Title: Women Safety App

Problem Statement

Women face significant safety concerns when carrying out daily activities, especially in public spaces. Instances of harassment and other forms of violence against women remain prevalent, making it crucial to develop a solution that ensures their security and provides immediate assistance in times of distress.

Summary

The "Women Safety App" is a mobile application designed to provide immediate assistance, peace of mind, and enhanced security in potentially dangerous situations. By offering features such as one-tap emergency alerts, real-time location sharing, and direct communication with local law enforcement and trusted contacts, the app aims to deliver a quick and efficient response in emergencies.

Key Features

1. SOS Panic Button with Real-time Alerts
    - A prominently placed, easily accessible SOS button that, when pressed, immediately alerts emergency contacts, law enforcement, and security services with the user's real-time location.
2. Voice-Activated Emergency Response
    - In cases where the user is unable to physically interact with the phone, a voice-activated feature allows them to trigger an alert by saying a preset command (e.g., "Help me").
3. Location Tracking & Safe Routes
    - Users can share their live location with trusted contacts and access safe route suggestions to avoid high-risk areas.
4. Crowdsourced Rescue Assistance
    - If someone is in danger, the app can alert nearby users who have opted to assist, ensuring faster help in emergencies.
5. Invisible SOS Signal
    - The app allows users to send a secret emergency message to their contacts by pressing a hidden button or entering a discreet code, ensuring safety even in hostile situations.

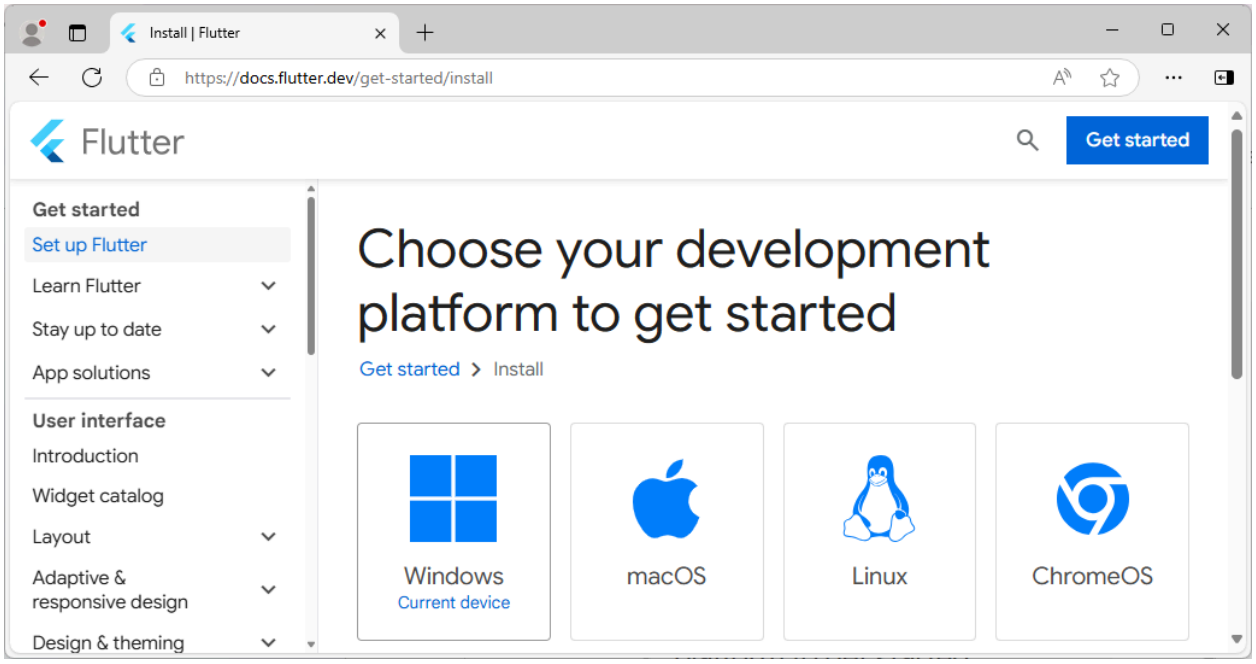
Conclusion

Through this project, I have learned how technology plays a vital role in ensuring safety, security, and empowerment for women in today's world. This app goes beyond merely providing emergency services—it fosters a sense of security, independence, and peace of mind by offering a suite of features that cater to different safety needs.

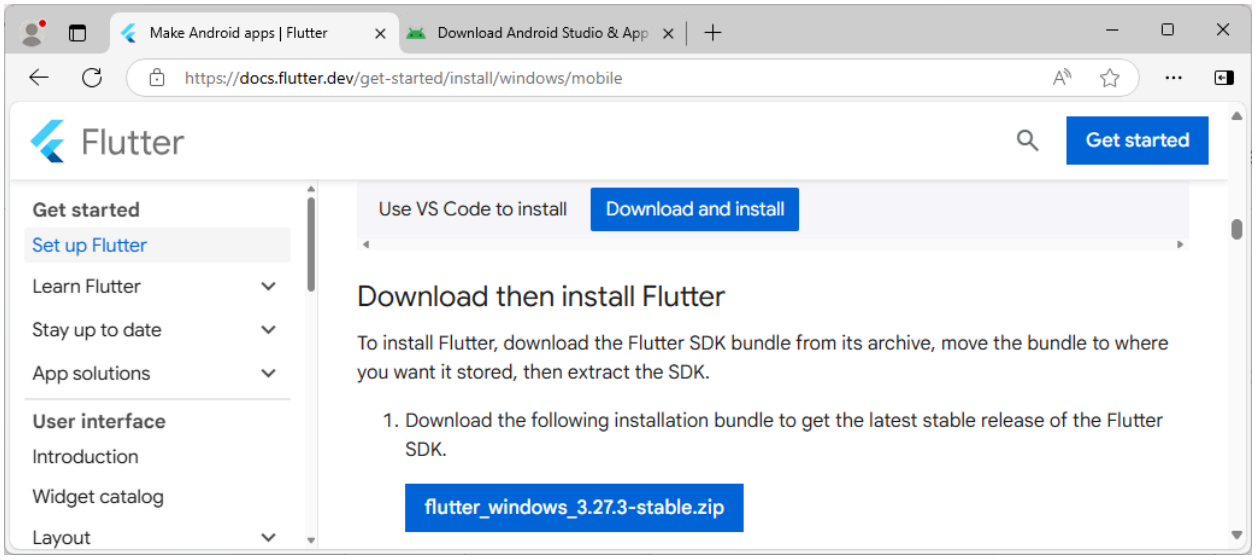
Aim: Installation and Configuration of Flutter Environment with Android Studio.

Install the Flutter SDK

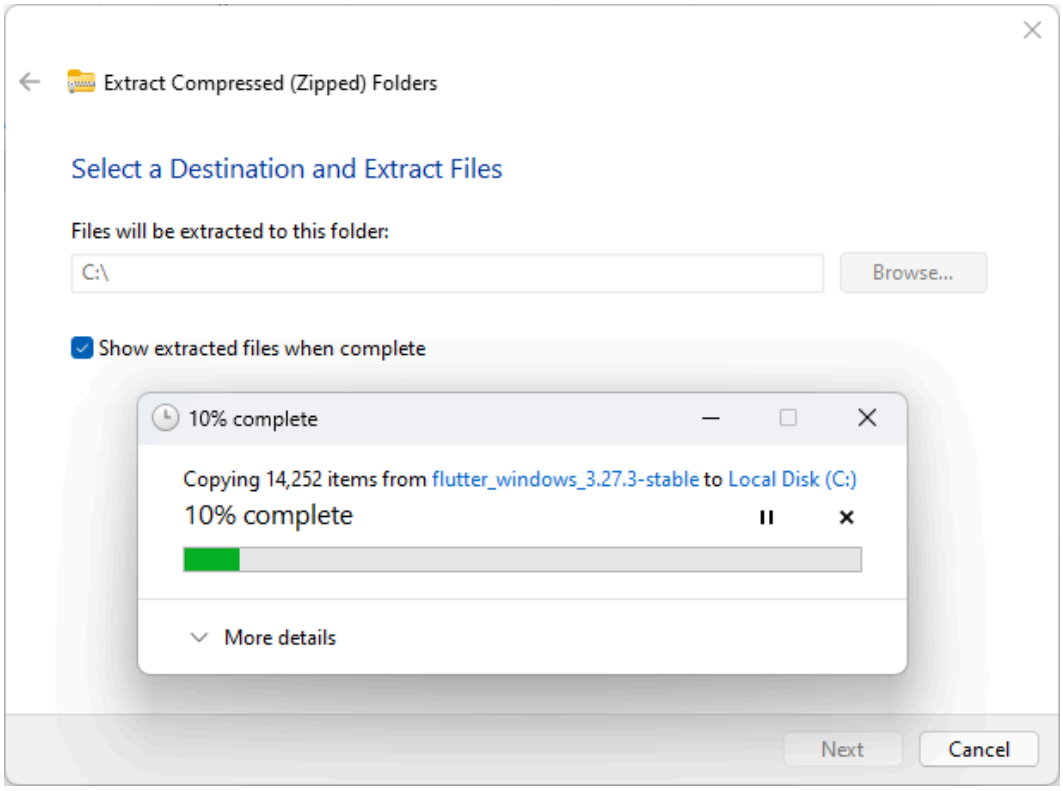
Step 1: Download the installation bundle of the Flutter Software Development Kit for windows. To download Flutter SDK, Go to its official website https://docs.flutter.dev/get-started/install , you will get the following screen.



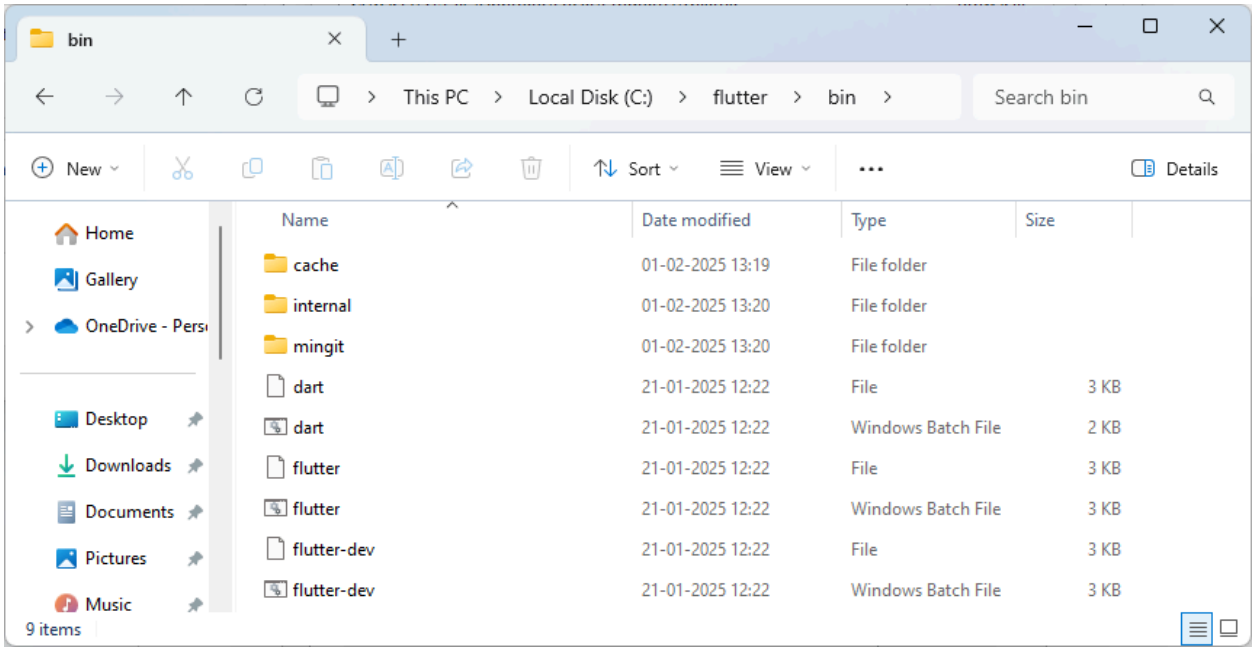
Step 2: Next, to download the latest Flutter SDK, click on the Windows icon. Here, you will find the download link for SDK.



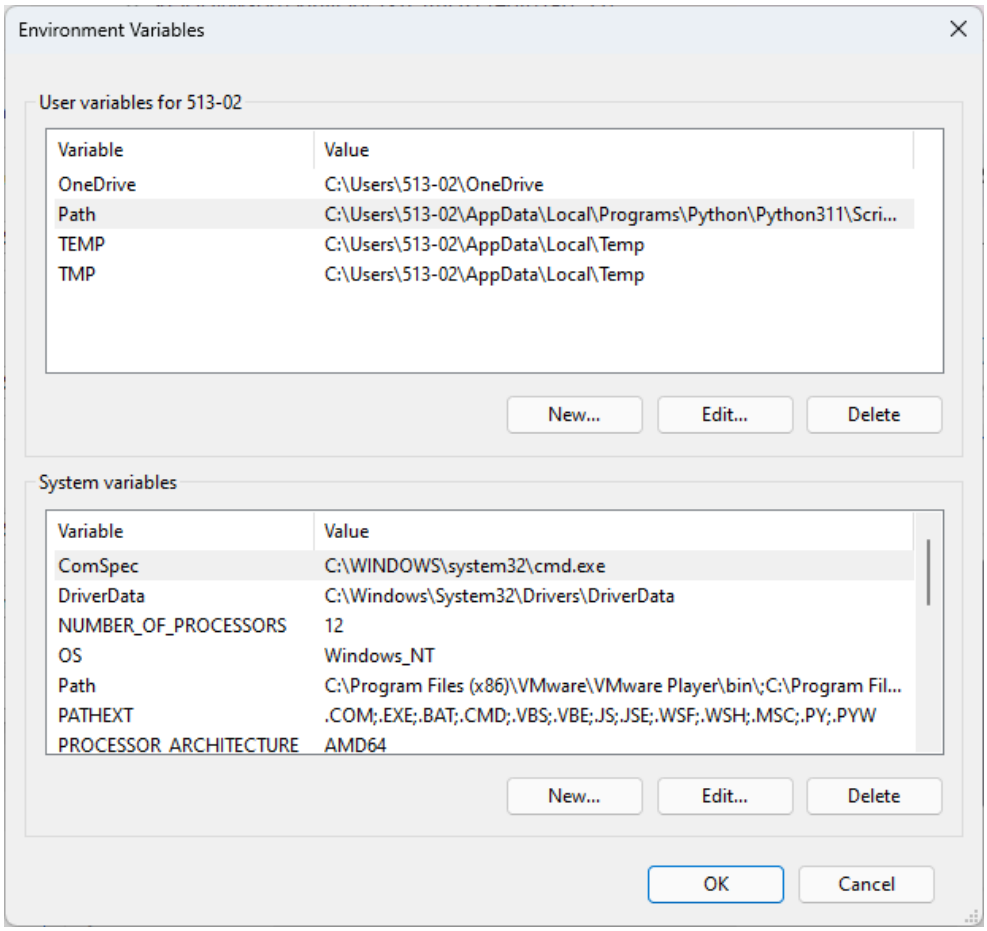
Step 3: When your download is complete, extract the zip file and place it in the desired installation folder or location, for example, C: /Flutter.



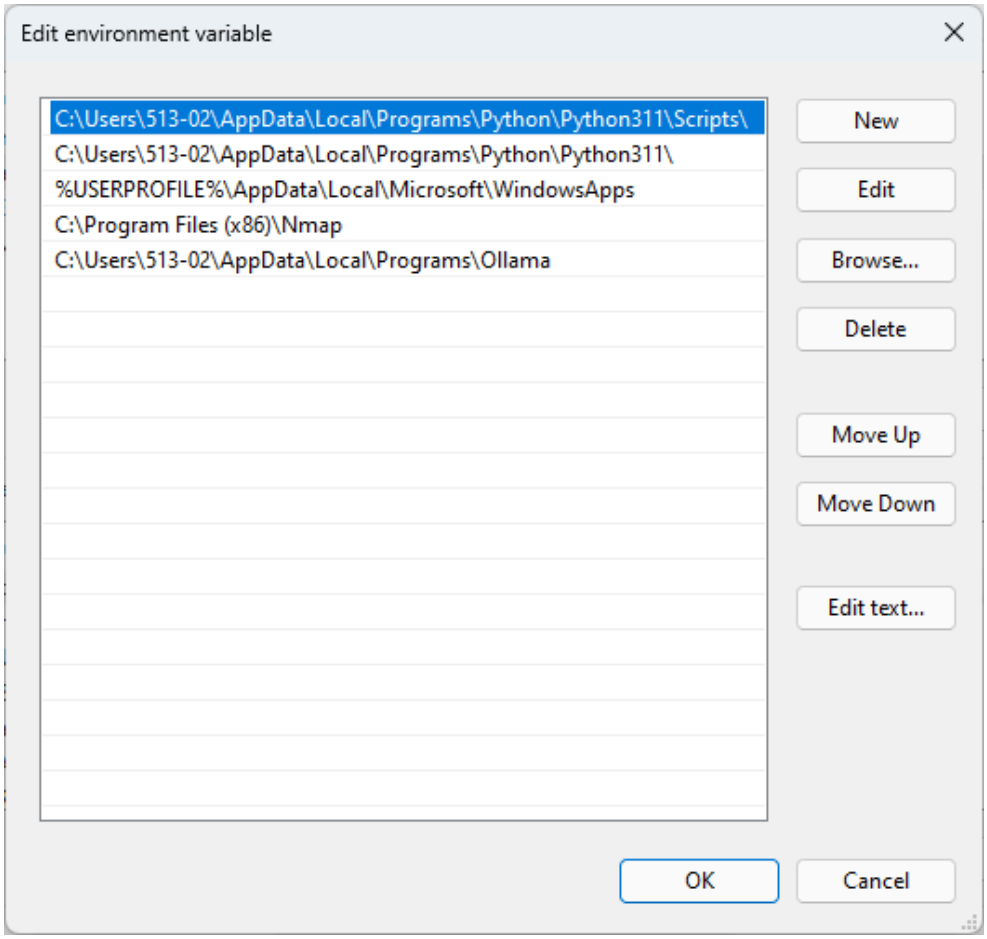
Step 4: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:



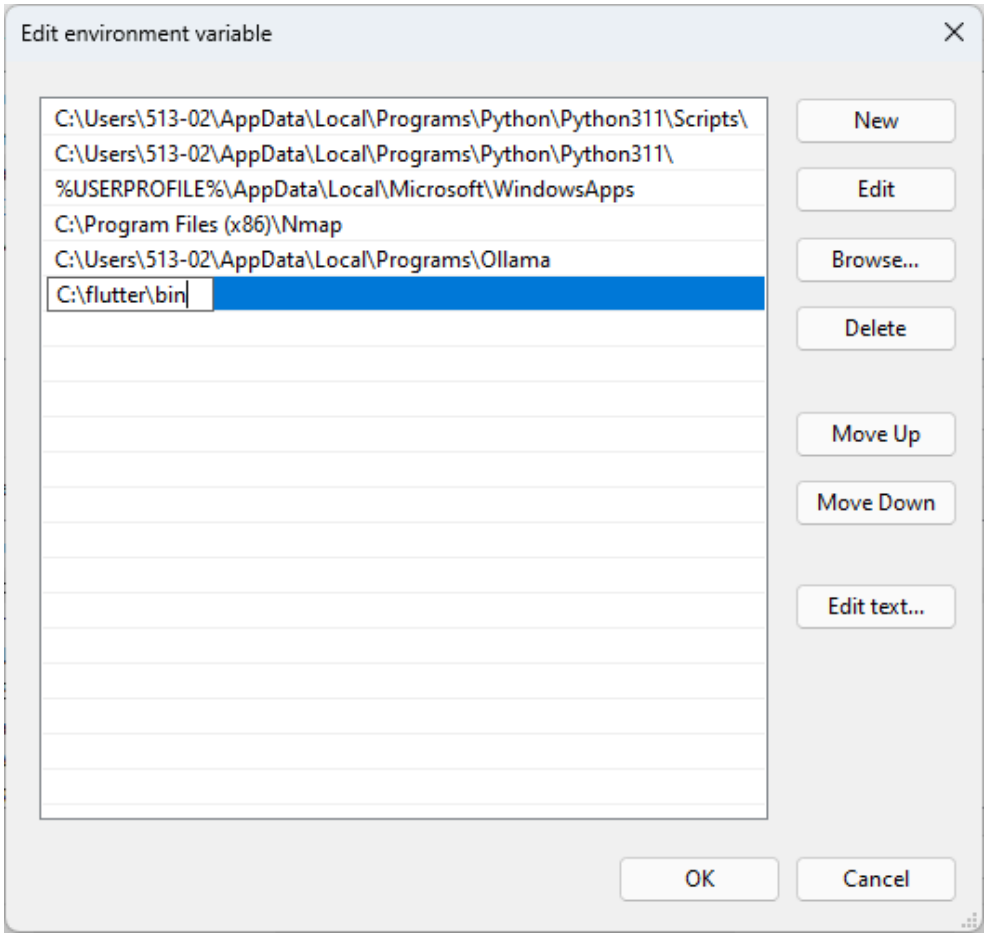
Step 4.1: Go to MyComputer properties -> advanced tab -> environment variables. You will get the following screen.



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

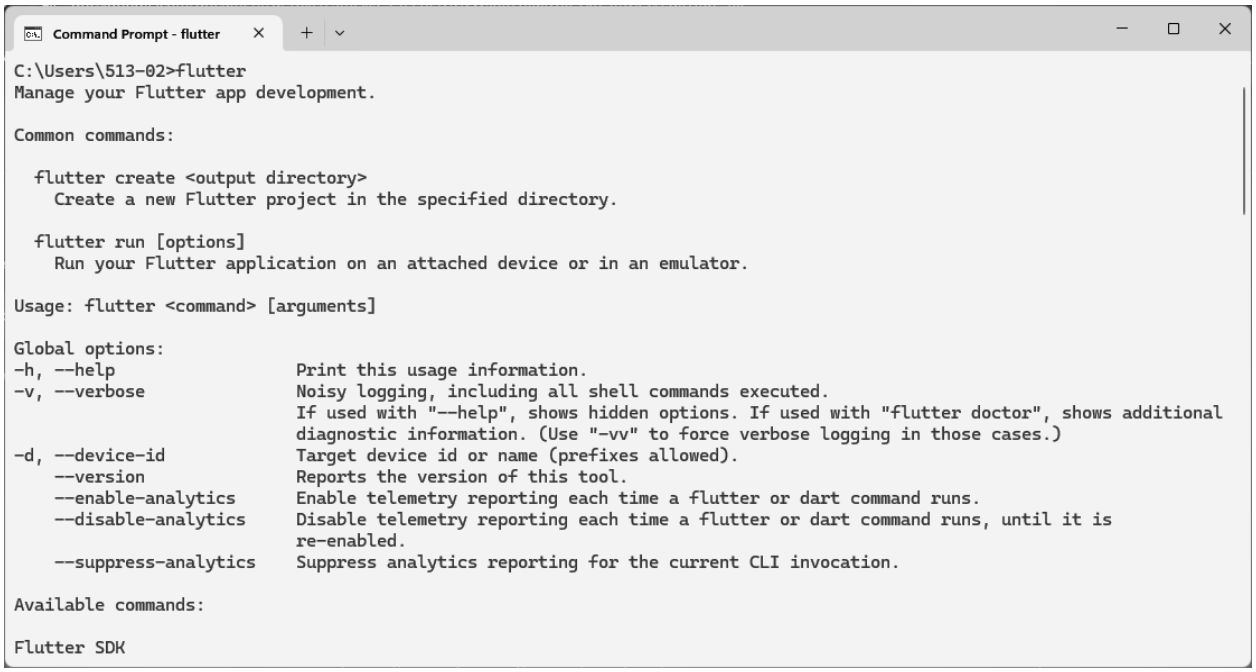


Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value -> ok -> ok -> ok.

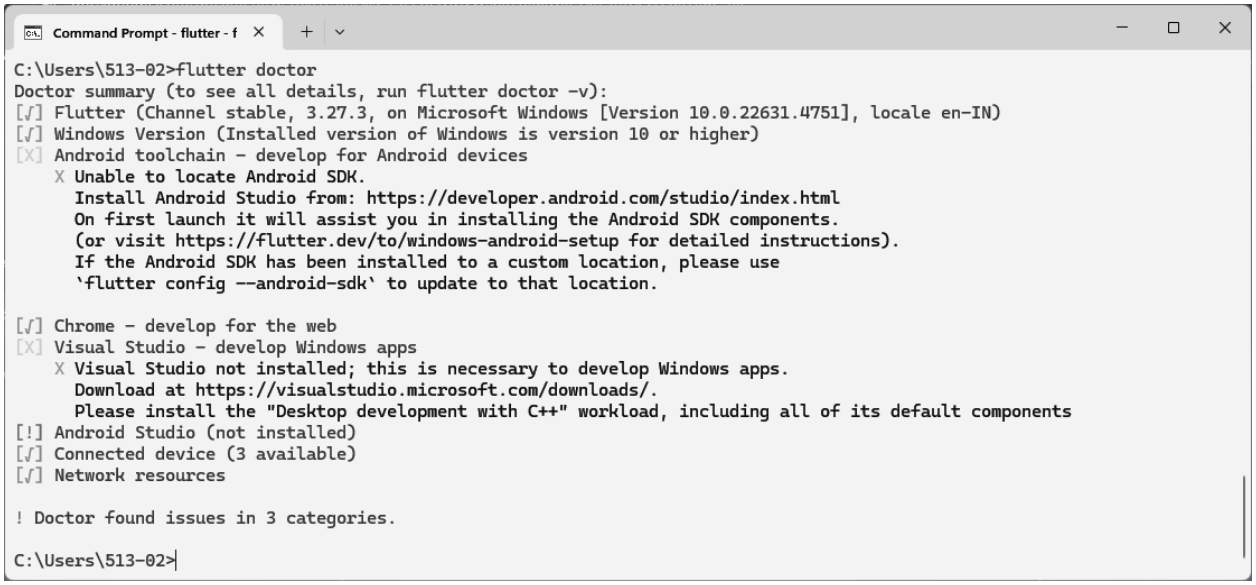


Step 5: Now, run the $ flutter command in the command prompt.

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.

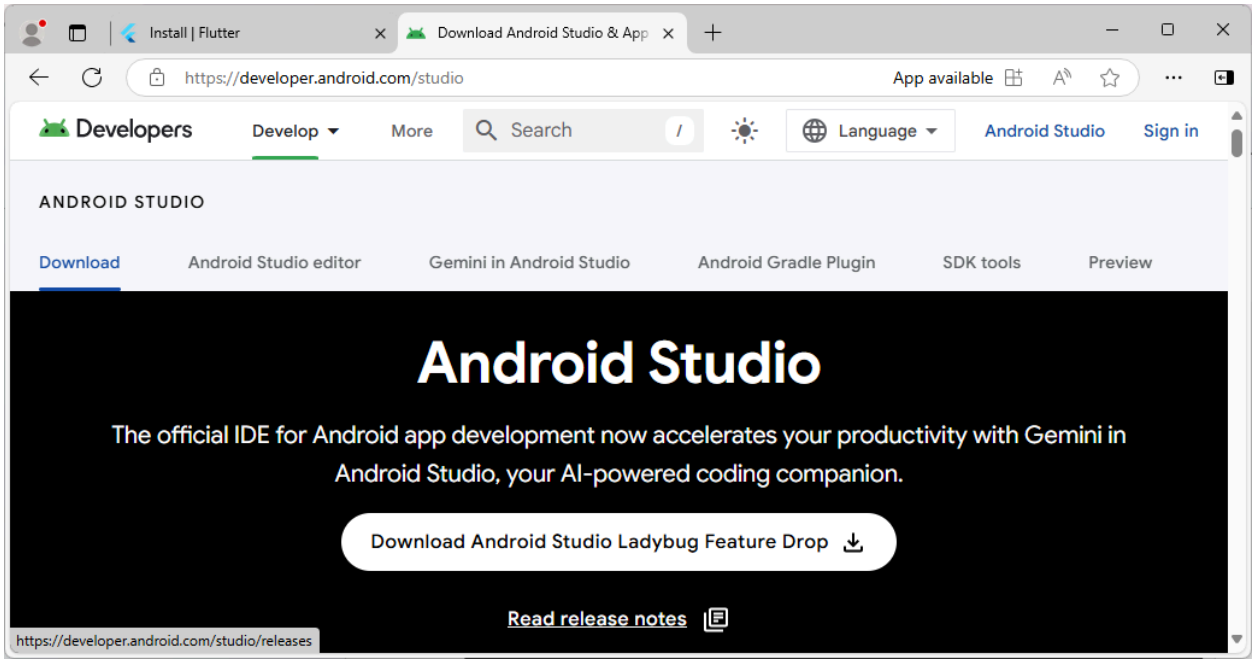


Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

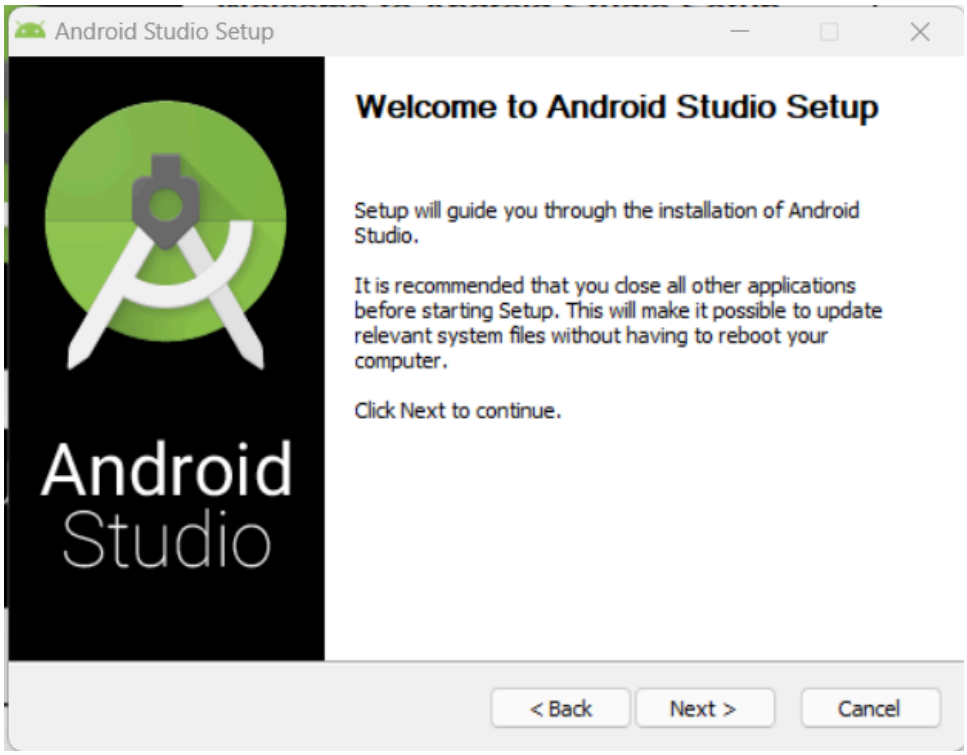


Step 7: Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.

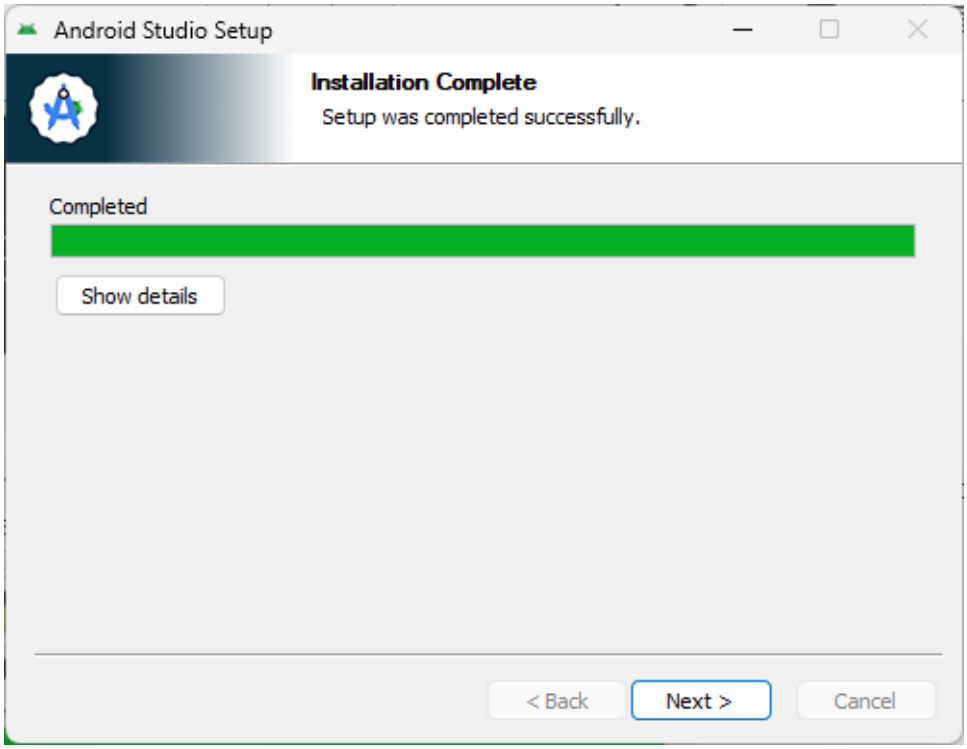
Step 7.1: Download the latest Android Studio executable or zip file from the official site.



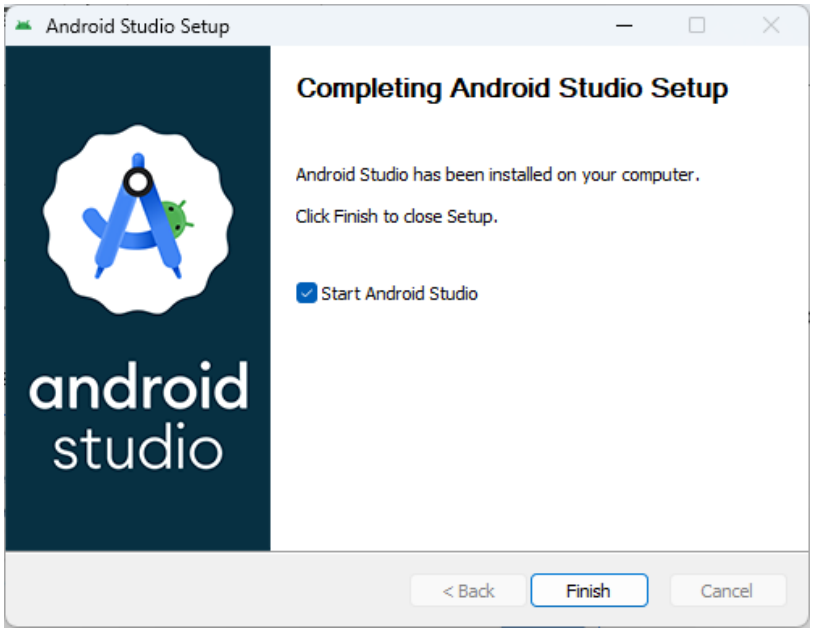
Step 7.2: When the download is complete, open the .exe file and run it. You will get the following dialog box.



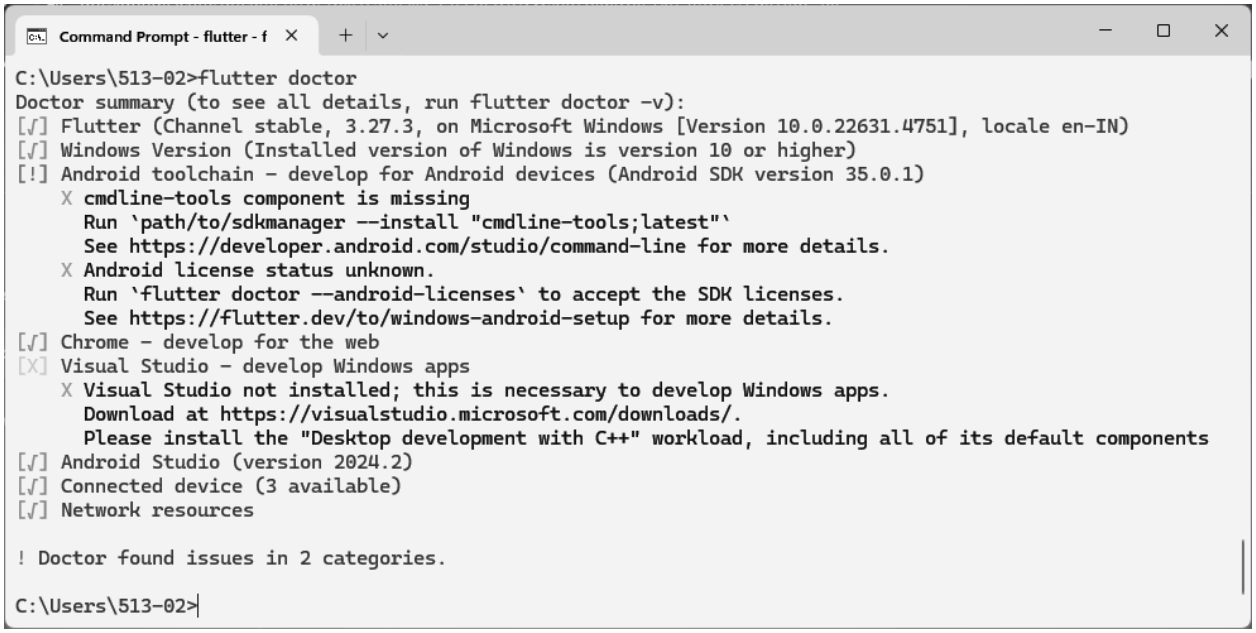
Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.

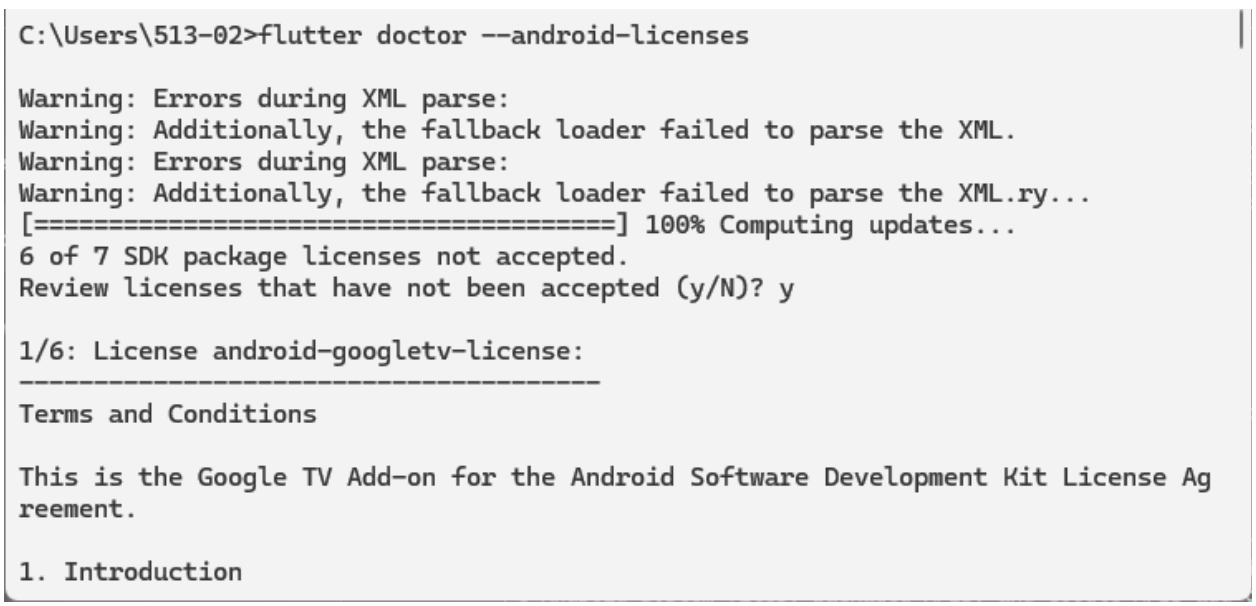


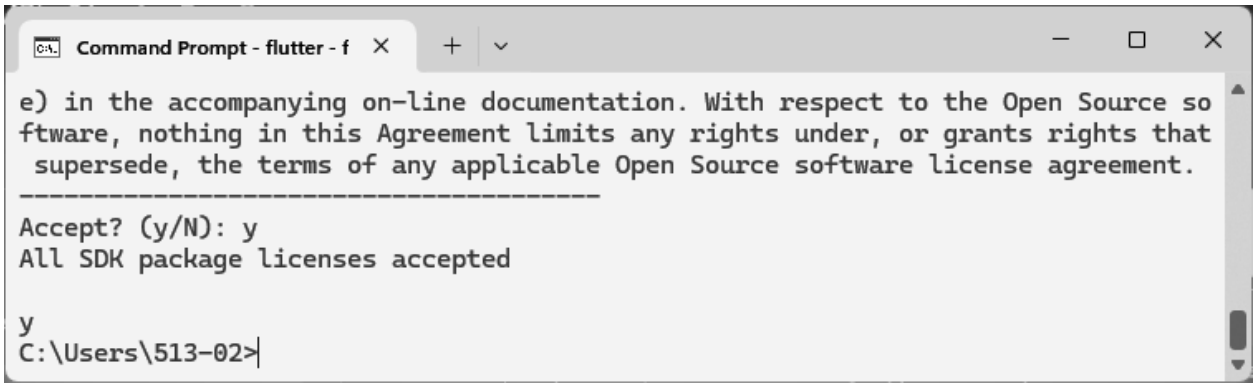
Step 7.4: In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to choose the 'Don't import Settings option' and click OK. It will start the Android Studio.



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

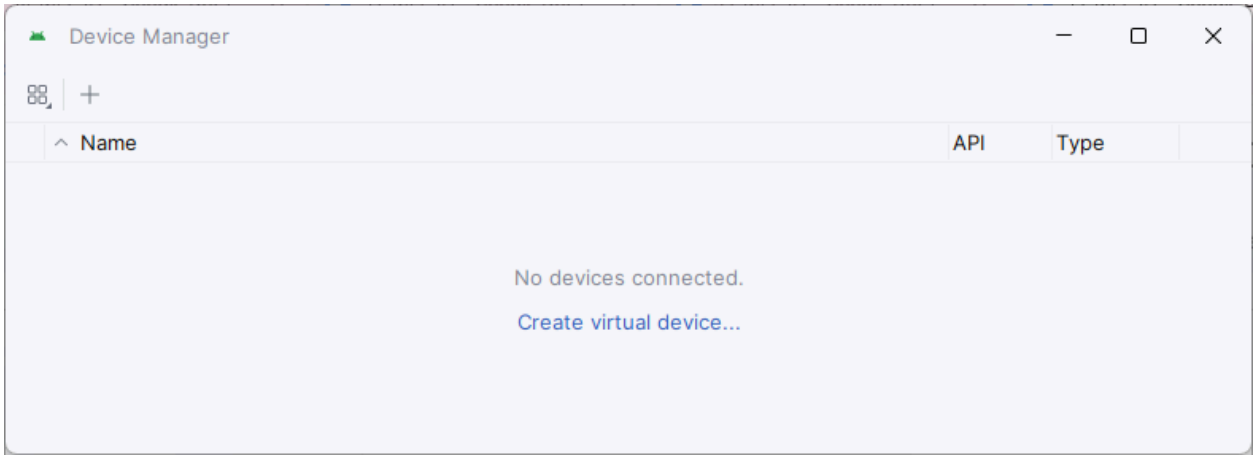




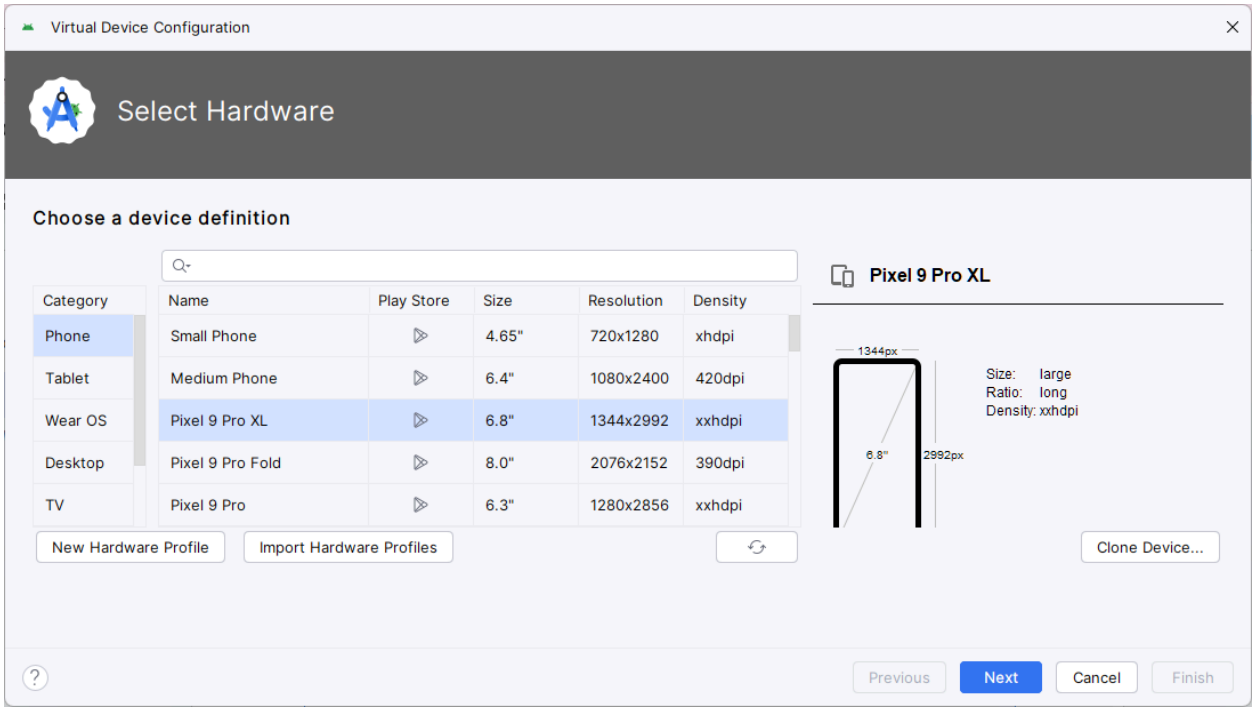


Step 8: Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.

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.

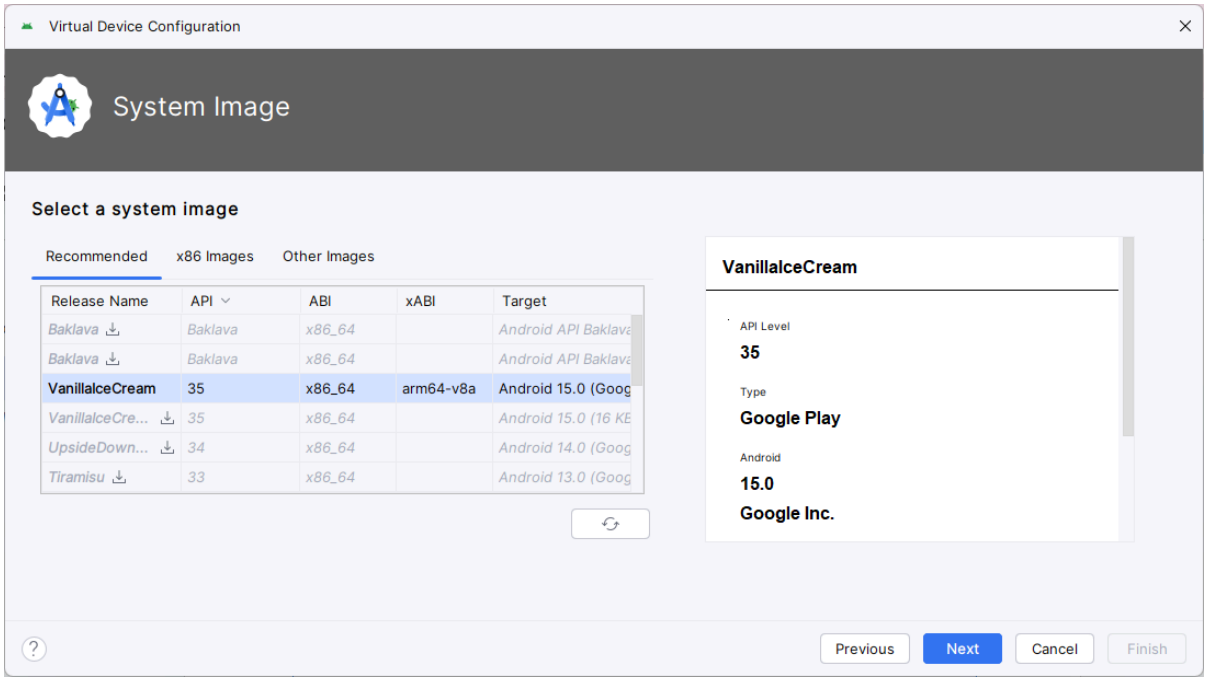


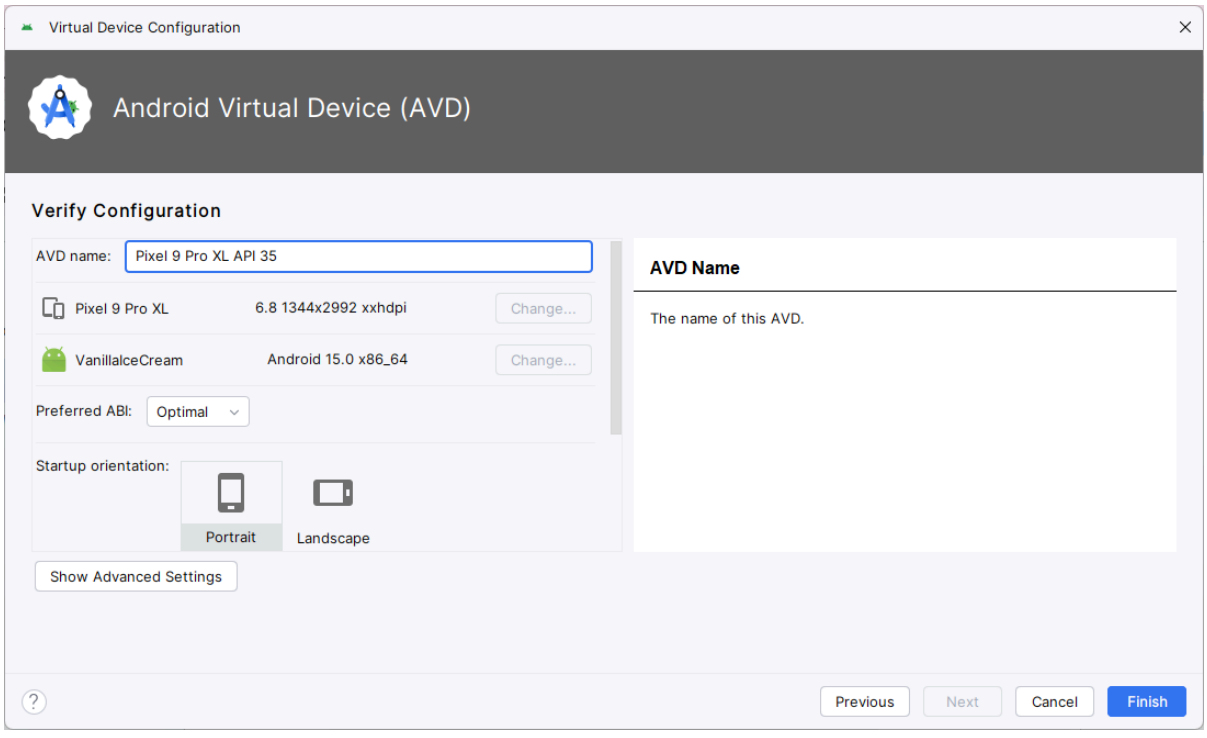
Step 8.2: Choose your device definition and click on Next.



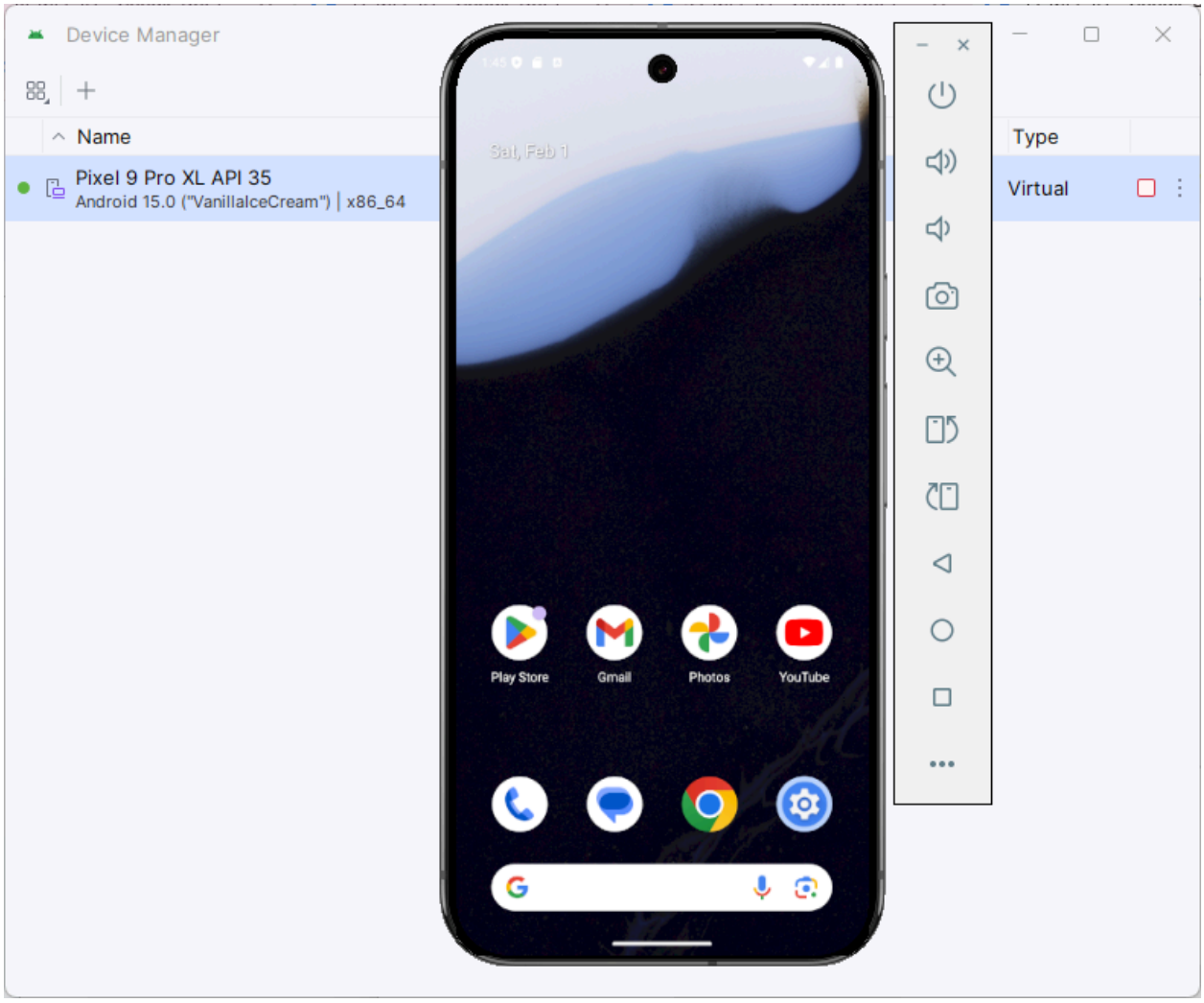
Step 8.3: Select the system image for the latest Android version and click on Next.

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



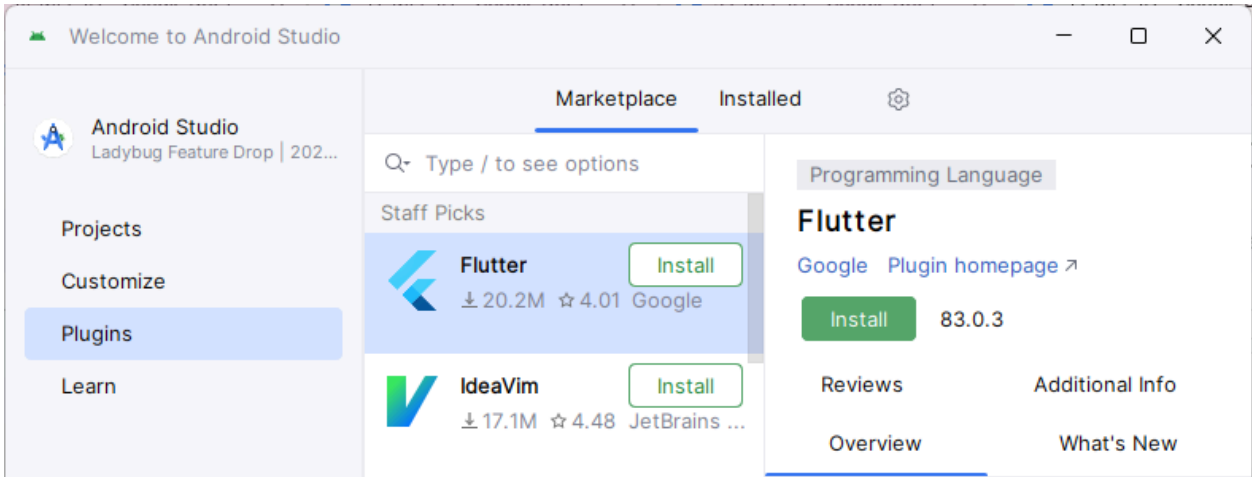


Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

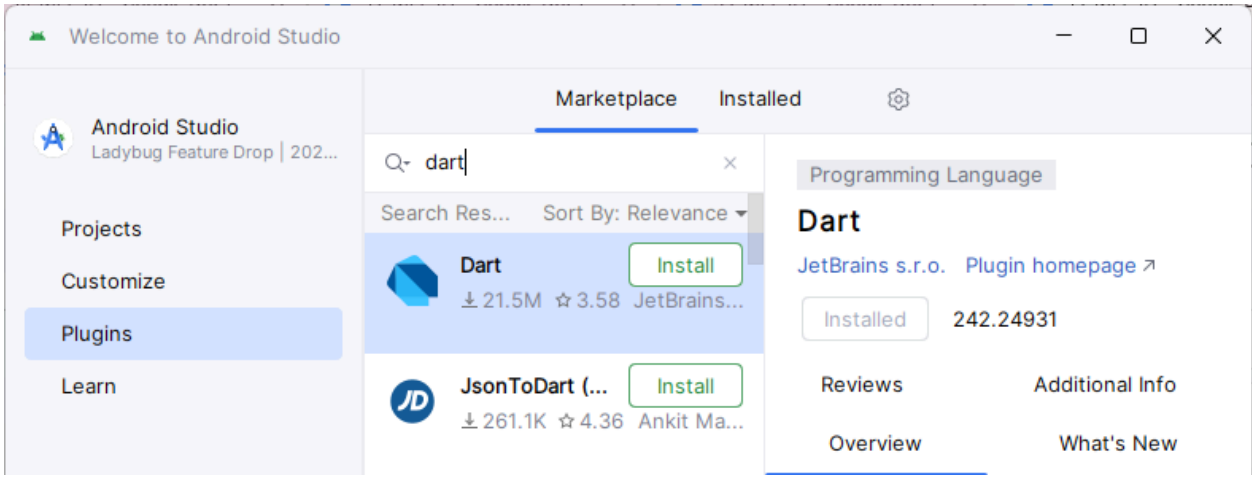


Step 9: Now, install Flutter and Dart plugin for building Flutter application in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.

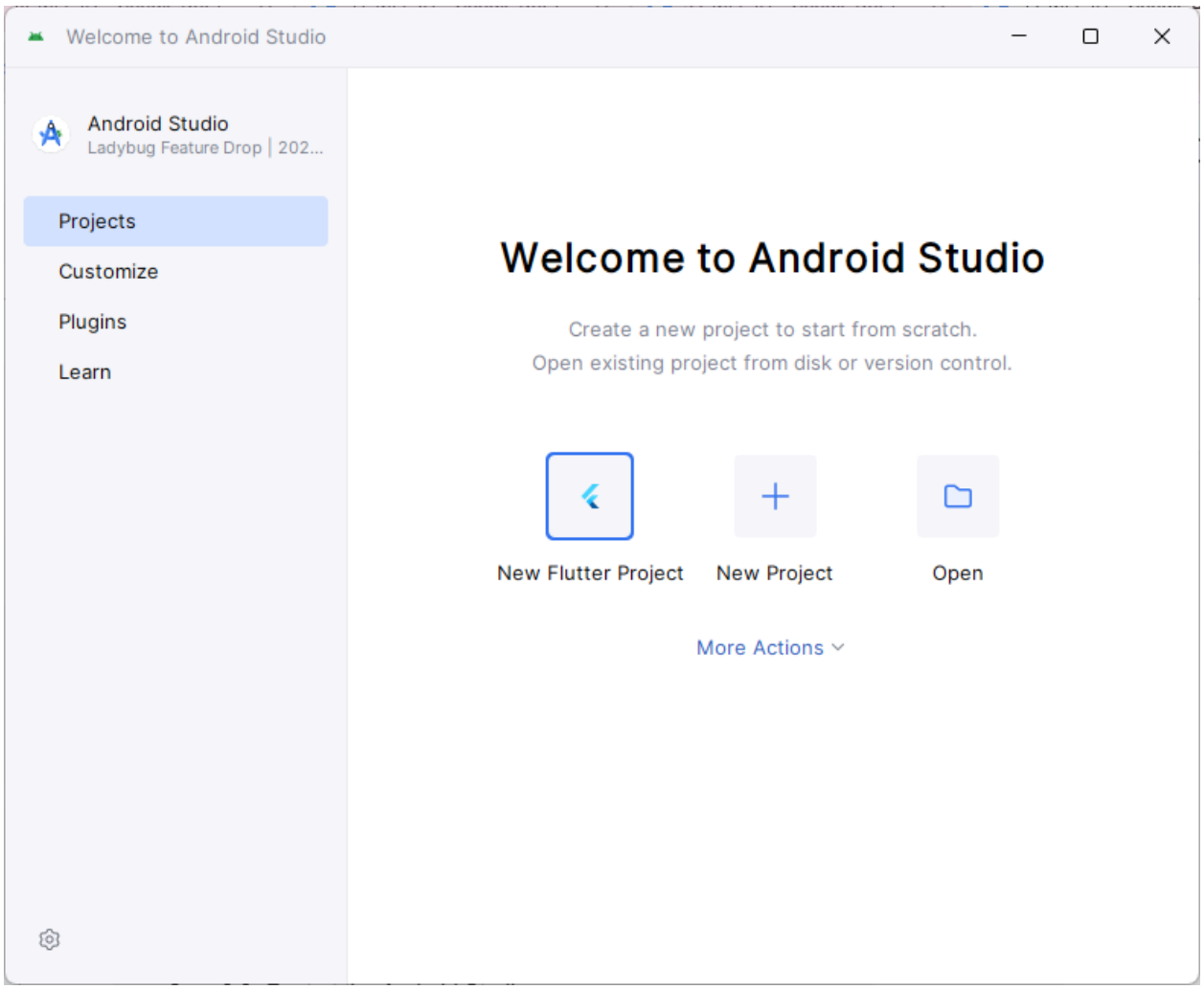
Step 9.1: Open the Android Studio and then go to File->Settings->Plugins.



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.



Aim: To design flutter ui by including common widgets.

Introduction to Flutter UI

Flutter is a UI toolkit developed by Google that helps in building natively compiled applications for mobile, web, and desktop using a single codebase. It uses the Dart programming language and provides a rich set of widgets to create beautiful and responsive user interfaces.

Common Widgets in Flutter

Flutter provides various widgets that help in designing UI easily. Some of the most commonly used widgets include:

- Scaffold: Provides a basic structure for the app, including an app bar and body.
- AppBar: Displays the title and actions at the top of the screen.
- Text: Displays text content.
- ElevatedButton: A button with elevation, used for clickable actions.
- SizedBox: Adds spacing between widgets.
- Column & Row: Arranges widgets vertically and horizontally.
- Icon: Displays icons like home, settings, or emergency symbols.
- Navigator: Handles screen navigation.

Implementation in Our Code

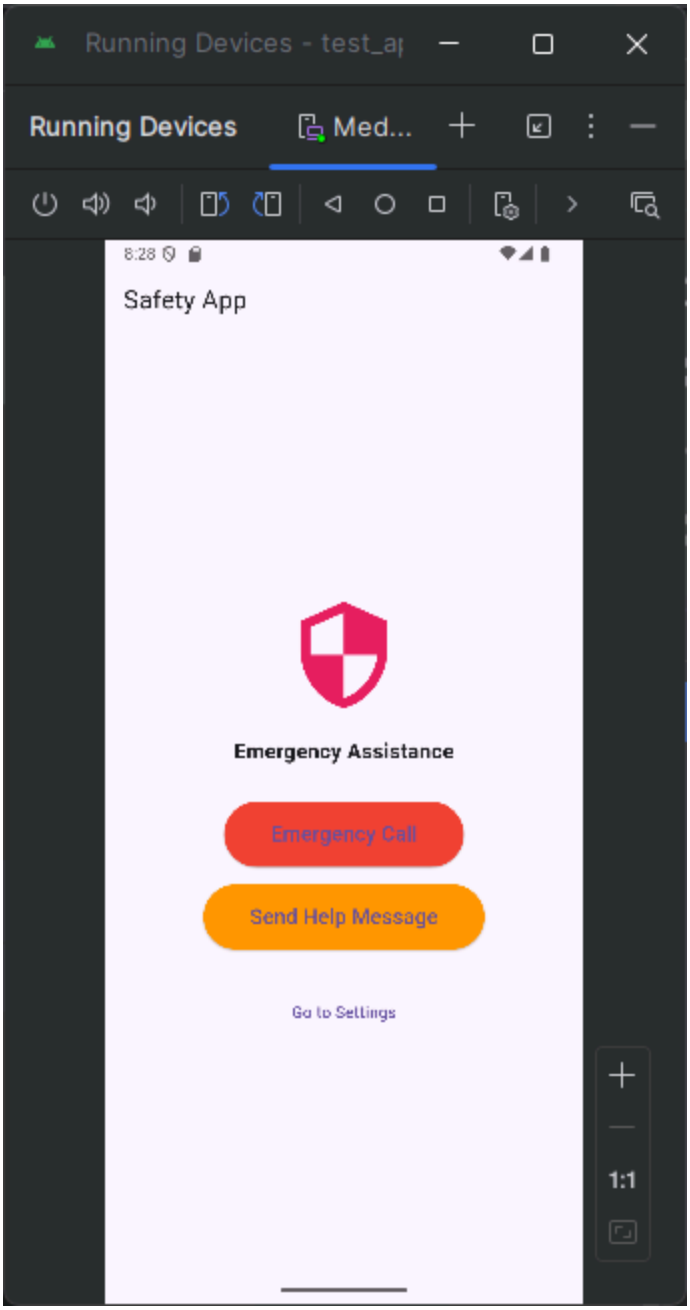
In this lab, we have designed a simple UI for a Woman Safety App using Flutter. The key features of our app include:

1. Home Screen
    - Displays an emergency assistance section.
    - Includes an Emergency Call button (simulated using a print statement).
    - Includes a Send Help Message button to alert contacts.
    - Uses ElevatedButton for interactions.
2. Navigation
    - Clicking on "Go to Settings" takes the user to another screen.
    - Navigator.push() is used to switch between screens.
3. Settings Screen
    - A simple page displaying "Settings Page" text.
    - Shows how multiple screens can be handled in Flutter.

Code :

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

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

class WomanSafetyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Woman Safety App',
      theme: ThemeData(
        primarySwatch: Colors.pink,
      ),
      home: HomeScreen(),
    );
  }
}

class HomeScreen extends StatelessWidget {
  void sendHelpMessage() {
    // Placeholder for SMS or alert functionality
    print("Help message sent!");
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Safety App')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(Icons.security, size: 100, color: Colors.pink),
            SizedBox(height: 20),
            Text(
              "Emergency Assistance",
              style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 30),
            ElevatedButton(
              onPressed: () {
                // Placeholder for emergency call function
                print("Calling emergency number...");
              },
              style: ElevatedButton.styleFrom(
```

```
              backgroundColor: Colors.red,
              padding: EdgeInsets.symmetric(horizontal: 40, vertical: 15),
            ),
            child: Text("Emergency Call", style: TextStyle(fontSize: 18)),
          ),
          SizedBox(height: 15),
          ElevatedButton(
            onPressed: sendHelpMessage,
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.orange,
              padding: EdgeInsets.symmetric(horizontal: 40, vertical: 15),
            ),
            child: Text("Send Help Message", style: TextStyle(fontSize: 18)),
          ),
          SizedBox(height: 30),
          TextButton(
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => SettingsScreen()),
              );
            },
            child: Text("Go to Settings"),
          ),
        ],
      ),
    ),
  );
 }
}

class SettingsScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text('Settings')),
    body: Center(child: Text('Settings Page')),
  );
 }
}
```



Conclusion

In this experiment, we successfully implemented a basic UI for a woman safety app using common Flutter widgets and navigation. Initially, we faced errors like missing widget structuring and incorrect navigation syntax, but we resolved them by carefully using the `Scaffold` structure and properly implementing `Navigator.push()`.

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

Theory:

Introduction

Flutter provides a rich set of widgets to create visually appealing user interfaces. Among these, icons, images, and custom fonts help improve the app's design and usability. Icons provide quick visual cues, images enhance user experience, and custom fonts allow unique styling to match the app's theme.

Implementation in Our Code

In this experiment, we added the following elements to our Woman Safety App:

1. Icons: Used Flutter's built-in Icons.security for the home screen and Icons.settings in the app bar.
2. Images: Added a safety-related image (safety_image.png) from the assets/ folder.
3. Fonts: Integrated the custom Google font Poppins for better typography.

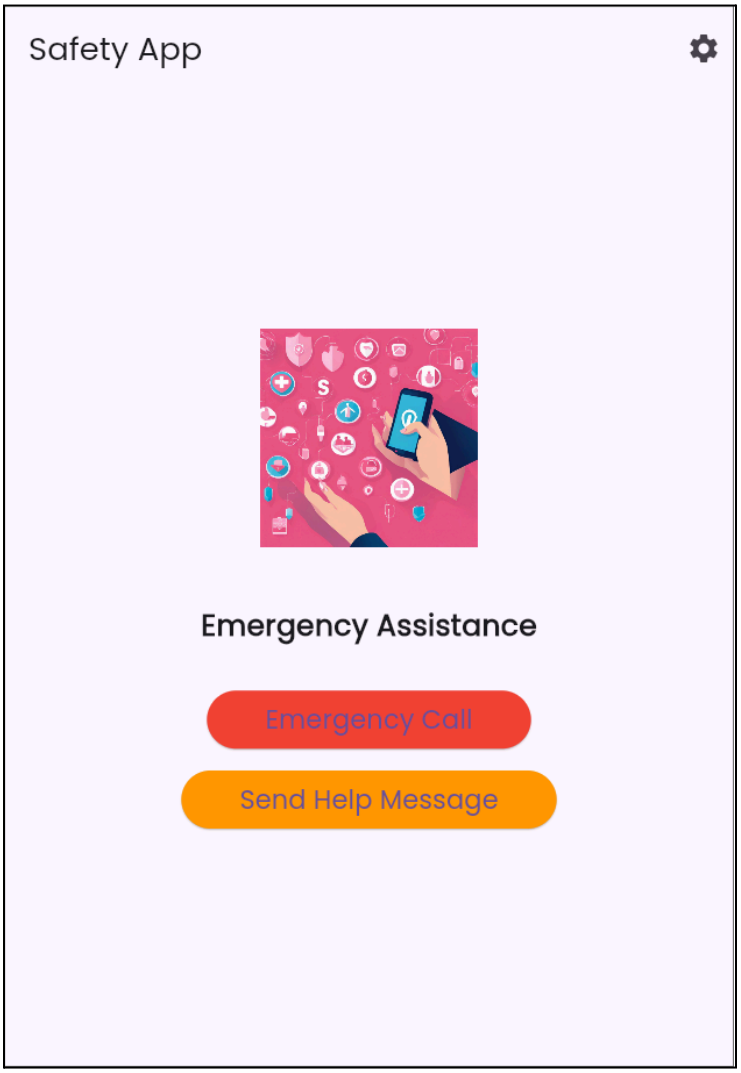
Code :

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

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

class WomanSafetyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Woman Safety App',
      theme: ThemeData(
        primarySwatch: Colors.pink,
        fontFamily: 'Poppins', // Custom font
      ),
      home: HomeScreen(),
    );
  }
}

class HomeScreen extends StatelessWidget {
  void sendHelpMessage() {
    print("Help message sent!");
  }

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

```
    return Scaffold(
      appBar: AppBar(
        title: Text('Safety App'),
        actions: [
          IconButton(
            icon: Icon(Icons.settings),
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => SettingsScreen()),
              );
            },
          ),
        ],
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Image.asset('assets/safety_image.png', width: 150), // Image
            SizedBox(height: 20),
            // Icon(Icons.security, size: 80, color: Colors.pink), // Icon
            SizedBox(height: 20),
            Text(
              "Emergency Assistance",
              style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 30),
            ElevatedButton(
              onPressed: () {
                print("Calling emergency number...");
              },
              style: ElevatedButton.styleFrom(
                backgroundColor: Colors.red,
                padding: EdgeInsets.symmetric(horizontal: 40, vertical: 15),
              ),
              child: Text("Emergency Call", style: TextStyle(fontSize: 18)),
            ),
            SizedBox(height: 15),
            ElevatedButton(
              onPressed: sendHelpMessage,
              style: ElevatedButton.styleFrom(
                backgroundColor: Colors.orange,
                padding: EdgeInsets.symmetric(horizontal: 40, vertical: 15),
              ),
              child: Text("Send Help Message", style: TextStyle(fontSize: 18)),
            ),
          ],
        ),
```

```
      ),
    );
  }
}

class SettingsScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Settings')),
      body: Center(
        child: Text(
          'Settings Page',
          style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
        ),
      ),
    );
  }
}
```

Screenshot:



Conclusion

In this experiment, we successfully added icons, images, and custom fonts to our Woman Safety App, enhancing its visual appeal and usability. Initially, we faced issues like missing asset configuration and font not applying, but we resolved them by correctly updating the pubspec.yaml file and ensuring assets were placed in the right directories.

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

Theory:

A form in Flutter is a collection of input fields that allow users to enter and submit data. The Form widget provides a way to validate user input before processing it. The TextFormField widget is commonly used within a form because it supports validation, unlike TextField. A GlobalKey is assigned to the form to track its state and check if all inputs are valid before submission.

Implementation in Our Code

In our Woman Safety App, we created a login form using the Form widget. The form includes:

- Email Input Field – Accepts an email and ensures it follows the correct format.
- Password Input Field – Accepts a password and checks that it has at least six characters.
- Validation – If any field is empty or incorrect, an error message is displayed.
- Login Button – Validates input before navigating to the HomeScreen.

Code :

login.dart

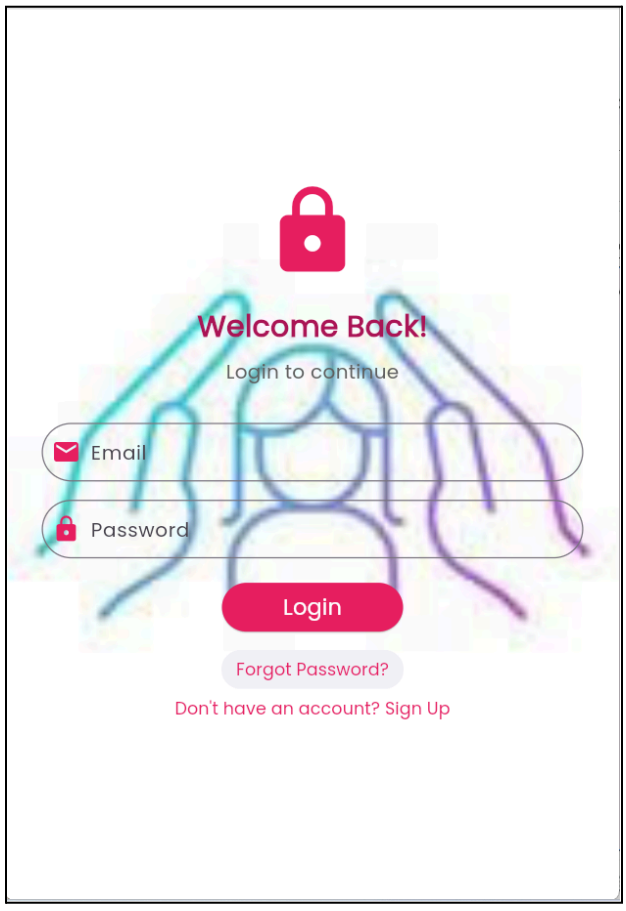
```
import 'package:flutter/material.dart';
import 'main.dart'; // Import main.dart to navigate to HomeScreen

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

class _LoginPageState extends State<LoginPage> {
  final _formKey = GlobalKey<FormState>(); // Key to track form state
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();

  void _login() {
    if (_formKey.currentState!.validate()) {
      // If the form is valid, navigate to HomeScreen
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => HomeScreen()),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        children: [
          Positioned.fill(
```

```
            child: Image.asset(
              'assets/bg.png', // Background image
              fit: BoxFit.cover,
            ),
          ),
          Center(
            child: Padding(
              padding: EdgeInsets.symmetric(horizontal: 30),
              child: Form(
                key: _formKey, // Assign form key
                child: Column(
                  mainAxisSize: MainAxisSize.min,
                  children: [
                    Icon(Icons.lock, size: 80, color: Colors.pink),
                    SizedBox(height: 20),
                    Text(
                      "Welcome Back!",
                      style: TextStyle(
                        fontSize: 24,
                        fontWeight: FontWeight.bold,
                        color: Colors.pink.shade800,
                      ),
                    ),
                    SizedBox(height: 10),
                    Text(
                      "Login to continue",
                      style: TextStyle(fontSize: 16, color: Colors.grey[700]),
                    ),
                    SizedBox(height: 30),

                    // Email Input
                    TextFormField(
                      controller: _emailController,
                      decoration: InputDecoration(
                        labelText: "Email",
                        prefixIcon: Icon(Icons.email, color: Colors.pink),
                        border: OutlineInputBorder(
                          borderRadius: BorderRadius.circular(30),
                        ),
                      ),
                      validator: (value) {
                        if (value == null || value.isEmpty) {
                          return "Please enter your email";
                        } else if
(!RegExp(r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$")
                            .hasMatch(value)) {
                          return "Enter a valid email";
                        }
                        return null;
```

```
                    },
                  ),
                  SizedBox(height: 15),

                  // Password Input
                  TextFormField(
                    controller: _passwordController,
                    obscureText: true,
                    decoration: InputDecoration(
                      labelText: "Password",
                      prefixIcon: Icon(Icons.lock, color: Colors.pink),
                      border: OutlineInputBorder(
                        borderRadius: BorderRadius.circular(30),
                      ),
                    ),
                    validator: (value) {
                      if (value == null || value.isEmpty) {
                        return "Please enter your password";
                      } else if (value.length < 6) {
                        return "Password must be at least 6 characters";
                      }
                      return null;
                    },
                  ),
                  SizedBox(height: 20),

                  // Login Button
                  ElevatedButton(
                    onPressed: _login,
                    style: ElevatedButton.styleFrom(
                      backgroundColor: Colors.pink,
                      padding: EdgeInsets.symmetric(horizontal: 50, vertical:
15),

                      shape: RoundedRectangleBorder(
                        borderRadius: BorderRadius.circular(30),
                      ),
                    ),
                    child: Text(
                      "Login",
                      style: TextStyle(fontSize: 18, color: Colors.white),
                    ),
                  ),

                  // Sign Up & Forgot Password
                  SizedBox(height: 15),
                  TextButton(
                    onPressed: () {
                      print("Forgot Password?");
                    },
```

```
                    child: Text("Forgot Password?", style: TextStyle(color:
Colors.pink)),
                  ),
                  TextButton(
                    onPressed: () {
                      print("Go to Sign Up");
                    },
                    child: Text("Don't have an account? Sign Up", style:
TextStyle(color: Colors.pink)),
                  ),
                ],
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```

Screenshot:



Conclusion: In this experiment, we successfully implemented an interactive login form using the Form widget with validation for email and password. Initially, we faced issues with validating user input and preventing navigation on invalid credentials, but we resolved them by using TextFormField, adding validation logic, and checking the form state before allowing login.

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

Theory:

In Flutter, navigation allows users to move between different screens (pages). There are two types of navigation: direct navigation (using Navigator.push) and named routing (using Navigator.pushNamed). Named routes make navigation cleaner and easier to manage.
Gestures in Flutter are used to detect user interactions like taps, swipes, and long presses. The GestureDetector widget helps in handling these gestures, allowing us to add custom interactions.

Implementation in Our Code

In our Woman Safety App, we implemented:

- Named Routes – For smooth navigation between Login, Home, and Settings pages.
- Gesture Detection – Added a double-tap gesture on the home screen to send a help message.
- Settings Page Features – Users can update an emergency contact, toggle dark mode, and log out.

Code :

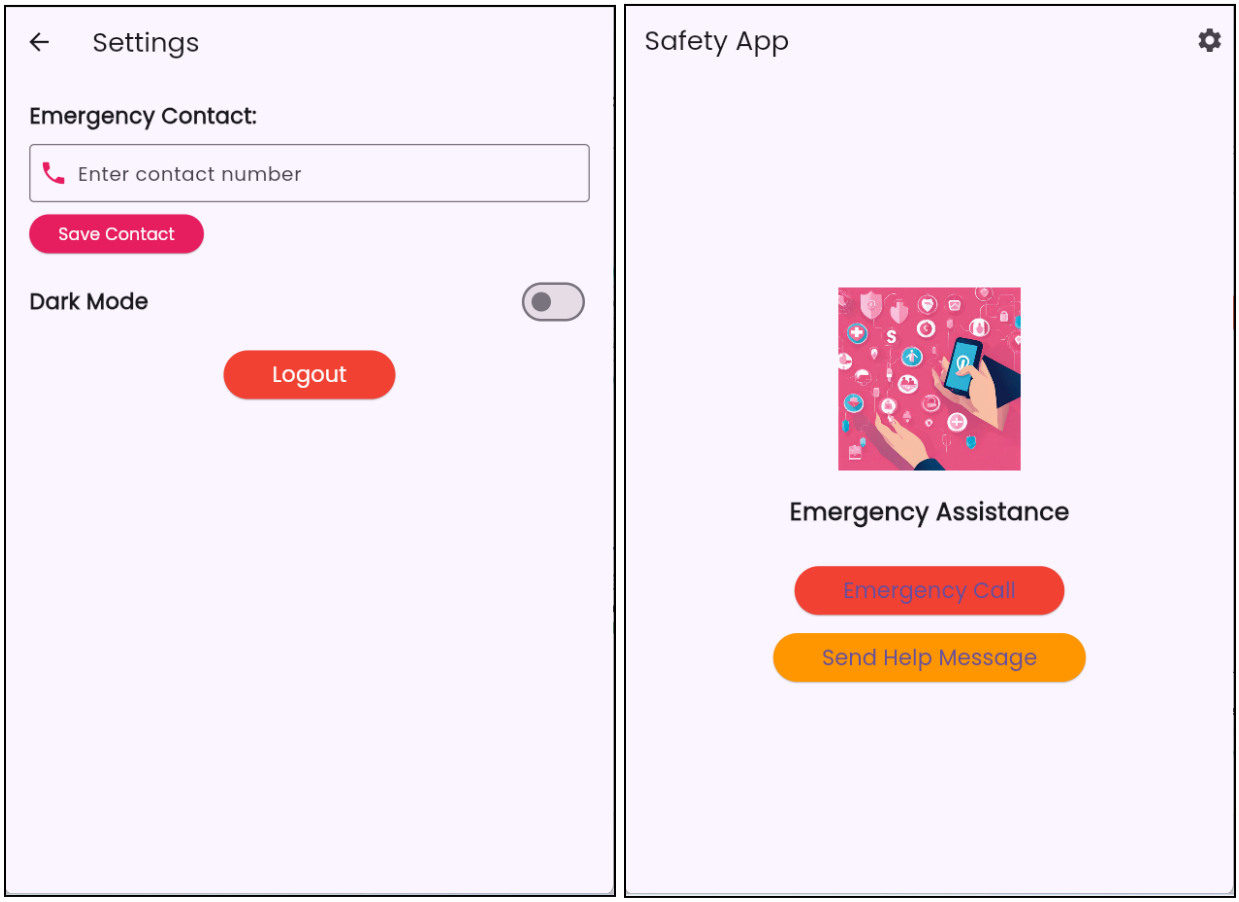
settings.dart

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

class SettingsScreen extends StatefulWidget {
  @override
  _SettingsScreenState createState() => _SettingsScreenState();
}

class _SettingsScreenState extends State<SettingsScreen> {
  bool isDarkMode = false; // Dark mode toggle
  final TextEditingController _contactController = TextEditingController();

  void _saveContact() {
    String contact = _contactController.text;
    if (contact.isNotEmpty) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Emergency contact saved: $contact")),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Settings')),
      body: Padding(
        padding: EdgeInsets.all(20),
        child: Column(
```

```
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            // Change Emergency Contact
            Text("Emergency Contact:", style: TextStyle(fontSize: 18,
fontWeight: FontWeight.bold)),
            SizedBox(height: 10),
            TextField(
              controller: _contactController,
              keyboardType: TextInputType.phone,
              decoration: InputDecoration(
                labelText: "Enter contact number",
                border: OutlineInputBorder(),
                prefixIcon: Icon(Icons.phone, color: Colors.pink),
              ),
            ),
            SizedBox(height: 10),
            ElevatedButton(
              onPressed: _saveContact,
              style: ElevatedButton.styleFrom(backgroundColor: Colors.pink),
              child: Text("Save Contact", style: TextStyle(color:
Colors.white)),
            ),
            SizedBox(height: 20),

            // Dark Mode Toggle
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                Text("Dark Mode", style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
                Switch(
                  value: isDarkMode,
                  onChanged: (value) {
                    setState(() {
                      isDarkMode = value;
                      print("Dark Mode: $isDarkMode");
                    });
                  },
                ),
              ],
            ),
            SizedBox(height: 20),

            // Logout Button
            Center(
              child: ElevatedButton(
                onPressed: () {
                  Navigator.pushReplacementNamed(context, '/'); // Go to Login
Page
```

```
                },
                style: ElevatedButton.styleFrom(
                  backgroundColor: Colors.red,
                  padding: EdgeInsets.symmetric(horizontal: 40, vertical: 15),
                ),
                child: Text("Logout", style: TextStyle(fontSize: 18, color:
Colors.white)),
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

Screenshot:



Conclusion:

In this experiment, we successfully implemented navigation using named routes and gesture detection to enhance user interaction. Initially, we faced issues with incorrect route navigation and gesture recognition, but we resolved them by properly defining routes in MaterialApp and using GestureDetector correctly.

Aim: To Connect Flutter UI with fireBase database.

Theory:

Firebase is a cloud-based backend platform that provides a wide range of services, including authentication, real-time databases, cloud storage, and hosting. In Flutter, Firebase allows developers to integrate backend functionalities without setting up complex servers. The Firebase Authentication module enables user authentication using methods like email-password login, Google sign-in, and password reset.

Implementation in Our Code

1. User Authentication with Firebase
- Implemented sign-up (signUp) and login (signIn) functionalities using Firebase Authentication.
- Used FirebaseAuth API to create a user account with an email and password.
- Implemented a password reset feature to allow users to recover their accounts.

2. Secure User Authentication
- Used Firebase's secure token-based authentication to validate users.
- Stored user session details using Firebase's current user session tracking.

3. Sign-Out Functionality
- Implemented a sign-out method that logs users out of Firebase.

Code :

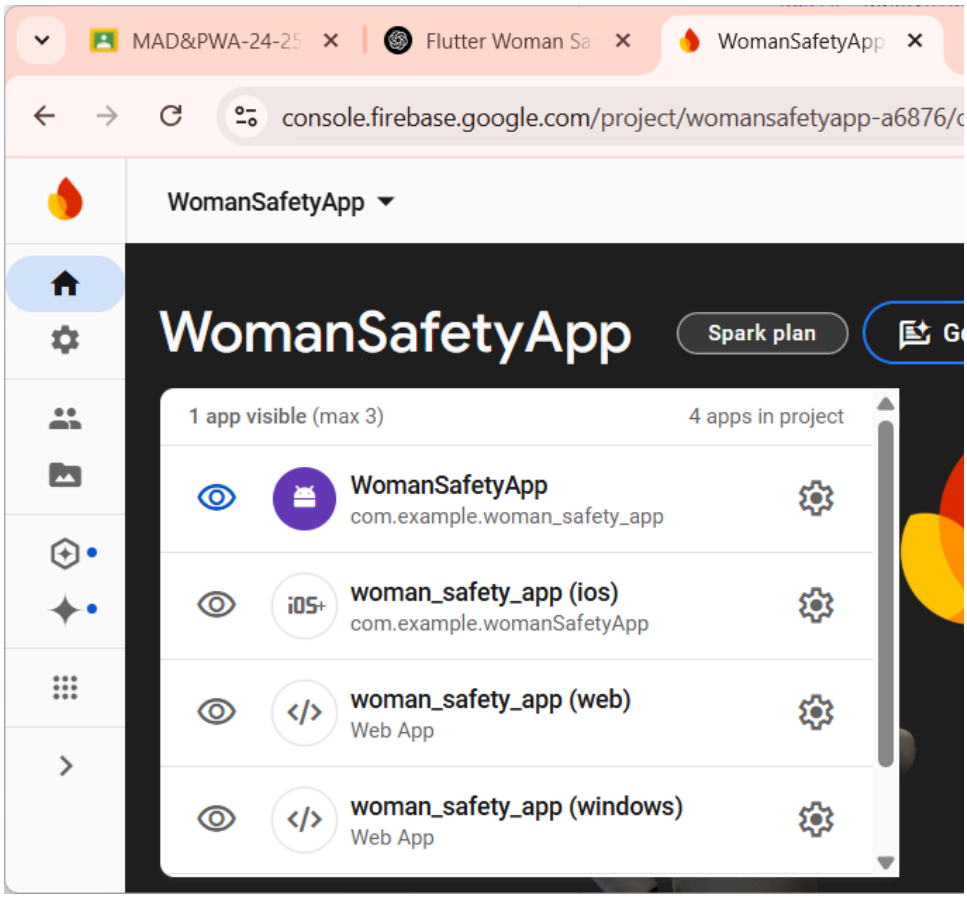
auth_service.dart

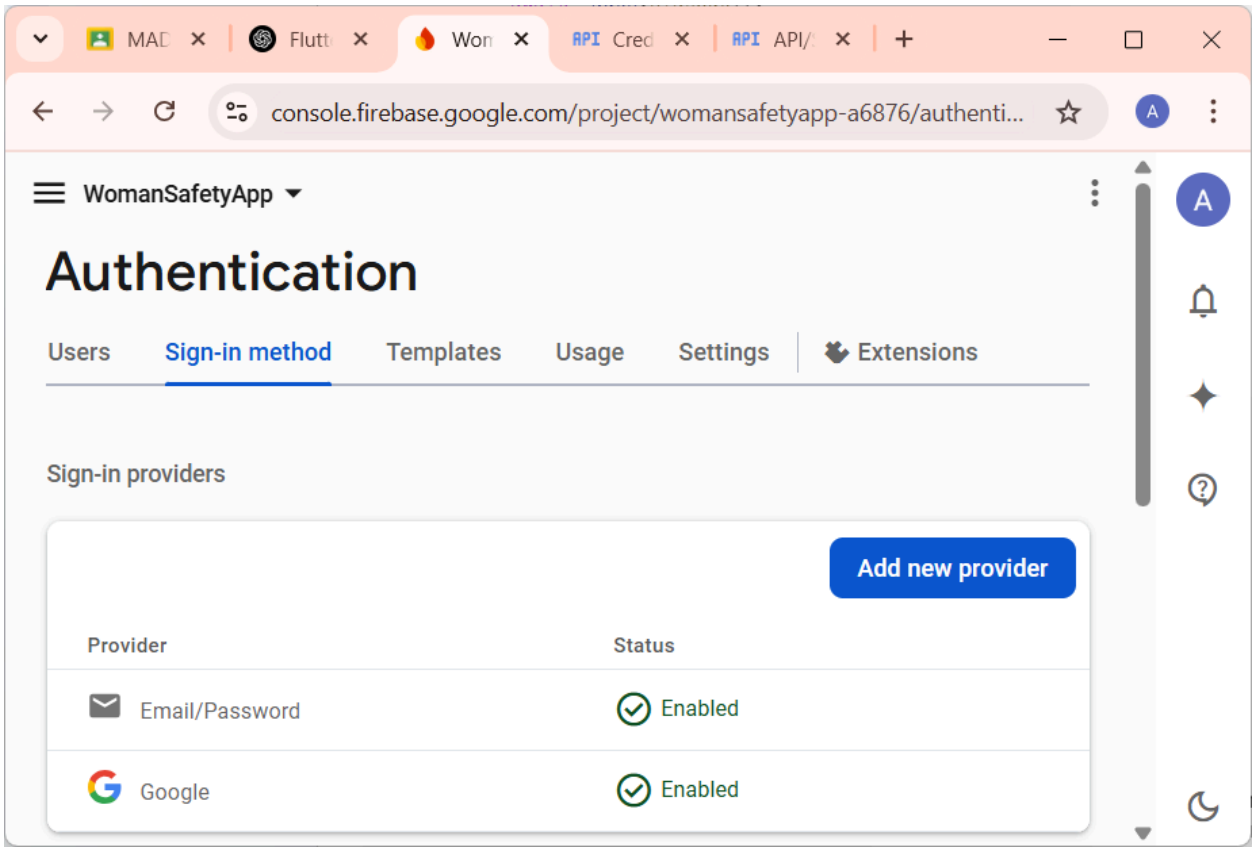
```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:google_sign_in/google_sign_in.dart';

class AuthService {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final GoogleSignIn _googleSignIn = GoogleSignIn(
    clientId:
"369364366716-52nj14de3gasogbs56f7i9js8tpvor2g.apps.googleusercontent.com",
  );

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

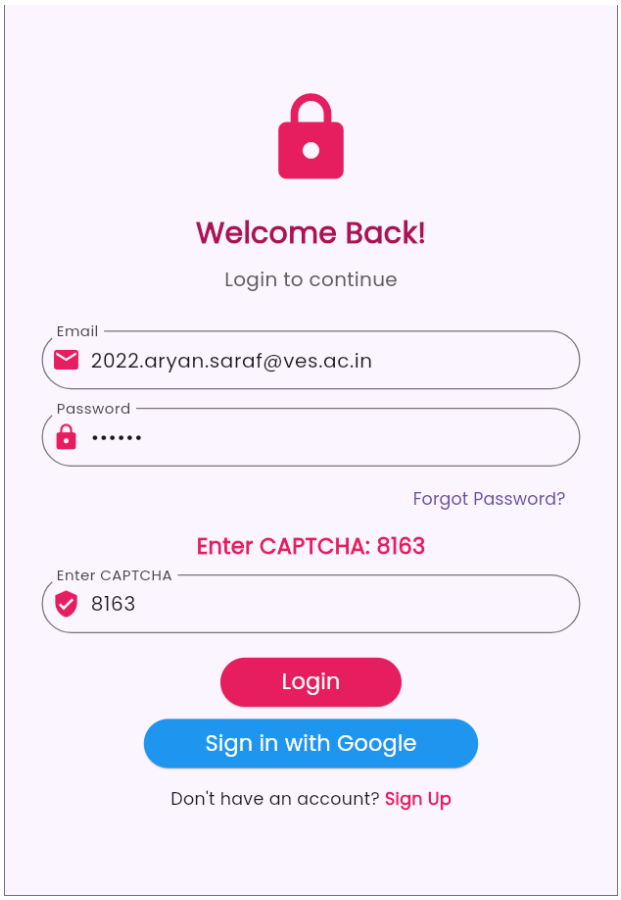
```
      return null;
    }
  }

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

  // Get current user
  User? getCurrentUser() {
    return _auth.currentUser;
  }

  // Reset Password
  Future<bool> resetPassword(String email) async {
    try {
      await _auth.sendPasswordResetEmail(email: email);
      return true; // Email sent successfully
    } catch (e) {
      print("Password Reset Error: $e");
      return false; // Failed to send email
    }
  }

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

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

      UserCredential userCredential = await
_auth.signInWithCredential(credential);
```

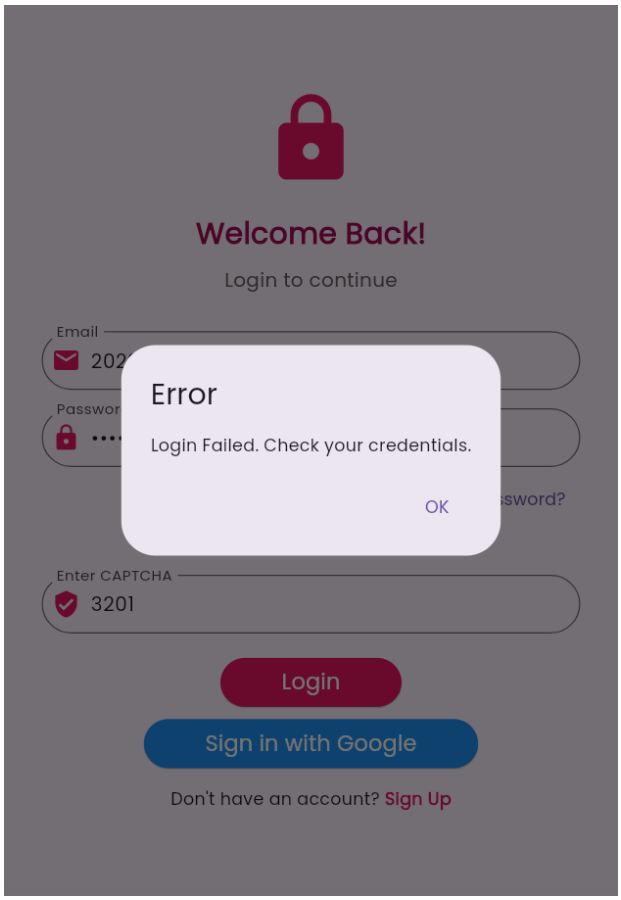
```
      return userCredential.user;
    } catch (e) {
      print("Google Sign-In Error: $e");
      return null;
    }
  }

  // Sign Out (Google & Email)
  Future<void> signOut() async {
    await _auth.signOut();
    await _googleSignIn.signOut();
  }
}
```

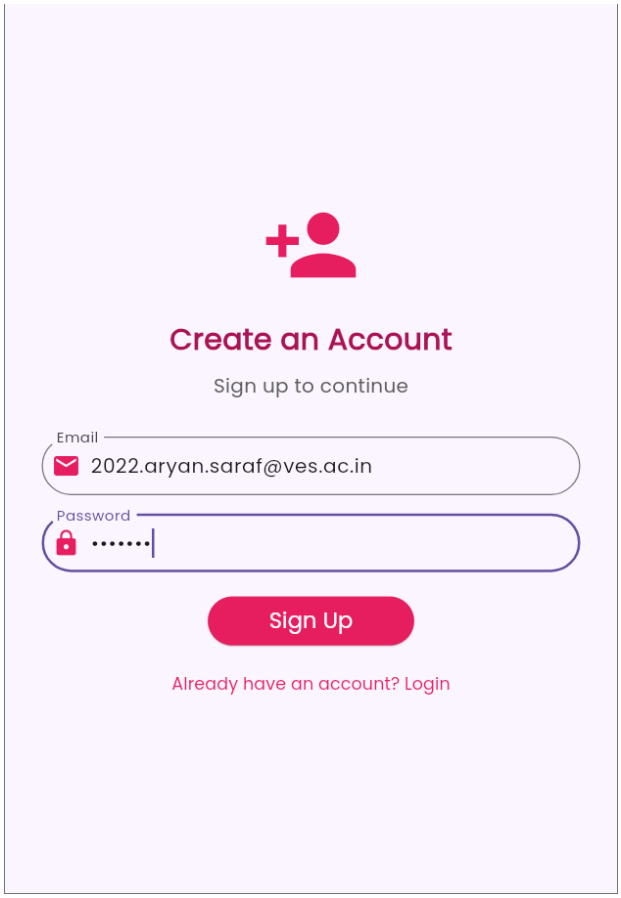
Screenshot:

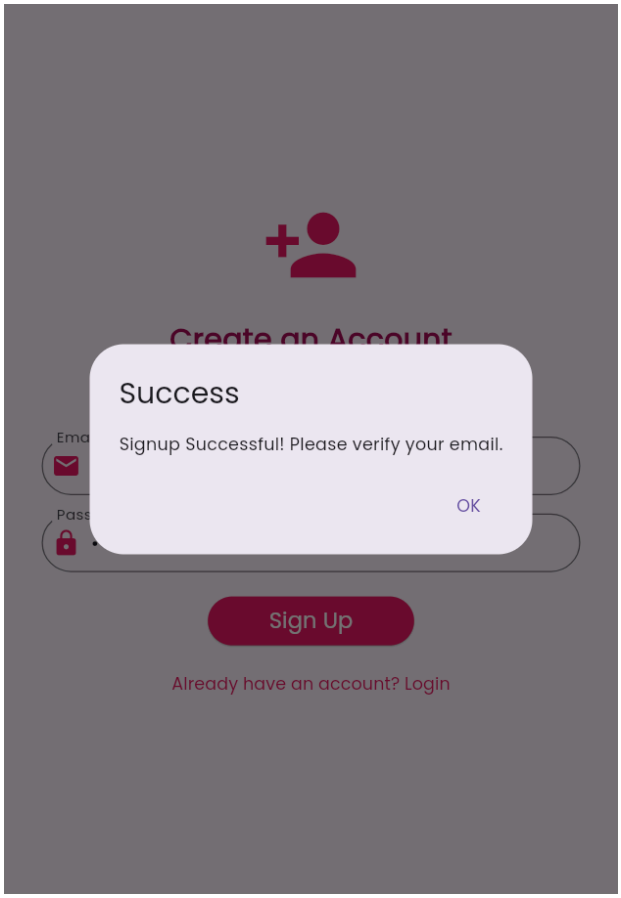


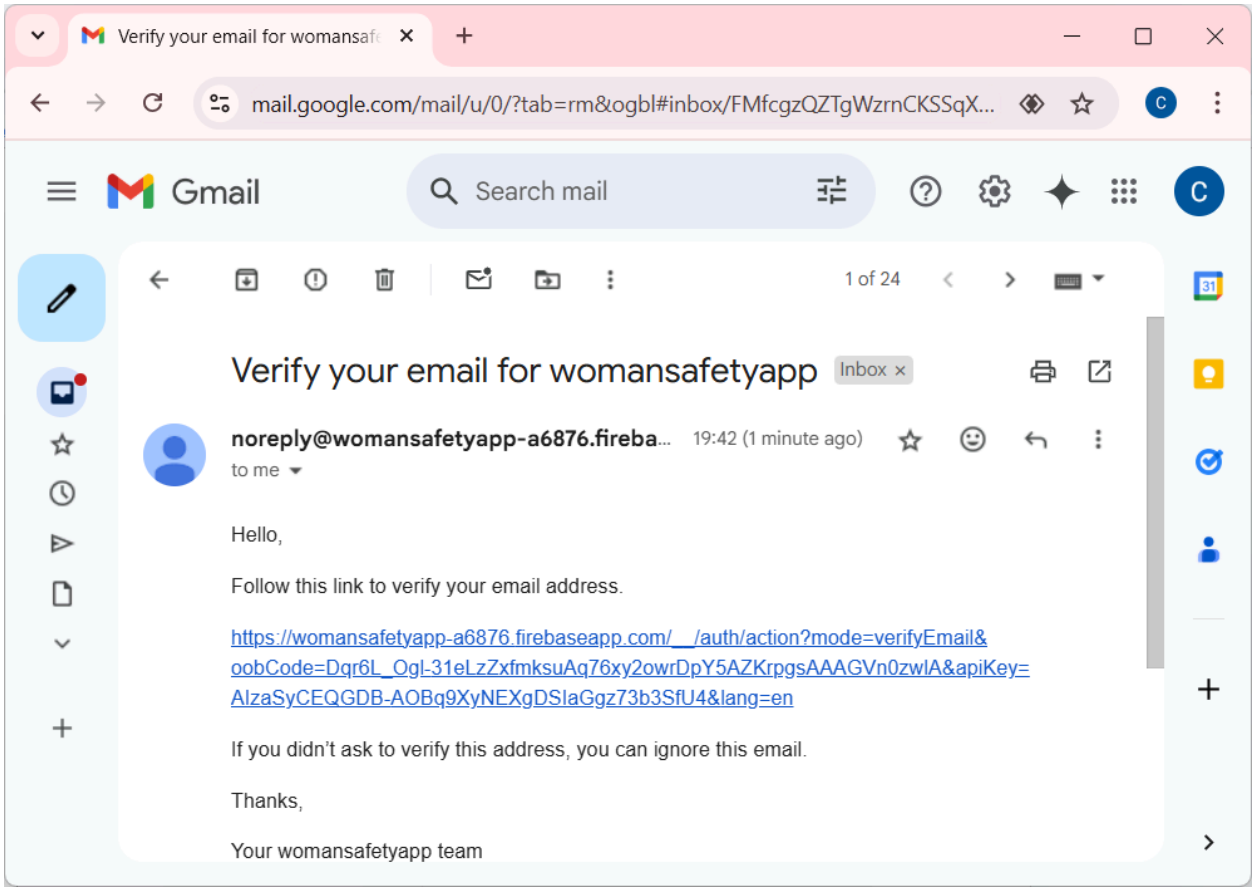

console.firebase.google.com/project/womansafetyapp-a6876/authenti...
WomanSafetyApp
Authentication
Users
Sign-in method
Templates
Usage
Settings
Extensions
Sign-in providers
Add new provider
Provider
Status
Email/Password
Enabled
Google
Enabled

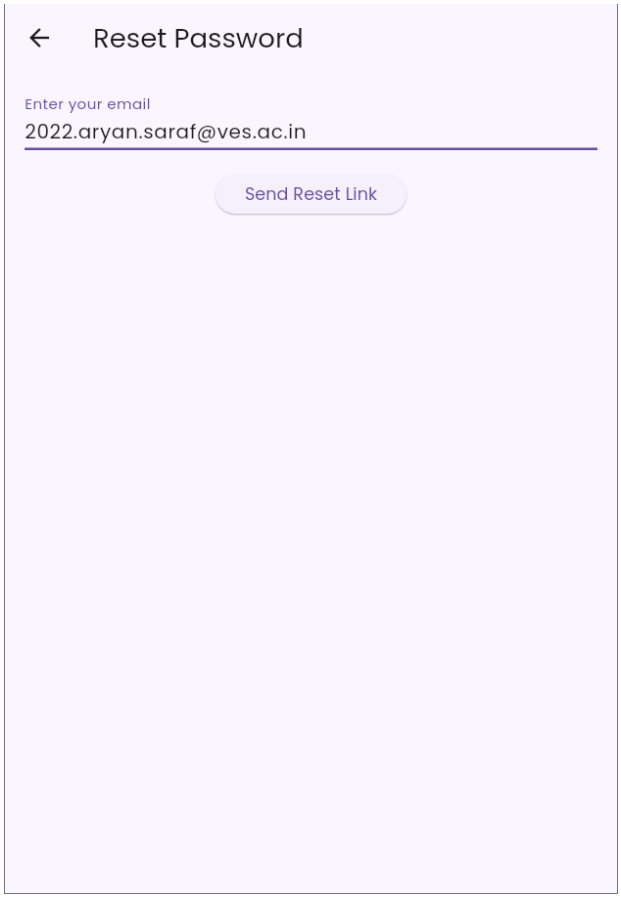

Welcome Back!
Login to continue
Email
2022.aryan.saraf@ves.ac.in
Password
Forgot Password?
Enter CAPTCHA: 8163
Enter CAPTCHA
8163
Login
Sign in with Google
Don't have an account? Sign Up

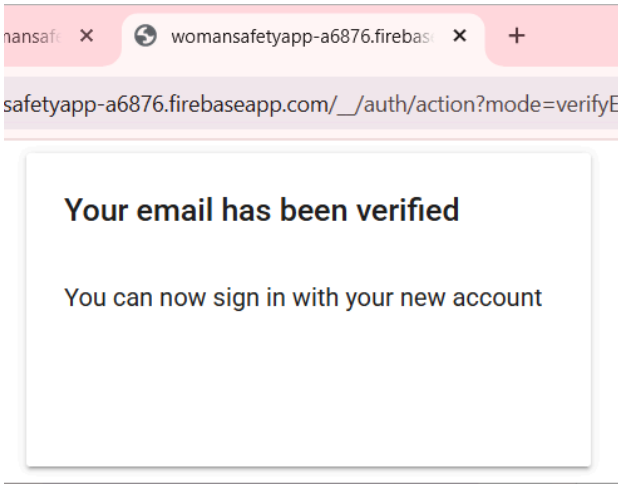

Welcome Back!
Login to continue
Email
Password
Error
Login Failed. Check your credentials.
OK
Enter CAPTCHA
3201
Login
Sign in with Google
Don't have an account? Sign Up

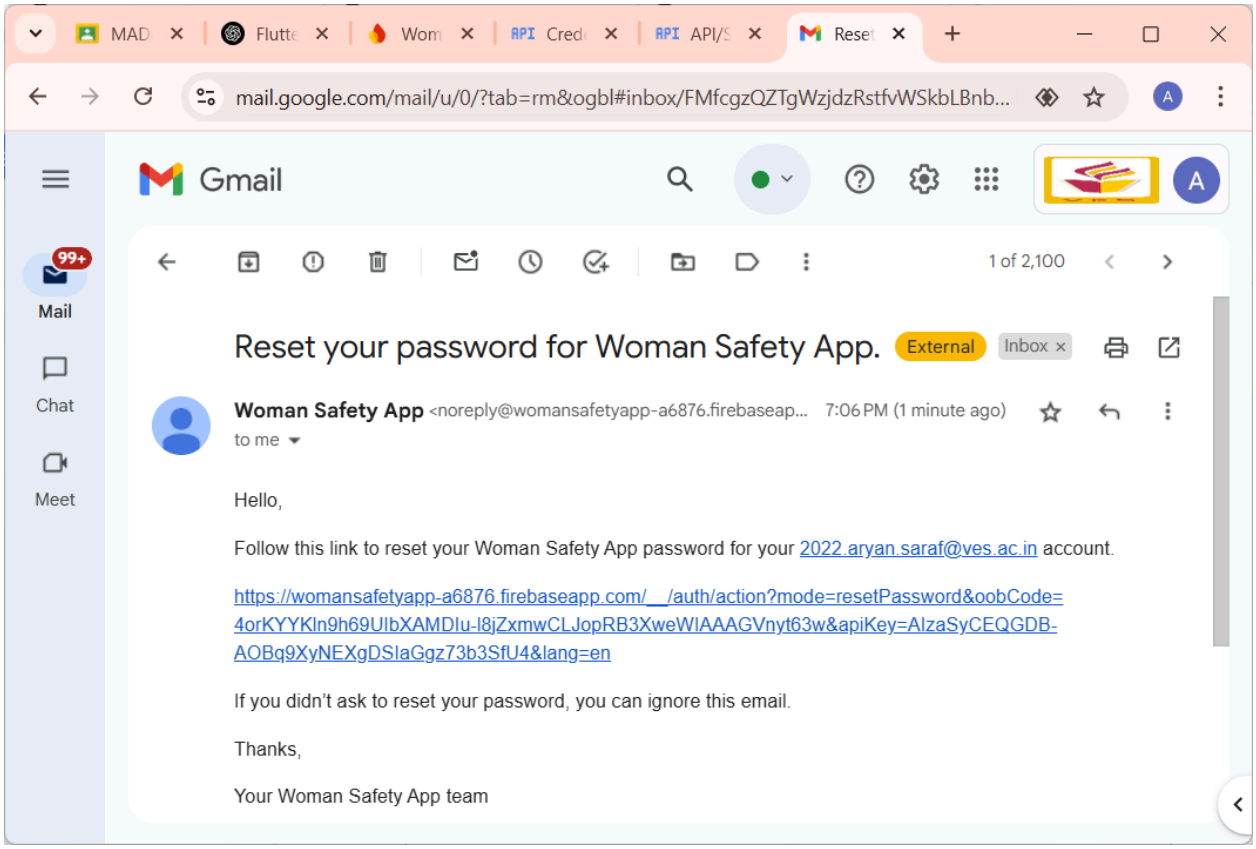

Create an Account
Sign up to continue
Email
2022.aryan.saraf@ves.ac.in
Password
Sign Up
Already have an account? Login

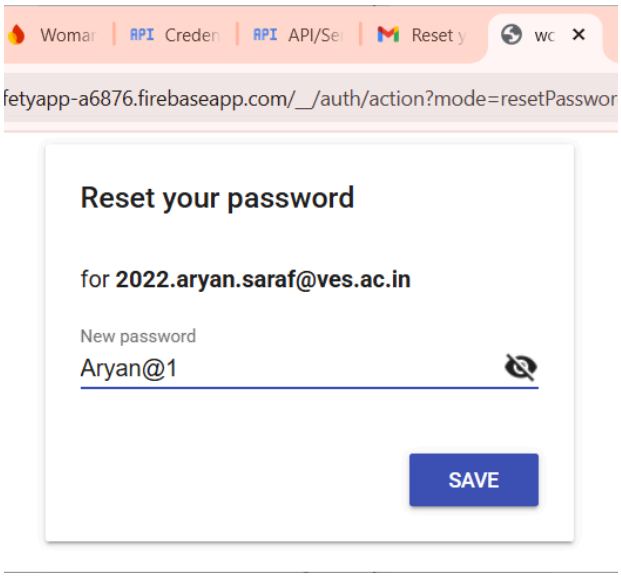

Success
Signup Successful! Please verify your email.
OK
Sign Up
Already have an account? Login

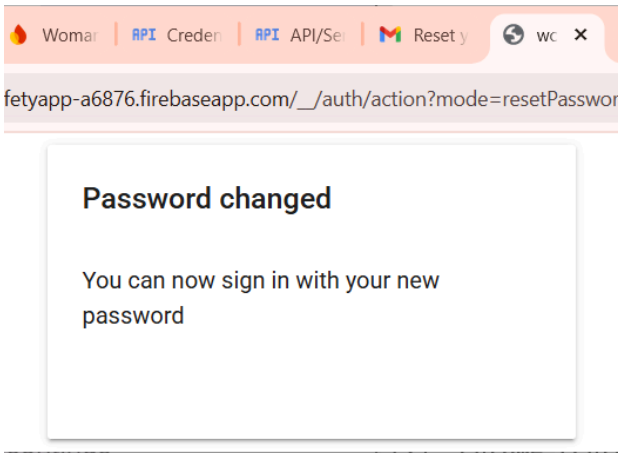

Verify your email for womansaf
mail.google.com/mail/u/0/?tab=rm&ogbl#inbox/FMfcgzQZTgWzrnCKSSqX...
Gmail
Search mail
1 of 24
Verify your email for womansafetyapp
Inbox
noreply@womansafetyapp-a6876.fireba...
19:42 (1 minute ago)
to me
Hello,
Follow this link to verify your email address.
https://womansafetyapp-a6876.firebaseapp.com/__/auth/action?mode=verifyEmail&oobCode=Dqr6L_Ogl-31eLzZxfmksuAq76xy2owrDpY5AZKrpgsAAAGVn0zwlA&apiKey=AIzaSyCEQGDB-AOBq9XyNEXgDSlaGgz73b3SfU4&lang=en
If you didn't ask to verify this address, you can ignore this email.
Thanks,
Your womansafetyapp team

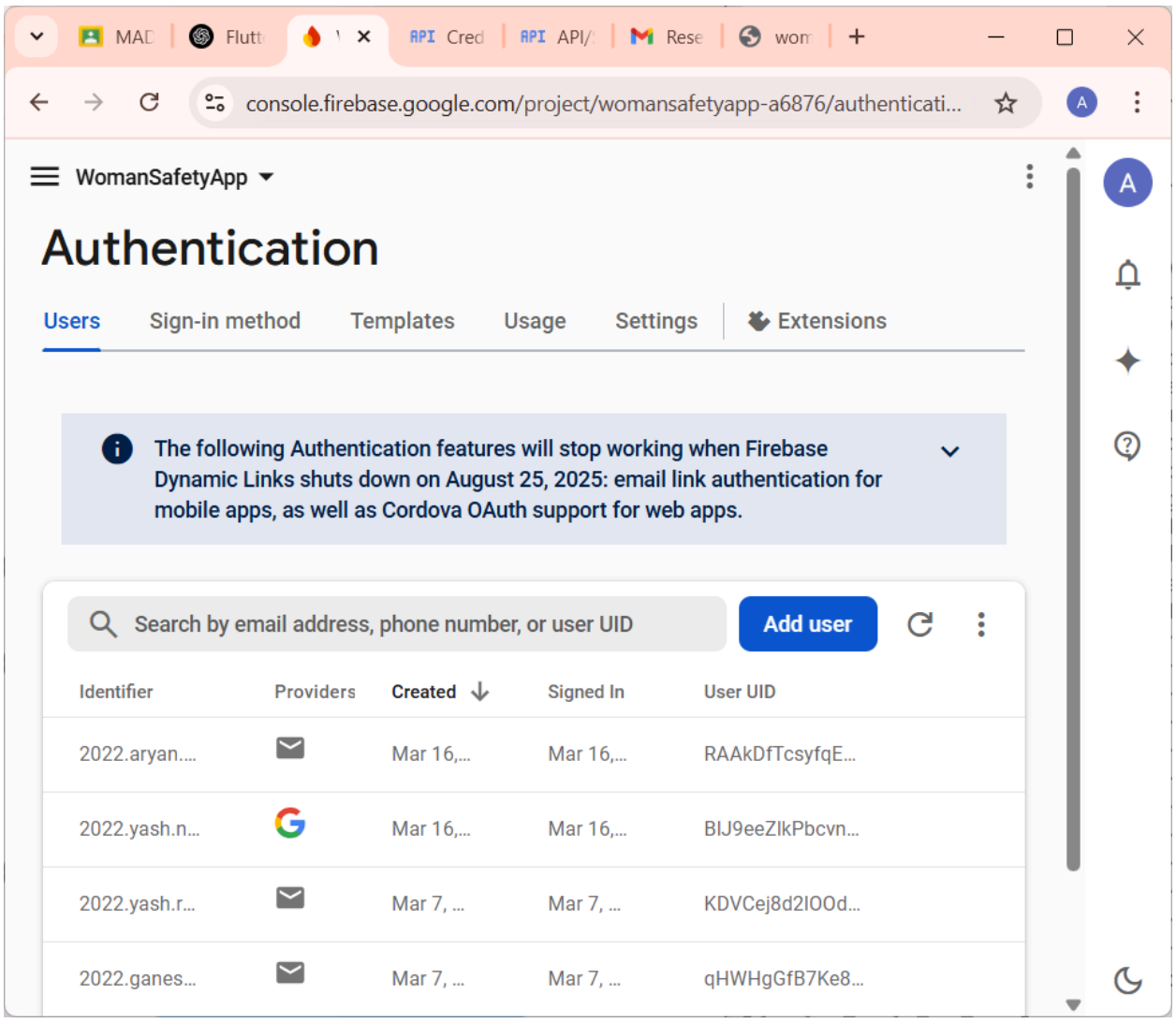

Reset Password
Enter your email
2022.aryan.saraf@ves.ac.in
Send Reset Link

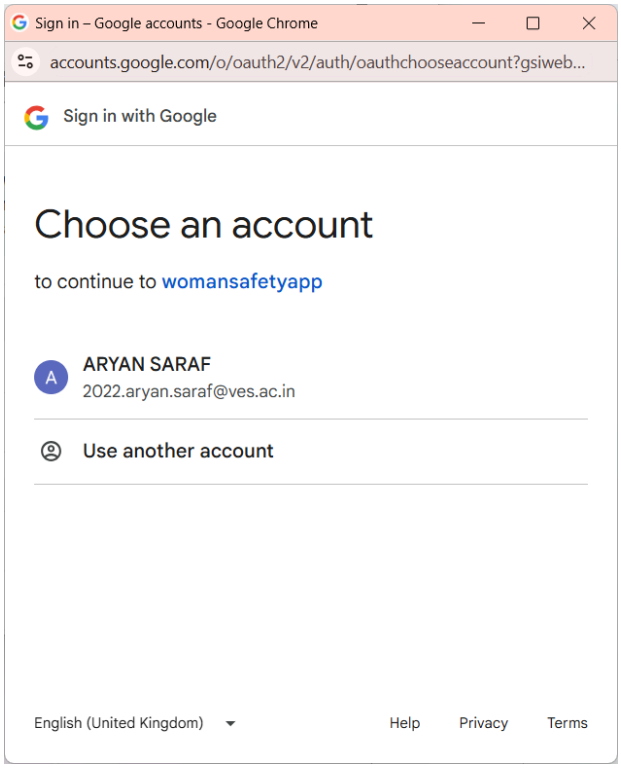

womansafetyapp-a6876.firebas
safetyapp-a6876.firebaseapp.com/__/auth/action?mode=verifyE
Your email has been verified
You can now sign in with your new account

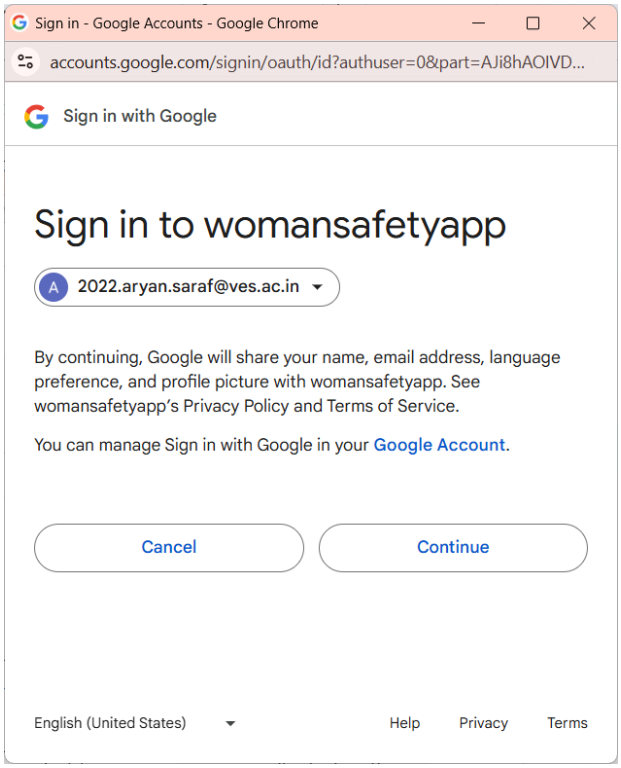

mail.google.com/mail/u/0/?tab=rm&ogbl#inbox/FMfcgzQZTgWzjdzRstfvWSkbLBnb...
Gmail
Mail
Chat
Meet
1 of 2,100
Reset your password for Woman Safety App.
External
Inbox ×
Woman Safety App <noreply@womansafetyapp-a6876.firebaseap... 7:06 PM (1 minute ago)
to me
Hello,
Follow this link to reset your Woman Safety App password for your 2022.aryan.saraf@ves.ac.in account.
https://womansafetyapp-a6876.firebaseapp.com/__/auth/action?mode=resetPassword&oobCode=4orKYYKln9h69UlbXAMDlu-l8jZxmwCLJopRB3XweWIAAAGVnyt63w&apiKey=AIzaSyCEQGDB-AOBq9XyNEXgDSlaGgz73b3SfU4&lang=en
If you didn't ask to reset your password, you can ignore this email.
Thanks,
Your Woman Safety App team

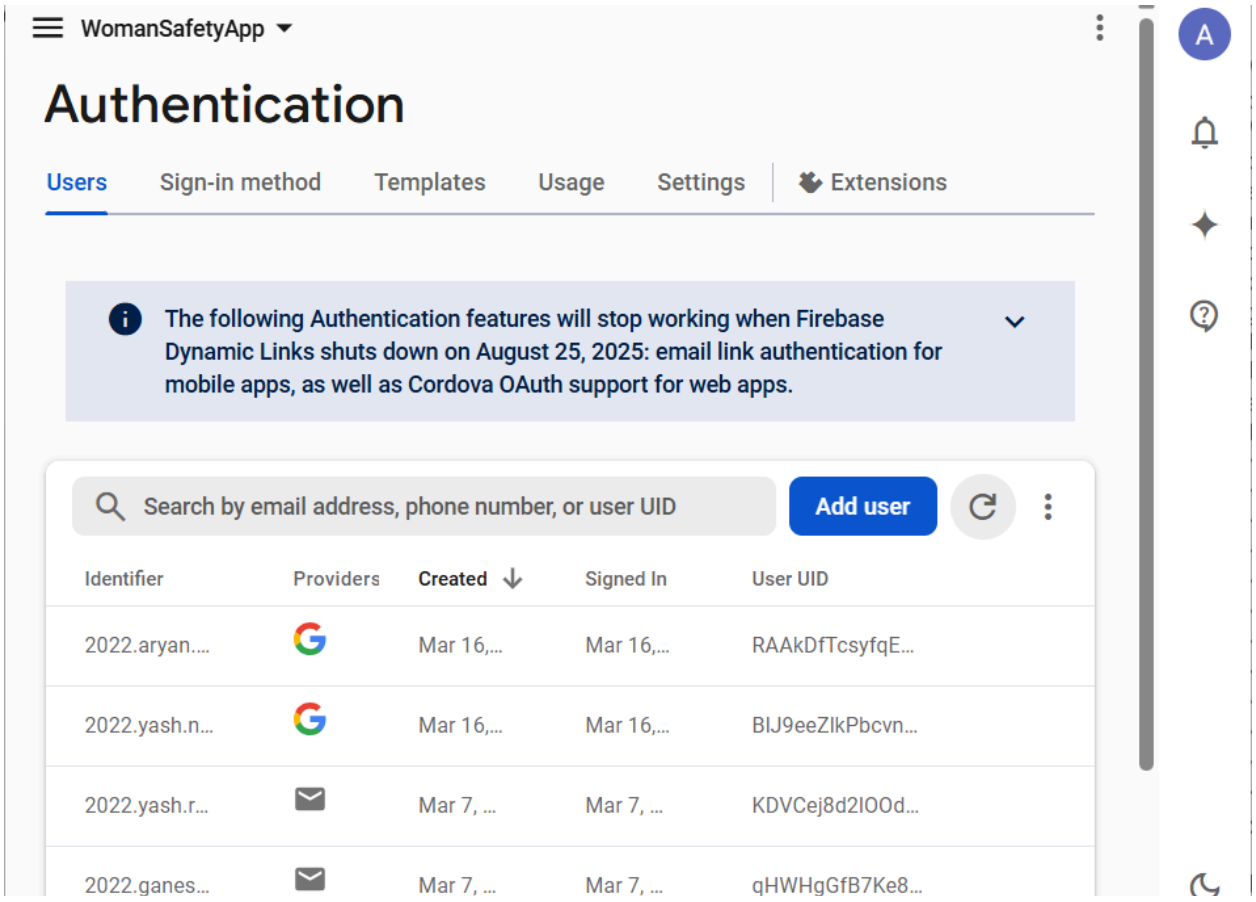

fetyapp-a6876.firebaseapp.com/__/auth/action?mode=resetPasswor
Reset your password
for 2022.aryan.saraf@ves.ac.in
New password
Aryan@1
SAVE


fetyapp-a6876.firebaseapp.com/__/auth/action?mode=resetPasswor
Password changed
You can now sign in with your new password


console.firebase.google.com/project/womansafetyapp-a6876/authenticati...
WomanSafetyApp
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
2022.aryan....
Mar 16,...
Mar 16,...
RAAkDfTcsyfqE...
2022.yash.n...
Mar 16,...
Mar 16,...
BIJ9eeZlkPbcvn...
2022.yash.r...
Mar 7, ...
Mar 7, ...
KDVCej8d2lOOd...
2022.ganes...
Mar 7, ...
Mar 7, ...
qHWHgGfB7Ke8...







Conclusion:

In this experiment, we successfully integrated Firebase Authentication, implementing email-password login, and password reset while ensuring secure user authentication. During development, we faced issues like OAuth client errors, redirect URI mismatches, and Firebase configuration mismatches, which we resolved by correctly setting up the Google Cloud OAuth client, updating redirect URIs, and verifying Firebase authentication settings.

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

Theory:

A Progressive Web App (PWA) is a web application that combines the best features of both web and native apps. It offers a reliable, fast, and engaging user experience by allowing users to install the app directly on their device's home screen without visiting an app store.

Web App Manifest

The Web App Manifest is a JSON file that contains metadata about the application, such as its name, icons, theme color, display mode, and start URL. This file enables browsers to recognize the app as installable, making it possible to add the app to the home screen.

Key Properties in the Manifest

1. name: The full name of the application displayed when the app is launched.
2. short_name: A shorter version of the app name used on the home screen.
3. start_url: The entry point of the app when launched from the home screen.
4. display: Defines how the app appears (e.g., standalone makes it look like a native app).
5. background_color: Sets the background color of the splash screen when the app is launched.
6. theme_color: Defines the color of the browser's address bar.
7. icons: Specifies the icons used for the app on the home screen in different sizes and formats.
8. scope: Defines the range of URLs the app controls.

Service Worker and Caching

A Service Worker is a script that runs in the background and handles caching, allowing the app to work offline.

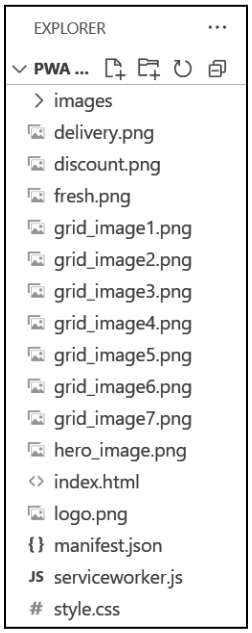
- During installation, the service worker caches essential files.
- When the app is fetched, it first checks the cache for faster loading.

Implementation in the Code

In this experiment:

- We created a manifest.json file containing metadata like the app name, icons, colors, and display settings.
- The index.html file links to the manifest file, enabling the "Add to Homescreen" feature.
- We registered a Service Worker (serviceworker.js) to cache the necessary files, ensuring offline functionality.
- This makes the web app feel more like a native app, providing a smooth and reliable experience for users.

Folder Structure:



Code:

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <meta name="theme-color" content="#007BFF">
    <title>Restaurant Website</title>
    <link rel="stylesheet" href="style.css" />
    <link rel="manifest" href="manifest.json">
    <script
      src="https://kit.fontawesome.com/7a4b62b0a4.js"
      crossorigin="anonymous"
    ></script>
  </head>
  <body>
    <nav>
      <div class="navigation container">
        <div class="logo_container">
          <img src="logo.png" alt="logo" />
        </div>
        <div class="bar_icon">
          <i class="fas fa-bars"></i>
        </div>
      </div>
```

```
    </nav>
    <!-- Hero Section -->
    <div class="container">
      <div class="hero">
        <div class="hero_image">
          <img src="hero_image.png" alt="hero image" />
        </div>
        <div class="hero_content">
          <div class="tag">50% Off on All Products</div>
          <h1>Enjoy Your Delicious Food</h1>
          <p>
            Lorem ipsum dolor sit amet, consectetur adipisicing elit.
            Voluptates, iste corporis tempore necessitatibus inventore ex?
          </p>
          <button class="explore_btn">Explore Now</button>
        </div>
      </div>
      <section class="features">
        <div class="feature">
          <img src="discount.png" alt="" />
          <div class="feature_content">
            <h3>Discount Voucher</h3>
            Lorem ipsum dolor sit, amet consectetur adipisicing elit.
          </div>
        </div>
        <div class="feature">
          <img src="fresh.png" alt="" />
          <div class="feature_content">
            <h3>Fresh Healthy Food</h3>
            Lorem ipsum dolor sit, amet consectetur adipisicing elit.
          </div>
        </div>
        <div class="feature">
          <img src="delivery.png" alt="" />
          <div class="feature_content">
            <h3>Fast Home Delivery</h3>
            Lorem ipsum dolor sit, amet consectetur adipisicing elit.
          </div>
        </div>
      </section>
      <div class="divider"></div>
      <div class="menu">
        <div class="tag">Our Menu</div>
        <h2>Explore Our Menu</h2>
```

```
<div class="grid">
  <div class="item1">
    <img
      class="grid-image"
      src="grid_image1.png"
      alt="image 1"
    />
  </div>
  <div class="item2">
    <img
      class="grid-image"
      src="grid_image2.png"
      alt="image 2"
    />
  </div>
  <div class="item3">
    <img
      class="grid-image"
      src="grid_image3.png"
      alt="image 3"
    />
  </div>
  <div class="item4">
    <img
      class="grid-image"
      src="grid_image4.png"
      alt="image 4"
    />
  </div>
  <div class="item5">
    <img
      class="grid-image"
      src="grid_image5.png"
      alt="image 5"
    />
  </div>
  <div class="item6">
    <img
      class="grid-image"
      src="grid_image6.png"
      alt="image 6"
    />
```

```
        </div>
        <div class="item7">
          <img
            class="grid-image"
            src="grid_image7.png"
            alt="image 7"
          />
        </div>
      </div>
    </div>
  </div>
  <footer>
    <div class="footer_container container">
      <div class="footer_logo">
        <img src="logo.png" alt="logo" />
      </div>
      <div class="link_lists">
        <h3>Main Links</h3>
        <ul>
          <li>Order Tracking</li>
          <li>New Order</li>
          <li>Contact Us</li>
          <li>News & Blogs</li>
        </ul>
      </div>
      <div class="link_lists">
        <h3>Support</h3>
        <ul>
          <li>About Us</li>
          <li>Privacy Policy</li>
          <li>Terms & Conditons</li>
        </ul>
      </div>
      <div class="news_letter">
        <h3>Support</h3>
        <input type="email" placeholder="Enter your Email..." />
        <h3>Follow Us</h3>
        <div class="icon_container">
          <div class="icon">
            <i class="fa fa-facebook"></i>
          </div>
          <div class="icon">
            <i class="fa fa-twitter" aria-hidden="true"></i>
          </div>
```

```
            <div class="icon">
              <i class="fa fa-instagram" aria-hidden="true"></i>
            </div>
            <div class="icon">
              <i class="fa fa-youtube" aria-hidden="true"></i>
            </div>
          </div>
        </div>
      </div>
    </footer>
    <script>
        if ('serviceWorker' in navigator) {
            navigator.serviceWorker.register('serviceworker.js')
                .then(() => console.log("Service Worker Registered"))
                .catch(error => console.log("Service Worker Registration
Failed", error));
        }
    </script>
  </body>
</html>
```

manifest.json

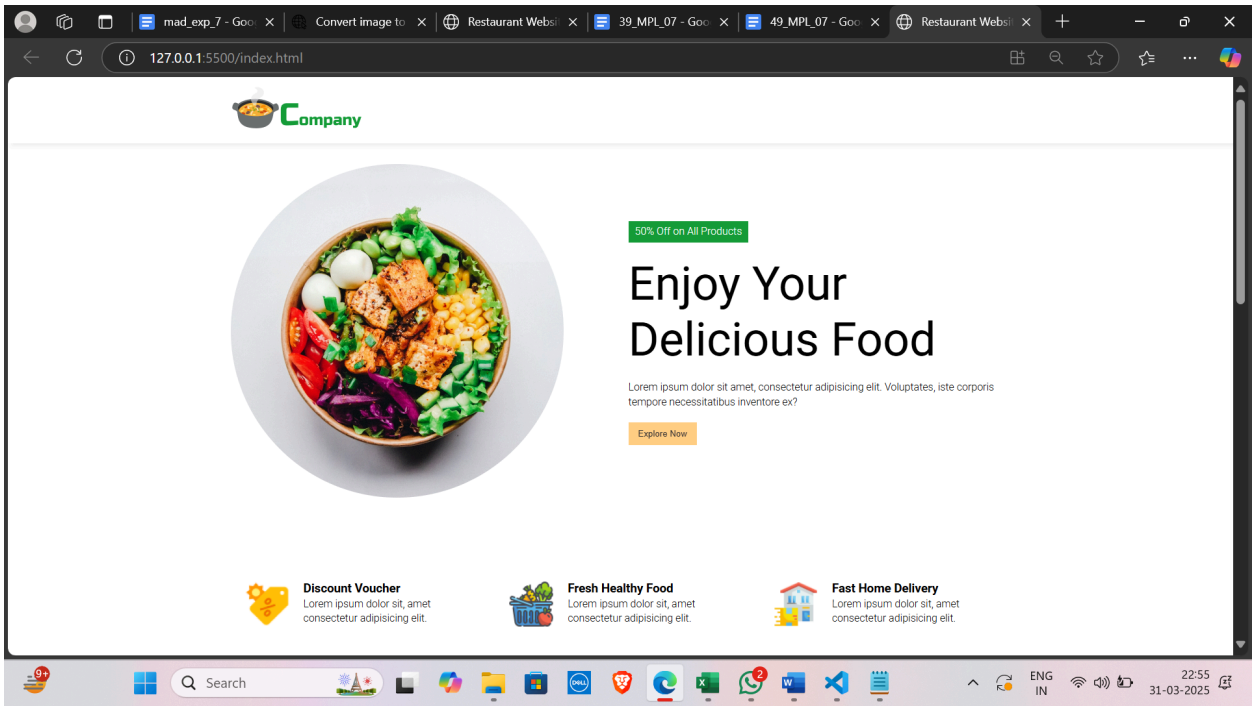
```
{
  "name": "Restaurant Website",
  "short_name": "cooking",
  "start_url": "index.html",
  "display": "standalone",
  "background_color": "#5900b3",
  "theme_color": "black",
  "scope": ".",
  "description": "Cooking, also known as cookery or professionally as the
culinary arts, is the art, science and craft of using heat to make food
more palatable, digestible, nutritious, or safe.",
  "icons": [
    {
      "src": "images/app.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "images/big.png",
      "sizes": "512x512",
      "type": "image/png"
    }
```

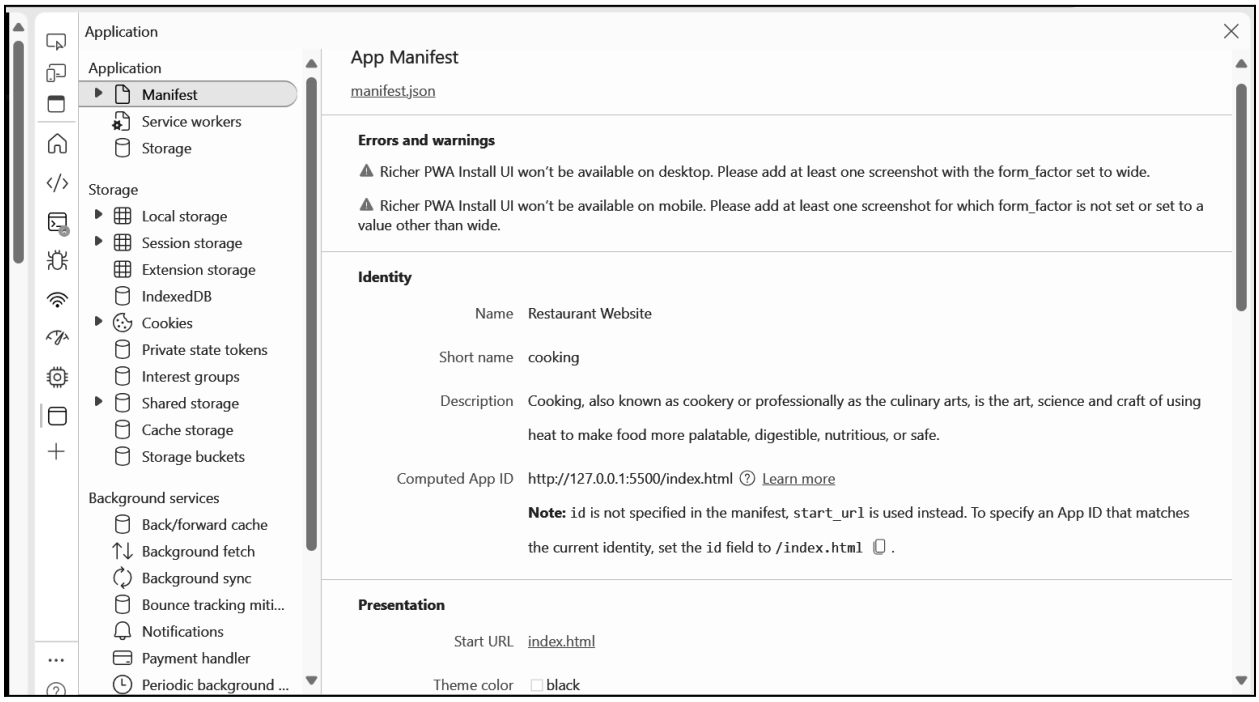
```
  ]
}
```

serviceworker.js

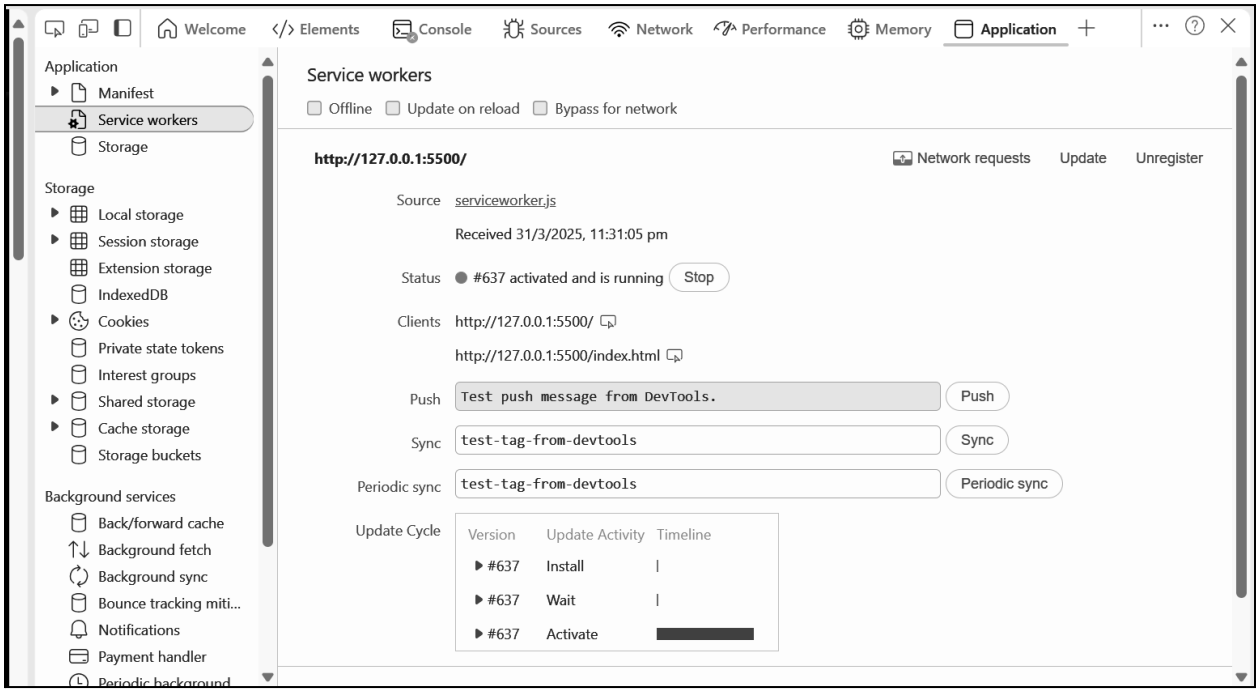
```
const CACHE_NAME = "cooking-cache-v1";
const FILES_TO_CACHE = [
    "index.html",
    "style.css",
    "manifest.json",
    "images/app.png",
    "images/big.png"
];

self.addEventListener("install", event => {
    event.waitUntil(
        caches.open(CACHE_NAME).then(cache => {
            return cache.addAll(FILES_TO_CACHE);
        })
    );
});

self.addEventListener("fetch", event => {
    event.respondWith(
        caches.match(event.request).then(response => {
            return response || fetch(event.request);
        })
    );
});
```

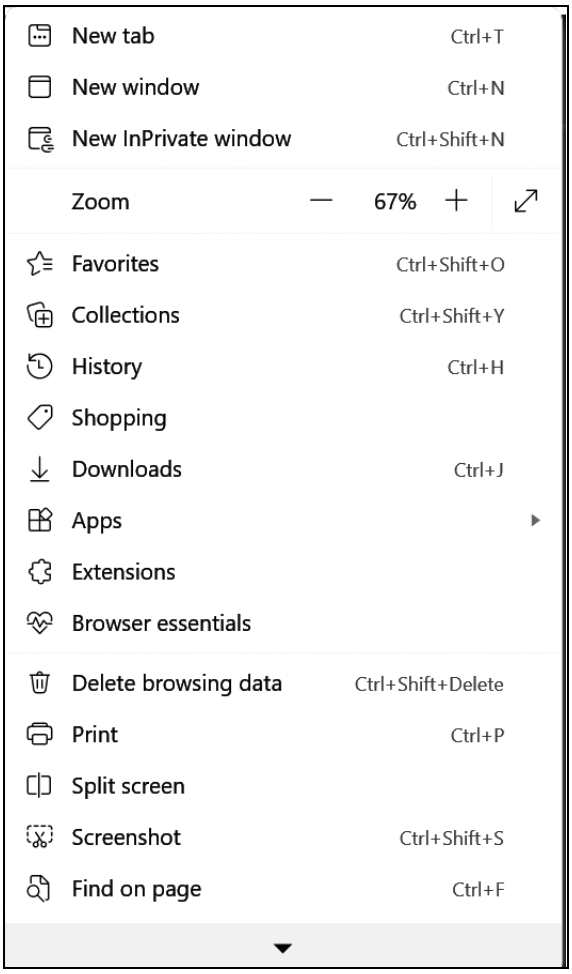
Screenshot:

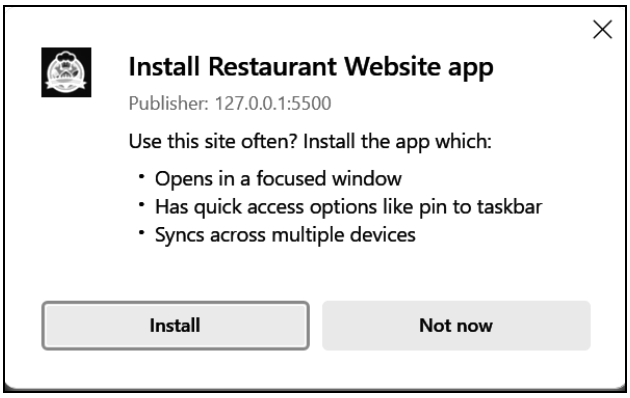

127.0.0.1:5500/index.html
Company
50% Off on All Products
Enjoy Your Delicious Food
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Voluptates, iste corporis tempore necessitatibus inventore ex?
Explore Now
Discount Voucher
Lorem ipsum dolor sit, amet consectetur adipisicing elit.
Fresh Healthy Food
Lorem ipsum dolor sit, amet consectetur adipisicing elit.
Fast Home Delivery
Lorem ipsum dolor sit, amet consectetur adipisicing elit.
Search
ENG IN
22:55
31-03-2025

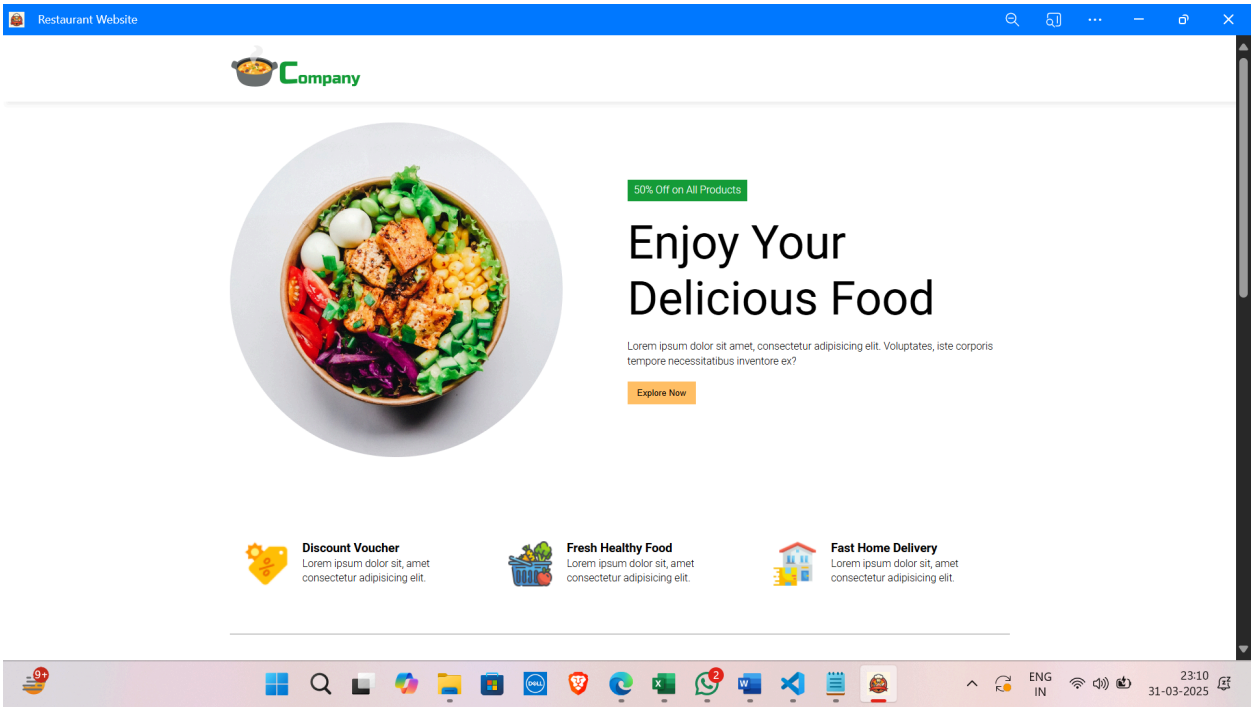

Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
Back/forward cache
Background fetch
Background sync
Bounce tracking miti...
Notifications
Payment handler
Periodic background ...
App Manifest
manifest.json
Errors and warnings
Richer PWA Install UI won't be available on desktop. Please add at least one screenshot with the form_factor set to wide.
Richer PWA Install UI won't be available on mobile. Please add at least one screenshot for which form_factor is not set or set to a value other than wide.
Identity
Name Restaurant Website
Short name cooking
Description Cooking, also known as cookery or professionally as the culinary arts, is the art, science and craft of using heat to make food more palatable, digestible, nutritious, or safe.
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 black


Welcome
Elements
Console
Sources
Network
Performance
Memory
Application
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
Back/forward cache
Background fetch
Background sync
Bounce tracking miti...
Notifications
Payment handler
Periodic background ...
Icons
Show only the minimum safe area for maskable icons
Need help? Read the App Image Generator.
192×192px
image/png
512×512px
image/png


Welcome
Elements
Console
Sources
Network
Performance
Memory
Application
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
Back/forward cache
Background fetch
Background sync
Bounce tracking miti...
Notifications
Payment handler
Periodic background
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5500/
Network requests
Update
Unregister
Source
serviceworker.js
Received 31/3/2025, 11:31:05 pm
Status
#637 activated and is running
Stop
Clients
http://127.0.0.1:5500/
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
Update Cycle
Version
Update Activity
Timeline
#637
Install
#637
Wait
#637
Activate







Conclusion

In this experiment, we successfully implemented the "Add to Homescreen" feature by creating a web app manifest file and registering a service worker for caching. During the process, we faced minor issues with the service worker registration due to incorrect file paths, which we resolved by properly specifying the cache files and verifying the manifest link in the HTML.

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:

In Progressive Web Apps (PWAs), a service worker is a script that runs in the background of the browser, separate from the web page. It helps enhance user experience by enabling features like offline access, background sync, and push notifications. The service worker works using a lifecycle with three main stages:

1. Install – This is when the service worker is first registered and used to cache important files.
2. Activate – After installation, the service worker takes control and clears old caches if needed.
3. Fetch – It intercepts network requests and serves responses from the cache when offline.

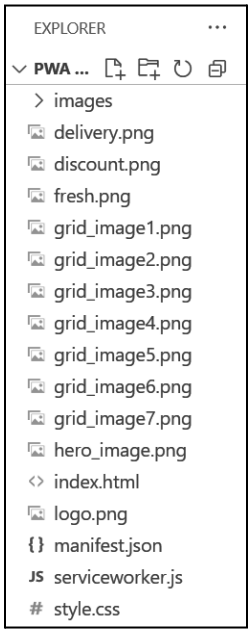
What We Implemented in This Experiment

In our E-commerce PWA project, we performed the following:

- Registered a Service Worker in index.html using JavaScript. This ensures that the browser knows where the service worker file (serviceworker.js) is located.
- In serviceworker.js:
  - We defined a cache name and listed essential files (index.html, style.css, manifest.json, and icons) that need to be stored in cache.
  - During the install event, we used cache.addAll() to store these files so that they can load offline.
  - In the fetch event, the service worker checks if a requested file is in the cache. If it is, it serves the cached version; if not, it fetches from the network.

This implementation allows the app to load essential content even without an internet connection, improving reliability and user experience.

Folder Structure:



Code:

index.html

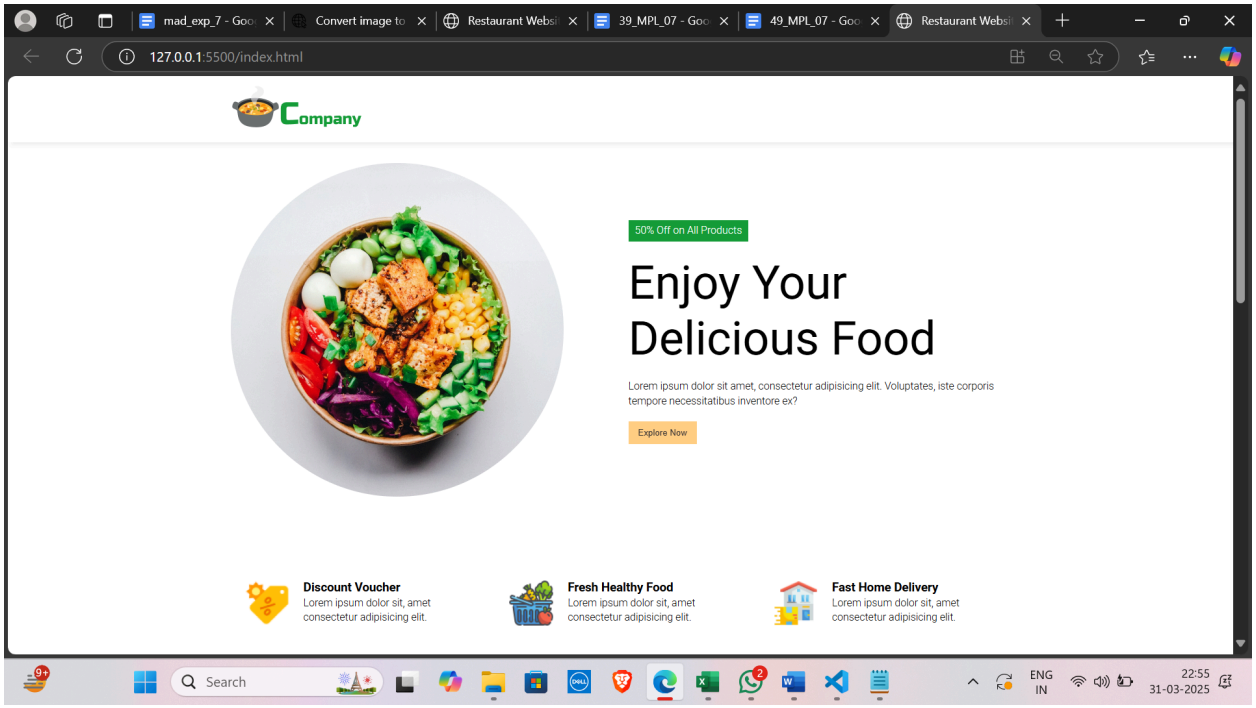
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <meta name="theme-color" content="#007BFF">
    <title>Restaurant Website</title>
    <link rel="stylesheet" href="style.css" />
    <link rel="manifest" href="manifest.json">
    <script
      src="https://kit.fontawesome.com/7a4b62b0a4.js"
      crossorigin="anonymous"
    ></script>
  </head>
  <body>

    <script>
        if ('serviceWorker' in navigator) {
            navigator.serviceWorker.register('serviceworker.js')
                .then(() => console.log("Service Worker Registered"))
                .catch(error => console.log("Service Worker Registration
Failed", error));
        }
    </script>
  </body>
</html>
```

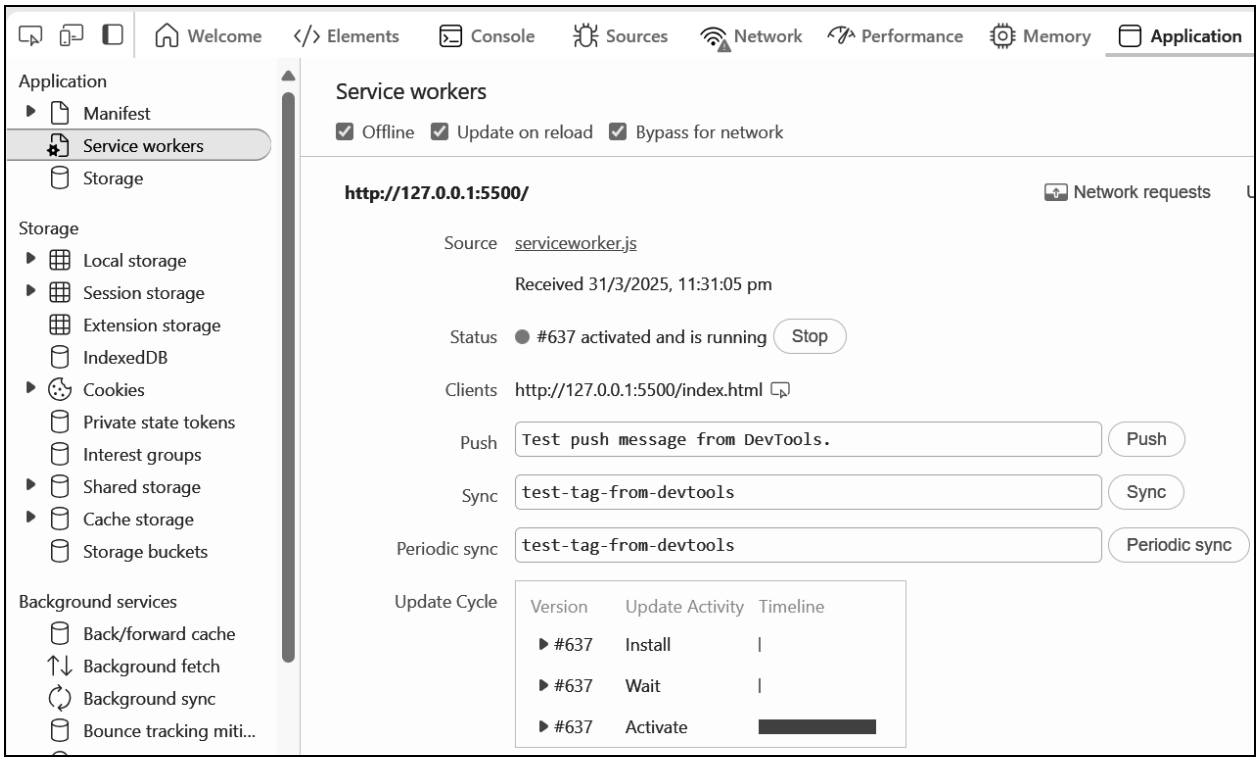
serviceworker.js

```
const CACHE_NAME = "cooking-cache-v1";
const FILES_TO_CACHE = [
    "index.html",
    "style.css",
    "manifest.json",
    "images/app.png",
    "images/big.png"
];

self.addEventListener("install", event => {
    event.waitUntil(
```

```
        caches.open(CACHE_NAME).then(cache => {
            return cache.addAll(FILES_TO_CACHE);
        })
    );
});

self.addEventListener("fetch", event => {
    event.respondWith(
        caches.match(event.request).then(response => {
            return response || fetch(event.request);
        })
    );
});
```

Screenshot:





Conclusion

In this experiment, we successfully implemented and registered a service worker to enable offline support for our E-commerce website by caching essential files. Initially, we faced issues with incorrect file paths and browser caching, but we resolved them by verifying resource locations and updating the cache version for proper service worker activation.

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

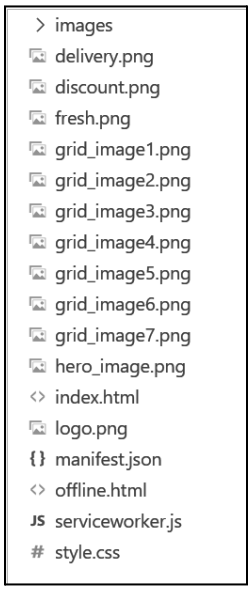
Theory:

In Progressive Web Apps (PWAs), Service Workers play a major role in enabling offline capabilities, background data synchronization, and push notifications. The three key events implemented in this experiment are fetch, sync, and push.

- The fetch event is triggered whenever a network request is made. In our Restaurant PWA, we used this to serve cached files when the user is offline and to ensure faster load times using a Cache First strategy for same-origin requests and Network First for others.
- The sync event allows background synchronization when the user comes back online. We registered a sync event (sync-data) which, in a real-world scenario, could be used to sync orders or updates with a server once connectivity is restored.
- The push event is used for push notifications. In our implementation, we requested permission for notifications and displayed a custom message when a push event is triggered, simulating a real-time alert or update from the server.

This experiment helped us understand how service workers enhance the user experience in PWAs by enabling offline support, background tasks, and real-time communication—all essential for modern E-commerce applications.

Folder Structure:



Code:

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <meta name="theme-color" content="#007BFF" />
    <title>Restaurant Website</title>
    <link rel="stylesheet" href="style.css" />
    <link rel="manifest" href="manifest.json" />
    <script
      src="https://kit.fontawesome.com/7a4b62b0a4.js"
      crossorigin="anonymous"
    ></script>
  </head>
  <body>

    <script>
      if ("serviceWorker" in navigator) {
        window.addEventListener("load", () => {
          navigator.serviceWorker
            .register("/serviceworker.js")
            .then((registration) => {
              console.log(
                "✅ Service Worker registered! Scope:",
                registration.scope
              );

              // 🔔 Request Push Notification Permission
              if ("PushManager" in window) {
                Notification.requestPermission().then((permission) => {
                  if (permission === "granted") {
                    console.log("🔔 Push notifications granted.");
                  } else {
                    console.log("🔕 Push notifications denied.");
                  }
                });
              }

              // 🔄 Register Background Sync
              if ("SyncManager" in window) {
                navigator.serviceWorker.ready.then((swReg) => {
                  swReg.sync
                    .register("sync-data")
                    .then(() => {
                      console.log("🔄 Sync registered");
                    })
                    .catch((err) => {
```

```
                    console.log("❌ Sync registration failed:", err);
                  });
              });
            }
          })
          .catch((error) => {
            console.log("❌ Service Worker registration failed:",
error);
          });
      });
    }
  </script>
</body>
</html>
```

serviceworker.js

```
const CACHE_NAME = "cooking-cache-v2";
const FILES_TO_CACHE = [
  "/index.html",
  "/style.css",
  "/manifest.json",
  "/offline.html",
  "/images/app.png",
  "/images/big.png",
  "delivery.png",
  "discount.png",
  "fresh.png",
  "grid_image1.png",
  "grid_image2.png",
  "grid_image3.png",
  "grid_image4.png",
  "grid_image5.png",
  "grid_image6.png",
  "grid_image7.png",
  "hero_image.png",
  "logo.png"
];


// Install Event
self.addEventListener("install", (event) => {
  console.log("[ServiceWorker] Install");
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
```

```
        console.log("[ServiceWorker] Caching files");
        return cache.addAll(FILES_TO_CACHE);
      })
    );
});


// Activate Event
self.addEventListener("activate", (event) => {
  console.log("[ServiceWorker] Activate");
  event.waitUntil(
    caches.keys().then((keyList) =>
      Promise.all(
        keyList.map((key) => {
          if (key !== CACHE_NAME) {
            console.log("[ServiceWorker] Removing old cache", key);
            return caches.delete(key);
          }
        })
      )
    )
  );
  return self.clients.claim();
});


// Enhanced Fetch Event
self.addEventListener("fetch", (event) => {
  console.log("[ServiceWorker] Fetch", event.request.url);
  const requestURL = new URL(event.request.url);


  // If request is same-origin, use Cache First
  if (requestURL.origin === location.origin) {
    event.respondWith(
      caches.match(event.request).then((cachedResponse) => {
        return ( cachedResponse ||
          fetch(event.request).catch(() => caches.match("offline.html"))
        );
      })
    );
  } else {
    // Else, use Network First
    event.respondWith(
```

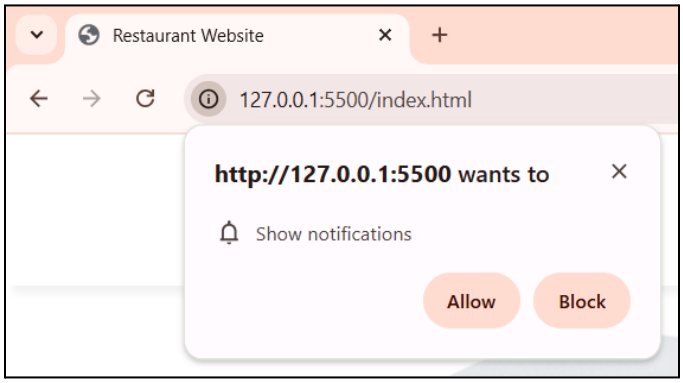
```
        fetch(event.request)
          .then((response) => {
            return response;
          })
          .catch(() =>
            caches.match(event.request).then((res) => {
              return res || caches.match("offline.html");
            })
          )
      );
    }
  });

// Sync Event (simulation)
self.addEventListener("sync", (event) => {
    if (event.tag === "sync-data") {
      event.waitUntil(
        (async () => {
          console.log("Sync event triggered: 'sync-data'");
          // Here you can sync data with server when online
        })()
      );
    }
  });


  // Push Event
  self.addEventListener("push", function (event) {
    if (event && event.data) {
      let data = {};
      try {
        data = event.data.json();
      } catch (e) {
        data = {
          method: "pushMessage",
          message: event.data.text(),
        };
      }


      if (data.method === "pushMessage") {
        console.log("Push notification sent");
        event.waitUntil(
          self.registration.showNotification("Restaurant", {
```

```
            body: data.message,
          })
        );
      }
    }
  });
```

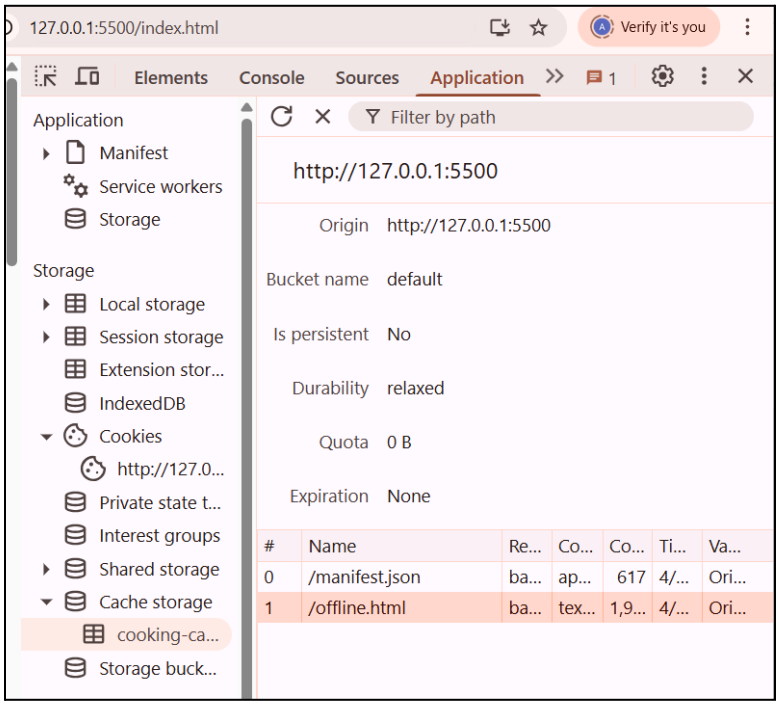
offline.html

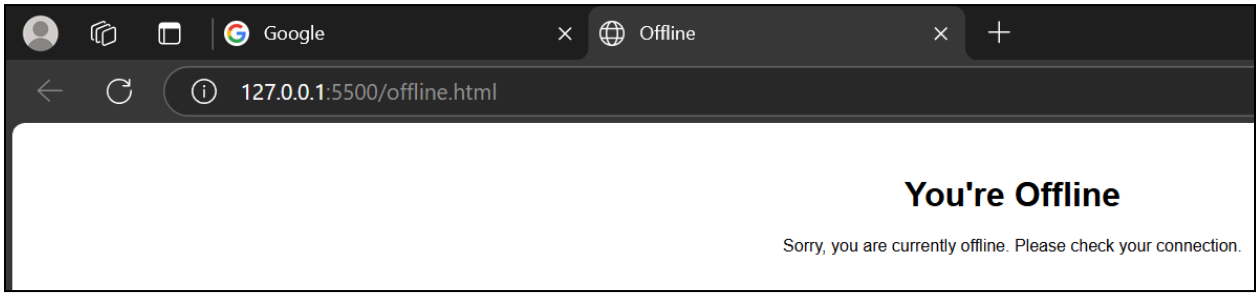
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Offline</title>
    <style>
        body { font-family: Arial, sans-serif; text-align: center; margin:
50px; }
    </style>
</head>
<body>
    <h1>You're Offline</h1>
    <p>Sorry, you are currently offline. Please check your connection.</p>
</body>
</html>
```

Screenshot:

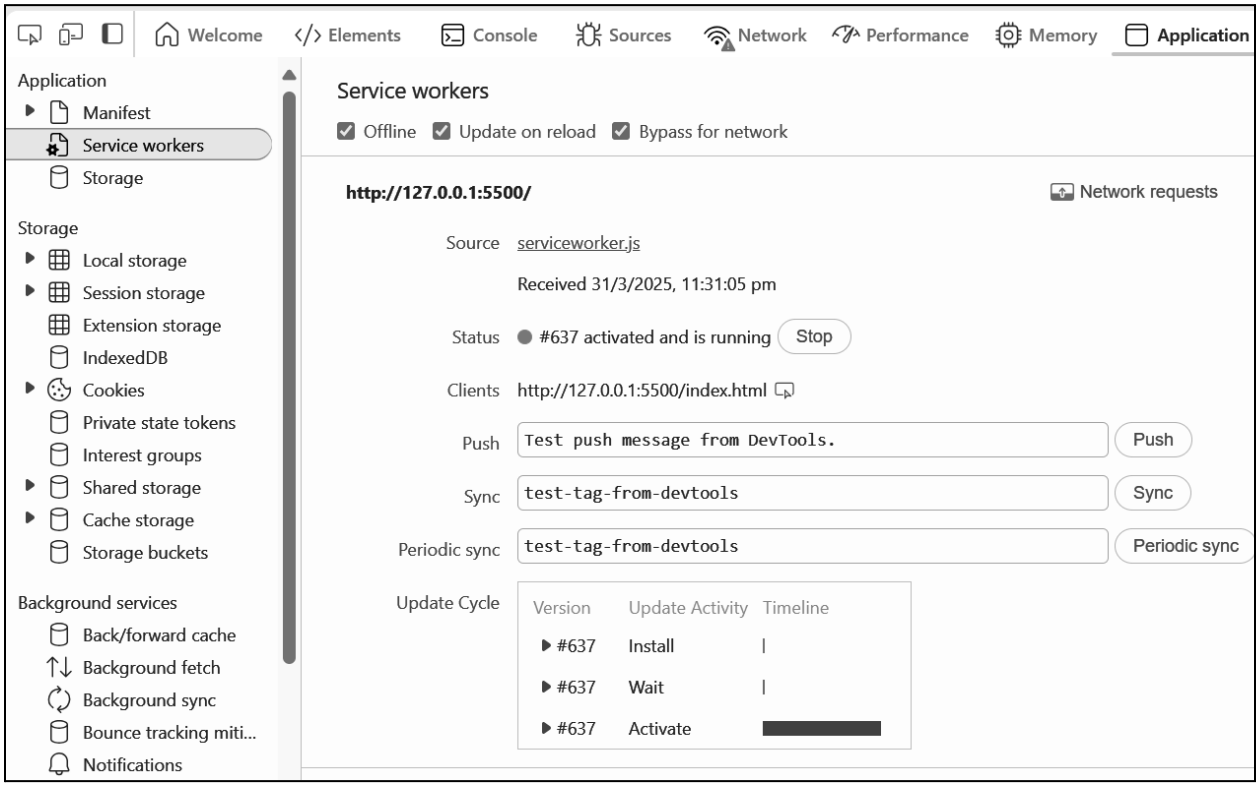


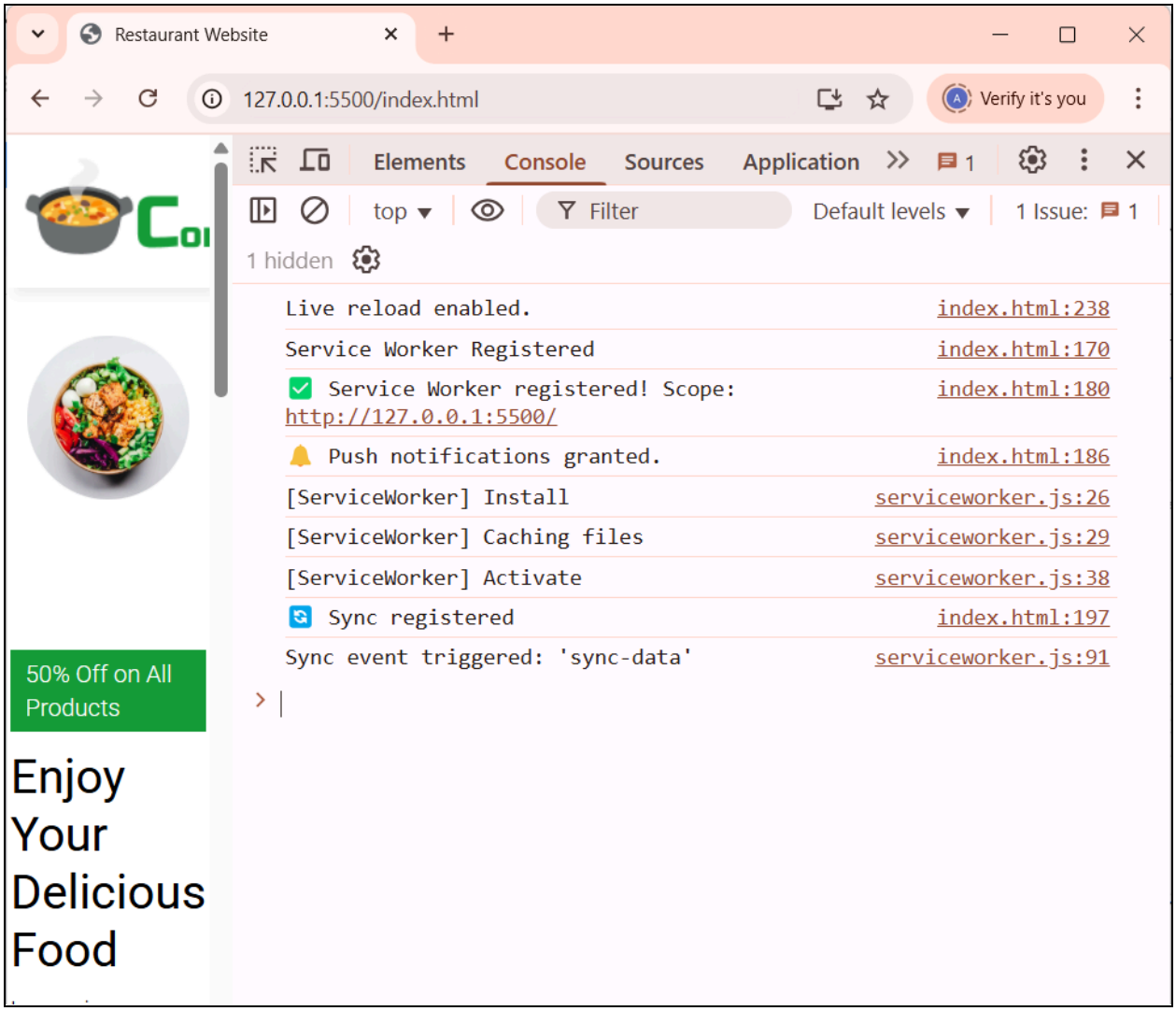
Fetch


127.0.0.1:5500/index.html
Verify it's you
Elements
Console
Sources
Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension stor...
IndexedDB
Cookies
http://127.0...
Private state t...
Interest groups
Shared storage
Cache storage
cooking-ca...
Storage buck...
Filter by path
http://127.0.0.1:5500
Origin http://127.0.0.1:5500
Bucket name default
Is persistent No
Durability relaxed
Quota 0 B
Expiration None
Name Re... Co... Co... Ti... Va...
0 /manifest.json ba... ap... 617 4/... Ori...
1 /offline.html ba... tex... 1,9... 4/... Ori...

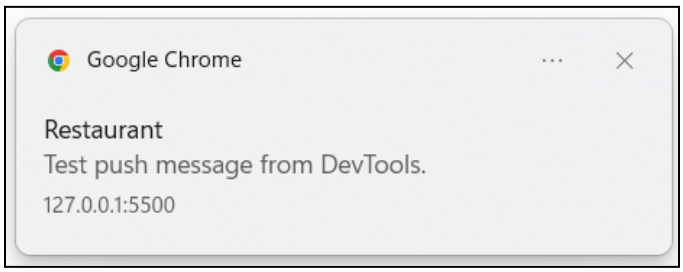

Google
Offline
127.0.0.1:5500/offline.html
You're Offline
Sorry, you are currently offline. Please check your connection.

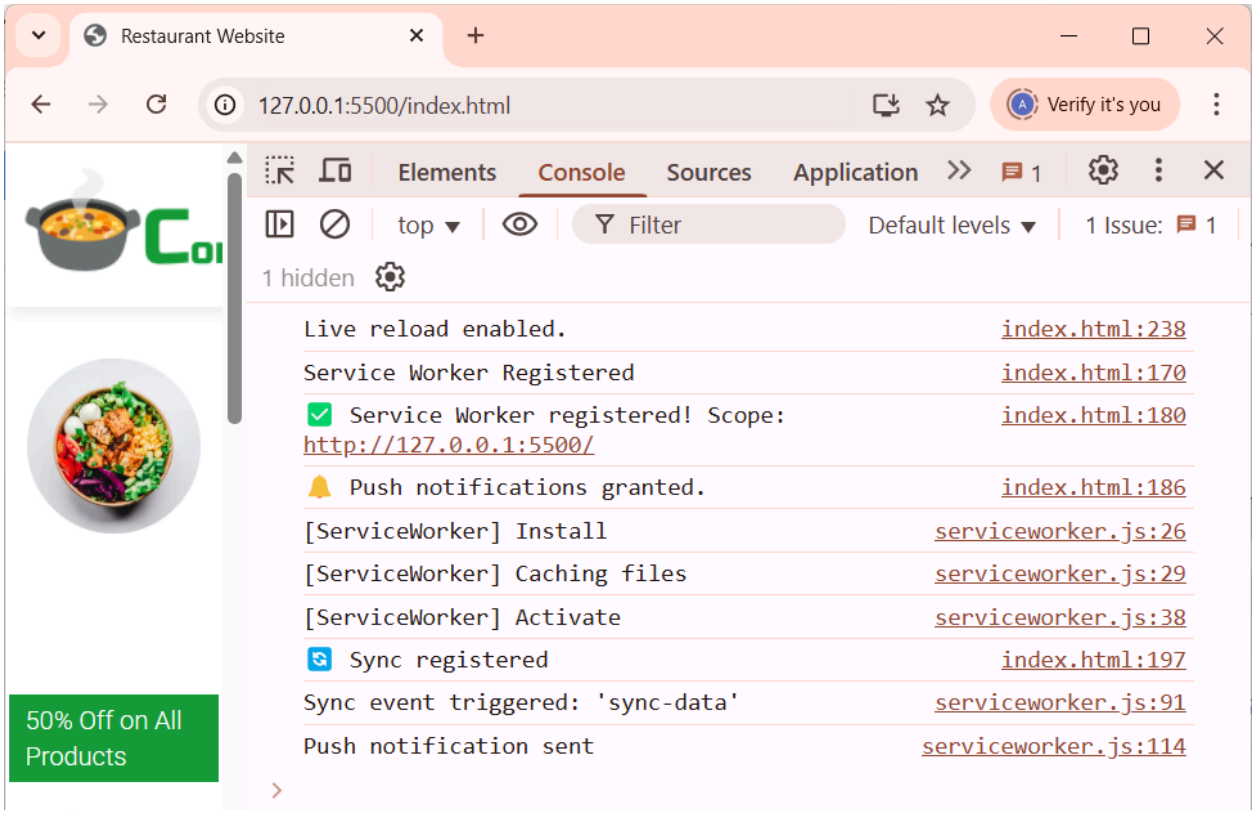
Sync


Welcome
Elements
Console
Sources
Network
Performance
Memory
Application
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
Back/forward cache
Background fetch
Background sync
Bounce tracking miti...
Notifications
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5500/
Network requests
Source serviceworker.js
Received 31/3/2025, 11:31:05 pm
Status #637 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
Update Cycle
Version Update Activity Timeline
#637 Install
#637 Wait
#637 Activate


Restaurant Website
127.0.0.1:5500/index.html
Verify it's you
Elements
Console
Sources
Application
top
Filter
Default levels
1 Issue: 1
1 hidden
Live reload enabled. index.html:238
Service Worker Registered index.html:170
Service Worker registered! Scope: http://127.0.0.1:5500/ index.html:180
Push notifications granted. index.html:186
[ServiceWorker] Install serviceworker.js:26
[ServiceWorker] Caching files serviceworker.js:29
[ServiceWorker] Activate serviceworker.js:38
Sync registered index.html:197
Sync event triggered: 'sync-data' serviceworker.js:91
50% Off on All Products
Enjoy Your Delicious Food

Push


Google Chrome
Restaurant
Test push message from DevTools.
127.0.0.1:5500



Conclusion

In this experiment, we successfully implemented fetch, sync, and push events in the service worker to enable offline access, background sync, and push notifications for our Restaurant PWA. Initially, we faced issues with incorrect file paths and sync event registration, but we resolved them by carefully checking cache file references and ensuring the service worker was ready before registering sync.

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

Theory:

GitHub Pages is a free hosting service provided by GitHub to deploy static websites directly from a GitHub repository. It supports custom domains, Jekyll integration, and has a simple workflow—just push your code to the gh-pages branch, and your site goes live.

Why GitHub Pages?

- Free and easy to use
- Direct integration with GitHub
- Fast setup without third-party tools
- Custom domain support
- Used by many known companies like Lyft and HubSpot

Steps in General Deployment Process:

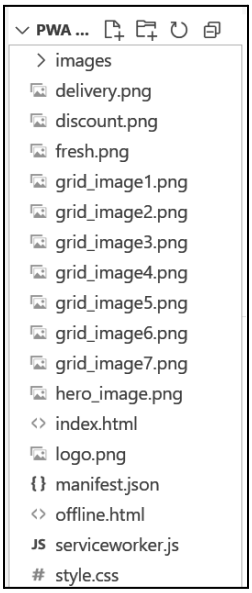
1. Prepare your PWA files (HTML, CSS, JS, manifest, service worker).
2. Create a GitHub repository.
3. Push your local project to GitHub.
4. Use the gh-pages branch or configure GitHub Pages from the repo settings.
5. Your site will be live at https://yourusername.github.io/repository-name.

What We Implemented:

In our E-commerce PWA:

- We already created the manifest.json and registered the service worker in earlier experiments to support offline capability and home screen installation.
- Now, in this experiment, we deployed our complete website (including all PWA features) to GitHub Pages.
- We created a repository on GitHub, pushed our files, and enabled GitHub Pages via the gh-pages branch.
- After deployment, we verified that the PWA still works properly with install prompt and offline support.

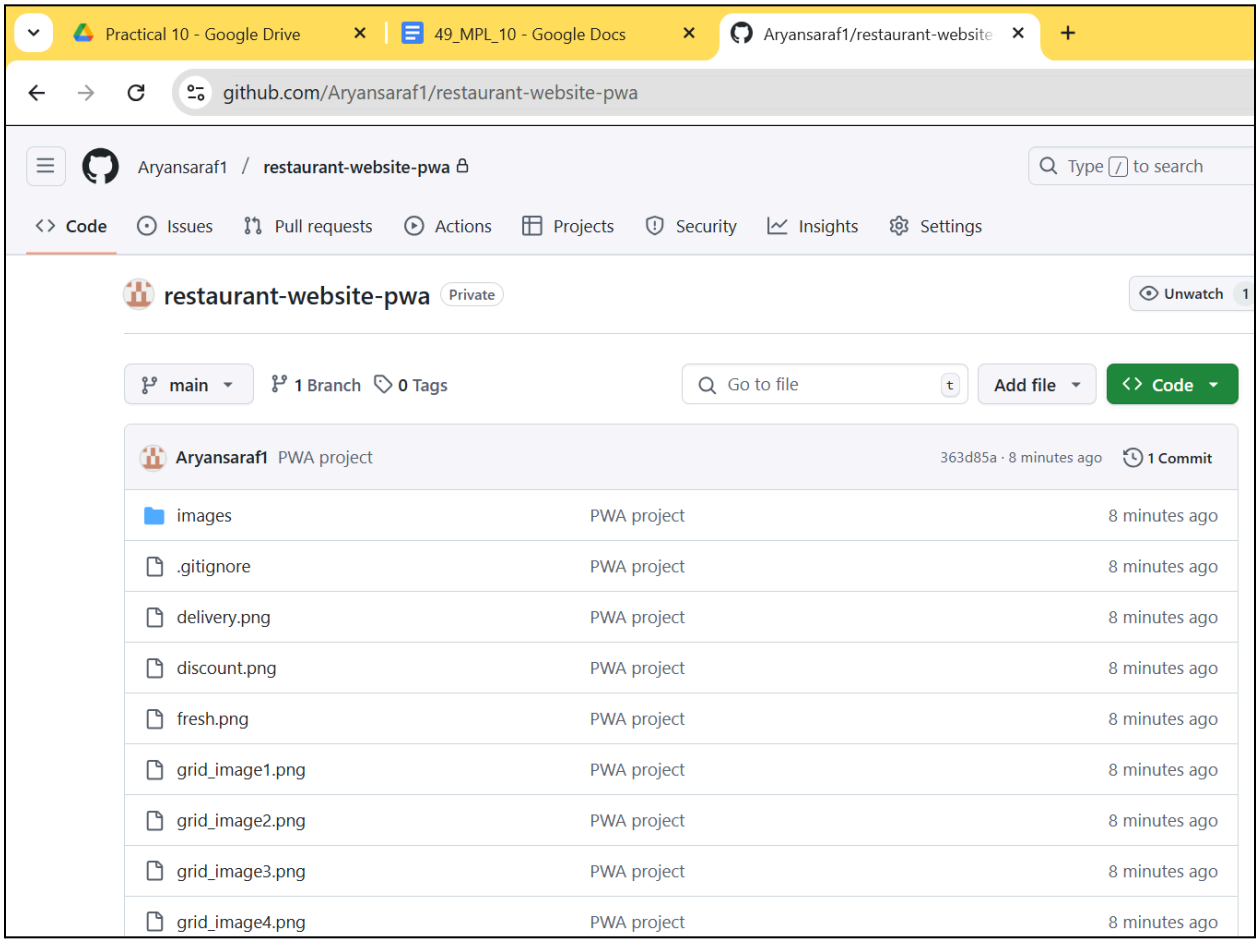
Folder Structure

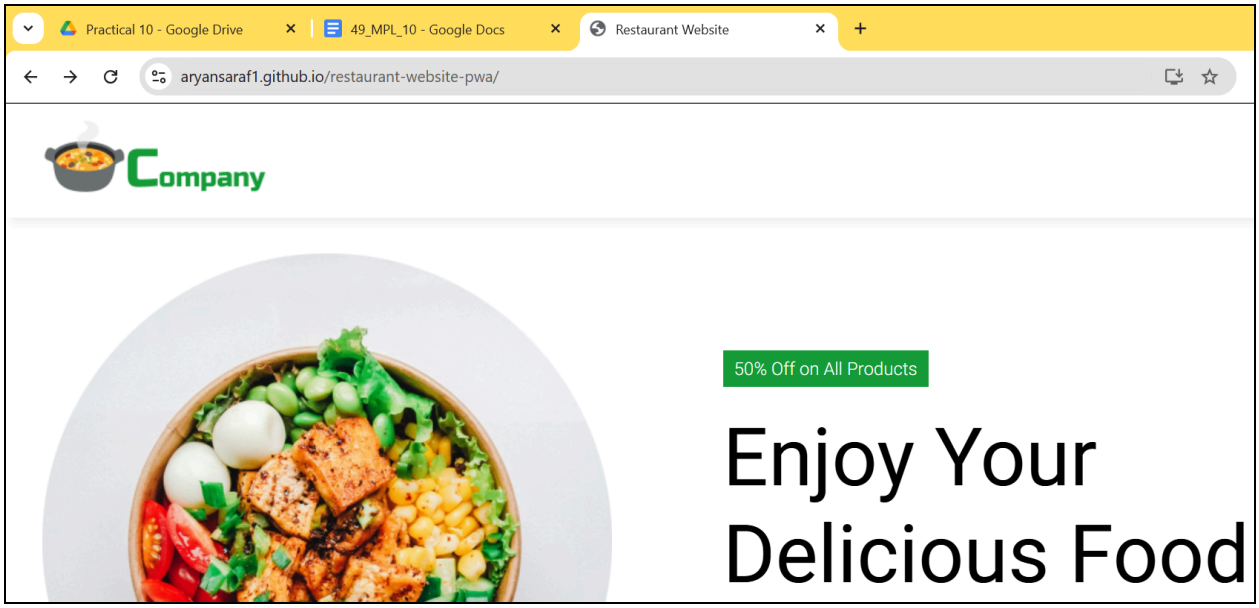


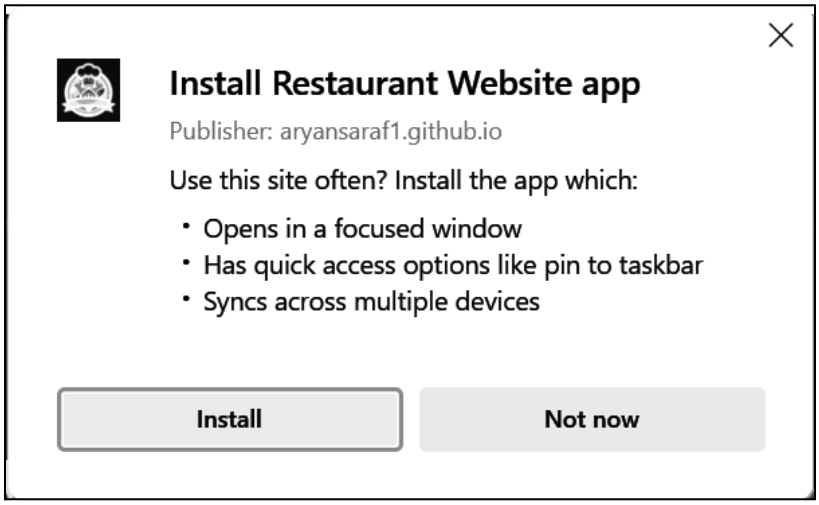
Create a GitHub Repository & Link Your Local Project to GitHub and Push Changes.

```
PS C:\Users\aryan\OneDrive\Desktop\pwa aryan> git init
Initialized empty Git repository in C:/Users/aryan/OneDrive/Desktop/pwa aryan/.git/
PS C:\Users\aryan\OneDrive\Desktop\pwa aryan> git add .
PS C:\Users\aryan\OneDrive\Desktop\pwa aryan> git commit -m "PWA project"
[main (root-commit) 363d85a] PWA project
 20 files changed, 498 insertions(+)
```

```
PS C:\Users\aryan\OneDrive\Desktop\pwa aryan> git remote add origin https://github.com/Aryansaraf1/restaurant-website-pwa.git
PS C:\Users\aryan\OneDrive\Desktop\pwa aryan> git branch -M main
PS C:\Users\aryan\OneDrive\Desktop\pwa aryan> git push -u origin main
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 12 threads
Compressing objects: 100% (22/22), done.
Writing objects: 100% (23/23), 2.24 MiB | 2.22 MiB/s, done.
Total 23 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Aryansaraf1/restaurant-website-pwa.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\aryan\OneDrive\Desktop\pwa aryan>
```







Conclusion

In this experiment, we successfully deployed our Restaurant PWA to GitHub Pages after facing a few issues like incorrect file paths and the service worker not registering initially. We resolved these by updating the relative paths for assets and carefully checking the scope and start_url in the manifest file to ensure the PWA worked smoothly after deployment.

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

Theory:

In this experiment, we used Google Lighthouse, a developer tool available in Chrome, to analyze and test how well our Progressive Web App (PWA) performs. Lighthouse helps measure a web app's performance in terms of speed, accessibility, best practices, SEO, and specifically PWA compliance.

Lighthouse checks if the app behaves like a proper PWA — including features like:

- Fast loading speed,
- Offline support,
- Mobile responsiveness,
- Add to Home Screen prompt,
- Proper service worker registration.

In our project, we already implemented key PWA features like:

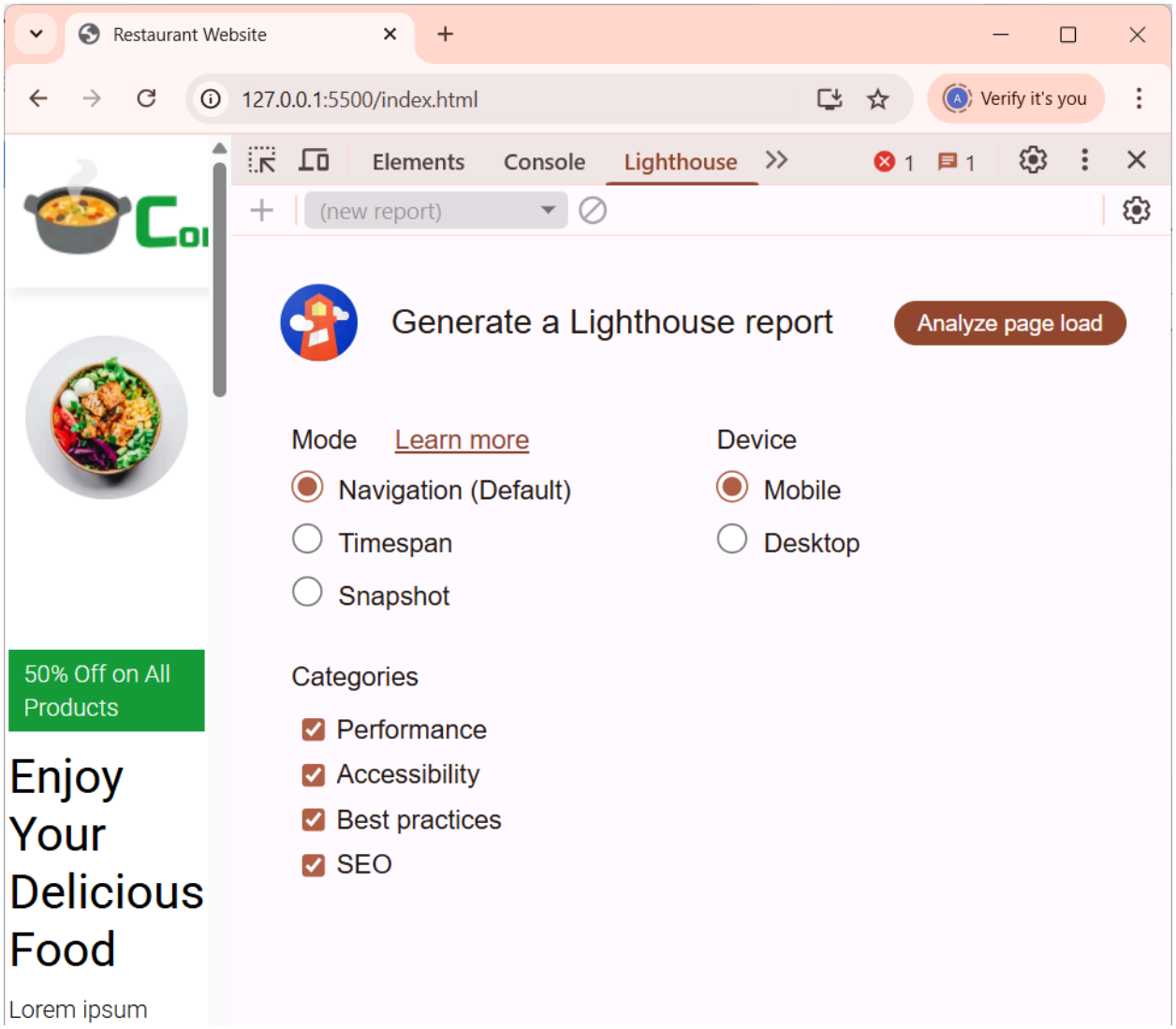
- A registered service worker that handles caching of assets,
- A valid Web App Manifest with app name, icons, and theme colors,
- Offline functionality using cache strategies,
- HTTPS and responsive design for mobile-friendliness.

After running Lighthouse, we got high scores in Performance (96), Accessibility, Best Practices, and SEO. This confirms that our PWA is well-optimized and meets most of the modern standards set for web apps.
This analysis helped us understand the strengths of our PWA and also suggested small improvements, if needed, to make it even better.

manifest.json

```
{
  "name": "Restaurant Website",
  "short_name": "cooking",
  "start_url": "index.html",
  "display": "standalone",
  "background_color": "#5900b3",
  "theme_color": "black",
  "scope": ".",
  "description": "Cooking, also known as cookery or professionally as the culinary arts, is the art, science and craft of using heat to make food more palatable, digestible, nutritious, or safe.",
  "icons": [
    {
      "src": "images/app.png",
      "sizes": "192x192",
      "type": "image/png",
      "purpose": "any maskable"
    },
```

```
    {
      "src": "images/big.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "any maskable"
    }
  ]
}
```



Conclusion

In this experiment, we implemented key PWA features like service worker, manifest, and caching in our restaurant website and used Google Lighthouse to analyze its performance. Initially, the performance score was low due to unoptimized images and unused CSS, but we resolved it by compressing images and removing unnecessary styles, which improved our score significantly.

ARYAN DEEPAK SARAF
D15B
49
MPL ASSIGN - 1

SDS | Page No.
Date / /

a.1] a] Explain key features and advantages of using Flutter for mobile app development.

=> Key features of Flutter :-
1] Single codebase :- Write one code for both Android and iOS.
2] Fast Performance :- Uses Dart languages and a high-performance rendering engine.
3] Hot Reload :- See changes instantly without restarting the app.
4] Rich UI components :- Comes with customizable widgets for smooth UI design.
5] Native-like Experience :- Provides high-quality animations and fast execution.

Advantages of Using Flutter :-
1] Saves & Effort :- Single codebase for multiple platforms.
2] High Speed Development :- Hot Reload feature speeds up coding.
3] Cost-Effective :- Reduces development cost and time.
4] Attractive UI :- Provides beautiful and customizable widgets.
5] Good Performance :- Use Dart and Skia for fast and smooth rendering.

b] Discuss how the flutter framework differs from traditional approaches and why it has gained popularity in developer community

SDS | Page No.
Date | |

=> How Flutter Differs from ~~Trad~~ Traditional Approaches :-

1] Single codebase :- Traditional methods need separate code for Android and iOS, but Flutter uses one code for both.

2] Hot Reload :- Traditional apps require full restart after changes, but Flutter updates instantly.

3] UI Rendering :- Traditional apps use native components, while Flutter has its own rendering engine for faster performance.

4] Performance :- Flutter compiles directly to native machine code, making it faster than framework that uses a bridge.

Why Flutter is Popular Among Developers :-

1] Fast Development :- Hot Reload and single codebase save time.

2] Cross - Platform support :- Works on mobile, web and desktop.

3] Beautiful UI :- Rich, customizable widgets for modern design.

4] High Performance :- Runs smoothly without a bridge like React Native.

Q.2] a] Describe concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interface

DATE:

Concept of widget Tree in Flutter :-
In Flutter, everthing is a widget. Widgets are arranged in a tree structure, called widget tree. This tree represents UI of the app, where parent widget contain child widget.

For example, a Scaffold widget can have a column widget, which contains Text and Button widgets.

Widget composition for complex UI :-
Flutter uses small, reusable widgets to build complex UI, instead of creating a single large UI block, developers combine multiple small widgets like Rows, columns, containers and Buttons
For example :-
1] A ListView can contain multiple card widgets.
2] A column can hold text, Images & Buttons.
the modular approach make UI flexible, readable and easy to manage.

b] Provide examples of commonly used widgets and their roles in creating widget tree.

=> 1] Scaffold :- Provides basics layout structure (AppBar, Body, Floating Button)
2] AppBar :- Displays top navigation bar with title.

DATE:

3] Text :- Displays simple text on the screen
4] Image :- Shows images from assets or URLs.
5] Container :- Used for styling (background color, padding, margin).
6] Row :- Arranges child widgets horizontally.
7] Column :- Arranges child widgets vertically.
8] ListView :- Displays scrollable lists.
9] ElevatedButton - A clickable button with elevation.
10] Textfield :- Used for user input.

Example Widget Tree :-

DATE:

Q.3] a] Discuss the importance of state management in Flutter application.

=> Importance of State Management in Flutter Application :-

State Management is important because it controls how the app stores, updates and displays data when the user interacts with it.

Why State Management is Needed?

1] Keeps UI Updated :- Ensures that the app reflects changes (eg. button clicks, text inputs).

2] Improves Performance :- Updates only necessary parts of UI instead of reloading everything.

3] Manages complex Data :- Helps handle user inputs, API data and navigation efficiently.

4] Ensure smooth User Experience :- Keeps the app responsive & interactive.

Types of state in Flutter :-

1] local state :- Managed within a single widget using statefulWidget.

2] Global state :- Shared across multiple screens using Provider, Riverpod, Bloc or Redux.

DATE:

5] compare and contrast the different state management approaches available in Flutter, such as setState, Provider & Riverpod.

=>

Approach	How it works	When to use
setState	Updates UI by calling setState() in a stateful widget	Best for small apps or managing state within a single widget. Example: Toggling a button color.
Provider	Uses InheritedWidget to share state across widgets efficiently.	suitable for medium sized apps where data needs to be shared between multiple widgets Example: Managing user authentication
Riverpod	An improved version of Provider with better performance & simpler syntax	Best for large apps that need complex state management with dependency injection. Example: Handling API data & app-wide themes

choosing Right Approach :-
- Use setState for simple UI updates.
- Use Provider for moderate state sharing across widgets.
- Use Riverpod for scalable, well-structured applications.

Q.4] a) Explain the process of integrating Firebase with Flutter application. Discuss the benefits of using Firebase as a backend solution.

=> Process of integrating Firebase with a Flutter Application :-

1] create a Firebase Project :- go to [Firebase console] (https://console.firebase.google.com/) , create a new project.
2] Add Firebase to Flutter App :- Register the app (Android / iOS) and download the google-services.json or GoogleService-Info.plist (iOS).
3] Install Firebase Packages :- Add dependencies like 'firebase_core' and 'firebase_auth' in 'pubspec.yaml'.
4] Initialize Firebase
5] Use Firebase services :- Implement authentication, database, or cloud functions as needed.

Benefits of Using Firebase as a Backend solution :-

1] Real-time Database :- syncs data instantly across devices.
2] Authentication :- Provides ready to use sign-in options (Google, Email etc)
3] Cloud Firestore :- Stores structured data efficiently.
4] Hosting & storage :- Hosts web apps & stores files securely.
5] Scalability :- Handles large user bases without managing servers.

b] Highlight the Firebase services commonly used in Flutter development & provide a brief overview of how data synchronization is achieved.

=> Common Firebase services used in Flutter Development :-

1] Firebase Authentication :- Provides user sign-in methods (Google, Email, Facebook)
2] Cloud Firestore :- A NoSQL database that stores & syncs data in real time
3] Firebase Realtime Database :- Stores & Updates data instantly across all connected devices.
4] Firebase cloud storage :- Used for storing & retrieving files like images & videos.
5] Firebase Analytics :- Tracks user behaviour & app performance

SDS | Page No.
Date / /

How data synchronization is Achieved :-

1] Real-time updates :- Firestore & Realtime Database sync data across devices instantly.

2] Listeners & streams :- Widgets listen for changes & update the UI automatically.

3] Offline support :- Firebase caches data, allowing apps to work offline & sync when online.

This ensures fast, smooth & automatic data updates in Flutter apps.

ARYAN DEEPAK SARAF
D15B
49
MPL ASSIGN-2

05/02

SDS | Page No. | Date / /

Q.1] Define Progressive Web App and explain its significance in modern web development. Discuss the key characteristics that differentiate PWAs from traditional mobile apps.

⇒ A progressive web App (PWA) is a type of web application that works like a mobile app but runs in a browser. It can be installed on a device, work offline and provides a fast and smooth user experience.

Significance of PWA in Modern web development :-

1] cross - Platform compatibility :- Works on both mobile and desktop with a single codebase.
2] offline support :- Can function without the internet using cached data.
3] Fast Performe :- Loads quickly, even on slow networks.
4] NO App store required :- Users can install it directly from browser.
5] lower Development cost :- One PWA can replace separate Android and iOS apps.

Key Differences Between PWA and Traditional Mobile Apps :-

Feature	PWA	Traditional Mobile App
Installation	Direct from browser	Download App From App store
Internet Required	Works offline with caching	Usually requires internet
Performance	Fast with service worker	Faster but needs installation
Updates	Automatic, no app store approval	Manual updates needed.
Development cost	Lower (one codebase for all)	Higher (separate apps for each platform)

Q.2) Define responsive web design and explain its importance in context of Progressive web Apps. Compare and constract responsive fluid, and adaptive web design approaches.

=> Definition of Responsive Web Design:
Responsive Web Design (RWD) is techrique that makes web pages adjust automatically to different screen sizes and devices. It ensures a good user experience on mobiles, tablets and desktops without

needing separate versions of website.

Importance of Responsive Design in PWAs :-
1] Better User Experience :- PWAs work smoothly on any device.
2] Faster load time :- optimized design improves speed
2] SEO Benefits :- Google ranks responsive sites higher.

comparison of ~~web~~ Web Design Approaches :-

Approach	How It works	Pros	cons
Responsive	Uses Flexible grids and CSS media queries to adjust layout.	Works on all devices improved SEO.	can be complex to design
Fluid	Uses percent-based widths instead of fixed pixels, so elements resize smoothly	Works well on different screen sizes, easy to implement	less control over layout on large screens.
Adaptive	Uses fixed layout that change at	optimized for known	More efforts required

screen size to design for each screen size.

Key Differences :-

- Responsive adapts dynamically to all screens.
- Fluid resizes smoothly but may not be fully optimized.
- Adaptive loads different layouts based on device type.

Q.3] Describe the lifecycle of service workers, including registration installation and activation phases.

⇒ lifecycle of Service Workers
A Service Worker is a script that runs in background and helps a web app work offline, load faster and send push notifications. The lifecycle has three main phases :-

1] Registration Phase
- The browser registers the service worker using Javascript.

Code Example :-

```
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/sw.js')
    .then(() => console.log('Service Worker
      Registered'))
    .catch(error => console.log('Registeration
      Failed:', error));
}
```

- This tells the browser to install and activate the Service Worker.

2] Installation Phase

- The Service Worker downloads necessary files (HTML, CSS, JS) and stores them in cache.
- If successful, it moves to the activation phase.

Code Example :-

```
self.addEventListener("install", event => {
    event.waitUntil(
        caches.open('app-cache').then(cache => {
    return cache.addAll(['/', '/index.html',
      '/styles.css']);
    })
    );
});
```

- This ensures the app loads even without the internet.

3] Activation Phase

- The old service Worker is replaced with the new one.
- Unused cache files from previous version are deleted.

code Example :-

```
self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(keys => {
      return Promise.all(keys.map(key => {
    if (key !== 'app-cache'){
      return caches.delete(key);
    }
  }));
 })
);
});
```

- The service Worker is now fully active and controls network request.

Final Step : Fetch & Sync

Once activated, the Service Worker intercepts network requests, serves cached files and syncs data when the internet is available.

Q.4] Explain the use of IndexedDB in the service worker for Data storage.

=> Use of IndexedDB in service Worker for Data Storage

IndexedDB is a browser database that stores large amounts of structured data like JSON objects. It helps PWAs work offline by saving and retrieving data efficiently.

Why Use IndexedDB in Service Workers?

1] Offline support :- stores data when offline and syncs it later.

2] Efficient storage :- saves structured data like user settings, cart items or form inputs.

3] Faster Access :- Retrieves data quickly without needing a network request.

4] Persistent Data :- Data remains saved even after the browser is closed.

How Service Workers Use IndexedDB?

Opening the Database

```
let db;
let request = indexDB.open('My Database', 1);

request.onsuccess = function (event) {
    db = event.target.result;
};
```

Creating a store & Adding Data

```
request.onupgradeneeded = function (event) {
    let db = event.target.result;
    let store = db.createObjectStore('users',
    { keyPath: 'id' });
store.add ({id: 1, name: 'John Doe', age: 25
        });
};
```

Fetching Data in service worker

```
let transaction = db.transaction("users', 'readonly');
let store = transaction.objectStore("users");
let getUser = store.get(1);

getUser.onsuccess = function() {
    console.log(getUser.result);
}
```