widget in a horizontal direction, and Column widget laying out of its child widget in a vertical direction. If we combine the Row and Column widget, then it can build any level of the complex widget.

Type of Widgetss

➢ **StatefulWidget**

A StatefulWidget has state information. It contains mainly two classes: the state object and the widget. It is dynamic because it can change the inner data during the widget lifetime. This widget does not have a build() method. It has createState() method, which returns a class that extends the Flutters State Class. The examples of the StatefulWidget are Checkbox, Radio, Slider, InkWell, Form, and TextField.

➢ **StatelessWidget**

The StatelessWidget does not have any state information. It remains static throughout its lifecycle. The examples of the StatelessWidget are Text, Row, Column, Container, etc.

Some of the commonly used widgets

Container – A box widget used for styling with padding, margins, colors, borders, and constraints. It helps in layout structuring and positioning.

Row & Column – Used to arrange widgets in horizontal (Row) or vertical (Column) orientation. They manage spacing, alignment, and distribution of child widgets.

Stack – Overlaps widgets on top of each other, useful for creating layered UIs like banners, tooltips, or floating elements.

Text – Displays text on the screen with customizable font size, color, alignment, and styling

options Image – Loads and displays images from assets, network, or memory with scaling, fit,

properties.

Scaffold – Provides a basic layout structure with an app bar, body, floating action button, and bottom navigation.

ListView – A scrollable list widget that efficiently renders large amounts of dynamic content. Supports both vertical and horizontal scrolling.

GridView – Displays widgets in a grid format, useful for galleries, product listings, or dashboards. It supports dynamic column adjustments.

SizedBox – Used to create space between widgets or define fixed width and height for layout adjustments.

ElevatedButton – A button with elevation that provides a raised effect, customizable with color, shape, and click actions.

TextField – A user input field that supports text entry, keyboard configurations, validation.

AppBar – A top navigation bar that includes a title, actions, and menu icons, commonly used in Scaffold.

BottomNavigationBar – A bar at the bottom of the screen used for navigation between different app sections with icons and labels.

Drawer – A side navigation panel that slides out from the left, typically used for app menus and quick navigation.

Card – A material design component that displays content inside a box with elevation.

Code: -

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









void main() {



  runApp(MyApp());



}
```

```dart
class MyApp extends StatelessWidget {

  const MyApp({super.key});




  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      title: 'Telegram',

      theme: ThemeData(

        primarySwatch: Colors.blue,

        appBarTheme: const AppBarTheme(
```

```
      color: Colors.white,

      iconTheme: IconThemeData(color: Colors.blue),

      elevation: 0,

      titleTextStyle: TextStyle(

        color: Colors.black,

        fontSize: 20,

        fontWeight: FontWeight.bold,

      ),

    ),

    scaffoldBackgroundColor: Colors.white,
```

```
      ),

      home: const HomePage(), // Corrected: Now points to HomePage

    );

  }

}

class Chat {

  final String name;

  final String lastMessage;

  final String time;
```

```dart
final int unreadCount;

final bool isOnline;

final bool isTyping;

final bool isMuted;

final bool isPinned;




Chat({

  required this.name,

  required this.lastMessage,

  required this.time,
```

```
    this.unreadCount = 0,

    this.isOnline = false,

    this.isTyping = false,

    this.isMuted = false,

    this.isPinned = false,

  });

}

class HomePage extends StatelessWidget {

  const HomePage({super.key});
```

```dart
@override

Widget build(BuildContext context) {

  final List<Chat> chats = [ // Initialized here!

    Chat(

      name: 'Emma',

      lastMessage: 'Hey, are you coming to the party?',

      time: '10:02',

      unreadCount: 2,

      isOnline: true,
```

```
    ),

    Chat(
      name: 'Noah',
      lastMessage: 'Check out this new app!',
      time: '09:45',
      isTyping: true,
    ),

    Chat(
      name: 'Olivia',
      lastMessage: '📷 Photo',
```

```
    time: '09:30',
    isMuted: true,
  ),
  Chat(
    name: 'Liam',
    lastMessage: 'Thanks for the help!',
    time: '08:15',
    unreadCount: 1,
  ),
  Chat(
```

```
      name: 'Telegram',

      lastMessage: 'Update available',

      time: '07:00',

      isPinned: true,

    ),

  ];

  return Scaffold(

    appBar: AppBar(

      title: const Text('Telegram'),
```

```
      actions: [

        IconButton(

          icon: const Icon(Icons.search, color: Colors.blue),

          onPressed: () {},

        ),

        PopupMenuButton<String>(

          icon: const Icon(Icons.more_vert, color: Colors.blue),

          onSelected: (value) {},

          itemBuilder: (BuildContext context) => [

            const PopupMenuItem(value: 'New Group', child: Text('New
```

```
Group')),


              const PopupMenuItem(value: 'New Channel', child: Text('New
Channel')),


              const PopupMenuItem(value: 'Settings', child:
Text('Settings')),


            ],


          ),


        ],


      ),


      body: ListView.separated(


        itemCount: chats.length,


        separatorBuilder: (context, index) => const Divider(height: 1),
```

```
      itemBuilder: (context, index) {

        final chat = chats[index];

        return ListTile(

          leading: _buildAvatar(chat),

          title: Row(

            children: [

              Text(

                chat.name,

                style: TextStyle(

                  fontWeight: chat.unreadCount > 0 || chat.isPinned
```

```
                    ? FontWeight.bold


                    : FontWeight.normal,


                ),


              ),


              if (chat.isMuted)


                const Icon(Icons.volume_off, size: 16, color:
Colors.grey),


            ],


          ),


          subtitle: chat.isTyping
```

```
              ? const Text('typing...', style: TextStyle(color:
Colors.green))



              : Text(



                  chat.lastMessage,



                  style: TextStyle(



                    color: chat.unreadCount > 0 ? Colors.black :
Colors.grey,



                    fontWeight: chat.unreadCount > 0



                        ? FontWeight.bold



                        : FontWeight.normal,



                  ),
```

```
                maxLines: 1,

                overflow: TextOverflow.ellipsis,

              ),

        trailing: _buildTrailing(chat),

        onTap: () {},

      );

    },

  ),

  floatingActionButton: FloatingActionButton(

    backgroundColor: Colors.blue,
```

```
      onPressed: () {},

      child: const Icon(Icons.edit, color: Colors.white),

    ),

    bottomNavigationBar: BottomNavigationBar(

      currentIndex: 0,

      selectedItemColor: Colors.blue,

      unselectedItemColor: Colors.grey,

      type: BottomNavigationBarType.fixed,

      items: const [ // Added const here

        BottomNavigationBarItem(
```

```
        icon: Icon(Icons.chat_bubble),

        label: 'Chats',

      ),

      BottomNavigationBarItem(

        icon: Icon(Icons.call),

        label: 'Calls',

      ),

      BottomNavigationBarItem(

        icon: Icon(Icons.people),

        label: 'Contacts',
```

```
        ),

        BottomNavigationBarItem(

          icon: Icon(Icons.settings),

          label: 'Settings',

        ),

      ],

    ),

  );

}
```

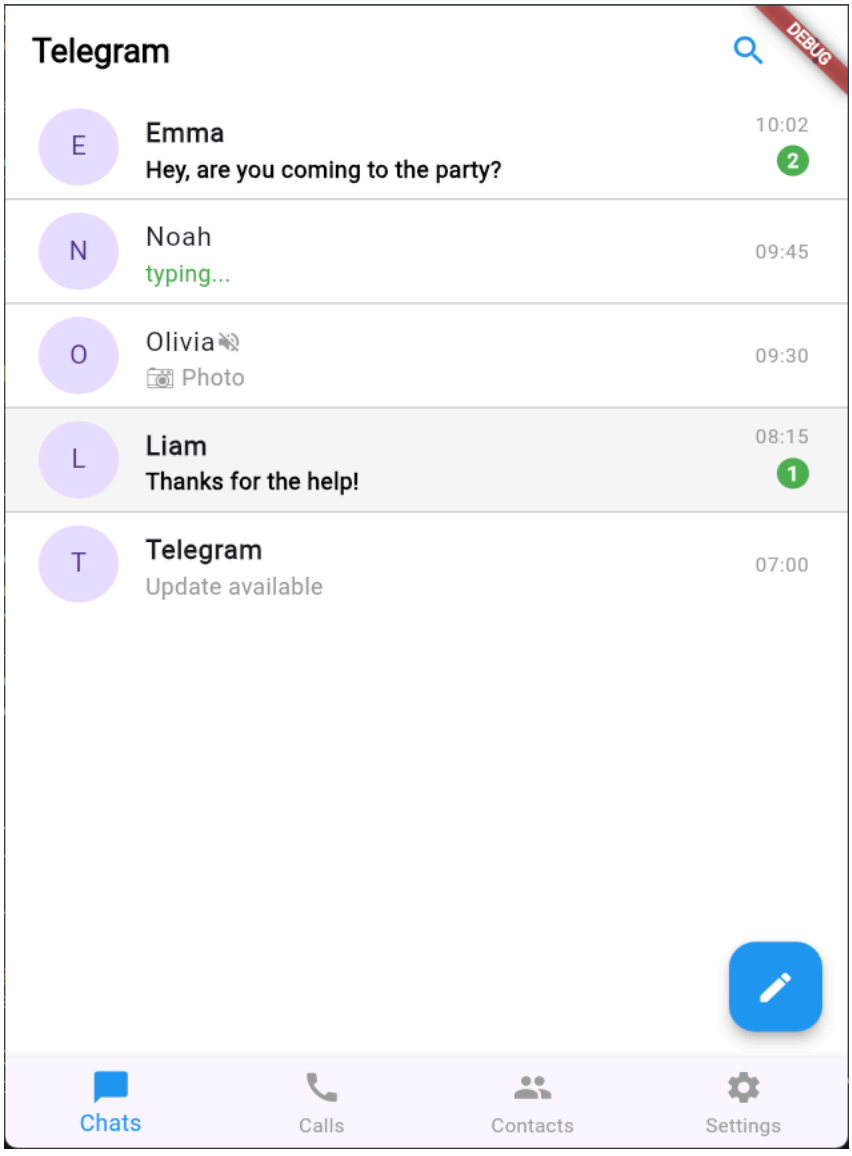
```
Widget _buildAvatar(Chat chat) {

  return CircleAvatar(

    radius: 28,

    child: Text(chat.name[0]),

  );

}



Widget _buildTrailing(Chat chat) {

  return Column(

    mainAxisAlignment: MainAxisAlignment.center,
```

```
    crossAxisAlignment: CrossAxisAlignment.end,

    children: [

      Text(

        chat.time,

        style: const TextStyle(

          color: Colors.grey,

          fontSize: 12,

        ),

      ),

      if (chat.unreadCount > 0)
```

```
        Container(

          padding: const EdgeInsets.all(6),

          decoration: const BoxDecoration(

            color: Colors.green,

            shape: BoxShape.circle,

          ),

          child: Text(

            chat.unreadCount.toString(),

            style: const TextStyle(

              color: Colors.white,
```

```
                fontSize: 12,


                fontWeight: FontWeight.bold,


              ),


            ),


          ),


      ],


    );


  }


}
```

OUTPUT:


Telegram
DEBUG
E
Emma
Hey, are you coming to the party?
10:02
2
N
Noah
typing...
09:45
O
Olivia
Photo
09:30
L
Liam
Thanks for the help!
08:15
1
T
Telegram
Update available
07:00
Chats
Calls
Contacts
Settings

MAD & PWA LAB
Experiment 3

Name: Swaraj Patil **Class:** D15A **Roll no:39**

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

Code:

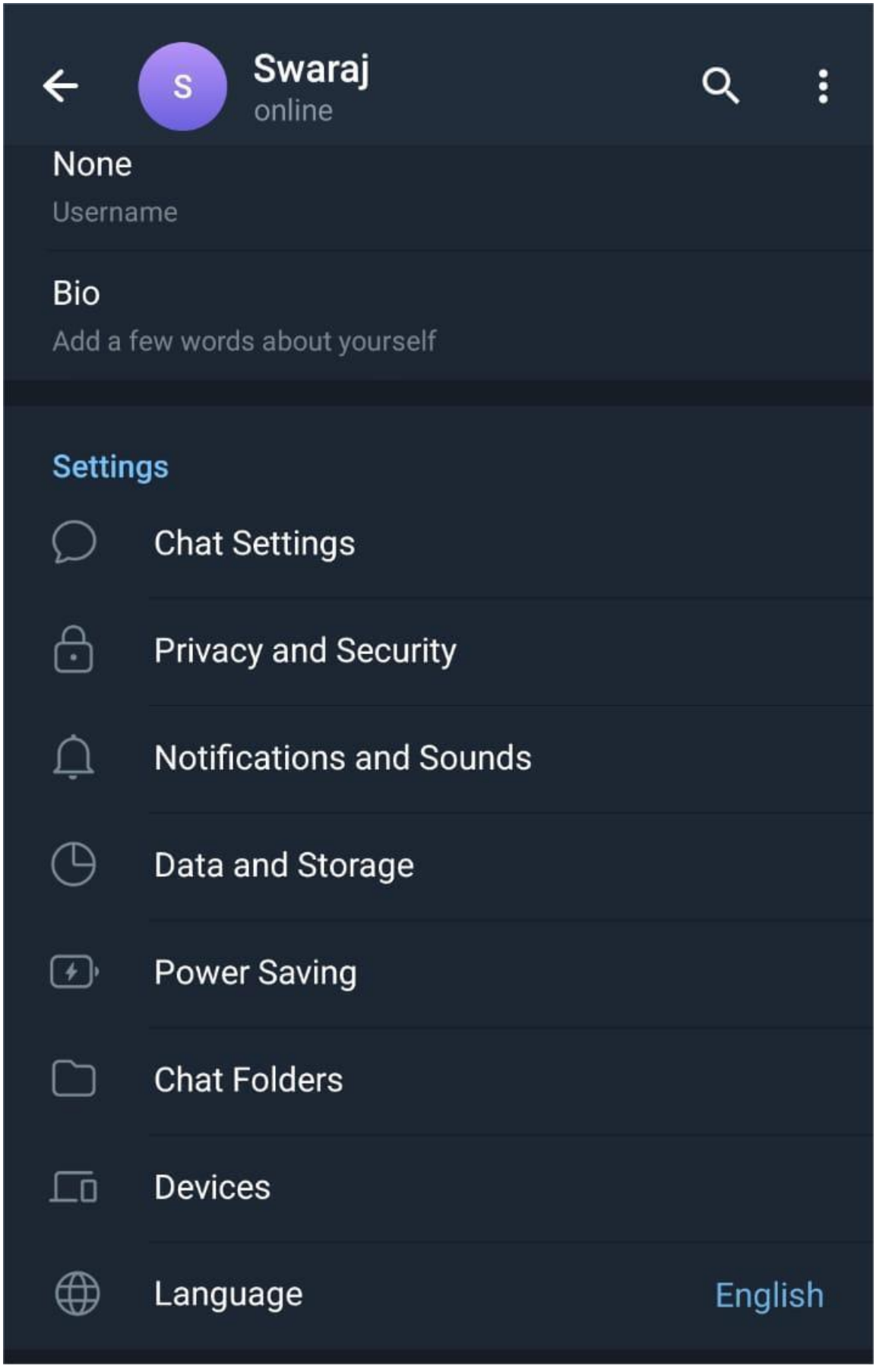
```
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Telegram UI Elements',
      theme: ThemeData.dark().copyWith(
        primaryColor: const Color(0xFF0088cc), // Telegram blue
        scaffoldBackgroundColor: const Color(0xFF17212B), // Telegram dark background
        appBarTheme: const AppBarTheme(
          backgroundColor: Color(0xFF1C2B36), // Telegram app bar color
          elevation: 0,
        ),
      ),
      home: const TelegramUIPage(),
    );
  }
}

class TelegramUIPage extends StatelessWidget {
  const TelegramUIPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(
          'Telegram',
          style: GoogleFonts.heebo(
            fontWeight: FontWeight.w600,
            fontSize: 20,
```

```
      color: const Color(0xFFCED6E0), // Telegram text
    ),
  ),
  actions: [
    IconButton(
      icon: const Icon(Icons.search, color: Color(0xFFCED6E0)),
      onPressed: () {},
    ),
    IconButton(
      icon: const Icon(Icons.more_vert, color: Color(0xFFCED6E0)),
      onPressed: () {},
    ),
  ],
),
body: Padding(
  padding: const EdgeInsets.all(16.0),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        'Main Navigation Icons:',
        style: GoogleFonts.heebo(
          fontSize: 16,
          fontWeight: FontWeight.w600,
          color: const Color(0xFFCED6E0),
        ),
      ),
      const SizedBox(height: 16),
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: const [
          Icon(Icons.home, size: 24, color: Color(0xFFCED6E0)),
          Icon(Icons.chat_bubble, size: 24, color: Color(0xFF0088cc)),
          Icon(Icons.group, size: 24, color: Color(0xFFCED6E0)),
          Icon(Icons.call, size: 24, color: Color(0xFFCED6E0)),
        ],
      ),
      const SizedBox(height: 32),
      Text(
        'Chat Actions:',
        style: GoogleFonts.heebo(
          fontSize: 16,
          fontWeight: FontWeight.w600,
          color: const Color(0xFFCED6E0),
        ),
      ),
      const SizedBox(height: 16),
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: const [
          Icon(Icons.mic, size: 24, color: Color(0xFF0088cc)),
```

```
    Icon(Icons.attach_file, size: 24, color: Color(0xFFCED6E0)),
    Icon(Icons.camera_alt, size: 24, color: Color(0xFFCED6E0)),
    Icon(Icons.emoji_emotions, size: 24, color: Color(0xFFCED6E0)),
   ],
  ),
  const SizedBox(height: 32),
  Text(
   'Profile/Status Style:',
   style: GoogleFonts.heebo(
    fontSize: 16,
    fontWeight: FontWeight.w600,
    color: const Color(0xFFCED6E0),
   ),
  ),
  const SizedBox(height: 16),
  Row(
   children: [
    Container(
     width: 50,
     height: 50,
     decoration: BoxDecoration(
      shape: BoxShape.circle,
      border: Border.all(
       color: const Color(0xFF0088cc),
       width: 2,
      ),
      color: const Color(0xFF1E2C36),
     ),
     child: const Icon(
      Icons.person,
      color: Color(0xFFCED6E0),
      size: 30,
     ),
    ),
    const SizedBox(width: 16),
    Column(
     crossAxisAlignment: CrossAxisAlignment.start,
     children: [
      Text(
       'Contact Name',
       style: GoogleFonts.heebo(
        fontSize: 16,
        fontWeight: FontWeight.w600,
        color: Colors.white,
       ),
      ),
      Text(
       'Last seen recently',
       style: GoogleFonts.heebo(
        fontSize: 13,
        color: const Color(0xFFCED6E0),
```

```
                ),
              ),
            ],
          ),
        ],
      ),
      const SizedBox(height: 32),
      Container(
        margin: const EdgeInsets.only(right: 50),
        padding: const EdgeInsets.all(12),
        decoration: BoxDecoration(
          color: const Color(0xFF1E2C36),
          borderRadius: BorderRadius.circular(8),
        ),
        child: Text(
          'This is a Telegram message bubble example',
          style: GoogleFonts.heebo(
            fontSize: 15,
            color: Colors.white,
          ),
        ),
      ),
    ],
  ),
 ),
 floatingActionButton: FloatingActionButton(
  backgroundColor: const Color(0xFF0088cc),
  onPressed: () {},
  child: const Icon(Icons.edit), // Telegram uses pencil/edit icon
 ),
 );
 }
}
```

Output:


Swaraj
online
None
Username
Bio
Add a few words about yourself
Settings
Chat Settings
Privacy and Security
Notifications and Sounds
Data and Storage
Power Saving
Chat Folders
Devices
Language
English

EXPERIMENT NO: - 04

Name:- Swaraj Patil
Class:- D15A
Roll:No: - 39

AIM: - To create an interactive Form using form widget.

Theory: -

A form in Flutter is a structured container that collects user input through various fields like text fields, dropdowns, checkboxes, and buttons. It plays a crucial role in applications that require user data entry, such as login pages, registration forms, and feedback submissions. Flutter provides the Form widget, which works alongside TextFormField and other input elements to manage validation, state handling, and error messages efficiently. By using form validation techniques, developers can ensure data accuracy and enhance user experience.

When you create a form, it is necessary to provide the GlobalKey. This key uniquely identifies the form and allows you to do any validation in the form fields. The form widget uses child widget TextFormField to provide the users to enter the text field. This widget renders a material design text field and also allows us to display validation errors when they occur.

Creation of a Form

- While creating a form in Flutter, the **Form widget** is essential as it acts as a container for grouping multiple form fields and managing validation.

- A **GlobalKey<FormState>** is required to uniquely identify the form and enable validation or data retrieval from the form fields.

- The **TextFormField widget** is used to provide input fields where users can enter data such as names, phone numbers, or email addresses.

- To enhance the appearance and usability of input fields, **InputDecoration** is used, allowing customization of labels, icons, borders, and hint text.

- Validation plays a crucial role in forms, and the **validator property** within **TextFormField** ensures user input meets specific criteria before submission.

- Different types of input require appropriate **keyboard types**, such as TextInputType.number for numeric fields or TextInputType.emailAddress for email fields.

- Proper **state management** is needed to store and retrieve user input, ensuring the form data is processed correctly.

- A **submit button** is necessary to trigger form validation and submit the collected data for further processing.

Some Properties of Form Widget

- **key:** A GlobalKey that uniquely identifies the Form. You can use this

key to interact with the form, such as validating, resetting, or saving its state.

- **child:** The child widget that contains the form fields. Typically, this is a Column, ListView, or another widget that allows you to arrange the form fields vertically.
- **autovalidateMode:** An enum that specifies when the form should automatically validate its fields.

Some Methods of Form Widget

- **validate():** This method is used to trigger the validation of all the form fields within the Form. It returns true if all fields are valid, otherwise false. You can use it to check the overall validity of the form before submitting it.
- **save():** This method is used to save the current values of all form fields. It invokes the onSaved callback for each field. Typically, this method is called after validation succeeds.
- **reset():** Resets the form to its initial state, clearing any user-entered data.
- **currentState:** A getter that returns the current FormState associated with the Form.

Code: -

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


class OTPVerificationScreen extends
StatefulWidget {
  final String phoneNumber;


  const
OTPVerificationScreen({super.key,
required this.phoneNumber});


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


class _OTPVerificationScreenState
extends State<OTPVerificationScreen>
{
  final Color _primaryColor = const
Color(0xFF0088CC);
  final List<TextEditingController>
_otpControllers = List.generate(5,
(index) => TextEditingController());
  final List<FocusNode>
_otpFocusNodes = List.generate(5,
(index) => FocusNode());

  @override
  void dispose() {
    for (var controller in
_otpControllers) {
      controller.dispose();
    }
    for (var node in _otpFocusNodes)
{
      node.dispose();
    }
    super.dispose();
  }


  @override
  Widget build(BuildContext context)
{
    return Scaffold(
      backgroundColor: Colors.white,
      appBar: AppBar(
        elevation: 0,
        backgroundColor:
Colors.white,
        leading: IconButton(
          icon: const
Icon(Icons.arrow_back, color:
Colors.black),
          onPressed: () =>
```

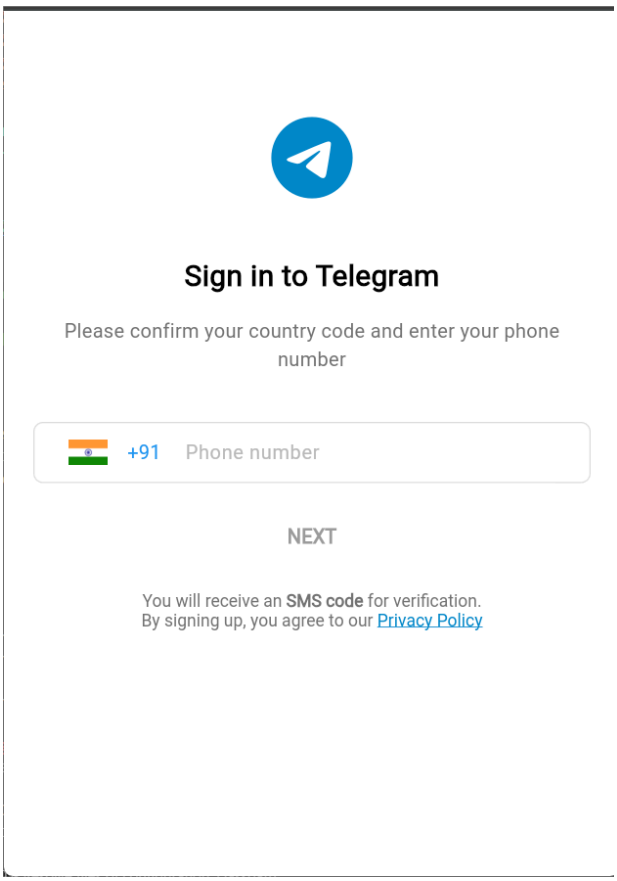
```
Navigator.pop(context),
        ),
      ),
      body: SingleChildScrollView(
        child: Padding(
          padding: const
EdgeInsets.symmetric(horizontal:
24),
          child: Column(
            crossAxisAlignment:
CrossAxisAlignment.start,
            children: [
              const SizedBox(height:
40),
              Text(
                'Verify your
number',
                style: TextStyle(
                  fontSize: 24,
                  fontWeight:
FontWeight.w600,
                  color:
Colors.grey[800],
                ),
              ),
              const SizedBox(height:
16),
              RichText(
                text: TextSpan(
                  style:
TextStyle(color: Colors.grey[600],
fontSize: 16),
                  children: [
                    const
TextSpan(text: 'We have sent a code
to '),
                    TextSpan(
                      text:
widget.phoneNumber,
                      style: const
TextStyle(fontWeight:
FontWeight.bold),
                    ),
                    const
TextSpan(text: '. Enter it below to
continue.'),
                  ],
                ),
              ),
              const SizedBox(height:
40),
              Row(
                mainAxisAlignment:
MainAxisAlignment.spaceBetween,
                children:
List.generate(5, (index) {
                  return SizedBox(
```

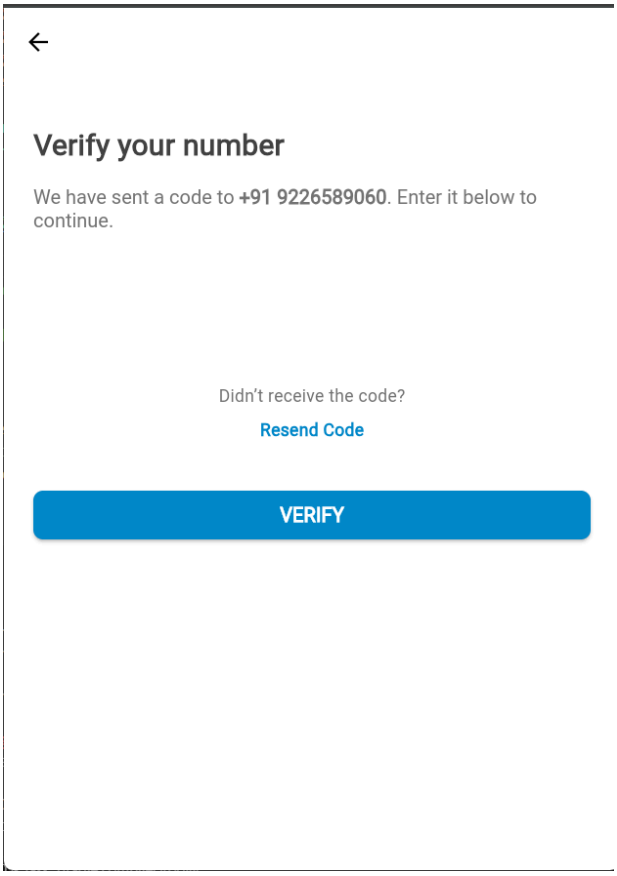
```
                    width: 50,
                    child:
TextField(
                      controller:
_otpControllers[index],
                      focusNode:
_otpFocusNodes[index],
                      textAlign:
TextAlign.center,
                      keyboardType:
TextInputType.number,
                      maxLength: 1,
                      style: const
TextStyle(fontSize: 24),
                      decoration:
InputDecoration(
                        counterText:
'',
                        border:
OutlineInputBorder(
borderRadius:
BorderRadius.circular(8),
borderSide: BorderSide(color:
Colors.grey[300]!),
                        ),
focusedBorder: OutlineInputBorder(
borderRadius:
BorderRadius.circular(8),
borderSide: BorderSide(color:
_primaryColor),
                        ),
                      ),
                      onChanged:
(value) {
                        if
(value.isNotEmpty && index < 4) {
_otpFocusNodes[index +
1].requestFocus();
                        } else if
(value.isEmpty && index > 0) {
_otpFocusNodes[index -
1].requestFocus();
                        }
                      },
                    ),
                  );
                }),
              ),
              const SizedBox(height:
24),
              Center(
                child: Text(
```

```
                'Didn't receive the code?',
                style: TextStyle(color: Colors.grey[600], fontSize: 14),
              ),
            ),
            const SizedBox(height: 8),
            Center(
              child: GestureDetector(
                onTap: () {
                  // Handle resend OTP logic
                },
                child: Text(
                  'Resend Code',
                  style: TextStyle(
                    color: _primaryColor,
                    fontSize: 14,
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ),
            ),
            const SizedBox(height: 40),
            SizedBox(
              width: double.infinity,
              child: MaterialButton(
                height: 48,
                color: _primaryColor,
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(8),
                ),
                onPressed: () {
                  // Handle OTP verification logic
                  String otp = _otpControllers.map((controller) => controller.text).join();
                  if (otp.length == 5) {
                    // Verify OTP
                  }
                },
                child: const Text(
```

```
                    'VERIFY',
                    style:
TextStyle(
                      color:
Colors.white,
                      fontSize: 16,
                      fontWeight:
FontWeight.bold,
                    ),
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

OUTPUT:


Sign in to Telegram
Please confirm your country code and enter your phone number
+91
Phone number
NEXT
You will receive an SMS code for verification.
By signing up, you agree to our Privacy Policy


Verify your number
We have sent a code to +91 9226589060. Enter it below to continue.
Didn't receive the code?
Resend Code
VERIFY

MAD & PWA LAB
Experiment 5

Name: Swaraj Patil **Class:** D15A **Roll no: 39**

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

Theory:

1. Navigation in Flutter
Navigation allows users to move between screens in an app. It is managed using the Navigator and a stack-based system. The most common navigation actions include:

- Push Navigation: Moves to a new screen.
- Pop Navigation: Goes back to the previous screen.
- Named Routes: Uses predefined names for navigation instead of directly creating new screen instances.

2. Routing in Flutter
Routing defines the path and logic for screen transitions. There are two main types:

- Static Routing: Predefined routes set in MaterialApp.
- Dynamic Routing: Routes created dynamically using onGenerateRoute, allowing flexible screen navigation.

3. Gestures in Flutter
Gestures enable user interactions such as tapping, swiping, and dragging. Flutter provides:

- GestureDetector: Detects custom gestures like tap, double-tap, and long press.
- InkWell: Adds ripple effects for a better touch response.

Code:

Bottom Navigation

```
bottomNavigationBar:
BottomNavigationBar(
 items: const
<BottomNavigationBarItem>[
  BottomNavigationBarItem(icon:
Icon(Icons.chat), label: 'Chats'),
  BottomNavigationBarItem(icon:
Icon(Icons.person), label: 'Profile'),
  BottomNavigationBarItem(
    icon: Icon(Icons.settings),
    label: 'Settings',
  ),
 ],
 currentIndex: _selectedIndex,
 onTap: _onItemTapped,
)
```

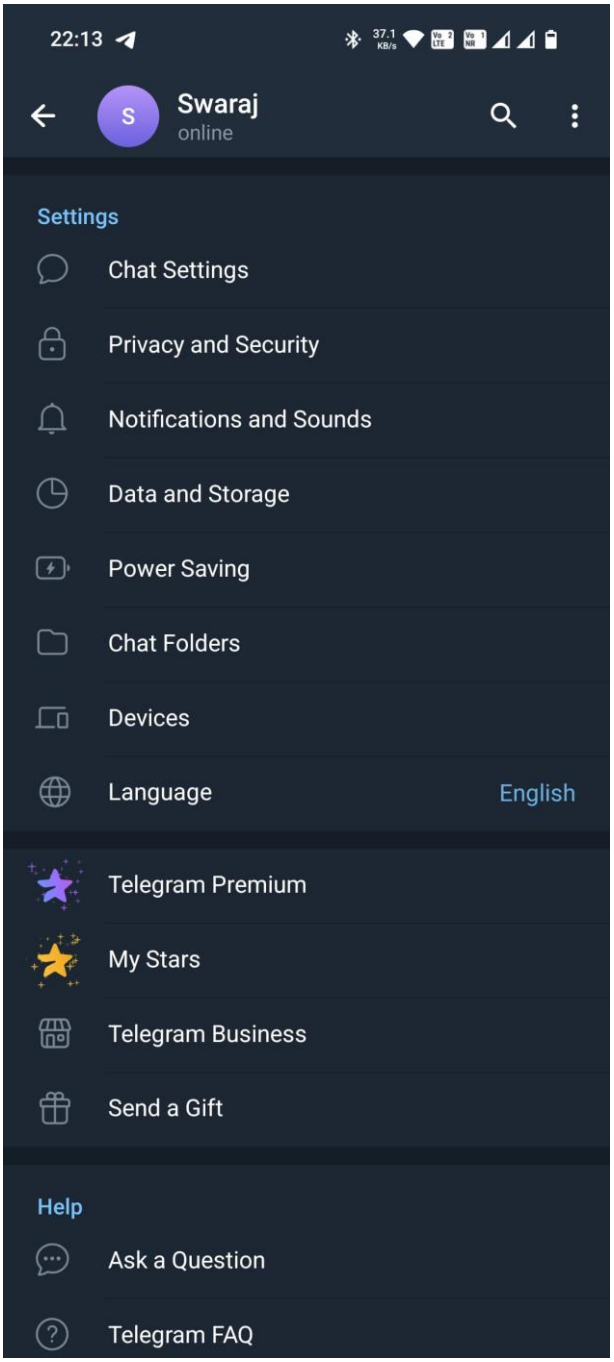
Navigation to Detail Screen

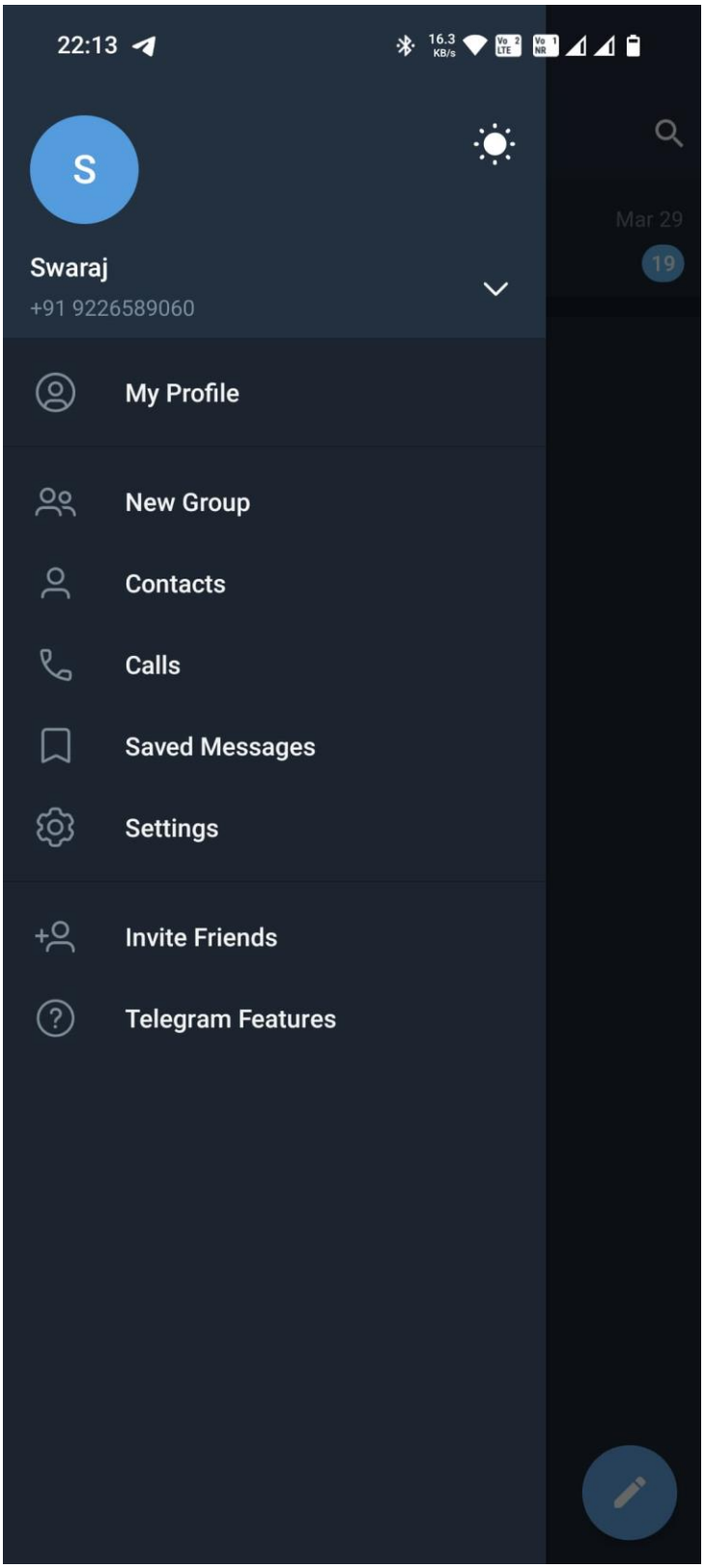
```
onTap: () {
  Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => DemoChatScreen(userName: 'User ${index + 1}'),
    ),
  );
}
```

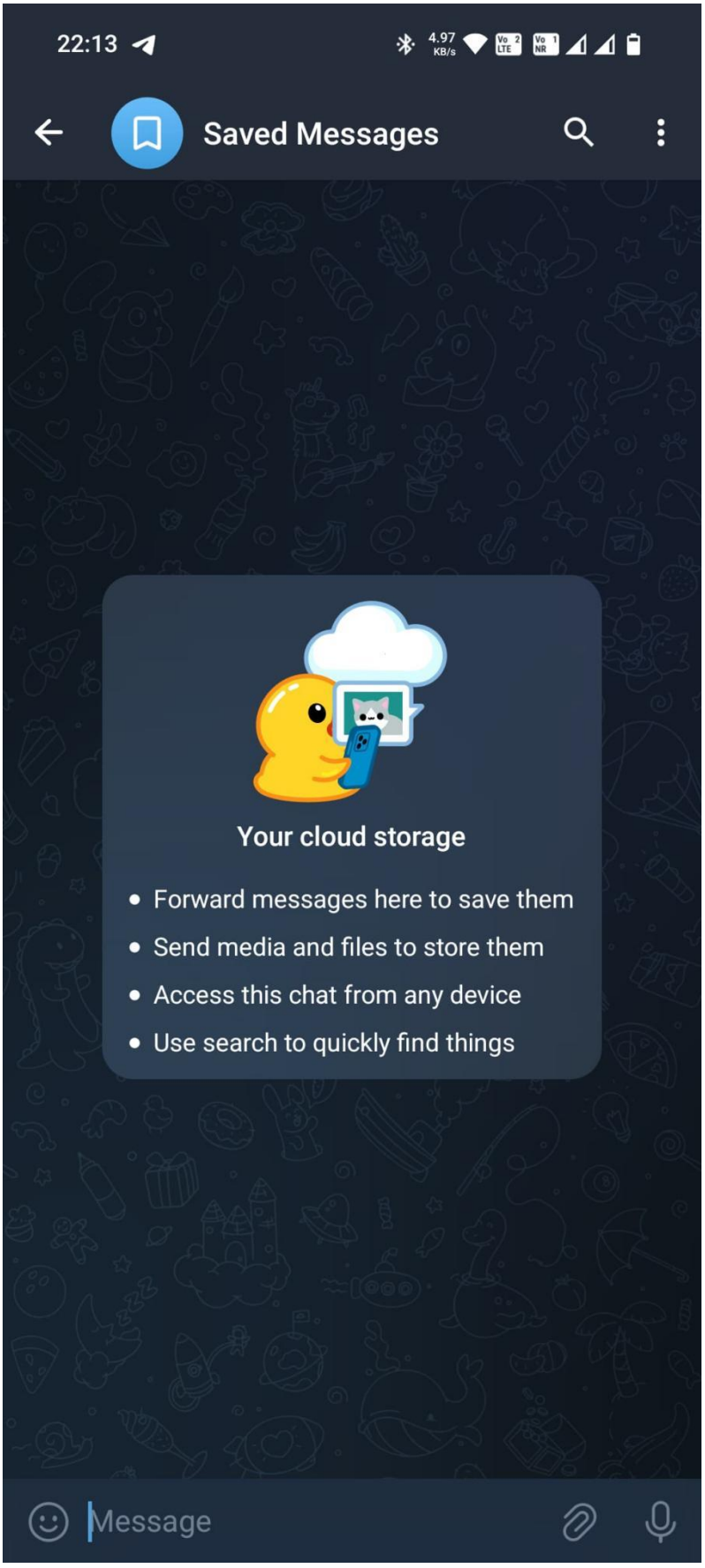
Navigation with Replacement

```
Navigator.of(context).pushReplacement(
  MaterialPageRoute(
    builder: (context) => const DemoLoginScreen(),
  ),
);
```

Output:


22:13
Swaraj
online
Settings
Chat Settings
Privacy and Security
Notifications and Sounds
Data and Storage
Power Saving
Chat Folders
Devices
Language
English
Telegram Premium
My Stars
Telegram Business
Send a Gift
Help
Ask a Question
Telegram FAQ


22:13
16.3 KB/s
S
Swaraj
+91 9226589060
Mar 29
19
My Profile
New Group
Contacts
Calls
Saved Messages
Settings
Invite Friends
Telegram Features


22:13
4.97 KB/s
Saved Messages
Your cloud storage
Forward messages here to save them
Send media and files to store them
Access this chat from any device
Use search to quickly find things
Message

EXPERIMENT NO:- 6

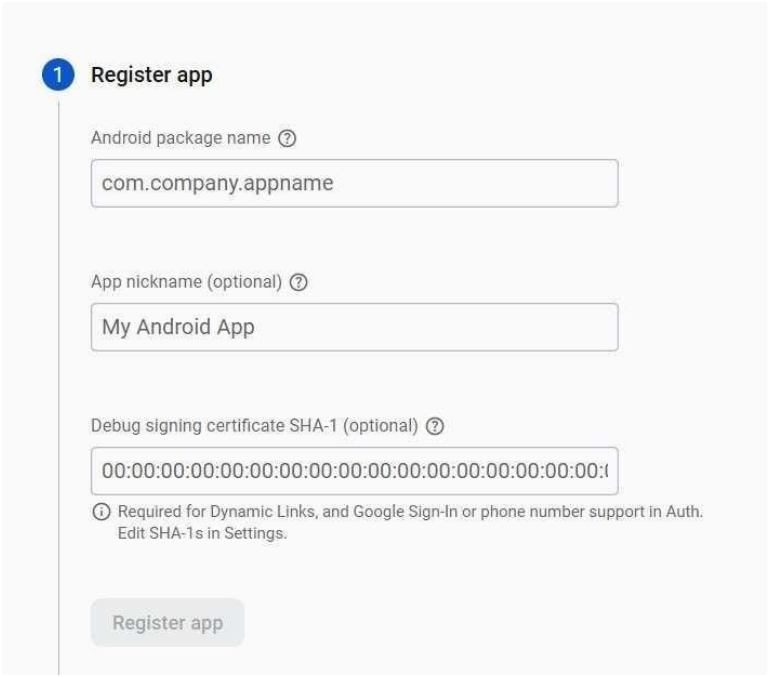
Name:-Swaraj Patil D15A Roll-no:-39

Aim:- To Connect flutter UI with firebase database

Creating a New Firebase Project

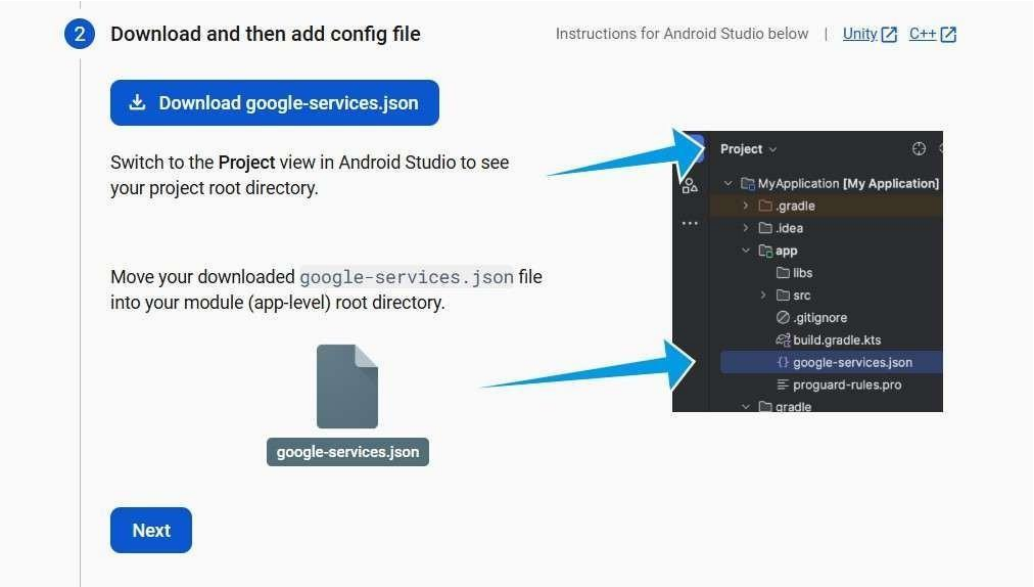
First, log in with your Google account to manage your Firebase projects. From within the Firebase dashboard, select the Create new project button and give it a name

In order to add Android support to our Flutter application, select the Android logo from the dashboard. This brings us to the following screen:



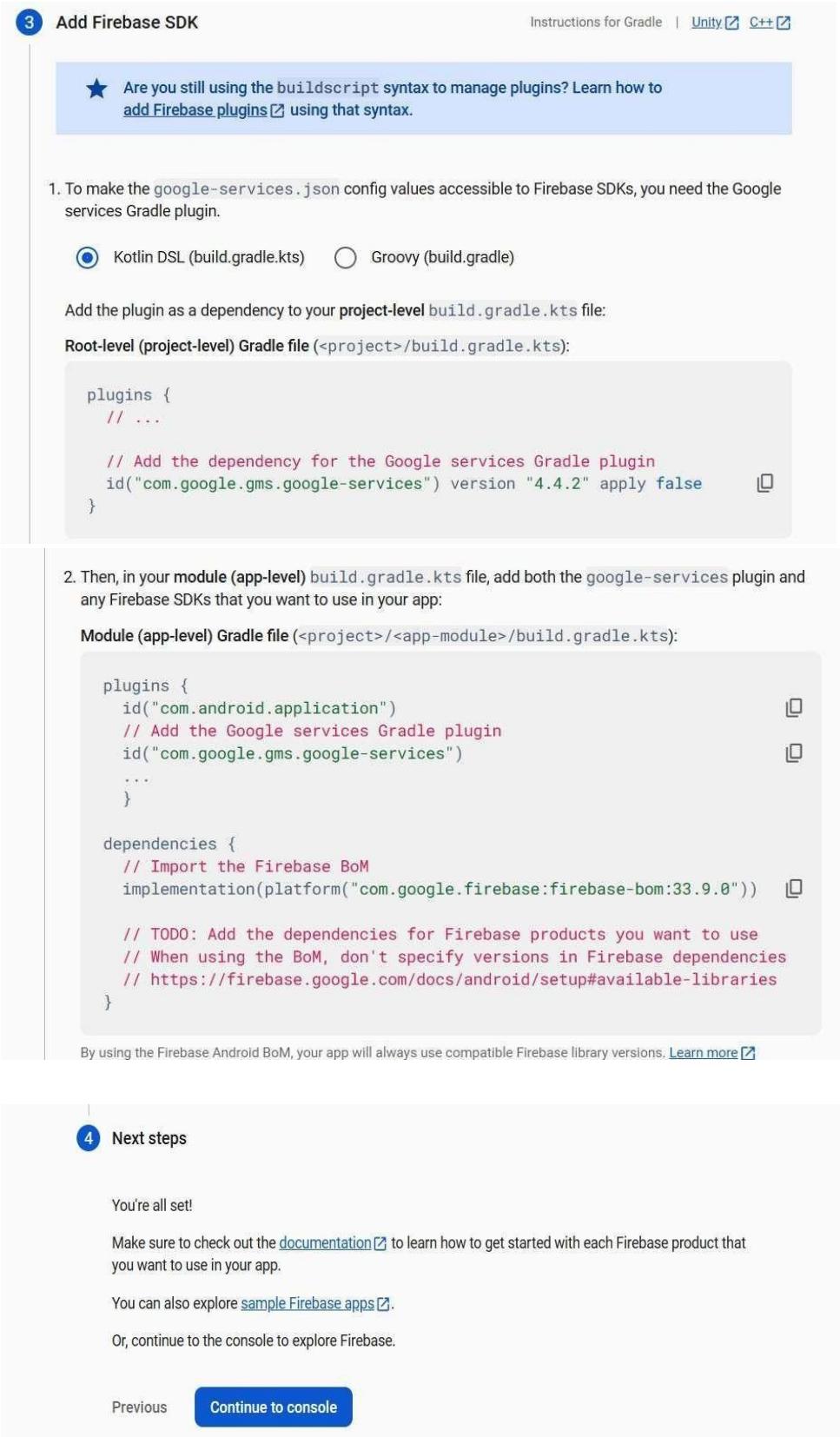
The most important thing here is to match up the Android package name that you choose here with the one inside of our application.

Then download the google-services.json file, that you will get.



put that file in the android folder (root level)

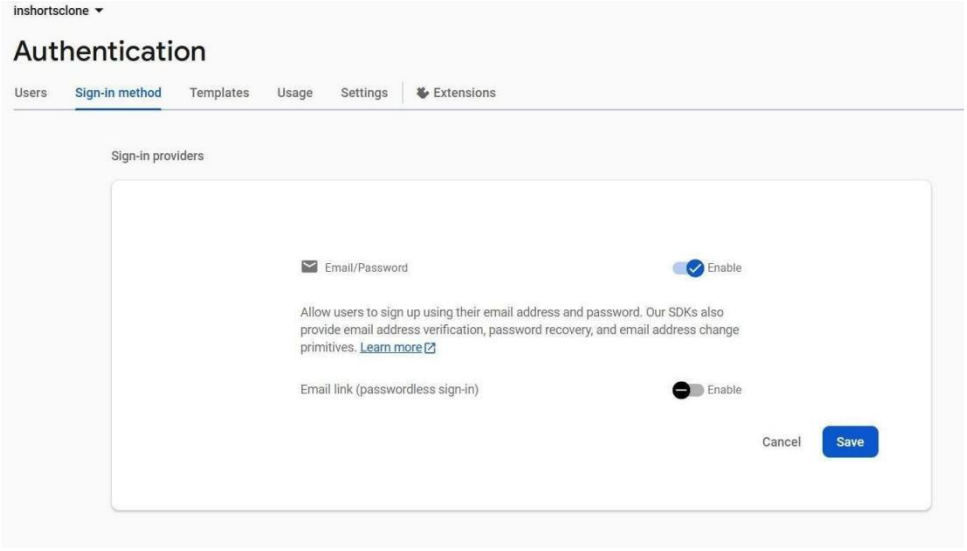
then select the build.gradle.kts (Kotlin DSL) part, and then follow the rest instructions



Generate the firebase_options.dart file, based on the google-services.json file

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

class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    return const FirebaseOptions(
      apiKey: "AIzaSyA_VIR7fj4d-b6tNZhW9qJ6GRRx5EXKqs0",
      appId: "1:388272292768:android:33180b2382688b18781ac5",
      messagingSenderId: "388272292768",
      projectId: "inshortsclone-848a9",
      storageBucket: "inshortsclone-848a9.firebasestorage.app",
      androidClientId: "1:388272292768:android:33180b2382688b18781ac5",
    );
  }
}
```

In your part select the sign-in method and enable it.



Code:-

Authenction_logic

```
import
'package:firebase_auth/firebase_auth.dart'
;
import
'package:google_sign_in/google_sign_in.d art';


class AuthService {
  final FirebaseAuth _auth =
FirebaseAuth.instance;


  // Sign in with Email & Password
  Future<User?> signInWithEmail(String
email, String password) async {
    try {
      UserCredential userCredential = await
_auth.signInWithEmailAndPassword( email:
        email,
        password: password,
      );
      return userCredential.user;
    } catch (e) {
      print("Error: $e");
      return null;
    }
  }


  // Register with Email & Password
  Future<User?> registerWithEmail(String
email, String password) async {
    try {
      UserCredential userCredential = await
_auth.createUserWithEmailAndPassword(
        email: email,
        password: password,
      );
      return userCredential.user;
    } catch (e) {
      print("Error: $e");
      return null;
    }
  }


  // Google Sign-In
  Future<User?> signInWithGoogle() async {
    try {
      final GoogleSignInAccount? googleUser
= await GoogleSignIn().signIn();
      if (googleUser == null) return null;


      final GoogleSignInAuthentication
googleAuth = await googleUser.authentication;
      final AuthCredential credential =
GoogleAuthProvider.credential(
        accessToken: googleAuth.accessToken,
        idToken: googleAuth.idToken,
      );


      UserCredential userCredential = await
_auth.signInWithCredential(credential); return
      userCredential.user;
    } catch (e) {
      print("Google Sign-In Error: $e"); return
      null;
    }
  }


  // Sign Out
  Future<void> signOut() async { await
    _auth.signOut();
    await GoogleSignIn().signOut();
  }


  // Get current user
```

```
  User? getCurrentUser() {                                    }
    return _auth.currentUser;                                 }
```

Login screen

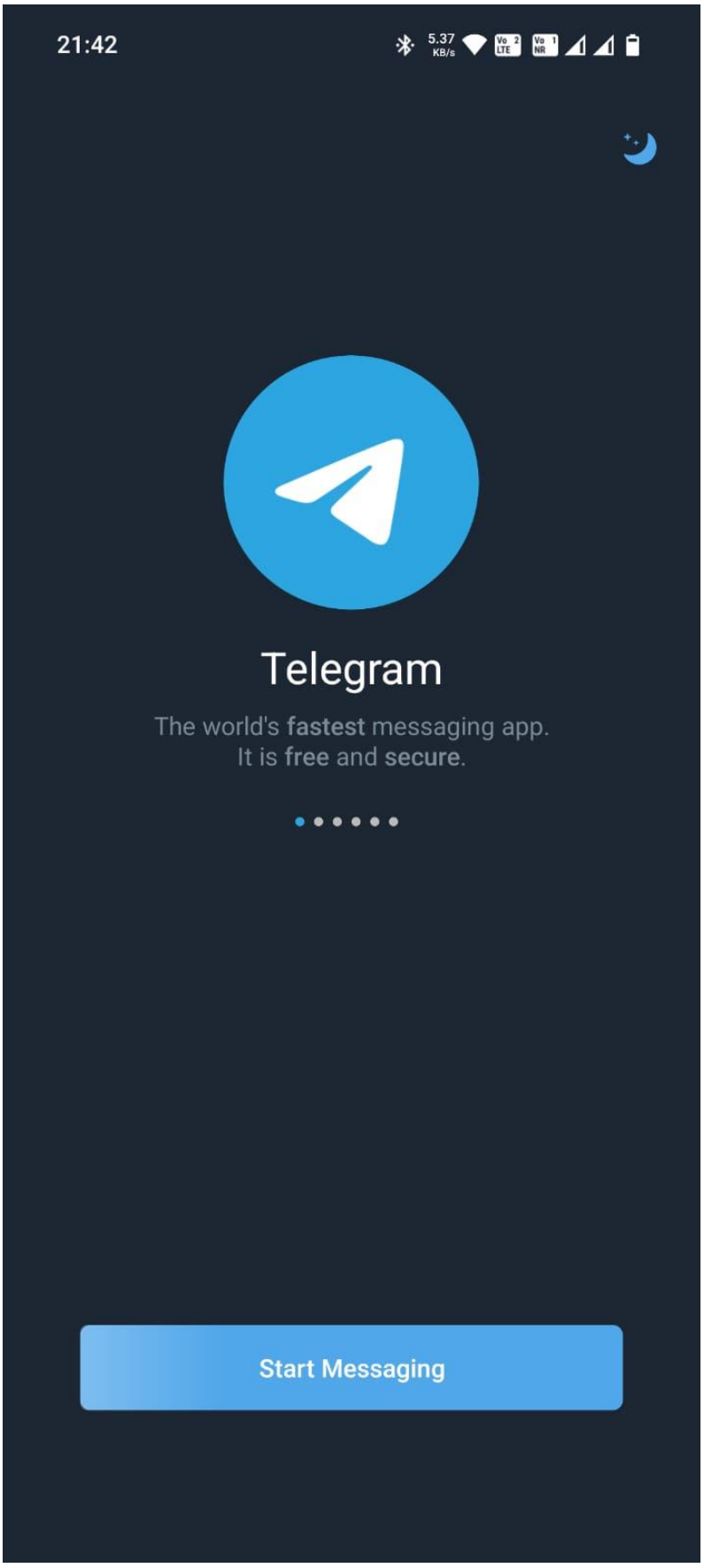
```
import      'package:flutter/material.dart';
import       '../services/auth_service.dart';
import 'home_screen.dart';

class LoginScreen extends StatefulWidget {
  @override
  _LoginScreenState createState() =>
_LoginScreenState();
}

class _LoginScreenState extends
State<LoginScreen> {
  final TextEditingController emailController =
TextEditingController();
  final TextEditingController passwordController
= TextEditingController();
  final AuthService _authService = AuthService();

  void _login() async {
    String email = emailController.text.trim(); String
    password =
passwordController.text.trim(); var
    user = await
_authService.signInWithEmail(email, password);
    if (user != null) {
      Navigator.pushReplacement(context,
MaterialPageRoute(builder: (context) =>
HomeScreen()));
    } else {

ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text("Login failed!")));
    }
  }

  void _googleSignIn() async { var
    user = await
_authService.signInWithGoogle(); if
    (user != null) {
      Navigator.pushReplacement(context,
MaterialPageRoute(builder: (context) =>
HomeScreen()));
    } else {

ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text("Google Sign-In failed!")));
    }
  }

  @override
  Widget build(BuildContext context) { return
    Scaffold(
      appBar: AppBar(title: Text('Rapido Login')),
      body: Padding(
        padding: EdgeInsets.all(16.0),
        child: Column(



          mainAxisAlignment:
MainAxisAlignment.center,
          children: [
            TextField(controller: emailController,
decoration: InputDecoration(labelText: 'Email')),
            TextField(controller: passwordController,
decoration: InputDecoration(labelText:
'Password'), obscureText: true),
            SizedBox(height: 20),
            ElevatedButton(onPressed: _login,
child: Text('Login')),
            SizedBox(height: 10),
            ElevatedButton(onPressed:
_googleSignIn, child: Text('Sign in with Google')),
          ],
        ),
      ),
    );
  }
}
```

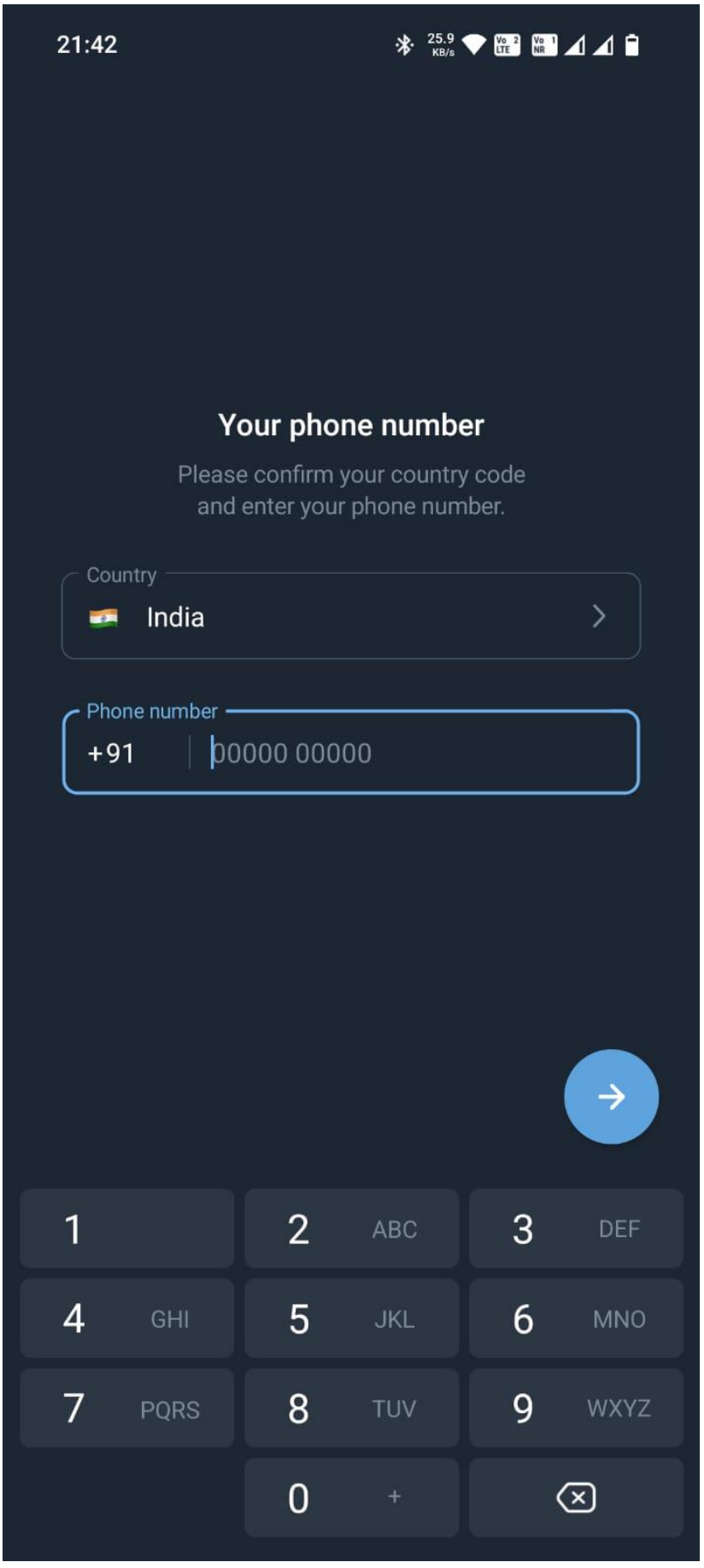
Home_Screen

```
import      'package:flutter/material.dart';
import       '../services/auth_service.dart';
import 'login_screen.dart';

class HomeScreen extends StatelessWidget { final
AuthService _authService = AuthService();

  void _logout(BuildContext context) async { await
    _authService.signOut();
    Navigator.pushReplacement(context,
MaterialPageRoute(builder: (context) =>
LoginScreen()));
  }

  @override
  Widget build(BuildContext context) { return
    Scaffold(
      appBar: AppBar(title: Text('Rapido
```

```
Home')),
      body: Center( child:
        Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text('Welcome to Rapido!'), SizedBox(height:
            20), ElevatedButton(onPressed: () =>
_logout(context), child: Text('Logout')),
          ],
        ),
```

OUTPUT:


21:42
5.37
KB/s
Telegram
The world's fastest messaging app.
It is free and secure.
Start Messaging


21:42
25.9 KB/s
Your phone number
Please confirm your country code and enter your phone number.
Country
India
Phone number
+91
00000 00000
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
0 +

Experiment No. 7

Swaraj Patil
D15A - 39

Aim: To write meta data of your Ecommerce PWA in a Web app manifest file to enable "add to homescreen feature".

Online Reference:
https://developer.mozilla.org/en-US/docs/Web/Manifest
https://www.geeksforgeeks.org/making-a-simple-pwa-under-5-minutes/

Theory:

Regular Web App
A regular web app is a website that is designed to be accessible on all mobile devices such that the content gets fit as per the device screen. It is designed using a web technology stack (HTML, CSS, JavaScript, Ruby, etc.) and operates via a browser. They offer various native-device features and functionalities. However, it entirely depends on the browser the user is using. In other words, it might be possible that you can access a native-device feature on Chrome but not on Safari or Mozilla Firefox because the browsers are incompatible with that feature.

Progressive Web App
Progressive Web App (PWA) is a regular web app, but some extras enable it to deliver an excellent user experience. It is a perfect blend of desktop and mobile application experience to give both platforms to the end-users.

Difference between PWAs vs. Regular Web Apps:
A Progressive Web is different and better than a Regular Web app with features like:

1. Native Experience
Though a PWA runs on web technologies (HTML, CSS, JavaScript) like a Regular web app, it gives user experience like a native mobile application. It can use most native device features, including push notifications, without relying on the browser or any other entity. It offers a seamless and integrated user experience that it is quite tough for one to differentiate between a PWA and a Native application by considering its look and feel.

2. Ease of Access
Unlike other mobile apps, PWAs do not demand longer download time and make memory space available for installing the applications. The PWAs can be shared and installed by a link, which cuts down the number of steps to install and use. These applications can easily keep an app icon on the user's home screen, making the app easily accessible to the users and helps the brands remain in the users' minds, and improving the chances of interaction.

3. Faster Services
PWAs can cache the data and serve the user with text stylesheets, images, and other web content even before the page loads completely. This lowers the waiting time for the end-users and helps the brands improve the user engagement and retention rate, which eventually adds value to their business.

4. Engaging Approach
As already shared, the PWAs can employ push notifications and other native device features more efficiently. Their interaction does not depend on the browser user uses. This eventually improves the chances of notifying the user regarding your services, offers, and other options related to your brand and keeping them hooked to your brand. In simpler words, PWAs let you maintain the user engagement and retention rate.

5. Updated Real-Time Data Access
Another plus point of PWAs is that these apps get updated on their own. They do not demand the end-users to go to the App Store or other such platforms to download the update and wait until installed.
In this app type, the web app developers can push the live update from the server, which reaches the apps residing on the user's devices automatically. Therefore, it is easier for the mobile app developer to provide the best of the updated functionalities and services to the end-users without forcing them to update their app.

6. Discoverable
PWAs reside in web browsers. This implies higher chances of optimizing them as per the Search Engine Optimization (SEO) criteria and improving the Google rankings like that in websites and other web apps.

7. Lower Development Cost
Progressive web apps can be installed on the user device like a native device, but it does not demand submission on an App Store. This makes it far more cost-effective than native mobile applications while offering the same set of functionalities.

Pros and cons of the Progressive Web App
The main features are:

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

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

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

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

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

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

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

- **Installable** — They allow the user to “save” the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to face all the steps and problems related to the use of the app store.

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

- **Offline** — Once more it is about putting the user before everything, avoiding the usual error message in case of weak or no connection. The PWA are based on two particularities: first of all the ‘skeleton’ of the app, which recalls the page structure, even if its contents do not respond and its elements include the header, the page layout, as well as an illustration that signals that the page is loading.

Weaknesses refer to:

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

Code:

manifest.json:-

```
{
   "name": "Xtrack",
   "short_name": "Xtrack",
   "start_url": "ani.html",
   "scope": "./",
   "icons": [
      {
        "src": "contract.png",
        "sizes": "192x192",
        "type": "image/png"
      },
      {
```

```
      "src": "contract.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "theme_color": "#ffd31d",
 "background_color": "#333",
 "display": "standalone"
}
```

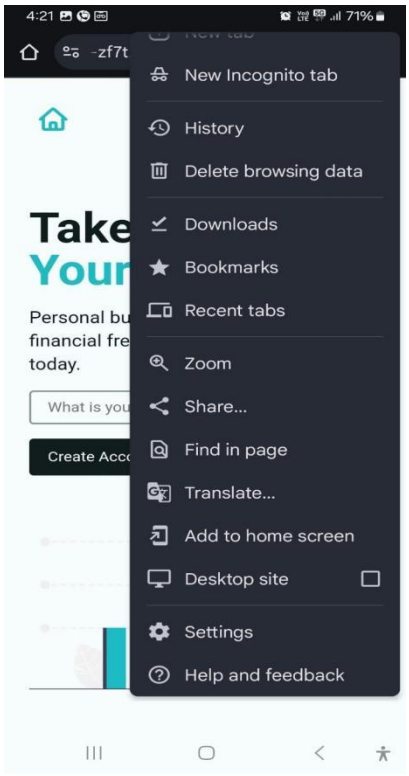
Add the link tag to link to the manifest.json file

```
<html>
  <head>
    <link rel="manifest" href="manifest.json">
    <script src="myscript.js"></script>
    <title>Xtrack</title>
    <style>
```

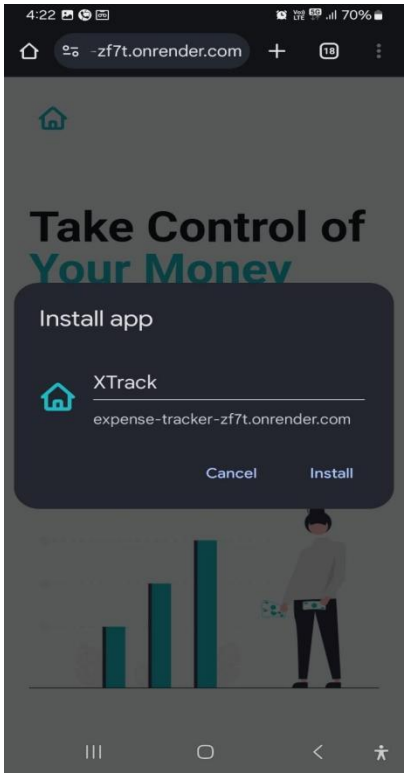
Output with Steps:

1. Go to google chrome and open your website link:

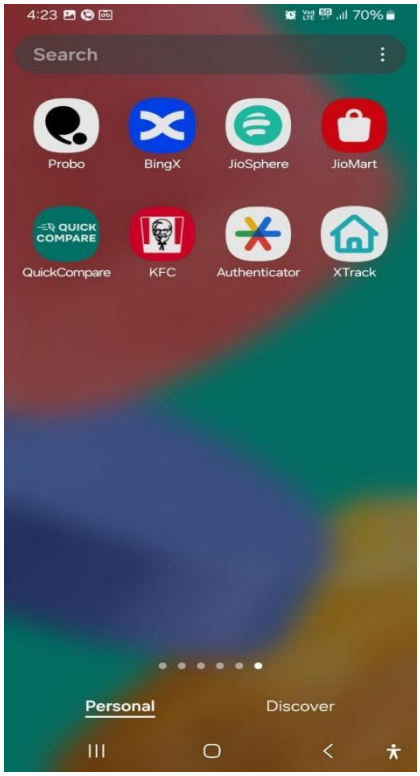
2. Then click on three dots at top right corner, find and click on add to home screen:



3. Then click on install:



4. Once installed you will see application in your mobile:



5. Final Output:

Conclusion:

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

Experiment No. 8

Swaraj Patil
D15A -39

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

Theory:

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

Things to note about Service Worker:

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

What can we do with Service Workers?

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

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

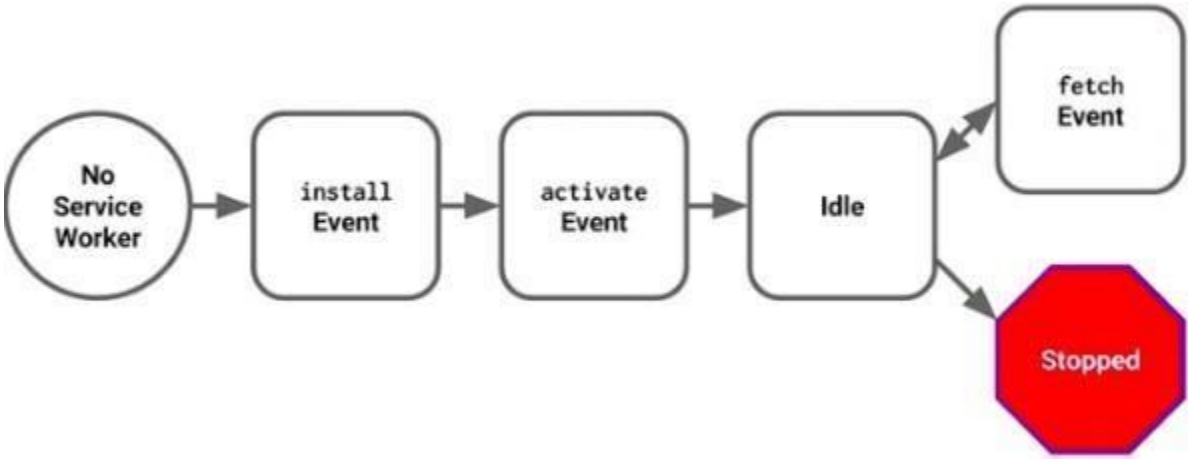
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can't we do with Service Workers?

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

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

Service Worker Cycle



A service worker goes through three steps in its life cycle:

- Registration
- Installation
- Activation

Registration
To install a service worker, you need to register it in your main JavaScript code.

Registration tells the browser where your service worker is located, and to start installing it in the background. Let's look at an example:

main.js

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

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

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

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

example: main.js

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

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

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

main.js

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

Installation

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

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

service-worker.js

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

Activation

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

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

service-worker.js

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

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

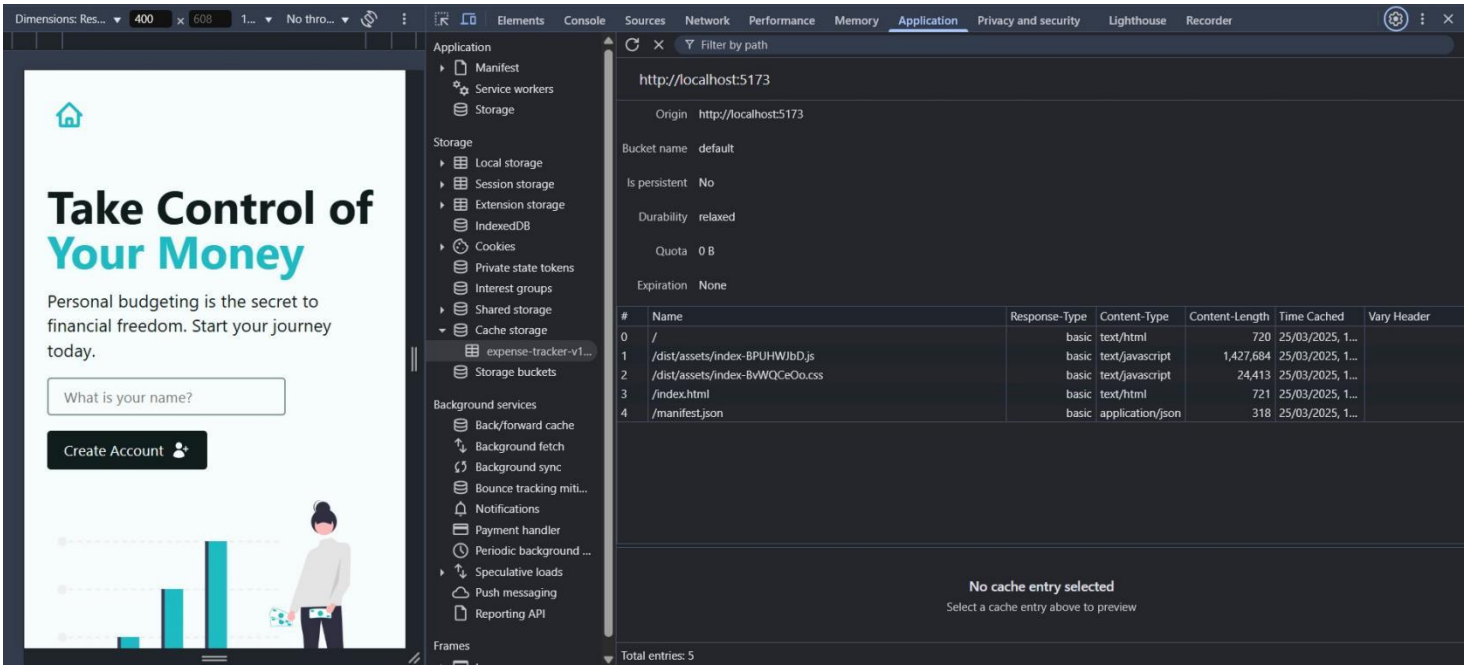
Code

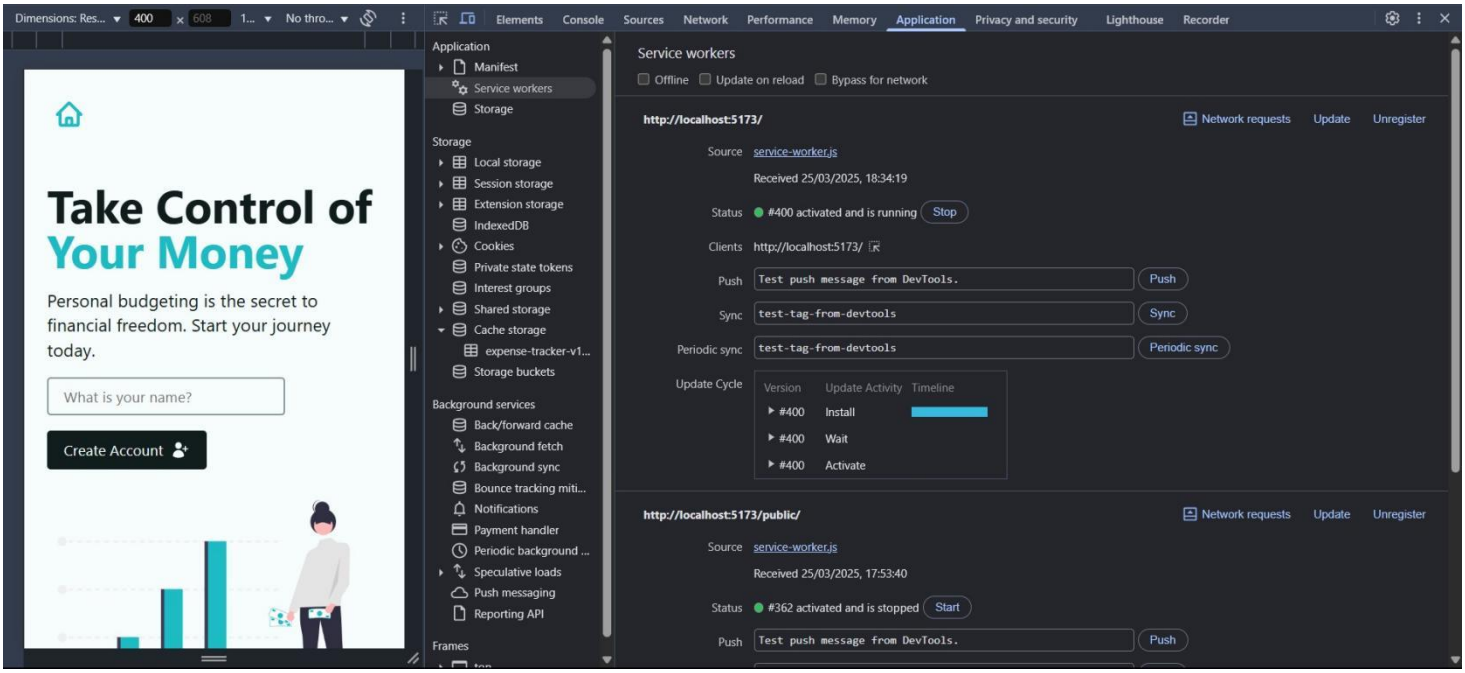
sw.js

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

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);
   });
};

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:


Take Control of
Your Money
Personal budgeting is the secret to financial freedom. Start your journey today.
What is your name?
Create Account
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
expense-tracker-v1...
Storage buckets
Background services
Back/forward cache
Background fetch
Background sync
Bounce tracking miti...
Notifications
Payment handler
Periodic background ...
Speculative loads
Push messaging
Reporting API
Frames
http://localhost:5173
Origin http://localhost:5173
Bucket name default
Is persistent No
Durability relaxed
Quota 0 B
Expiration None
Name Response-Type Content-Type Content-Length Time Cached Vary Header
0 / basic text/html 720 25/03/2025, 1...
1 /dist/assets/index-BPUHWJbD.js basic text/javascript 1,427,684 25/03/2025, 1...
2 /dist/assets/index-BvWQCeOo.css basic text/javascript 24,413 25/03/2025, 1...
3 /index.html basic text/html 721 25/03/2025, 1...
4 /manifest.json basic application/json 318 25/03/2025, 1...
No cache entry selected
Select a cache entry above to preview
Total entries: 5


Take Control of
Your Money
Personal budgeting is the secret to financial freedom. Start your journey today.
What is your name?
Create Account
Service workers
Offline
Update on reload
Bypass for network
http://localhost:5173/
Network requests
Update
Unregister
Source service-worker.js
Received 25/03/2025, 18:34:19
Status #400 activated and is running
Stop
Clients http://localhost:5173/
Push Test push message from DevTools.
Push
Sync test-tag-from-devtools
Sync
Periodic sync test-tag-from-devtools
Periodic sync
Update Cycle
Version Update Activity Timeline
#400 Install
#400 Wait
#400 Activate
http://localhost:5173/public/
Source service-worker.js
Received 25/03/2025, 17:53:40
Status #362 activated and is stopped
Start

Sample Code with Output

Index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/favicon.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="manifest" href="manifest.json" />
    <title>XTrack</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.jsx'
import './index.css'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)

// Service Worker Registration
if ('serviceWorker' in navigator) {
  window.addEventListener('load', () => {
    navigator.serviceWorker.register('public/service-worker.js') // Path to your service worker
file
      .then((registration) => {
        console.log('[Service Worker] Registered with scope:', registration.scope);

        // Check if the service worker is active
        if (registration.active) {
          console.log('[Service Worker] Active');
```

```
        }

        // Check if the service worker is installing
        if (registration.installing) {
          console.log('[Service Worker] Installing');
        }

        // Check if the service worker is waiting
        if (registration.waiting) {
          console.log('[Service Worker] Waiting');
        }

        // Listen for state changes
        registration.addEventListener('updatefound', () => {
          console.log('[Service Worker] Update found');
          const installingWorker = registration.installing;
          if (installingWorker) {
            installingWorker.addEventListener('statechange', () => {
              if (installingWorker.state === 'installed') {
                if (navigator.serviceWorker.controller) {
                  console.log('[Service Worker] New content is available and will be used when
all tabs for this page are closed.');
                  // You can prompt the user to reload here if needed
                } else {
                  console.log('[Service Worker] Content is cached for offline use.');
                }
              }
            });
          }
        });
      })
      .catch((error) => {
        console.error('[Service Worker] Registration failed:', error);
      });
  });
} else {
  console.warn('[Service Worker] Not supported in this browser.');
}
```

service-worker.js

```
// public/service-worker.js

const CACHE_NAME = 'expense-tracker-v1'; // Change this when you update the service worker
const urlsToCache = [
  '/', // Cache the root URL (index.html)
  '/index.html',
  '/manifest.json', // if you have one
  // Correct paths to your built assets
  'dist/assets/index-BPUHWJbD.js', // Replace with the actual path
  'dist/assets/index-BvWQCeOo.css',
];

// Install Event: Cache static assets
self.addEventListener('install', (event) => {
  console.log('[Service Worker] Install');
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => {
        console.log('[Service Worker] Caching app shell');
        return cache.addAll(urlsToCache);
      })
      .catch((error) => {
        console.error('[Service Worker] Caching failed:', error);
      })
  );
});

// Activate Event: Clean up old caches
self.addEventListener('activate', (event) => {
  console.log('[Service Worker] Activate');
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cacheName) => {
          if (cacheName !== CACHE_NAME) {
            console.log('[Service Worker] Deleting old cache:', cacheName);
            return caches.delete(cacheName);
          }
        })
      );
    })
```

```
  );
});

// Fetch Event: Serve from cache or network
self.addEventListener('fetch', (event) => {
  console.log('[Service Worker] Fetch', event.request.url);
  event.respondWith(
    caches.match(event.request)
      .then((response) => {
        // Cache hit - return response
        if (response) {
          console.log('[Service Worker] Found in cache:', event.request.url);
          return response;
        }
        // Not in cache - return fetch request
        console.log('[Service Worker] Not found in cache, fetching:', event.request.url);
        return fetch(event.request);
      })
  );
});
```

Conclusion: Successfully installed and activated the process of new service worker for pwa.

Experiment No. 9

Swaraj Patil
D15A - 39

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.

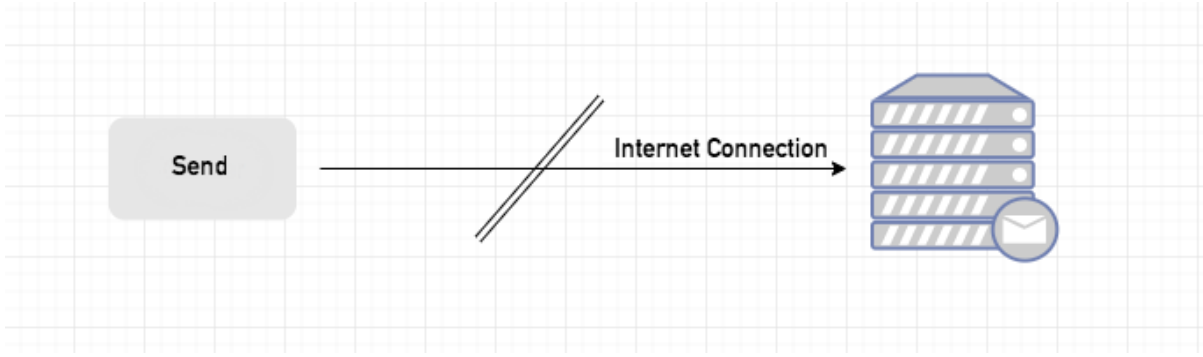
```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);

    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

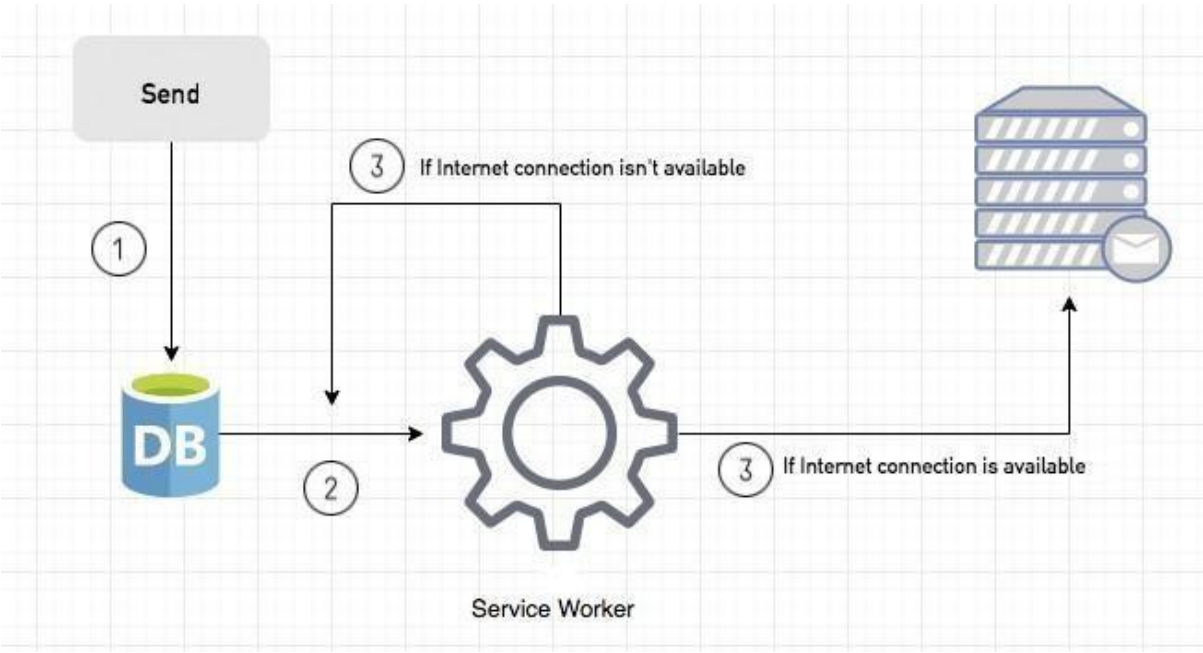
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are 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.

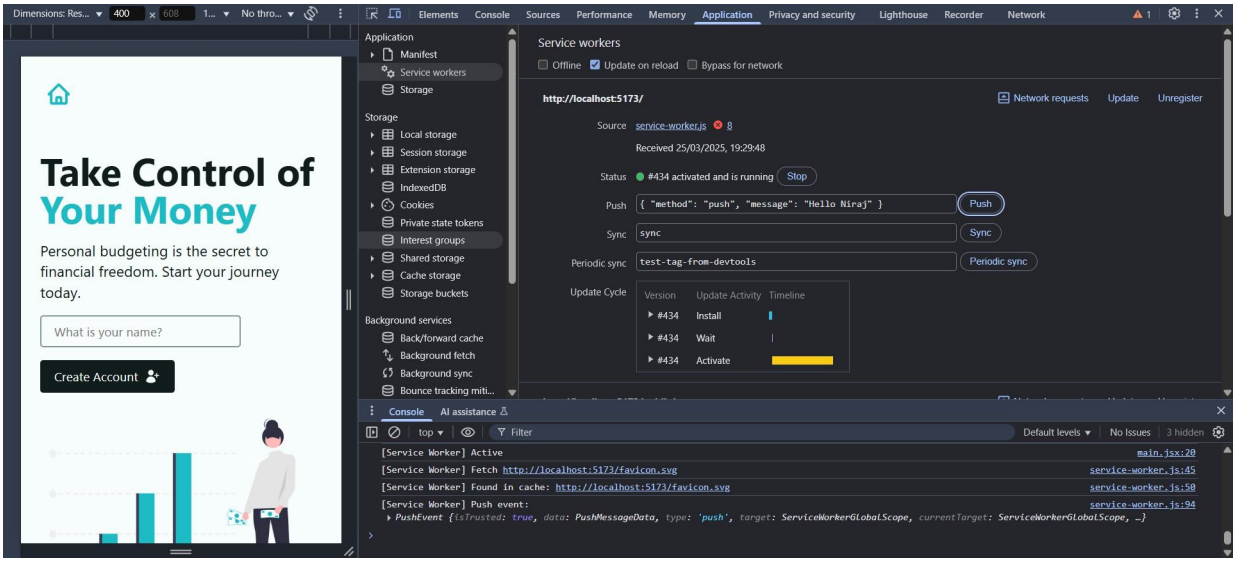
We can check in the following example.
“Notification.requestPermission();” is the necessary line to show notification to the user. If you don’t want to show any notification, you don’t need this line.

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:

service-worker.js

```
// public/service-worker.js


const CACHE_NAME = 'expense-tracker-v1'; // Change this when you update the service worker
const urlsToCache = [
  '/', // Cache the root URL (index.html)
  '/index.html',
  '/manifest.json', // if you have one
  // Correct paths to your built assets
  'dist/assets/index-BPUHWJbD.js', // Replace with the actual path
  'dist/assets/index-BvWQCeOo.css',
];


self.addEventListener('install', (event) => {
  console.log('[Service Worker] Install');
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => {
        console.log('[Service Worker] Caching app shell');
        return cache.addAll(urlsToCache);
      })
      .catch((error) => {
        console.error('[Service Worker] Caching failed:', error);
      })
  );
});


self.addEventListener('activate', (event) => {
  console.log('[Service Worker] Activate');
  event.waitUntil(
    caches.keys().then((cacheNames) => {
```

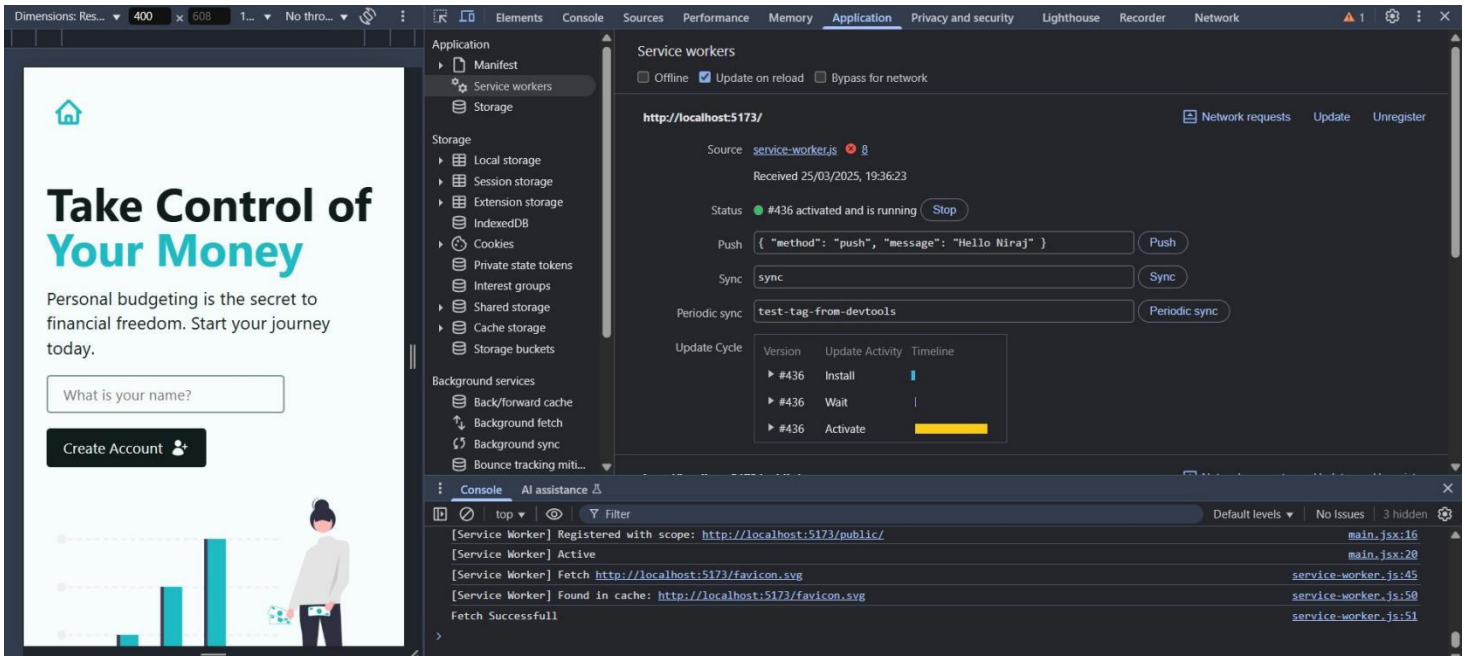
```
      return Promise.all(
        cacheNames.map((cacheName) => {
          if (cacheName !== CACHE_NAME) {
            console.log('[Service Worker] Deleting old cache:', cacheName);
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});

// Fetch Event: Cache-first strategy
self.addEventListener('fetch', (event) => {
  console.log('[Service Worker] Fetch', event.request.url);
  event.respondWith(
    caches.match(event.request)
      .then((response) => {
        if (response) {
          console.log('[Service Worker] Found in cache:', event.request.url);
          return response;
        }
        return fetch(event.request)
          .then((response) => {
            if (!response || response.status !== 200 || response.type !== 'basic') {
              return response;
            }
            const responseToCache = response.clone();
            caches.open(CACHE_NAME)
              .then((cache) => {
                cache.put(event.request, responseToCache);
              });
            return response;
```

```
          })
          .catch((error) => {
            console.error('[Service Worker] Fetch failed:', error);
            // Handle network errors (e.g., show offline message)
          });
      })
  );
});



// Sync Event: Handle background sync
self.addEventListener('sync', (event) => {
  console.log('[Service Worker] Sync event:', event.tag);
  if (event.tag === 'sync-expenses') {
    event.waitUntil(
      // Your background sync logic here (e.g., send expenses to server)
      syncExpenses()
        .then(() => {
          console.log('[Service Worker] Expenses synced successfully');
        })
        .catch((error) => {
          console.error('[Service Worker] Expense sync failed:', error);
          // Handle sync errors (e.g., retry later)
        })
    );
  }
});


// Push Event: Handle push notifications
self.addEventListener('push', function(event) {
  console.log('[Service Worker] Push event:', event);
  let data;
  try {
```

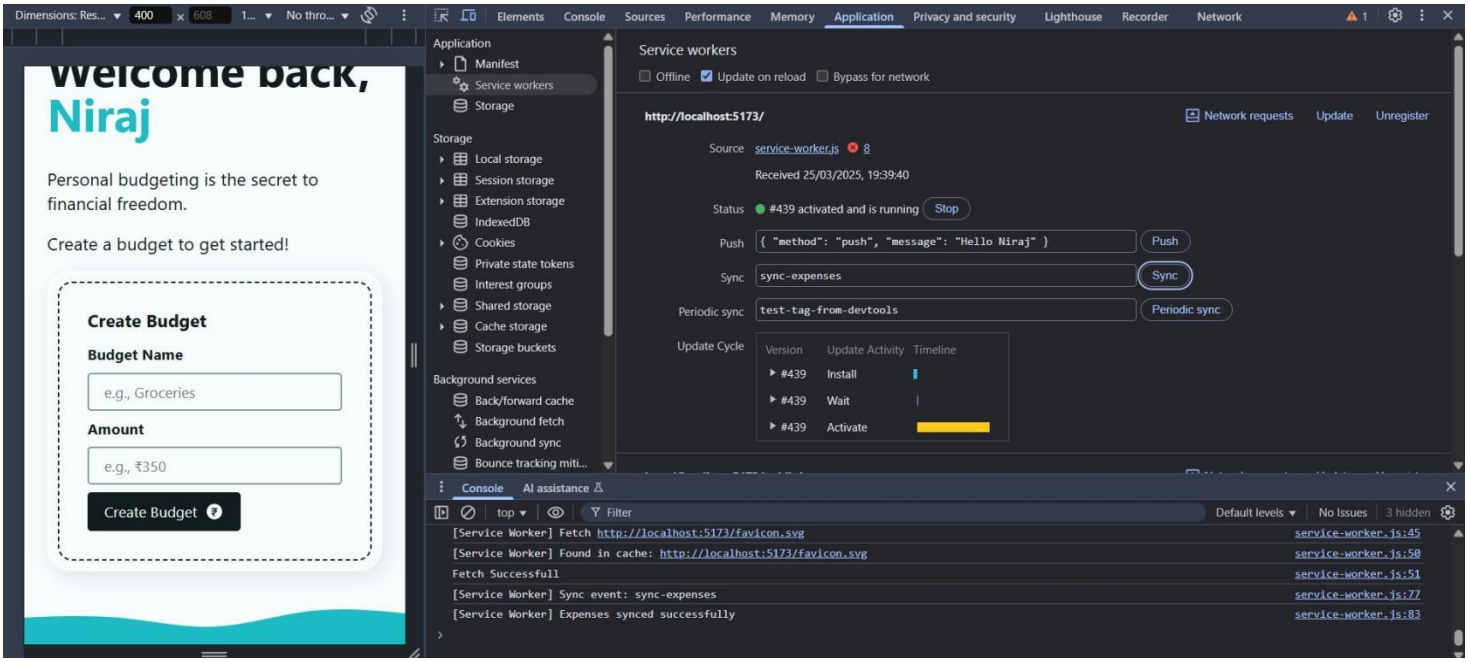
```
    data = event.data ? event.data.json() : {};
  } catch (error) {
    console.error('Error parsing push notification data:', error);
    data = { title: 'Error', message: 'Could not parse notification data.' };
  }

  if (Notification.permission === 'granted') {
    event.waitUntil(
      self.registration.showNotification(data.title || 'Xtrack', {
        body: data.message || 'No message provided.',
        icon: data.icon || '/your-notification-icon.png', // Provide a default icon
      })
    );
  } else {
    console.log('Notification permission not granted. Notification will not be shown.');
  }
});


//Helper function for sync event
async function syncExpenses() {
  // Replace this with your actual expense syncing logic
  // This example just simulates a delay
  return new Promise((resolve) => {
    setTimeout(resolve, 2000); // Simulate a 2-second delay
  });
}
```

Output:

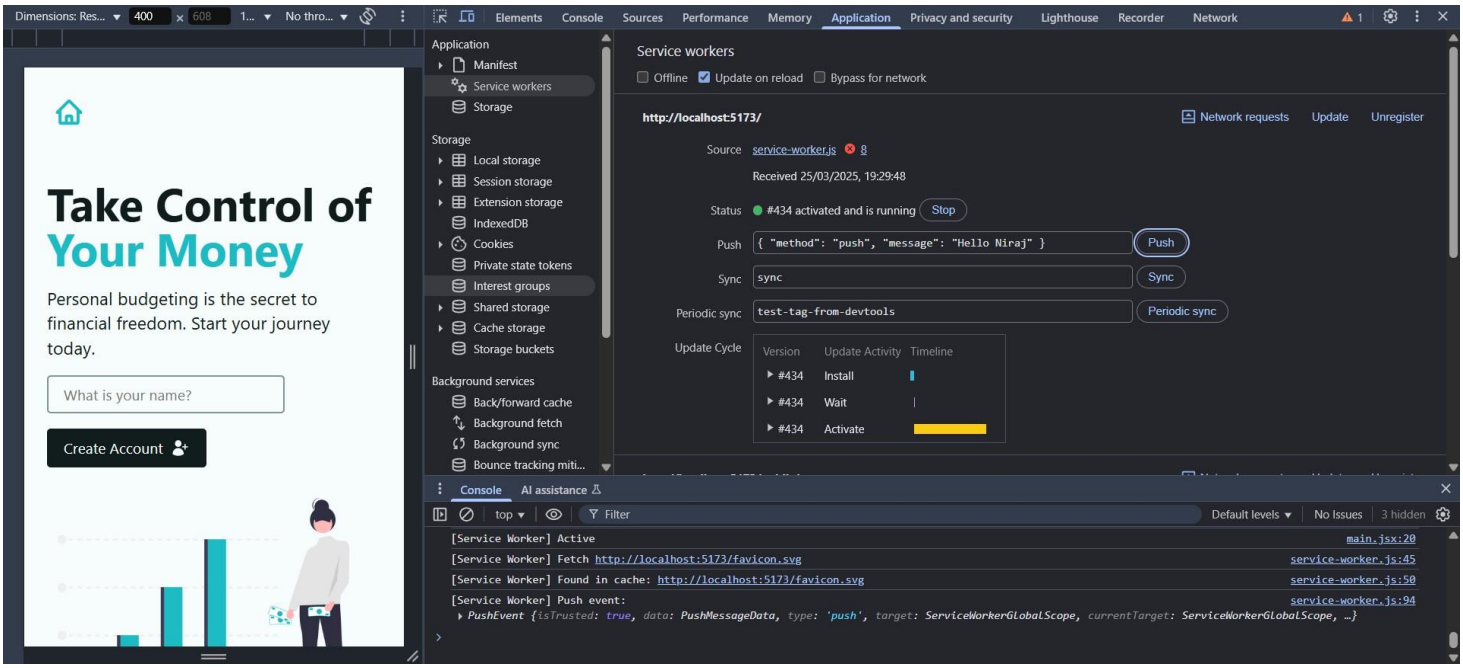
Fetchevent



Sync event



Push event



Conclusion: Successfully implemented Service worker events like fetch, sync and push for PWA.

Experiment No. 10
MAD & PWA Lab

Swaraj Patil

D15A - 39

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.

Pros

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

Cons

1. The code of your website will be public, unless you pay for a private repository.
2. Currently, there is no support for HTTPS for custom domains. It's probably coming soon though.

3. Although Jekyll is supported, plug-in support is rather spotty.

Firebase

The Realtime App Platform. Firebase is a cloud service designed to power real-time, collaborative applications. Simply add the Firebase library to your application to gain access to a shared data structure; any changes you make to that data are automatically synchronized with the Firebase cloud and with other clients within milliseconds.

Some of the features offered by Firebase are:

1. Add the Firebase library to your app and get access to a shared data structure. Any changes made to that data are automatically synchronized with the Firebase cloud and with other clients within milliseconds.
2. Firebase apps can be written entirely with client-side code, update in real-time out-of-the-box, interoperate well with existing services, scale automatically, and provide strong data security.
3. Data Accessibility- Data is stored as JSON in Firebase. Every piece of data has its own URL which can be used in Firebase's client libraries and as a REST endpoint. These URLs can also be entered into a browser to view the data and watch it update in real-time.

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

Pros

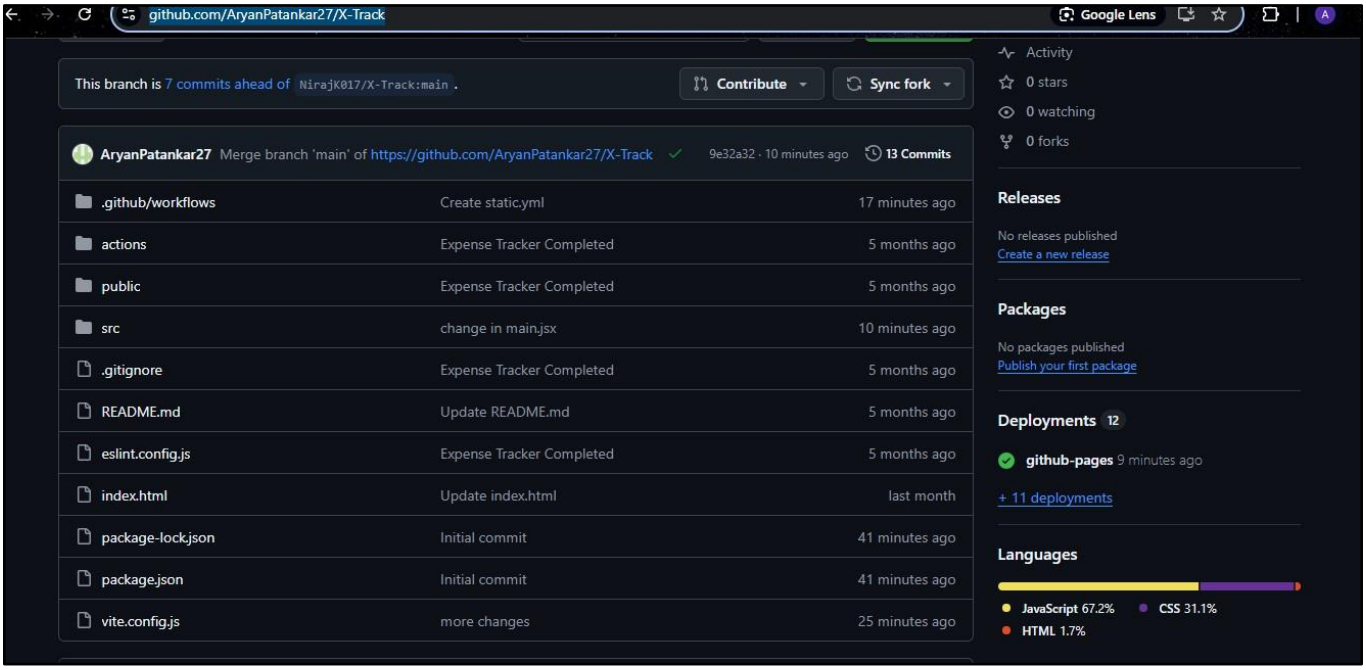
1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5. Support for HTTPS. A free certificate will be provisioned for your custom domain within 24 hours.

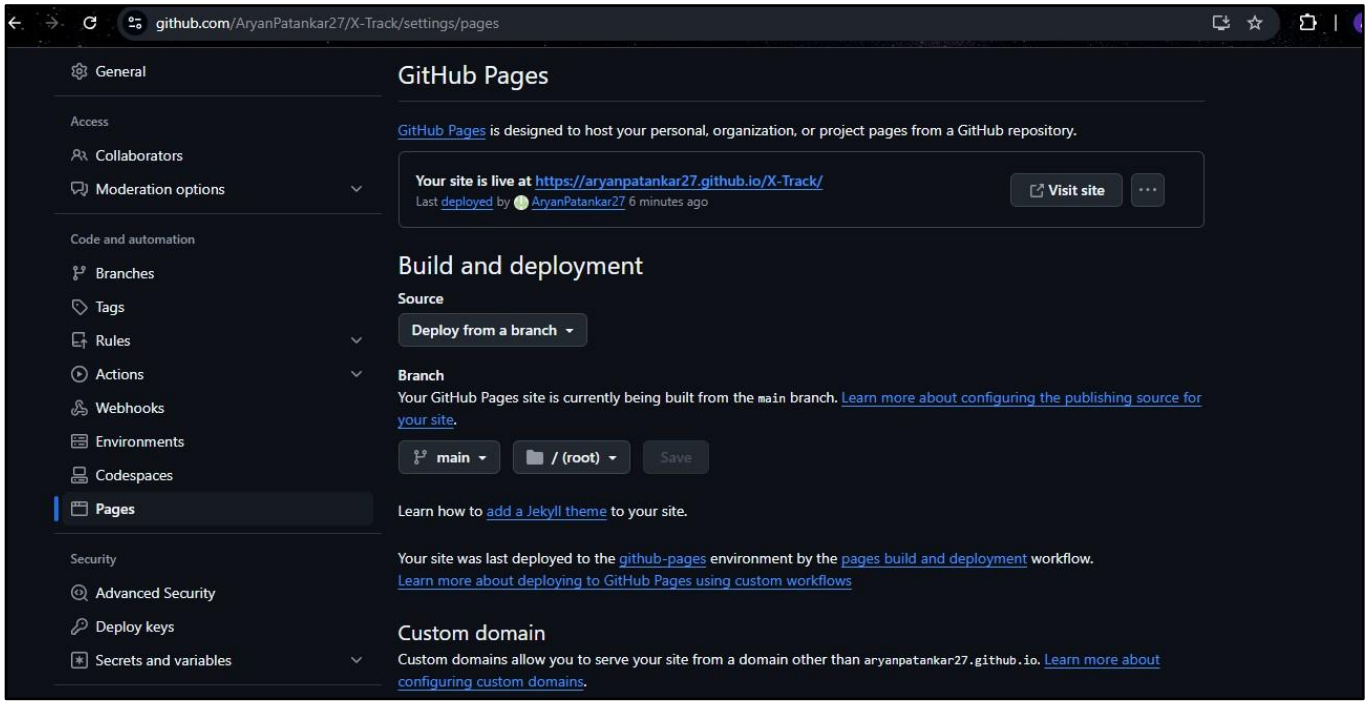
Cons

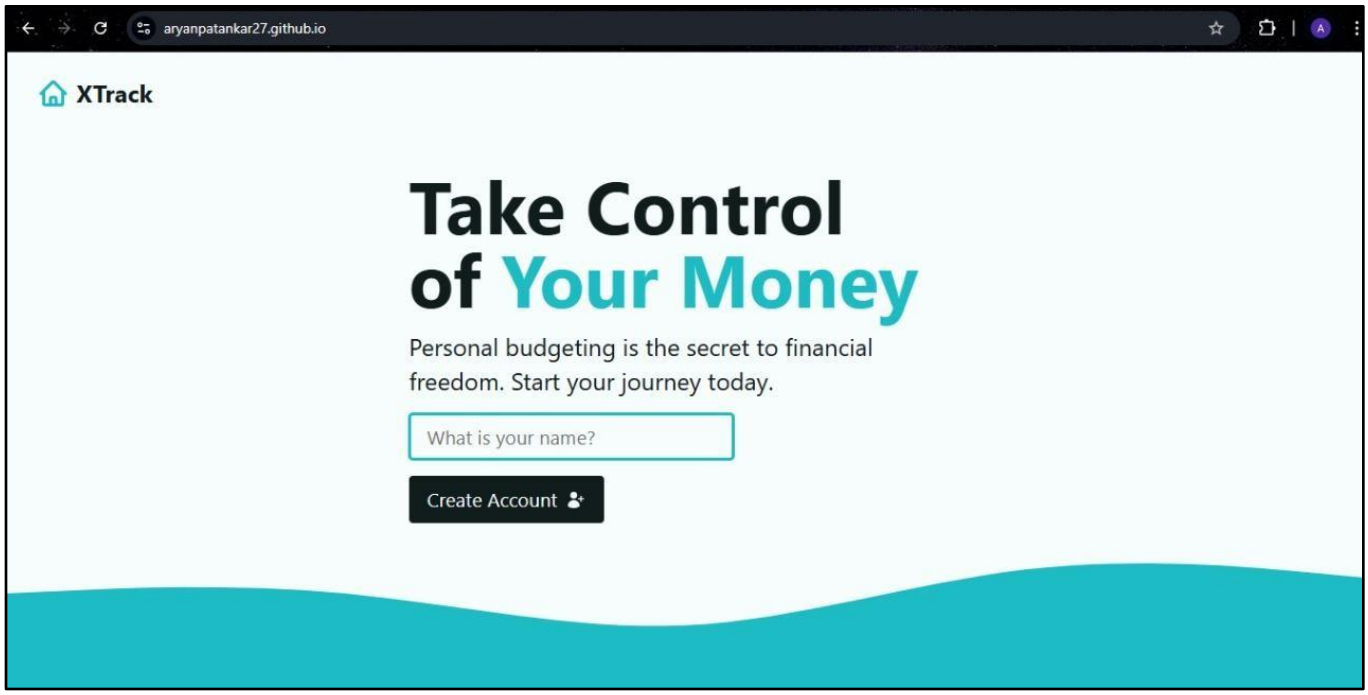
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

Link to our GitHub repository: https://github.com/AryanPatankar27/X-Track

Github Screenshot:






aryanpatankar27.github.io
XTrack
Take Control
of Your Money
Personal budgeting is the secret to financial freedom. Start your journey today.
What is your name?
Create Account

Experiment 11

Swaraj Patil
D15A - 39

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

Theory :

Google Lighthouse :

Google Lighthouse is a tool that lets you audit your web application based on a number of parameters including (but not limited to) performance, based on a number of metrics, mobile compatibility, Progressive Web App (PWA) implementations, etc. All you have to do is run it on a page or pass it a URL, sit back for a couple of minutes and get a very elaborate report, not much short of one that a professional auditor would have compiled in about a week. The best part is that you have to set up almost nothing to get started. Let's begin by looking at some of the top features and audit criteria used by Lighthouse.

Key Features and Audit Metrics

Google Lighthouse has the option of running the Audit for Desktop as well as mobile version of your page(s). The top metrics that will be measured in the Audit are:

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

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

Accessibility: As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.

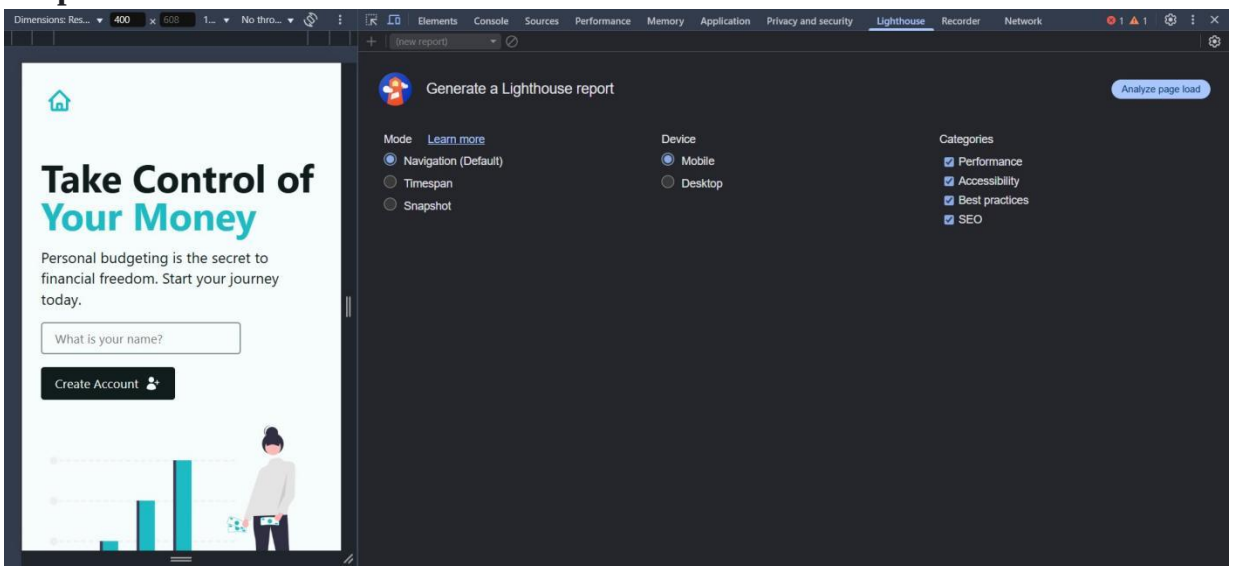
Best Practices: As any developer would know, there are a number of practices that have been deemed 'best' based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS Avoiding the use of deprecated code elements like tags, directives, libraries, etc. Password input with paste-into disabled
Geo-Location and cookie usage alerts on load, etc.

Improvement In Code:

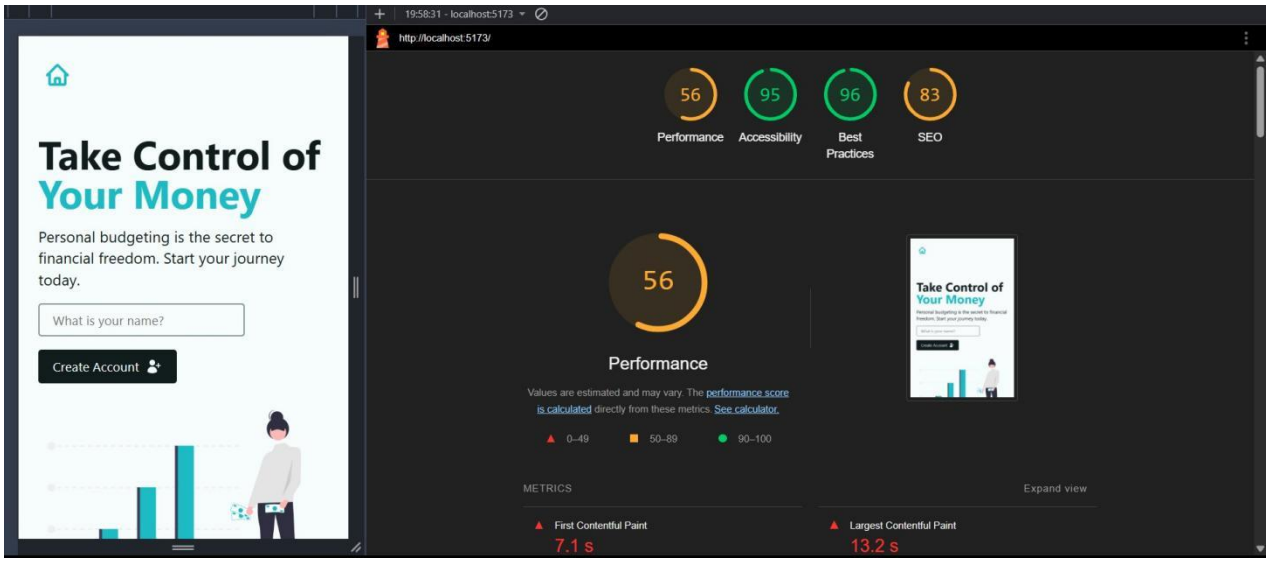
Index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/favicon.svg" />
     <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
     <meta name="description" content="A concise and accurate description
of your Expense Tracker app.  This will appear in search engine results.">
    <link rel="manifest" href="manifest.json" />
    <title>XTrack</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

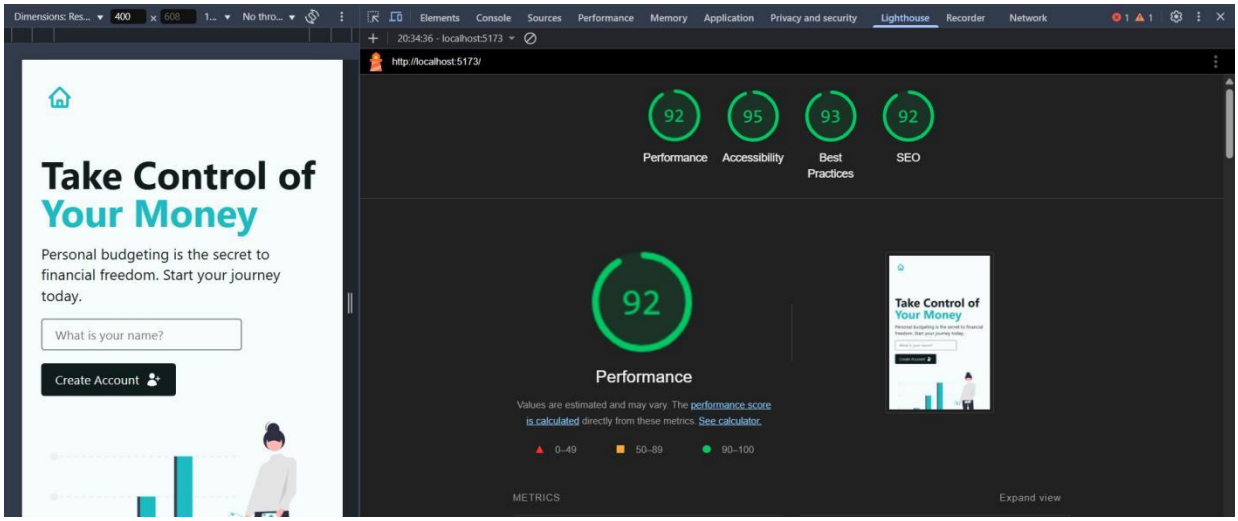
Output:



Before Optimizing Code:



After Optimizing Code:



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