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 **PRATHAMESH PALVE** of **D15A/D15B** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2024-2025.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Project Title: ChatGPT **Roll No. 31**

Name of the Course : MAD & PWA Lab **Course Code :** ITL604

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

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers : Mrs. Kajal Joseph.

Email : kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: ChatGPT **Roll No. 31**

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			
2.	To design Flutter UI by including common widgets.	LO2			
3.	To include icons, images, fonts in Flutter app	LO2			
4.	To create an interactive Form using form widget	LO2			
5.	To apply navigation, routing and gestures in Flutter App	LO2			
6.	To Connect Flutter UI with fireBase database	LO3			
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4			
8.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA	LO5			
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5			
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5			
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6			
12.	Assignment-1	LO1,LO2 ,LO3			
13.	Assignment-2	LO4,LO5 ,LO6			

MAD & PWA Lab
Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

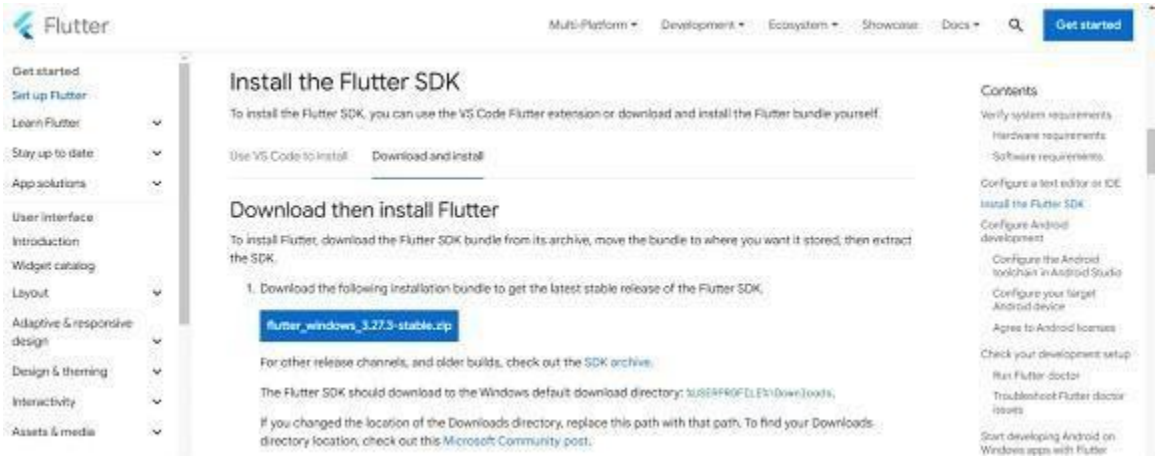
MPL Experiment 1

Name: Prathamesh Palve **Class:** D15A **Roll no:31**

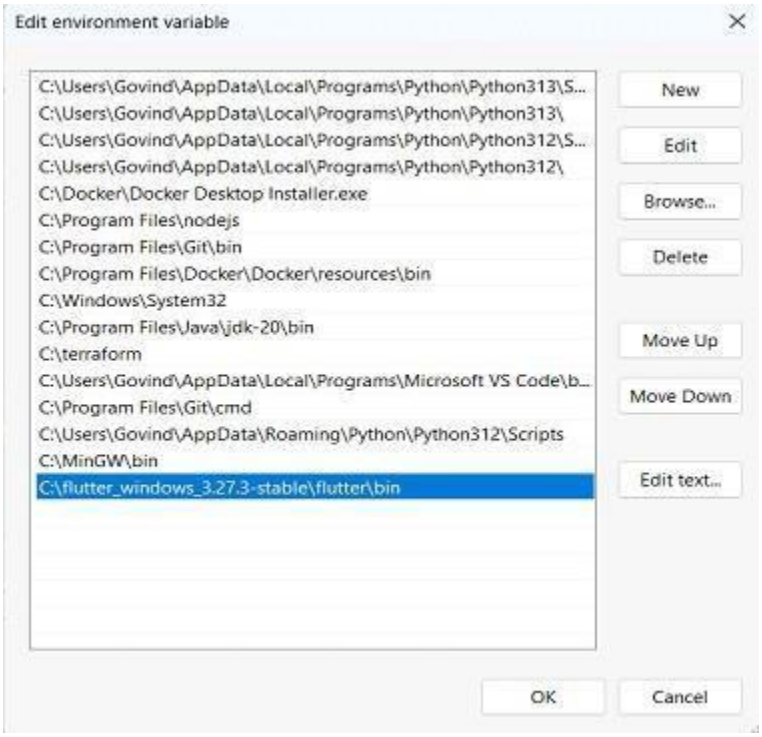
Aim: Installation and Configuration of Flutter Environment.

Step 1: Install Flutter

- Download Flutter SDK from the official Flutter website (https://flutter.dev/).
- Download the Flutter SDK for your operating system (Windows, macOS, or Linux).



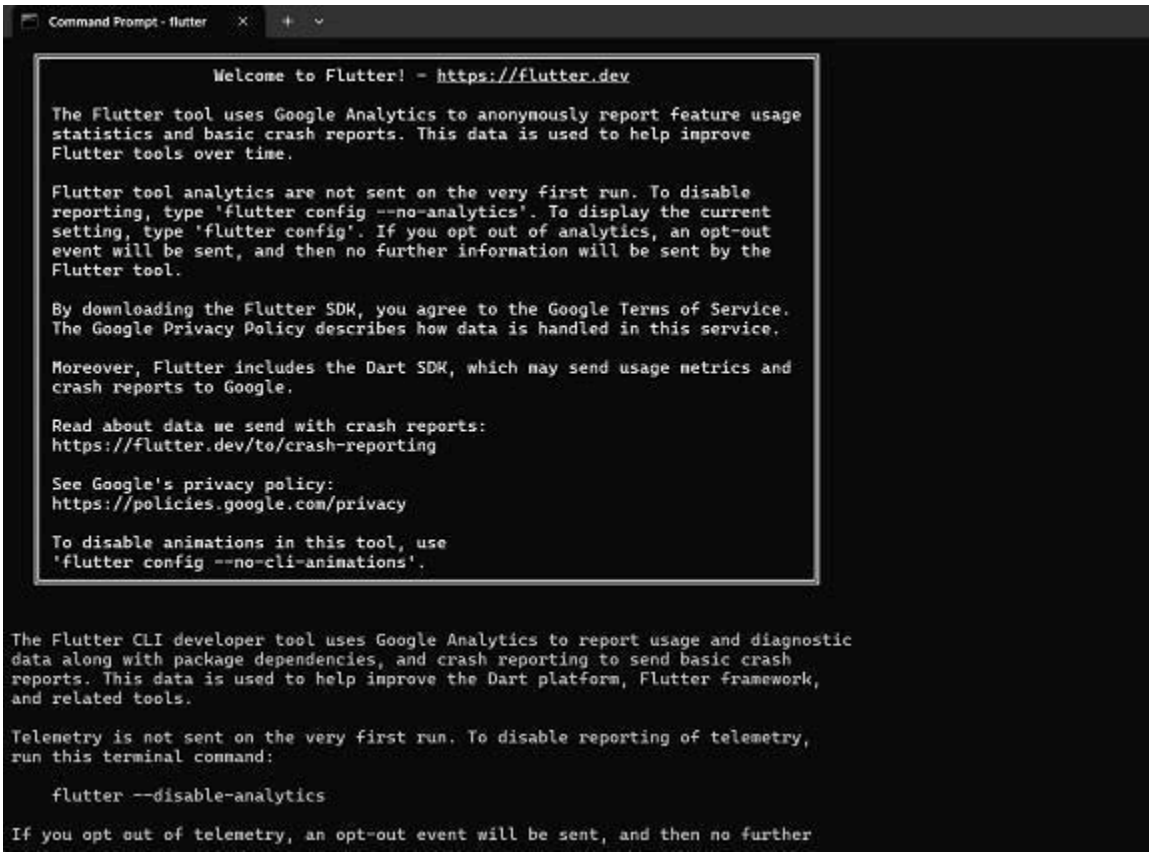
- Extract the downloaded zip file to a preferred location on your computer (e.g., C:\src\flutter for Windows).
- Add Flutter to the PATH
- Locate the flutter\bin directory in the extracted Flutter folder.
- Add this directory to your system's PATH environment variable.



Verify the Installation

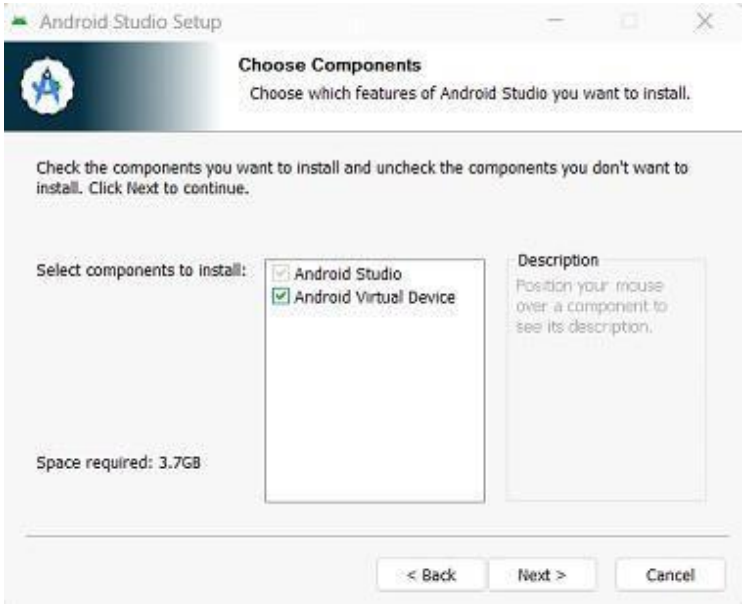
- Open a terminal or command prompt.
- Run the command: **flutter** and **flutter doctor**.

```
Microsoft Windows [Version 10.0.22631.4751]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.27.1, on Microsoft Windows [Version 10.0.22631.4751], locale en-IN)
[✗] Windows Version (the doctor check crashed)
    X Due to an error, the doctor check did not complete. If the error message below is not helpful, please let us know
      about this issue at https://github.com/flutter/flutter/issues.
    X ProcessException: Failed to find "powershell" in the search path.
        Command: powershell
[✓] Android toolchain - develop for Android devices (Android SDK version 34.0.0)
[✓] 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 2023.3)
[✓] VS Code (version 1.96.4)
[✓] Connected device (4 available)
[✓] Network resources

! Doctor found issues in 2 categories.

C:\Users\User>
```



Step 2: Install Android Studio

- Download Android Studio
- Go to the Android Studio website. (https://developer.android.com/studio)
- Download the installer for your operating system ((Windows, macOS, or Linux)).

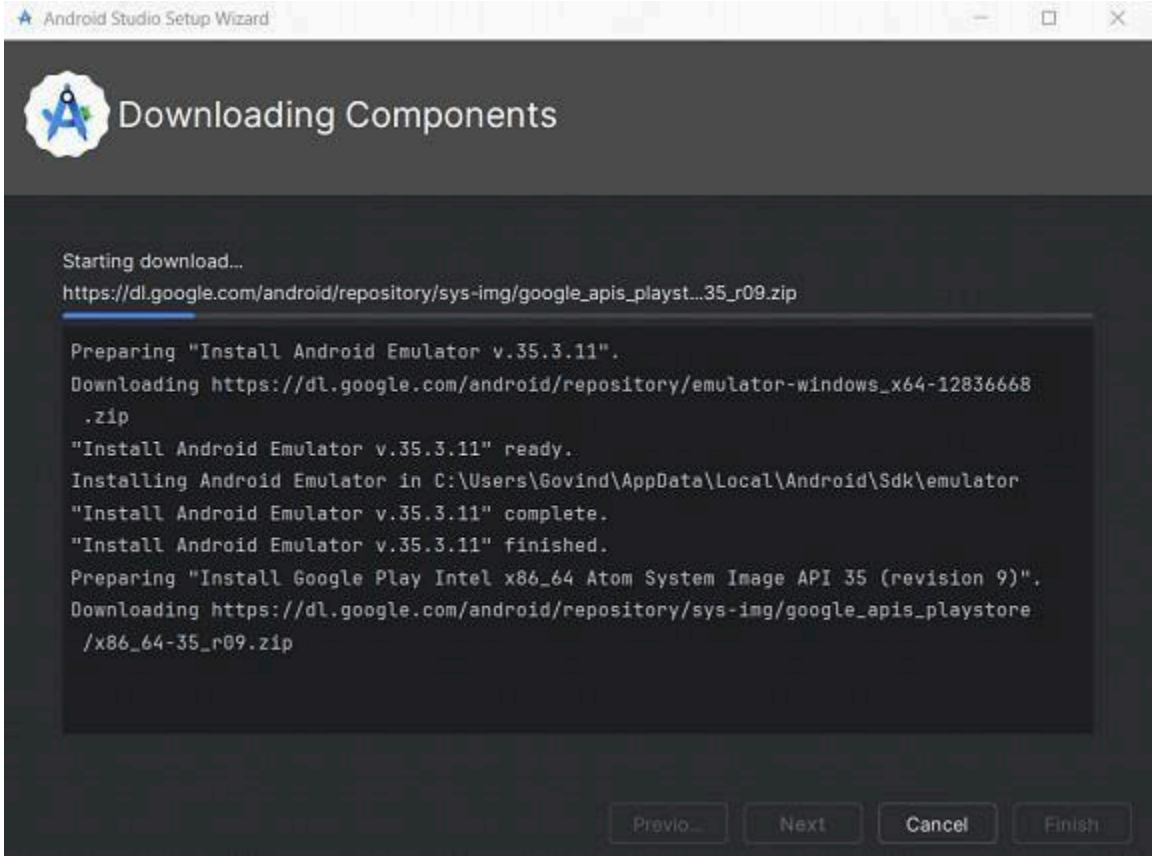


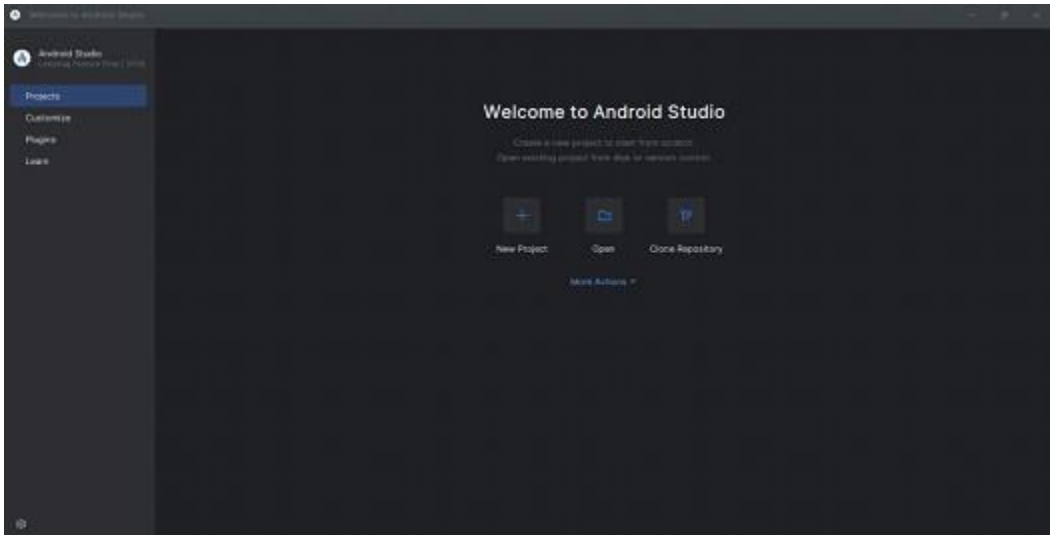
- Run the installer and follow the setup wizard.
- Choose the standard installation option.

- Install Android SDK Tools
- Open Android Studio.

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

- Select the latest Android API level.
- Ensure "Android SDK Platform" and "Android Virtual Device (AVD)" are selected.
- Click "Apply" and wait for the components to install.

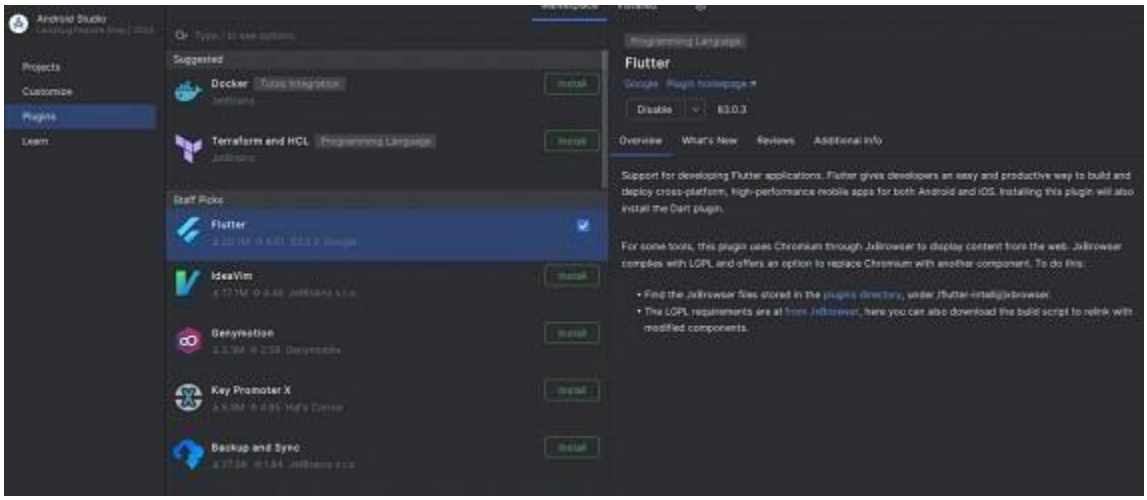




Step 3: Connect Flutter with Android Studio

- Install Flutter and Dart Plugins
- Open Android Studio. Go to File > Settings (Windows/Linux) > Plugins.

- Search for "Flutter" and click "Install." Dart will be installed automatically.
- Restart Android Studio.



Step 4: Create a New Flutter Project

- Click on New Flutter Project.
- Enter project details and select the Flutter SDK path.
- Click "Finish" to create the project.
- Connect the USB to the device and run the flutter application.
  NOTE: In your mobile device, make sure the USB debugging is turned on.

Code:

```dart
class HomePage extends StatelessWidget {
  const HomePage({super.key});

  @override
  Codeium: Refactor | Explain | Generate Function Comment | ×
  Widget build(BuildContext context) {
    final User? user = FirebaseAuth.instance.currentUser;

    // If user is not logged in, redirect to login page
    if (user == null) {
      Future.delayed(Duration.zero, () {
        Navigator.pushReplacement(
          context,
          MaterialPageRoute(builder: (context) => const LoginPage()),
        );
      }); // Future.delayed
      return const Scaffold(
        body: Center(child: CircularProgressIndicator()), // Loading spinner
      ); // Scaffold
    }

    return Scaffold(
      appBar: AppBar(
        title: const Text("Welcome"),
        actions: [
          IconButton(
            icon: const Icon(Icons.logout),
            onPressed: () async {
              await FirebaseAuth.instance.signOut();
              Navigator.pushReplacement(
                context,
                MaterialPageRoute(builder: (context) => const LoginPage()),
              );
            },
          ), // IconButton
        ],
      ), // AppBar
      body: const Center(
        child: Text(
          "Hi im prathamesh !",
          style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
        ), // Text
      ), // Center
    ); // Scaffold
  }
}
```

Conclusion: Hello message , is successfully run on the flutter app.

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

MPL Experiment 2

Name: Prathamesh Palve **Class:** D15A **Roll no:**31

Aim: To design Flutter UI by including common widgets.

Theory:

Designing a **ChatGPT-like UI** in Flutter involves utilizing various widgets to create a clean, structured, and interactive chat interface. Layout widgets like **Container, Column, Row, and ListView** help organize chat bubbles, while **GestureDetector, InkWell, and TextField** enhance user interaction.

For displaying chat messages, widgets like **Text, Card, and ListView** are essential. Navigation widgets such as **Drawer, AppBar, and Navigator** ensure smooth transitions between different sections of the app. Managing state using **setState, Provider, or Riverpod** ensures real-time updates to chat messages and responses.

Key UI Components for a ChatGPT App

ListView (reversed) – Displays chat messages in a conversation format from bottom to top.
TextField with a Send Button – Allows users to input messages.
ChatBubble Widget – Custom-designed widget to differentiate user and AI responses.
Bottom Navigation or Drawer – Provides easy navigation to different sections like settings, saved chats, and history.

Steps:

Step 1: Create a new Flutter project or open an existing one.

Set up a Flutter project and add necessary dependencies like `firebase_auth` and `cloud_firestore` for chat storage.

Step 2: Design the layout using Scaffold, incorporating AppBar and Drawer.

- The **AppBar** contains options like settings and premium subscriptions.
- The **Drawer (LeftSlider)** helps navigate between different chat conversations.

Step 3: Implement ListView (with reverse: true) for displaying chat messages.

- Fetch messages from Firestore using a **StreamBuilder**.
- Display chat messages using a **ChatBubble widget**.
- Ensure new messages appear at the bottom (like ChatGPT's UI).

Step 4: Add a TextField and a Send Button.

- Use a `TextField` for message input.
- Implement an **onSend function** that:
    - Saves the user's message in Firestore.
    - Calls an API (e.g., Hugging Face or a local model) for AI-generated responses.
    - Stores the AI's response in Firestore.

Step 5: Use ChatBubble Widgets to Differentiate User and AI Responses.

- User messages appear aligned to the right.

- AI responses appear aligned to the left.

Step 6: Implement a Scroll Controller to Auto-Scroll New Messages.

- Ensure the chat screen scrolls to the bottom when a new message arrives.
- Set `reverse: true` in `ListView` to display messages from bottom to top.

Step 7: Manage State Using setState or Provider.

- Use `setState` for simple state management.
- For complex applications, integrate **Provider or Riverpod** to handle chat state across multiple screens

Code:

```
//home_.dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import '../services/api_service.dart';
import '../services/firestore_service.dart';
import '../widgets/chat_bubble.dart';
import '../widgets/empty_state.dart';
import '../widgets/bottom_bar.dart';
import 'setting.dart';
import 'left_slidder.dart';

class HomeScreen extends StatefulWidget {
 final String chatId; // Accepts chatId dynamically

 HomeScreen({required this.chatId});

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

class _HomeScreenState extends State<HomeScreen> {
 final TextEditingController _messageController = TextEditingController();
 final FirebaseAuth _auth = FirebaseAuth.instance;
 final FirestoreService _firestoreService = FirestoreService();
 final ApiService _apiService = ApiService();
 final ScrollController _scrollController = ScrollController();

 void _sendMessage() async {
   final user = _auth.currentUser;
   if (user == null || _messageController.text.trim().isEmpty) return;

   String userId = user.uid;
   String chatId = widget.chatId;
   String userMessage = _messageController.text.trim();

   try {
     // Store user message in Firestore
     await _firestoreService.addMessage(userId, chatId, userMessage, true);

     setState(() {
       _messageController.clear();
     });

     // Scroll to bottom after sending message
     Future.delayed(Duration(milliseconds: 300), () {
       _scrollToBottom();
     });

     // Get response from Hugging Face API
     String aiResponse = await _apiService.getHuggingFaceResponse(userMessage);

     // Store AI response in Firestore
     await _firestoreService.addMessage(userId, chatId, aiResponse, false);

     print("✅ AI Response stored successfully!");
   } catch (e) {
     print("❌ Error sending message: $e");
   }
```

```dart
}

void _scrollToBottom() {
  if (_scrollController.hasClients) {
    _scrollController.animateTo(
      0, // Since we reversed ListView, scroll to position 0
      duration: Duration(milliseconds: 300),
      curve: Curves.easeOut,
    );
  }
}

@override
Widget build(BuildContext context) {
  final user = _auth.currentUser;
  if (user == null) {
    return Scaffold(
      body: Center(child: Text("User not logged in")),
    );
  }

  return Scaffold(
    backgroundColor: Colors.black,
    appBar: AppBar(
      backgroundColor: Colors.black,
      elevation: 0,
      leading: Builder(
        builder: (context) => IconButton(
          icon: Icon(Icons.menu, color: Colors.white),
          onPressed: () {
            Scaffold.of(context).openDrawer();
          },
        ),
      ),
      actions: [
        TextButton(
          onPressed: () {},
          child: Text("Get Plus ✨", style: TextStyle(color: Colors.white)),
        ),
        IconButton(
          icon: Icon(Icons.more_vert, color: Colors.white),
          onPressed: () {},
        ),
      ],
    ),
    drawer: LeftSlider(currentChatId: widget.chatId),
    body: Column(
      children: [
        Expanded(
          child: StreamBuilder<QuerySnapshot>(
            stream: _firestoreService.getMessages(user.uid, widget.chatId),
            builder: (context, snapshot) {
              if (snapshot.connectionState == ConnectionState.waiting) {
                return Center(child: CircularProgressIndicator());
              }

              if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
                return EmptyState(); // Show empty state if no messages
              }

              return ListView(
```

```
                controller: _scrollController,
                padding: EdgeInsets.all(16),
                reverse: true, // 🔥 Messages appear from bottom to top
                children: snapshot.data!.docs.map((doc) {
                  Map<String, dynamic> data = doc.data() as Map<String,
dynamic>;
                  bool isUser = data['isUser'] ?? false;
                  return ChatBubble(message: data['message'] ?? '', isUser:
isUser);
                }).toList(),
              );
            },
          ),
        ),
        BottomBar(
          messageController: _messageController,
          onSendPressed: _sendMessage,
        ),
      ],
    ),
  );
 }
}
```

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

class SettingsScreen extends StatelessWidget {
 final FirebaseAuth _auth = FirebaseAuth.instance;

 @override
 Widget build(BuildContext context) {
   final User? user = _auth.currentUser;

   return Scaffold(
     backgroundColor: Colors.black,
     appBar: AppBar(
       backgroundColor: Colors.black,
       elevation: 0,
       leading: IconButton(
         icon: Icon(Icons.arrow_back, color: Colors.white),
         onPressed: () {
           Navigator.pop(context);
```

```
        },
      ),
      title: Text("Settings", style: TextStyle(color: Colors.white)),
    ),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        children: [
          Row(
            children: [
              CircleAvatar(
                radius: 30,
                backgroundColor: Colors.blue,
                child: Text(
                  user?.email != null ? user!.email![0].toUpperCase() : "U",
                  style: TextStyle(color: Colors.white),
                ),
              ),
              SizedBox(width: 12),
              Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    user?.displayName ?? "User",
                    style: TextStyle(color: Colors.white, fontSize: 18,
fontWeight: FontWeight.bold),
                  ),
                  SizedBox(height: 4),
                  Text(
                    user?.email ?? "No email found",
                    style: TextStyle(color: Colors.white70),
                  ),
                ],
              ),
            ],
          ),
          SizedBox(height: 20),
          _buildSettingItem(Icons.email, "Email", user?.email ?? "No email
found"),
          _buildSettingItem(Icons.phone, "Phone number", user?.phoneNumber ??
"Not linked"),
          _buildSettingItem(Icons.star, "Upgrade to Plus"),
          _buildSettingItem(Icons.tune, "Customize"),
          _buildSettingItem(Icons.storage, "Data Controls"),
          _buildSettingItem(Icons.mic, "Voice"),
```

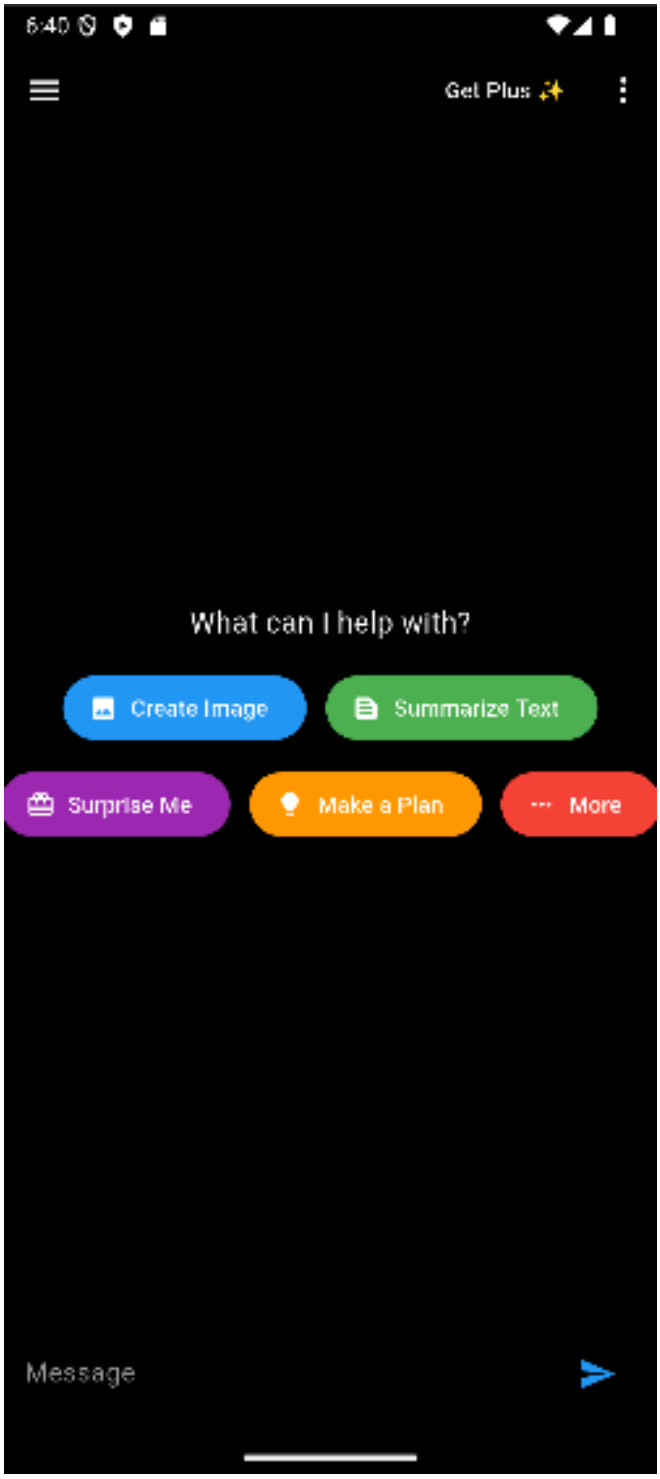
```
          _buildSettingItem(Icons.info, "About"),
          Divider(color: Colors.grey),
          ListTile(
            leading: Icon(Icons.logout, color: Colors.red),
            title: Text("Sign out", style: TextStyle(color: Colors.red)),
            onTap: () async {
              await _auth.signOut();
              Navigator.popUntil(context, (route) => route.isFirst); // Go back to login
            },
          ),
        ],
      ),
    ),
  );
 }

 Widget _buildSettingItem(IconData icon, String title, [String? subtitle]) {
   return ListTile(
     leading: Icon(icon, color: Colors.white),
     title: Text(title, style: TextStyle(color: Colors.white)),
     subtitle: subtitle != null ? Text(subtitle, style: TextStyle(color: Colors.white70)) : null,
     onTap: () {},
   );
 }
}
```

```dart
//sidebar.dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'setting.dart';
import 'home.dart';

class LeftSlider extends StatelessWidget {
 final String currentChatId; // Accepts currentChatId

 LeftSlider({required this.currentChatId});

 final FirebaseAuth _auth = FirebaseAuth.instance;
 final FirebaseFirestore _firestore = FirebaseFirestore.instance;

 void _startNewChat(BuildContext context) async {
   final user = _auth.currentUser;
   if (user == null) return;

   // Generate a new chat ID
   DocumentReference newChatRef = _firestore
       .collection('users')
       .doc(user.uid)
       .collection('chats')
       .doc();

   // Create an empty chat document
   await newChatRef.set({
     'createdAt': FieldValue.serverTimestamp(),
   });

   // Navigate to HomeScreen with the new chat ID
   Navigator.pushReplacement(
     context,
     MaterialPageRoute(
       builder: (context) => HomeScreen(chatId: newChatRef.id),
     ),
   );
 }

 @override
 Widget build(BuildContext context) {
```

```
  return Drawer(
    child: Container(
      color: Colors.black87,
      child: Column(
        children: [
          SizedBox(height: 40),
          // Search Bar
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 16),
            child: TextField(
              decoration: InputDecoration(
                hintText: "Search",
                filled: true,
                fillColor: Colors.grey[850],
                prefixIcon: Icon(Icons.search, color: Colors.white70),
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(8),
                  borderSide: BorderSide.none,
                ),
              ),
            ),
          ),
          SizedBox(height: 20),
          // New Chat Button
          ListTile(
            leading: Icon(Icons.add, color: Colors.white),
            title: Text("New Chat", style: TextStyle(color: Colors.white)),
            onTap: () => _startNewChat(context),
          ),
          Divider(color: Colors.grey),
          Expanded(
            child: StreamBuilder<QuerySnapshot>(
              stream: _firestore
                  .collection('users')
                  .doc(_auth.currentUser?.uid)
                  .collection('chats')
                  .orderBy('createdAt', descending: true)
                  .snapshots(),
              builder: (context, snapshot) {
                if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
                  return Center(
                    child: Text(
                      "No chat history",
                      style: TextStyle(color: Colors.white70),
```

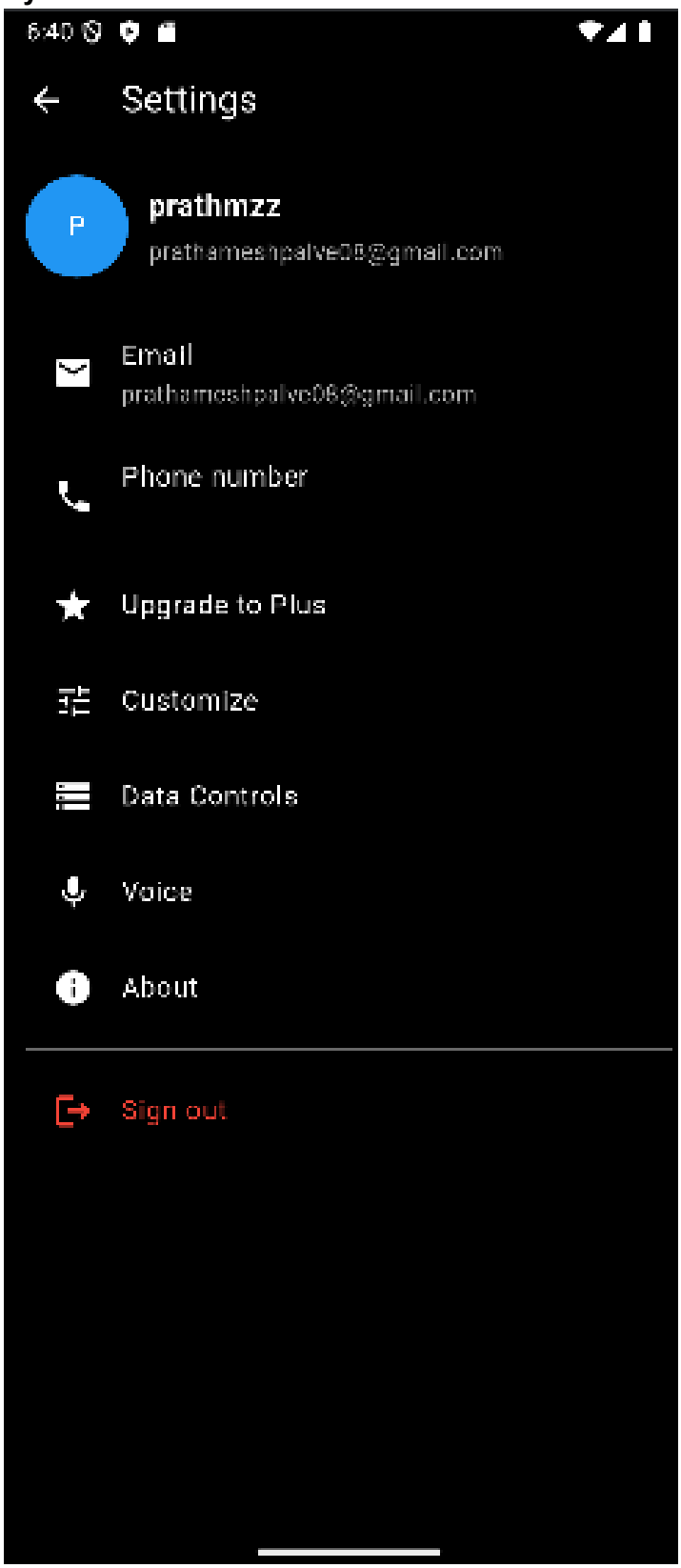
```
                    ),
                  );
                }
                return ListView(
                  children: snapshot.data!.docs.map((doc) {
                    String chatId = doc.id;
                    return ListTile(
                      title: Text(
                        "Chat ${chatId.substring(0, 8)}",
                        style: TextStyle(color: Colors.white70),
                      ),
                      selected: chatId == currentChatId, // Highlight active
chat
                      onTap: () {
                        Navigator.pushReplacement(
                          context,
                          MaterialPageRoute(
                            builder: (context) => HomeScreen(chatId: chatId),
                          ),
                        );
                      },
                    );
                  }).toList(),
                );
              },
            ),
          ),
          Divider(color: Colors.grey),
          // Profile Section with Navigation to Settings
          ListTile(
            leading: CircleAvatar(
              backgroundColor: Colors.blue,
              child: Text(_auth.currentUser?.email?.substring(0, 1) ?? "U"),
            ),
            title: Text(
              _auth.currentUser?.email ?? "User",
              style: TextStyle(color: Colors.white),
            ),
            trailing: Icon(Icons.expand_more, color: Colors.white70),
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => SettingsScreen()),
              );
            },
```

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

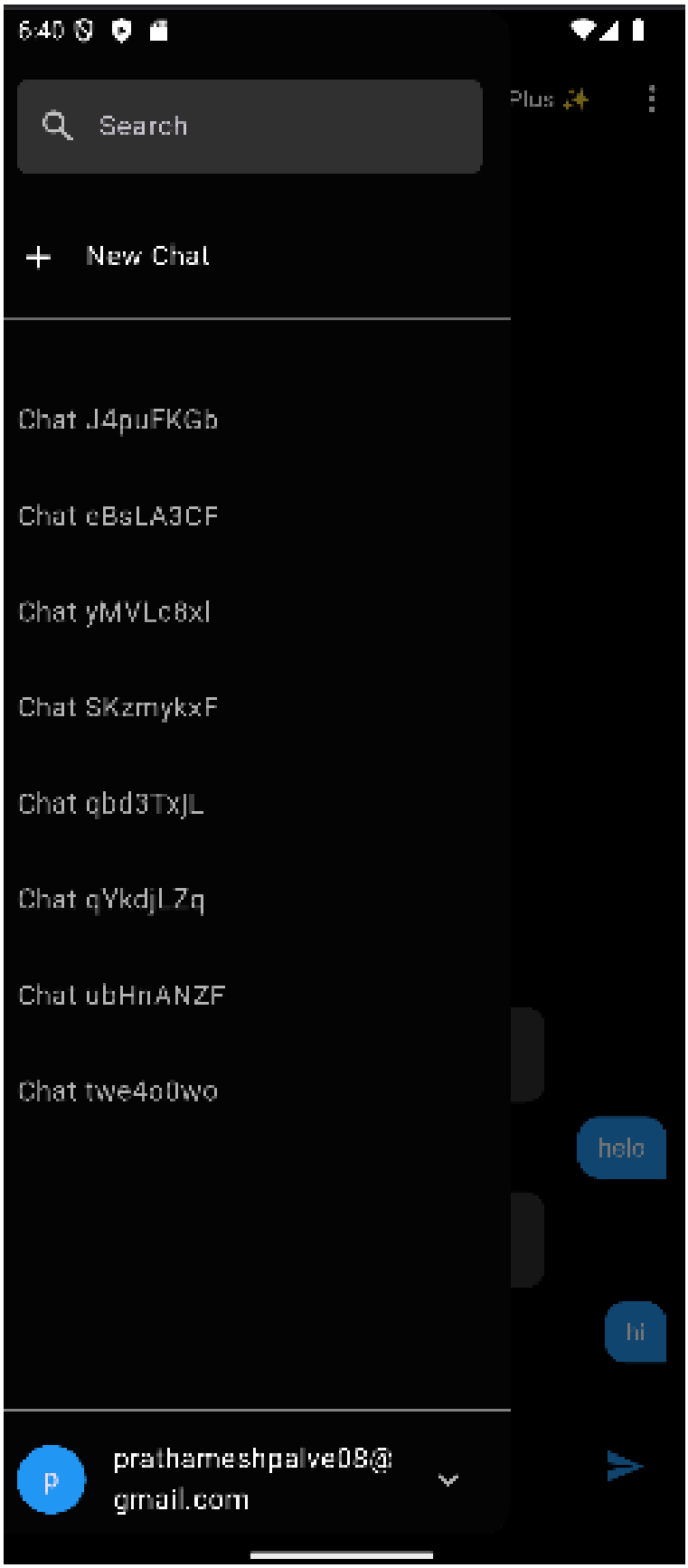


Output:
HomePage:

MyAccount:


6:40
Settings
P
prathmzz
prathameshpalve05@gmail.com
Email
prathameshpalve06@gmail.com
Phone number
Upgrade to Plus
Customize
Data Controls
Voice
About
Sign out

Sidebar:


6:40
Search
Plus
New Chat
Chat J4puFKGb
Chat cBsLA3CF
Chat yMVLc8xl
Chat SKzmykxF
Chat qbd3TxJL
Chat qVkdjl 7q
Chat ubHnANZF
Chat twe4o0wo
holo
hi
p
prathameshpalve08@
gmail.com

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

MPL Experiment 3

Name: Prathamesh Palve **Class:** D15A **Roll no:**31

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

Theory:

Incorporating icons, images, and custom fonts in a Flutter application enhances the visual appeal and improves the user experience. Flutter provides different ways to add these elements:

1. Icons:

Icons can be added using the built-in `Icons` class with the `Icon` widget. Custom icons can also be used via the `flutter_launcher_icons` package.

2. Images:

Images can be displayed in two ways:

- **From assets**: Images stored locally in the app's assets folder can be loaded using the `Image.asset()` method.
- **From the internet**: Images can be fetched dynamically from a URL using the `Image.network()` method.

Examples:

- **Loading an image from assets:**

```
Image.asset(
  'assets/images/sample.png'
  , fit: BoxFit.cover,
  width: double.infinity,
)
```

- **Loading an image from a URL:**

```
Image.network(
  'https://example.com/sample.jpg',
```

3. Fonts:

)

Custom fonts improve typography and branding. To add custom fonts, font files must be placed in the `assets/fonts/` directory and declared in `pubspec.yaml` under the `flutter` section. Using `TextStyle` with the `fontFamily` property applies the custom font to text elements.

Steps:

Step 1: Create an `assets` folder inside the project directory. Inside the `assets` folder, create an `images` folder and add the required images.

Step 2: Open `pubspec.yaml` and add the following under the `flutter` section:

`flutter:`

```
name: chatgpt_clone_08
description: "A new Flutter project."

publish_to: 'none'

version: 1.0.0+1

environment:
 sdk: ">=3.0.0 <4.0.0"

dependencies:
 flutter:
   sdk: flutter
 cupertino_icons: ^1.0.8
 http: ^1.3.0
 flutter_dotenv: ^5.1.0

 # Firebase Core and Services
 firebase_core: ^2.27.0
 firebase_auth: ^4.17.8
 cloud_firestore: ^4.15.7
 firebase_storage: ^11.6.8   # For storing user profile pictures

 # State Management & Utilities
 shared_preferences: ^2.2.3  # For caching some user data locally

 # UI and Animations
 google_fonts: ^6.2.1  # Custom fonts for better UI
 fluttertoast: ^8.2.4  # Show small notifications
 intl: ^0.19.0  # Date formatting
 lottie: ^3.1.0  # For animations (optional)

dev_dependencies:
 flutter_test:
   sdk: flutter
 flutter_lints: ^3.0.0
```

Step 3: Run the following command in the terminal to apply the changes:

`flutter pub get`

Code:

home.dart

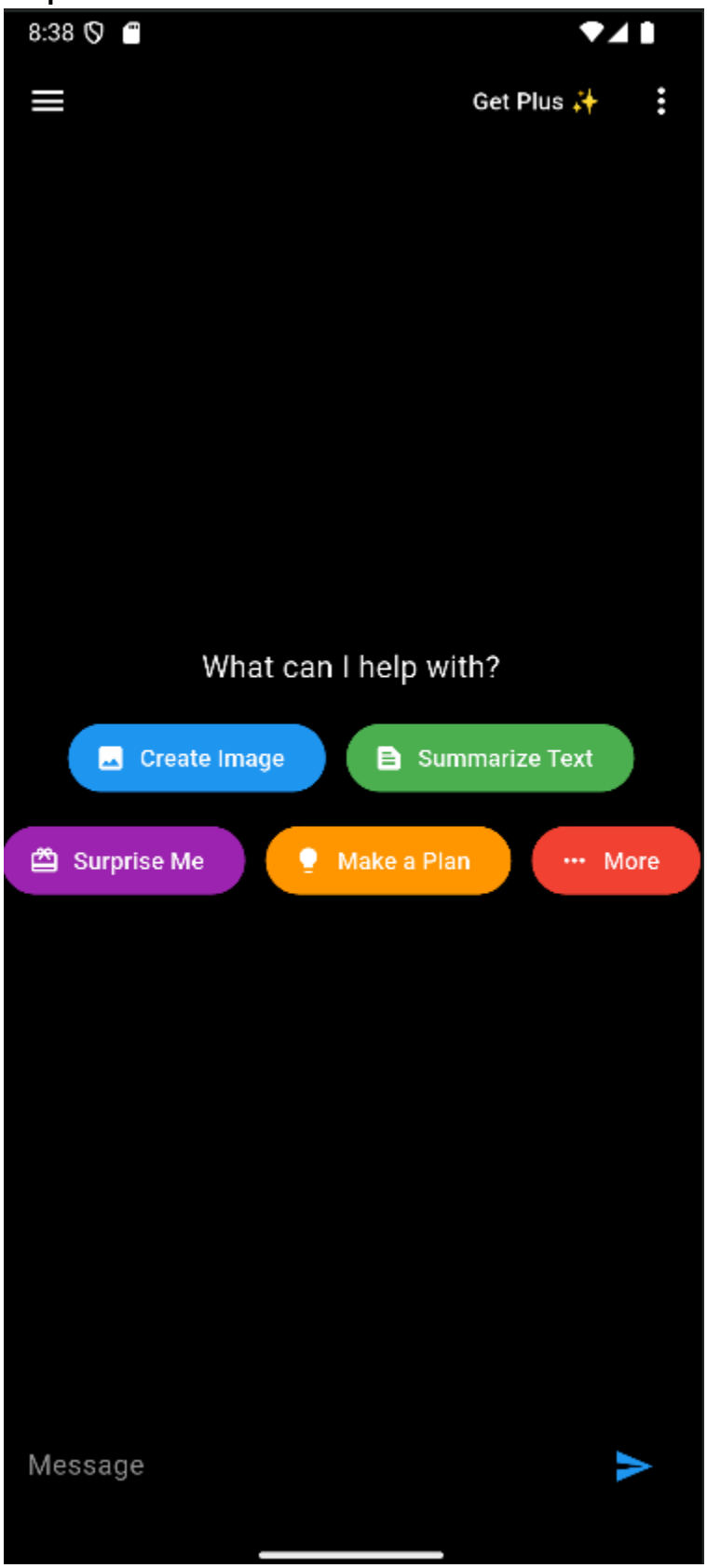
```dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import '../services/api_service.dart';
import '../services/firestore_service.dart';
import '../widgets/chat_bubble.dart';
import '../widgets/empty_state.dart';
import '../widgets/bottom_bar.dart';
import 'setting.dart';
import 'left_slidder.dart';

class HomeScreen extends StatefulWidget {
 final String chatId; // Accepts chatId dynamically

 HomeScreen({required this.chatId});

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

class _HomeScreenState extends State<HomeScreen> {
 final TextEditingController _messageController = TextEditingController();
 final FirebaseAuth _auth = FirebaseAuth.instance;
 final FirestoreService _firestoreService = FirestoreService();
 final ApiService _apiService = ApiService();
 final ScrollController _scrollController = ScrollController();

 void _sendMessage() async {
   final user = _auth.currentUser;
   if (user == null || _messageController.text.trim().isEmpty) return;

   String userId = user.uid;
   String chatId = widget.chatId;
   String userMessage = _messageController.text.trim();

   try {
     // Store user message in Firestore
     await _firestoreService.addMessage(userId, chatId, userMessage, true);

     setState(() {
       _messageController.clear();
     });
```

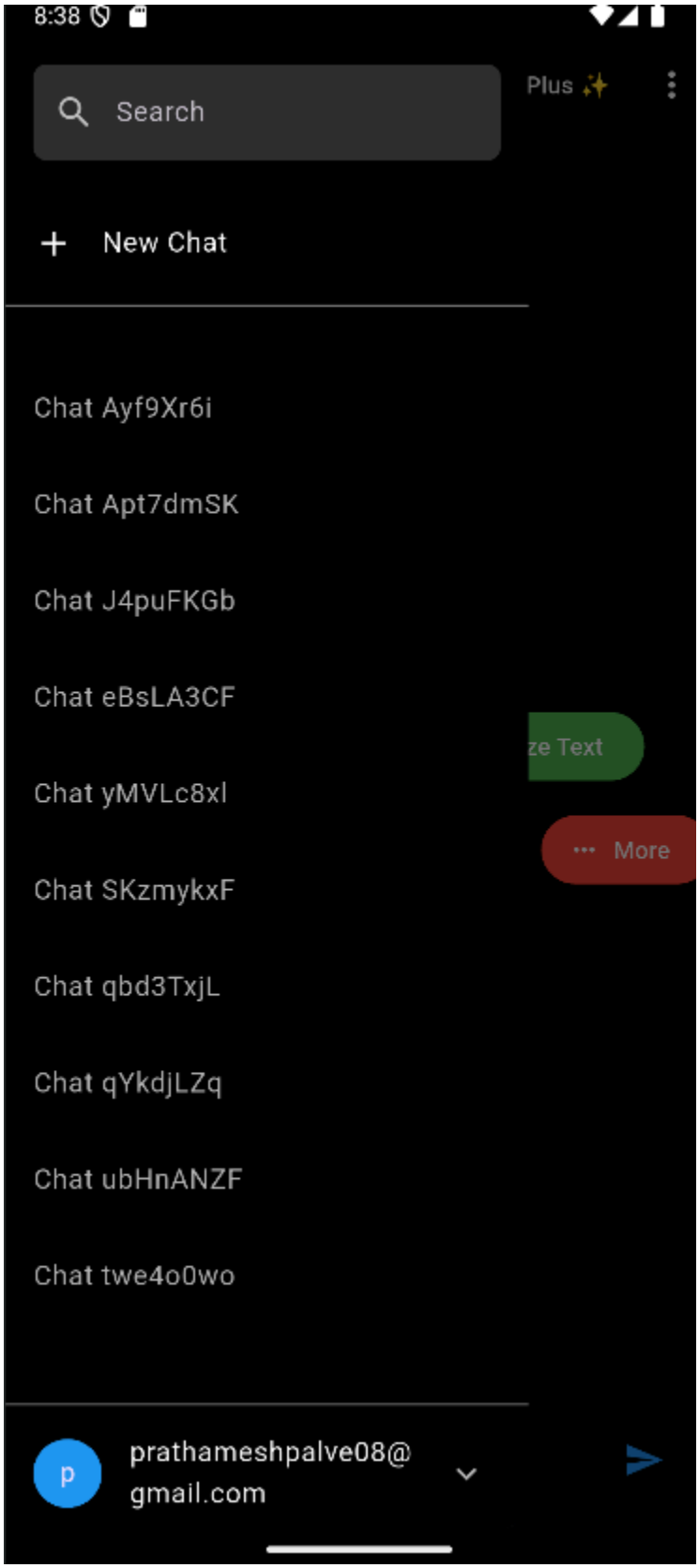
```
    // Scroll to bottom after sending message
    Future.delayed(Duration(milliseconds: 300), () {
      _scrollToBottom();
    });

    // Get response from Hugging Face API
    String aiResponse = await _apiService.getHuggingFaceResponse(userMessage);

    // Store AI response in Firestore
    await _firestoreService.addMessage(userId, chatId, aiResponse, false);

    print("✅ AI Response stored successfully!");
  } catch (e) {
    print("❌ Error sending message: $e");
  }
}

void _scrollToBottom() {
  if (_scrollController.hasClients) {
    _scrollController.animateTo(
      0, // Since we reversed ListView, scroll to position 0
      duration: Duration(milliseconds: 300),
      curve: Curves.easeOut,
    );
  }
}

@override
Widget build(BuildContext context) {
  final user = _auth.currentUser;
  if (user == null) {
    return Scaffold(
      body: Center(child: Text("User not logged in")),
    );
  }

  return Scaffold(
    backgroundColor: Colors.black,
    appBar: AppBar(
      backgroundColor: Colors.black,
      elevation: 0,
      leading: Builder(
        builder: (context) => IconButton(
          icon: Icon(Icons.menu, color: Colors.white),
          onPressed: () {
```

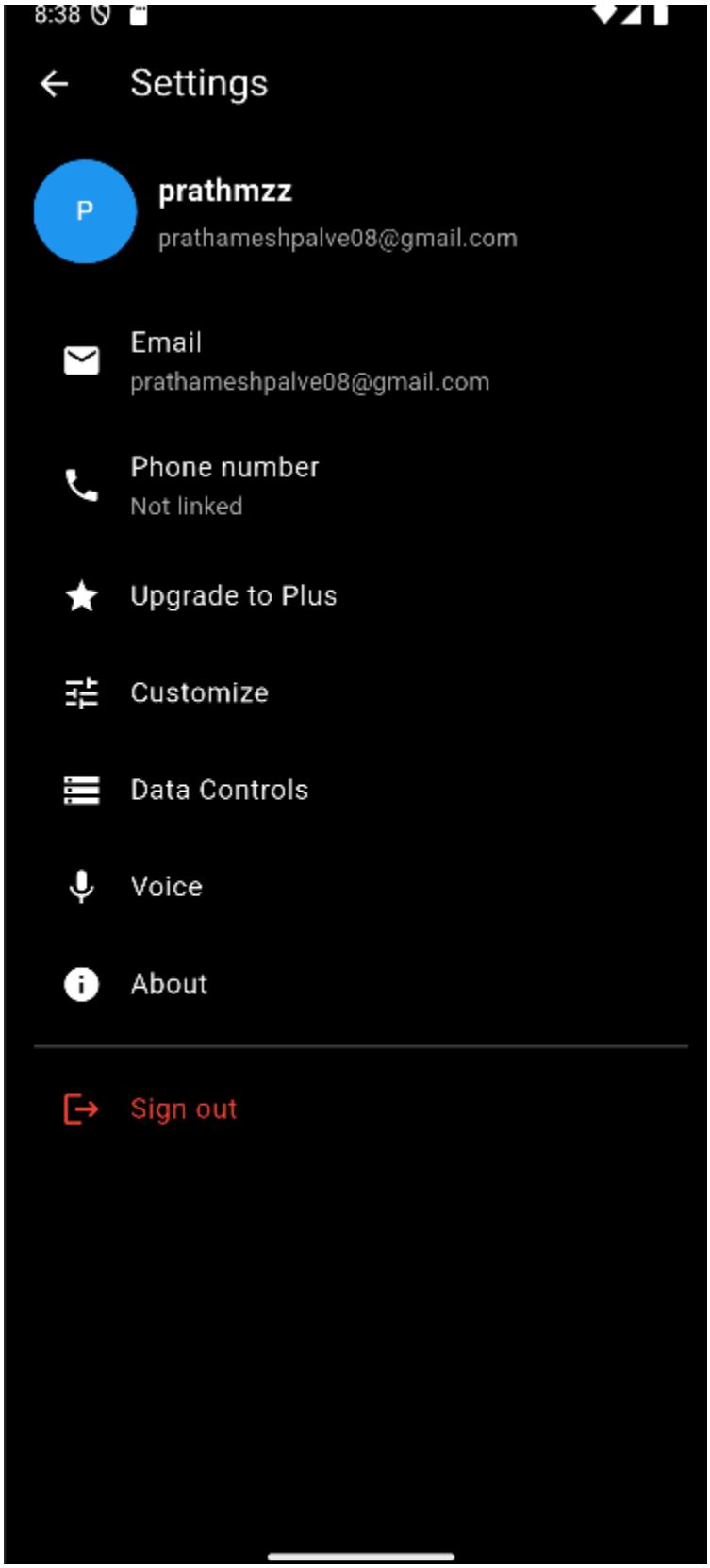
```
              Scaffold.of(context).openDrawer();
            },
          ),
        ),
        actions: [
          TextButton(
            onPressed: () {},
            child: Text("Get Plus ✨", style: TextStyle(color: Colors.white)),
          ),
          IconButton(
            icon: Icon(Icons.more_vert, color: Colors.white),
            onPressed: () {},
          ),
        ],
      ),
      drawer: LeftSlider(currentChatId: widget.chatId),
      body: Column(
        children: [
          Expanded(
            child: StreamBuilder<QuerySnapshot>(
              stream: _firestoreService.getMessages(user.uid, widget.chatId),
              builder: (context, snapshot) {
                if (snapshot.connectionState == ConnectionState.waiting) {
                  return Center(child: CircularProgressIndicator());
                }

                if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
                  return EmptyState(); // Show empty state if no messages
                }

                return ListView(
                  controller: _scrollController,
                  padding: EdgeInsets.all(16),
                  reverse: true, // 🔥 Messages appear from bottom to top
                  children: snapshot.data!.docs.map((doc) {
                    Map<String, dynamic> data = doc.data() as Map<String, dynamic>;
                    bool isUser = data['isUser'] ?? false;
                    return ChatBubble(message: data['message'] ?? '', isUser:
isUser);
                  }).toList(),
                );
              },
            ),
          ),
          BottomBar(
            messageController: _messageController,
```

```
            onSendPressed: _sendMessage,
          ),
        ],
      ),
    );
  }
}
```

Output:


8:38
Get Plus ✨
What can I help with?
Create Image
Summarize Text
Surprise Me
Make a Plan
More
Message


8:38
Search
Plus
New Chat
Chat Ayf9Xr6i
Chat Apt7dmSK
Chat J4puFKGb
Chat eBsLA3CF
ze Text
Chat yMVLc8xl
More
Chat SKzmykxF
Chat qbd3TxjL
Chat qYkdjLZq
Chat ubHnANZF
Chat twe4o0wo
p
prathameshpalve08@
gmail.com


8:38
Settings
P
prathmzz
prathameshpalve08@gmail.com
Email
prathameshpalve08@gmail.com
Phone number
Not linked
Upgrade to Plus
Customize
Data Controls
Voice
About
Sign out

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

MPL Experiment 4

Name: Prathamesh Palve **Class:** D15A **Roll no:**31

Aim: To create an interactive Form using form widget

Theory:

Creating an interactive form in Flutter requires using form-related widgets to collect and validate user input efficiently. The `Form` widget, combined with `TextFormField`, provides a structured way to manage input fields. Various input widgets like `TextField`, `DropdownButton`, `Checkbox`, `Radio`, and `Switch` allow users to enter data in different formats.

Validation and state management can be handled using the `GlobalKey<FormState>` to validate inputs before submission. Wrapping the form in a `SingleChildScrollView` ensures smooth scrolling when multiple fields are present.

A `RaisedButton` (deprecated) or `ElevatedButton` can trigger validation and submission logic. To enhance usability and create a responsive experience, proper padding, spacing, and `InputDecoration` should be applied.

Steps:

Step 1: Create a new Flutter project or open an existing one.

Step 2: Define a `Form` widget inside a StatefulWidget to manage user input.

Step 3: Use `TextFormField` for text input fields with validation logic.

Step 4: Include other input widgets such as `DropdownButton`, `Checkbox`, `Radio`, and `Switch` for additional user selections.

Step 5: Wrap the form inside a `SingleChildScrollView` to ensure smooth scrolling.

Step 6: Implement an `ElevatedButton` to trigger form validation and submission.

Step 7: Use `GlobalKey<FormState>` to manage form validation.

Code:

```dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'home.dart'; // Import HomeScreen

class AuthScreen extends StatefulWidget {
 @override
 _AuthScreenState createState() => _AuthScreenState();
```

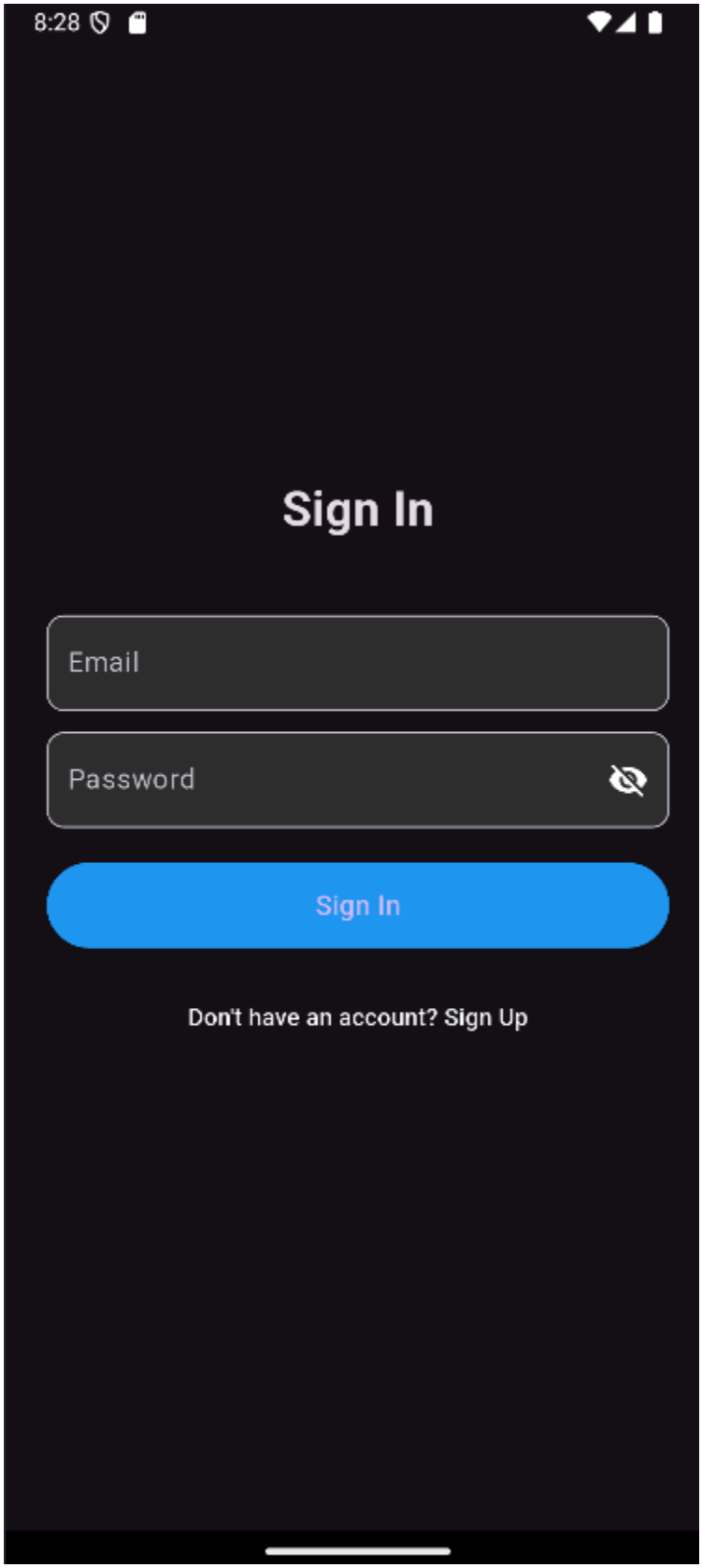
```dart
}

class _AuthScreenState extends State<AuthScreen> {
 bool isSignUp = false;
 bool isPasswordVisible = false;
 bool isLoading = false;

 final FirebaseAuth _auth = FirebaseAuth.instance;
 final TextEditingController _emailController = TextEditingController();
 final TextEditingController _passwordController = TextEditingController();
 final TextEditingController _usernameController = TextEditingController();

 @override
 void initState() {
   super.initState();
   _auth.setPersistence(Persistence.SESSION); // Ensures user logs in again if
app restarts
 }

 Future<void> _authenticate() async {
   setState(() => isLoading = true);

   String email = _emailController.text.trim();
   String password = _passwordController.text.trim();
   String username = _usernameController.text.trim();

   if (email.isEmpty || password.isEmpty || (isSignUp && username.isEmpty)) {
     ScaffoldMessenger.of(context).showSnackBar(
       SnackBar(content: Text("All fields are required!")),
     );
     setState(() => isLoading = false);
     return;
   }

   try {
     UserCredential userCredential;
     if (isSignUp) {
       // Sign Up
       userCredential = await _auth.createUserWithEmailAndPassword(
         email: email,
         password: password,
       );
       await userCredential.user?.updateDisplayName(username);
       ScaffoldMessenger.of(context).showSnackBar(
         SnackBar(content: Text("Account Created. Please Login!")),
       );
```

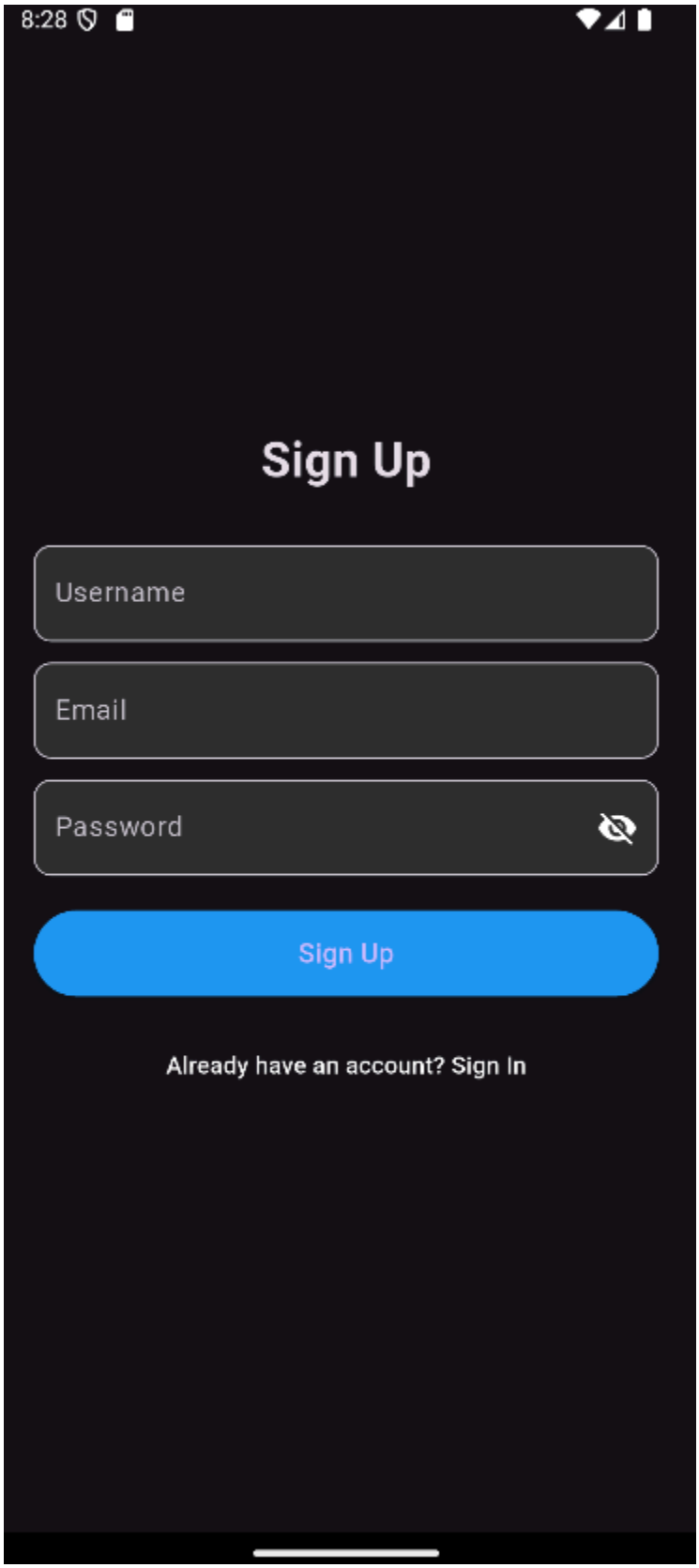
```
      setState(() => isSignUp = false); // Switch to sign-in mode
    } else {
      // Sign In
      userCredential = await _auth.signInWithEmailAndPassword(
        email: email,
        password: password,
      );
      String chatId = userCredential.user?.uid ?? "defaultChatId";

      if (mounted) {
        Navigator.pushReplacement(
          context,
          MaterialPageRoute(builder: (context) => HomeScreen(chatId: chatId)),
// Pass chatId
        );
      }
    }
  } on FirebaseAuthException catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(e.message ?? "Authentication error")),
    );
  } finally {
    if (mounted) setState(() => isLoading = false);
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Padding(
      padding: EdgeInsets.symmetric(horizontal: 24),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.center,
        children: [
          Text(
            isSignUp ? "Sign Up" : "Sign In",
            style: TextStyle(fontSize: 28, fontWeight: FontWeight.bold),
          ),
          SizedBox(height: 30),
          if (isSignUp)
            TextField(
              controller: _usernameController,
              decoration: InputDecoration(
                hintText: "Username",
                filled: true,
```

```
                fillColor: Colors.grey[850],
                border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10)),
              ),
            ),
          SizedBox(height: 12),
          TextField(
            controller: _emailController,
            decoration: InputDecoration(
              hintText: "Email",
              filled: true,
              fillColor: Colors.grey[850],
              border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10)),
            ),
          ),
          SizedBox(height: 12),
          TextField(
            controller: _passwordController,
            obscureText: !isPasswordVisible,
            decoration: InputDecoration(
              hintText: "Password",
              filled: true,
              fillColor: Colors.grey[850],
              border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10)),
              suffixIcon: IconButton(
                icon: Icon(
                  isPasswordVisible ? Icons.visibility : Icons.visibility_off,
                  color: Colors.white,
                ),
                onPressed: () {
                  setState(() {
                    isPasswordVisible = !isPasswordVisible;
                  });
                },
              ),
            ),
          ),
          SizedBox(height: 20),
          isLoading
              ? CircularProgressIndicator()
              : ElevatedButton(
            onPressed: _authenticate,
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.blue,
```

```
              minimumSize: Size(double.infinity, 50),
            ),
            child: Text(isSignUp ? "Sign Up" : "Sign In", style:
TextStyle(fontSize: 16)),
          ),
          SizedBox(height: 16),
          TextButton(
            onPressed: () => setState(() => isSignUp = !isSignUp),
            child: Text(
              isSignUp ? "Already have an account? Sign In" : "Don't have an
account? Sign Up",
              style: TextStyle(color: Colors.white),
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

Output:
Login Page




8:28
Sign Up
Username
Email
Password
Sign Up
Already have an account? Sign In

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

MPL Experiment 5

Name: Prathamesh Palve **Class:** D15A **Roll no:**31

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

Theory:

Navigation & Routing:
Flutter provides multiple ways to navigate between screens (pages):

1. **Using Navigator.push() and Navigator.pop()**
   Push a new screen onto the stack and pop it to return to the previous one.

2. **Named Routes**
   Define routes in MaterialApp and navigate using Navigator.pushNamed().

3. **GoRouter Package**
   A declarative approach to handle navigation more efficiently.

Gestures:
Flutter's GestureDetector and InkWell widgets help in detecting user interactions like taps, swipes, and long presses. Some common gestures include:

- **onTap:** Detects a tap event.
- **onDoubleTap:** Recognizes double taps.
- **onLongPress:** Detects a long press on a widget.
- **onHorizontalDrag & onVerticalDrag:** Detects drag/swipe motions.

Steps:
Step 1: Create a Flutter project and define multiple screens.
Step 2: Set up named routes in `MaterialApp`.
Step 3: Implement navigation using `Navigator.push()` and `Navigator.pushNamed()`.
Step 4: Use `GestureDetector` to detect taps, swipes, and long presses.
Step 5: Add buttons or swipes to navigate between screens.

Code:

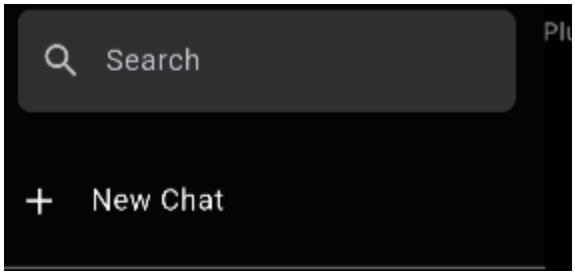
```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart'; // For accessing Mistral
API key
import '../services/api_service.dart';
import '../services/firestore_service.dart';
import '../widgets/chat_bubble.dart';
import '../widgets/empty_state.dart';
import '../widgets/bottom_bar.dart';
```

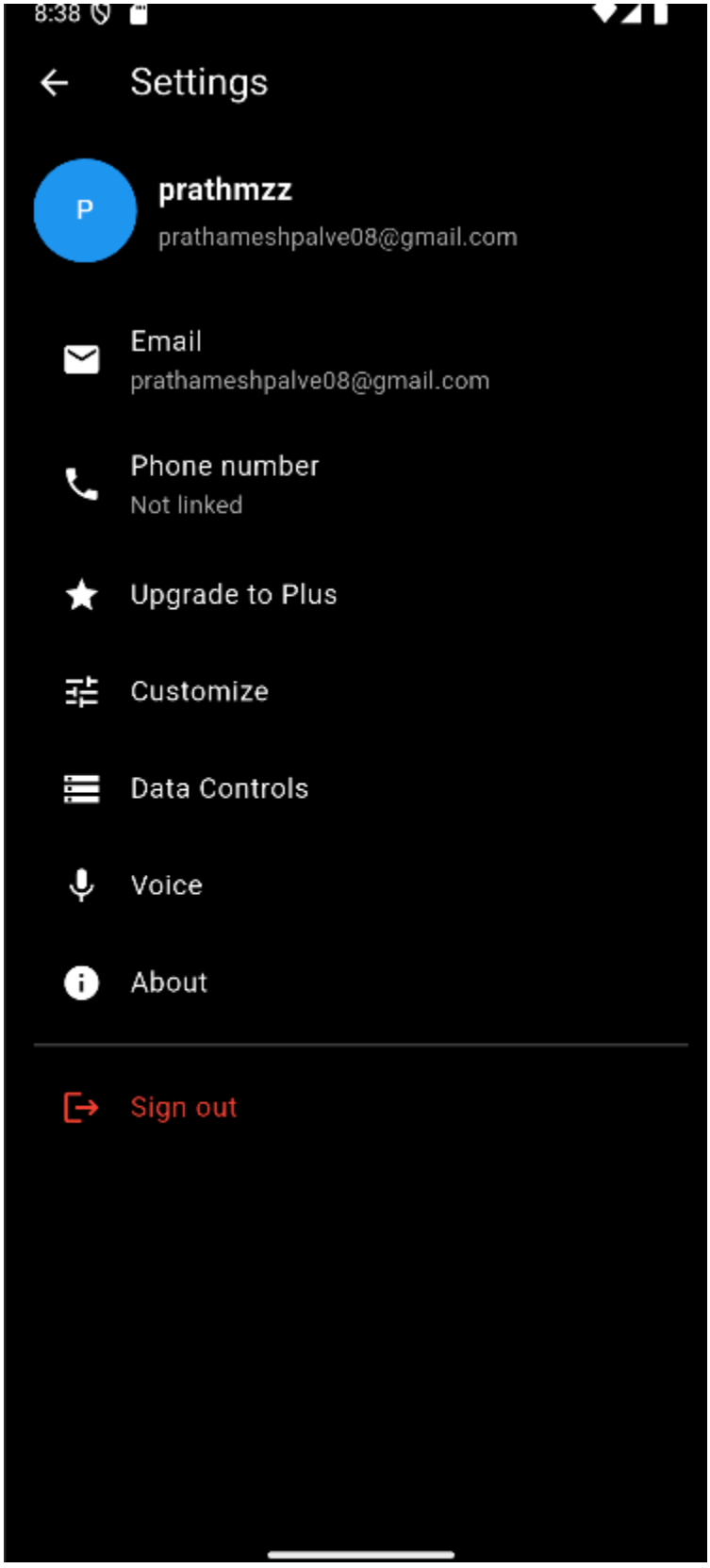
```dart
import 'setting.dart';
import 'left_slidder.dart';

class HomeScreen extends StatefulWidget {
 final String chatId; // Accepts chatId dynamically

 HomeScreen({required this.chatId});

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

class _HomeScreenState extends State<HomeScreen> {
 final TextEditingController _messageController = TextEditingController();
 final FirebaseAuth _auth = FirebaseAuth.instance;
 final FirestoreService _firestoreService = FirestoreService();
 final ApiService _apiService = ApiService();
 final ScrollController _scrollController = ScrollController();

 void _sendMessage() async {
   final user = _auth.currentUser;
   if (user == null || _messageController.text.trim().isEmpty) return;

   String userId = user.uid;
   String chatId = widget.chatId;
   String userMessage = _messageController.text.trim();

   try {
     // Store user message in Firestore
     await _firestoreService.addMessage(userId, chatId, userMessage, true);

     setState(() {
       _messageController.clear();
     });

     // Scroll to bottom after sending message
     Future.delayed(Duration(milliseconds: 300), () {
       _scrollToBottom();
     });

     // Get response from Mistral API
     String aiResponse = await _apiService.getMistralResponse(userMessage);

     // Store AI response in Firestore
     await _firestoreService.addMessage(userId, chatId, aiResponse, false);
```

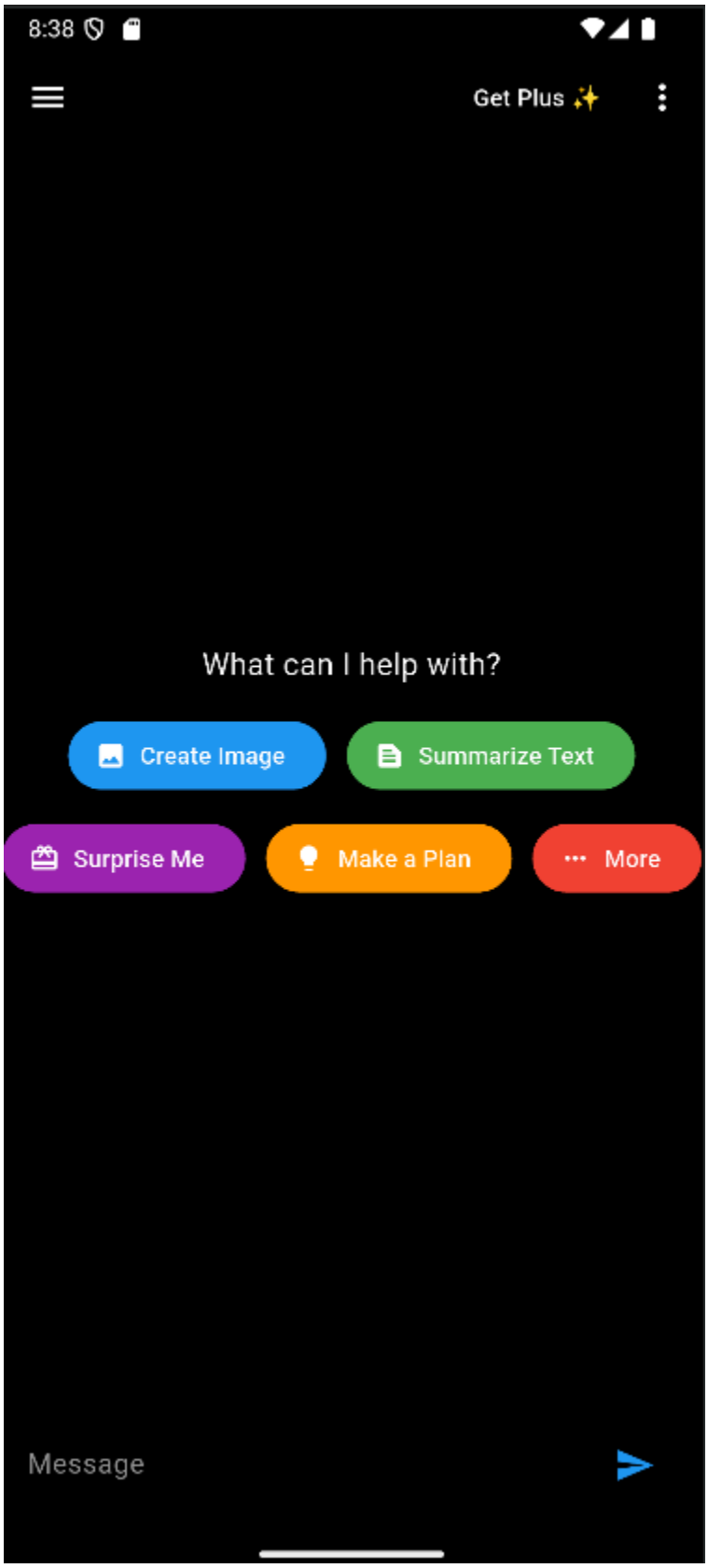
```
    print("✅ AI Response stored successfully!");
  } catch (e) {
    print("❌ Error sending message: $e");
  }
}

void _scrollToBottom() {
  if (_scrollController.hasClients) {
    _scrollController.animateTo(
      0, // Since we reversed ListView, scroll to position 0
      duration: Duration(milliseconds: 300),
      curve: Curves.easeOut,
    );
  }
}

@override
Widget build(BuildContext context) {
  final user = _auth.currentUser;
  if (user == null) {
    return Scaffold(
      body: Center(child: Text("User not logged in")),
    );
  }

  return Scaffold(
    backgroundColor: Colors.black,
    appBar: AppBar(
      backgroundColor: Colors.black,
      elevation: 0,
      leading: Builder(
        builder: (context) => IconButton(
          icon: Icon(Icons.menu, color: Colors.white),
          onPressed: () {
            Scaffold.of(context).openDrawer();
          },
        ),
      ),
      actions: [
        TextButton(
          onPressed: () {},
          child: Text("Get Plus ✨", style: TextStyle(color: Colors.white)),
        ),
        IconButton(
```

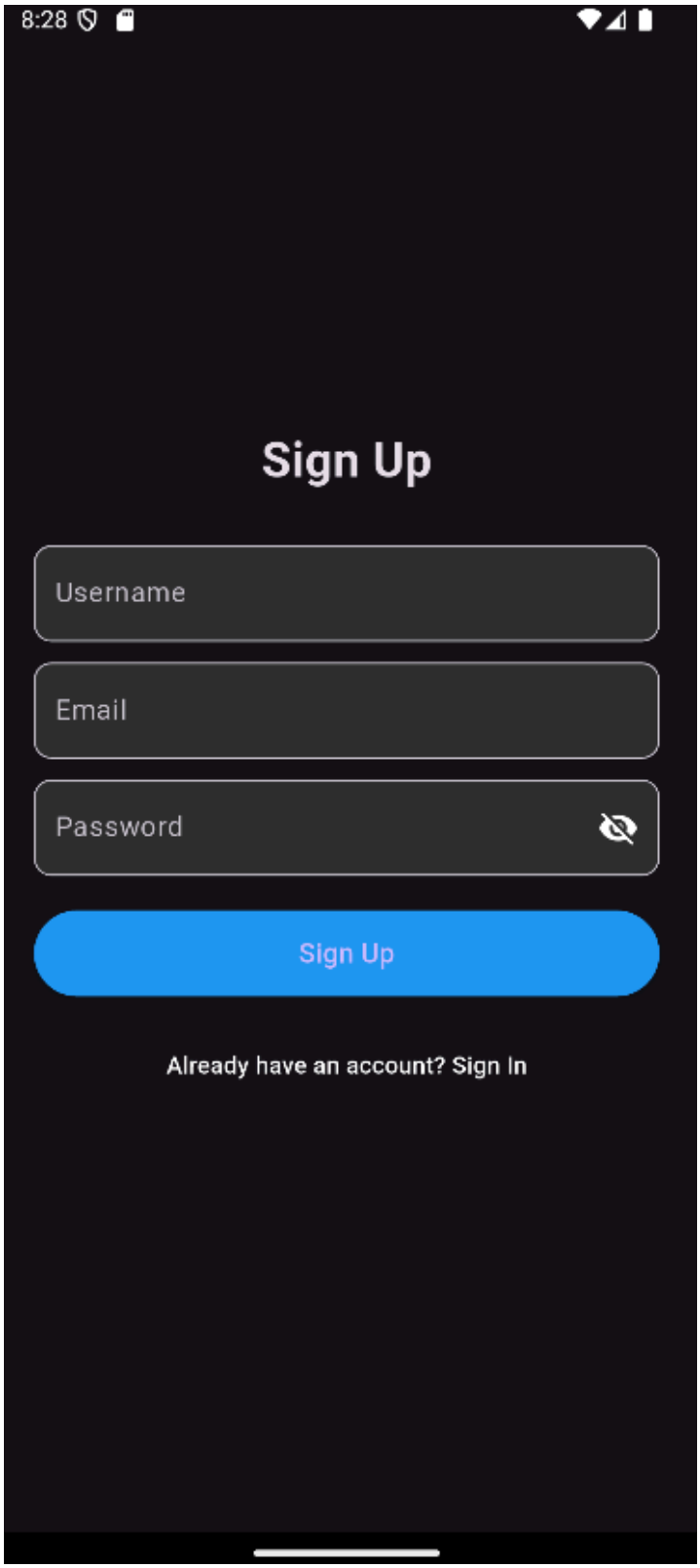
```
          icon: Icon(Icons.more_vert, color: Colors.white),
          onPressed: () {},
        ),
      ],
    ),
    drawer: LeftSlider(currentChatId: widget.chatId),
    body: Column(
      children: [
        Expanded(
          child: StreamBuilder<QuerySnapshot>(
            stream: _firestoreService.getMessages(user.uid, widget.chatId),
            builder: (context, snapshot) {
              if (snapshot.connectionState == ConnectionState.waiting) {
                return Center(child: CircularProgressIndicator());
              }

              if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
                return EmptyState(); // Show empty state if no messages
              }

              return ListView(
                controller: _scrollController,
                padding: EdgeInsets.all(16),
                reverse: true, // 🔥 Messages appear from bottom to top
                children: snapshot.data!.docs.map((doc) {
                  Map<String, dynamic> data = doc.data() as Map<String,
dynamic>;

                  bool isUser = data['isUser'] ?? false;
                  return ChatBubble(message: data['message'] ?? '', isUser:
isUser);

                }).toList(),
              );
            },
          ),
        ),
        BottomBar(
          messageController: _messageController,
          onSendPressed: _sendMessage,
        ),
      ],
    ),
  );
 }
}
```

Output:




8:38
Settings
P
prathmzz
prathameshpalve08@gmail.com
Email
prathameshpalve08@gmail.com
Phone number
Not linked
Upgrade to Plus
Customize
Data Controls
Voice
About
Sign out


8:38
Get Plus ✨
What can I help with?
Create Image
Summarize Text
Surprise Me
Make a Plan
More
Message


8:28
Sign Up
Username
Email
Password
Sign Up
Already have an account? Sign In

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

MPL Experiment 6

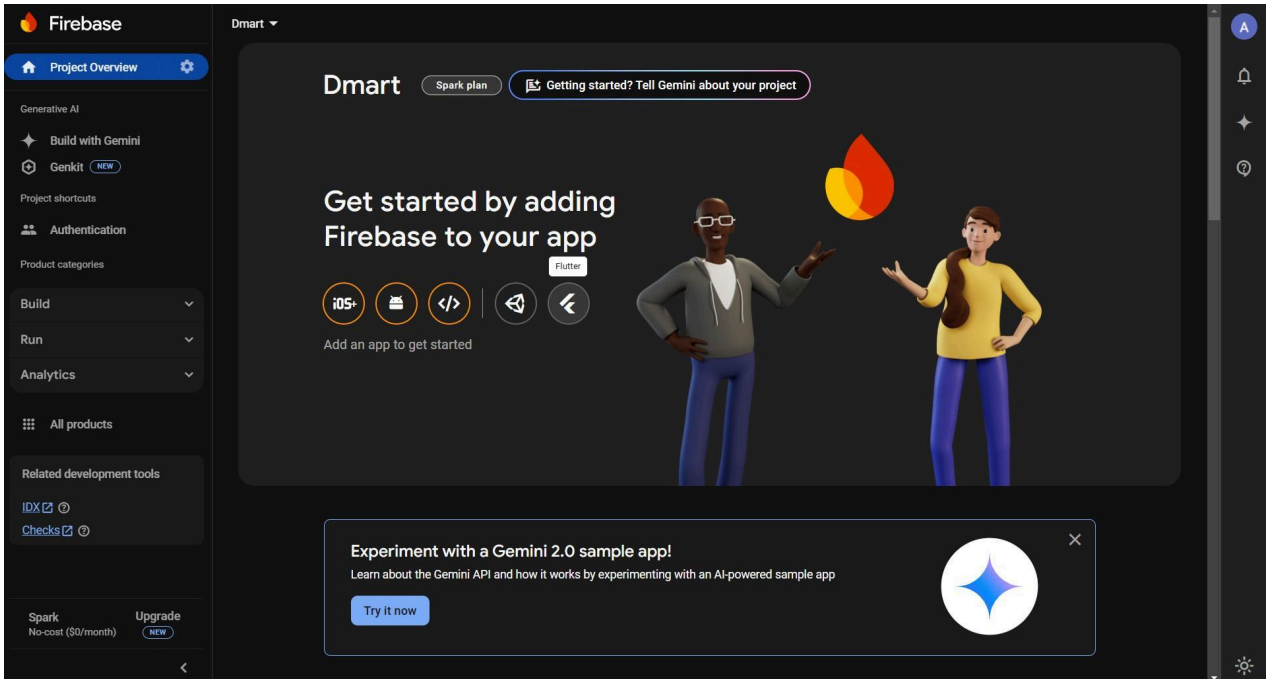
Name: Prathamesh Palve **Class:** D15A **Roll no:**31

Aim: How To Set Up Firebase with Flutter for iOS and Android Apps

Steps to Set Up Firebase with Flutter:

Step 1:

Go to the Firebase Console (https://console.firebase.google.com/). Click on "Add Project" and follow the steps to create your Firebase project. Once the project is created, select the Flutter option for connecting your app with Firebase.



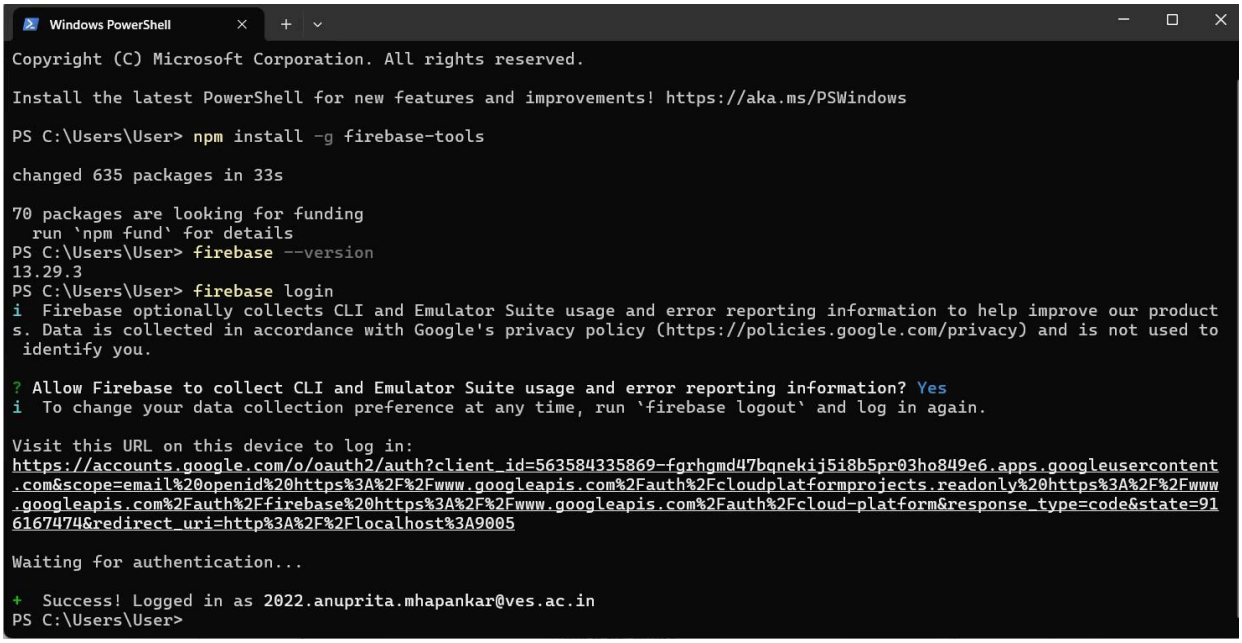
Step 2:

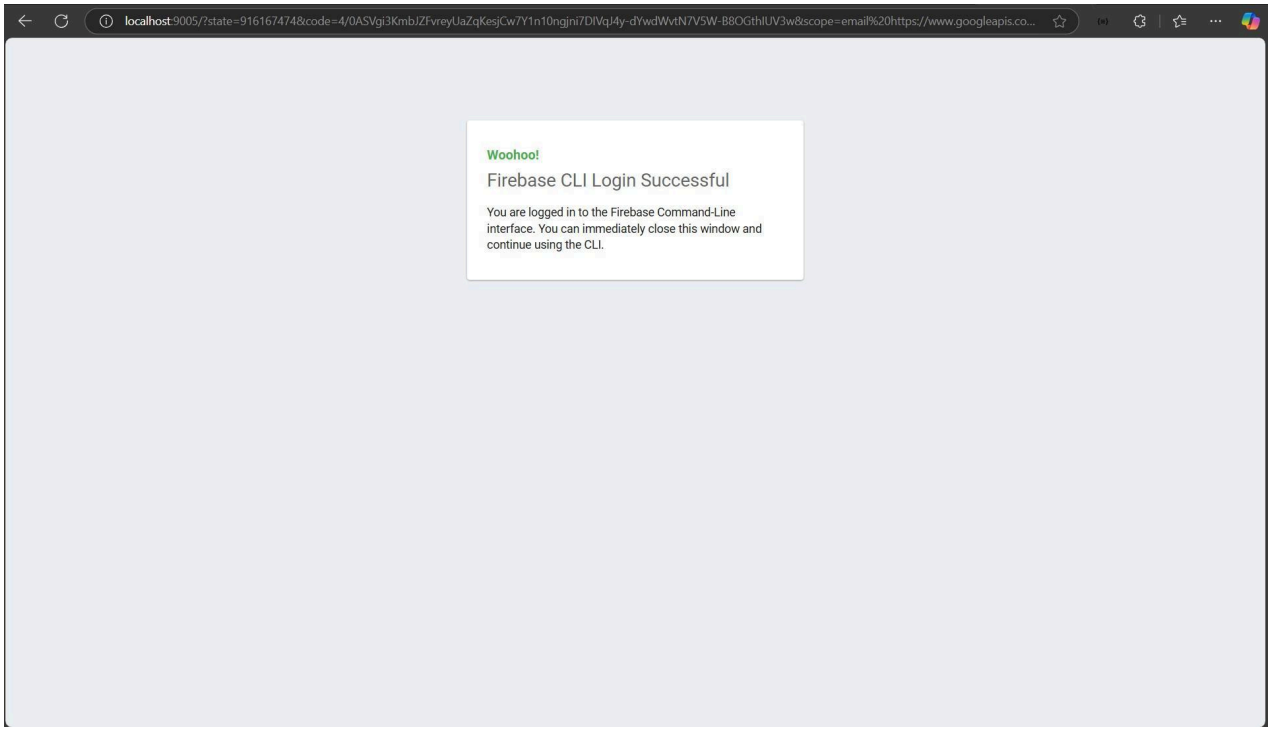
Open **Windows PowerShell** (or any terminal you prefer).

Run the following commands to install Firebase CLI and verify the installation:

```
npm install -g firebase-tools
firebase --version
firebase login
```

This will install the **Firebase CLI**, check the version, and log you into your Firebase account.





Step 3:

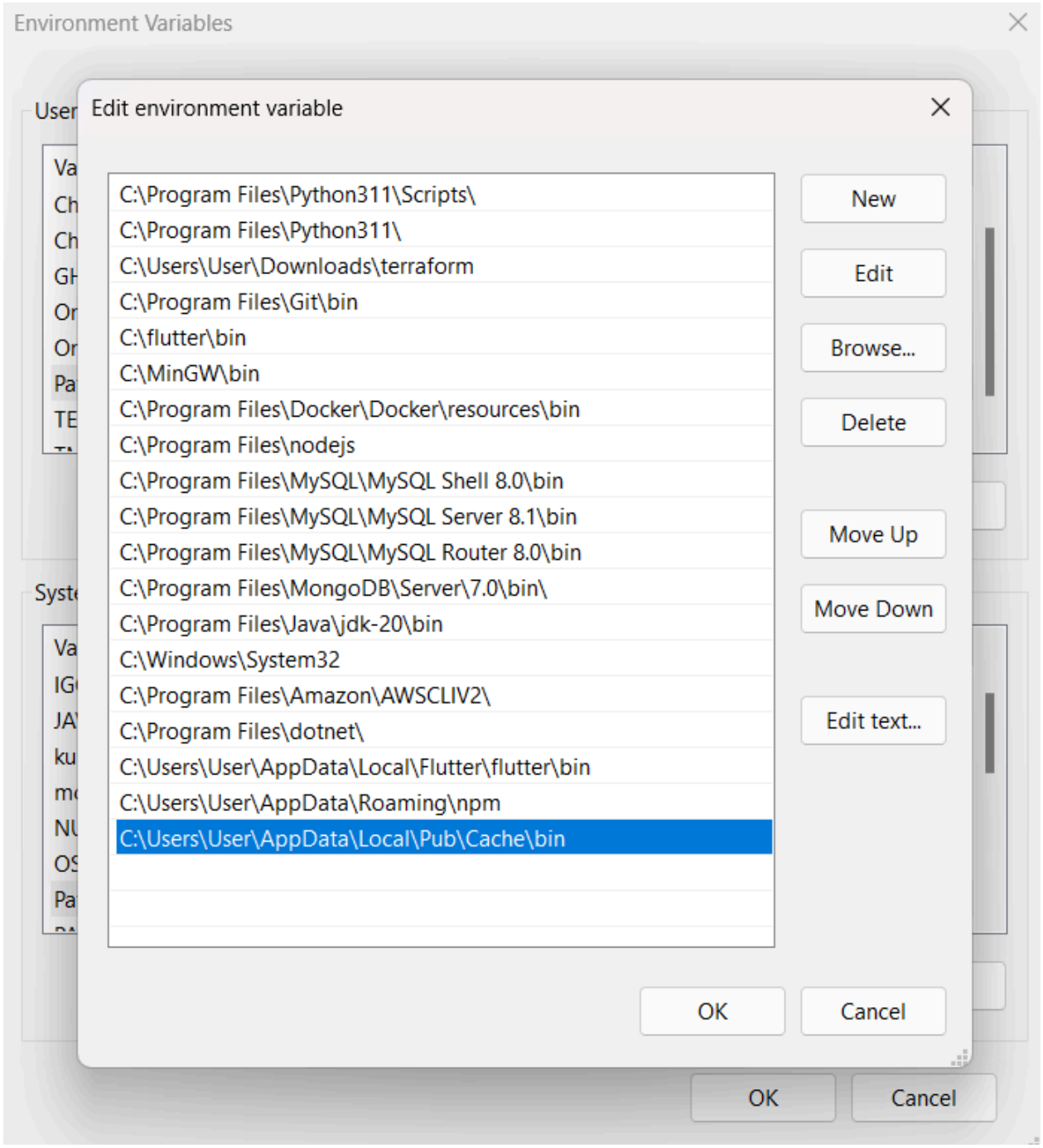
Open your Flutter app in **Android Studio**.

In the terminal of Android Studio, run the following command to activate flutterfire_cli:

```
dart pub global activate flutterfire_cli
```

Add flutterfire to your Environment Variables. You may need to restart Android Studio after

adding it.



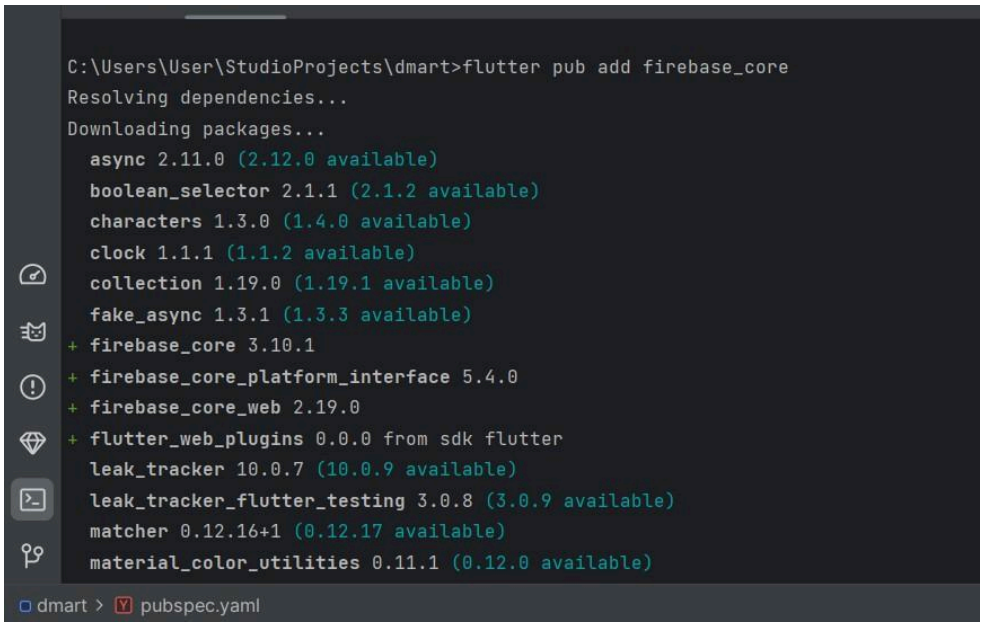
Step 4:

Run the following command to configure Firebase with your project: `flutterfire configure --project=chatGPT_08`

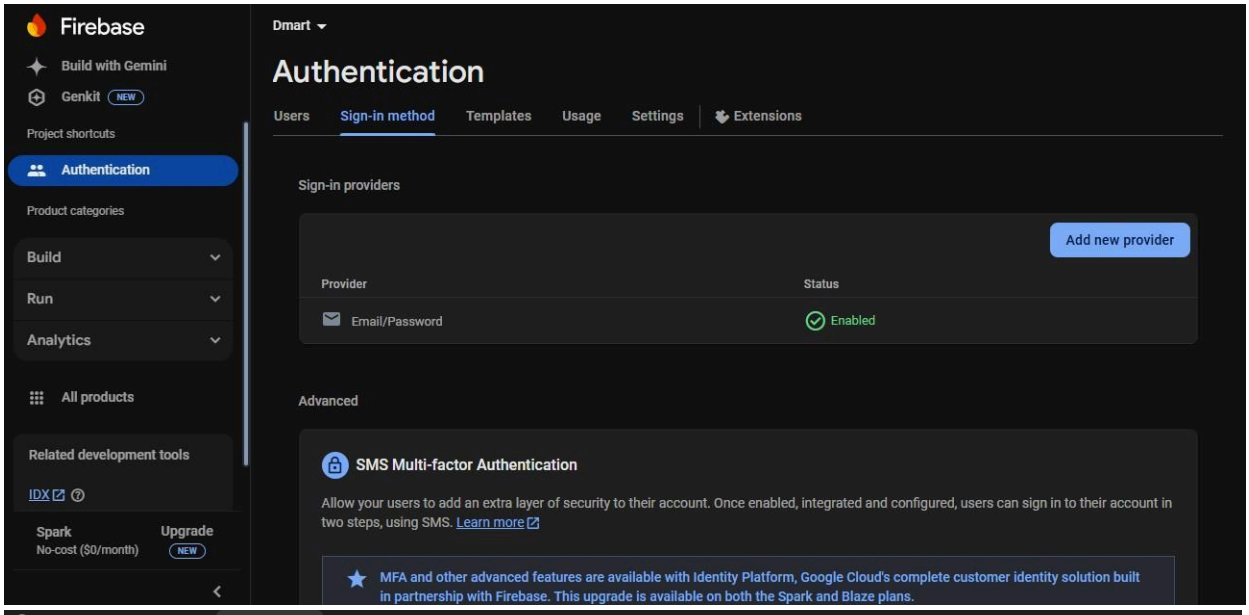
Replace `chatGPT_08` with your Firebase project ID

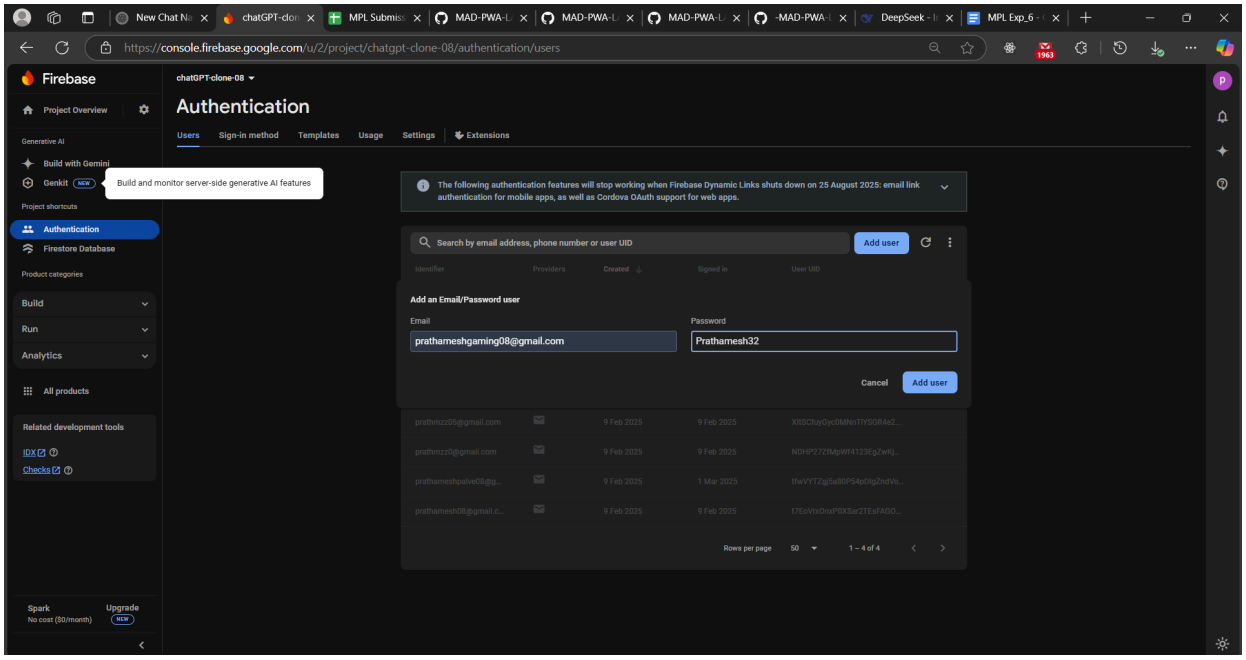
Step 5:
Run this command to add Firebase Core dependency to your app:

```
flutter pub add firebase_core
```



Firebase Authentication SetUp
Step 1:





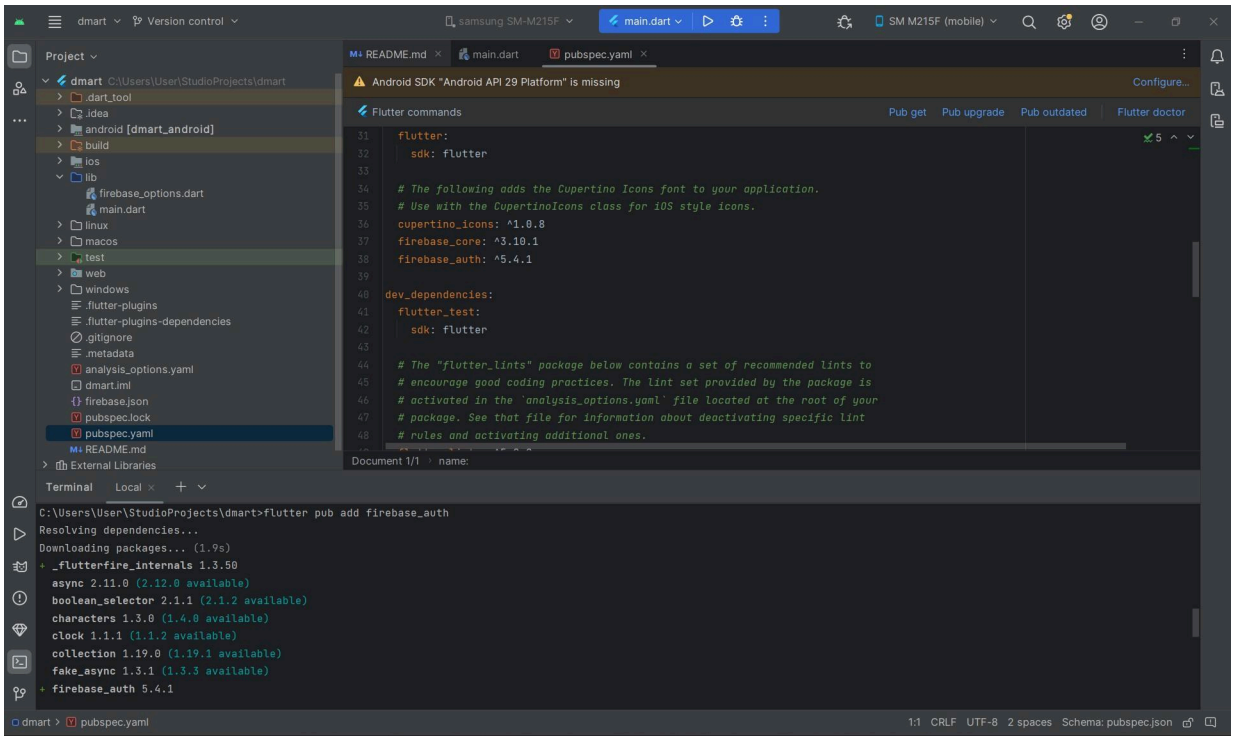
Go to the Firebase Console (https://console.firebase.google.com/). In your Firebase project, navigate to **Authentication**. Under the **Sign-in method** tab, enable **Email/Password** sign-in. Once enabled, go to the **Users** section and click on **Add User**. Enter a **username** (email) and a **password** for the new user.

Step 2:

Open your app in **Android Studio**.

In the terminal, run the following command to add the Firebase Authentication dependency:

```
flutter pub add firebase_auth
```

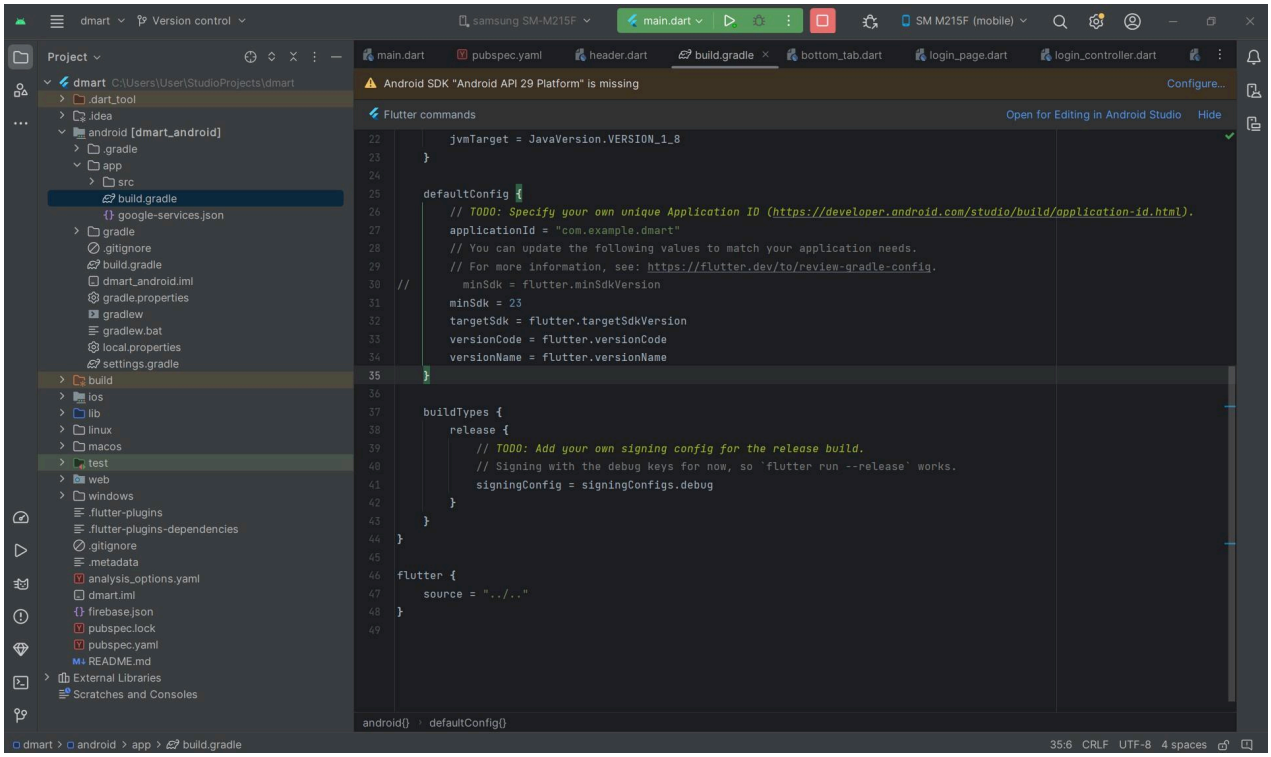


Step 3:

If you encounter any issues, add the following configuration to your `android/app/build.gradle` file:

```
android {
    defaultConfig
        { minSdk =
        23
        targetSdk =
        flutter.targetSdkVersion
        versionCode = flutter.versionCode
        versionName = flutter.versionName
    }
}
```

This ensures that the Firebase dependencies are compatible with your Android app.

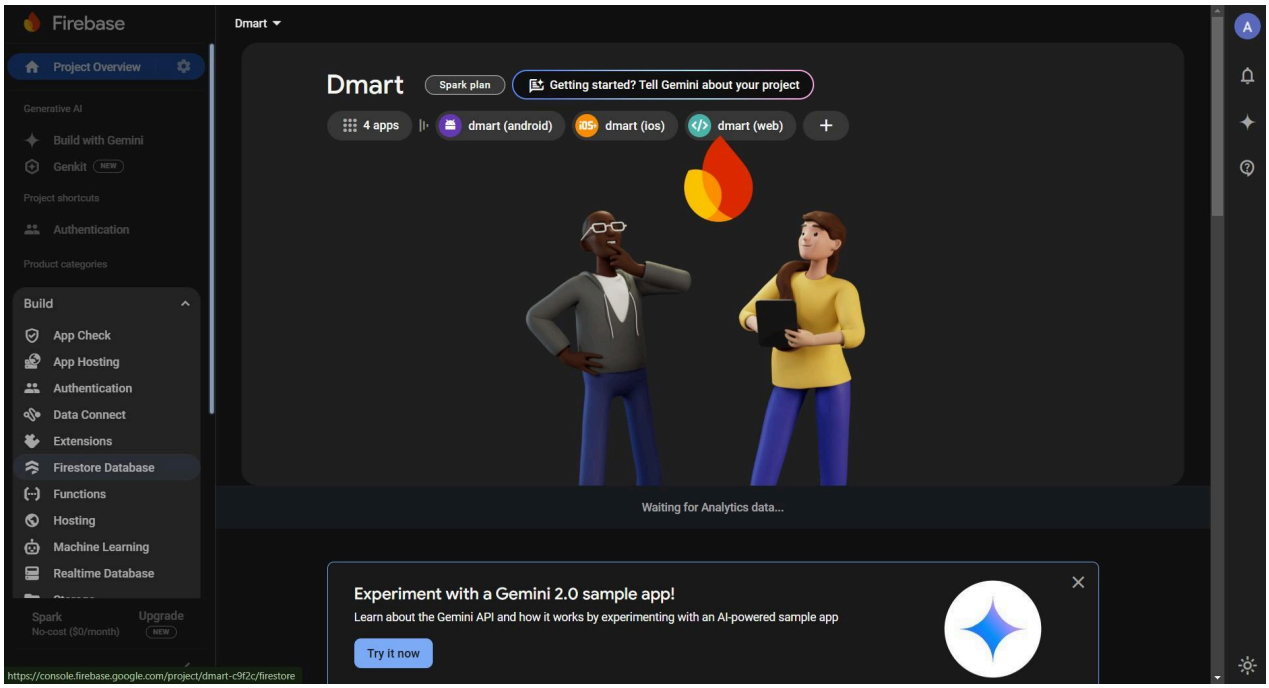


Now, firebase is successfully connected to our app.

Firestore Database Setup

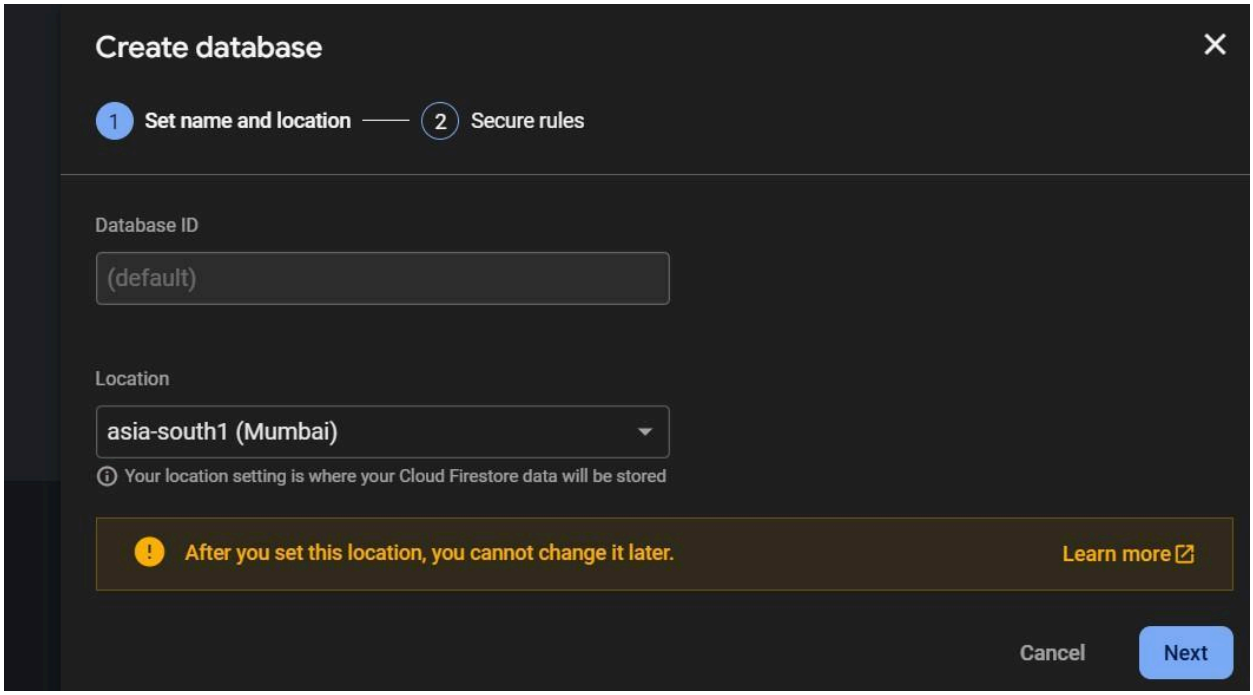
Step 1:

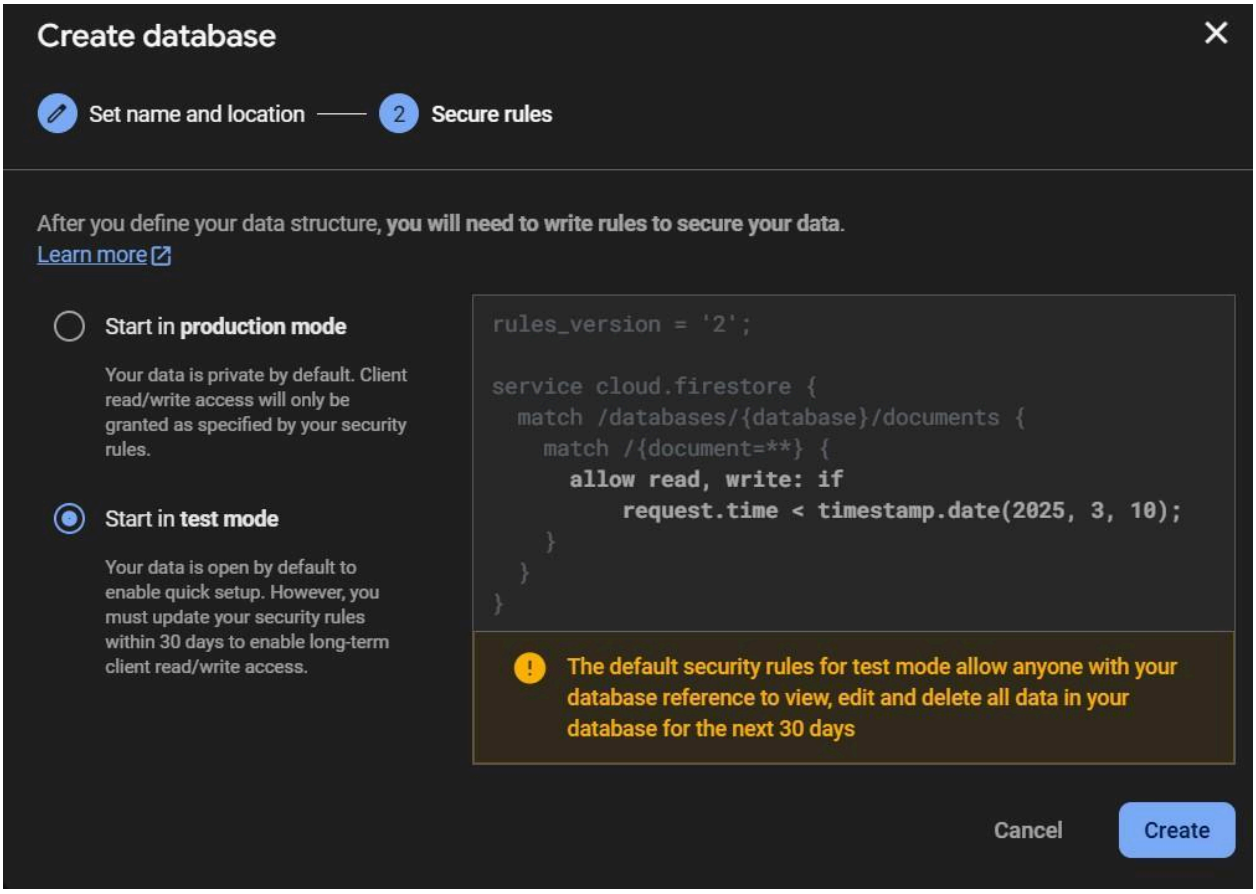
Select **Firestore Database** from **Build** in the sidebar.



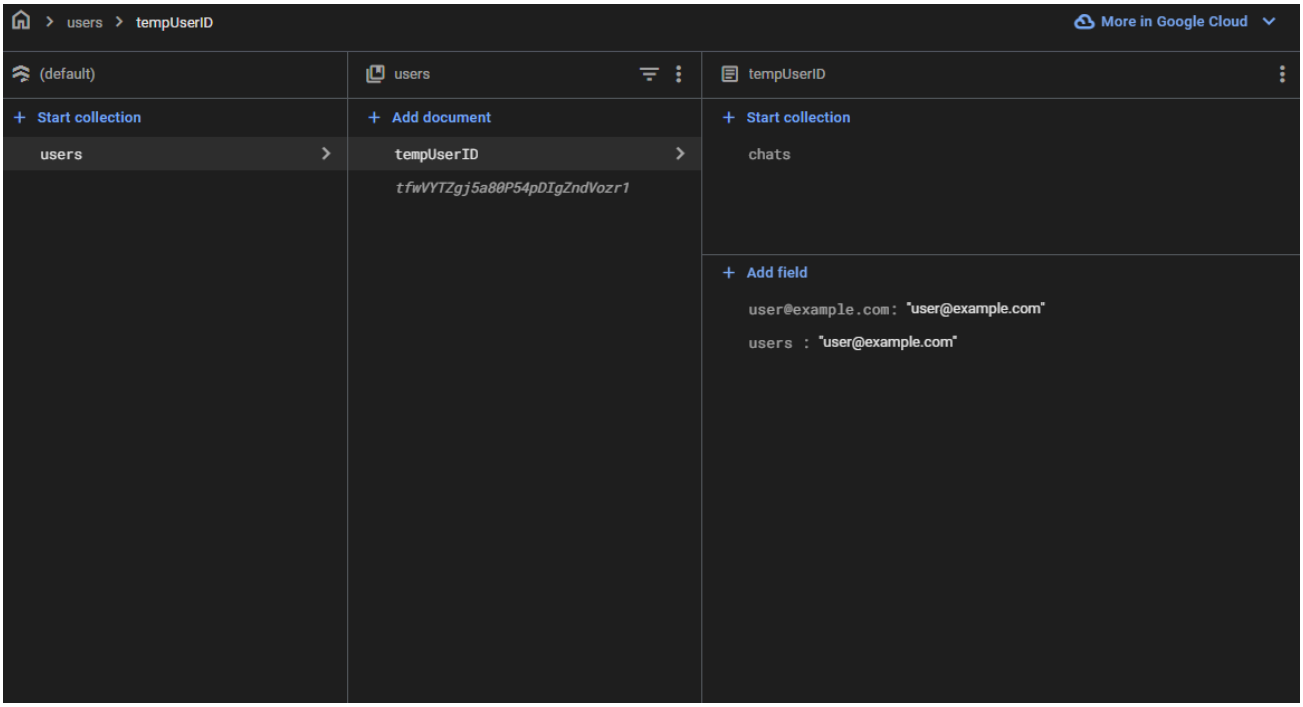
Then click on Create database. For location select asia-south1(Mumbai). Then choose **Start in test mode** (for development).

Click **Next** and choose a location, then **Enable**.





Step 2:
Start by creating a **collection**. Add a **document** within the collection. Define the **fields** as per project



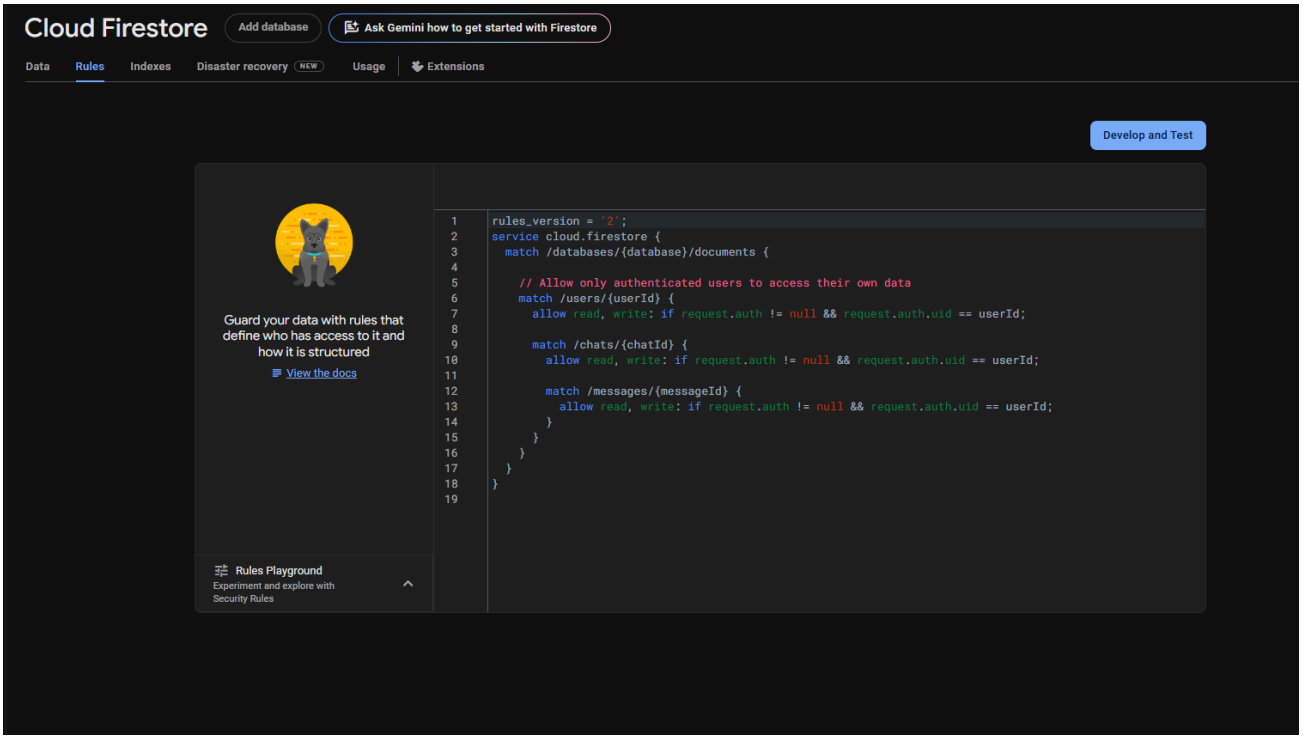
Step 3:
Run the following command to add Firestore to your Flutter app:

```
flutter pub add cloud_firestore
```

Step 4:
Go to Firebase Console → Firestore Database → Rules
Set the following rules:

```
rules_version = '2';
service cloud.firestore
{
  match /databases/{database}/documents {
    match /categories/{categoryId} {
      allow read, write: if true;
      match /subcategories/{subcategoryId}
        { allow read, write: if true;
      }
    }
  }
}
```

and then click **Publish** to apply changes.



Code:

```
//auth.dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'home.dart'; // Import HomeScreen

class AuthScreen extends StatefulWidget {
 @override
 _AuthScreenState createState() => _AuthScreenState();
}

class _AuthScreenState extends State<AuthScreen> {
 bool isSignUp = false;
 bool isPasswordVisible = false;
 bool isLoading = false;

 final FirebaseAuth _auth = FirebaseAuth.instance;
 final TextEditingController _emailController = TextEditingController();
 final TextEditingController _passwordController = TextEditingController();
 final TextEditingController _usernameController = TextEditingController();

 @override
 void initState() {
   super.initState();
   _auth.setPersistence(Persistence.SESSION); // Ensures user logs in again if app
restarts
 }
```

```dart
Future<void> _authenticate() async {
  setState(() => isLoading = true);

  String email = _emailController.text.trim();
  String password = _passwordController.text.trim();
  String username = _usernameController.text.trim();

  if (email.isEmpty || password.isEmpty || (isSignUp && username.isEmpty)) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text("All fields are required!")),
    );
    setState(() => isLoading = false);
    return;
  }

  try {
    UserCredential userCredential;
    if (isSignUp) {
      // Sign Up
      userCredential = await _auth.createUserWithEmailAndPassword(
        email: email,
        password: password,
      );
      await userCredential.user?.updateDisplayName(username);
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Account Created. Please Login!")),
      );
      setState(() => isSignUp = false); // Switch to sign-in mode
    } else {
      // Sign In
      userCredential = await _auth.signInWithEmailAndPassword(
        email: email,
        password: password,
      );
      String chatId = userCredential.user?.uid ?? "defaultChatId";

      if (mounted) {
        Navigator.pushReplacement(
          context,
          MaterialPageRoute(builder: (context) => HomeScreen(chatId: chatId)), //
Pass chatId
        );
      }
    }
  } on FirebaseAuthException catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(e.message ?? "Authentication error")),
    );
```

```
    } finally {
      if (mounted) setState(() => isLoading = false);
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Padding(
        padding: EdgeInsets.symmetric(horizontal: 24),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            Text(
              isSignUp ? "Sign Up" : "Sign In",
              style: TextStyle(fontSize: 28, fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 30),
            if (isSignUp)
              TextField(
                controller: _usernameController,
                decoration: InputDecoration(
                  hintText: "Username",
                  filled: true,
                  fillColor: Colors.grey[850],
                  border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10)),
                ),
              ),
            SizedBox(height: 12),
            TextField(
              controller: _emailController,
              decoration: InputDecoration(
                hintText: "Email",
                filled: true,
                fillColor: Colors.grey[850],
                border: OutlineInputBorder(borderRadius: BorderRadius.circular(10)),
              ),
            ),
            SizedBox(height: 12),
            TextField(
              controller: _passwordController,
              obscureText: !isPasswordVisible,
              decoration: InputDecoration(
                hintText: "Password",
                filled: true,
                fillColor: Colors.grey[850],
```

```
              border: OutlineInputBorder(borderRadius: BorderRadius.circular(10)),
              suffixIcon: IconButton(
                icon: Icon(
                  isPasswordVisible ? Icons.visibility : Icons.visibility_off,
                  color: Colors.white,
                ),
                onPressed: () {
                  setState(() {
                    isPasswordVisible = !isPasswordVisible;
                  });
                },
              ),
            ),
          ),
          SizedBox(height: 20),
          isLoading
              ? CircularProgressIndicator()
              : ElevatedButton(
            onPressed: _authenticate,
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.blue,
              minimumSize: Size(double.infinity, 50),
            ),
            child: Text(isSignUp ? "Sign Up" : "Sign In", style:
TextStyle(fontSize: 16)),
          ),
          SizedBox(height: 16),
          TextButton(
            onPressed: () => setState(() => isSignUp = !isSignUp),
            child: Text(
              isSignUp ? "Already have an account? Sign In" : "Don't have an
account? Sign Up",
              style: TextStyle(color: Colors.white),
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

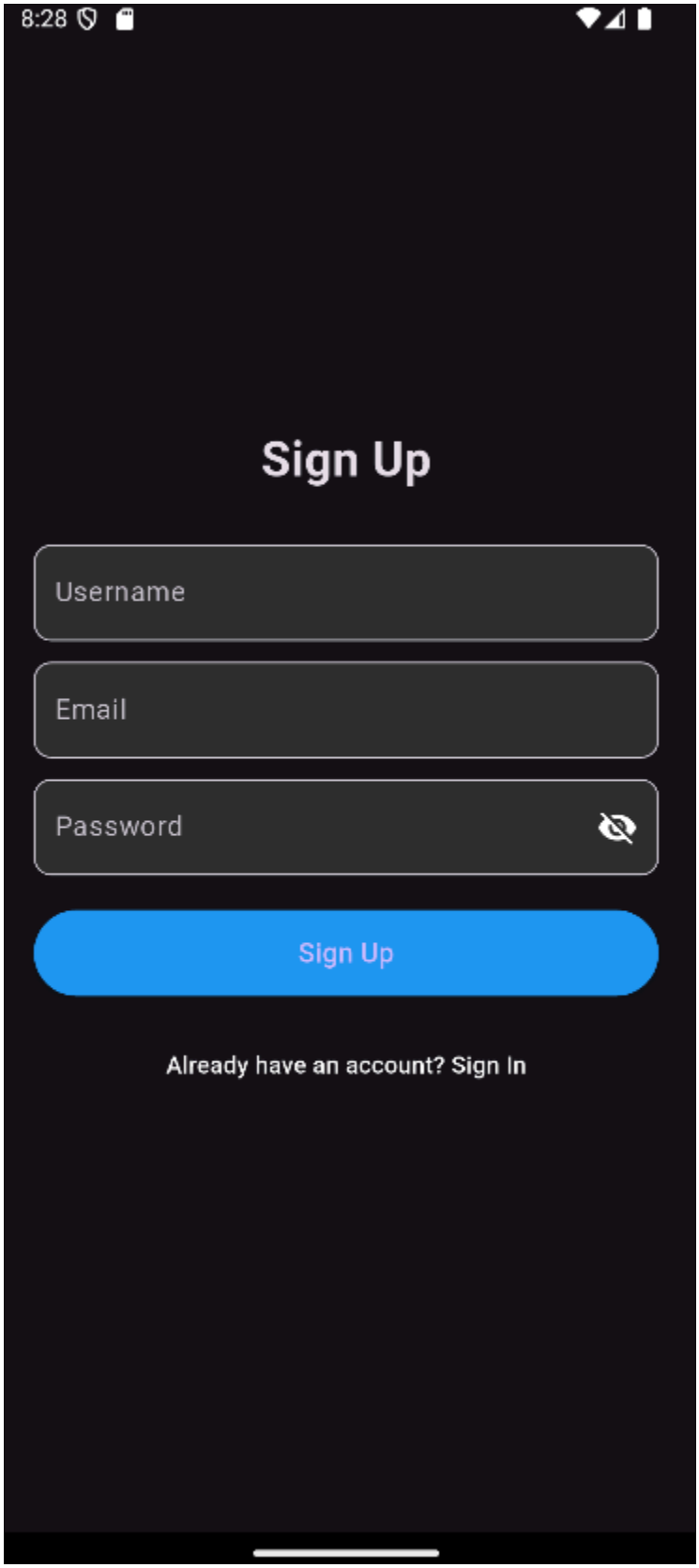
```dart
//home.dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import '../services/api_service.dart';
import '../services/firestore_service.dart';
import '../widgets/chat_bubble.dart';
import '../widgets/empty_state.dart';
import '../widgets/bottom_bar.dart';
import 'setting.dart';
import 'left_slidder.dart';

class HomeScreen extends StatefulWidget {
 final String chatId; // Accepts chatId dynamically

 HomeScreen({required this.chatId});

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

class _HomeScreenState extends State<HomeScreen> {
 final TextEditingController _messageController = TextEditingController();
 final FirebaseAuth _auth = FirebaseAuth.instance;
 final FirestoreService _firestoreService = FirestoreService();
 final ApiService _apiService = ApiService();
 final ScrollController _scrollController = ScrollController();

 void _sendMessage() async {
   final user = _auth.currentUser;
   if (user == null || _messageController.text.trim().isEmpty) return;

   String userId = user.uid;
   String chatId = widget.chatId;
   String userMessage = _messageController.text.trim();

   try {
     // Store user message in Firestore
     await _firestoreService.addMessage(userId, chatId, userMessage, true);

     setState(() {
       _messageController.clear();
     });

     // Scroll to bottom after sending message
     Future.delayed(Duration(milliseconds: 300), () {
       _scrollToBottom();
```

```
    });

    // Get response from Hugging Face API
    String aiResponse = await _apiService.getHuggingFaceResponse(userMessage);

    // Store AI response in Firestore
    await _firestoreService.addMessage(userId, chatId, aiResponse, false);

    print("✅ AI Response stored successfully!");
  } catch (e) {
    print("❌ Error sending message: $e");
  }
}

void _scrollToBottom() {
  if (_scrollController.hasClients) {
    _scrollController.animateTo(
      0, // Since we reversed ListView, scroll to position 0
      duration: Duration(milliseconds: 300),
      curve: Curves.easeOut,
    );
  }
}

@override
Widget build(BuildContext context) {
  final user = _auth.currentUser;
  if (user == null) {
    return Scaffold(
      body: Center(child: Text("User not logged in")),
    );
  }

  return Scaffold(
    backgroundColor: Colors.black,
    appBar: AppBar(
      backgroundColor: Colors.black,
      elevation: 0,
      leading: Builder(
        builder: (context) => IconButton(
          icon: Icon(Icons.menu, color: Colors.white),
          onPressed: () {
            Scaffold.of(context).openDrawer();
          },
        ),
      ),
      actions: [
        TextButton(
```

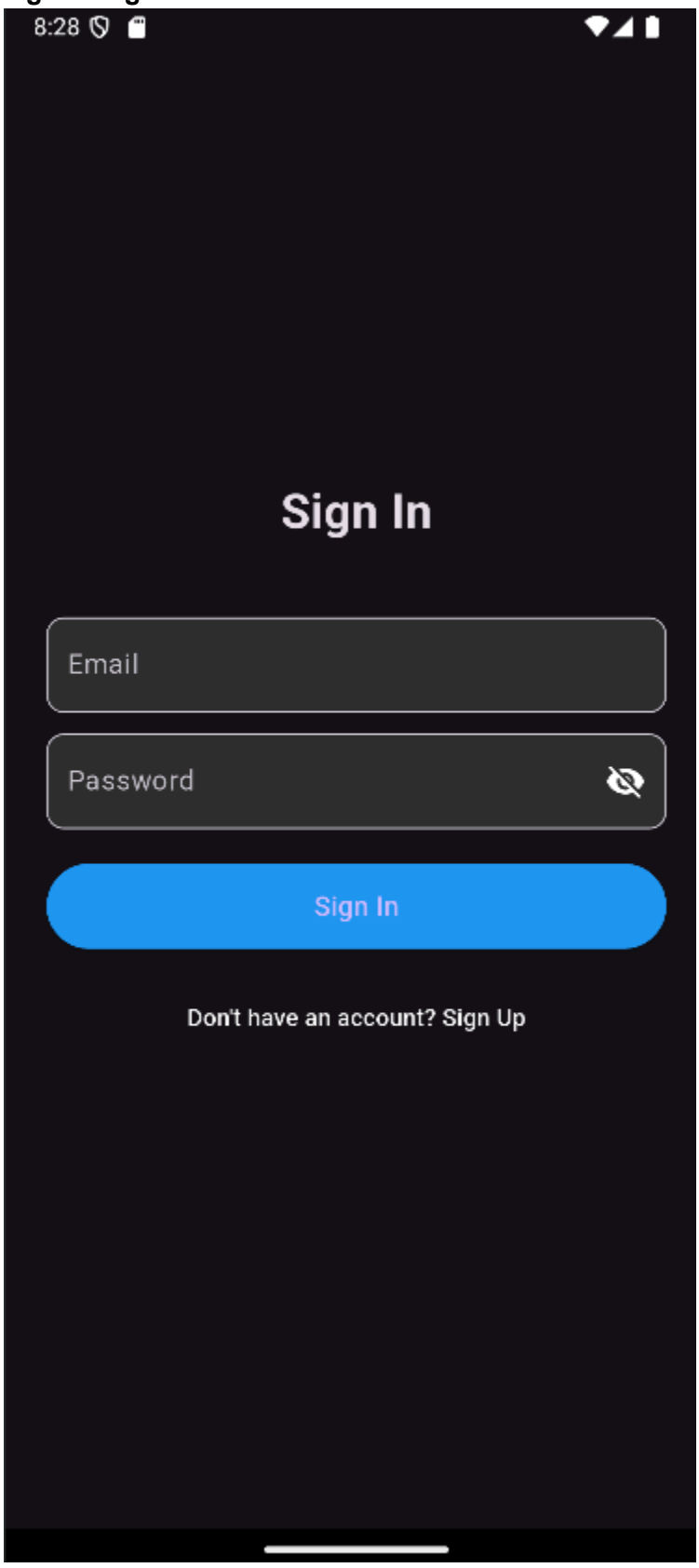
```
            onPressed: () {},
            child: Text("Get Plus ✨", style: TextStyle(color: Colors.white)),
          ),
          IconButton(
            icon: Icon(Icons.more_vert, color: Colors.white),
            onPressed: () {},
          ),
        ],
      ),
      drawer: LeftSlider(currentChatId: widget.chatId),
      body: Column(
        children: [
          Expanded(
            child: StreamBuilder<QuerySnapshot>(
              stream: _firestoreService.getMessages(user.uid, widget.chatId),
              builder: (context, snapshot) {
                if (snapshot.connectionState == ConnectionState.waiting) {
                  return Center(child: CircularProgressIndicator());
                }

                if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
                  return EmptyState(); // Show empty state if no messages
                }

                return ListView(
                  controller: _scrollController,
                  padding: EdgeInsets.all(16),
                  reverse: true, // 🔥 Messages appear from bottom to top
                  children: snapshot.data!.docs.map((doc) {
                    Map<String, dynamic> data = doc.data() as Map<String, dynamic>;
                    bool isUser = data['isUser'] ?? false;
                    return ChatBubble(message: data['message'] ?? '', isUser:
isUser);
                  }).toList(),
                );
              },
            ),
          ),
          BottomBar(
            messageController: _messageController,
            onSendPressed: _sendMessage,
          ),
        ],
      ),
    );
  }
}
```

Output:

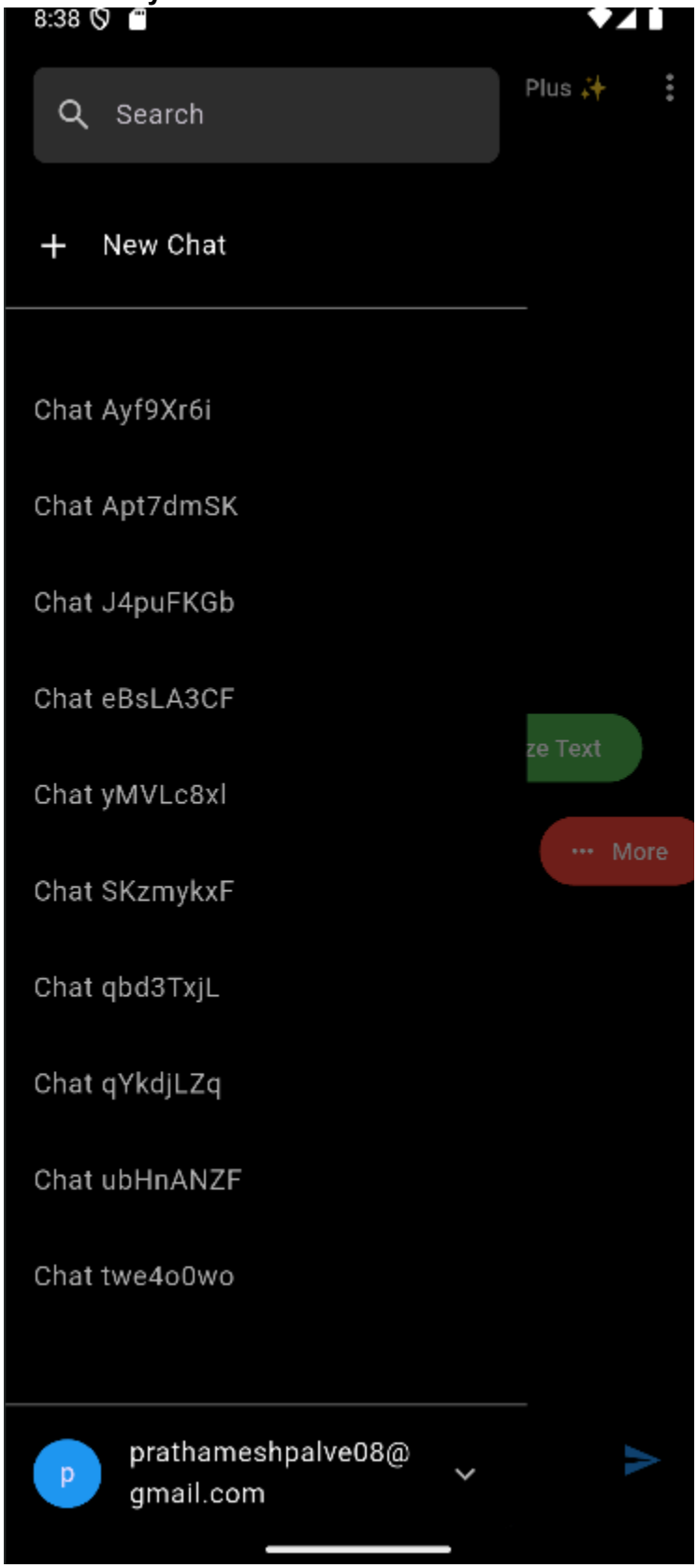
Signup Page:


8:28
Sign Up
Username
Email
Password
Sign Up
Already have an account? Sign In

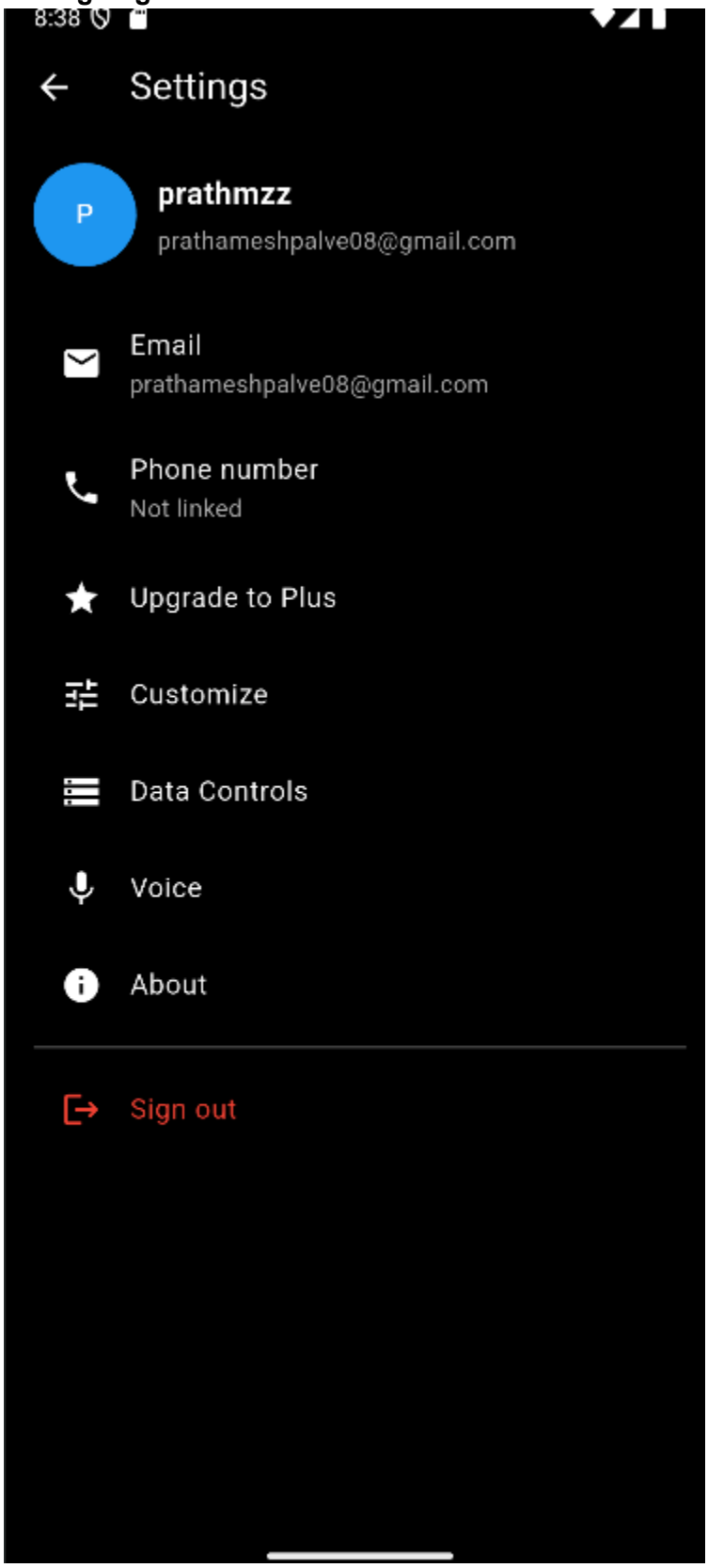
Signin Page:


8:28
Sign In
Email
Password
Sign In
Don't have an account? Sign Up

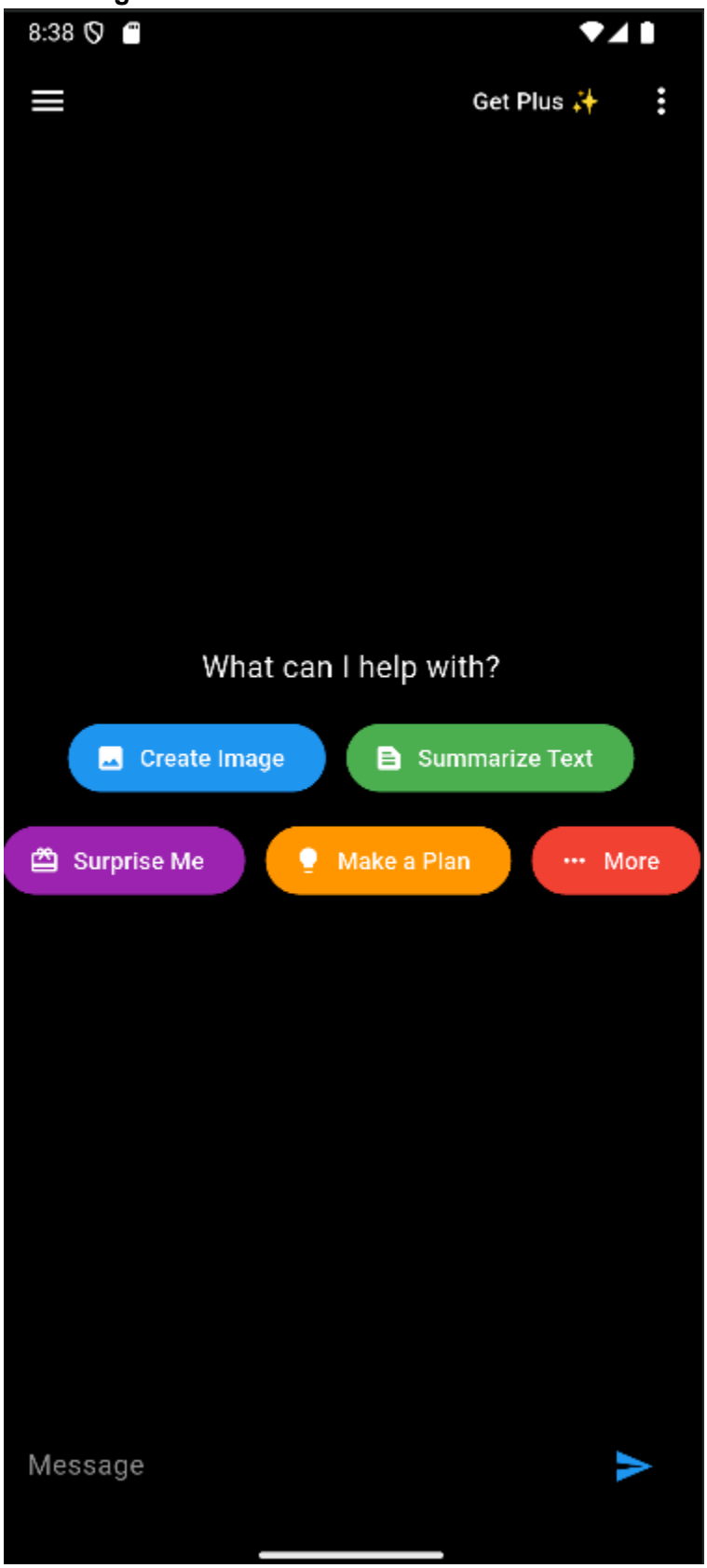
Chat History:



Setting Page:



home Page:


8:38
Get Plus ✨
What can I help with?
Create Image
Summarize Text
Surprise Me
Make a Plan
More
Message

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.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

MPL Experiment 7 (PWA)

Name: Prathamesh Palve **Class:** D15A **Roll no:**31

Aim: To write meta data of your 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.

Difference between PWAs vs. Regular Web Apps:

A Progressive Web is different and better than a Regular Web app with features like:

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

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

3. Faster Services
PWAs can cache the data and serve the user with text stylesheets, images, and other web content even before the page loads completely. This lowers the waiting time for the end-users

and helps the brands improve the user engagement and retention rate, which eventually adds value to their business.

The main features are:

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

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

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

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

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

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

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

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

- Linkable
  Easily shared via URL without complex installations.

Codes:
Manifest.json

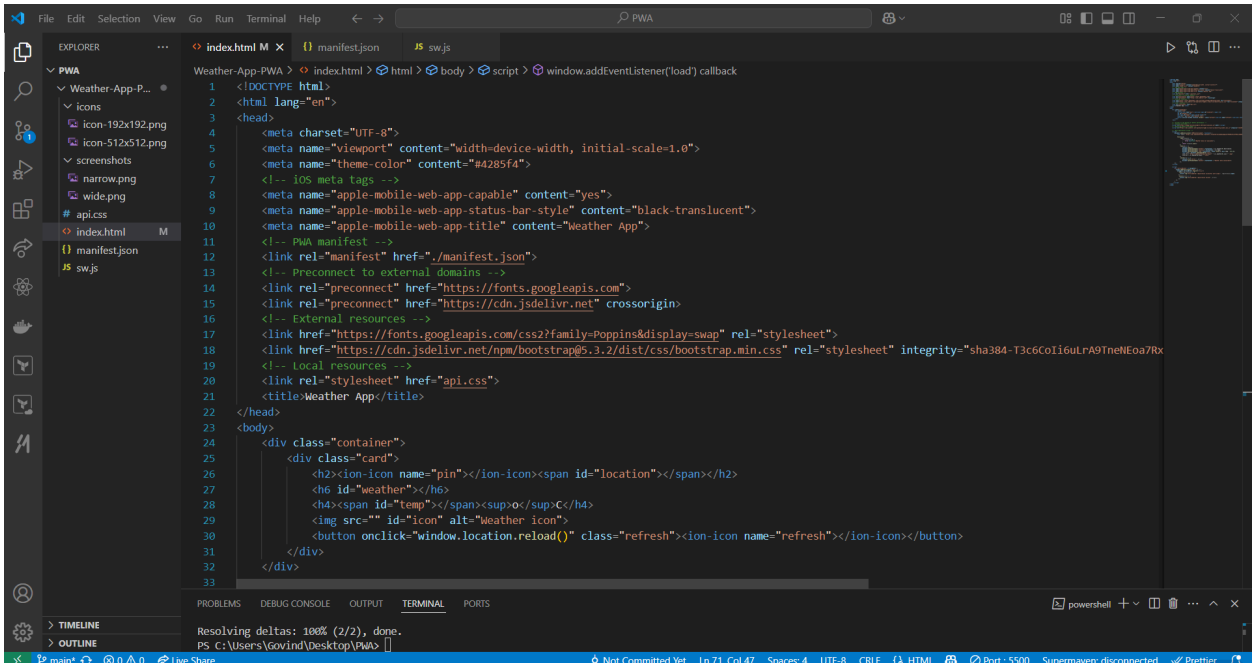
```
{
   "name": "Weather App",
```

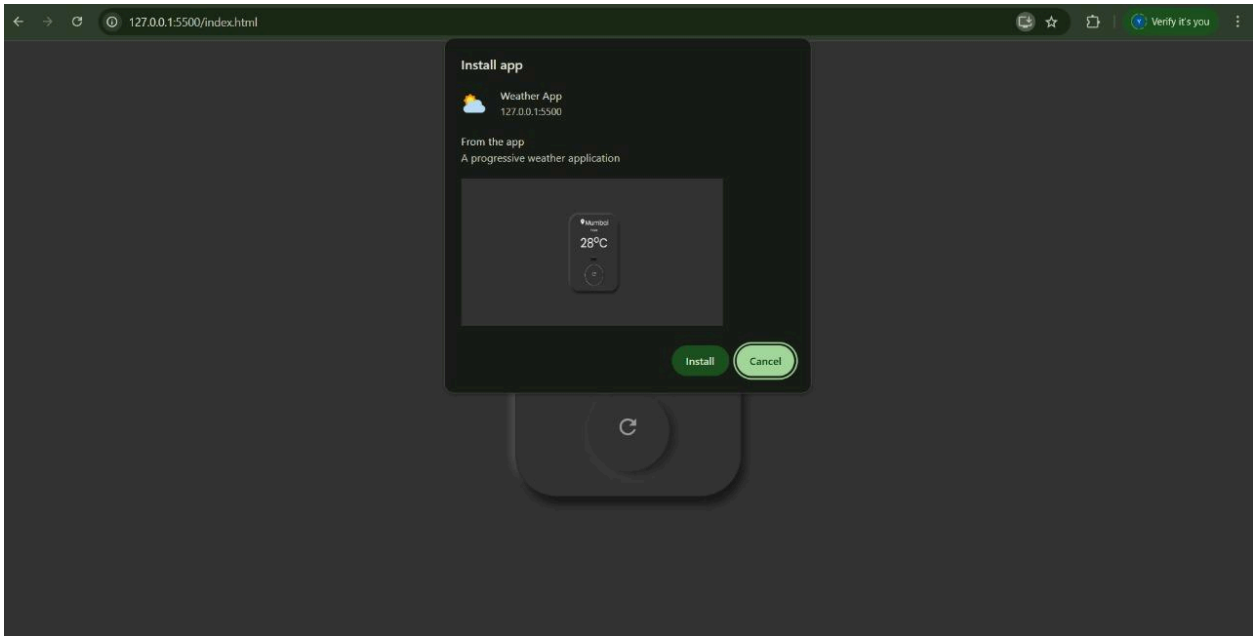
```
"short_name": "Weather",
"description": "A progressive weather application",
"start_url": "/index.html",
"scope": "/",
"display": "standalone",
"orientation": "portrait-primary",
"background_color": "#ffffff",
"theme_color": "#4285f4",
"categories": ["weather", "utilities"],
"lang": "en-US",
"dir": "ltr",
"icons": [
  {
    "src": "icons/icon-72x72.png",
    "sizes": "72x72",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "icons/icon-96x96.png",
    "sizes": "96x96",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "icons/icon-128x128.png",
    "sizes": "128x128",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "icons/icon-144x144.png",
    "sizes": "144x144",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "icons/icon-152x152.png",
    "sizes": "152x152",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "icons/icon-192x192.png",
```

```
    "sizes": "192x192",
    "type": "image/png",
    "purpose": "any maskable"
  },
  {
    "src": "icons/icon-384x384.png",
    "sizes": "384x384",
    "type": "image/png",
    "purpose": "any"
  },
  {
    "src": "icons/icon-512x512.png",
    "sizes": "512x512",
    "type": "image/png",
    "purpose": "any maskable"
  }
],
"screenshots": [
  {
    "src": "screenshots/wide.png",
    "sizes": "1280x720",
    "type": "image/png",
    "form_factor": "wide",
    "label": "Desktop view"
  },
  {
    "src": "screenshots/narrow.png",
    "sizes": "412x915",
    "type": "image/png",
    "form_factor": "narrow",
    "label": "Mobile view"
  }
],
"shortcuts": [
  {
    "name": "Current Weather",
    "short_name": "Now",
    "description": "View current weather conditions",
    "url": "/?view=current",
    "icons": [
      {
        "src": "icons/icon-96x96.png",
        "sizes": "96x96"
      }
```

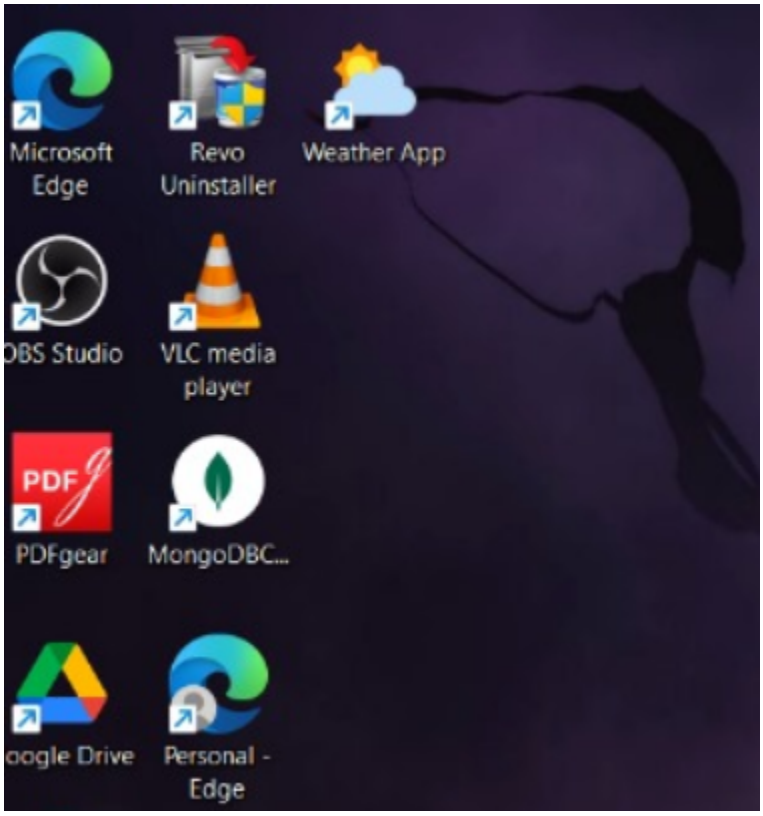
```
      ]
    },
    {
      "name": "Forecast",
      "short_name": "Forecast",
      "description": "View weather forecast",
      "url": "/?view=forecast",
      "icons": [
        {
          "src": "icons/icon-96x96.png",
          "sizes": "96x96"
        }
      ]
    }
  ]
}
```

Output:





Shortcut Add to Screen Button:



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.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

MPL Experiment 8 (PWA)

Name: Prathamesh Palve **Class:** D15A **Roll no:**31

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

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

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

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

What can’t we do with Service Workers?

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

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

Codes:

Serviceworker.js

```
// sw.js - Complete Service Worker for E-commerce PWA
const CACHE_NAME = 'ecommerce-pwa-v2';
const API_CACHE = 'ecommerce-api-v1';
const ASSETS_TO_CACHE = [
  '/',
  '/index.html',
  '/manifest.json',
  '/offline.html',
  '/css/main.min.css',
  '/js/app.min.js',
  '/icons/icon-192x192.png',
  '/icons/icon-512x512.png',
  '/images/placeholder-product.jpg'
];

// ======================
// Install Event
// ======================
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => {
        console.log('[Service Worker] Cache opened');
        return cache.addAll(ASSETS_TO_CACHE);
      })
      .then(() => self.skipWaiting())
  );
});
```

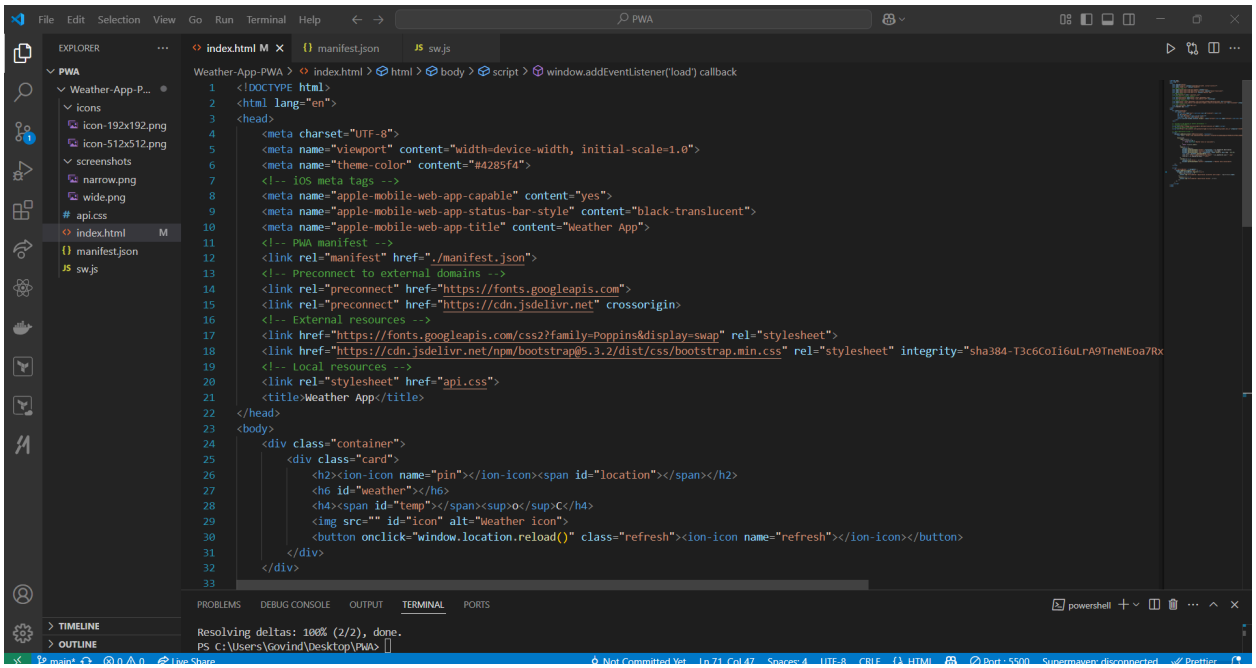
```
// ======================
// Activate Event
// ======================
self.addEventListener('activate', (event) => {
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cacheName) => {
          if (cacheName !== CACHE_NAME && cacheName !==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

// ======================
// Fetch Event
// ======================
self.addEventListener('fetch', (event) => {
  const { request } = event;
  const url = new URL(request.url);

  // 1. Skip non-GET requests and chrome-extension
  if (request.method !== 'GET' || url.protocol === 'chrome-extension:') {
    return;
  }

  // 2. API Requests (Network First with Cache Fallback)
  if (url.pathname.startsWith('/api/')) {
    event.respondWith(
      fetch(request)
        .then(networkResponse => {
          // Cache successful API responses
          if (networkResponse.ok) {
            const clone = networkResponse.clone();
            caches.open(API_CACHE)
              .then(cache => cache.put(request, clone));
          }
          return networkResponse;
```

```
      })
      .catch(() => {
        // Return cached version if available
        return caches.match(request)
          .then(cachedResponse => cachedResponse || Response.json(
            { error: 'Network error' },
            { status: 503 }
          ));
      })
  );
  return;
}

// 3. Static Assets (Cache First with Network Fallback)
event.respondWith(
  caches.match(request)
    .then(cachedResponse => {
      // Return cached version if found
      if (cachedResponse) {
        return cachedResponse;
      }

      // Otherwise fetch from network
      return fetch(request)
        .then(networkResponse => {
          // Cache successful responses
          if (networkResponse.ok) {
            const clone = networkResponse.clone();
            caches.open(CACHE_NAME)
              .then(cache => cache.put(request, clone));
          }
          return networkResponse;
        })
        .catch(() => {
          // Special handling for HTML pages
          if (request.headers.get('accept').includes('text/html')) {
            return caches.match('/offline.html');
          }
          // Return placeholder for images
          if (request.headers.get('accept').includes('image')) {
            return caches.match('/images/placeholder-product.jpg');
          }
        });
    })
```

```
  );
});

// ======================
// Background Sync
// ======================
self.addEventListener('sync', (event) => {
  if (event.tag === 'sync-cart') {
    event.waitUntil(
      // Get cart data from IndexedDB
      getCartData()
        .then(cartItems => {
          return fetch('/api/cart/sync', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(cartItems)
          });
        })
        .then(() => {
          return showNotification('Cart Synced', 'Your cart has been updated');
        })
        .catch(err => {
          console.error('Sync failed:', err);
        })
    );
  }
});

// ======================
// Push Notifications
// ======================
self.addEventListener('push', (event) => {
  let data = {};
  try {
    data = event.data.json();
  } catch (e) {
    data = {
      title: 'New Update',
      body: 'Check out our latest products!',
      icon: '/icons/icon-192x192.png',
      url: '/'
    };
  }
```

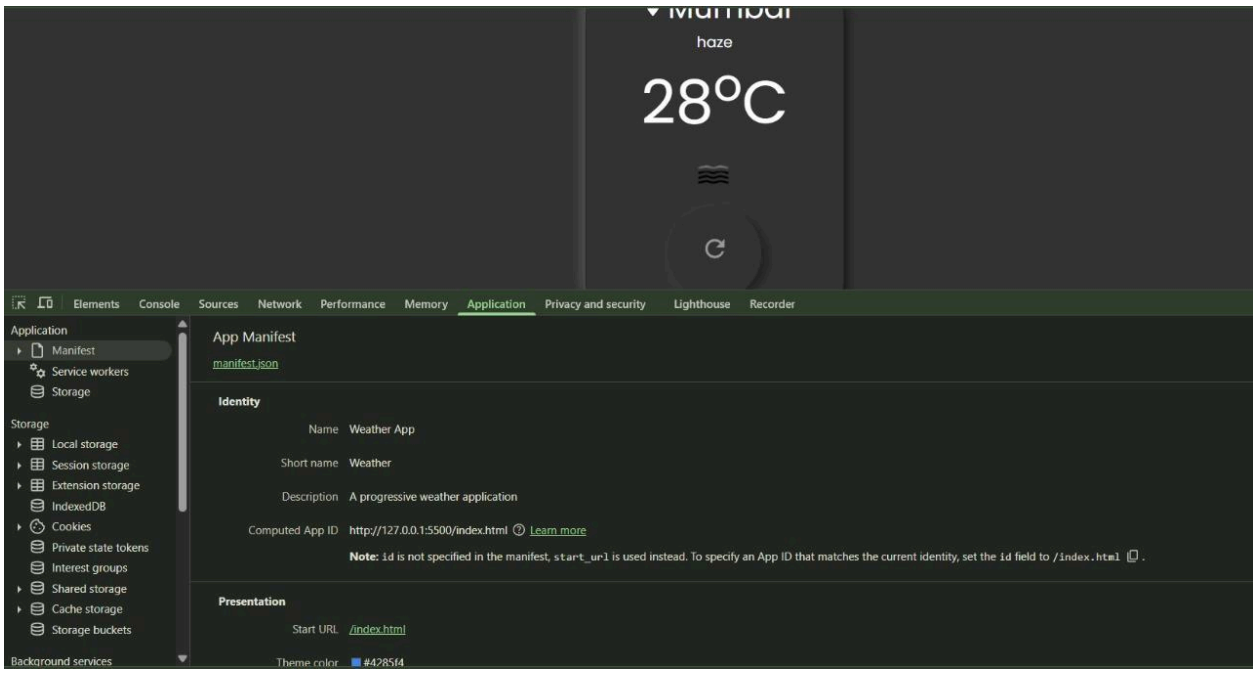
```
  const options = {
    body: data.body,
    icon: data.icon || '/icons/icon-192x192.png',
    badge: '/icons/icon-96x96.png',
    data: {
      url: data.url || '/'
    }
  };

  event.waitUntil(
    self.registration.showNotification(data.title, options)
  );
});

self.addEventListener('notificationclick', (event) => {
  event.notification.close();
  event.waitUntil(
    clients.matchAll({ type: 'window' })
      .then(clientList => {
        for (const client of clientList) {
          if (client.url === event.notification.data.url && 'focus' in client) {
            return client.focus();
          }
        }
        if (clients.openWindow) {
          return clients.openWindow(event.notification.data.url);
        }
      })
  );
});

// ======================
// Helper Functions
// ======================
async function getCartData() {
  // In a real app, you would use IndexedDB
  return new Promise(resolve => {
    resolve([]);
  });
}

async function showNotification(title, body) {
  return self.registration.showNotification(title, { body });
}
```

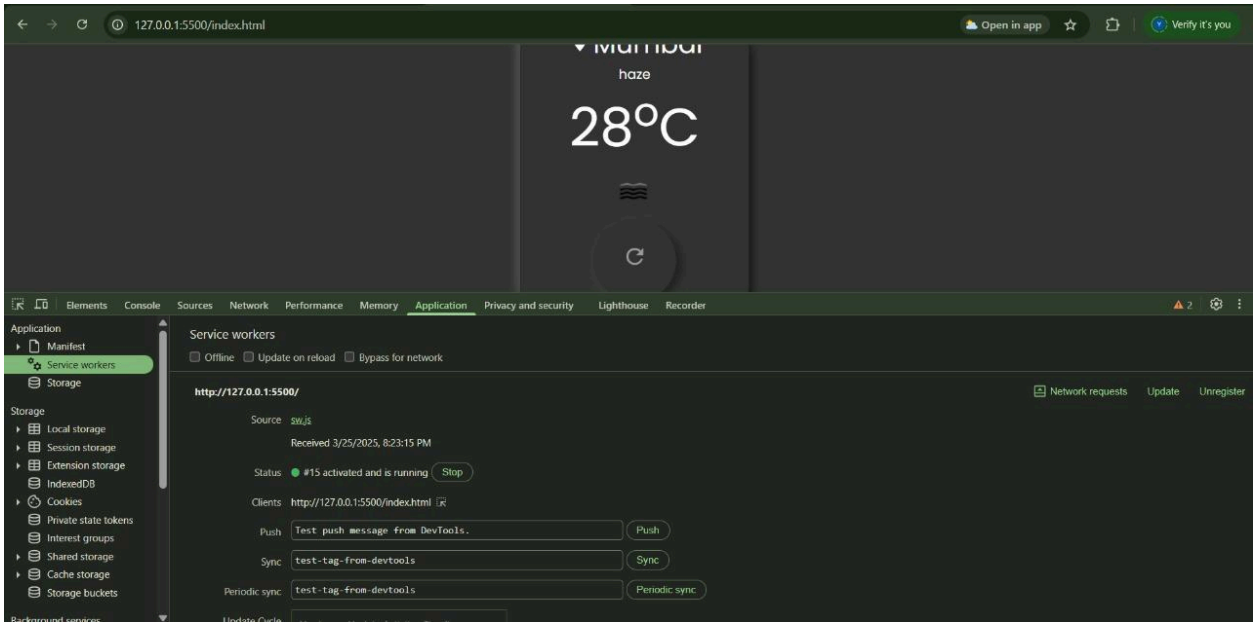
Output:


File Edit Selection View Go Run Terminal Help
PWA
EXPLORER
index.html M
manifest.json
sw.js
Weather-App-PWA > index.html > html > body > script > window.addEventListener('load') callback
PWA
Weather-App-P...
icons
icon-192x192.png
icon-512x512.png
screenshots
narrow.png
wide.png
api.css
index.html M
manifest.json
sw.js
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="theme-color" content="#4285f4">
<!-- iOS meta tags -->
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">
<meta name="apple-mobile-web-app-title" content="Weather App">
<!-- PWA manifest -->
<link rel="manifest" href="./manifest.json">
<!-- Preconnect to external domains -->
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://cdn.jsdelivr.net" crossorigin>
<!-- External resources -->
<link href="https://fonts.googleapis.com/css2?family=Poppins&display=swap" rel="stylesheet">
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7Rx
<!-- Local resources -->
<link rel="stylesheet" href="api.css">
<title>Weather App</title>
</head>
<body>

<h2><ion-icon name="pin"></ion-icon></h2>
<h6 id="weather"></h6>
<h4>^oC</h4>
<img src="" id="icon" alt="Weather icon">
<button onclick="window.location.reload()" class="refresh"><ion-icon name="refresh"></ion-icon></button>

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS
powershell
Resolving deltas: 100% (2/2), done.
PS C:\Users\Govind\Desktop\PWA>
TIMELINE
OUTLINE


haze
28°C
Elements Console Sources Network Performance Memory Application Privacy and security Lighthouse Recorder
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
App Manifest
manifest.json
Identity
Name Weather App
Short name Weather
Description A progressive weather application
Computed App ID http://127.0.0.1:5500/index.html Learn more
Note: id is not specified in the manifest, start_url is used instead. To specify an App ID that matches the current identity, set the id field to /index.html .
Presentation
Start URL /index.html
Theme color #4285f4


127.0.0.1:5500/index.html
Open in app
Verify it's you
haze
28°C
Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Storage buckets
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5500/
Network requests
Update
Unregister
Source
sw.js
Received 3/25/2025, 8:23:15 PM
Status
#15 activated and is running
Stop
Clients
http://127.0.0.1:5500/index.html
Push
Test push message from DevTools.
Push
Sync
test-tag-from-devtools
Sync
Periodic sync
test-tag-from-devtools
Periodic sync

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.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

MPL Experiment 9 (PWA)

Name: Prathamesh Palve **Class:** D15A **Roll no:**31

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.

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

Code:

Serviceworker.js

```
// sw.js - Complete Service Worker for E-commerce PWA
const CACHE_NAME = 'ecommerce-pwa-v2';
const API_CACHE = 'ecommerce-api-v1';
const ASSETS_TO_CACHE = [
  '/',
  '/index.html',
  '/manifest.json',
  '/offline.html',
  '/css/main.min.css',
  '/js/app.min.js',
  '/icons/icon-192x192.png',
  '/icons/icon-512x512.png',
  '/images/placeholder-product.jpg'
```

```
];

// ======================
// Install Event
// ======================
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => {
        console.log('[Service Worker] Cache opened');
        return cache.addAll(ASSETS_TO_CACHE);
      })
      .then(() => self.skipWaiting())
  );
});

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

// ======================
// Fetch Event
// ======================
self.addEventListener('fetch', (event) => {
  const { request } = event;
  const url = new URL(request.url);

  // 1. Skip non-GET requests and chrome-extension
  if (request.method !== 'GET' || url.protocol === 'chrome-extension:') {
```

```
  return;
}

// 2. API Requests (Network First with Cache Fallback)
if (url.pathname.startsWith('/api/')) {
  event.respondWith(
    fetch(request)
      .then(networkResponse => {
        // Cache successful API responses
        if (networkResponse.ok) {
          const clone = networkResponse.clone();
          caches.open(API_CACHE)
            .then(cache => cache.put(request, clone));
        }
        return networkResponse;
      })
      .catch(() => {
        // Return cached version if available
        return caches.match(request)
          .then(cachedResponse => cachedResponse || Response.json(
            { error: 'Network error' },
            { status: 503 }
          ));
      })
  );
  return;
}

// 3. Static Assets (Cache First with Network Fallback)
event.respondWith(
  caches.match(request)
    .then(cachedResponse => {
      // Return cached version if found
      if (cachedResponse) {
        return cachedResponse;
      }

      // Otherwise fetch from network
      return fetch(request)
        .then(networkResponse => {
          // Cache successful responses
          if (networkResponse.ok) {
            const clone = networkResponse.clone();
            caches.open(CACHE_NAME)
```

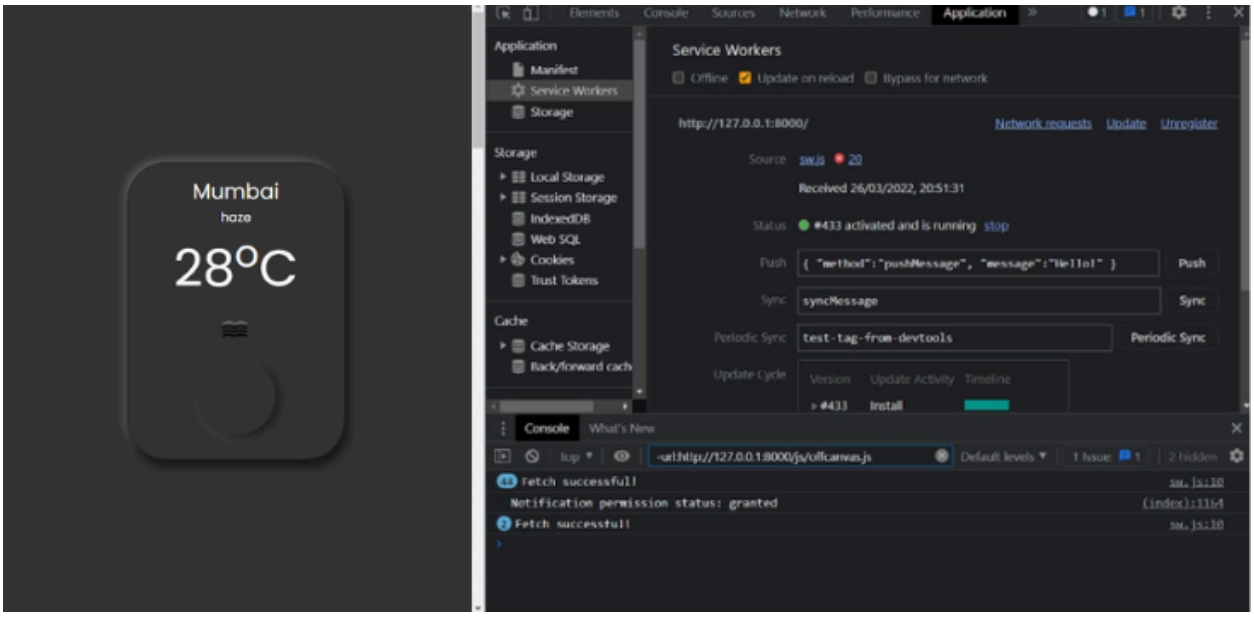
```
            .then(cache => cache.put(request, clone));
        }
        return networkResponse;
      })
      .catch(() => {
        // Special handling for HTML pages
        if (request.headers.get('accept').includes('text/html')) {
          return caches.match('/offline.html');
        }
        // Return placeholder for images
        if (request.headers.get('accept').includes('image')) {
          return caches.match('/images/placeholder-product.jpg');
        }
      });
    })
  );
});

// ======================
// Background Sync
// ======================
self.addEventListener('sync', (event) => {
  if (event.tag === 'sync-cart') {
    event.waitUntil(
      // Get cart data from IndexedDB
      getCartData()
        .then(cartItems => {
          return fetch('/api/cart/sync', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(cartItems)
          });
        })
        .then(() => {
          return showNotification('Cart Synced', 'Your cart has been updated');
        })
        .catch(err => {
          console.error('Sync failed:', err);
        })
    );
  }
});

// ======================
```

```
// Push Notifications
// ======================
self.addEventListener('push', (event) => {
  let data = {};
  try {
    data = event.data.json();
  } catch (e) {
    data = {
      title: 'New Update',
      body: 'Check out our latest products!',
      icon: '/icons/icon-192x192.png',
      url: '/'
    };
  }

  const options = {
    body: data.body,
    icon: data.icon || '/icons/icon-192x192.png',
    badge: '/icons/icon-96x96.png',
    data: {
      url: data.url || '/'
    }
  };

  event.waitUntil(
    self.registration.showNotification(data.title, options)
  );
});

self.addEventListener('notificationclick', (event) => {
  event.notification.close();
  event.waitUntil(
    clients.matchAll({ type: 'window' })
      .then(clientList => {
        for (const client of clientList) {
          if (client.url === event.notification.data.url && 'focus' in client) {
            return client.focus();
          }
        }
        if (clients.openWindow) {
          return clients.openWindow(event.notification.data.url);
        }
      })
  );
```

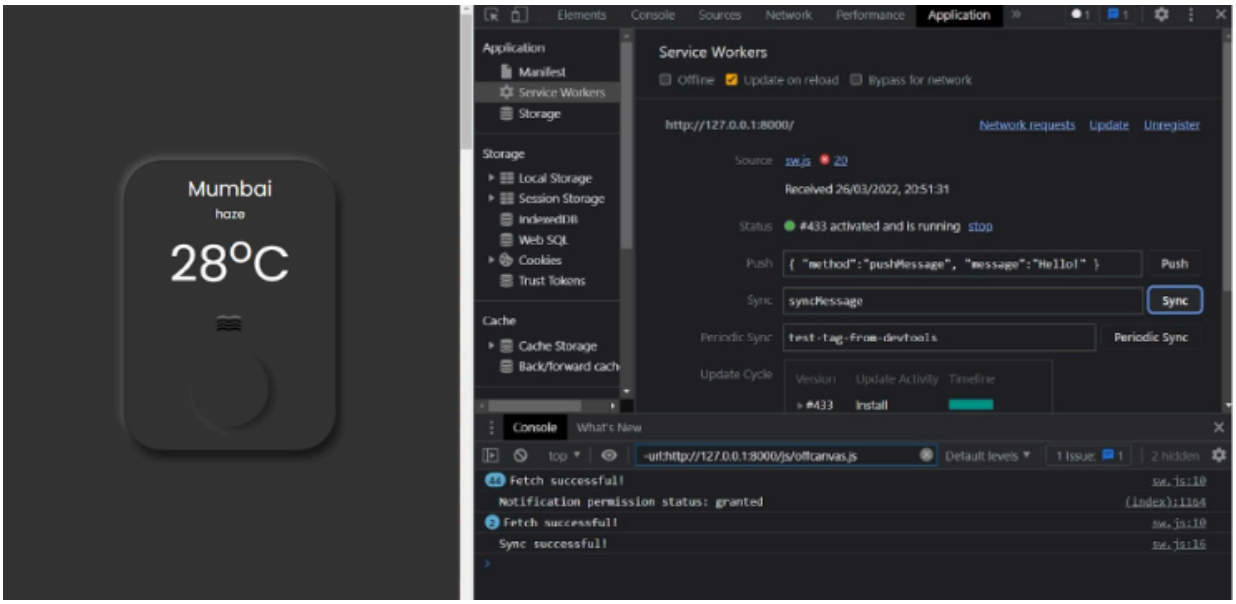
```
});

// ======================
// Helper Functions
// ======================
async function getCartData() {
  // In a real app, you would use IndexedDB
  return new Promise(resolve => {
    resolve([]);
  });
}

async function showNotification(title, body) {
  return self.registration.showNotification(title, { body });
}
```

Output:

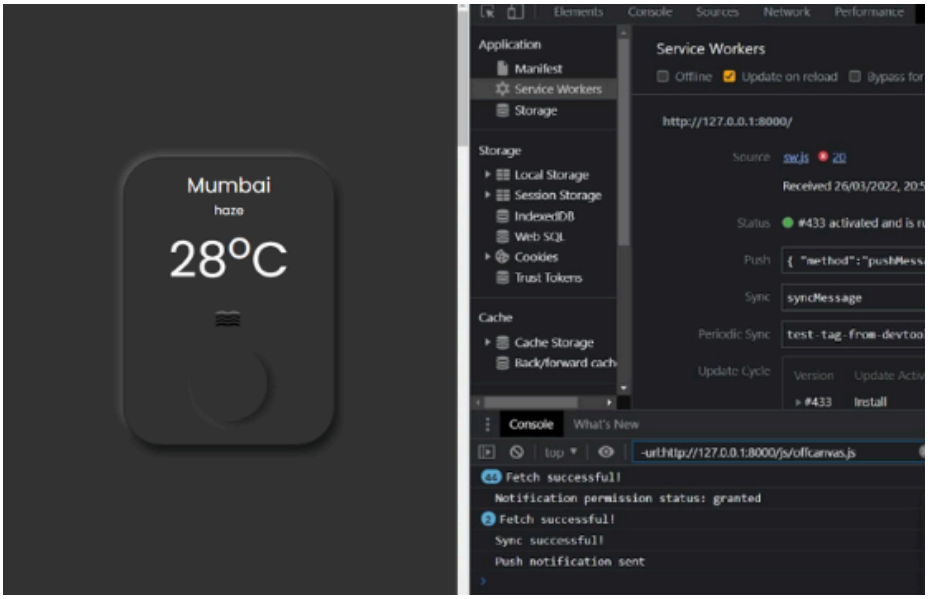
Fetch Event



Sync Event



Push event



MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

MPL Experiment 10 (PWA)

Name: Prathamesh Palve **Class:** D15A **Roll no:**31

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

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:

- Blogging with Jekyll
- Custom URL
- Automatic Page Generator

Reasons for favoring this over Firebase:

- Free to use
- Right out of github
- 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

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

Cons

- The code of your website will be public, unless you pay for a private repository.
- Currently, there is no support for HTTPS for custom domains. It's probably coming soon though.
- Although Jekyll is supported, plug-in support is rather spotty.
- Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

- Realtime backend made easy
- 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

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

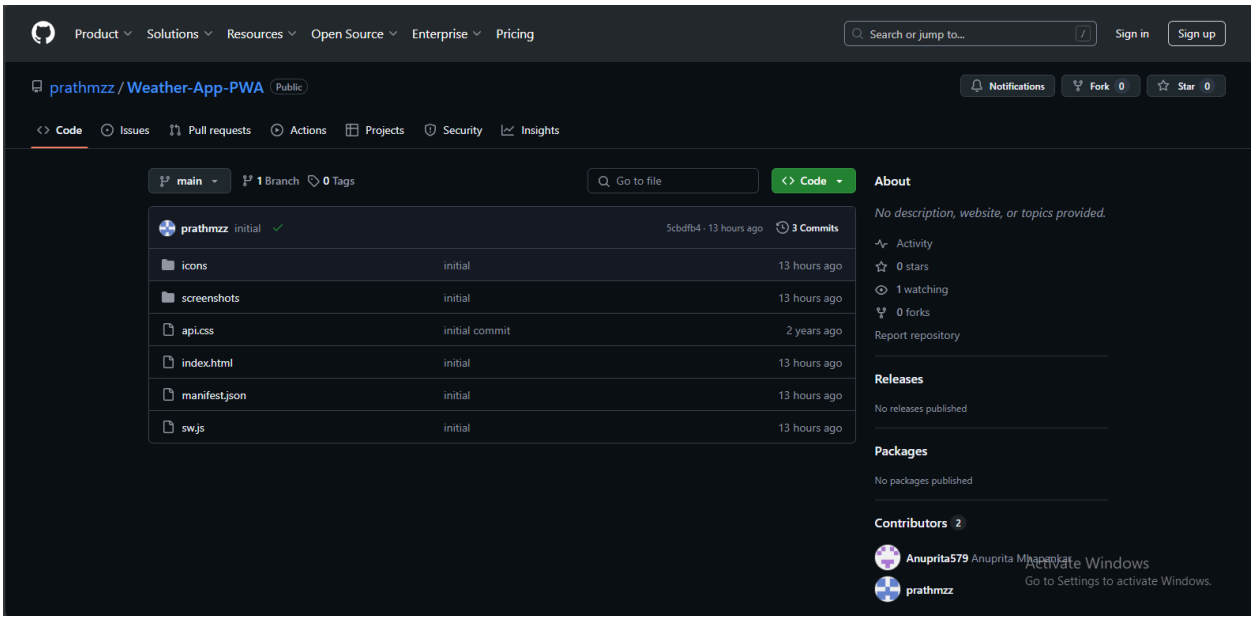
Cons

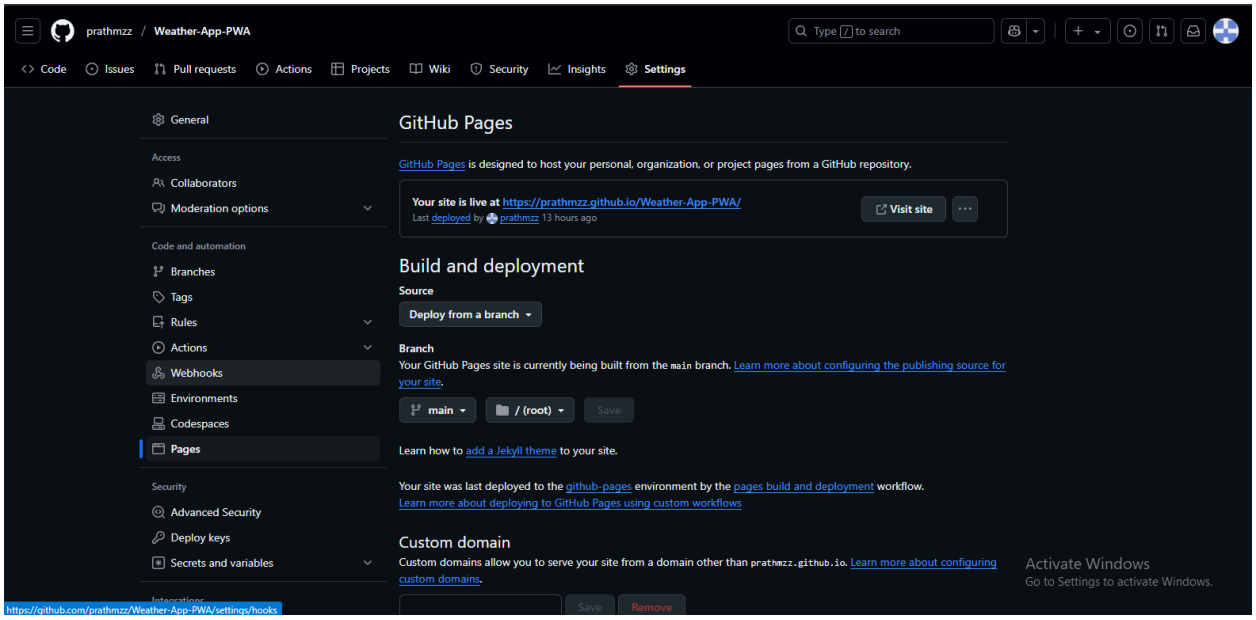
- Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
- Command-line interface only.
- No in-built support for any static site generator.

Link to our GitHub repository:
https://github.com/prathmzz/Weather-App-PWA

Link to our Hosted website:
https://prathmzz.github.io/Weather-App-PWA/

Github Screenshot:





MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

MPL Experiment 11 (PWA)

Name: Prathamesh Palve **Class:** D15A **Roll no:**31

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

Theory:

Google Lighthouse :

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

The best part is that you have to set up almost nothing to get started. Let's begin by looking at some of the top features and audit criteria used by Lighthouse.

Key Features and Audit Metrics

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

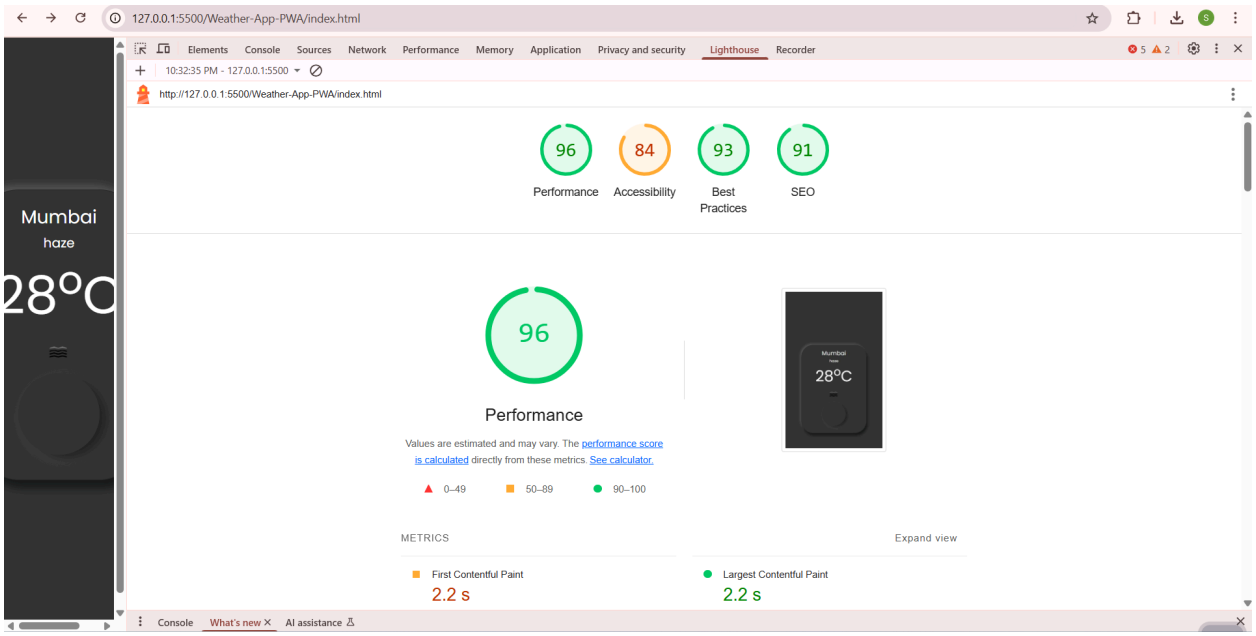
- **Performance:**
  This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score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

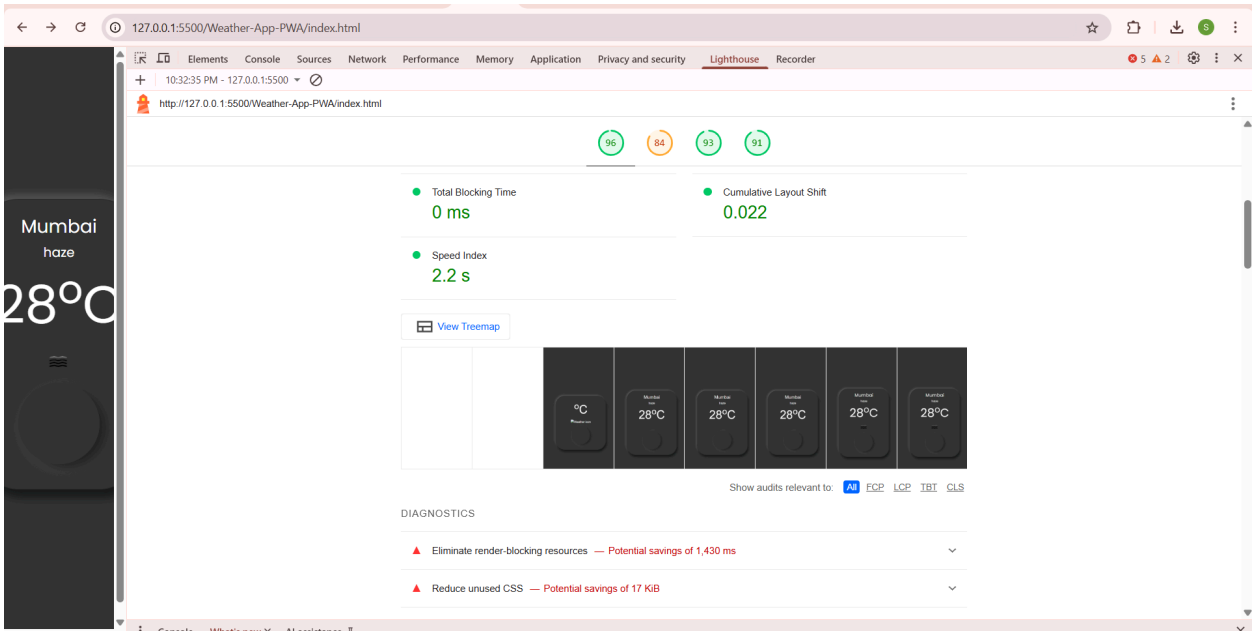
- **PWA Score (Mobile):**
  Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards the PWA paradigm, where the objective is to make the application behave as close as possible to native mobile

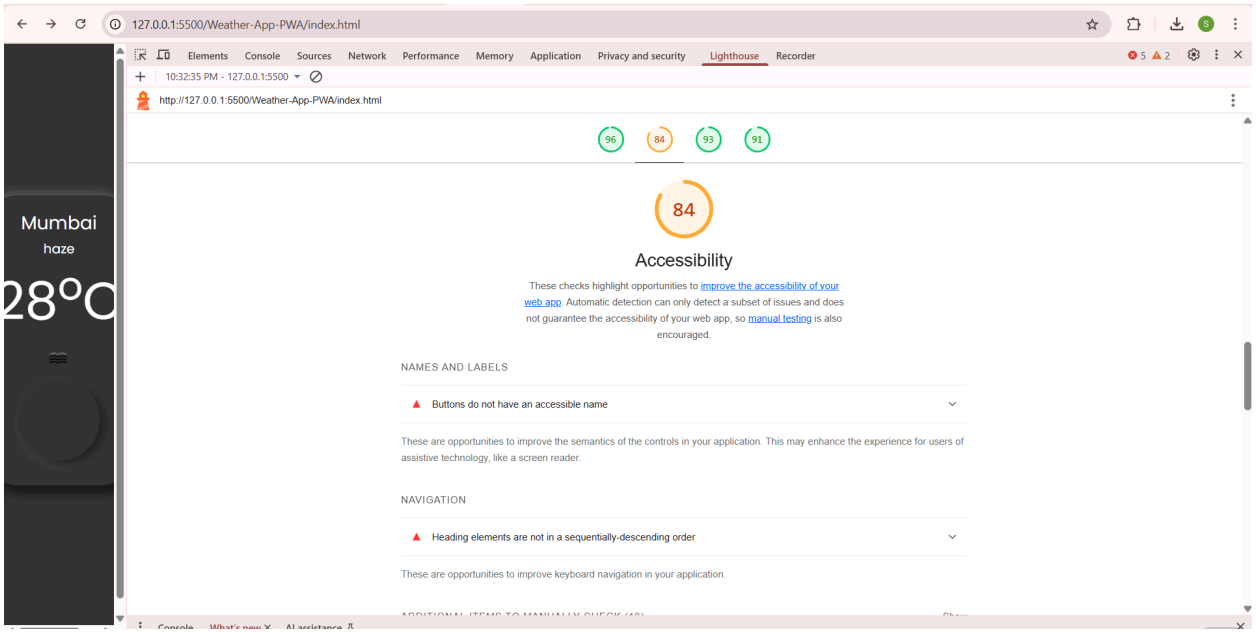
applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled environments, etc.

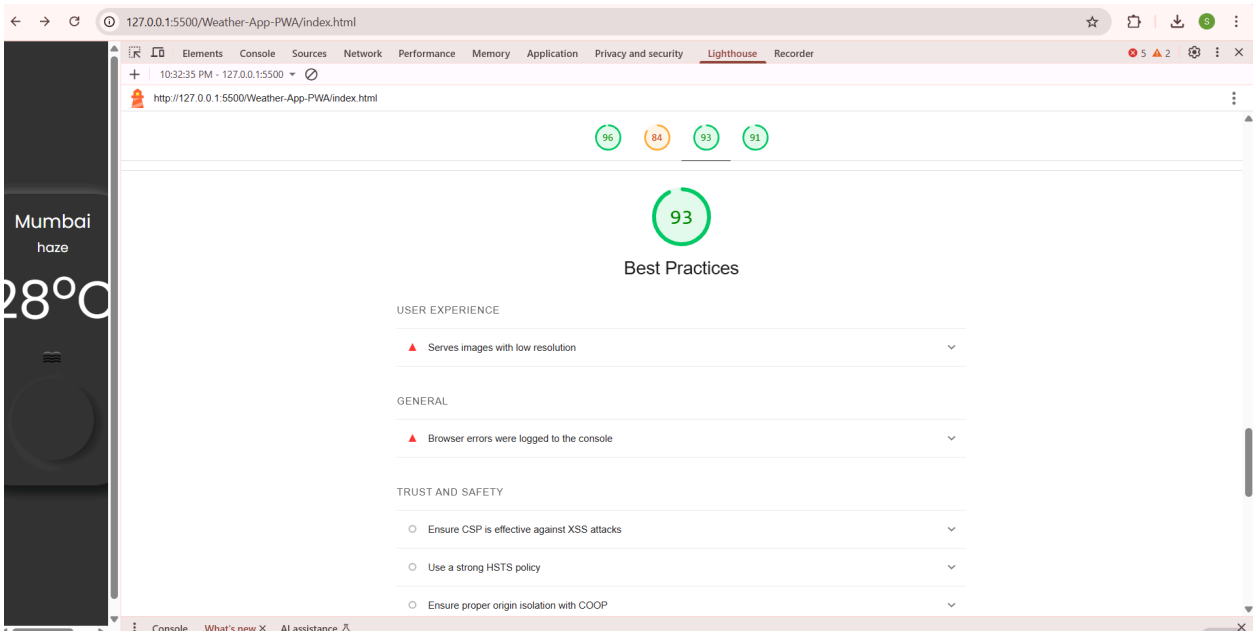
- **Accessibility:**
  As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the ‘aria-’ attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.
  Best Practices: As any developer would know, there are a number of practices that have been deemed ‘best’ based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS

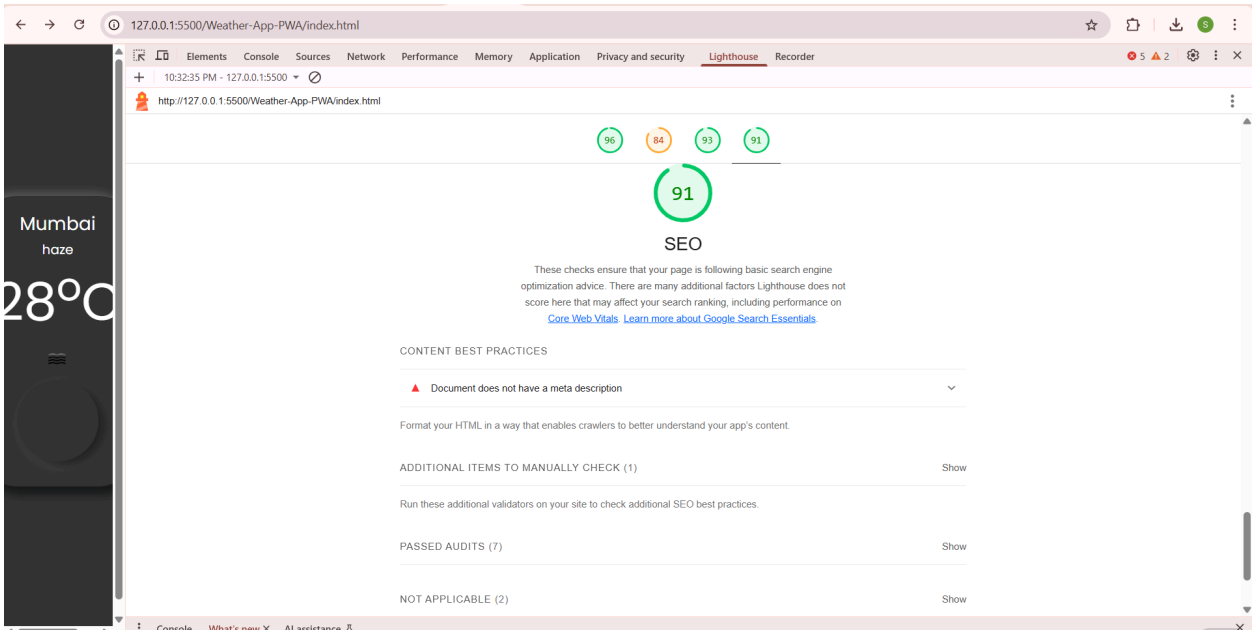
0utput:


127.0.0.1:5500/Weather-App-PWA/index.html
Elements Console Sources Network Performance Memory Application Privacy and security Lighthouse Recorder
http://127.0.0.1:5500/Weather-App-PWA/index.html
Mumbai
haze
28°C
96
Performance
84
Accessibility
93
Best Practices
91
SEO
96
Performance
Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.
0–49
50–89
90–100
METRICS
Expand view
First Contentful Paint
2.2 s
Largest Contentful Paint
2.2 s
Console
What's new
AI assistance


127.0.0.1:5500/Weather-App-PWA/index.html
Elements Console Sources Network Performance Memory Application Privacy and security Lighthouse Recorder
http://127.0.0.1:5500/Weather-App-PWA/index.html
Mumbai
haze
28°C
Total Blocking Time
0 ms
Cumulative Layout Shift
0.022
Speed Index
2.2 s
View Treemap
Show audits relevant to:
All
FCP
LCP
TBT
CLS
DIAGNOSTICS
Eliminate render-blocking resources — Potential savings of 1,430 ms
Reduce unused CSS — Potential savings of 17 KiB


127.0.0.1:5500/Weather-App-PWA/index.html
Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
http://127.0.0.1:5500/Weather-App-PWA/index.html
96
84
93
91
Mumbai
haze
28°C
84
Accessibility
These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.
NAMES AND LABELS
Buttons do not have an accessible name
These are opportunities to improve the semantics of the controls in your application. This may enhance the experience for users of assistive technology, like a screen reader.
NAVIGATION
Heading elements are not in a sequentially-descending order
These are opportunities to improve keyboard navigation in your application.


127.0.0.1:5500/Weather-App-PWA/index.html
Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
http://127.0.0.1:5500/Weather-App-PWA/index.html
96
84
93
91
Mumbai
haze
28°C
93
Best Practices
USER EXPERIENCE
Serves images with low resolution
GENERAL
Browser errors were logged to the console
TRUST AND SAFETY
Ensure CSP is effective against XSS attacks
Use a strong HSTS policy
Ensure proper origin isolation with COOP



Conclusion:

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

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.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

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.	31
Name	Prathamesh Palve
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	