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 **Abhinaya Danda** of **D15A** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Assistant

Subject Teacher

Mrs. Kajal Jewani

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Name of the Course : MAD & PWA Lab

Course Code : ITL604

Project Title: Gmail Clone **Roll No. 12**

Year/Sem/Class **:** D15A/D15B **A.Y.:** 23-24

Faculty Incharge : Mrs. Kajal Jewani.

Lab Teachers **:** Mrs. Kajal Jewani.

Email **:** kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: Gmail Clone **Roll No. 12**

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

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1			15
2.	To design Flutter UI by including common widgets.	LO2			15
3.	To include icons, images, fonts in Flutter app	LO2			15
4.	To create an interactive Form using form widget	LO2			15
5.	To apply navigation, routing and gestures in Flutter App	LO2			15
6.	To Connect Flutter UI with fireBase database	LO3			15
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4			15
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			15
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5			15
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5			15
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6			15
12.	Assignment-1	LO1,LO2 ,LO3			5
13.	Assignment-2	LO4,LO5 ,LO6			4

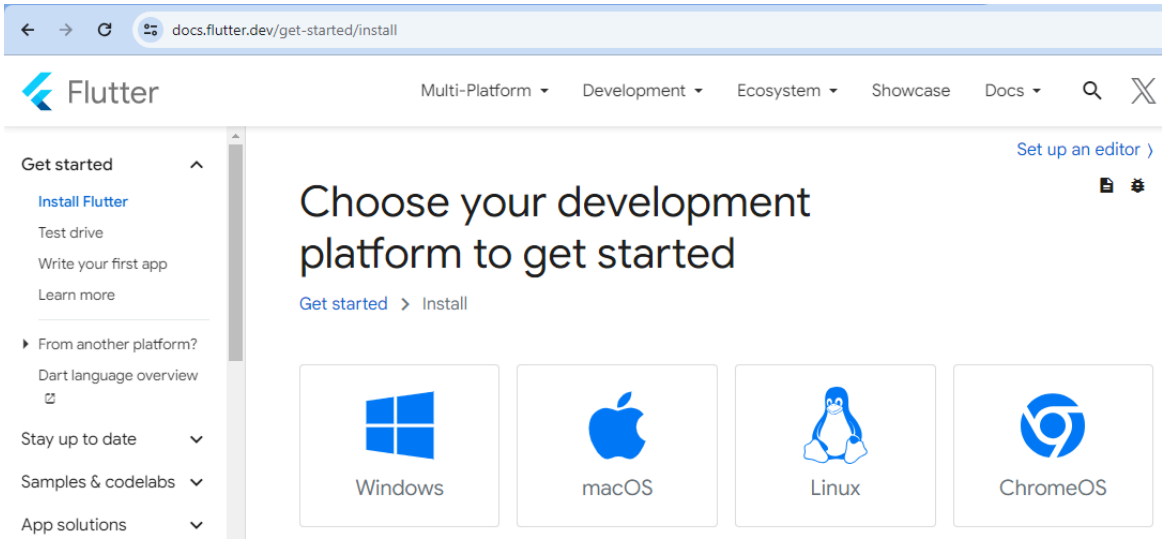
MAD & PWA Lab
Journal

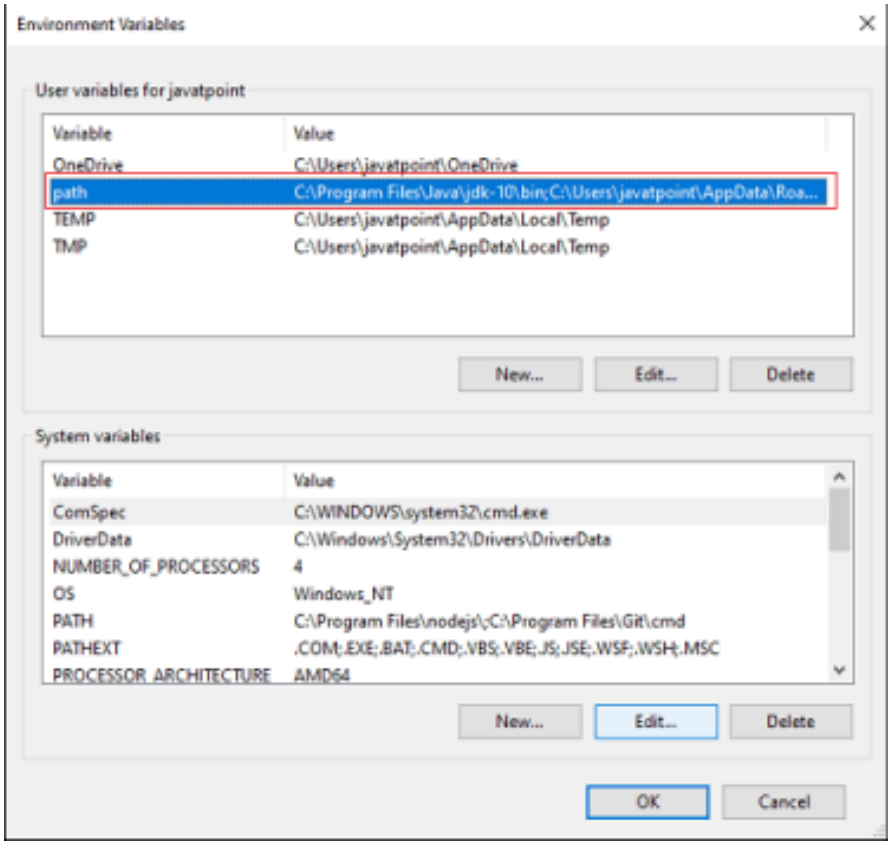
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	12
Name	Abhinaya Danda
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	15

ABHINAYA DANDA
D15A Roll No:12

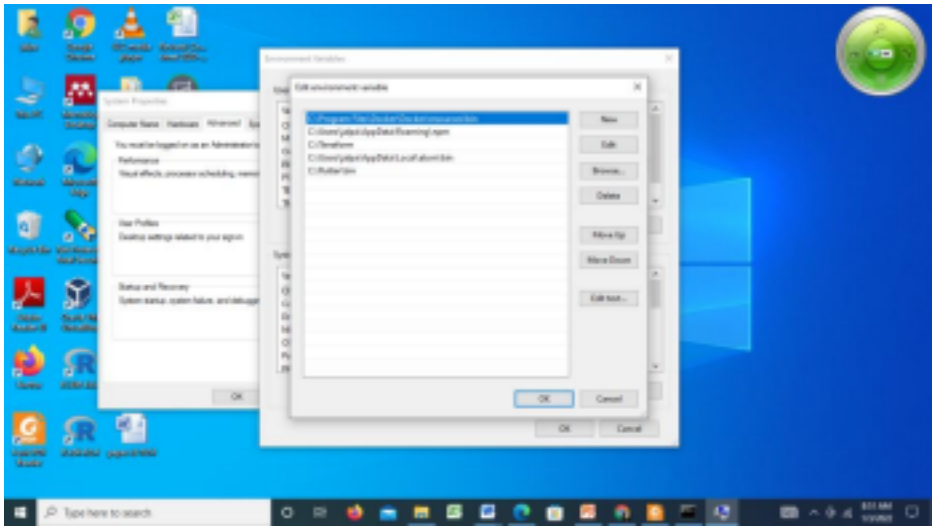
EXP 1: Installation and Configuration of Flutter Environment.

Visit **https://docs.flutter.dev/get-started/install**





Project Title: Gmail Clone **Roll No. 12**

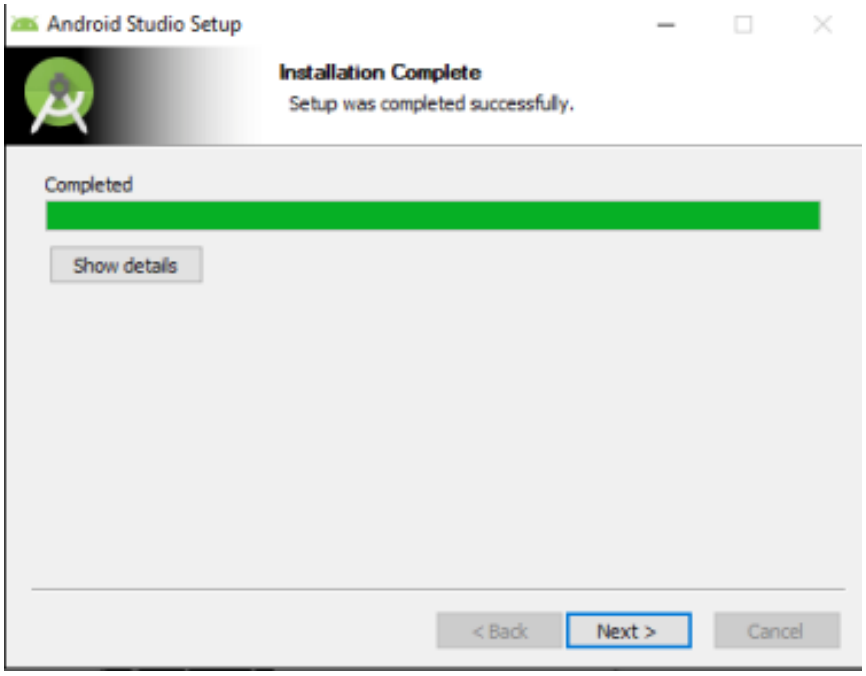


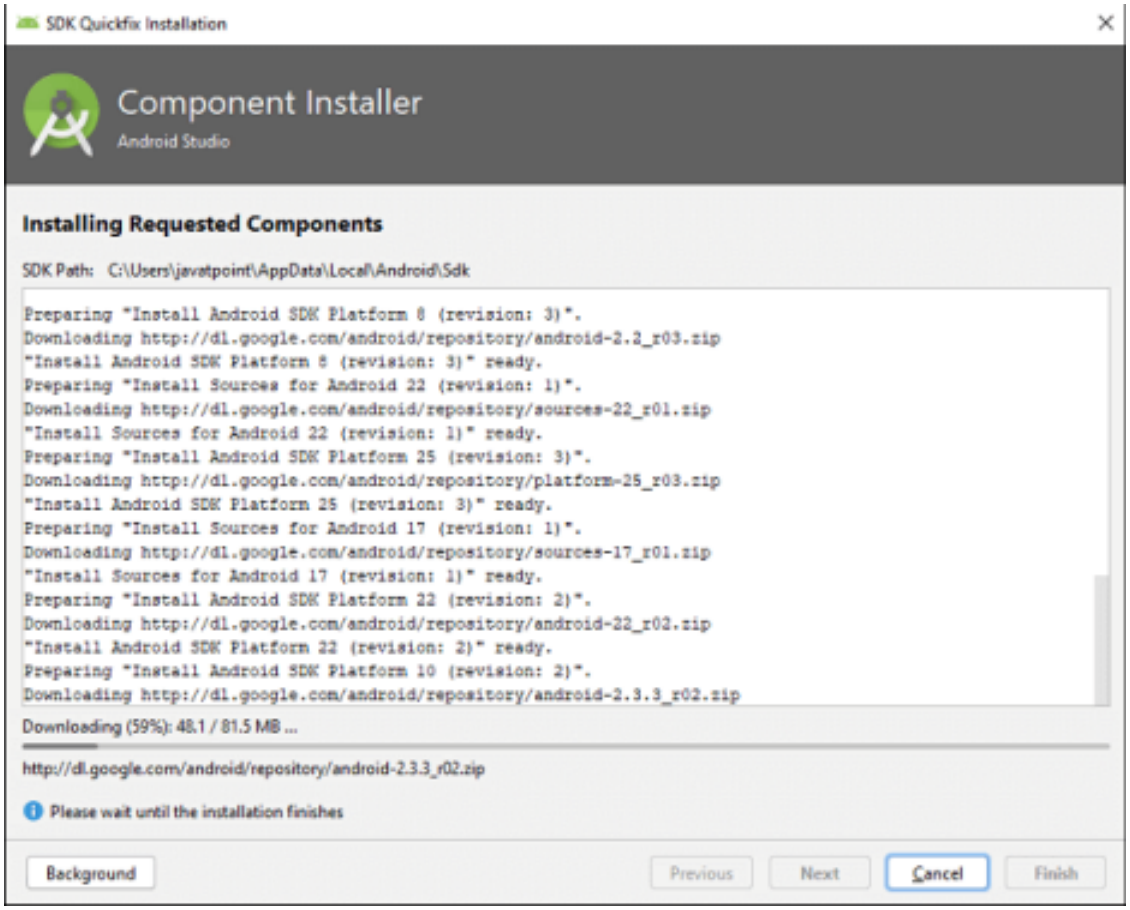
```
C:\>flutter --version
Flutter 3.16.7 • channel stable • https://github.com/flutter/flutter.git
Framework • revision ef1af02aea (4 days ago) • 2024-01-11 15:19:26 -0600
Engine • revision 4a585b7929
Tools • Dart 3.2.4 • DevTools 2.28.5

C:\>
```

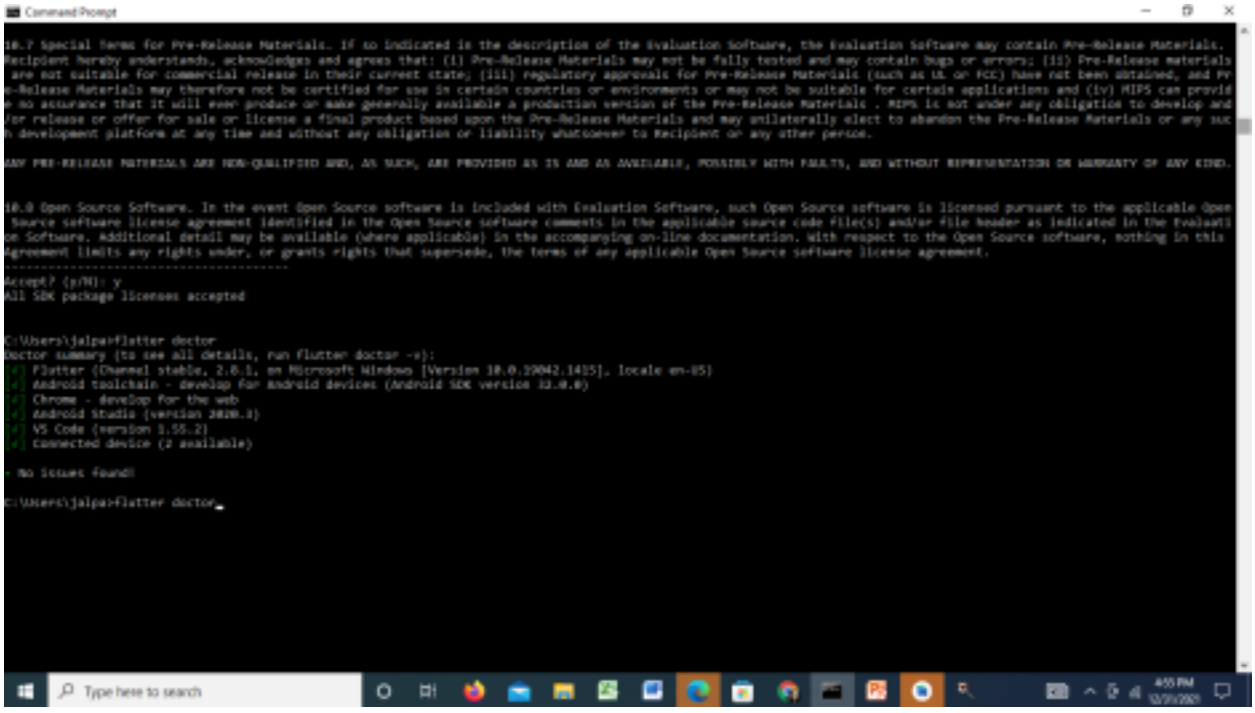
C:\Windows\System32\cmd.exe - flutter --version - flutter doctor - flutter create first_app

```
C:\>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.16.7, on Microsoft Windows [Version 10.0.19045.3448], locale en-IN)
[√] Windows Version (Installed version of Windows is version 10 or higher)
[!] Android toolchain - develop for Android devices (Android SDK version 34.0.0)
    X cmdline-tools component is missing
      Run `path/to/sdkmanager --install "cmdline-tools;latest"`
      See https://developer.android.com/studio/command-line for more details.
    X Android license status unknown.
      Run `flutter doctor --android-licenses` to accept the SDK licenses.
      See https://flutter.dev/docs/get-started/install/windows#android-setup for more details.
[√] Chrome - develop for the web
[X] Visual Studio - develop Windows apps
    X Visual Studio not installed; this is necessary to develop Windows apps.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++" workload, including all of its default components
[√] Android Studio (version 2023.1)
[√] VS Code (version 1.81.1)
[√] Connected device (3 available)
[√] Network resources

! Doctor found issues in 2 categories.
```

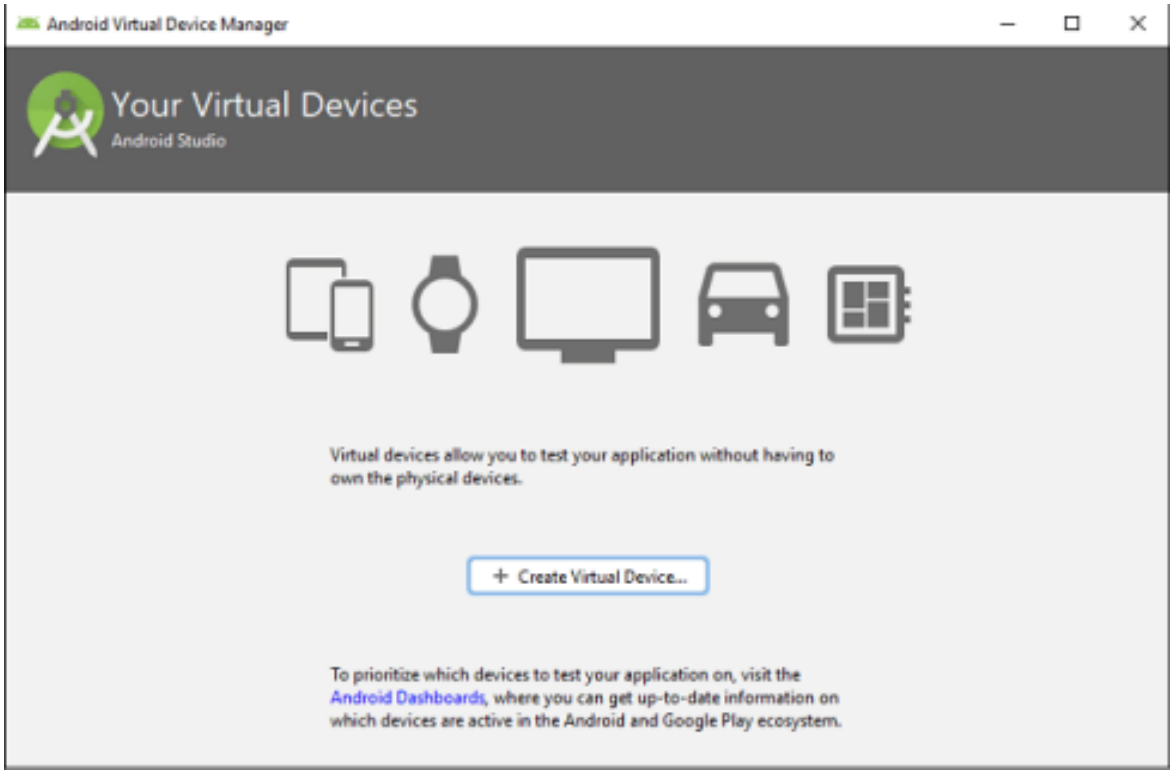
Download the latest **Android Studio** executable or zip file from the official site.

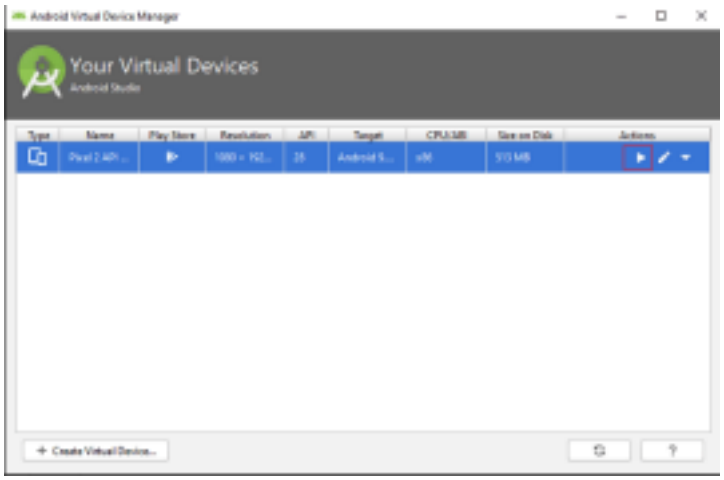






CreateVirtual Device


Android Virtual Device Manager
Your Virtual Devices
Android Studio
Virtual devices allow you to test your application without having to own the physical devices.
+ Create Virtual Device...
To prioritize which devices to test your application on, visit the Android Dashboards, where you can get up-to-date information on which devices are active in the Android and Google Play ecosystem.


Your Virtual Devices

Project Title: Gmail Clone **Roll No. 12**

```
C:\>flutter create first_app
Creating project C:\first_app...
Resolving dependencies in C:\first_app... (48:05.7s)Terminate batch job (Y/N)? y
```

Creating Hello world app

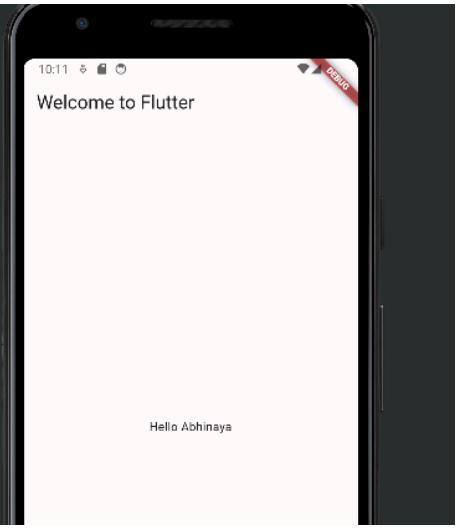
CODE:

```
import 'package:flutter/material.dart';
void main() {
 runApp(const MyApp());
 }
class MyApp extends StatelessWidget {
 const MyApp({Key? key}) : super(key: key);
 @override
 Widget build(BuildContext context) {
 return MaterialApp(
 title: 'Welcome to Flutter',
 home: Scaffold(
 appBar: AppBar(
 title: const Text('Welcome to Flutter'),
 ),
 body: const Center(
 child: Text('Hello Abhinaya'),
 ),
 ),
```

```
        );
      }
    }
```

OUTPUT:


```
}
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ),  // AppBar
        body: const Center(
          child: Text('Hello Abhinaya'),
        ),  // Center
      ),  // Scaffold
    );  // MaterialApp
  }
}
```



MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	12
Name	Abhinaya Danda
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

ABHINAYA DANDA
D15A Roll No:12

Experiment 2: Exploring Flutter Widgets

Aim:
To design Flutter UI by including common widgets.

Theory:

In Flutter, widgets are the building blocks of the UI. Widgets can be either stateless or stateful and can be combined to create complex UIs.

- **Flutter Scaffold:** Scaffold is a basic structure of the visual interface of an app. It provides functionality like an app bar, a drawer, and a bottom navigation bar.
- **Flutter Container:** Container is a box model that allows you to create a box with specific dimensions, padding, margin, and decoration.
- **Flutter Row & Column:** Row and Column are flex widgets used for arranging child widgets horizontally (Row) or vertically (Column).
- **Flutter Text:** Text widget is used for displaying a short piece of text. It supports styling and formatting.
- **Flutter TextField:** TextField is an input widget that allows users to enter text.
- **Flutter Buttons:** Flutter provides various button widgets like ElevatedButton, TextButton, and OutlinedButton for user interaction.
- **Flutter Stack:** Stack is a widget that allows you to overlay widgets on top of each other.
- **Flutter Forms:** Flutter forms are used to collect user input. They often involve using TextFields and validation.
- **Flutter AlertDialog:** AlertDialog is a popup dialog that displays important information or asks the user for input.
- **Flutter Icons:** Flutter comes with a set of customizable icons that can be used in your app.
- **Flutter Images:** Flutter supports the loading and display of images using the Image widget.
- **Flutter Card:** Card is a material design card. It's a container with rounded corners and elevation.
- **Flutter Tabbar:** TabBar is a widget that displays a horizontal row of tabs.
- **Flutter Drawer:** Drawer is a slide-in menu that is typically used for navigation.
- **Flutter Lists:** Lists are used to display a scrolling list of widgets.
- **Flutter GridView:** GridView is a scrollable grid of widgets.
- **Flutter Toast:** Toast is a temporary notification that appears at the bottom of the screen.
- **Flutter Checkbox:** Checkbox is a UI element that allows users to toggle between two states.
- **Flutter Radio Button:** Radio buttons allow users to select one option from a set.
- **Flutter Progress Bar:** Progress Bar indicates the progress of an ongoing task.
- **Flutter Snackbar:** Snackbar provides lightweight feedback about an operation.
- **Flutter Tooltip:** Tooltip provides additional information when the user hovers over a widget.
- **Flutter Slider:** Slider allows users to select a value from a range.

- **Flutter Switch:** Switch is a UI element for toggling between two states.
- **Flutter Charts:** Flutter supports various charting libraries for visualizing data.
- **Bottom Navigation Bar:** A bar at the bottom of the screen for navigation.
- **Flutter Themes:** Themes define the colors, fonts, and styles used in an app.
- **Flutter Table:** Table is used to create a two-dimensional array of widgets.
- **Flutter Calendar:** Calendar widgets help in displaying and selecting dates.
- **Flutter Animation:** Flutter supports various animation widgets and APIs for creating smooth animations.

Syntax:

Flutter Scaffold:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('My App'),
        ),
        body: Center(
          child: Text('Hello, World!'),
        ),
      ),
    );
  }
}
```

Flutter Container:

```
Container(
  width: 100.0,
  height: 100.0,
  color: Colors.blue,
  child: Text('Container'),
)
```

Flutter Row & Column:

```
Row(
  children: [
```

```
    Text('Item 1'),
    Text('Item 2'),
  ],
)

Column(
  children: [
    Text('Item 1'),
    Text('Item 2'),
  ],
)
```

Flutter Text:

```
Text(
  'Hello, World!',
  style: TextStyle(fontSize: 20.0),
)
```

Flutter Buttons:

```
ElevatedButton(
  onPressed: () {
    // Handle button press
  },
  child: Text('Elevated Button'),
)

TextButton(
  onPressed: () {
    // Handle button press
  },
  child: Text('Text Button'),
)
```

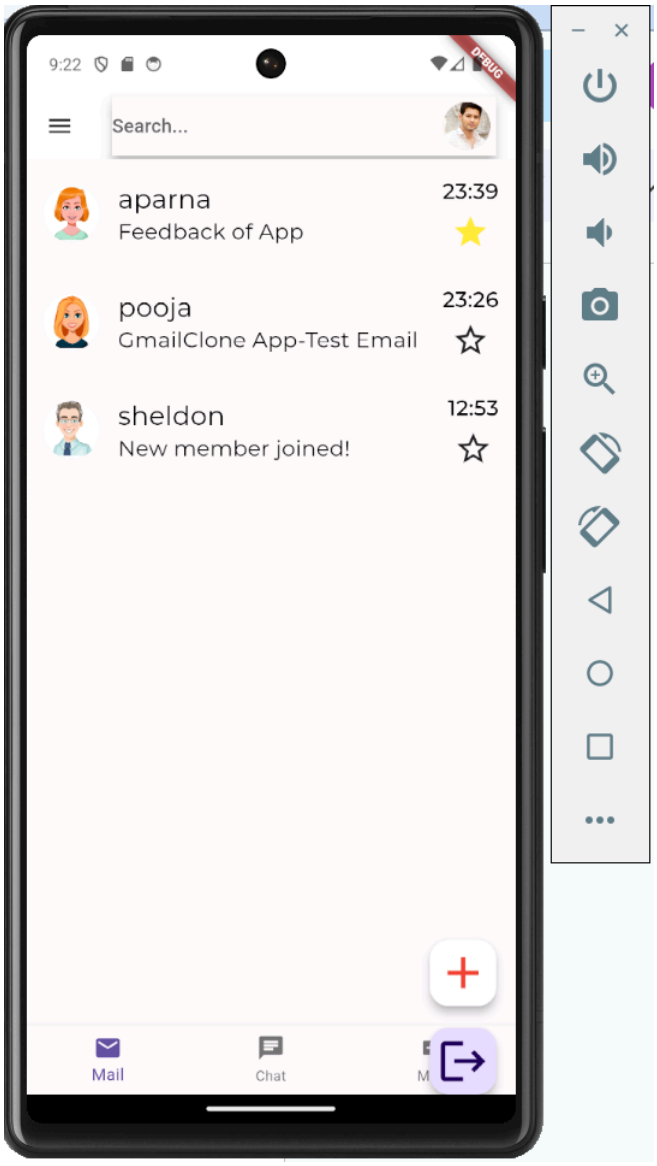
Code and output:

```
import 'package:flutter/material.dart';
import 'package:gmail_clone/addmail.dart';
import 'package:gmail_clone/variables.dart';

class Mails extends StatefulWidget {
  const Mails({super.key});

  @override
  State<Mails> createState() => _MailsState();
}

class _MailsState extends State<Mails> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.white,
        title: Material(
          elevation: 8.0,
          child: TextFormField(
            decoration: InputDecoration(
              hintText: 'Search...',
              border: InputBorder.none,
              icon: Container(
                margin: EdgeInsets.only(left: 5),
                child: Icon(Icons.dehaze),
              ),
              suffixIcon: Container(
                margin: EdgeInsets.all(5),
                child: CircleAvatar(
                  backgroundImage: NetworkImage(exampleimage),
                ),
              ),
            ),
          ),
        ),
      ),
      floatingActionButton: FloatingActionButton(
        backgroundColor: Colors.white,
        onPressed: ()=>Navigator.push(
          context,MaterialPageRoute(builder: (context)=>AddMail())),
        child: Icon(
          Icons.add,
          size: 45,
          color: Colors.red,
        ),
      ),
```

```
//cards for mails,loop through this list
body: ListView.builder(
   itemCount: 8,
   itemBuilder: (BuildContext context, int index) {
    return InkWell(
     onTap: () {},
     child: Padding(
      padding: EdgeInsets.all(14),
      child: Row(
       mainAxisAlignment: MainAxisAlignment.spaceBetween,
       children: <Widget>[
        Row(
         children: [
          CircleAvatar(
           backgroundImage: NetworkImage(exampleimage),
           radius: 24,
          ),
          SizedBox(
           width: 15.0,
          ),
          Column(
           crossAxisAlignment: CrossAxisAlignment.start,
           children: <Widget>[
            Text(
             'Abhinaya',
             style: mystyle(22, Colors.black, FontWeight.w700),
            ),
            Text(
             'Hello,how are u?',
             overflow: TextOverflow.ellipsis,
             style: mystyle(18, Colors.black, FontWeight.w600),
            )
           ],
          ),
         ],
        ),
        SizedBox(
         width: 5.0,
        ),
        Column(children: <Widget>[
         Text(
          '12:30',
          style: mystyle(18, Colors.black, FontWeight.w500),
         ),
         SizedBox(
          height: 5.0,
         ),
         InkWell(
            onTap: () {},
```

```
                    child: Icon(Icons.star_border, size: 32))
                ])
              ],
            ),
          ),
        );
      }),
    );
  }
}
```

Output:



Conclusion:

Understood the use of different widgets in flutter and implemented it in my gmail clone project.

****

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	12
Name	Abhinaya Danda
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Project Title: Gmail Clone **Roll No. 12**

ABHINAYA DANDA

D15A Roll no.12

Experiment 3

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

Theory:

Flutter widget are of two categories:

Visible (Output and Input)
Invisible (Layout and Control)
Visible widget
The visible widgets are related to the user input and output data. Some of the important types of this widget are:
1. Text
A Text widget holds some text to display on the screen. We can align the text widget by using textAlign property, and style property allow the customization of Text that includes font, font weight, font style, letter spacing, color, and many more. We can use it as like below code snippets.

```
new Text(
'Hello, ALL!',
textAlign: TextAlign.center,
style: new TextStyle(fontWeight: FontWeight.bold),
)
```

2. Button
This widget allows you to perform some action on click. Flutter does not allow you to use the Button widget directly; instead, it uses a type of buttons like a FlatButton and a RaisedButton. We can use it as like below code snippets

```
//FlatButton Example
new FlatButton(
child: Text("Click here"),
onPressed: () {
// Do something here

},
),

//RaisedButton Example
new RaisedButton(
child: Text("Click here"),
elevation: 5.0,
onPressed: () {
// Do something here
},
```

3. Image
This widget holds the image which can fetch it from multiple sources like from the asset folder or directly from the URL. It provides many constructors for loading image, which are given below:

o Image: It is a generic image loader, which is used by ImageProvider.
o asset: It load image from your project asset folder.
o file: It loads images from the system folder.
o memory: It load image from memory.
o network: It loads images from the network.
To add an image in the project, you need first to create an assets folder where you keep your images and then add the below line in pubspec.yaml file.

```
assets:
- assets/comp.jpg
```

Steps to Add an Image:
Step 1. Create a new folder

- It should be in the root of your flutter project. You can name it whatever you want, but assets are preferred.
- If you want to add other assets to your app, like fonts, it is preferred to make another subfolder named images.

Step 2. Now you can copy your image to images sub-folder. The path should look like assets/images/yourImage. Before adding images also check the above-mentioned supported image formats.

Step 3. Register the assets folder in pubspec.yaml file and update it.
To add images, write the following code:

```
flutter:
  assets:
       - assets/images/yourFirstImage.jpg
       - assets/image/yourSecondImage.jpg
```

If you want to include all the images of the assets folder then add this:

```
flutter:
  assets:
        - assets/images/
```

Step 4. Insert the image code in the file, where you want to add the image.

Code and output:

Mails.dart file:

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:gmail_clone/addmail.dart';
import 'package:gmail_clone/sentmails.dart';
import 'package:gmail_clone/starredmails.dart';
```

```
import 'package:gmail_clone/variables.dart';
import 'package:gmail_clone/viewmail.dart';
import 'package:intl/intl.dart';

class Mails extends StatefulWidget {
  const Mails({super.key});

  @override
  State<Mails> createState() => _MailsState();
}

class _MailsState extends State<Mails> {
  String? usermail;
  Stream? mystream;
  int _currentIndex = 0;
  @override
  initState() {
    super.initState();
    getuserdata();
    getstream();
  }

  getstream() {
    setState(() {
      mystream = usercollection.doc(usermail).collection('inbox').snapshots();
    });
  }

  getuserdata() async {
    var firebaseuser = FirebaseAuth.instance.currentUser;
    setState(() {
      usermail = firebaseuser!.email;
    });
  }

  starmessage(String id) async {
    DocumentSnapshot document =
        await usercollection.doc(usermail).collection('inbox').doc(id).get();

    if (document['stared'] == false) {
      usercollection
          .doc(usermail)
          .collection('inbox')
          .doc(id)
          .update({'stared': true});
    } else {
      usercollection
          .doc(usermail)
          .collection('inbox')
```

```
        .doc(id)
        .update({'stared': false});
  }
 }

 searchmail(String str) async {
  setState(() {
    mystream = usercollection
        .doc(usermail)
        .collection('inbox')
        //where-->used for querying
        .where('subject', isGreaterThanOrEqualTo: str)
        .snapshots();
  });
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      backgroundColor: Colors.white,
      title: Material(
        elevation: 8.0,
        child: TextFormField(
          onFieldSubmitted: (str) => searchmail(str),
          decoration: InputDecoration(
            hintText: 'Search...',
            border: InputBorder.none,
            // icon: Builder(
            //   builder: (context) => IconButton(
            //     // margin: EdgeInsets.only(left: 5),
            //     icon: Icon(Icons.dehaze),
            //     onPressed: () {
            //       Scaffold.of(context).openDrawer();
            //     },
            //   ),
            // ),
            suffixIcon: Container(
              margin: EdgeInsets.all(5),
              child: CircleAvatar(
                    backgroundImage: NetworkImage(exampleimage),
              ),
            ),
          ),
        ),
      ),
    ),
    drawer: Drawer(
      child: ListView(
```

```
padding: EdgeInsets.zero,
children: <Widget>[
 DrawerHeader(
  decoration: BoxDecoration(
   color: Color.fromARGB(255, 255, 255, 255),
   border: Border.all(
    color: Colors.grey, // Border color
    width: 1, // Border width
   ),
  ),
  child: Row(
   children: [
    Image.asset(
     'assets/com.png', // Replace with the actual path to your Gmail logo image
     height: 40, // Adjust the height as needed
     width: 40, // Adjust the width as needed
    ),
    SizedBox(width: 10),
    Text(
     'Gmail',
     style: TextStyle(
      color: Color.fromRGBO(161, 160, 160, 1),
      fontSize: 24,
     ),
    ),
   ],
  ),
 ),

 ListTile(
  leading: Icon(Icons.inbox),
  title: Text('Inbox'),
  onTap: () {
   Navigator.pop(context); // Close the drawer
   // Add your logic to navigate to Inbox screen
  },
 ),
 ListTile(
  leading: Icon(Icons.star),
  title: Text('Starred'),
  onTap: () {
   Navigator.pop(context);
   Navigator.push(
    context,
    MaterialPageRoute(
     builder: (context) => StarredMailsScreen(),
    ),
   ); // Close the drawer
   // Add your logic to navigate to Starred screen
```

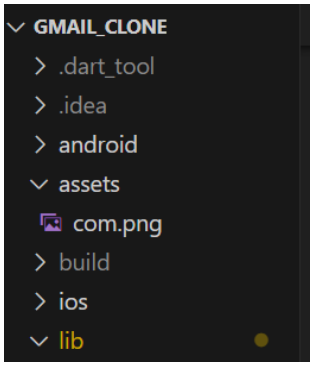
```
          },
        ),
        ListTile(
          leading: Icon(Icons.send),
          title: Text('Sent'),
          onTap: () {
            Navigator.pop(context); // Close the drawer
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => SentMailsScreen()),
            );
          },
        ),
        ListTile(
          leading: Icon(Icons.drafts),
          title: Text('Drafts'),
          onTap: () {
            Navigator.pop(context); // Close the drawer
            // Add your logic to navigate to Drafts screen
          },
        ),
        ListTile(
          leading: Icon(Icons.access_time),
          title: Text('Scheduled'),
          onTap: () {
            Navigator.pop(context); // Close the drawer
            // Add your logic to navigate to Scheduled screen
          },
        ),
        ListTile(
          leading: Icon(Icons.error),
          title: Text('Spam'),
          onTap: () {
            Navigator.pop(context); // Close the drawer
            // Add your logic to navigate to Spam screen
          },
        ),
        ListTile(
          leading: Icon(Icons.delete),
          title: Text('Bin'),
          onTap: () {
            Navigator.pop(context); // Close the drawer
            // Add your logic to navigate to Bin screen
          },
        ),
        // Add more options as needed
      ],
    ),
  ),
```

```
floatingActionButton: FloatingActionButton(
  backgroundColor: Colors.white,
  onPressed: () => Navigator.push(
      context, MaterialPageRoute(builder: (context) => AddMail())),
  child: Icon(
    Icons.add,
    size: 45,
    color: Colors.red,
  ),
),
floatingActionButtonLocation: FloatingActionButtonLocation.endFloat,

//cards for mails,loop through this list
//continuous data from firebase-->StreamBuilder
body: StreamBuilder(
    stream: mystream,
    builder: (context, snapshot) {
      if (!snapshot.hasData) {
        return CircularProgressIndicator();
      }
      print(snapshot.data!.docs.length);

      return ListView.builder(
          itemCount: snapshot.data!.docs.length,
          itemBuilder: (BuildContext context, int index) {
            DocumentSnapshot email = snapshot.data!.docs[index];
            print(email);
            return InkWell(
              onTap: () => Navigator.push(
                  context,
                  MaterialPageRoute(
                      builder: (context) => ViewMail(
                          email['id'],
                          email['sender'],
                          email['time'],
                          email['picture'],
                          email['mail'],
                          email['subject'],
                          email['profilepic'],
                          email['username']))),
              child: Padding(
                padding: EdgeInsets.all(14),
                child: Row(
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  children: <Widget>[
                    Row(
                      children: [
                        CircleAvatar(
                          backgroundImage:
```

```
          NetworkImage(email['profilepic']),
      radius: 24,
    ),
    SizedBox(
      width: 15.0,
    ),
    Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
        Text(
          // 'Abhinaya',
          email['username'],
          style: email['hasred'] == false
              ? mystyle(
                  22, Colors.black, FontWeight.w700)
              : mystyle(
                  22, Colors.black, FontWeight.w400),
        ),
        Text(
          // 'Hello,how are u?',
          email['subject'],
          overflow: TextOverflow.ellipsis,
          style: email['hasred'] == false
              ? mystyle(
                  18, Colors.black, FontWeight.w700)
              : mystyle(
                  18, Colors.black, FontWeight.w400),
        )
      ],
    ),
  ],
),
SizedBox(
  width: 5.0,
),
Column(children: <Widget>[
  Text(
    DateFormat.Hm()
        .format(email['time'].toDate())
        .toString(),
    style: mystyle(18, Colors.black, FontWeight.w500),
  ),
  SizedBox(
    height: 5.0,
  ),
  InkWell(
      onTap: () => starmessage(email['id']),
      // child: Icon(Icons.star_border, size: 32))
      child: email['stared'] == false
```

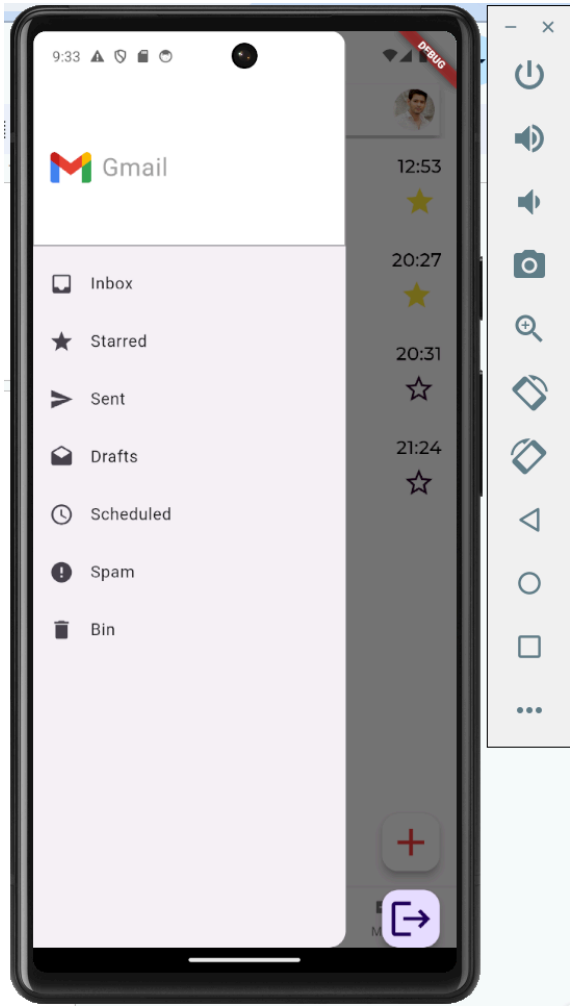
```
                                ? Icon(Icons.star_border, size: 32)
                                : Icon(
                                    Icons.star,
                                    color: Colors.yellow,
                                    size: 32,
                                  )),
                          ])
                        ],
                      ),
                    ),
                  );
                });
          }),
      //TO DO:bottombar
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: _currentIndex,
        onTap: (index) {
          // Handle navigation to different screens based on the selected index
          setState(() {
            _currentIndex = index;
          });
        },
        items: [
          BottomNavigationBarItem(
            icon: Icon(Icons.mail),
            label: 'Mail',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.chat),
            label: 'Chat',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.video_call),
            label: 'Meet',
          ),
        ],
      ),
    );
  }
}
```

Project Title: Gmail Clone **Roll No. 12**




```
# To add assets to your application, add an assets section, like this:
assets:
  - assets/com.png
#   - images/a_dot_ham.jpeg
```

Output:



Conclusion: Understood how to add icons and images in flutter app

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	12
Name	Abhinaya Danda
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

ABHINAYA DANDA

D15A Roll no.12

Experiment : 04

AIM : To create an interactive Form using form widget

THEORY:

Flutter Form Widget:

The Form widget in Flutter provides a way to create and manage a group of related form elements. It helps with form validation, submission, and data storage. When using the Form widget, you typically also use TextFormField for text input and other form-related widgets.

TextFormFields:

- Use **TextFormField** widgets for text input fields within the form. These widgets handle user input, validation, and saving data.

Form Submission:

- Typically, a button triggers form submission. The **onPressed** callback of the button calls a function, which in turn validates and saves the form data.

Code:

Signup.dart

```
import 'dart:io';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:gmail_clone/models/user.dart' as CustomUser;
import 'package:flutter/material.dart';
import 'package:gmail_clone/variables.dart';
import 'package:image_picker/image_picker.dart';

class SignUp extends StatefulWidget {
  const SignUp({super.key});
  @override
  State<SignUp> createState() => _SignUpState();
}

class _SignUpState extends State<SignUp> {
  final GlobalKey<ScaffoldMessengerState> scaffoldMessengerKey =
      GlobalKey<ScaffoldMessengerState>();
  File? imagepath;
  var usernamecontroller = TextEditingController();
  var passwordcontroller = TextEditingController();
  pickimage(ImageSource imageSource) async {
    final image = await ImagePicker().pickImage(
      source: imageSource,
```

```
    maxWidth: 670,
    maxHeight: 800,
  );
  if (image != null) {
    setState(() {
      imagepath = File(image.path);
    });
    Navigator.pop(context);
  }
}

pickimagedialog() {
  return showDialog(
      context: context,
      builder: (context) {
        return SimpleDialog(
          children: <Widget>[
            SimpleDialogOption(
              onPressed: () => pickimage(ImageSource.gallery),
              child:
                  Text("From gallery", style: mystyle(20, Colors.lightBlue)),
            ),
            SimpleDialogOption(
              onPressed: () => pickimage(ImageSource.camera),
              child:
                  Text("From camera", style: mystyle(20, Colors.lightBlue)),
            ),
            SimpleDialogOption(
              onPressed: () => Navigator.pop(context),
              child: Text("Cancel", style: mystyle(20, Colors.lightBlue)),
            ),
          ],
        );
      });
}

uploadimage() async {
  //Store image
  UploadTask storage =
      profilepics.child(usernamecontroller.text).putFile(imagepath!);
  //complete storage
  TaskSnapshot storageTaskSnapshot = await storage;
  //download pic
  String downloadedpic = await storageTaskSnapshot.ref.getDownloadURL();
  return downloadedpic;
}

registeruser() async {
  try {
```

```
    String downloadpic = imagepath == null
        ? 'https://fdlc.org/157-1578186_user-profile-default-image-png-clipart-png/'
        : await uploadimage();
    FirebaseAuth.instance
        .createUserWithEmailAndPassword(
            email: usernamecontroller.text + '@gmail.com',
            password: passwordcontroller.text)
        .then((signeduser) {
      CustomUser.User().storeuser(usernamecontroller.text + '@gmail.com',
          usernamecontroller.text, passwordcontroller.text, downloadpic);
    });
    Navigator.pop(context);
  } catch (e) {
    print(e);
    SnackBar snackBar = SnackBar(content: Text(e.toString()));
    scaffoldMessengerKey.currentState!.showSnackBar(snackBar);
  }
}
@override
Widget build(BuildContext context) {
  return Scaffold(
    key: scaffoldMessengerKey,
    body: SingleChildScrollView(
      child: Center(
          child: Container(
        width: MediaQuery.of(context).size.width,
        height: MediaQuery.of(context).size.height,
        child: Card(
          elevation: 10,
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Text("Google",
                  style: mystyle(
                    25,
                    Color.fromARGB(255, 4, 20, 243),
                  )),
              SizedBox(height: 5.0),
              Text("Create Account", style: mystyle(18, Colors.black)),
              SizedBox(height: 10.0),
              Text("Continue to gmail", style: mystyle(18, Colors.black)),
              SizedBox(height: 20),
              Container(
                width: MediaQuery.of(context).size.width,
                margin: EdgeInsets.only(left: 20, right: 20),
                child: TextField(
                  controller: usernamecontroller,
                  decoration: InputDecoration(
                      filled: true,
```

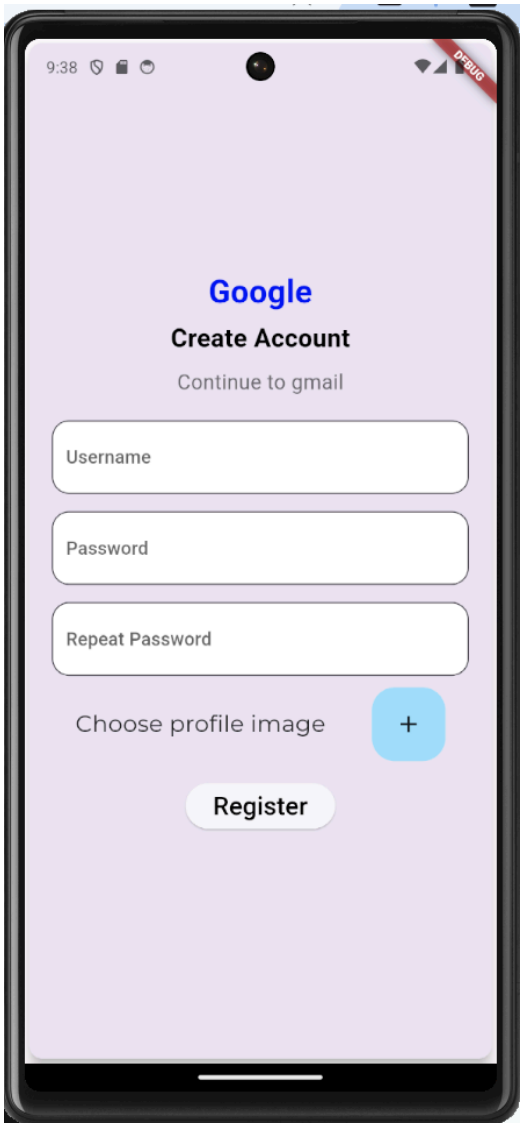
```
        hintText: "Username",
        border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(15.0))),
  ),
),
SizedBox(height: 15.0),
Container(
  width: MediaQuery.of(context).size.width,
  margin: EdgeInsets.only(left: 20, right: 20),
  child: TextField(
    controller: passwordcontroller,
    decoration: InputDecoration(
        filled: true,
        hintText: "Password",
        border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(15.0))),
  ),
),
SizedBox(height: 15.0),
Container(
  width: MediaQuery.of(context).size.width,
  margin: EdgeInsets.only(left: 20, right: 20),
  child: TextField(
    decoration: InputDecoration(
        filled: true,
        hintText: "Repeat Password",
        border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(15.0))),
  ),
),
SizedBox(height: 10.0),
Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: <Widget>[
    InkWell(
      onTap: () => pickimagedialog(),
      child: Container(
        width: 64,
        height: 64,
        decoration: BoxDecoration(
            color: Colors.lightBlue,
            borderRadius: BorderRadius.circular(20.0)),
        child: Center(
          child: imagepath == null
              ? Icon(Icons.add)
              : Image(image: FileImage(imagepath!)),
        ),
      ),
    ),
```

```
                InkWell(
                    onTap: () => pickimagedialog(),
                    child:
                        Text("Choose profile image", style: mystyle(20))),
              ],
            ),
            SizedBox(height: 15.0),
            ElevatedButton(
              onPressed: () => registeruser(),
              style: ElevatedButton.styleFrom(
                backgroundColor: Colors.lightBlue[200],
              ),
              child: Text("Register", style: mystyle(20)),
            )
          ],
        ),
      ),
    )),
  ),
 );
 }
}
```

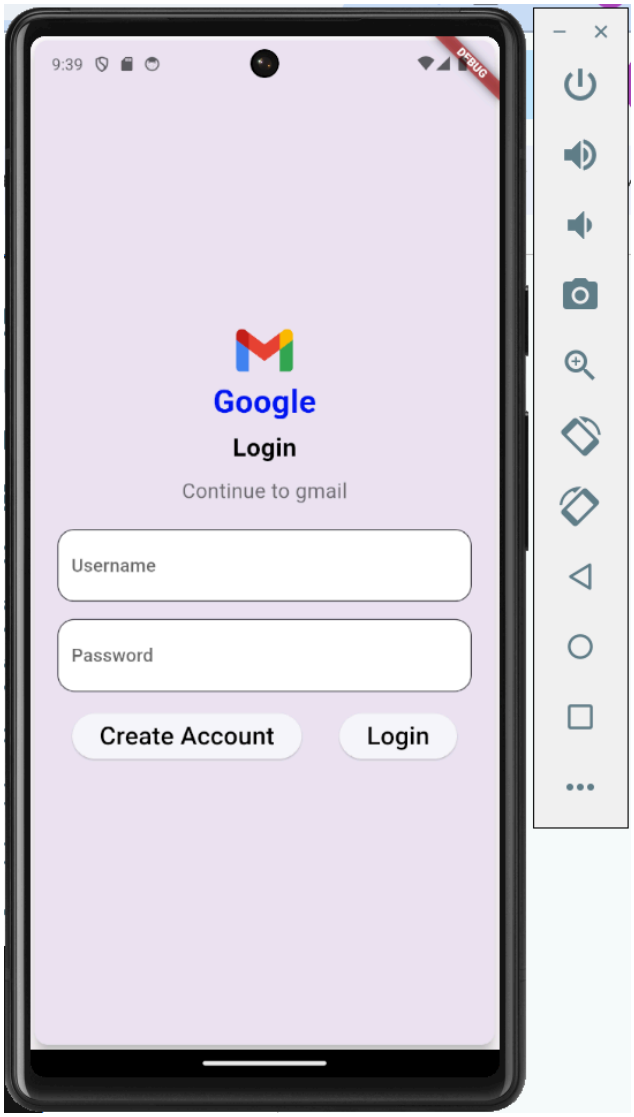
Project Title: Gmail Clone **Roll No. 12**

Output:

SignUp Page



Login Page



Project Title: Gmail Clone **Roll No. 12**

Code:

Addmail.dart file

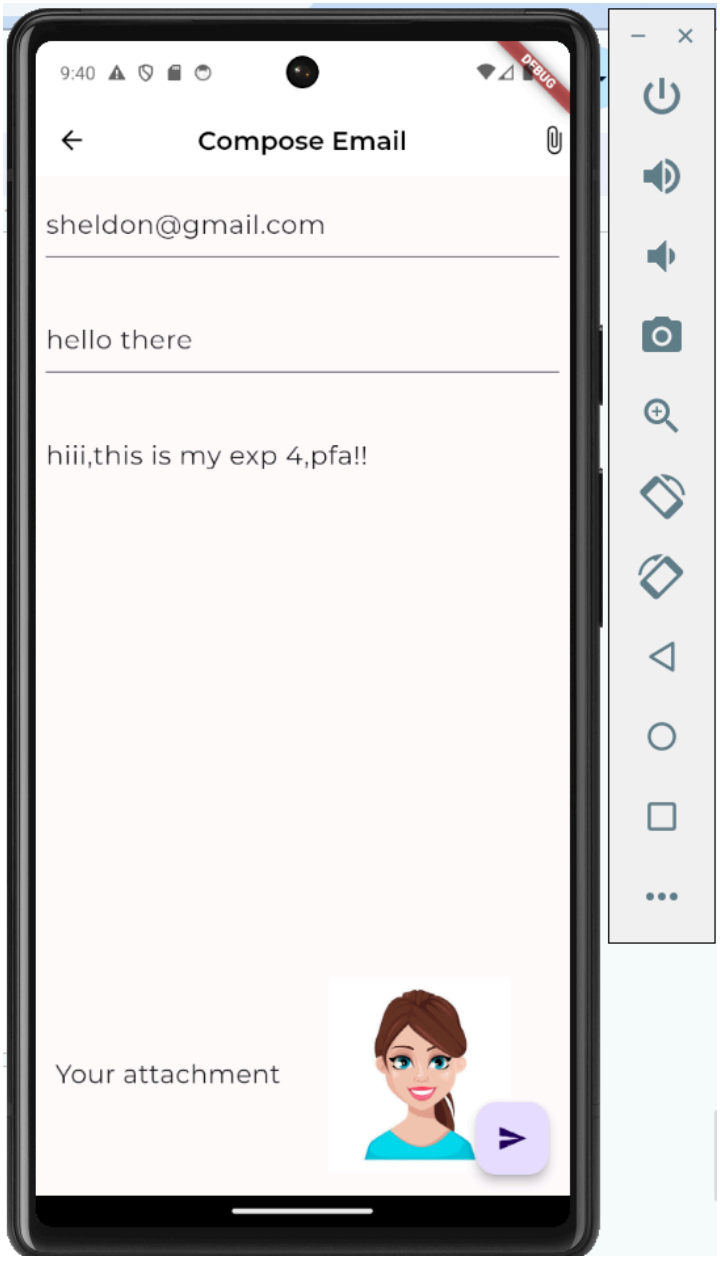
```
Widget build(BuildContext context) {
   return Scaffold(
      floatingActionButton: FloatingActionButton(
        onPressed: () => sendmail(),
        child: Icon(Icons.send),
      ),
      appBar: AppBar(
        backgroundColor: Colors.white,
        actions: <Widget>[
          InkWell(
            onTap: () => pickimagedialog(),
            child: Icon(
              Icons.attach_file,
              color: Colors.black,
            ),
          ),
        ],
        title: Text(
          "Compose Email",
          style: mystyle(20, Colors.black, FontWeight.w600),
        ),
        centerTitle: true,
        leading: InkWell(
          onTap: () => Navigator.pop(context),
          child: Icon(
            Icons.arrow_back,
            color: Colors.black,
          ),
        ),
      ),
      body: uploading == true
          ? Center(
              child: Text(
              "Sending mail ...",
              style: mystyle(35),
            ))
          : Padding(
              padding: EdgeInsets.all(8.0),
              child: Column(
                children: <Widget>[
                  Container(
                      width: MediaQuery.of(context).size.width,
                      height: MediaQuery.of(context).size.height * 0.1,
                      child: TextFormField(
                        controller: receiver,
                        style: mystyle(20),
```

```
          decoration: InputDecoration(
              hintText: "To", labelStyle: mystyle(20)),
        )),
    Container(
        width: MediaQuery.of(context).size.width,
        height: MediaQuery.of(context).size.height * 0.1,
        child: TextFormField(
          controller: subject,
          style: mystyle(20),
          decoration: InputDecoration(
              hintText: "Subject", labelStyle: mystyle(20)),
        )),
    Expanded(
      child: Container(
          width: MediaQuery.of(context).size.width,
          child: TextFormField(
            controller: mail,
            style: mystyle(20),
            maxLines: null,
            decoration: InputDecoration(
              hintText: "Mail",
              labelStyle: mystyle(20),
              border: InputBorder.none,
            ),
          )),
    ),
    imagepath == null
        ? Container()
        : MediaQuery.of(context).viewInsets.bottom > 0
            ? Container()
            : Container(
                alignment: Alignment.centerLeft,
                margin: EdgeInsets.only(bottom: 10),
                child: Row(
                  mainAxisAlignment:
                      MainAxisAlignment.spaceEvenly,
                  children: <Widget>[
                    Text(
                      "Your attachment",
                      style: mystyle(20),
                    ),
                    Container(
                      width: 200,
                      height: 150,
                      child: Image(image: FileImage(imagepath)),
                    )
                  ],
                ))
  ],
```

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

Output:

Compose Mail Page



Conclusion: Understood the use form widget,made an interactive form to compose email

****

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	12
Name	Abhinaya Danda
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Project Title: Gmail Clone **Roll No. 12**

ABHINAYA DANDA

D15A Roll no.12

Experiment : 05

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

THEORY:

Navigation and Routing:

Navigation refers to moving between different screens or pages within a Flutter app, while routing involves defining the paths or routes between those screens. Flutter provides a flexible and powerful navigation system with the help of the Navigator class and Route objects.

Navigator Class: The Navigator manages a stack of routes (screens/pages) and provides methods to push, pop, and replace routes.

Routes: A route represents a screen or page in the app. Flutter offers two types of routes: named routes and unnamed routes.

- Named Routes: Named routes provide a way to define routes with unique names, making it easier to navigate within the app using those names.
- Unnamed Routes: Unnamed routes are typically used for simple applications or when the route doesn't need a unique identifier.

MaterialPageRoute and CupertinoPageRoute: Flutter provides MaterialPageRoute for Material Design apps and CupertinoPageRoute for iOS-style apps. These classes handle animations and transitions when navigating between routes.

Navigation Methods: Common methods include **Navigator.push(), Navigator.pop(), Navigator.pushReplacement()**, etc. These methods allow you to navigate forward, backward, or replace the current route.

Steps:

1. **Create two routes. (In Flutter, screens and pages are called routes.)**
2. **Navigate to the second route using Navigator.push().**
3. **Return to the first route using Navigator.pop().**

Project Title: Gmail Clone **Roll No. 12**

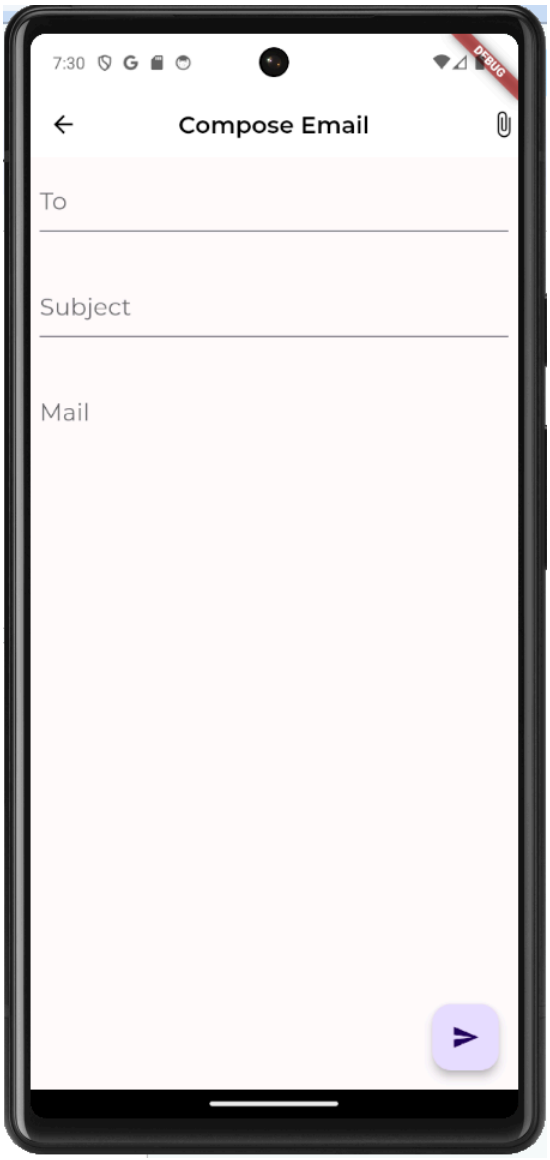
Code:

Mails.dart:
For navigation to Addmail.dart using '+' icon floatingActionButton

```
floatingActionButton: FloatingActionButton(
        backgroundColor: Colors.white,
        onPressed: () => Navigator.push(
            context, MaterialPageRoute(builder: (context) => AddMail())),
        child: Icon(
          Icons.add,
          size: 45,
          color: Colors.red,
        ),
      ),
```

Project Title: Gmail Clone **Roll No. 12**

Output:
Navigated to Addmail.dart

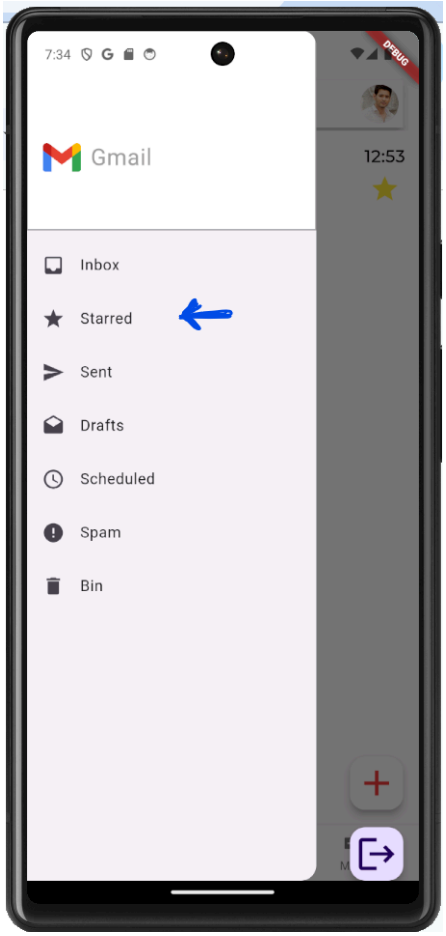


Project Title: Gmail Clone **Roll No. 12**

Mails.dart:

From the left side bar,when clicked on Starred mails,it will navigate to starred mails page:

```
ListTile(
            leading: Icon(Icons.star),
            title: Text('Starred'),
            onTap: () {
              Navigator.pop(context);
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => StarredMailsScreen(),
                ),
              ); // Close the drawer
              // Add your logic to navigate to Starred screen
            },
          ),
```



Starredmails.dart:

```
class _StarredMailsScreenState extends State<StarredMailsScreen> {
  String? usermail;
  Stream? mystream;
  int _currentIndex = 0;
  @override
  void initState() {
    super.initState();
    getuserdata();
    getStarredMailsStream();
  }

  getStarredMailsStream() {
    setState(() {
      mystream = usercollection
          .doc(usermail)
          .collection('inbox')
          .where('stared', isEqualTo: true)
          .snapshots();
    });
  }

  getuserdata() async {
    var firebaseuser = FirebaseAuth.instance.currentUser;
    setState(() {
      usermail = firebaseuser!.email;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Starred Mails'),
      ),
      body: StreamBuilder(
        stream: mystream,
        builder: (context, snapshot) {
          if (!snapshot.hasData) {
            return CircularProgressIndicator();
          }

          return ListView.builder(
            itemCount: snapshot.data!.docs.length,
```

```
itemBuilder: (BuildContext context, int index) {
  DocumentSnapshot email = snapshot.data!.docs[index];

  return InkWell(
    onTap: () {
      // Handle tap on a starred email
      // You can navigate to a detailed view or perform any action
      // based on the selected starred email.
    },
    child: Padding(
      padding: EdgeInsets.all(14),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: <Widget>[
          Row(
            children: [
              CircleAvatar(
                backgroundImage: NetworkImage(email['profilepic']),
                radius: 24,
              ),
              SizedBox(width: 15.0),
              Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: <Widget>[
                  Text(
                    email['username'],
                    style: mystyle(22, Colors.black),
                  ),
                  Text(
                    email['subject'],
                    overflow: TextOverflow.ellipsis,
                    style: mystyle(18, Colors.black),
                  )
                ],
              ),
            ],
          ),
          SizedBox(width: 5.0),
          Column(
            children: <Widget>[
              Text(
                DateFormat.Hm()
                    .format(email['time'].toDate())
                    .toString(),
```

```
                    style: mystyle(18, Colors.black),
                  ),
                  SizedBox(height: 5.0),
                ],
              ),
            ],
          ),
        ),
      );
    },
  );
},
),
```

Output:

Conclusion: Understood how to apply navigation, routing and gestures in Flutter App

****

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	12
Name	Abhinaya Danda
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	15

Project Title: Gmail Clone **Roll No. 12**

ABHINAYA DANDA

D15A Roll no.12

Experiment : 06

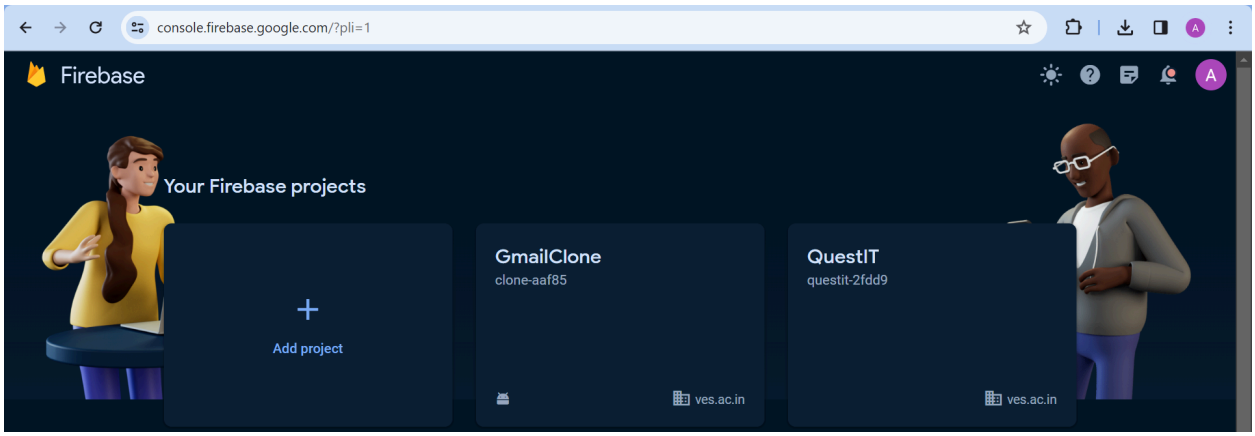
AIM : To Connect Flutter UI with fireBase database

THEORY:

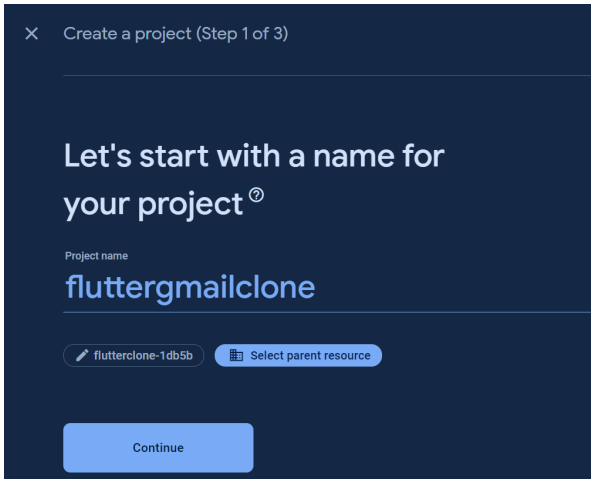
Integrating Firebase with a Flutter application involves several steps, and Firebase offers a variety of backend services that can enhance the functionality of your app. Here's a general process for integrating Firebase with a Flutter application:

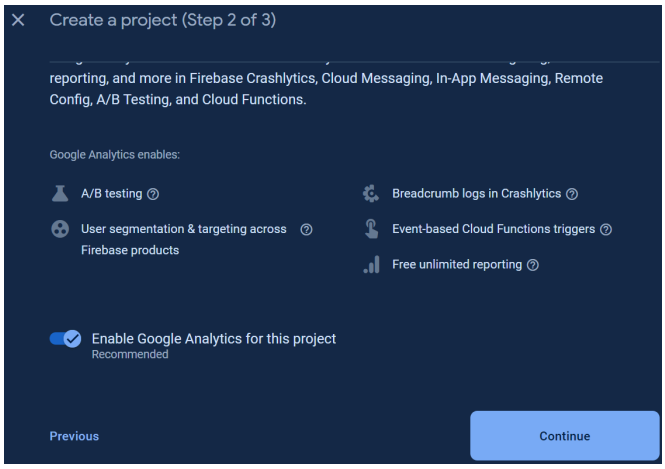
Steps for Firebase Integration in Flutter:

- Go to the **Firebase Console** and click on **"Add Project."**

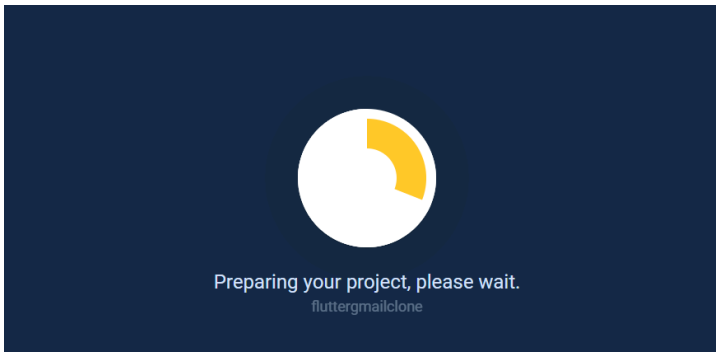


- Add project name:

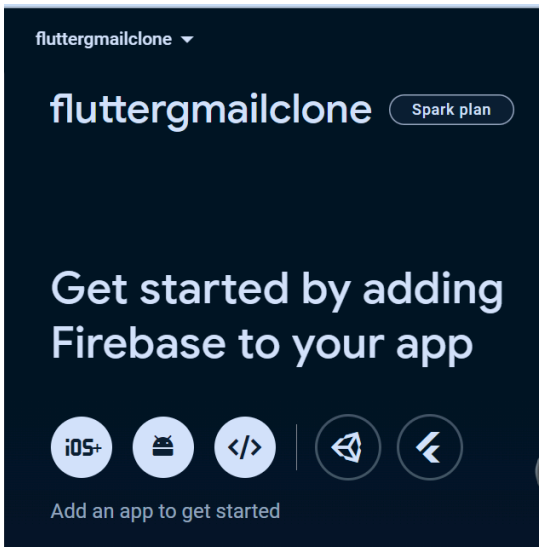




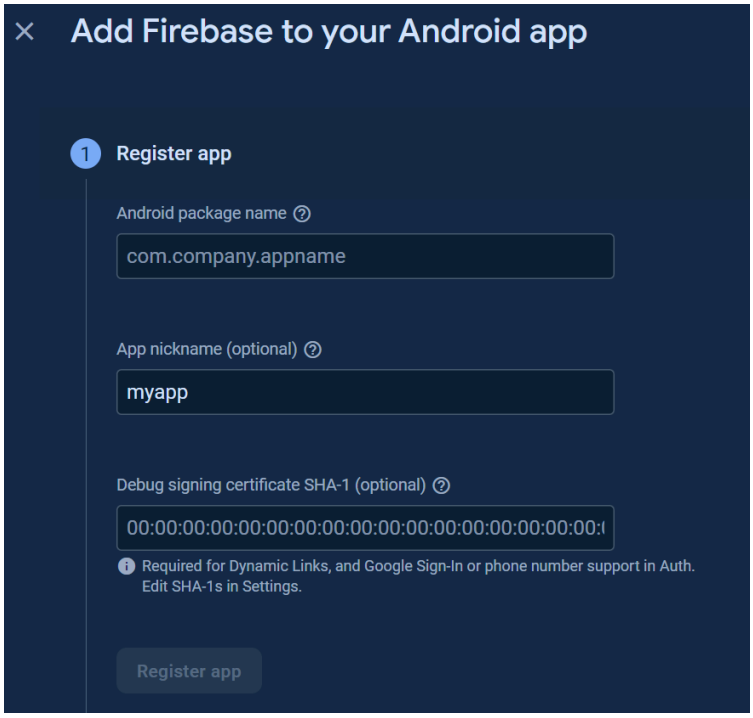
- Create project



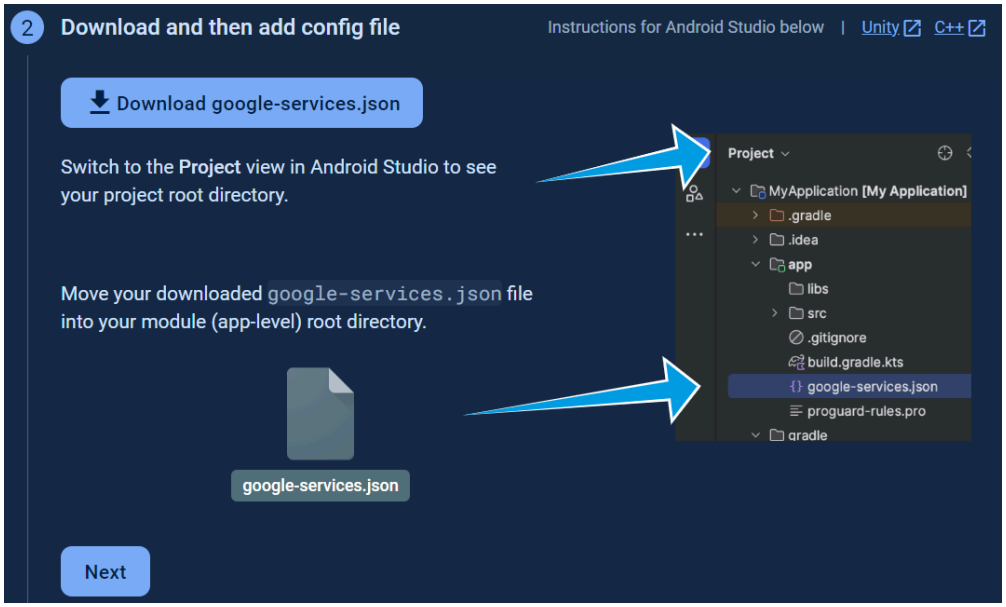
- **Create app ,click on Android icon**



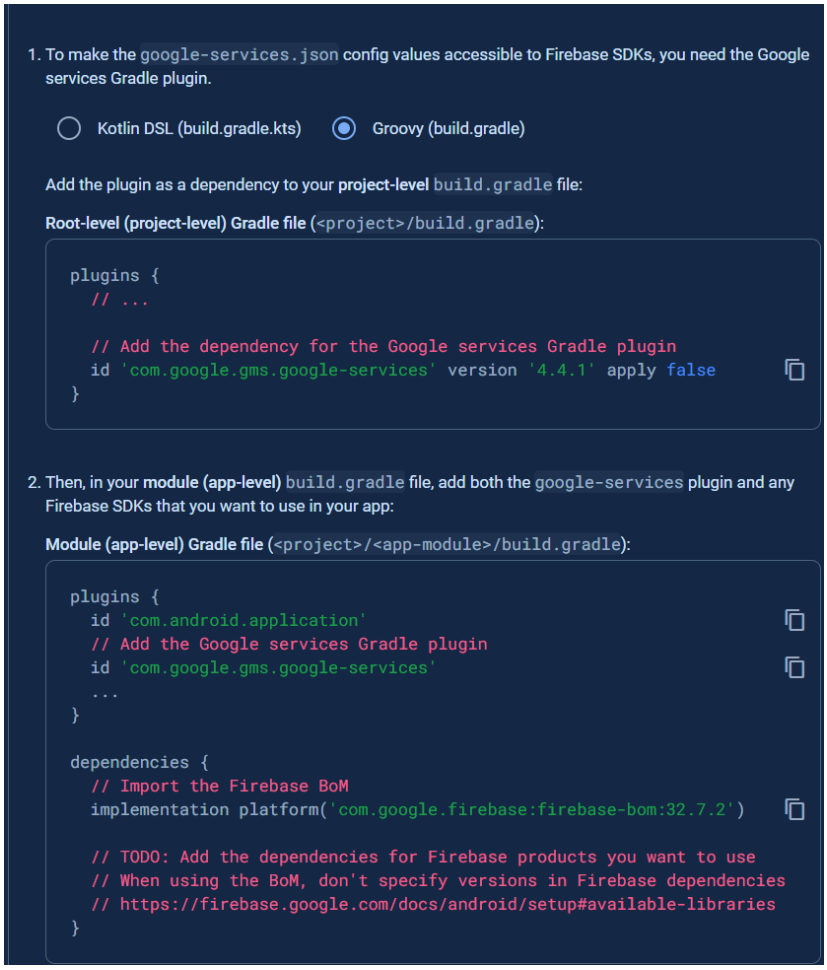
- Register app



- Download the file and place them in the respective directories app>src>google-services.json



- Make necessary changes in following files:



1. To make the `google-services.json` config values accessible to Firebase SDKs, you need the Google services Gradle plugin.

Kotlin DSL (build.gradle.kts) Groovy (build.gradle)

Add the plugin as a dependency to your **project-level** `build.gradle` file:

Root-level (project-level) Gradle file (`<project>/build.gradle`):

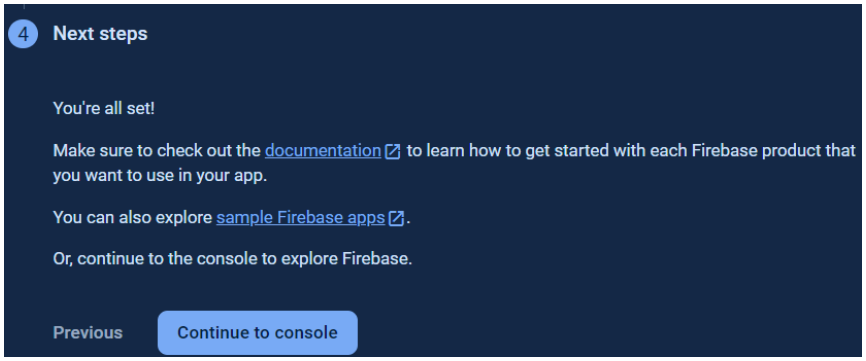
```
plugins {
  // ...

  // Add the dependency for the Google services Gradle plugin
  id 'com.google.gms.google-services' version '4.4.1' apply false
}
```

2. Then, in your **module (app-level)** `build.gradle` file, add both the `google-services` plugin and any Firebase SDKs that you want to use in your app:

Module (app-level) Gradle file (`<project>/<app-module>/build.gradle`):

```
plugins {
  id 'com.android.application'
  // Add the Google services Gradle plugin
  id 'com.google.gms.google-services'
  ...
}

dependencies {
  // Import the Firebase BoM
  implementation platform('com.google.firebase:firebase-bom:32.7.2')

  // TODO: Add the dependencies for Firebase products you want to use
  // When using the BoM, don't specify versions in Firebase dependencies
  // https://firebase.google.com/docs/android/setup#available-libraries
}
```

- Continue to console:



4 **Next steps**

You're all set!

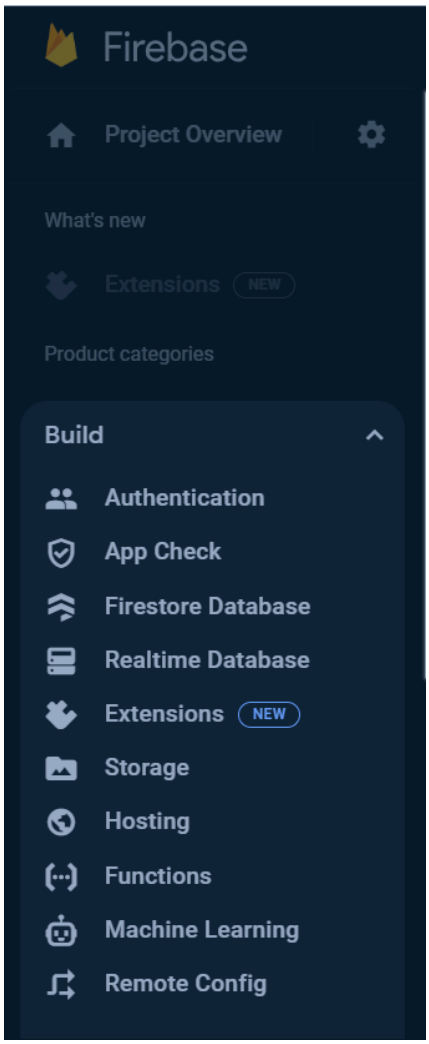
Make sure to check out the documentation to learn how to get started with each Firebase product that you want to use in your app.

You can also explore sample Firebase apps.

Or, continue to the console to explore Firebase.

Previous Continue to console

- Firebase Services like Authentication,Firestore Database,Realtime Database can be used,



- Add Firebase dependencies to your pubspec.yaml file:

pubspec.yaml

```
dependencies:
 firebase_core: ^latest_version
 firebase_auth: ^latest_version
 cloud_firestore: ^latest_version
```

Run **flutter pub get** to install the dependencies.

- **Initialize Firebase in your Flutter app**

main.dart file:

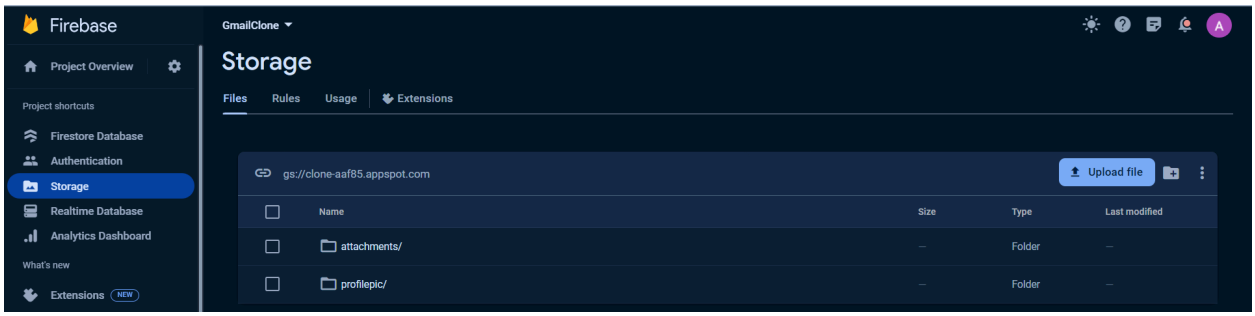
```
import 'package:firebase_core/firebase_core.dart';
void main() async {
 WidgetsFlutterBinding.ensureInitialized();
 await Firebase.initializeApp();
 runApp(MyApp());
}
```

In models>user.dart:

```
import 'package:gmail_clone/variables.dart';
class User{
  storeuser(email,username,password,profilepic){
      usercollection.doc(email).set({
        'email':email,
        'username':username,
        'password':password,
        'profilepic':profilepic
      });
  }
}
```

Variables.dart:

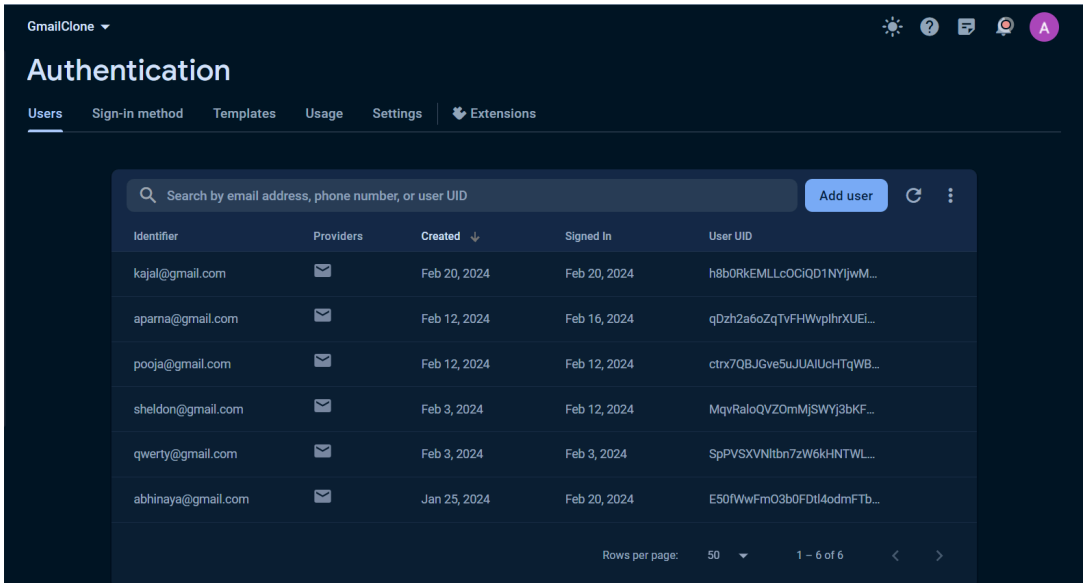
```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';

TextStyle mystyle(double size, [Color? color, FontWeight? fw]) {
  return GoogleFonts.montserrat(fontSize: size, fontWeight: fw, color: color);
}

CollectionReference usercollection =
    FirebaseFirestore.instance.collection('users');
Reference profilepics = FirebaseStorage.instance.ref().child('profilepic');

Reference attachments = FirebaseStorage.instance.ref().child('attachments');
```



SignUp.dart:

```
registeruser() async {
  try {
    String downloadpic = imagepath == null
        ? 'https://fdlc.org/157-1578186_user-profile-default-image-png-clipart-png/'
        : await uploadimage();
    FirebaseAuth.instance
        .createUserWithEmailAndPassword(
            email: usernamecontroller.text + '@gmail.com',
            password: passwordcontroller.text)
        .then((signeduser) {
      CustomUser.User().storeuser(usernamecontroller.text + '@gmail.com',
          usernamecontroller.text, passwordcontroller.text, downloadpic);
    });
```

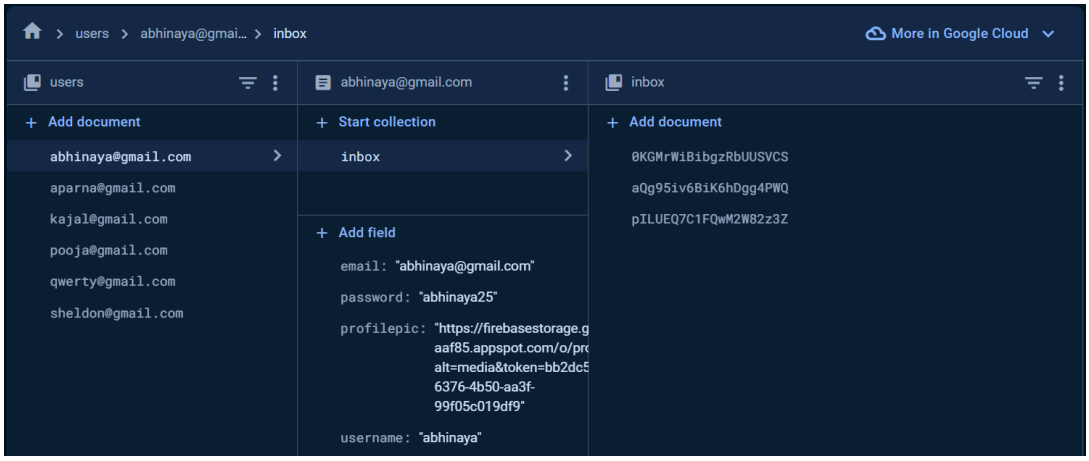
❖ **Authenticated Users**



- **Mails of each User**

Mails.dart:

```
class _MailsState extends State<Mails> {
  String? usermail;
  Stream? mystream;
  int _currentIndex = 0;
  @override
  initState() {
    super.initState();
    getuserdata();
    getstream();
  }

  getstream() {
    setState(() {
      mystream = usercollection.doc(usermail).collection('inbox').snapshots();
    });
  }

  getuserdata() async {
    var firebaseuser = FirebaseAuth.instance.currentUser;
    setState(() {
      usermail = firebaseuser!.email;
    });
  }

  starmessage(String id) async {
    DocumentSnapshot document =
        await usercollection.doc(usermail).collection('inbox').doc(id).get();

    if (document['stared'] == false) {
      usercollection
          .doc(usermail)
          .collection('inbox')
          .doc(id)
          .update({'stared': true});
    } else {
      usercollection
          .doc(usermail)
          .collection('inbox')
          .doc(id)
          .update({'stared': false});
    }
  }
```

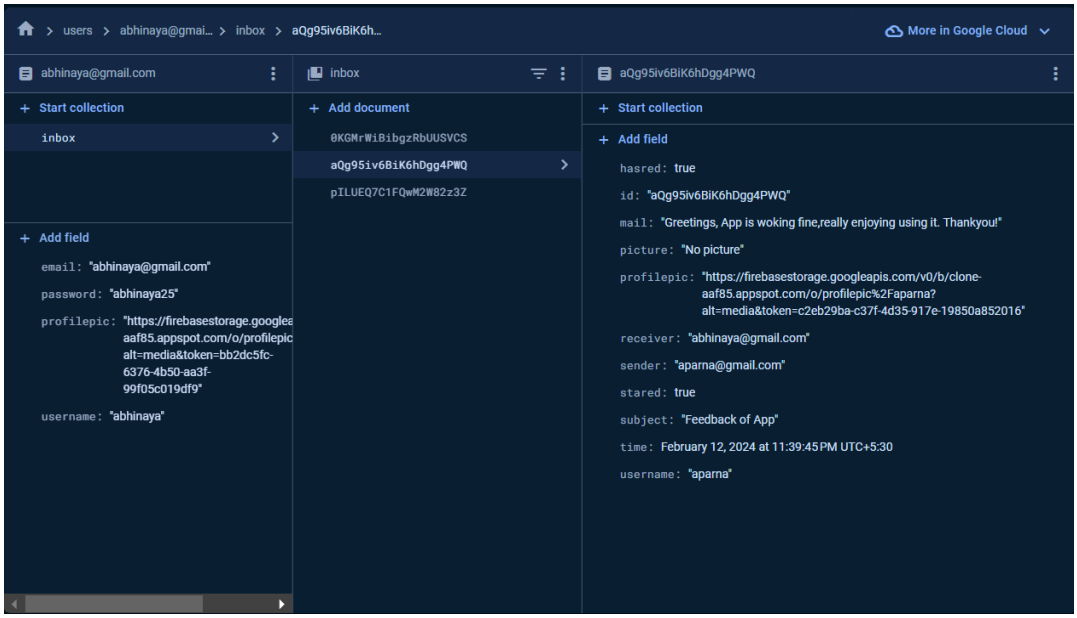


- **Mail Inbox stored**

addMail.dart:

```
sendmail() async {
   try {
    setState(() {
     uploading = true;
    });
    //made own change
    var firebaseuser = FirebaseAuth.instance.currentUser!;
    DocumentSnapshot userdocument =
       await usercollection.doc(firebaseuser.email).get();
    String picture = imagepath == null ? 'No picture' : await uploadimage();
    final id = usercollection.doc(receiver.text).collection('inbox').doc().id;
    usercollection.doc(receiver.text).collection('inbox').doc(id).set({
     'sender': firebaseuser.email,
     'receiver': receiver.text,
     'subject': subject.text,
     'mail': mail.text,
     'hasred': false,
     'stared': false,
     'id': id,
     'picture': picture,
     'time': DateTime.now(),
     'profilepic': userdocument['profilepic'],
     'username': userdocument['username'],
    });
    Navigator.pop(context);
   } catch (e) {
    print(e);
   }
  }
```

Project Title: Gmail Clone **Roll No. 12**



Conclusion: Connected Flutter UI with fireBase database,where users are authenticated,mail inboxes are stored,attachment and profile pic of each user is stored.

****

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.	12
Name	Abhinaya Danda
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

Project Title: Gmail Clone **Roll No. 12**

ABHINAYA DANDA

D15A 12

PWA EXPERIMENT NO: 07

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

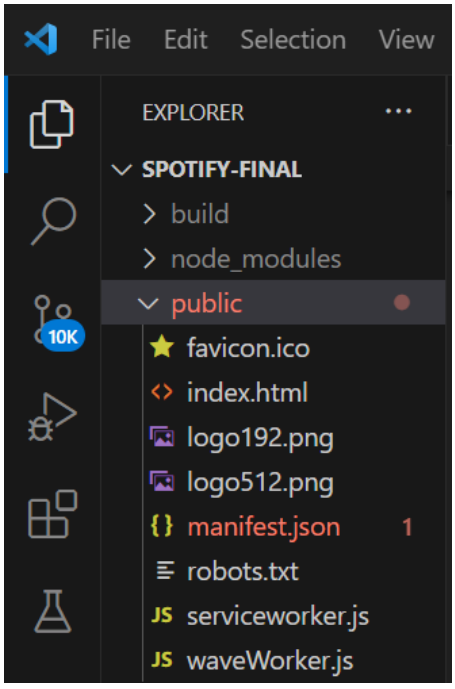
THEORY:

Regular Web App

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

Progressive Web App

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



manifest.json:

```
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
```

```
    {
      "src": "favicon.ico",
      // "sizes": "64x64 32x32 24x24 16x16",
      "sizes": "64x64",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

//Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
```

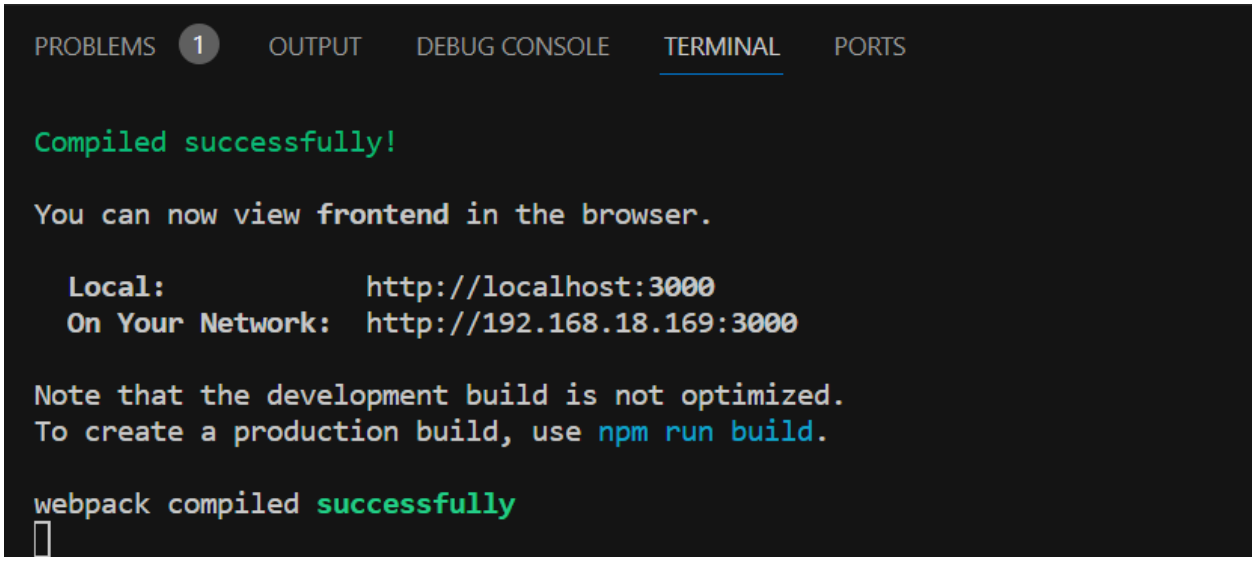
```
    Only files inside the `public` folder can be referenced from the HTML.

    Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
    work correctly both with client-side routing and a non-root public URL.
    Learn how to configure a non-root public URL by running `npm run build`.
  -->
  <title>React App</title>
 </head>
 <body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
  -->
  <script>
    window.addEventListener('load', () => {
      registerSW();
    });

    // Register the Service Worker
    async function registerSW() {
      if ('serviceWorker' in navigator) {
        try {
          await navigator
                .serviceWorker
                .register('serviceworker.js');
        }
        catch (e) {
          console.log('SW registration failed');
        }
      }
    }
  </script>
 </body>
</html>
```

//serviceworker.js

```
var staticCacheName = "pwa";

self.addEventListener("install", function (e) {
e.waitUntil(
```

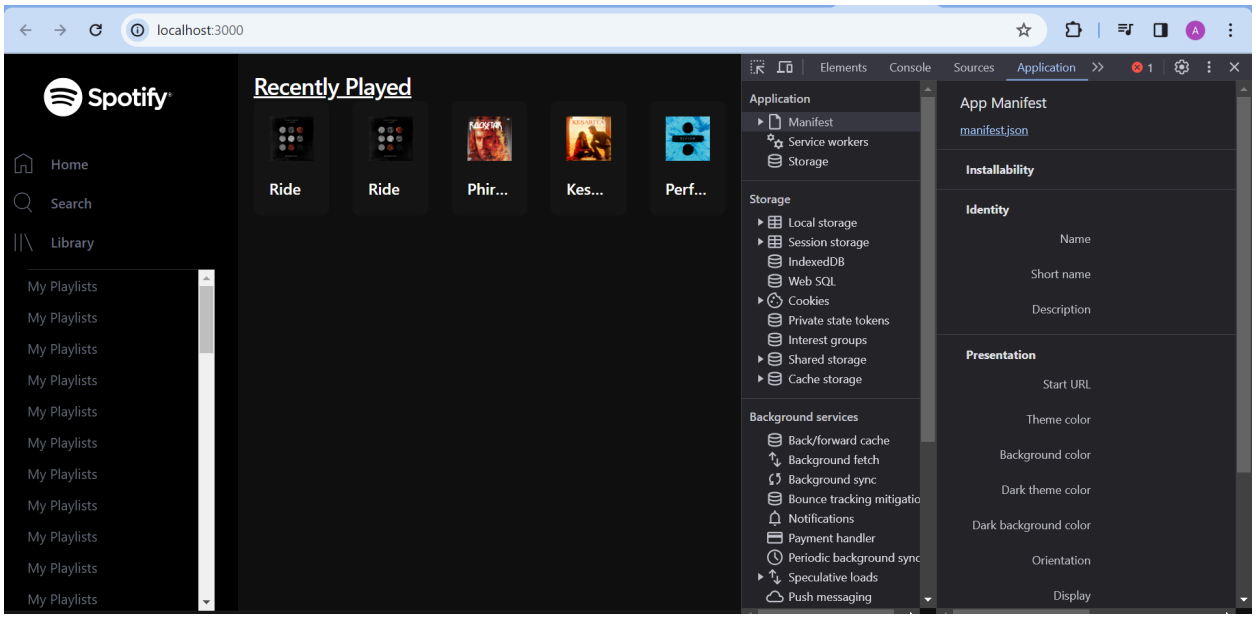
```
    caches.open(staticCacheName).then(function (cache) {
    return cache.addAll(["/"]);
    })
);
});

self.addEventListener("fetch", function (event) {
console.log(event.request.url);

event.respondWith(
    caches.match(event.request).then(function (response) {
    return response || fetch(event.request);
    })
);
});
```

>npm start



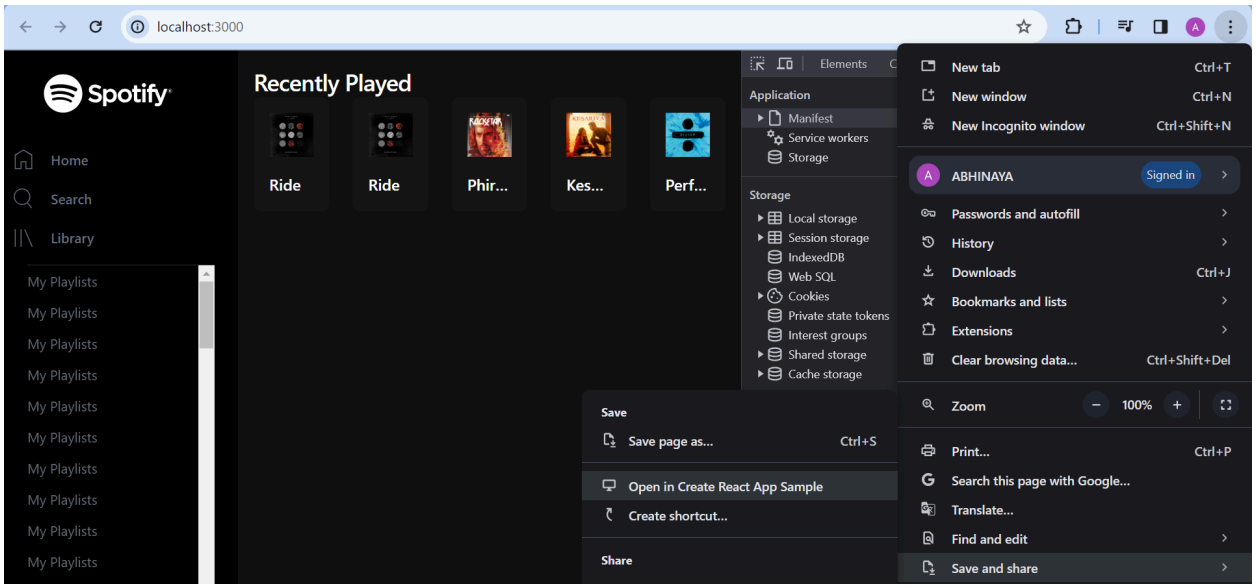
//open Developer tools options

Project Title: Gmail Clone **Roll No. 12**

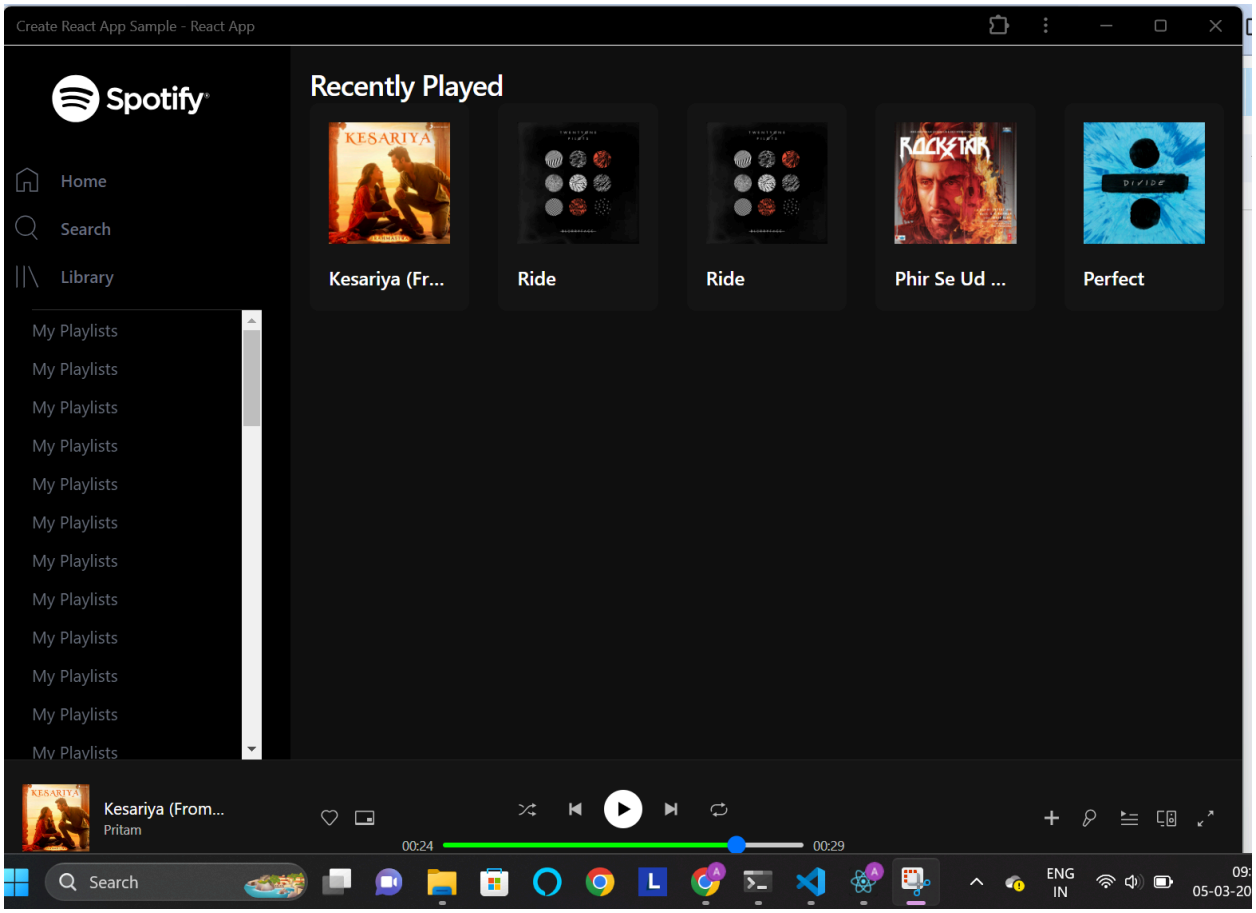


Project Title: Gmail Clone **Roll No. 12**

//click on 3 dots on top right corner of browser from app option install this site as an app



App icon will appear at the bottom



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.	12
Name	Abhinaya Danda
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Project Title: Gmail Clone **Roll No. 12**

Abhinaya Danda

D15A Roll No:12

PWA Experiment No:08

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?

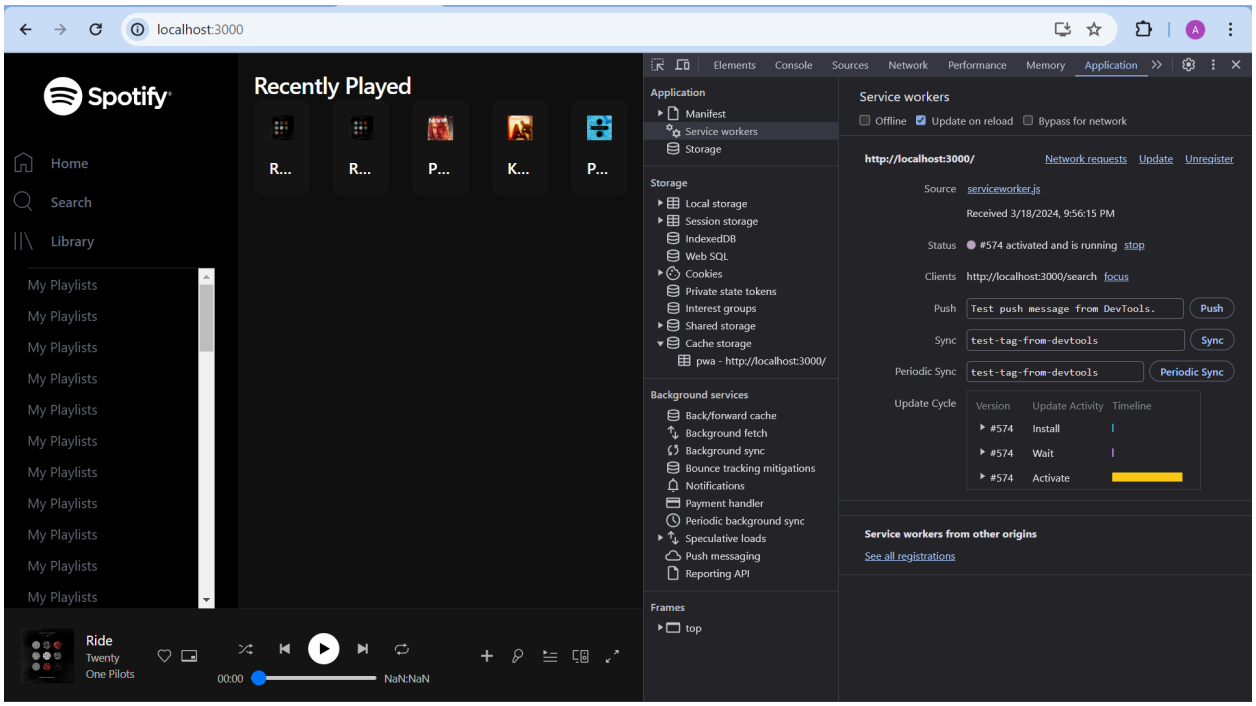
- You can dominate **Network Traffic**
  You can manage all network traffic of the page and do any manipulations. For example, when the page requests a CSS file, you can send plain text as a response or when the page requests an HTML file, you can send a png file as a response. You can also send a true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

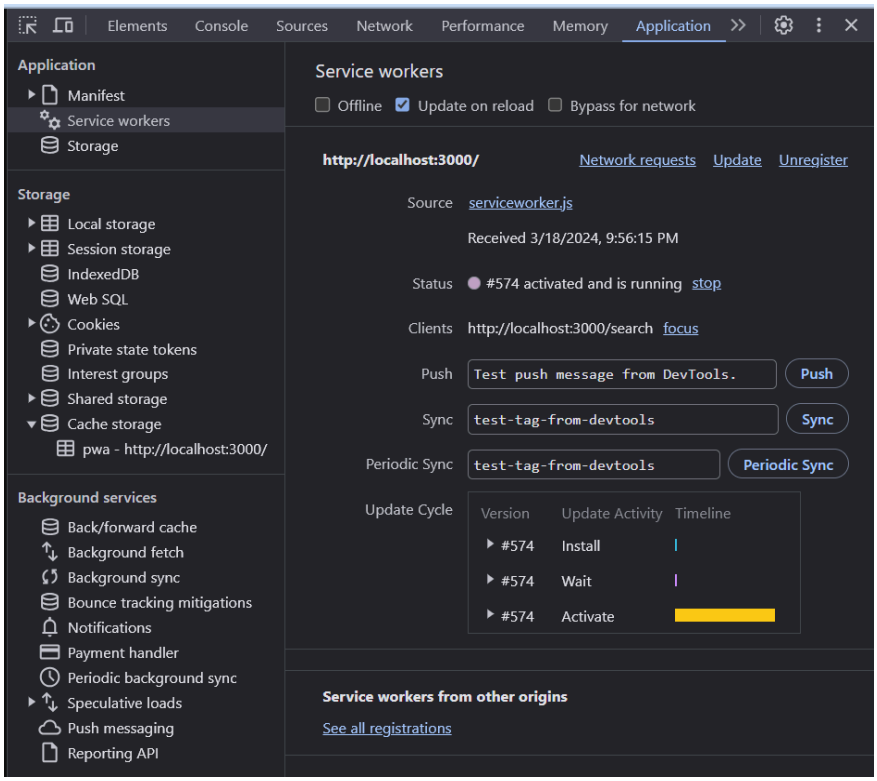
- You can **Cache**
  You can cache any request/response pair with Service Worker and Cache API and you can access these offline content anytime.

- 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.

Project Title: Gmail Clone **Roll No. 12**

Output:



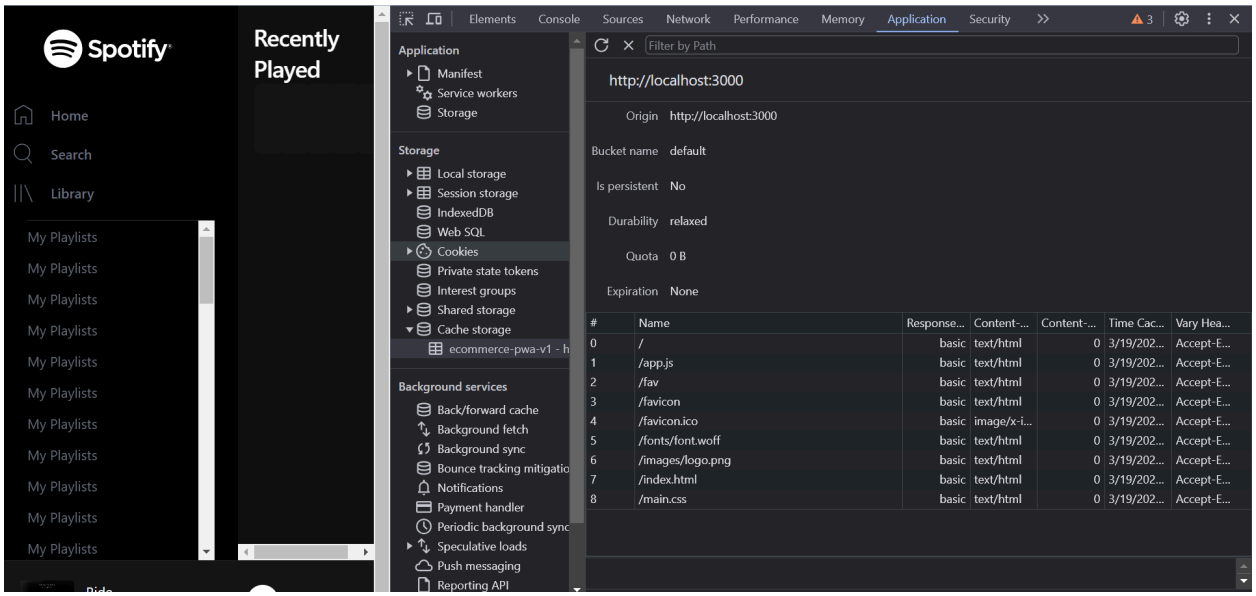


Project Title: Gmail Clone **Roll No. 12**

Changes in serviceworker.js:

```
const assetsToCache = [
    '/',
    '/index.html',
    '/favicon.ico',
    '/app.js',
    '/images/logo.png',
    '/fonts/font.woff',
];
```

In Developer tools->Cache storage



****

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.	12
Name	Abhinaya Danda
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Abhinaya Danda
D15A Roll No:12

PWA Experiment No:09

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.

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.

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.

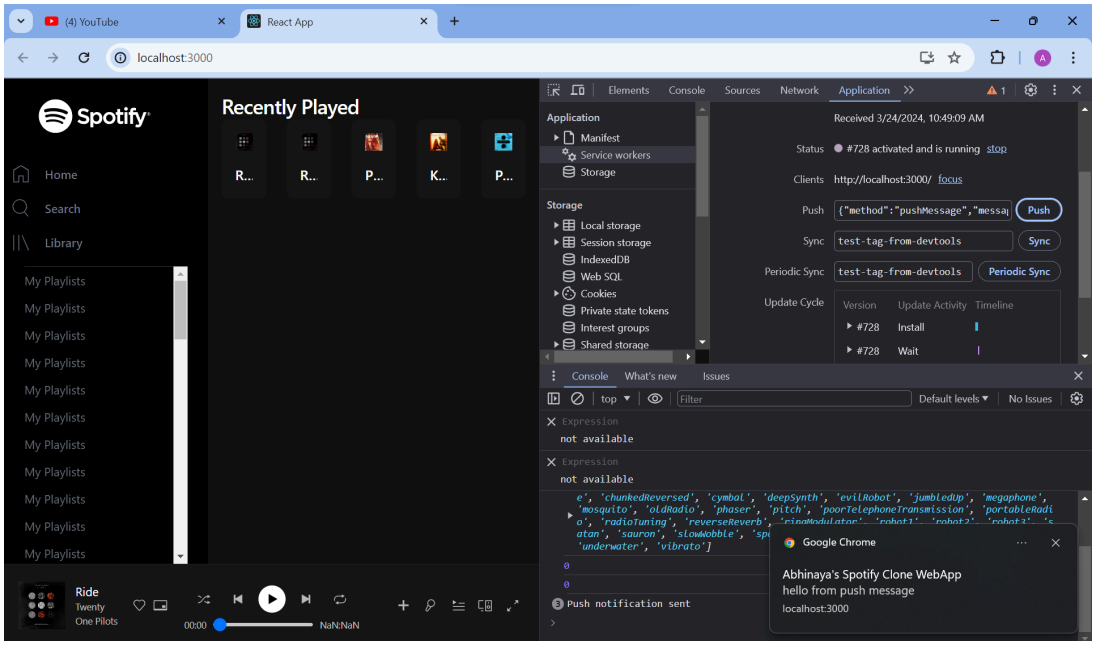
Push Notification:

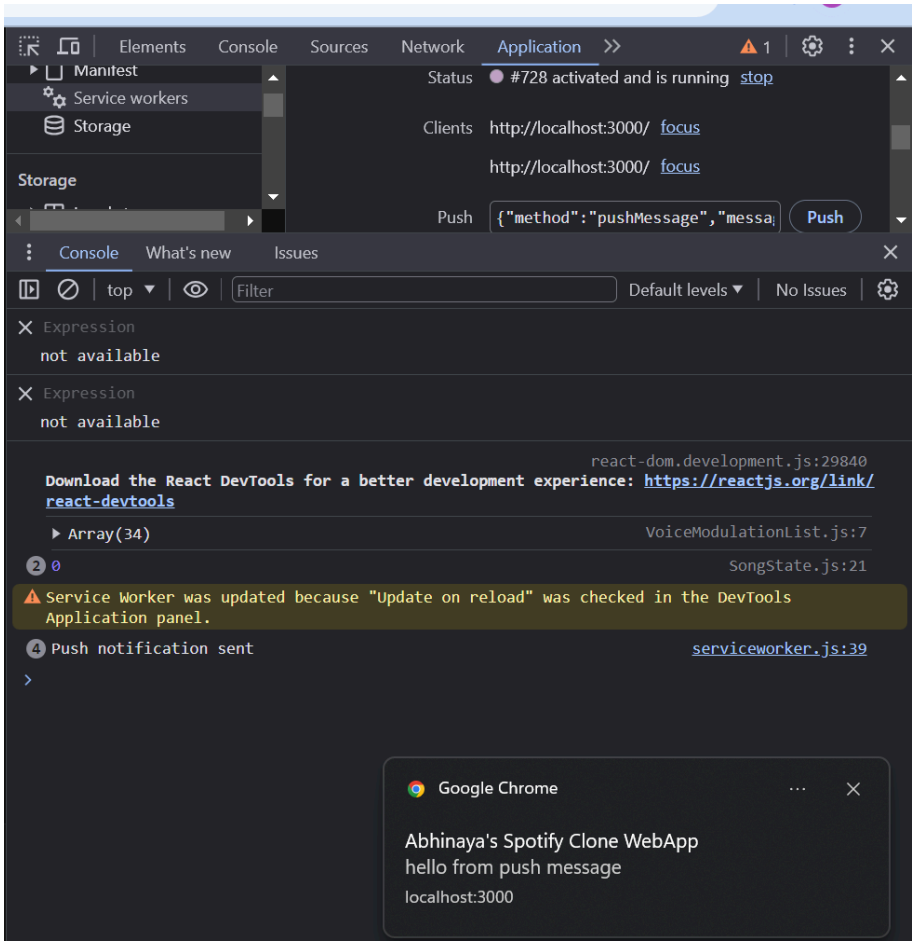
Code:

```
self.addEventListener("push", function (event) {
  if (event && event.data) {
    var data = event.data.json();
    if (data.method == "pushMessage") {
      console.log("Push notification sent");
      event.waitUntil(
        // Request permission and show notification if granted
        self.registration.showNotification("Abhinaya's Spotify Clone WebApp", {
          body: data.message,
        })
      );
    }
  }
});
```

Project Title: Gmail Clone **Roll No. 12**

Output:



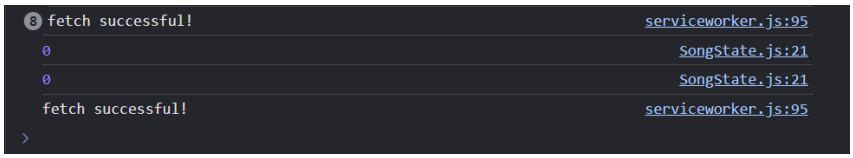


Fetch:

Code:

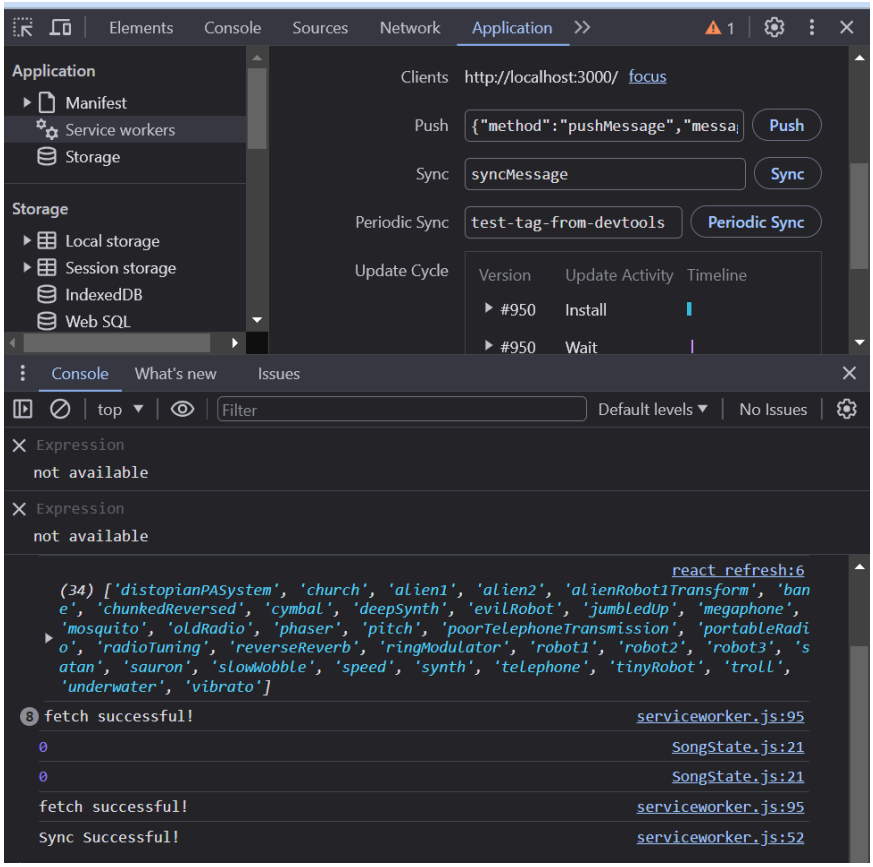
```
var checkResponse = function (request) {
  return new Promise(function (fulfill, reject) {
    fetch(request).then(function (response) {
      if (response.status !== 404) {
        fulfill(response);
      } else {
        reject();
      }
    }, reject);
  });
};
var addToCache = function (request) {
  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;
      }
    });
  });
};
self.addEventListener("fetch", function (event) {
  event.respondWith(
    checkResponse(event.request).catch(function () {
      console.log("Fetch from cache successful");
      return returnFromCache(event.request);
    })
  );
  console.log("fetch successful!");
  event.waitUntil(addToCache(event.request));
});
```

Output:



Sync:

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



Conclusion:

Understood & implemented Service worker events like fetch, sync and push for my website Spotify Clone PWA.

****

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	12
Name	Abhinaya Danda
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Abhinaya Danda

D15A Roll No:12

PWA Experiment No: 10

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

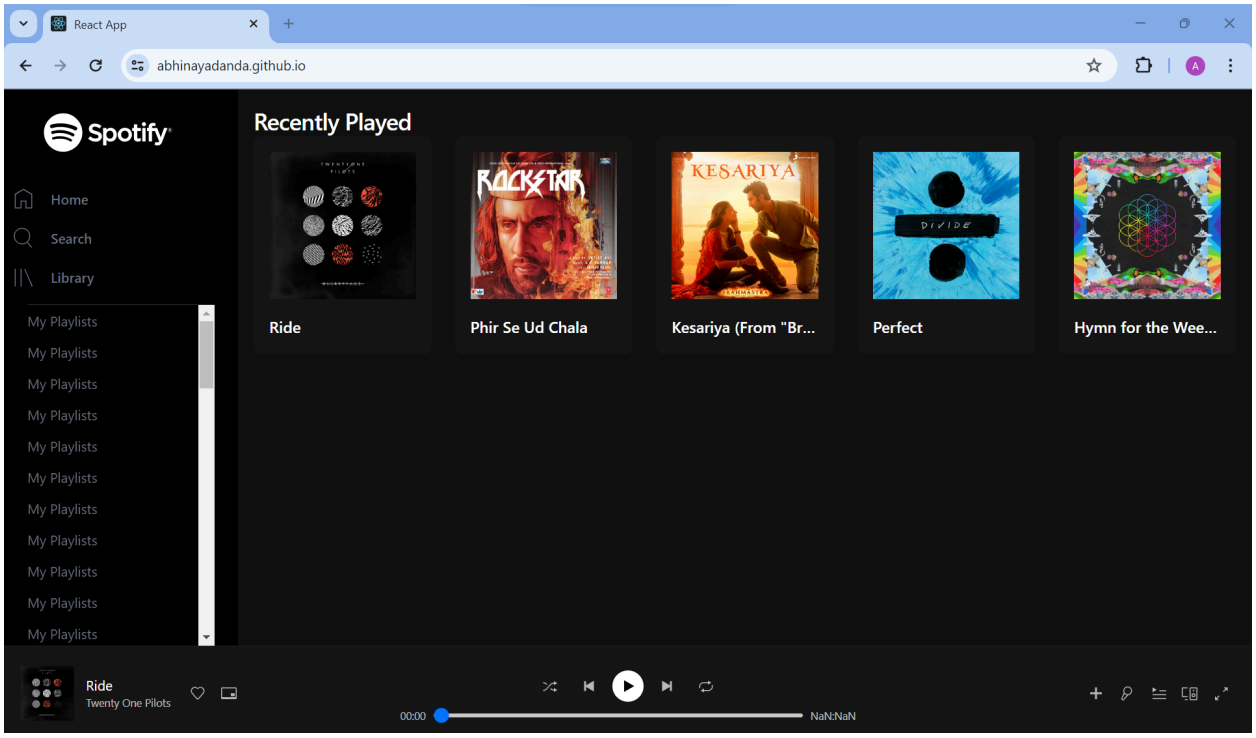
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

Project Title: Gmail Clone **Roll No. 12**

Link to our GitHub repository: https://github.com/Abhinayadanda/SpotifyClone_PWA

Site Published to github pages: https://abhinayadanda.github.io/SpotifyClone_PWA/

Github Screenshot:



Conclusion:

Deployed my React Web App to Github pages successfully.

****

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	12
Name	Abhinaya Danda
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

Abhinaya Danda

D15A Roll No:12

PWA

Experiment No: 11

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

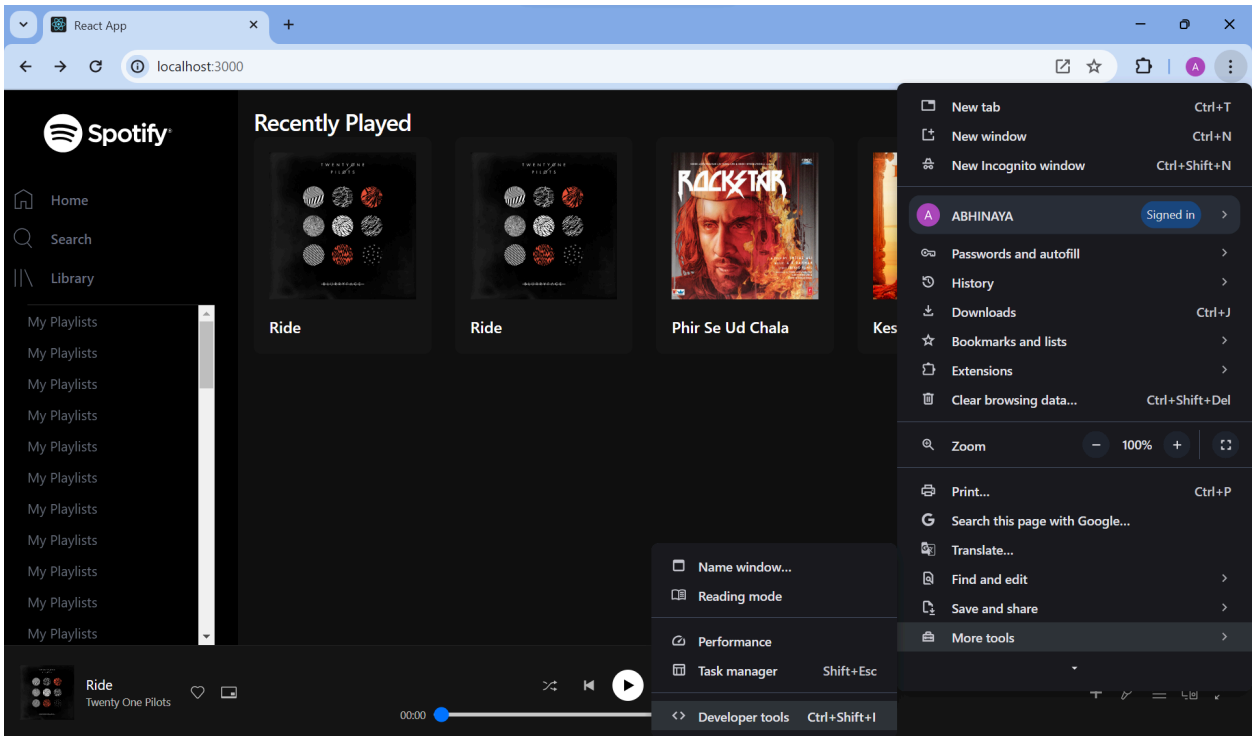
Google Lighthouse has the option of running the Audit for Desktop as well as mobile version of your page(s). The top metrics that will be measured in the Audit are:

1. 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.
2. 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.
3. 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.
4. 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

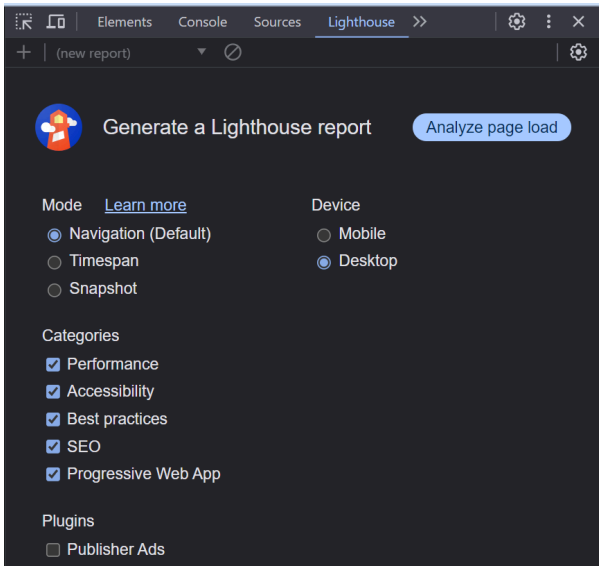
Project Title: Gmail Clone **Roll No. 12**

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.

Click on Developer tools:



Click on lighthouse:



Project Title: Gmail Clone **Roll No. 12**

Error:

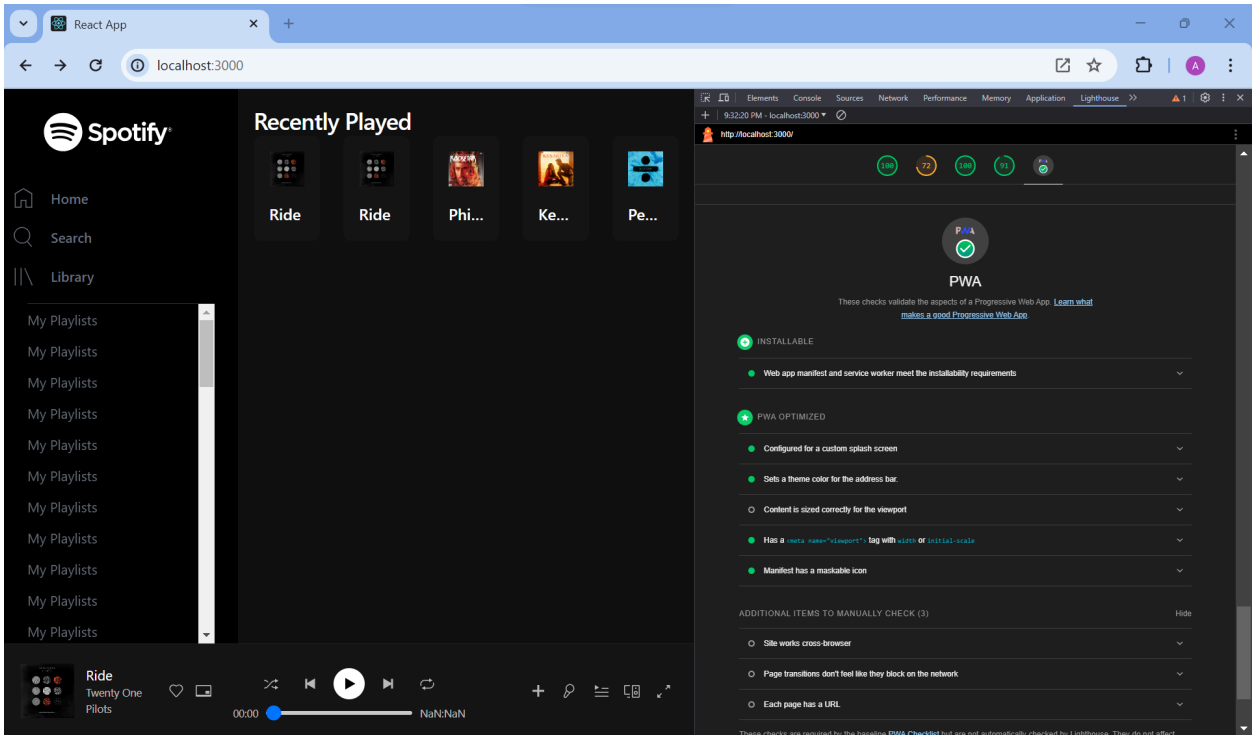
Changes to the code:

manifest.json:

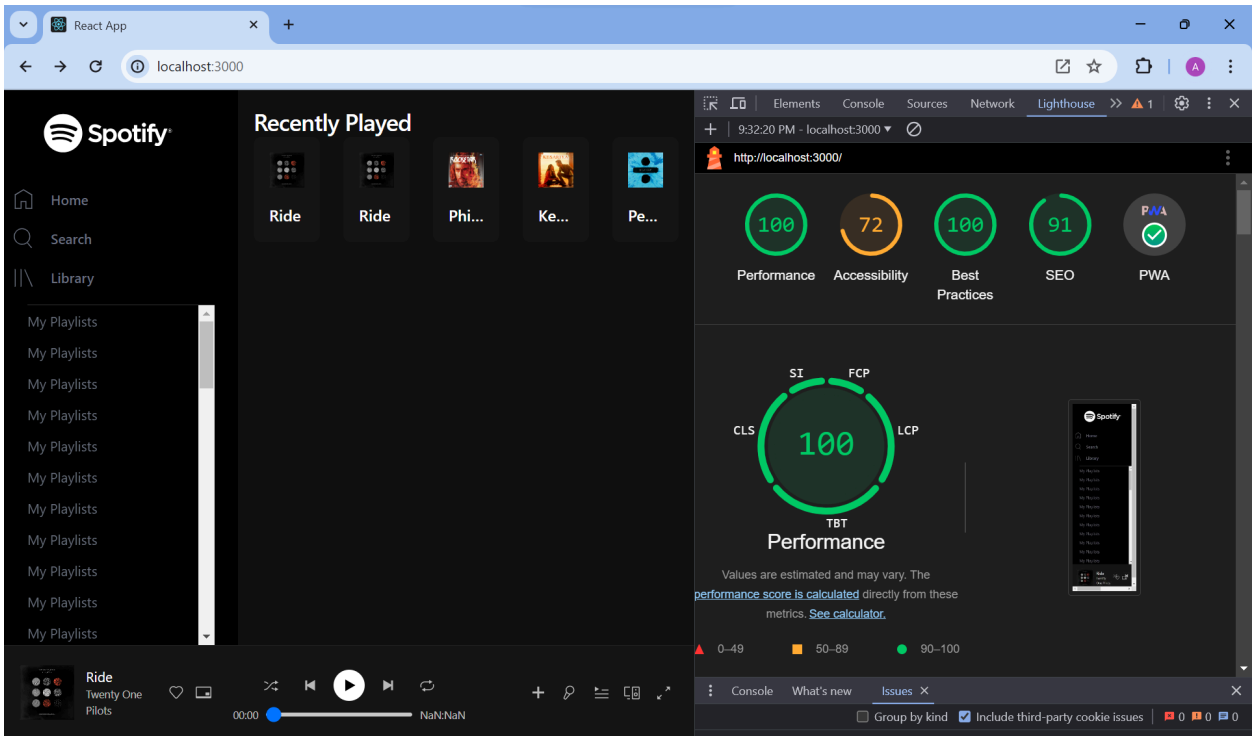
```
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192",
      "purpose": "any"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512",
      "purpose": "maskable"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

Project Title: Gmail Clone **Roll No. 12**

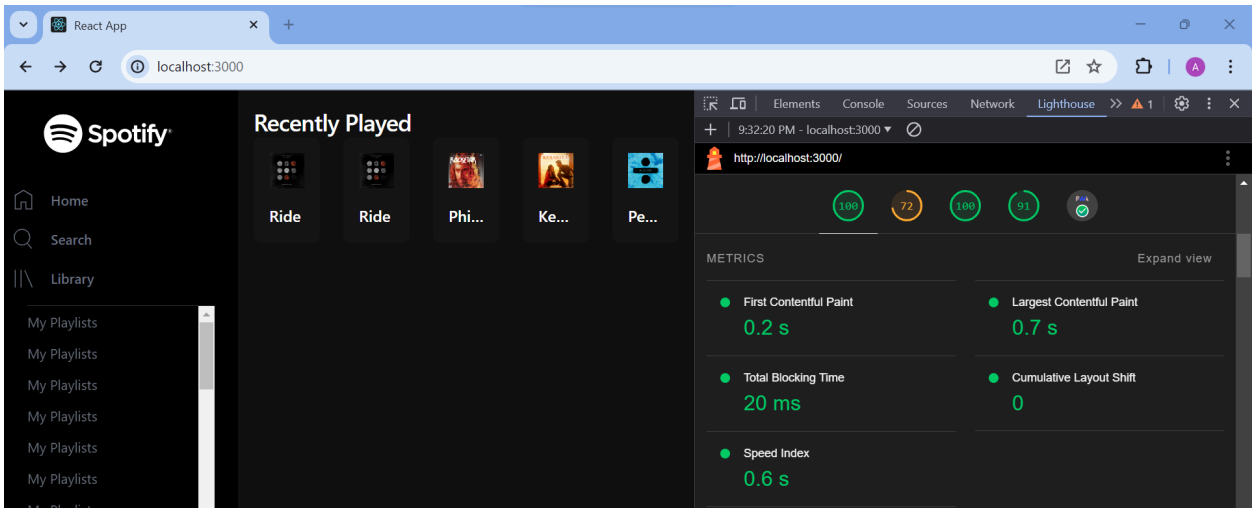
PWA Enabled after making above changes:



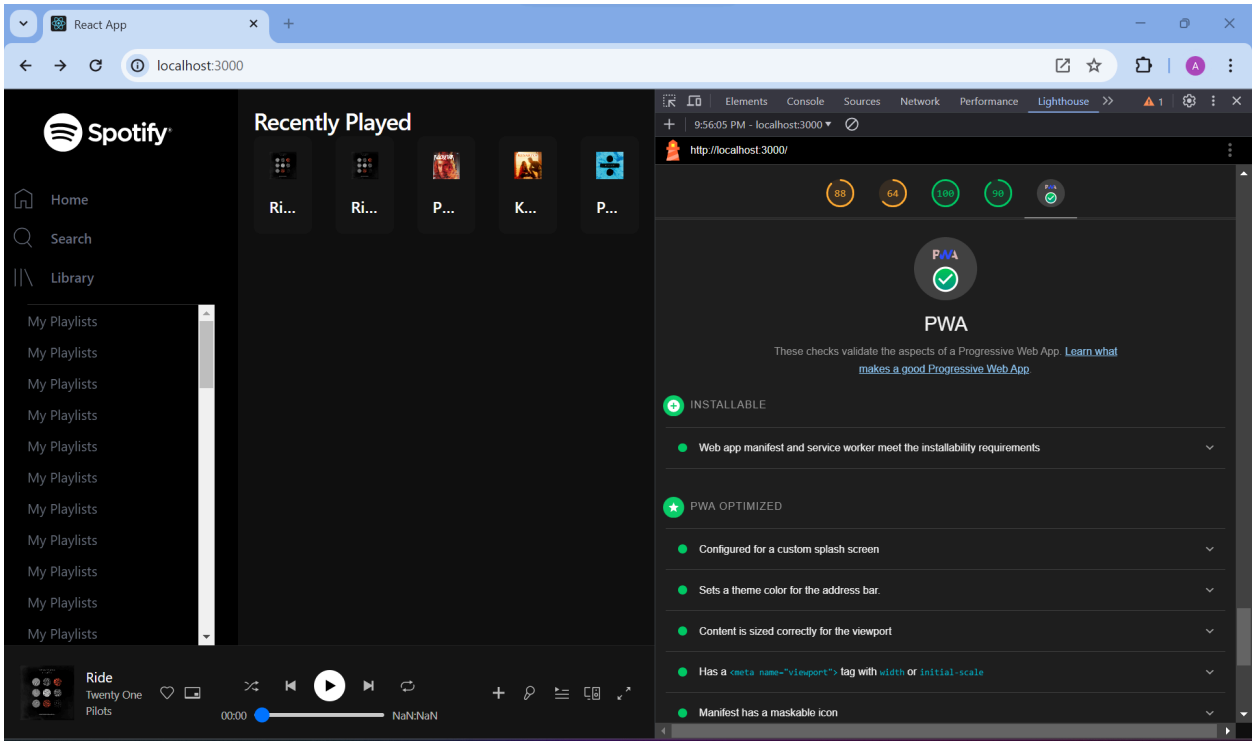
Output:



Project Title: Gmail Clone **Roll No. 12**



For Mobile:



Conclusion:

Understood the use of Lighthouse PWA Analysis Tool in analysing the website.

****

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.	12
Name	Abhinaya Danda
Class	D15A
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:	5

//

MAD - PWA Lab

Assignment - 01

ABHINAYA DANDA
D15A
Roll No : (12)

Q.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.

→ Key features of Flutter :

- Single Codebase for Multiple Platforms : Flutter allows developers to write code once and deploy it on both iOS and Android platforms.
- Hot Reload : Results of code changes can be seen instantly without restarting entire application.
- Rich Set of Widgets : Flutter comes with a comprehensive set of customizable widgets that help in creating responsive & native-like user interface.
- Dart Programming Language : Dart is a modern, object-oriented language that is easy to learn and offers features like strong typing, just-in-time compilation for optimized performance.

Advantages of Using Flutter :

1) Faster Development
2) Cost - Efficiency.
3) Consistent UI Across Platforms.
4) Ease of Learning.
5) Customization & Flexibility.

Differences from Traditional Approaches:
- No WebView Dependency
- Widget - Based Architecture
- Hot Reload.
- Uniform Development Environment.

Populasity:
- Backed by Google.
- Productivity & Time Savings.
- Growing Community Support.
- Versatility Across Platforms.

Q.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.

→ Widget Tree:

1) Root Widget: At the top of widget tree is the root widget, often a MaterialApp.
2) Parent & Child Widgets: The parent widget contains one or more child widgets, and each child widget may have its own children.
3) Leaf Nodes: Basic UI elements like Text, Image, Container, or Button.
4) Composition: Combining different widgets to build more intricate UI components.

Widget Composition:

1) Container : Versatile widget that can contain other widgets. It allows customization of dimensions, padding, margin & decoration.

e.g
```
Container(
    padding: EdgeInsets.all(16.0),
    child: Text('Hello, Flutter!'),
)
```

2) Column & Row: Allows Vertical & Horizontal arrangement of child widgets.

```
Column(
    children: [
        Text('Item 1'),
        Text('Item 2'),
        Text('Item 3'),
    ],
)
```

3) List View: Used to create scrollable list of widgets.

```
ListView(
    children: [
        ListTile(title: Text('Item 1')),
        ListTile(title: Text('Item 2')),
        ListTile(title: Text('Item 3')),
    ],
)
```

4) AppBar : Represents the top app bar that can include a title, actions & other components.

e.g:
```
AppBar (
    title : Text ('My App'),
    actions : [
        IconButton (
            icon : Icon (Icons.settings),
            onPressed : () {
                // handle button click
            },
        ),
    ],
)
```

Q.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.

→ In Flutter, UI is a function of the application state, and any changes to the state should trigger a rebuild of the UI.

- Reactivity – Respond to changes.
- Maintainability – helps in organizing & maintaining code.

- Performance - Inefficient state management can lead to unnecessary UI rebuilds, affecting performance.

State Management Approaches in Flutter:

1) setState:- Simplest form of state management in Flutter. It belongs to Stateful Widget class & is suitable for managing local state of a widget.

Suitable Scenario: Suitable for managing local state within a widget when the state changes only effect particular widget.

2) Provider:- Provider, a popular state management solution in Flutter, simple way to manage and access state across the entire application.

Suitable Scenario: Provider is suitable for medium-sized to large applications where state needs to be shared across multiple widgets.

3) Riverpod:- Riverpod is an extension of Provider that introduces a more modular and flexible approach to state management. It focuses on dependency injection & provider creation.

Suitable Scenario: For larger & more complex applications where fine-grained control over dependencies & state management.

Comparison:

SetState :- Simple easy to use :
- Suitable for small widgets with local state

Provider :- Efficient & easy to set up
- Suitable for medium to large application with shared state.

Riverpod :- Offers more flexibility & control.
- Suitable for large & complex application with complex state management needs.

Q.4] Firebase Integration in Flutter : Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Flutter as a backend solution. Highlight Firebase services commonly used in Flutter development & provide a brief overview of how data synchronization is achieved.

→ Steps for Firebase Integration in Flutter:

1) Create a Firebase Project : Go to Firebase Console, Create new project & follow setup instructions.
2) Add Flutter App to the Project : Register your app in project.
3) Download & Add the Configuration Files : download 'google-services.json' file for Android or from Console.

4) Add Dependencies: Update 'pubspec.yaml' file to include the necessary Firebase dependencies, common dependencies include 'Firebase_core' for core Firebase functionality and additional libraries for specific services like Firestore or Authentication.

pubspec.yaml:

```
dependencies:
    firebase_core: ^latest_version
    firebase_auth: ^latest_version.
    cloud_firestore: ^latest_version.
```

5) Initialize Firebase: Initialize firebase in your Flutter app. This is typically done in 'main.dart' file.

6) Use Firebase Services: By importing necessary packages.

Benefits of using Firebase as a Backend solution:

- Real-time Database.
- Authentication.
- Cloud Functions
- Hosting
- Cloud Storage
- Analytics & Crash Reporting.

Commonly used Firebase Services in Flutter development:

- Firestore - NoSQL database
- Authentication - email/password, Google Sign-In.
- Cloud Functions - HTTPs requests.
- Cloud Messaging (FCM) - send push notifications.

Data Synchronization in Firestore:

- When we subscribe to a document or a collection in Firestore, the app is notified of any changes, additions, or deletions in real-time. This is done through the 'onSnapshot' event listener in Flutter.

Firebase provides a powerful and scalable backend solutions for Flutter application, offering a range of services that simplify development and enhance overall user experience.

***

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.	12
Name	Abhinaya Danda
Class	D15A
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:	4