Aarya Chandhi
D15B / 58

MP Lab 1

Problem statement

Public misconduct and violations often go unreported due to the lack of an easy, accessible and accountable reporting mechanism. Current systems are ~~neither~~ either inefficient, slow or lack transparency. Citizens needs a real time platform to document, report, and track complaints regarding improper actions in public spaces, ensuring swift action by authorities.

Overview of the application

The applications acts as a public complaint platform where users can report improper actions or violence occuring in public spaces. Through this app users can ~~click~~ click photos via their mobile phone, describe the incident and send the info. directly to the concerned authorities.

The app emphasizes transparency, accountability, and ease of reporting enabling quick action against public misconduct or violations.

Pros of the application

1. Real time Reporting
2. Provides Evidence based Complaints.
3. Empowering citizens

Cons of the application
1. False Reporting
2. Potential misuse
3. Resource Intensity

Features of the app
1. Real time Complaint
2. Live location capture
3. Allows users to describe the event (misconduct happening
4. ~~Re~~

Similar apps and how they differ
1. SeeClickFix: Focuses on infrastructure issues (e.g. potholes, broken streetlights).
2. Citizen App: Provides crime alerts but lacks direct ~~inte~~ submission.

Conclusion: the public complaint app empowers citizen to report misconduct efficiently with photo/video evidence, GPS tracking and direct submission to authorities. While challenges like privacy and false reporting may be concerned, they can be mitigated with strong security and verification measures.
This app can enhance public safety, civic management making communities safer and more accountable.

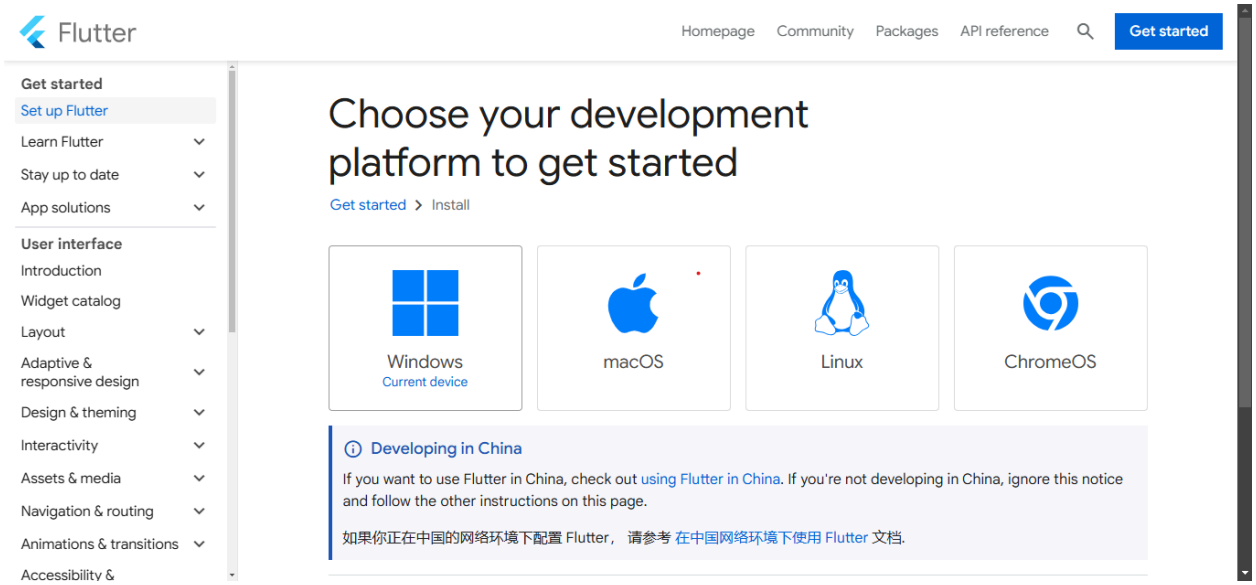
Name:Aarya Gandhi
Class:D15B
Roll no. 14

Aim:Installation and Configuration of Flutter Environment.

Theory:

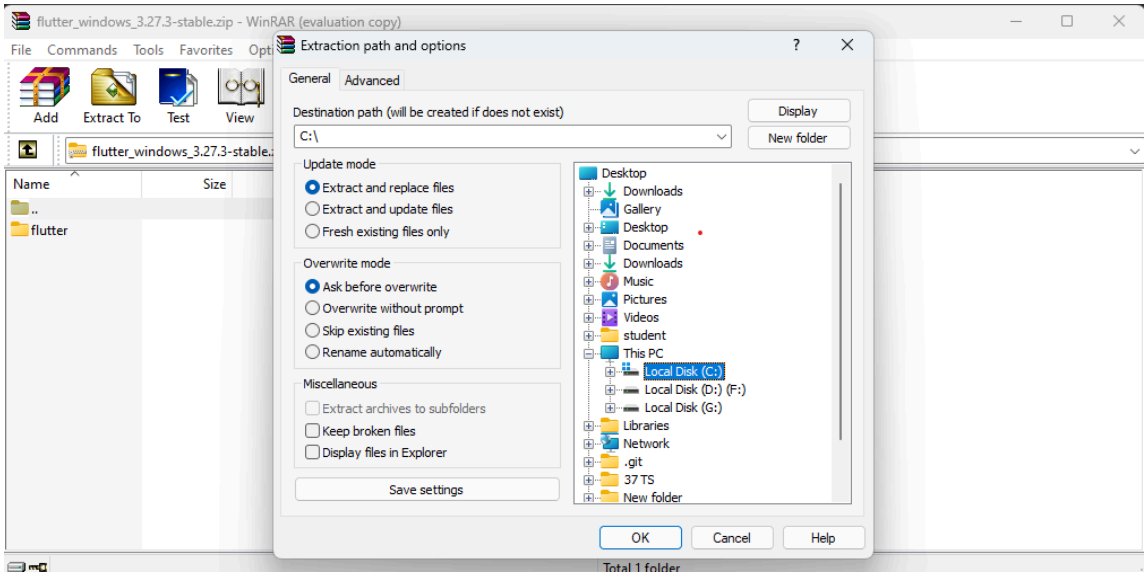
Install the Flutter SDK

Step 1: Download the installation bundle of the Flutter Software Development Kit for windows. To download Flutter SDK, Go to its official website https://docs.flutter.dev/get-started/install , you will get the following screen.



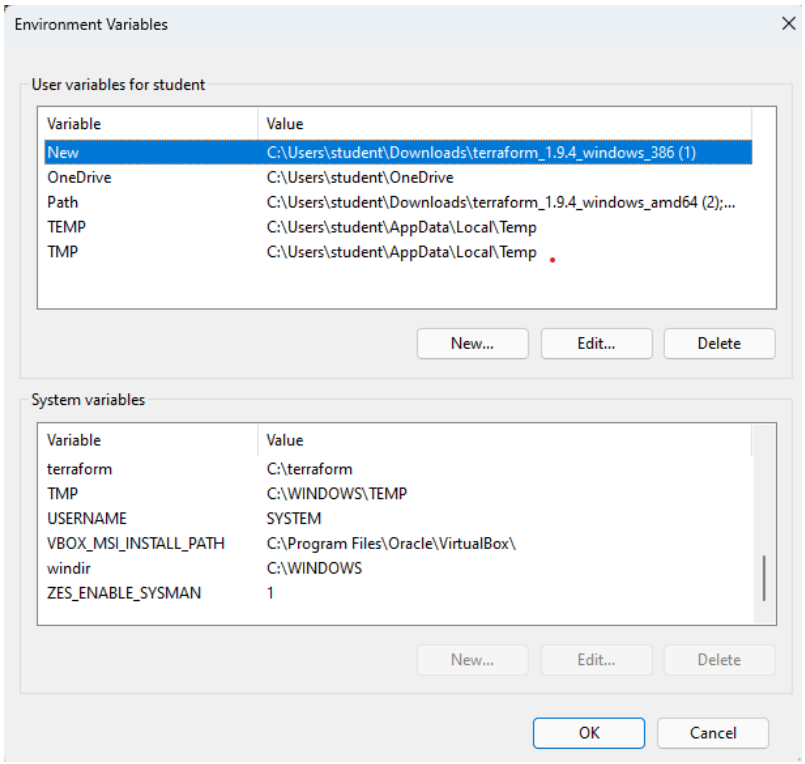
Step 2: Next, to download the latest Flutter SDK, click on the Windows icon. Here, you will find the download link for SDK.

Step 3: When your download is complete, extract the zip file and place it in the desired installation folder or location, for example, C: /Flutter.

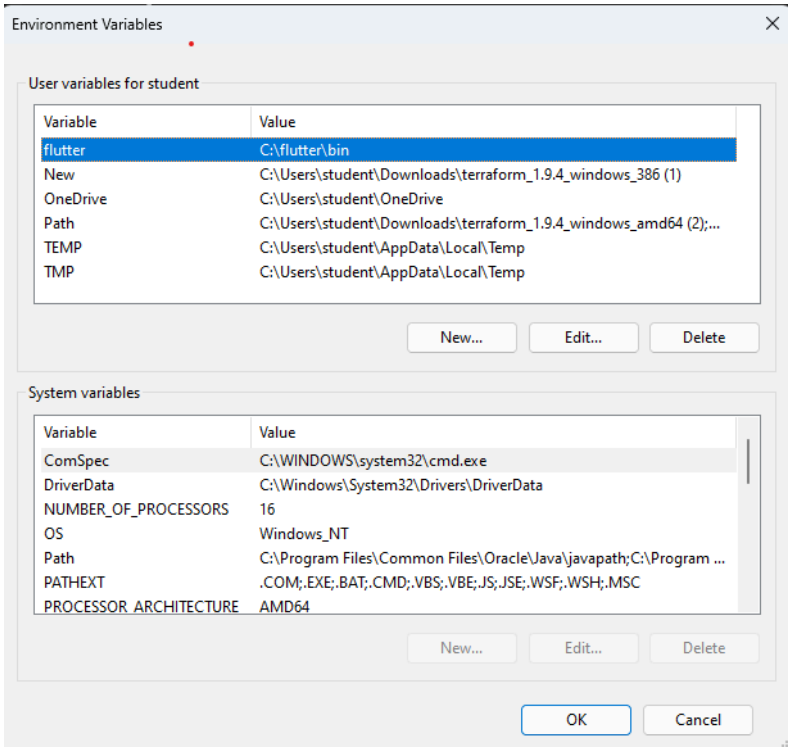


Step 4: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:

Step 4.1: Go to MyComputer properties -> advanced tab -> environment variables. You will get the following screen.

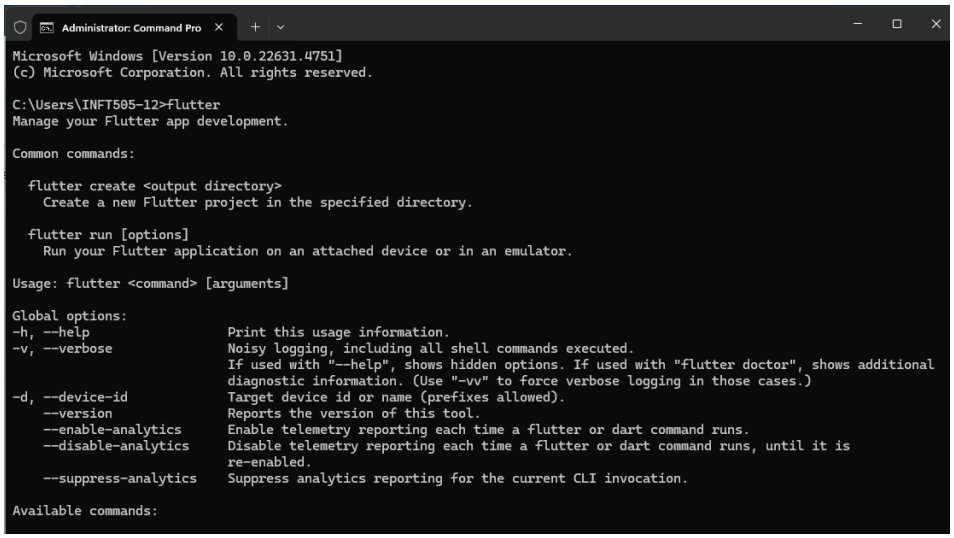


Step 4.2: Now, select path -> click on edit. The following screen appears



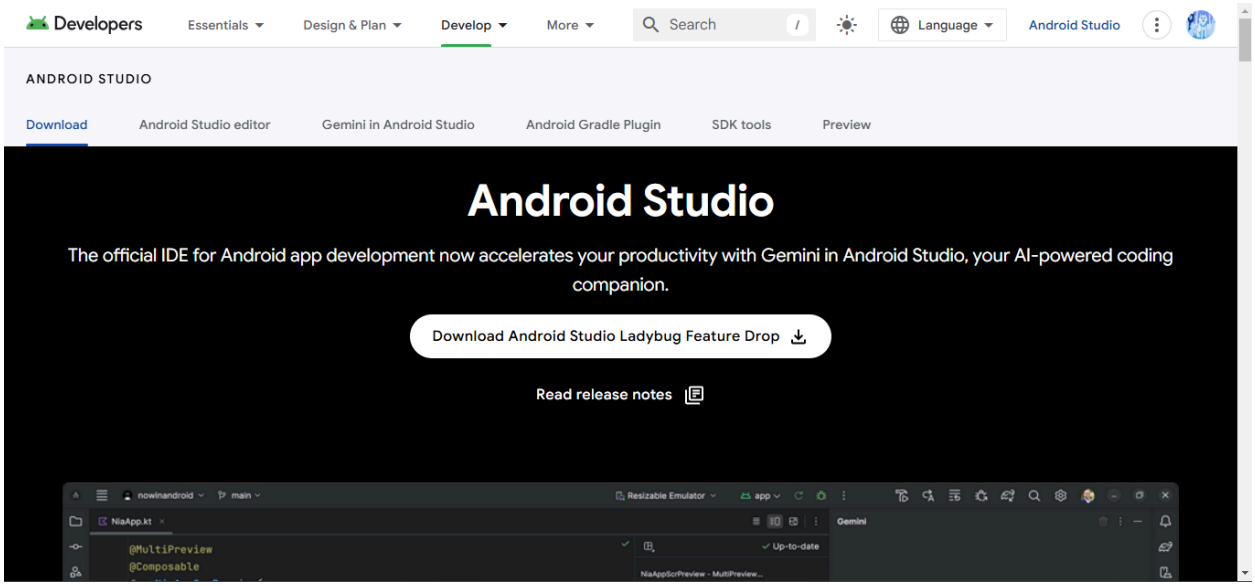
Step 5: Now, run the $ flutter command in command prompt.

Now, run the $ flutter doctor command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation.

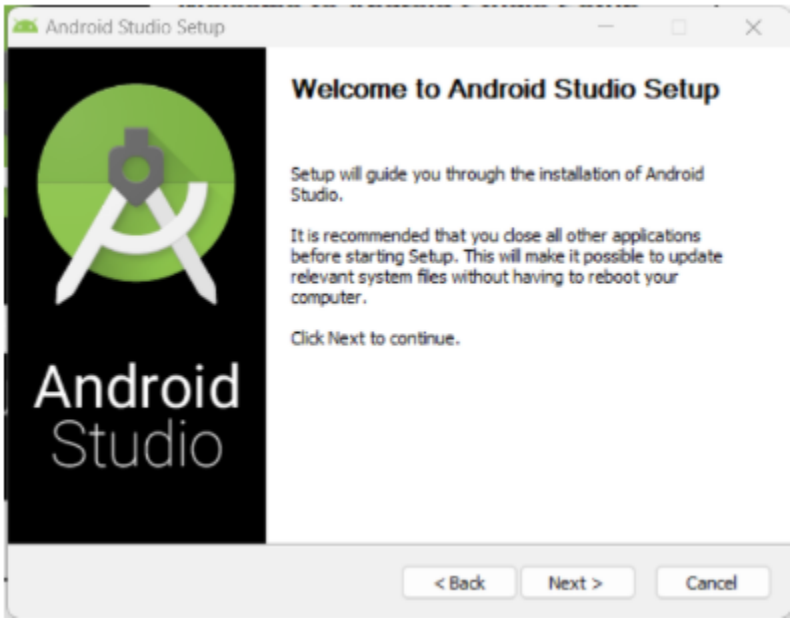


Step 7: Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.

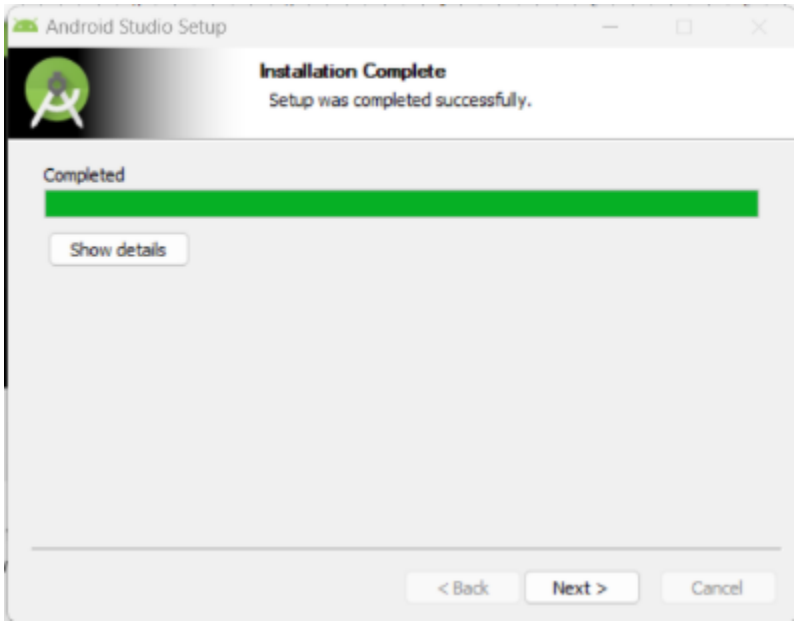
Step 7.1: Download the latest Android Studio executable or zip file from the official site.



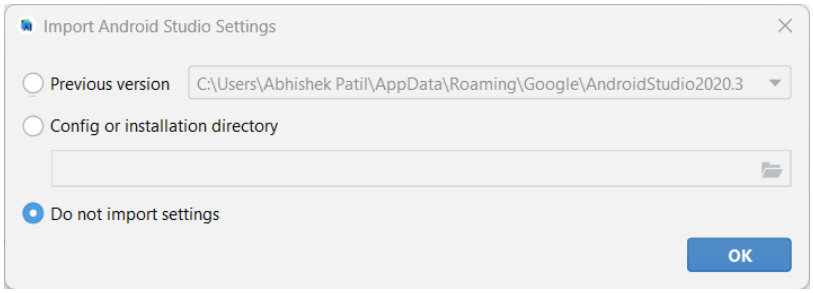
Step 7.2: When the download is complete, open the .exe file and run it. You will get the following dialog box.



Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



Step 7.4: In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to
choose the 'Don't import Settings option' and click OK. It will start the Android Studio.

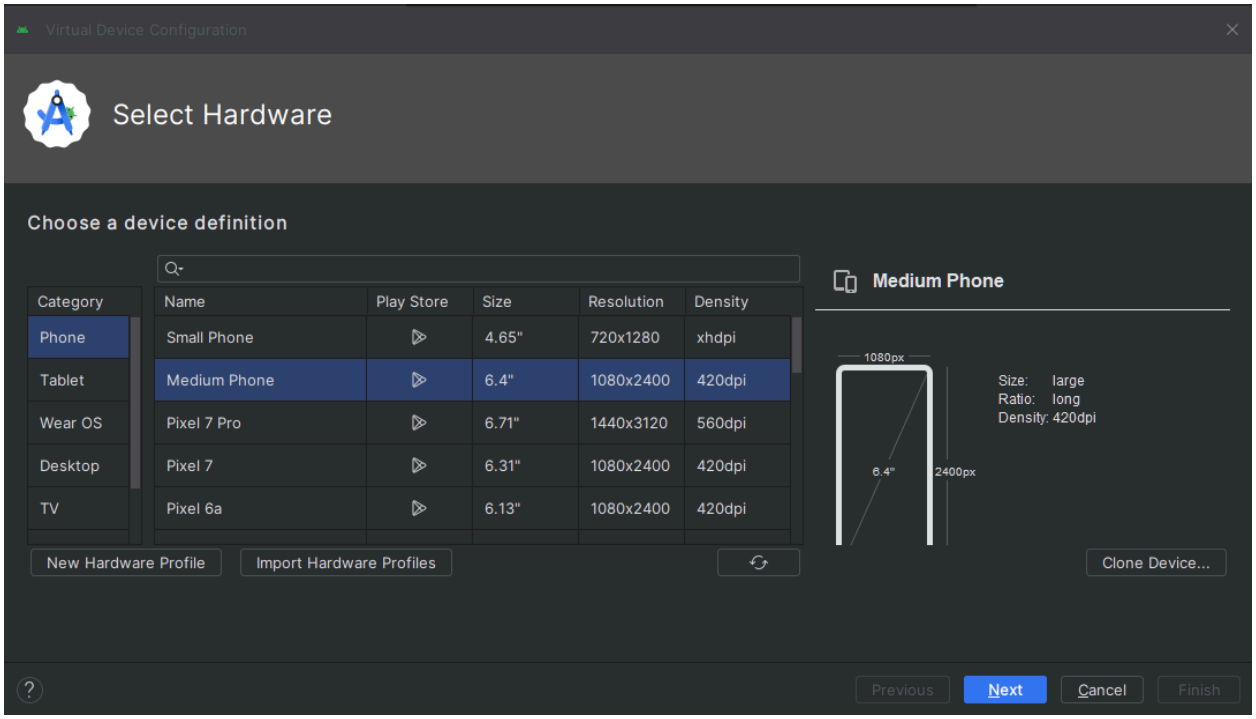


Step 7.5: run the $ flutter doctor command and Run flutter doctor --android-licenses command.

```
C:\Users\INFT505-12> flutter doctor --android-licenses
Warning: Errors during XML parse:
Warning: Additionally, the fallback loader failed to parse the XML.ry...
Warning: Errors during XML parse:
Warning: Additionally, the fallback loader failed to parse the XML.ry...
[======================================] 100% Computing updates...
All SDK package licenses accepted.
```

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

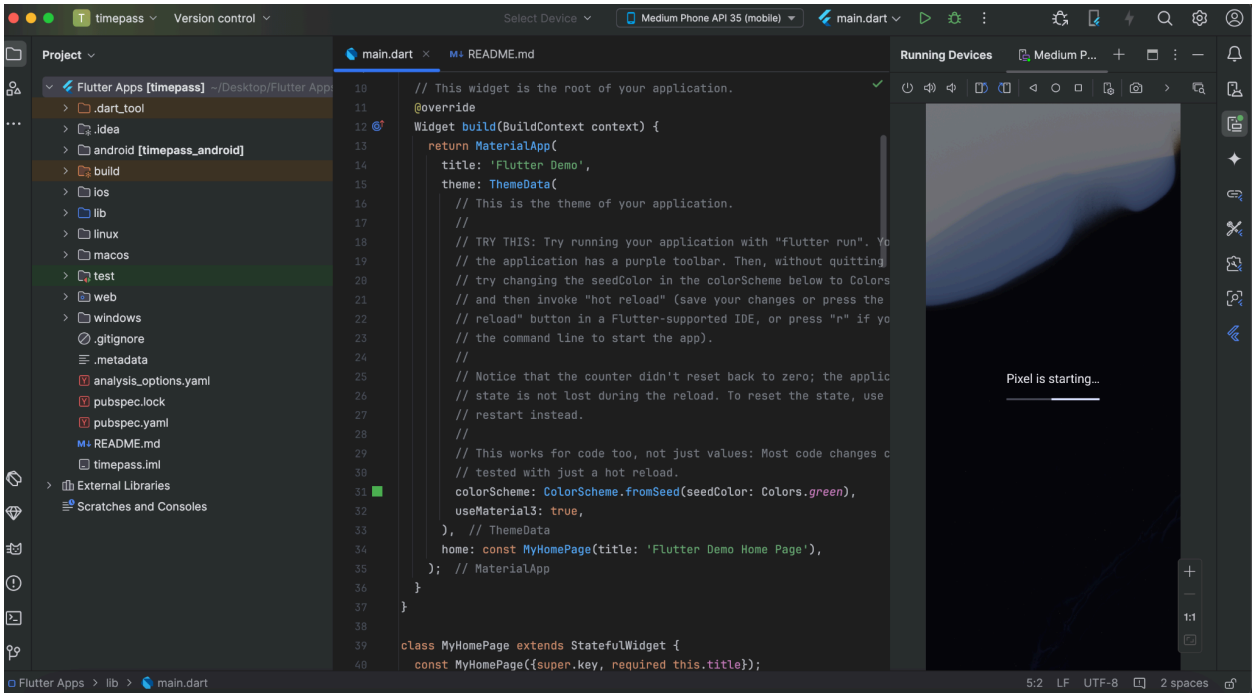
Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.



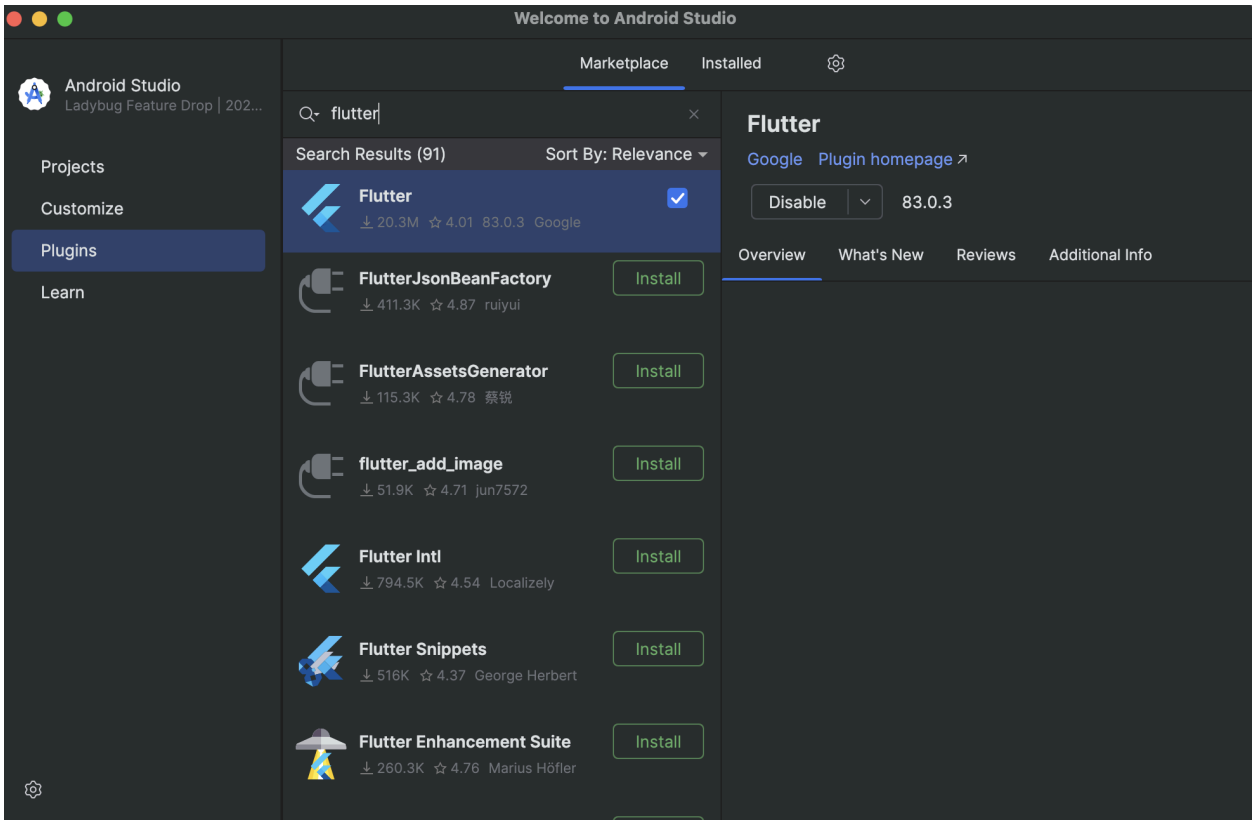
Step 8.3: Select the system image for the latest Android version and click on Next.

Step 8.4: Now, verify the all AVD configuration. If it is correct, click on Finish. The following screen appears.

Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.



Step 9: Now, install Flutter and Dart plugin for building Flutter application in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.



Step 9.1: Open the Android Studio and then go to File->Settings->Plugins.

Step 9.2: Now, search the Flutter plugin. If found, select Flutter plugin and click install. When you click on install, it will ask you to install Dart plugin as below screen. Click yes to proceed.

Step 9.3: Restart the Android Studio.

Conclusion: In conclusion, setting up the Flutter environment ensures a smooth development process by installing the SDK, configuring dependencies, and verifying the setup with flutter doctor.
Once complete, developers can efficiently build and test cross-platform applications.

Aarya Gandhi
D15B/14

MPL Experiment 10.2

Aim: To design Flutter UI by including common widgets.

Theory:

Flutter UI Design Using Common widgets

Flutter provides a comprehensive set of widgets to create visually appealing and interactive user interfaces. These widgets are categorized into stateless and stateful widgets.

1. Stateless Widgets

A stateless widget is immutable, meaning its properties cannot change once it is built. These widgets do not maintain any internal state and only update when external data triggers a rebuild.

2. Stateful widgets

A stateful widget can change dynamically based on user interactions or internal logic. These widgets maintain state, allowing them to be updated and re-rendered whenever necessary.

Flutter widgets

- Scaffold :- Provides the basic structure of screen, including an AppBar and body
- AppBar :- Displays a title for the signup page
- textField :- collects user input for username, email & password

Conclusion: In conclusion, Flutter's diverse widget set enables developers to create responsive, modern and engaging UIs, with ea

Name:Aarya Gandhi
Roll No. 14
Class: D15B

MPL Experiment 02

Aim:To design Flutter UI by including common widgets.

Theory:

Flutter UI Design Using Common Widgets

Flutter provides a comprehensive set of widgets to create visually appealing and interactive user interfaces. These widgets are categorized into **stateless** and **stateful** widgets.

- **Stateless Widgets** remain constant once built and do not update dynamically.
- **Stateful Widgets** can change based on user interaction or internal state changes.

Common Flutter Widgets

1. Basic Structural Widgets

These widgets provide the foundational layout structure of the app.

- **Scaffold** – Serves as the main layout structure with built-in support for `AppBar`, `Body`, and `FloatingActionButton`.
- **AppBar** – A top navigation bar that contains a title and optional action buttons.
- **Container** – A box-like widget used for styling, including background color, padding, and margins.
- **Column & Row** – Used to arrange child widgets in a vertical or horizontal direction, respectively.

2. User Input Widgets

These widgets allow users to provide input and interact with the application.

- **TextField** – Accepts user input in the form of text.
- **DropdownButton** – Displays a dropdown menu to select from multiple options.
- **Checkbox & Switch** – Used for enabling/disabling settings or selecting multiple options.

3. Display Widgets

These widgets are used to display text, images, and content.

- **Text** – Displays static or dynamic text content.
- **Image** – Loads images from assets, the network, or memory.
- **Card** – A material design component used to display content in an organized manner.

4. Interactive Widgets

These widgets handle user interactions like button clicks and gestures.

- **ElevatedButton** – A raised button used to perform actions on tap.
- **IconButton** – A button with an icon instead of text.
- **GestureDetector** – Detects gestures such as taps, swipes, and long presses.

5. Lists & Scrolling Widgets

These widgets help manage scrollable content.

- **ListView** – Displays a scrollable list of items.
- **GridView** – Creates a grid layout, ideal for galleries and product listings.
- **SingleChildScrollView** – Allows scrolling for a single child widget when content overflows.

6. Navigation Widgets

These widgets enable seamless navigation between different screens in the app.

- **Navigator** – Manages screen transitions using push and pop methods.
- **BottomNavigationBar** – A bottom menu that allows switching between different sections of the app.

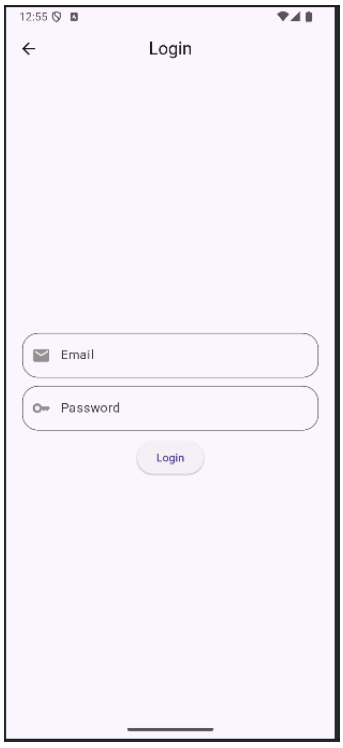
Implementation:

Login Page:

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

class SignupPage extends StatefulWidget {
  @override
  _SignupPageState createState() => _SignupPageState();
}

class _SignupPageState extends State<SignupPage> {
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();
  final FirebaseAuth _auth = FirebaseAuth.instance;

  void signUp() async {
    try {
      await _auth.createUserWithEmailAndPassword(
        email: emailController.text,
        password: passwordController.text,
      );
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Signup Successful!")),
      );
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Error: ${e.toString()}")),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(" Sign Up Page",
          style: TextStyle(
            fontWeight: FontWeight.bold
```

```
  ),),
  centerTitle: true,
),
body: Center(
  child: Padding(
    padding: EdgeInsets.all(16),
    child: Column(
      mainAxisSize: MainAxisSize.min,
      children: [
        TextField(

          controller: emailController,
          decoration: InputDecoration(
            prefixIcon: Icon(Icons.person,color: Colors.grey,),
            labelText: "Enter Your Username",
            border: OutlineInputBorder(
              borderRadius: BorderRadius.circular(20),
            ),
          ),
        ),
        SizedBox(height: 16),
        TextField(
          controller: emailController,
          decoration: InputDecoration(
            prefixIcon: Icon(Icons.mail,color: Colors.grey,),
            labelText: "Enter Your Email",
            border: OutlineInputBorder(
              borderRadius: BorderRadius.circular(20),
            ),
          ),
        ),
        SizedBox(height: 16),
        TextField(
          controller: passwordController,
          obscureText: true,
          decoration: InputDecoration(
            prefixIcon: Icon(Icons.key,color: Colors.grey,),
            labelText: "Enter Your Password",
            border: OutlineInputBorder(
```

```
                borderRadius: BorderRadius.circular(20),
              ),
            ),
          ),
          SizedBox(height: 16),
          ElevatedButton(
            onPressed: signUp,
            child: Text("Sign Up"),
          ),
        ],
      ),
    ),
  ),
 );
 }
}
```



Signup Page:

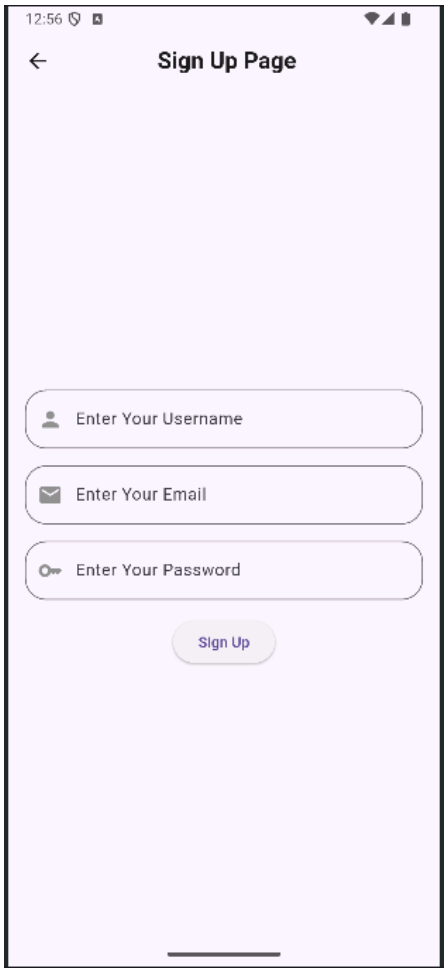
```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_auth/firebase_auth.dart';
```

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}

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

class LoginPage extends StatefulWidget {
  @override
  _LoginPageState createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();
  String errorMessage = '';

  Future<void> loginUser() async {
    try {
      await _auth.signInWithEmailAndPassword(
        email: emailController.text.trim(),
        password: passwordController.text.trim(),
      );
      // Navigate to home page after successful login
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => HomePage()),
      );
    } catch (error) {
```

```
    setState(() {
      errorMessage = error.toString();
    });
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text("Login"),
      centerTitle: true,
    ),
    body: Padding(
      padding: EdgeInsets.all(20.0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          TextField(
            controller: emailController,
            decoration: InputDecoration(
              prefixIcon: Icon(Icons.mail,color: Colors.grey,),
              labelText: "Email",
              border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(20)
              ),
            ),
          ),
          SizedBox(height: 10),
          TextField(
            controller: passwordController,
            obscureText: true,
            decoration: InputDecoration(
              labelText: "Password",
              border: OutlineInputBorder(
                borderRadius: BorderRadius.circular(20)
              ),
              prefixIcon: Icon(Icons.key,color: Colors.grey,)
            ),
```

```
            ),
            SizedBox(height: 10),
            ElevatedButton(
              onPressed: loginUser,
              child: Text("Login"),
            ),
            if (errorMessage.isNotEmpty)
              Text(
                errorMessage,
                style: TextStyle(color: Colors.red),
              ),
          ],
        ),
      ),
    );
  }
}

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

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

    return Scaffold(
      appBar: AppBar(
        title: Text("Home"),
        actions: [
          IconButton(
            icon: Icon(Icons.logout),
            onPressed: () async {
              await _auth.signOut();
              Navigator.pushReplacement(
                context,
                MaterialPageRoute(builder: (context) => LoginPage()),
              );
            },
          ),
        ],
```

```
      ),
      body: Center(
        child: Text(
          "Welcome, ${user?.email ?? 'User'}",
          style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
        ),
      ),
    );
  }
}
```



Conclusion

This app demonstrates Firebase authentication integration in Flutter with a well-structured UI. Using Flutter's Material Design widgets ensures a **responsive, smooth, and visually appealing** user experience.

Aarya Gaikwad
D15B1

MPL Practical No. 3

Aim: To integrate icon images and custom fonts in a Flutter application to enhance the UI and user experience.

Theory:

- Flutter provides various ways to include media and typography in an application.
- Icons Flutter has built in icons from the material library and supports custom icons via ImageIcon or external packages like font awesome Flutter.
- Images can be added from assets (AssetsImage) network sources (NetworkImage) or memory (MemoryImage)
- Fonts: Custom fonts can be added by defining them in the pubspace.yml file and using the TextStyle widgets for styling text.

Using these elements improves UI aesthetics and usability while maintaining app performance.

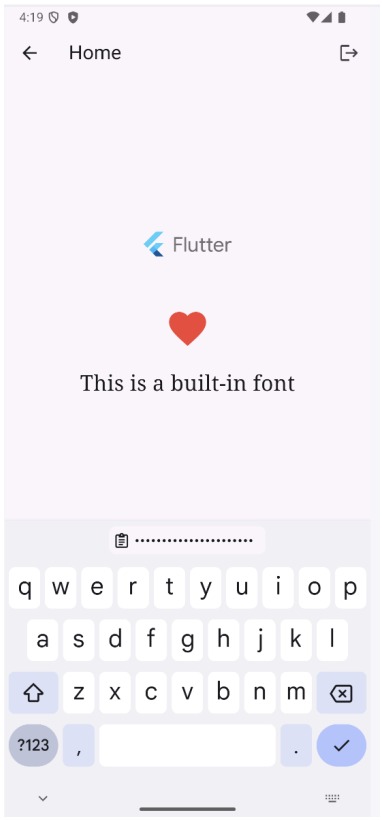
Conclusion

Adding icons, images and fonts in Flutter enhances the apps visual appeal and branding. However, reuses like incorrect and paths, missing dependencies in pubspec.yml or improper font declaration may occur. These can be resolved by ensuring correct file paths, running Flutter pub get after changes updates and verifying image formats and font compatibility. Debugging tools like Flutter doctor and logs help in identifying and fixing and related issues efficiently.

MPL experiment 3 SS

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

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

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

   return Scaffold(
     appBar: AppBar(
       title: Text("Home"),
       actions: [
         IconButton(
           icon: Icon(Icons.logout),
           onPressed: () async {
             await _auth.signOut();
             Navigator.pushReplacement(
               context,
               MaterialPageRoute(builder: (context) => LoginPage()),
             );
           },
         ),
       ],
     ),
     body: Center(
       child: Column(
         mainAxisAlignment: MainAxisAlignment.center,
         children: [
           // Displaying an image from assets
           Image.network(

'https://upload.wikimedia.org/wikipedia/commons/1/17/Google-flutter-logo.png',
             width: 100,
             height: 100,
           ),
           SizedBox(height: 20),
           // Displaying an icon from material icons
           Icon(Icons.favorite, color: Colors.red, size: 50),
           SizedBox(height: 20),
           // Displaying custom font text
           Text(
             'This is a built-in font',
             style: TextStyle(fontFamily: 'Georgia', fontSize: 24),
           )
```

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


4:19
Home
Flutter
This is a built-in font
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123
,
.

Utarga Gandhi
D15B/14

MPL Experiment No. 4

Aim: To integrate login and signup form in Flutter using the form widget enabling user authentication with input validation.

Theory:
Flutter forms, the form widget is used to manage input fields efficiently with validation.
- Text form field captures user input for email and password fields.
- Validation: Ensures required fields and filled correctly using the validator function.
- Controllers: TextEditing controller is used to handle input data Buttons & Navigations. Elevated Button triggers login/signup and navigator handles screen transitions
- Authentication: Can be integrated with firebase or an API to store and verify user credentials

Conclusions:
In this project is successfully implemented an interface login form using flutter form widget with input validation. Initially faced issues with forms validation as it allowed submission, even with empty fields which I resolved using the validator function.
Managing user input access multiple fields was also challenging but implementing textediting controller helped handle data effectively.
Overall the project provided in structured and user friendly authentication system.

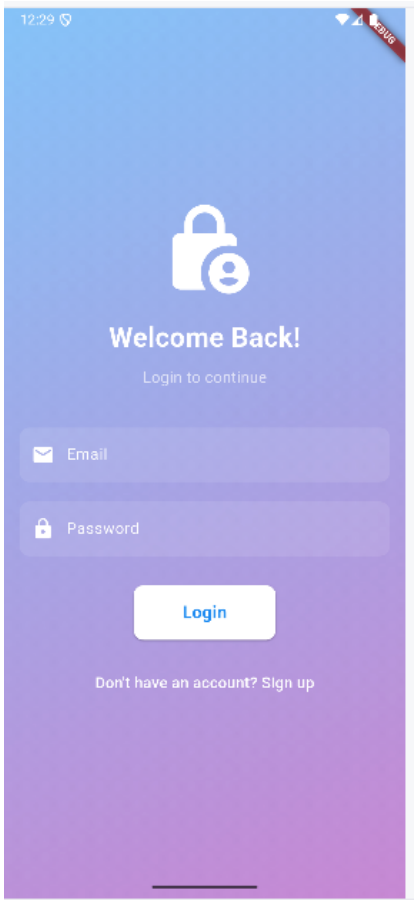
MPL Experiment 4

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

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

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

class _LoginPageState extends State<LoginPage> {
 final FirebaseAuth _auth = FirebaseAuth.instance;
 final TextEditingController emailController = TextEditingController();
 final TextEditingController passwordController = TextEditingController();
 bool isLoading = false;

 Future<void> _login() async {
   final email = emailController.text.trim();
   final password = passwordController.text.trim();

   if (email.isEmpty || password.isEmpty) {
     _showSnackbar('Please enter both email and password');
     return;
   }

   setState(() => isLoading = true);

   try {
     await _auth.signInWithEmailAndPassword(email: email, password: password);
     _showSnackbar('Login Successful');

     // Navigate to HomePage after successful login
     Navigator.pushReplacement(
       context,
       MaterialPageRoute(builder: (context) => const HomePage()),
     );
   } on FirebaseAuthException catch (e) {
     _handleFirebaseError(e);
   } finally {
     setState(() => isLoading = false);
   }
 }

 void _handleFirebaseError(FirebaseAuthException e) {
```

```
    String message = 'Login failed';
    if (e.code == 'user-not-found') {
      message = 'No account found for this email';
    } else if (e.code == 'wrong-password') {
      message = 'Incorrect password';
    } else if (e.code == 'invalid-email') {
      message = 'Invalid email format';
    }
    _showSnackbar(message);
  }

  void _showSnackbar(String message) {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content:
Text(message)));
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SingleChildScrollView(
        child: Container(
          padding: const EdgeInsets.all(16.0),
          height: MediaQuery.of(context).size.height,
          decoration: BoxDecoration(
            gradient: LinearGradient(
              colors: [Colors.blue.shade200, Colors.purple.shade200],
              begin: Alignment.topLeft,
              end: Alignment.bottomRight,
            ),
          ),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              // App Logo or Icon
              const Icon(
                Icons.lock_person,
                size: 100,
                color: Colors.white,
              ),
              const SizedBox(height: 20),
              const Text(
                'Welcome Back!',
                style: TextStyle(
                  fontSize: 28,
                  fontWeight: FontWeight.bold,
                  color: Colors.white,
                ),
              ),
              const SizedBox(height: 10),
```

```
const Text(
  'Login to continue',
  style: TextStyle(
    fontSize: 16,
    color: Colors.white70,
  ),
),
const SizedBox(height: 40),
// Email Field
TextField(
  controller: emailController,
  decoration: InputDecoration(
    labelText: 'Email',
    labelStyle: const TextStyle(color: Colors.white),
    filled: true,
    fillColor: Colors.white.withOpacity(0.1),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
      borderSide: BorderSide.none,
    ),
    prefixIcon: const Icon(Icons.email, color: Colors.white),
  ),
  style: const TextStyle(color: Colors.white),
  keyboardType: TextInputType.emailAddress,
),
const SizedBox(height: 20),
// Password Field
TextField(
  controller: passwordController,
  obscureText: true,
  decoration: InputDecoration(
    labelText: 'Password',
    labelStyle: const TextStyle(color: Colors.white),
    filled: true,
    fillColor: Colors.white.withOpacity(0.1),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
      borderSide: BorderSide.none,
    ),
    prefixIcon: const Icon(Icons.lock, color: Colors.white),
  ),
  style: const TextStyle(color: Colors.white),
),
const SizedBox(height: 30),
// Login Button
isLoading
    ? const CircularProgressIndicator(color: Colors.white)
    : ElevatedButton(
  onPressed: _login,
```

```
    style: ElevatedButton.styleFrom(
      backgroundColor: Colors.white,
      padding: const EdgeInsets.symmetric(
          horizontal: 50, vertical: 15),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(10),
      ),
    ),
    child: const Text(
      'Login',
      style: TextStyle(
        fontSize: 18,
        color: Colors.blue,
        fontWeight: FontWeight.bold,
      ),
    ),
  ),
),
const SizedBox(height: 20),
// Signup Link
white,
      fontSize: 16,
```



Page No.
Date

MP2 Experiment 5

:- To implement navigation, routing and gesture controls in a flutter application designed for allowing users to seamlessly navigate between different and interact with UI elements through touch gestures.

- Navigation and routing in flutter are essential for creating a smooth user experience, enabling users to move different app screens. flutter uses the navigator widget to manage routes, with methods like push() and pop(). Named routes provide a structured approach to screen transitions, improving code maintainability.
Gestures enhance interactivity by allowing users to perform actions like swiping, tapping and long pressing. The GestureDetector widget captures user input and triggers specific responses, such as moving a plane.

- The experiment successfully integrated navigation, routing and gestures controls, allowing users to move between plant care sections effortlessly and interact with intuitive touch gesture. A key challenge encountered was ensuring smooth transitions without unexpected screen behaviour, particularly when handling back navigation and gesture conflicts.

MPL Experiment 5

1. Navigation

Login Page

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

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

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

class _LoginPageState extends State<LoginPage> {
 final FirebaseAuth _auth = FirebaseAuth.instance;
 final TextEditingController emailController =
TextEditingController();
 final TextEditingController passwordController =
TextEditingController();
 bool isLoading = false;

 Future<void> _login() async {
   final email = emailController.text.trim();
   final password = passwordController.text.trim();

   if (email.isEmpty || password.isEmpty) {
     _showSnackbar('Please enter both email and
password');
     return;
   }

   setState(() => isLoading = true);

   try {
     await _auth.signInWithEmailAndPassword(email:
email, password: password);
     _showSnackbar('Login Successful');

     // Navigate to HomePage after successful login
     Navigator.pushReplacement(
       context,
       MaterialPageRoute(builder: (context) => const
HomePage()),
     );
   } on FirebaseAuthException catch (e) {
     _handleFirebaseError(e);
   } finally {
     setState(() => isLoading = false);
   }
```

Signup page

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

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

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

class _SignupPageState extends State<SignupPage>
{
 final FirebaseAuth _auth =
FirebaseAuth.instance;
 final TextEditingController emailController =
TextEditingController();
 final TextEditingController passwordController
= TextEditingController();
 final TextEditingController
confirmPasswordController =
TextEditingController();
 bool isLoading = false;

 Future<void> _signup() async {
   final email = emailController.text.trim();
   final password =
passwordController.text.trim();
   final confirmPassword =
confirmPasswordController.text.trim();

   if (email.isEmpty || password.isEmpty ||
confirmPassword.isEmpty) {
     _showSnackbar('Please fill all fields');
     return;
   }

   if (password != confirmPassword) {
     _showSnackbar('Passwords do not match');
     return;
   }

   setState(() => isLoading = true);

   try {
     await
```

```
 }

 void _handleFirebaseError(FirebaseAuthException e)
{
   String message = 'Login failed';
   if (e.code == 'user-not-found') {
     message = 'No account found for this email';
   } else if (e.code == 'wrong-password') {
     message = 'Incorrect password';
   } else if (e.code == 'invalid-email') {
     message = 'Invalid email format';
   }
   _showSnackbar(message);
 }

 void _showSnackbar(String message) {

ScaffoldMessenger.of(context).showSnackBar(SnackBar(
content: Text(message)));
 }

 @override
 Widget build(BuildContext context) {
   return Scaffold(
     body: SingleChildScrollView(
       child: Container(
         padding: const EdgeInsets.all(16.0),
         height: MediaQuery.of(context).size.height,
         decoration: BoxDecoration(
           gradient: LinearGradient(
             colors: [Colors.blue.shade200,
Colors.purple.shade200],
             begin: Alignment.topLeft,
             end: Alignment.bottomRight,
           ),
         ),
         child: Column(
           mainAxisAlignment:
MainAxisAlignment.center,
           children: [
             // App Logo or Icon
             const Icon(
               Icons.lock_person,
               size: 100,
               color: Colors.white,
             ),
             const SizedBox(height: 20),
             const Text(
               'Welcome Back!',
               style: TextStyle(
                 fontSize: 28,
                 fontWeight: FontWeight.bold,
                 color: Colors.white,
               ),
             ),
             const SizedBox(height: 10),
```

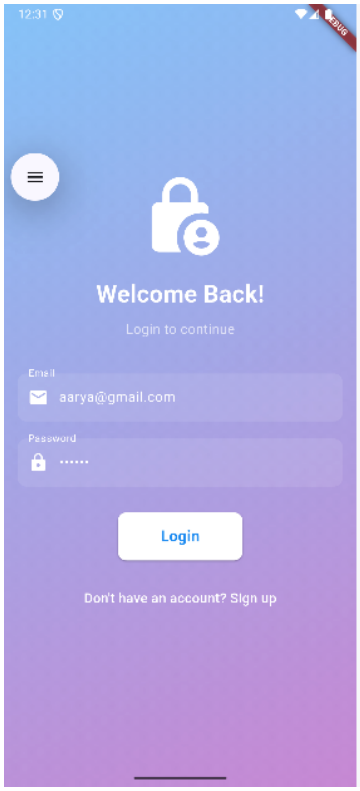
```
_auth.createUserWithEmailAndPassword(email:
email, password: password);
     _showSnackbar('Signup Successful');

     // Navigate to LoginPage after successful
signup
     Navigator.pushReplacement(
       context,
       MaterialPageRoute(builder: (context) =>
const LoginPage()),
     );
   } on FirebaseAuthException catch (e) {
     _handleFirebaseError(e);
   } finally {
     setState(() => isLoading = false);
   }
 }

 void _handleFirebaseError(FirebaseAuthException
e) {
   String message = 'Signup failed';
   if (e.code == 'email-already-in-use') {
     message = 'Email is already in use';
   } else if (e.code == 'weak-password') {
     message = 'Password is too weak';
   } else if (e.code == 'invalid-email') {
     message = 'Invalid email format';
   }
   _showSnackbar(message);
 }

 void _showSnackbar(String message) {

ScaffoldMessenger.of(context).showSnackBar(Snack
Bar(content: Text(message)));
 }

 @override
 Widget build(BuildContext context) {
   return Scaffold(
     body: SingleChildScrollView(
       child: Container(
         padding: const EdgeInsets.all(16.0),
         height:
MediaQuery.of(context).size.height,
         decoration: BoxDecoration(
           gradient: LinearGradient(
             colors: [Colors.blue.shade200,
Colors.purple.shade200],
             begin: Alignment.topLeft,
             end: Alignment.bottomRight,
           ),
         ),
         child: Column(
           mainAxisAlignment:
MainAxisAlignment.center,
```

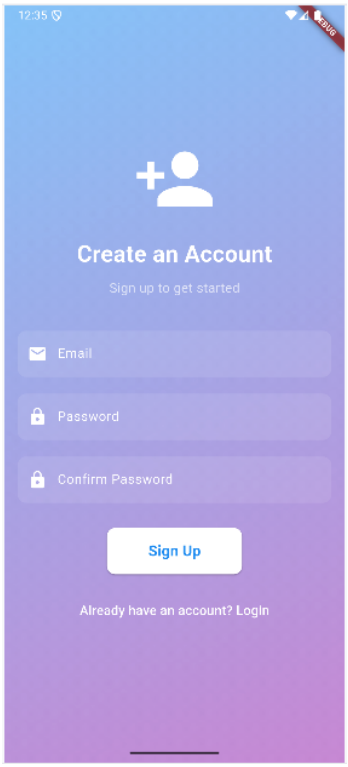
```
            const Text(
              'Login to continue',
              style: TextStyle(
                fontSize: 16,
                color: Colors.white70,
              ),
            ),
            const SizedBox(height: 40),
            // Email Field
            TextField(
              controller: emailController,
              decoration: InputDecoration(
                labelText: 'Email',
                labelStyle: const TextStyle(color:
Colors.white),
                filled: true,
                fillColor:
Colors.white.withOpacity(0.1),
                border: OutlineInputBorder(
                  borderRadius:
BorderRadius.circular(10),
                  borderSide: BorderSide.none,
                ),
                prefixIcon: const Icon(Icons.email,
color: Colors.white),
              ),
              style: const TextStyle(color:
Colors.white),
              keyboardType:
TextInputType.emailAddress,
            ),
            const SizedBox(height: 20),
            // Password Field
            TextField(
              controller: passwordController,
              obscureText: true,
              decoration: InputDecoration(
                labelText: 'Password',
                labelStyle: const TextStyle(color:
Colors.white),
                filled: true,
                fillColor:
Colors.white.withOpacity(0.1),
                border: OutlineInputBorder(
                  borderRadius:
BorderRadius.circular(10),
                  borderSide: BorderSide.none,
                ),
                prefixIcon: const Icon(Icons.lock,
color: Colors.white),
              ),
              style: const TextStyle(color:
Colors.white),
            ),
            const SizedBox(height: 30),
            // Login Button
```

```
          children: [
            // App Logo or Icon
            const Icon(
              Icons.person_add,
              size: 100,
              color: Colors.white,
            ),
            const SizedBox(height: 20),
            const Text(
              'Create an Account',
              style: TextStyle(
                fontSize: 28,
                fontWeight: FontWeight.bold,
                color: Colors.white,
              ),
            ),
            const SizedBox(height: 10),
            const Text(
              'Sign up to get started',
              style: TextStyle(
                fontSize: 16,
                color: Colors.white70,
              ),
            ),
            const SizedBox(height: 40),
            // Email Field
            TextField(
              controller: emailController,
              decoration: InputDecoration(
                labelText: 'Email',
                labelStyle: const
TextStyle(color: Colors.white),
                filled: true,
                fillColor:
Colors.white.withOpacity(0.1),
                border: OutlineInputBorder(
                  borderRadius:
BorderRadius.circular(10),
                  borderSide: BorderSide.none,
                ),
                prefixIcon: const
Icon(Icons.email, color: Colors.white),
              ),
              style: const TextStyle(color:
Colors.white),
              keyboardType:
TextInputType.emailAddress,
            ),
            const SizedBox(height: 20),
            // Password Field
            TextField(
              controller: passwordController,
              obscureText: true,
              decoration: InputDecoration(
                labelText: 'Password',
                labelStyle: const
```

```
              isLoading
                  ? const
CircularProgressIndicator(color: Colors.white)
                  : ElevatedButton(
                onPressed: _login,
                style: ElevatedButton.styleFrom(
                  backgroundColor: Colors.white,
                  padding: const
EdgeInsets.symmetric(
                      horizontal: 50, vertical: 15),
                  shape: RoundedRectangleBorder(
                    borderRadius:
BorderRadius.circular(10),
                  ),
                ),
                child: const Text(
                  'Login',
                  style: TextStyle(
                    fontSize: 18,
                    color: Colors.blue,
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ),
              const SizedBox(height: 20),
              // Signup Link
              TextButton(
                onPressed: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(builder:
(context) => const SignupPage()),
                  );
                },
                child: const Text(
                  'Don\'t have an account? Sign up',
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: 16,
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

```
TextStyle(color: Colors.white),
                  filled: true,
                  fillColor:
Colors.white.withOpacity(0.1),
                  border: OutlineInputBorder(
                    borderRadius:
BorderRadius.circular(10),
                    borderSide: BorderSide.none,
                  ),
                  prefixIcon: const
Icon(Icons.lock, color: Colors.white),
                ),
                style: const TextStyle(color:
Colors.white),
              ),
              const SizedBox(height: 20),
              // Confirm Password Field
              TextField(
                controller:
confirmPasswordController,
                obscureText: true,
                decoration: InputDecoration(
                  labelText: 'Confirm Password',
                  labelStyle: const
TextStyle(color: Colors.white),
                  filled: true,
                  fillColor:
Colors.white.withOpacity(0.1),
                  border: OutlineInputBorder(
                    borderRadius:
BorderRadius.circular(10),
                    borderSide: BorderSide.none,
                  ),
                  prefixIcon: const
Icon(Icons.lock, color: Colors.white),
                ),
                style: const TextStyle(color:
Colors.white),
              ),
              const SizedBox(height: 30),
              // Signup Button
              isLoading
                  ? const
CircularProgressIndicator(color: Colors.white)
                  : ElevatedButton(
                onPressed: _signup,
                style: ElevatedButton.styleFrom(
                  backgroundColor: Colors.white,
                  padding: const
EdgeInsets.symmetric(
                      horizontal: 50, vertical:
15),
                  shape: RoundedRectangleBorder(
                    borderRadius:
BorderRadius.circular(10),
                  )
```





2. Gestures

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

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

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

 @override
 State<HomePage> createState() => _MyAppState();
}

class _MyAppState extends State<HomePage> {
 late GoogleMapController mapController;

 final LatLng _center = const LatLng(45.521563, -122.677433);

 void _onMapCreated(GoogleMapController controller) {
   mapController = controller;
```

```
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        useMaterial3: true,
        colorSchemeSeed: Colors.green[700],
      ),
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Maps Sample App'),
          elevation: 2,
        ),
        body: GoogleMap(
          onMapCreated: _onMapCreated,
          initialCameraPosition: CameraPosition(
            target: _center,
            zoom: 11.0,
          ),
        ),
      ),
    );
  }
}
```

NAME: Aarya Chandhi STD.: ~~BE~~ IT DIV.: D15B

DATE :
PAGE :

MPL Practical No. 6 (A)

Aim: To set up Firebase with Flutter for iOS & Android apps.

Theory:
Firebase is a great backend solution for anyone that wants to use authentication, databases, cloud functions, ads, and countless other features within an app

Prereq/visites
To setup firebase with Flutter
- Google account for Firebase
- xcode (for iOS development)
- Flutter ~~is~~ installed
- Android studio & Visual studio

Creating a New Flutter project.
i) Run: flutter create flutterfirebaseexample cd flutter-firebase-example
ii) this creates a demo Flutter app that increments a counter when a button is clicked
iii)

Creating a Firebase project.
i) log into Firebase and click Create a new project.
ii) ~~Enable~~ Once created, go to the Firebase dashboard.

Adding Android support
i) select the android logo in Firebase

NAME: STD.: DIV.: DATE: PAGE:

ii) Use a unique android package name, e.g com.example.
6seeexample
iii) Download google-services.json and move it to android

Adding the Firebase SDK
i) Update android/build.gradle.
ii) Update android/app/build.gradle
iii) Run the app; Firebase should now be linked.

Adding iOS support.
i) In Firebase, select iOS & register with a Bundle ID the one in Xcode
ii) Open ios/Runner/Runner.xcodeproj & update the Bundle Identity under general.
iii) Download Google-service-info.plist.

Conclusion
In this lab I connected my app with Firebase, implemented firestore services to handle CRUD operation and Firebase authentication along with signin with google

MPL Experiment 6

Google-services.json


```
{
 "project_info": {
   "project_number": "663509823079",
   "project_id": "comaplaint1",
   "storage_bucket": "comaplaint1.firebasestorage.app"
 },
 "client": [
   {
     "client_info": {
       "mobilesdk_app_id": "1:663509823079:android:f919160ed188bc2ee42ec2",
       "android_client_info": {
         "package_name": "com.example.new_complaint"
       }
     },
     "oauth_client": [],
     "api_key": [
       {
         "current_key": "AIzaSyALbN6aQftIodwXpZmdIRzO0JDSUhD4ojk"
       }
     ],
     "services": {
       "appinvite_service": {
         "other_platform_oauth_client": []
       }
     }
   }
 ],
 "configuration_version": "1"
}
```


Login Page

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

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

 @override
 _LoginPageState createState() => _LoginPageState();
}
```

```dart
class _LoginPageState extends State<LoginPage> {
 final FirebaseAuth _auth = FirebaseAuth.instance;
 final TextEditingController emailController = TextEditingController();
 final TextEditingController passwordController = TextEditingController();
 bool isLoading = false;

 Future<void> _login() async {
   final email = emailController.text.trim();
   final password = passwordController.text.trim();

   if (email.isEmpty || password.isEmpty) {
     _showSnackbar('Please enter both email and password');
     return;
   }

   setState(() => isLoading = true);

   try {
     await _auth.signInWithEmailAndPassword(email: email, password: password);
     _showSnackbar('Login Successful');

     // Navigate to HomePage after successful login
     Navigator.pushReplacement(
       context,
       MaterialPageRoute(builder: (context) => const HomePage()),
     );
   } on FirebaseAuthException catch (e) {
     _handleFirebaseError(e);
   } finally {
     setState(() => isLoading = false);
   }
 }

 void _handleFirebaseError(FirebaseAuthException e) {
   String message = 'Login failed';
   if (e.code == 'user-not-found') {
     message = 'No account found for this email';
   } else if (e.code == 'wrong-password') {
     message = 'Incorrect password';
   } else if (e.code == 'invalid-email') {
     message = 'Invalid email format';
   }
   _showSnackbar(message);
 }

 void _showSnackbar(String message) {
   ScaffoldMessenger.of(context).showSnackBar(SnackBar(content:
Text(message)));
 }
```

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: SingleChildScrollView(
      child: Container(
        padding: const EdgeInsets.all(16.0),
        height: MediaQuery.of(context).size.height,
        decoration: BoxDecoration(
          gradient: LinearGradient(
            colors: [Colors.blue.shade200, Colors.purple.shade200],
            begin: Alignment.topLeft,
            end: Alignment.bottomRight,
          ),
        ),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            // App Logo or Icon
            const Icon(
              Icons.lock_person,
              size: 100,
              color: Colors.white,
            ),
            const SizedBox(height: 20),
            const Text(
              'Welcome Back!',
              style: TextStyle(
                fontSize: 28,
                fontWeight: FontWeight.bold,
                color: Colors.white,
              ),
            ),
            const SizedBox(height: 10),
            const Text(
              'Login to continue',
              style: TextStyle(
                fontSize: 16,
                color: Colors.white70,
              ),
            ),
            const SizedBox(height: 40),
            // Email Field
            TextField(
              controller: emailController,
              decoration: InputDecoration(
                labelText: 'Email',
                labelStyle: const TextStyle(color: Colors.white),
                filled: true,
```

```
    fillColor: Colors.white.withOpacity(0.1),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
      borderSide: BorderSide.none,
    ),
    prefixIcon: const Icon(Icons.email, color: Colors.white),
  ),
  style: const TextStyle(color: Colors.white),
  keyboardType: TextInputType.emailAddress,
),
const SizedBox(height: 20),
// Password Field
TextField(
  controller: passwordController,
  obscureText: true,
  decoration: InputDecoration(
    labelText: 'Password',
    labelStyle: const TextStyle(color: Colors.white),
    filled: true,
    fillColor: Colors.white.withOpacity(0.1),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
      borderSide: BorderSide.none,
    ),
    prefixIcon: const Icon(Icons.lock, color: Colors.white),
  ),
  style: const TextStyle(color: Colors.white),
),
const SizedBox(height: 30),
// Login Button
isLoading
    ? const CircularProgressIndicator(color: Colors.white)
    : ElevatedButton(
  onPressed: _login,
  style: ElevatedButton.styleFrom(
    backgroundColor: Colors.white,
    padding: const EdgeInsets.symmetric(
        horizontal: 50, vertical: 15),
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(10),
    ),
  ),
  child: const Text(
    'Login',
    style: TextStyle(
      fontSize: 18,
      color: Colors.blue,
      fontWeight: FontWeight.bold,
    ),
```

```
              ),
            ),
            const SizedBox(height: 20),
            // Signup Link
            TextButton(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => const SignupPage()),
                );
              },
              child: const Text(
                'Don\'t have an account? Sign up',
                style: TextStyle(
                  color: Colors.white,
                  fontSize: 16,
                ),
              ),
            ),
          ),
        ],
      ),
    ),
  ),
 );
 }
}
```

Signup Page:

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

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

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

class _SignupPageState extends State<SignupPage> {
 final FirebaseAuth _auth = FirebaseAuth.instance;
 final TextEditingController emailController = TextEditingController();
 final TextEditingController passwordController = TextEditingController();
 final TextEditingController confirmPasswordController =
TextEditingController();
 bool isLoading = false;

 Future<void> _signup() async {
   final email = emailController.text.trim();
```

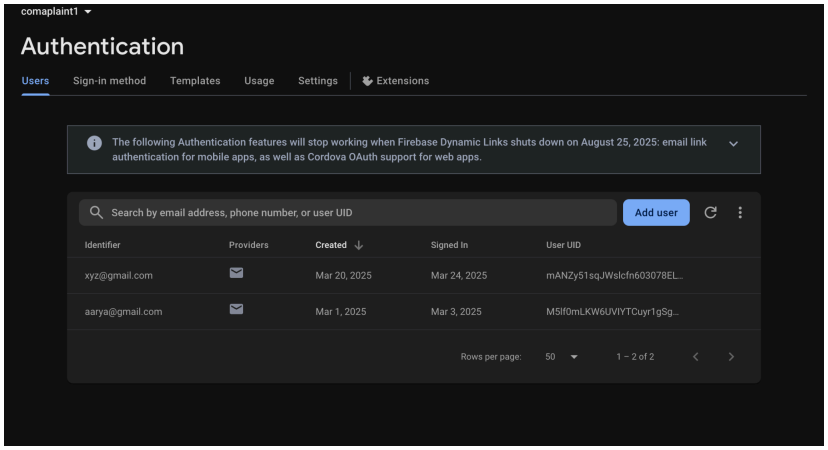
```
    final password = passwordController.text.trim();
    final confirmPassword = confirmPasswordController.text.trim();

    if (email.isEmpty || password.isEmpty || confirmPassword.isEmpty) {
      _showSnackbar('Please fill all fields');
      return;
    }

    if (password != confirmPassword) {
      _showSnackbar('Passwords do not match');
      return;
    }

    setState(() => isLoading = true);

    try {
      await _auth.createUserWithEmailAndPassword(email: email, password:
password);
      _showSnackbar('Signup Successful');

      // Navigate to LoginPage after successful signup
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => const LoginPage()),
      );
    } on FirebaseAuthException catch (e) {
      _handleFirebaseError(e);
    } finally {
      setState(() => isLoading = false);
    }
  }

  void _handleFirebaseError(FirebaseAuthException e) {
    String message = 'Signup failed';
    if (e.code == 'email-already-in-use') {
      message = 'Email is already in use';
    } else if (e.code == 'weak-password') {
      message = 'Password is too weak';
    } else if (e.code == 'invalid-email') {
      message = 'Invalid email format';
    }
    _showSnackbar(message);
  }

  void _showSnackbar(String message) {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content:
Text(message)));
  }
```

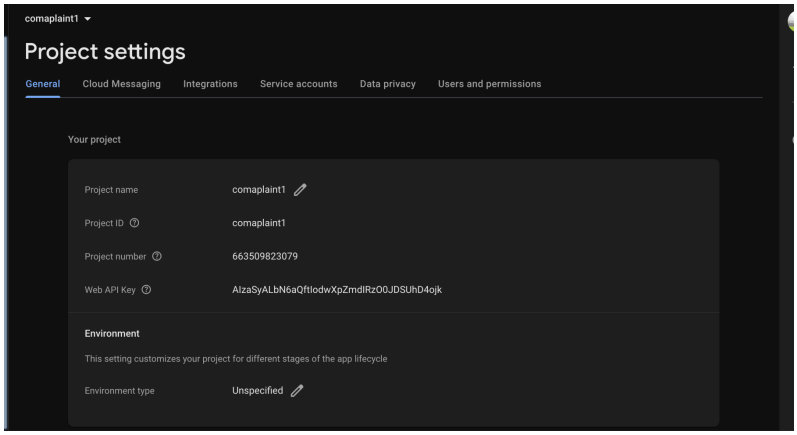
```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: SingleChildScrollView(
      child: Container(
        padding: const EdgeInsets.all(16.0),
        height: MediaQuery.of(context).size.height,
        decoration: BoxDecoration(
          gradient: LinearGradient(
            colors: [Colors.blue.shade200, Colors.purple.shade200],
            begin: Alignment.topLeft,
            end: Alignment.bottomRight,
          ),
        ),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            // App Logo or Icon
            const Icon(
              Icons.person_add,
              size: 100,
              color: Colors.white,
            ),
            const SizedBox(height: 20),
            const Text(
              'Create an Account',
              style: TextStyle(
                fontSize: 28,
                fontWeight: FontWeight.bold,
                color: Colors.white,
              ),
            ),
            const SizedBox(height: 10),
            const Text(
              'Sign up to get started',
              style: TextStyle(
                fontSize: 16,
                color: Colors.white70,
              ),
            ),
            const SizedBox(height: 40),
            // Email Field
            TextField(
              controller: emailController,
              decoration: InputDecoration(
                labelText: 'Email',
                labelStyle: const TextStyle(color: Colors.white),
                filled: true,
                fillColor: Colors.white.withOpacity(0.1),
```

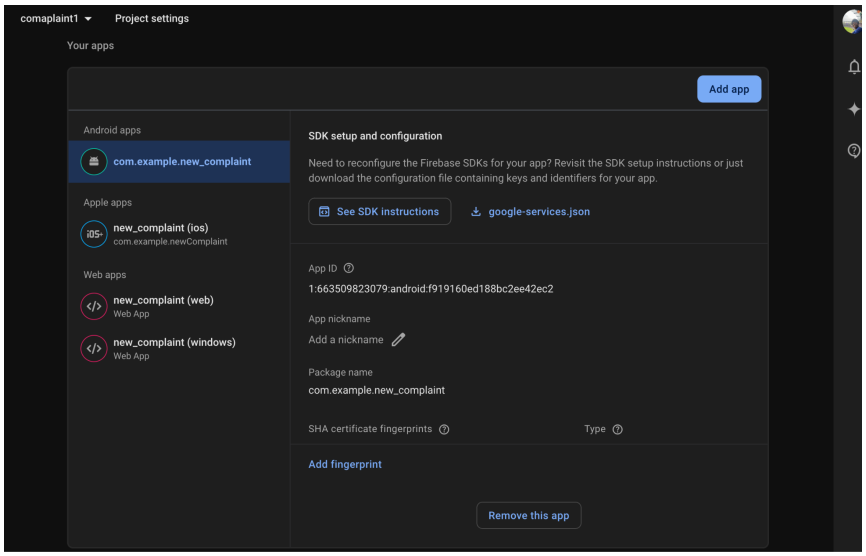
```
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(10),
        borderSide: BorderSide.none,
      ),
      prefixIcon: const Icon(Icons.email, color: Colors.white),
    ),
    style: const TextStyle(color: Colors.white),
    keyboardType: TextInputType.emailAddress,
  ),
  const SizedBox(height: 20),
  // Password Field
  TextField(
    controller: passwordController,
    obscureText: true,
    decoration: InputDecoration(
      labelText: 'Password',
      labelStyle: const TextStyle(color: Colors.white),
      filled: true,
      fillColor: Colors.white.withOpacity(0.1),
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(10),
        borderSide: BorderSide.none,
      ),
      prefixIcon: const Icon(Icons.lock, color: Colors.white),
    ),
    style: const TextStyle(color: Colors.white),
  ),
  const SizedBox(height: 20),
  // Confirm Password Field
  TextField(
    controller: confirmPasswordController,
    obscureText: true,
    decoration: InputDecoration(
      labelText: 'Confirm Password',
      labelStyle: const TextStyle(color: Colors.white),
      filled: true,
      fillColor: Colors.white.withOpacity(0.1),
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(10),
        borderSide: BorderSide.none,
      ),
      prefixIcon: const Icon(Icons.lock, color: Colors.white),
    ),
    style: const TextStyle(color: Colors.white),
  ),
  const SizedBox(height: 30),
  // Signup Button
  isLoading
      ? const CircularProgressIndicator(color: Colors.white)
```

```
                  : ElevatedButton(
                onPressed: _signup,
                style: ElevatedButton.styleFrom(
                  backgroundColor: Colors.white,
                  padding: const EdgeInsets.symmetric(
                      horizontal: 50, vertical: 15),
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(10),
                  ),
                ),
                child: const Text(
                  'Sign Up',
                  style: TextStyle(
                    fontSize: 18,
                    color: Colors.blue,
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ),
              const SizedBox(height: 20),
              // Login Link
              TextButton(
                onPressed: () {
                  Navigator.pushReplacement(
                    context,
                    MaterialPageRoute(builder: (context) => const LoginPage()),
                  );
                },
                child: const Text(
                  'Already have an account? Login',
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: 16,
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Firebase Screenshots:


comaplaint1
Authentication
Users
Sign-in method
Templates
Usage
Settings
Extensions
The following Authentication features will stop working when Firebase Dynamic Links shuts down on August 25, 2025: email link authentication for mobile apps, as well as Cordova OAuth support for web apps.
Search by email address, phone number, or user UID
Add user
Identifier
Providers
Created
Signed In
User UID
xyz@gmail.com
Mar 20, 2025
Mar 24, 2025
mANZy51sqJWslcfn603078EL...
aarya@gmail.com
Mar 1, 2025
Mar 3, 2025
M5lf0mLKW6UVlYTCuyr1gSg...
Rows per page:
50
1 – 2 of 2


comaplaint1
Project settings
General
Cloud Messaging
Integrations
Service accounts
Data privacy
Users and permissions
Your project
Project name
comaplaint1
Project ID
comaplaint1
Project number
663509823079
Web API Key
AIzaSyALbN6aQftIodwXpZmdlRzO0JDSUhD4ojk
Environment
This setting customizes your project for different stages of the app lifecycle
Environment type
Unspecified


comaplaint1
Project settings
Your apps
Add app
Android apps
com.example.new_complaint
Apple apps
new_complaint (ios)
com.example.newComplaint
Web apps
new_complaint (web)
Web App
new_complaint (windows)
Web App
SDK setup and configuration
Need to reconfigure the Firebase SDKs for your app? Revisit the SDK setup instructions or just download the configuration file containing keys and identifiers for your app.
See SDK instructions
google-services.json
App ID
1:663509823079:android:f919160ed188bc2ee42ec2
App nickname
Add a nickname
Package name
com.example.new_complaint
SHA certificate fingerprints
Type
Add fingerprint
Remove this app

Roll No. 14

NAME: Aarya Carnelli STD.: DSB DIV.:

DATE :

PAGE :

MPL Experiment 7

Aim: To develop a progressive web app (PWA) that can be installed on a device like a native application and provides offline functionality using a service worker and a web manifest file.

Theory: A progressive web app (PWA) is a modern web application that provides an app like experience on the web. It allows users to install the app on their device, work offline and load content faster. PWA's leverage the following key components.

1. Service Worker

A background script that enables offline functionality by caching resources. It intercepts network requests and serves cached responses when offline.

2. Web App manifest

A JSON file that provides meta data about the PWA, such as the app name, theme color, start url and icons. It makes the web app installed on device.

3. Caching strategy

The cache first approach used in this PWA means that if resources are available in cache they are served first, otherwise they are fetched from the network.

Conct

DATE :
PAGE :

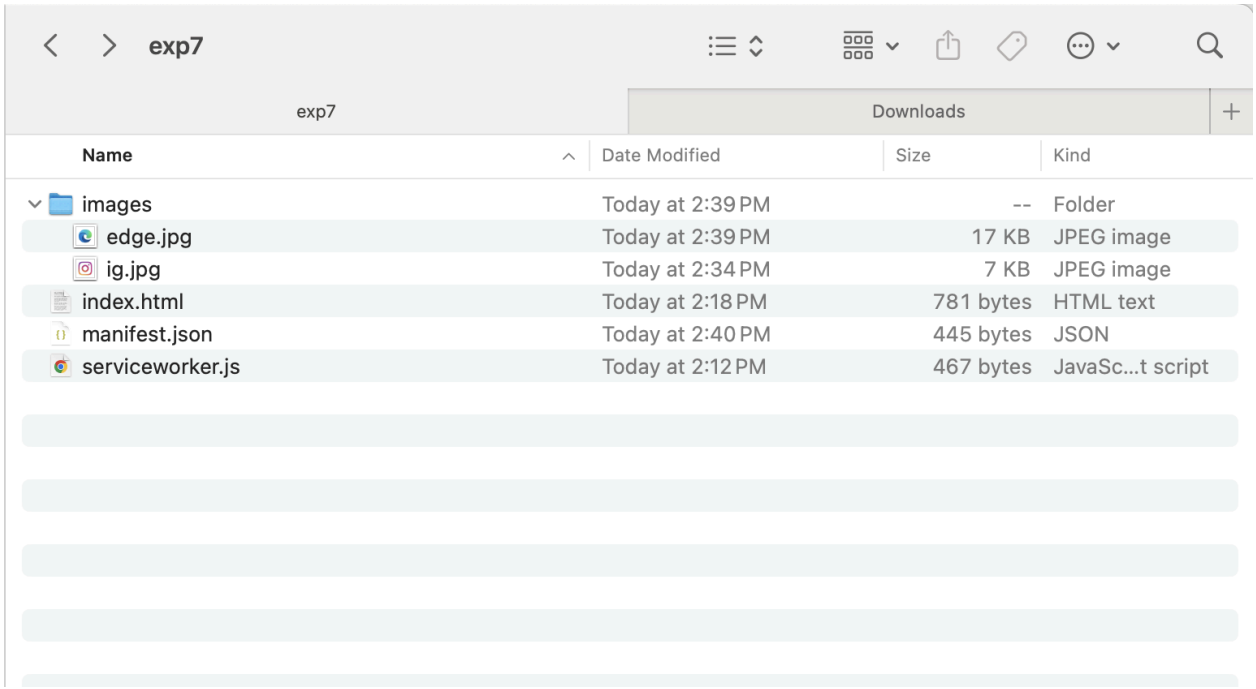
NAME: STD.: DIV.:

Conclusion:

The implementation successfully demonstrates the architecture by enabling installation, offline access, caching. Users can install the app and even once is lost, installed pages will still be accessible. The project highlights the benefits of PWAs improved performance, reliability, and user engagement making them a powerful alternative

MPL Experiment 7

Folder Structure



Index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>PWA Experiment 7</title>

   <!-- Manifest File -->
   <link rel="manifest" href="/manifest.json" />

   <style>
     body {
       font-family: "Arial", sans-serif;
       background: linear-gradient(to right, #4facfe, #00f2fe);
       color: #fff;
       text-align: center;
       margin: 0;
       padding: 0;
       display: flex;
```

```
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100vh;
  }

  h1 {
    font-size: 2.5rem;
    font-weight: bold;
    margin-bottom: 10px;
  }

  h4 {
    font-size: 1.2rem;
    font-weight: normal;
    margin-bottom: 20px;
  }

  .install-btn {
    background-color: #ff4d6d;
    color: white;
    border: none;
    padding: 12px 20px;
    font-size: 1rem;
    border-radius: 25px;
    cursor: pointer;
    transition: 0.3s;
  }

  .install-btn:hover {
    background-color: #ff1e4d;
  }

  .container {
    background: rgba(255, 255, 255, 0.2);
    padding: 20px;
    border-radius: 10px;
    backdrop-filter: blur(10px);
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
    max-width: 500px;
  }
</style>
```

```
    <script>
      window.addEventListener("load", () => {
        registerSW();
      });

      async function registerSW() {
        if ("serviceWorker" in navigator) {
          try {
            await navigator.serviceWorker.register("/serviceworker.js");
            console.log("Service Worker Registered Successfully");
          } catch (error) {
            console.error("Service Worker Registration Failed:", error);
          }
        } else {
          console.warn("Service Worker is not supported in this browser.");
        }
      }
    </script>
  </head>
  <body>
    <div class="container">
      <h1>Hello!!! This is Aarya Gandhi</h1>
      <h4>This is a sample page for the implementation of PWA Experiment 7</h4>
    </div>
  </body>
</html>
```

Manifest.json

```
{
  "name": "MPL experiment 7",
  "short_name": "PWA",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#5900b3",
  "theme_color": "black",
  "scope": "/",
  "description": "This is a PWA tutorial.",
  "icons": [
```

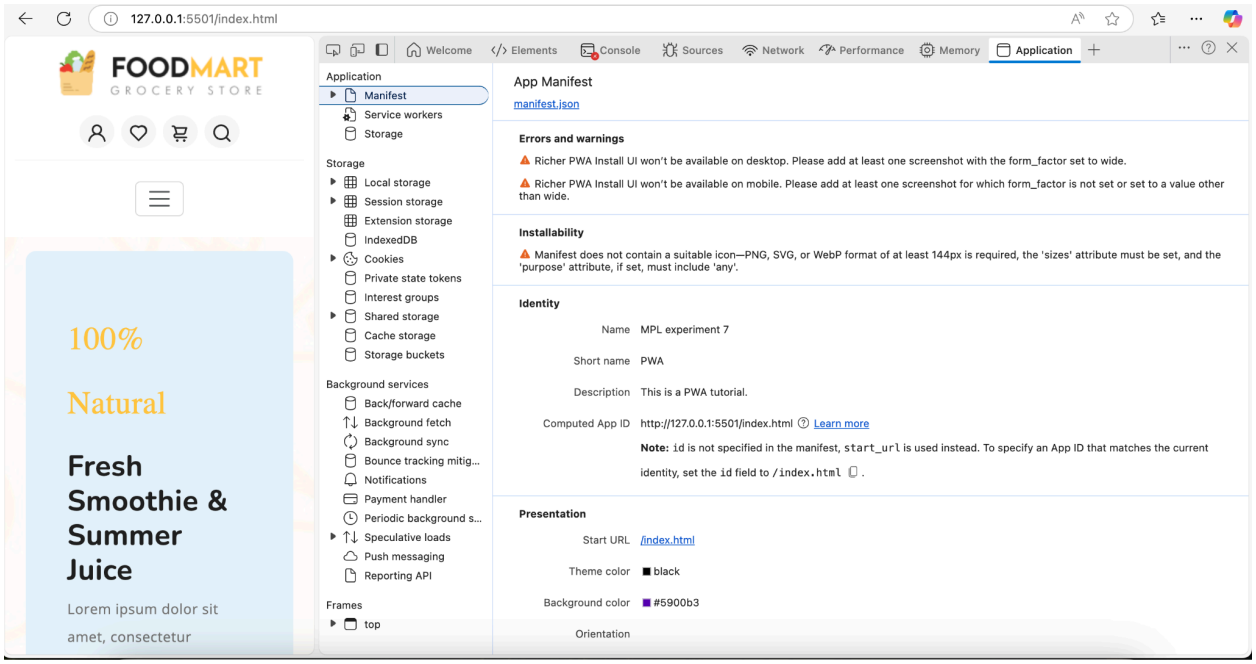
```
    {
      "src": "/images/ig.jpg",
      "sizes": "192x192",
      "type": "image/jpg"
    },
     {
      "src": "/images/edge.jpg",
      "sizes": "512x512",
      "type": "image/jpg"
    }

 ]
}
```

ServiceWorker.js

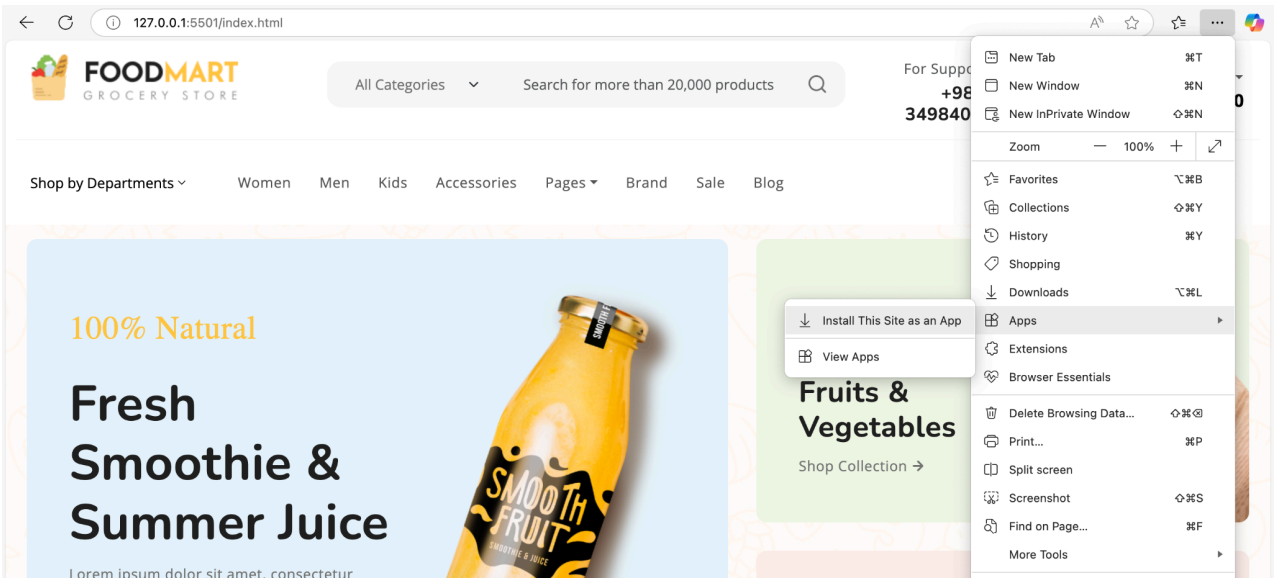
```
var staticCacheName="pwa";

self.addEventListener("Install",function(e){
    e.waitUntil(
             caches.open(staticCacheName).then(function(cache){
                 return cache.addAll(["/"])
             })
    );
});

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

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

});
```

Steps to Implement:

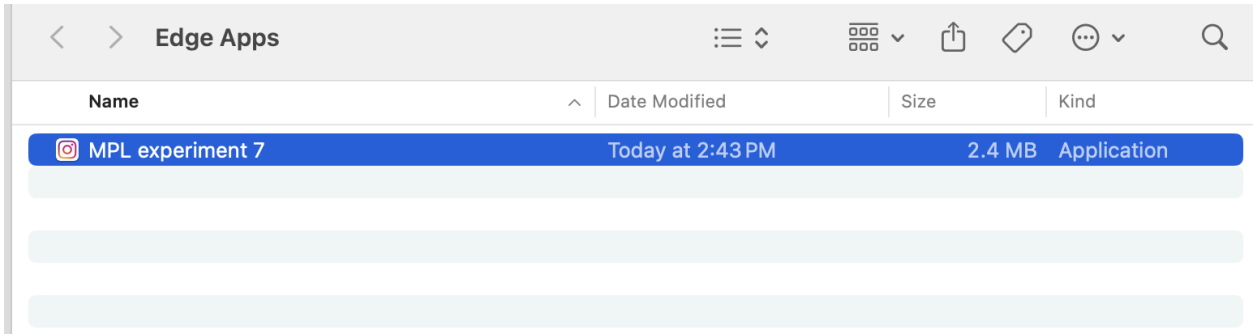
1. **open Developer tools options->g to applications**



2. **Click on the top right corner of browser from the app option install this site as an app**



3. **App icon**


Edge Apps
Name
Date Modified
Size
Kind
MPL experiment 7
Today at 2:43 PM
2.4 MB
Application

NAME: Aanya Gandhi STD.: D15B DIV.: Roll No. 14

DATE :
PAGE :

MPA Experiment 8 -

Aim: To develop progressive web App (PWA) that enables installation offline functionality using a service worker and a web manifest file.

Theory: A progressive web App (PWA) is a modern web application that provides an app like experience on the web. It allows user to install the app on their device, work offline and load content faster. PWA's leverage the following key components.

1. service worker
A background script that enables offline functionality by caching resources. It intercepts network requests and serves cached response when offline.
2. WebApp Manifest
A json file that provides metadata about the PWA such as the apps name theme color start URL and icons. It makes the web app installed on device.
3. Caching strategy
The cache first approach is used in this PWA meaning that if resources are available in cache they are served first otherwise they are fetched from the network.

NAME: STD.: DIV.:

DATE :
PAGE :

Conclusion:

The implementation successfully demonstrates the PWA architecture by enabling installation, offline access and caching. Users can install the app and even if there is no internet connection previously loaded pages will still be accessible. This project highlights the benefits of PWA's such as performance, reliability and user engagement making them a powerful alternative.

Experiment 8

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

Output:

1. **Service Worker Implementation (serviceworker.js)**

```
// Define a cache name for your app
const CACHE_NAME = 'e-commerce-pwa-v1';

// List of files to cache for offline use
const filesToCache = [
 'index.html',
 'manifest.json',
 'images/ig.jpg',
 'images/edge.jpg',
 // Add other assets your app needs
];

// Install event - Cache the essential files
self.addEventListener('install', (event) => {
 console.log('Service Worker: Installing...');
  // waitUntil() ensures the service worker won't install until
the code inside has completed
 event.waitUntil(
   caches.open(CACHE_NAME)
     .then((cache) => {
       console.log('Service Worker: Caching files');
       return cache.addAll(filesToCache);
     })
     .then(() => self.skipWaiting()) // Force waiting service
worker to become active
 );
});
```

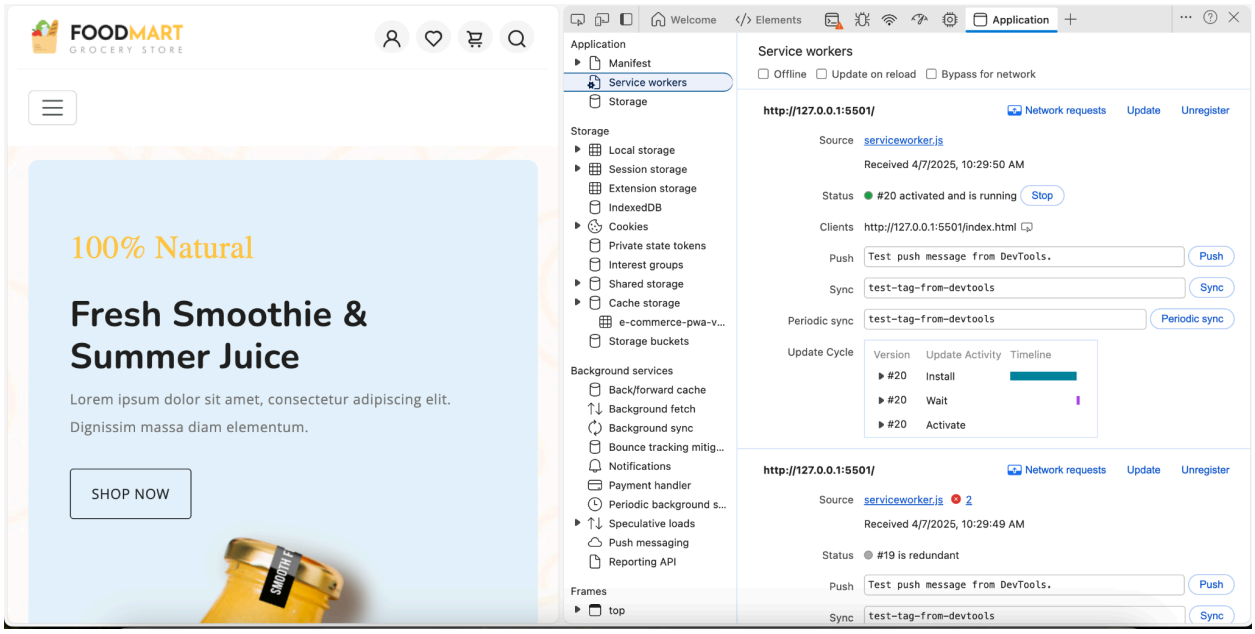
```
// Activate event - Clean up old caches
self.addEventListener('activate', (event) => {
 console.log('Service Worker: Activated');
  // Remove old caches
 event.waitUntil(
   caches.keys().then((cacheNames) => {
     return Promise.all(
       cacheNames.map((cache) => {
         if (cache !== CACHE_NAME) {
           console.log('Service Worker: Clearing old cache',
cache);
           return caches.delete(cache);
         }
       })
     );
   })
   .then(() => self.clients.claim()) // Take control of all
clients
 );
});

// Fetch event - Serve from cache first, then network
self.addEventListener('fetch', (event) => {
 console.log('Service Worker: Fetching', event.request.url);
  event.respondWith(
   // Try the cache first
   caches.match(event.request)
     .then((response) => {
       // Return cached response if found
       if (response) {
         return response;
       }

       // If not in cache, fetch from network
```

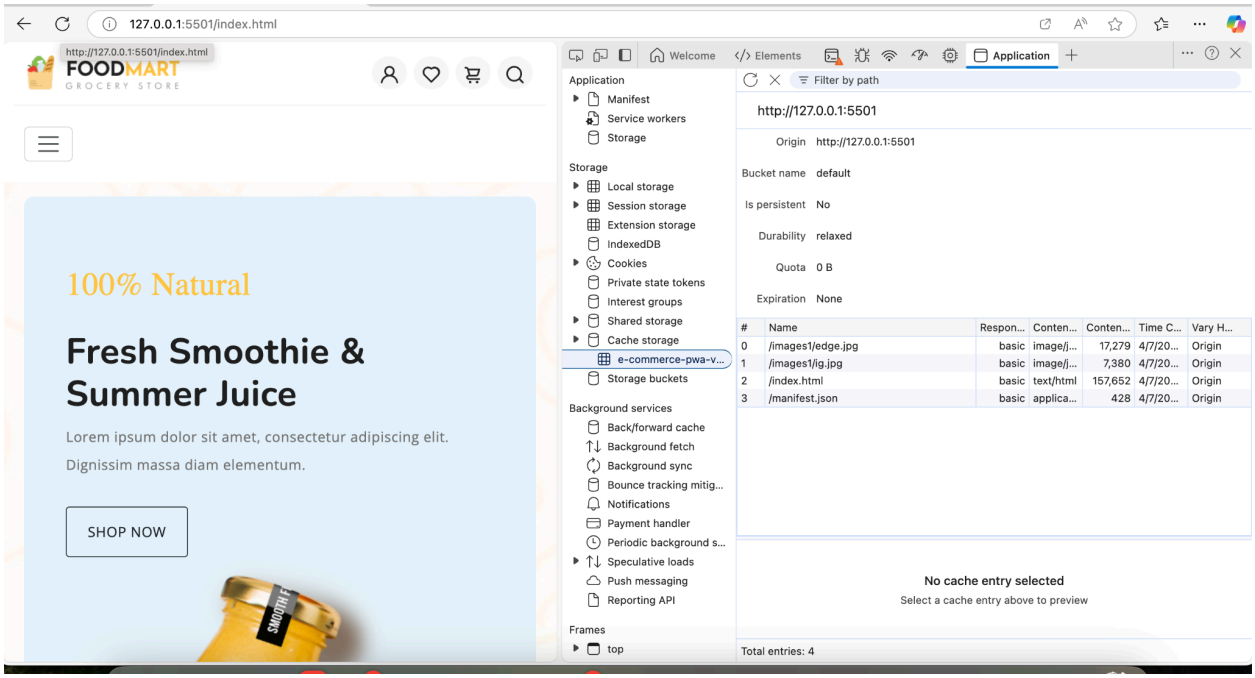
```
        return fetch(event.request)
          .then((res) => {
            // Check if valid response
            if (!res || res.status !== 200 || res.type !==
'basic') {
              return res;
            }

            // Clone the response
            let responseToCache = res.clone();

            // Add the new file to cache
            caches.open(CACHE_NAME)
              .then((cache) => {
                cache.put(event.request, responseToCache);
              });

            // Return the response
            return res;
          })
          .catch(() => {
            // If both cache and network fail, you could return
a fallback page
            if (event.request.url.indexOf('.html') > -1) {
              return caches.match('index.html');
            }
          });
      })
  );
});
```

2. Changes Made in the index.html

```
if ("serviceWorker" in navigator) {
        window.addEventListener("load", () => {
```

```
        navigator.serviceWorker
          .register("serviceworker.js")
          .then((registration) => {
            console.log(
              "ServiceWorker registration successful with
scope: ",
              registration.scope
            );
          })
          .catch((error) => {
            console.log("ServiceWorker registration failed: ",
error);
          });
      });
    }
```





Conclusion

The implementation successfully demonstrates all stages of the service worker lifecycle as outlined in the practical document. The service worker now enables offline access to the application by caching essential resources during installation and serving them when needed, providing a more reliable user experience even without an internet connection.

Roll No. 44

NAME: Aarya Gandhi STD.: D15B DIV.:

DATE :

PAGE :

MAD Experiment 9

Aim: To implement and understand service worker events like fetch, sync and push. For an E-commerce PWA.

Theory: Service Worker is a background script that runs independently in a browser and enables functionalities like offline caching background sync and push notifications.

Key Features of Service Worker:

- Runs in the background without user interaction.
- Intercepts network requests and manages caching using the cache API.
- Works only over HTTPS for security reasons.

Service Worker Event.

- Fetch Event: Manages network requests and caching.

Cache First: Uses cached data first fetching from the network only if needed

Network First: Tries fetching from the network first, falls back to cache if offline

- Sync Event: Enables background sync when the internet is restored
- Push Event: Handles push notifications for updates offers or order statuses

NAME: STD.: DIV.:

DATE :
PAGE :

Conclusion: Thus we successfully implemented service worker events like fetch, sync and push in E-commerce PWA, enabling offline support, background sync and real-time push notifications to improve user experience.

Experiment 9

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

Output:

Manifest.json

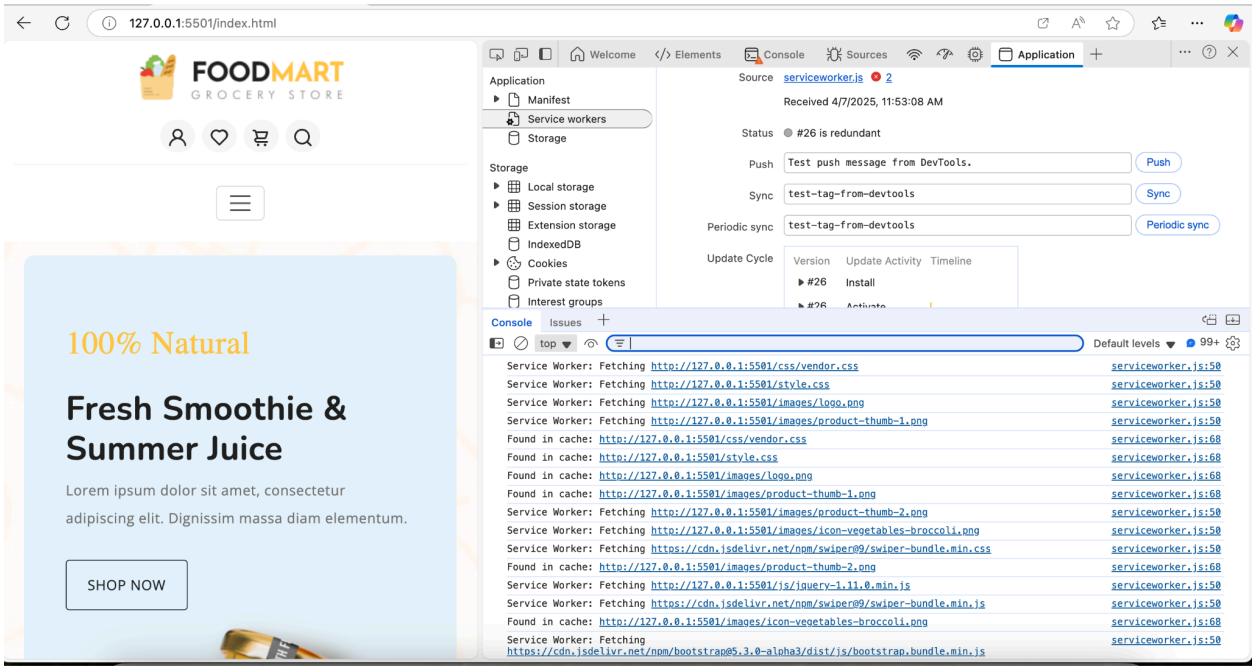
```
{
"name": "MPL experiment 7",
"short_name": "PWA",
"start_url": "/index.html",
"display": "standalone",
"background_color": "#5900b3",
"theme_color": "black",
"scope": "/",
"description": "This is a PWA tutorial.",
"icons": [
  {
    "src": "/images1/ig.jpg",
    "sizes": "192x192",
    "type": "image/jpg"
  },
   {
    "src": "/images1/edge.jpg",
    "sizes": "512x512",
    "type": "image/jpg"
  }

]
}
```

Index.html (Script)

```
document.addEventListener("DOMContentLoaded", function () {
   // Register Service Worker after page load
   window.addEventListener("load", () => {
     registerSW();
   });

   async function registerSW() {
     if ("serviceWorker" in navigator) {
```

```
    try {
      await navigator.serviceWorker.register("/serviceworker.js");
      console.log("Service Worker Registered Successfully");
    } catch (error) {
      console.error("Service Worker Registration Failed:", error);
    }
  } else {
    console.warn("Service Worker is not supported in this browser.");
  }
}

// Place Order Button Logic
const placeOrderBtn = document.getElementById("place-order-btn");
if (placeOrderBtn) {
  placeOrderBtn.addEventListener("click", function () {
    if ("serviceWorker" in navigator && "SyncManager" in window) {
      const order = {
        id: Date.now(),
        items: [
          { name: "Product 1", price: 99.99, quantity: 1 },
          { name: "Product 2", price: 49.99, quantity: 2 }
        ],
        total: 199.97,
        timestamp: new Date().toISOString()
      };

      console.log("Order saved locally:", order);
      alert("Order placed! It will sync when you're online.");

      navigator.serviceWorker.ready.then(function (registration) {
        registration.sync.register("order-sync")
          .then(() => {
            console.log("Sync registered for order");
          })
          .catch(err => {
            console.error("Sync registration failed:", err);
          });
      });
    } else {
      alert("Background Sync not supported in your browser");
    }
  });
```

```
  } else {
    console.warn("Place Order button not found");
  }

  // Test Push Notification Logic
  const testPushBtn = document.getElementById("test-push-btn");
  if (testPushBtn) {
    testPushBtn.addEventListener("click", function () {
      if ("serviceWorker" in navigator) {
        navigator.serviceWorker.ready.then(function (registration) {
          Notification.requestPermission().then(function (permission) {
            if (permission === "granted") {
              registration.active?.postMessage({
                type: "push",
                data: {
                  method: "pushMessage",
                  message: "Your order is ready for pickup!",
                  url: "/orders"
                }
              });
            } else {
              alert("Notification permission denied");
            }
          });
        });
      } else {
        alert("Service Worker not supported in your browser");
      }
    });
  } else {
    console.warn("Test Push button not found");
  }
});
```


127.0.0.1:5501/index.html
FOODMART
GROCERY STORE
100% Natural
Fresh Smoothie & Summer Juice
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Dignissim massa diam elementum.
SHOP NOW
Welcome
Elements
Console
Sources
Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Source serviceworker.js 2
Received 4/7/2025, 11:53:08 AM
Status #26 is redundant
Push Test push message from DevTools. Push
Sync test-tag-from-devtools Sync
Periodic sync test-tag-from-devtools Periodic sync
Update Cycle
Version Update Activity Timeline
#26 Install
Console
Issues
top
Default levels
99+
Service Worker: Fetching http://127.0.0.1:5501/css/vendor.css serviceworker.js:50
Service Worker: Fetching http://127.0.0.1:5501/style.css serviceworker.js:50
Service Worker: Fetching http://127.0.0.1:5501/images/logo.png serviceworker.js:50
Service Worker: Fetching http://127.0.0.1:5501/images/product-thumb-1.png serviceworker.js:50
Found in cache: http://127.0.0.1:5501/css/vendor.css serviceworker.js:68
Found in cache: http://127.0.0.1:5501/style.css serviceworker.js:68
Found in cache: http://127.0.0.1:5501/images/logo.png serviceworker.js:68
Found in cache: http://127.0.0.1:5501/images/product-thumb-1.png serviceworker.js:68
Service Worker: Fetching http://127.0.0.1:5501/images/product-thumb-2.png serviceworker.js:50
Service Worker: Fetching http://127.0.0.1:5501/images/icon-vegetables-broccoli.png serviceworker.js:50
Service Worker: Fetching https://cdn.jsdelivr.net/npm/swiper@9/swiper-bundle.min.css serviceworker.js:50
Found in cache: http://127.0.0.1:5501/images/product-thumb-2.png serviceworker.js:68
Service Worker: Fetching http://127.0.0.1:5501/js/jquery-1.11.0.min.js serviceworker.js:50
Service Worker: Fetching https://cdn.jsdelivr.net/npm/swiper@9/swiper-bundle.min.js serviceworker.js:50
Found in cache: http://127.0.0.1:5501/images/icon-vegetables-broccoli.png serviceworker.js:68
Service Worker: Fetching https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js serviceworker.js:50

Roll No. 14

NAME: Aarya Gramelli STD.: D15B DIV.:

DATE :

PAGE :

MPL Experiment 10

Aim: To study and implement the deployment of an E-commerce PWA to Github pages.

Theory: Github page is a free hosting service provided by github for deploying static websites directly from a repository. It is used for hosting personal projects, documentations and simple web apps.

-) Features of Github pages
supports Jekyll blogging
Allows custom URL's
provides an automatic page generator.

-) Pros of Github Pages
Free and easy to use
Direct integration with Github repositories.
support custom domains.

-) Cons of Github Pages
Code is public unless using a paid private repository
No https support for custom domains
limited plugin support for Jekyll

Conclusion
Thus we sucessfully deployed our E-commerce PWA using Github pages, leveraging its free and easy setup for static web hosting.

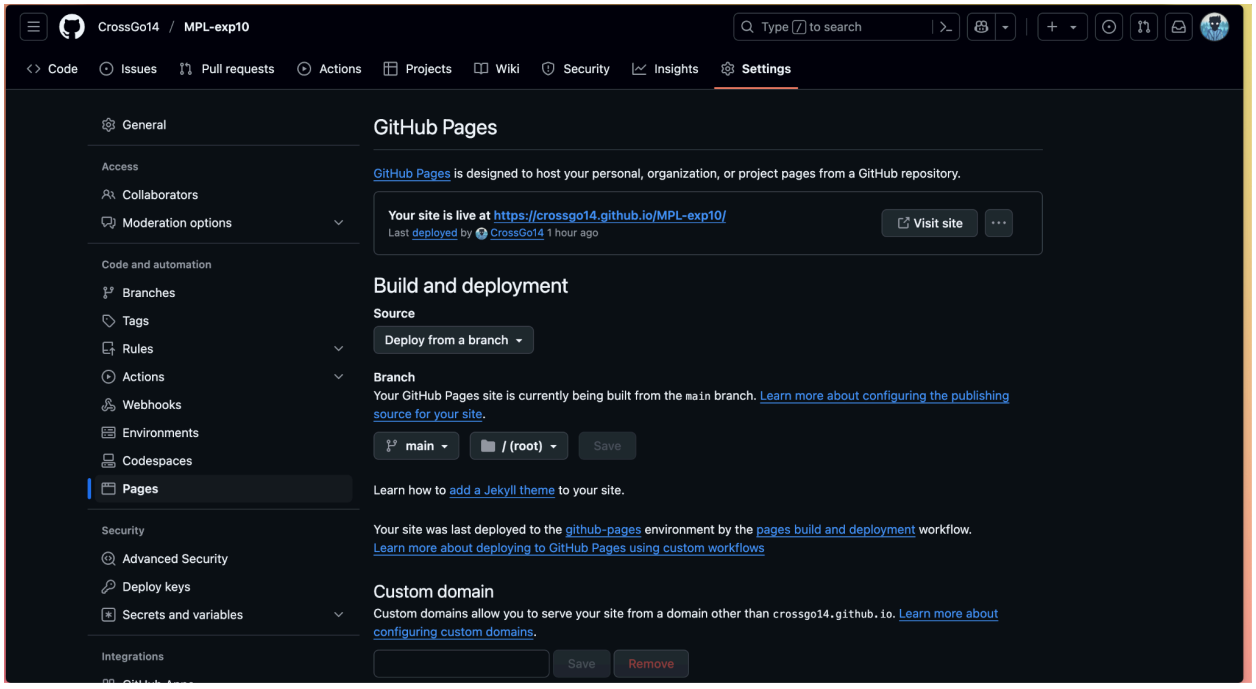
Experiment 10

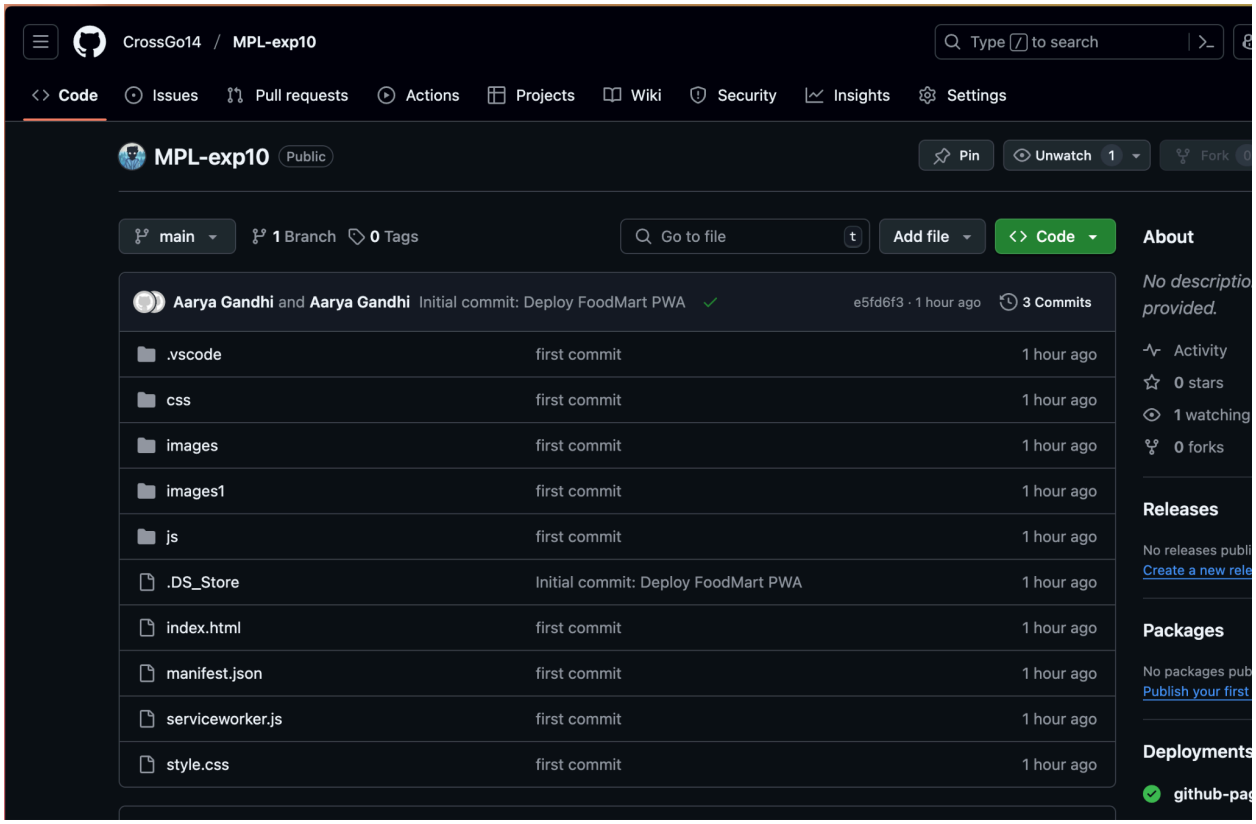
Aim:To study and implement deployment of Ecommerce PWA to GitHub Pages.

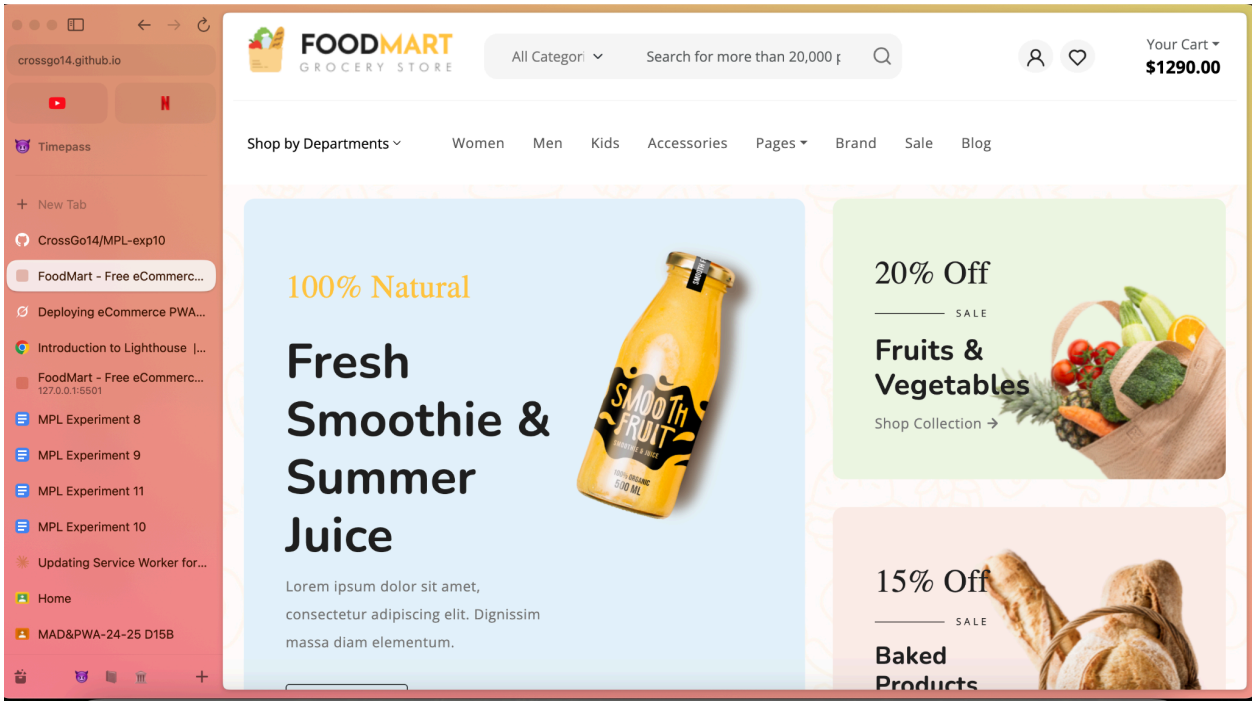
Output:

Steps to Deploy an E-commerce PWA to GitHub Pages

1. **Create a GitHub Repository** – Set up a new public repository on GitHub.
2. **Initialize Git** – Link your local project to the repository and commit your code.
3. **Push Code to GitHub** – Upload your project files to GitHub.
4. **Enable GitHub Pages** – Go to the repository's **Settings > Pages**, set the source to the main branch, and save.
5. **Install Deployment Tool** – Add the necessary package for GitHub Pages deployment.
6. **Update Project Settings** – Modify the project configuration to specify the deployment URL.
7. **Deploy the PWA** – Build the project and publish it to GitHub Pages.
8. **Access Live Site** – Visit the provided GitHub Pages link to view your deployed PWA




CrossGo14 / MPL-exp10
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
MPL-exp10
Public
Pin
Unwatch 1
Fork
main
1 Branch
0 Tags
Go to file
t
Add file
Code
About
No description provided.
Aarya Gandhi and Aarya Gandhi Initial commit: Deploy FoodMart PWA
e5fd6f3 · 1 hour ago
3 Commits
.vscode first commit 1 hour ago
css first commit 1 hour ago
images first commit 1 hour ago
images1 first commit 1 hour ago
js first commit 1 hour ago
.DS_Store Initial commit: Deploy FoodMart PWA 1 hour ago
index.html first commit 1 hour ago
manifest.json first commit 1 hour ago
serviceworker.js first commit 1 hour ago
style.css first commit 1 hour ago
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Deployments
github-pages


crossgo14.github.io
Timepass
New Tab
CrossGo14/MPL-exp10
FoodMart - Free eCommerc...
Deploying eCommerce PWA...
Introduction to Lighthouse |...
FoodMart - Free eCommerc...
127.0.0.1:5501
MPL Experiment 8
MPL Experiment 9
MPL Experiment 11
MPL Experiment 10
Updating Service Worker for...
Home
MAD&PWA-24-25 D15B
FOODMART
GROCERY STORE
All Categori
Search for more than 20,000 p
Your Cart
$1290.00
Shop by Departments
Women
Men
Kids
Accessories
Pages
Brand
Sale
Blog
100% Natural
Fresh Smoothie & Summer Juice
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Dignissim massa diam elementum.
SMOOTH FRUIT
20% Off
SALE
Fruits & Vegetables
Shop Collection
15% Off
SALE
Baked Products

Roll No. 14

NAME: Aarya Gandhi STD.: D15B DIV.:

DATE :

PAGE :

MPL Experiment 11

Aim: To study and implement the google lighthouse PWA analysis tool to test the progressive web app ~~(PWA)~~ (PWA) functioning.

Theory

Google lighthouse is a open source tool by google used to audit web applications based on various parameters including performance. PWA compliances accessibility and best practises. It helps developer analyze and optimize web apps to function.

Key audit metrics

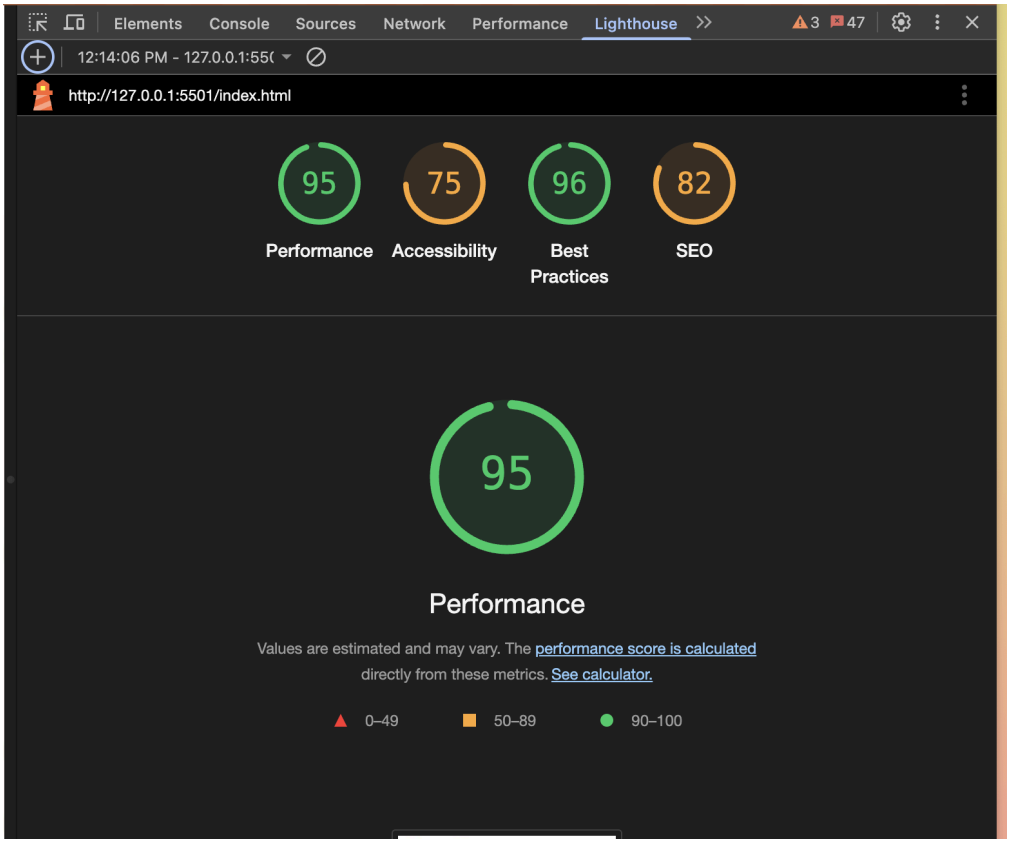
1. Performance :- Measures loading speed rendering time and responsiveness.
2. PWA compliance - Ensures the app meets PWA standards such as service workers, offline functionality and manifests.
3. Accessibility - Evaluates the apps usability for differently abled users including screen reader compatibility.

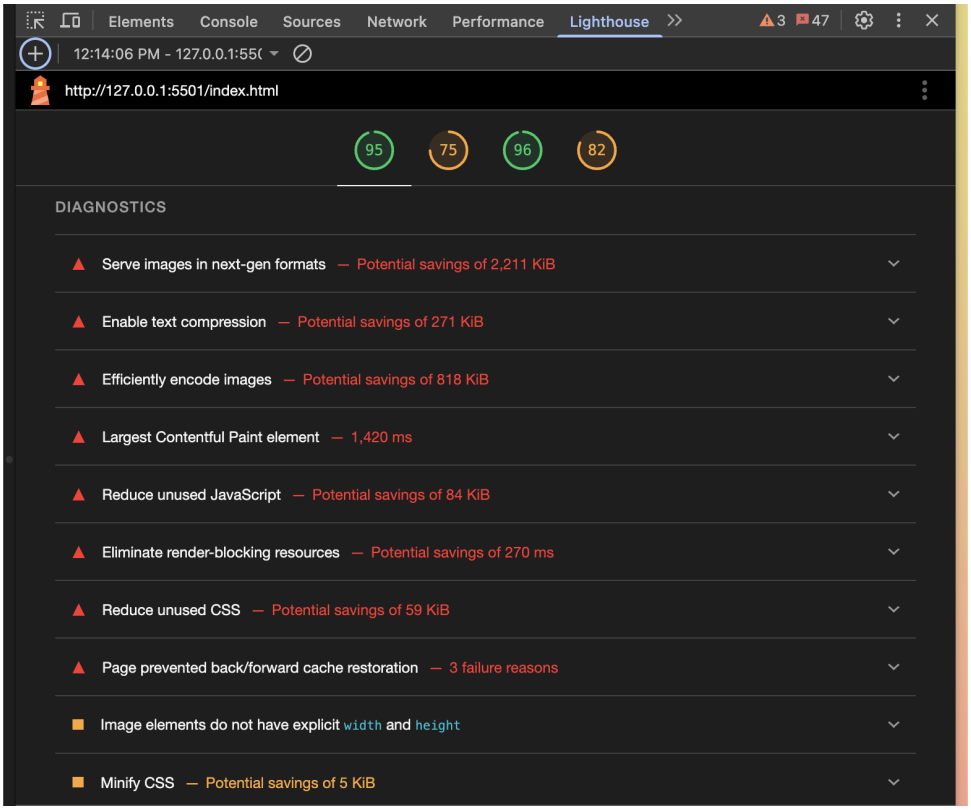
Lighthouse provides actionable insights to improve website efficiency, ensuring a better user experience and higher engagement.

Conclusion: Thus we successfully used the google lighthouse PWA analysis tool to evaluate and optimize the PWA improving its performance, accessibility and compliance.

Experiment 11

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





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