EXP 1: Installation and Configuration of Flutter Environment.

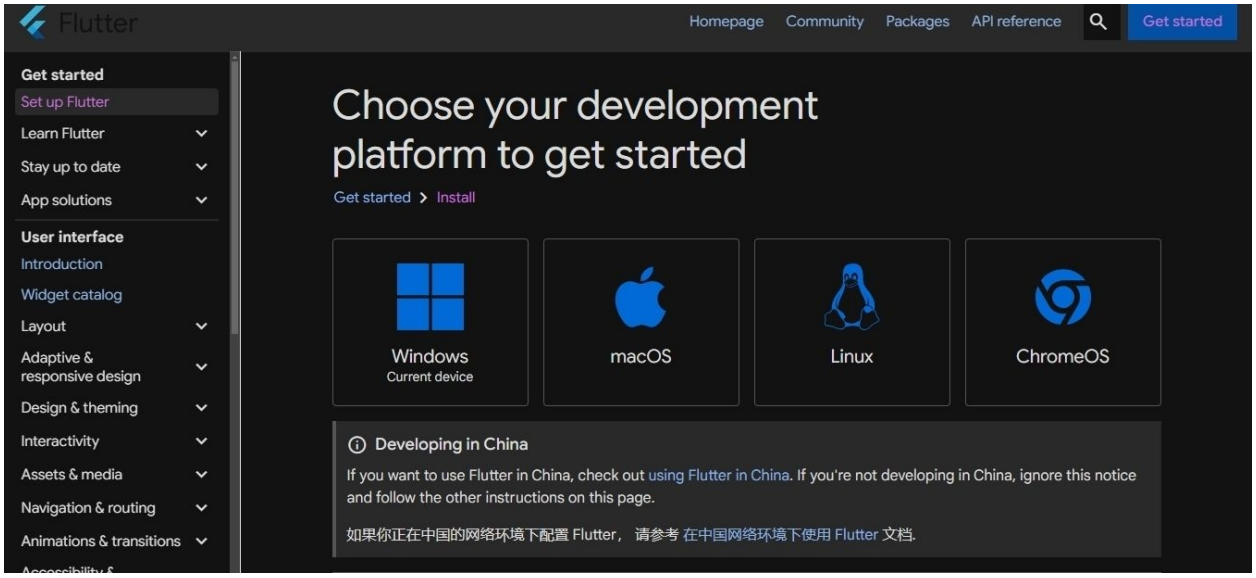
Aim: Installation and Configuration of Flutter Environment.

Theory:

Flutter is an open-source UI toolkit by Google for building natively compiled applications for mobile, web, and desktop using a single codebase. It is based on the Dart programming language and provides pre-designed widgets for creating interactive user interfaces. To start using Flutter, the development environment must be properly set up, including installing the Flutter SDK, configuring an IDE like Android Studio or Visual Studio Code, and setting up an emulator or a physical device for testing.Before installation, the system should meet certain requirements. Windows users need a 64-bit Windows 10 or later with at least 10 GB of free space, while macOS users require macOS 10.14 or later with Xcode installed for iOS development. Linux users should have a 64-bit distribution with necessary dependencies like bash and libstdc++ 6.4 or newer.To install Flutter, download the latest stable version from the official Flutter website, extract it to a preferred location, and add its path to the system's environment variables. Next, an IDE should be configured, with Android Studio being a common choice due to built-in Flutter support. An emulator can be set up using Android Studio's AVD Manager, while iOS development requires Xcode's Simulator. Finally, running flutter doctor in the terminal verifies the installation and checks for any missing dependencies. Once everything is properly configured, Flutter is ready for developing cross-platform applications.

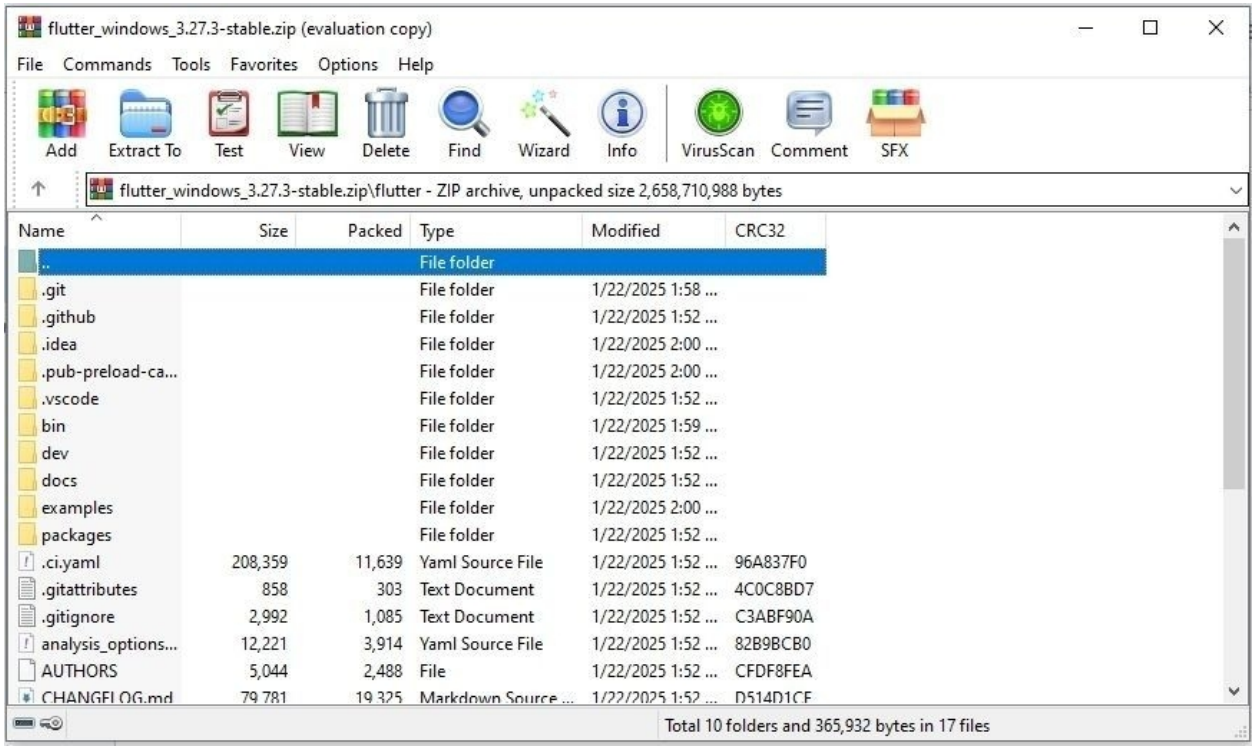
Install the Flutter SDK

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.



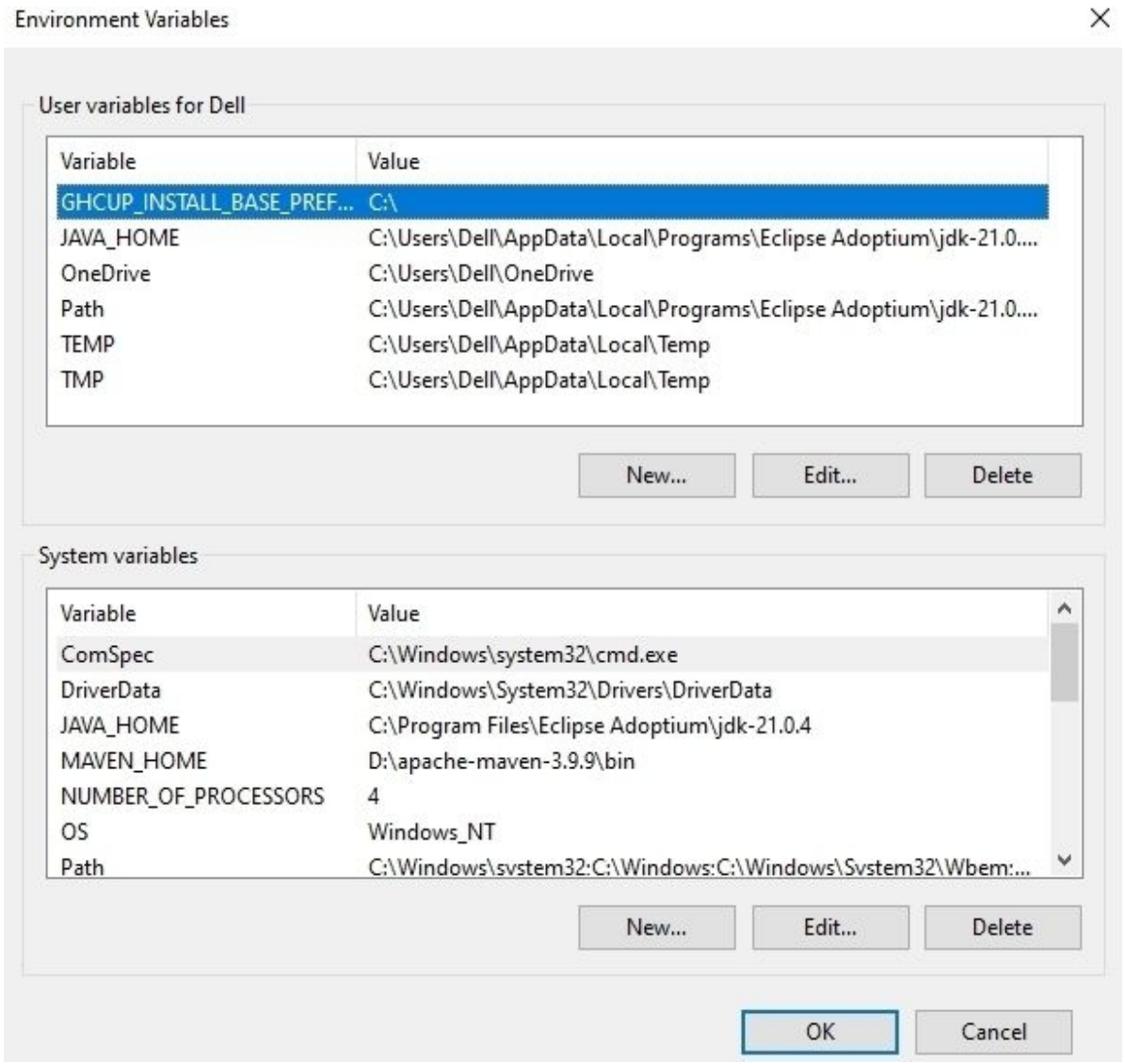
Step 2: Next, to download the latest Flutter SDK, click on the Windows icon. Here, you willfind the download link for SDK.

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.

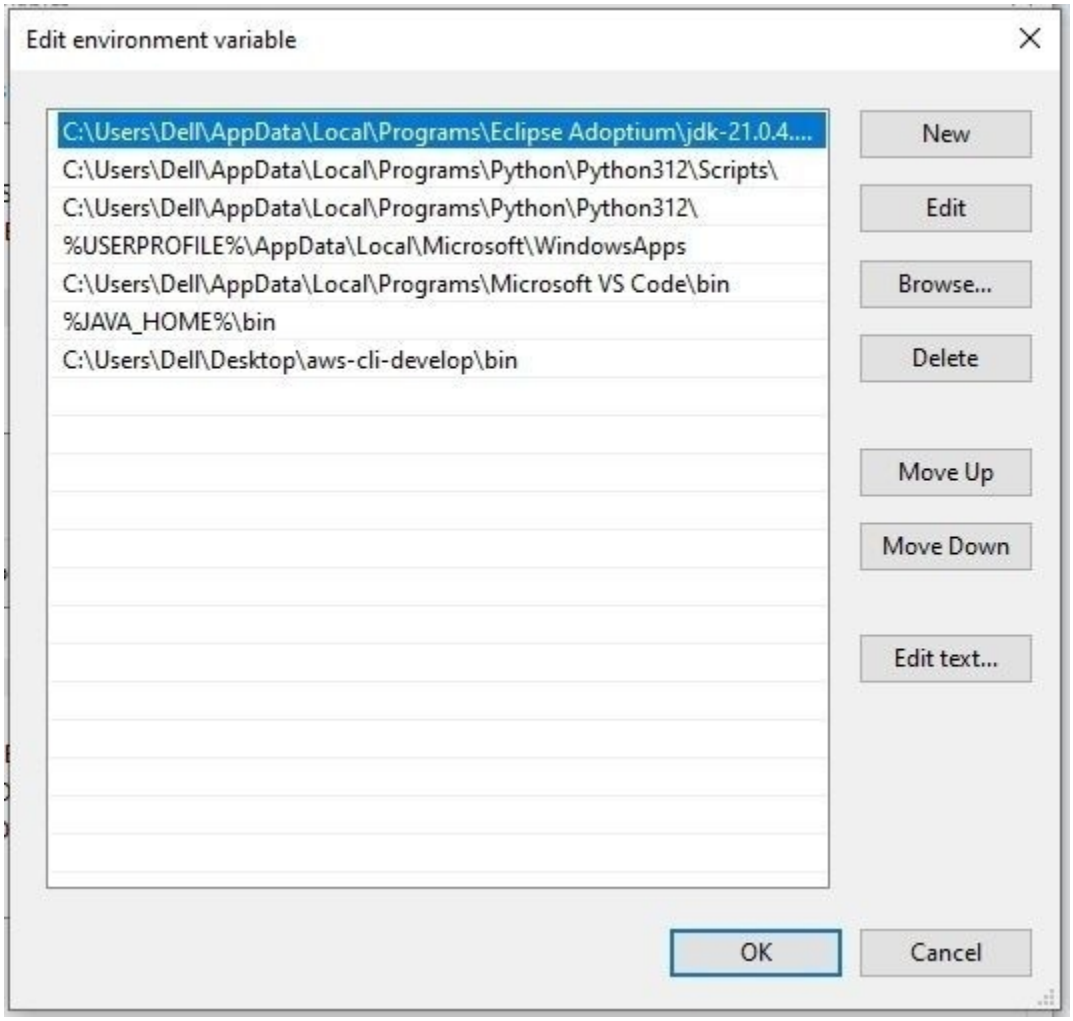


Step 4: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:

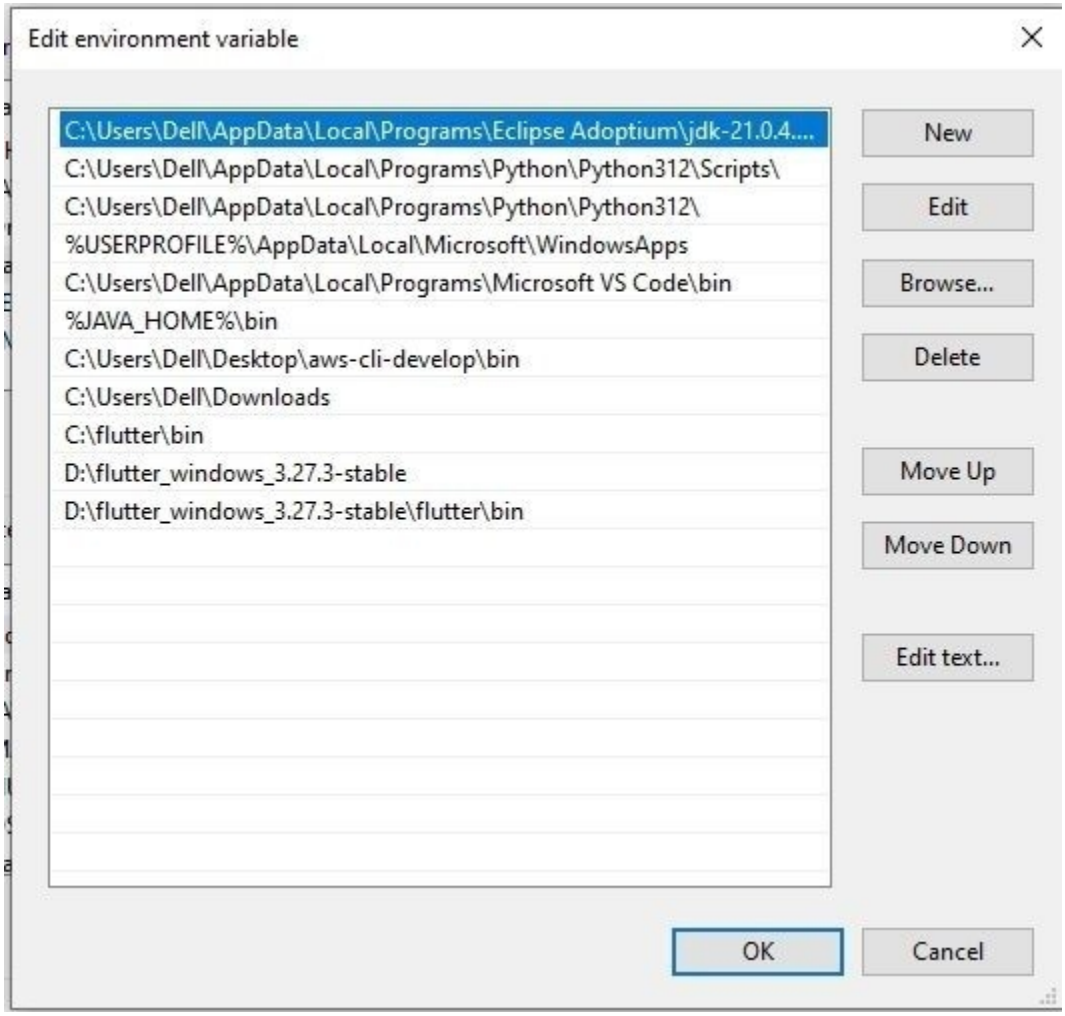
Step 4.1: Go to MyComputer properties -> advanced tab -> environment variables. You will get the following screen.



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 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.

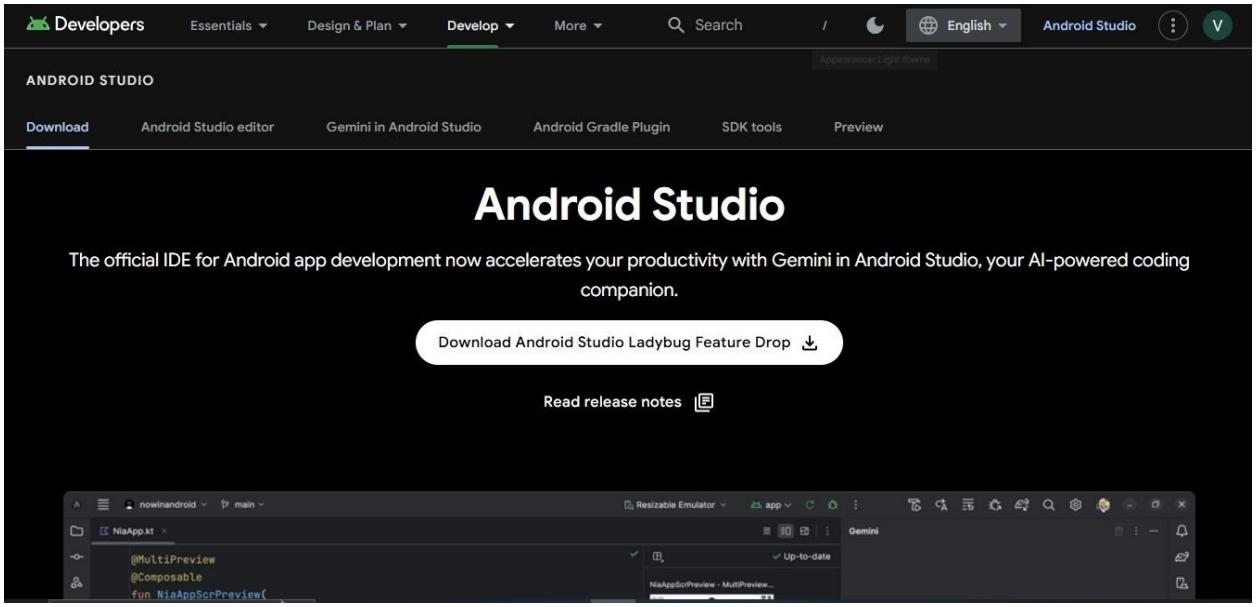
```
D:\>flutter
Manage your Flutter app development.

Common commands:

  flutter create <output directory>
    Create a new Flutter project in the specified directory.

  flutter run [options]
    Run your Flutter application on an attached device or in an emulator.

Usage: flutter <command> [arguments]

Global options:
-h, --help                  Print this usage information.
-v, --verbose               Noisy logging, including all shell commands executed.
                            If used with "--help", shows hidden options. If used with "flutter doctor", shows additional
on. (Use "-vv" to force verbose logging in those cases.)
-d, --device-id             Target device id or name (prefixes allowed).
    --version               Reports the version of this tool.
    --enable-analytics      Enable telemetry reporting each time a flutter or dart command runs.
    --disable-analytics     Disable telemetry reporting each time a flutter or dart command runs, until it is
                            re-enabled.
    --suppress-analytics    Suppress analytics reporting for the current CLI invocation.

Available commands:

Flutter SDK
  bash-completion   Output command line shell completion setup scripts.
  channel           List or switch Flutter channels.
  config            Configure Flutter settings.
  doctor            Show information about the installed tooling.
  downgrade         Downgrade Flutter to the last active version for the current channel.
  precache          Populate the Flutter tool's cache of binary artifacts.
  upgrade           Upgrade your copy of Flutter.

Project
  analyze           Analyze the project's Dart code.
  assemble          Assemble and build Flutter resources.
  build             Build an executable app or install bundle.
  clean             Delete the build/ and .dart_tool/ directories.
  create            Create a new Flutter project.
  drive             Run integration tests for the project on an attached device or emulator.
```

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.

```
D:\>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.27.3, on Microsoft Windows [Version 10.0.19045.5371], locale en-US)
[√] Windows Version (Installed version of Windows is version 10 or higher)
[X] Android toolchain - develop for Android devices
    X Unable to locate Android SDK.
      Install Android Studio from: https://developer.android.com/studio/index.html
      On first launch it will assist you in installing the Android SDK components.
      (or visit https://flutter.dev/to/windows-android-setup for detailed instructions).
      If the Android SDK has been installed to a custom location, please use
      `flutter config --android-sdk` to update to that location.

[√] Chrome - develop for the web
[X] Visual Studio - develop Windows apps
    X Visual Studio not installed; this is necessary to develop Windows apps.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++" workload, including all of its default components
[!] Android Studio (not installed)
[√] VS Code (version 1.96.4)
[√] Connected device (3 available)
[√] Network resources

! Doctor found issues in 3 categories.
```

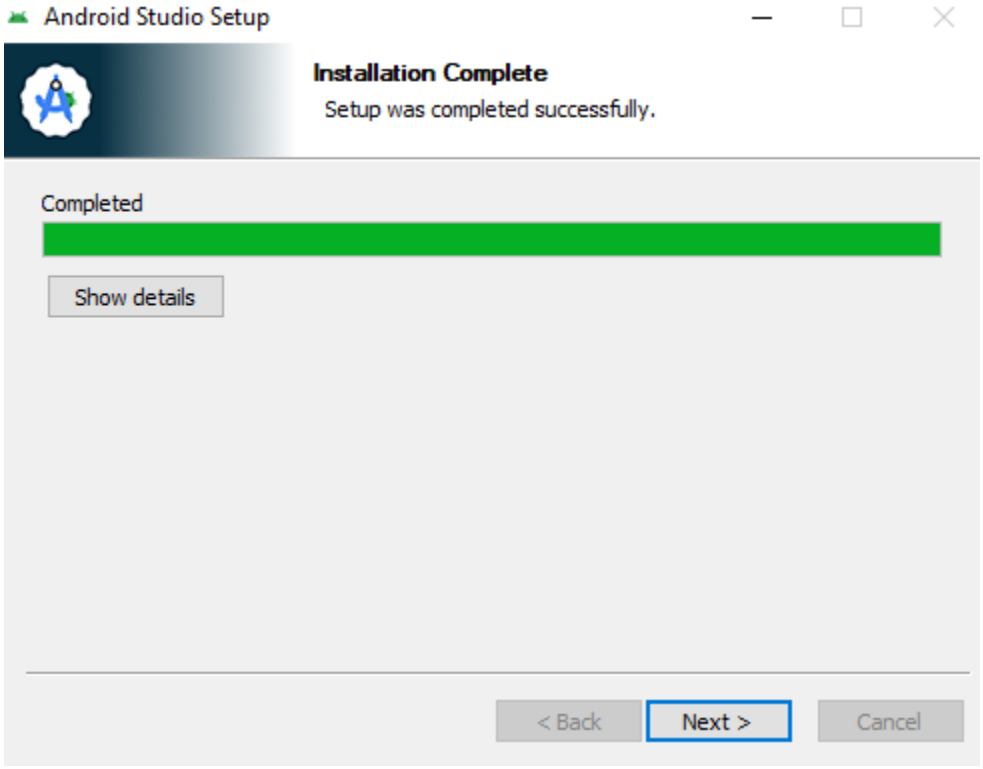
Step 7: Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.

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.

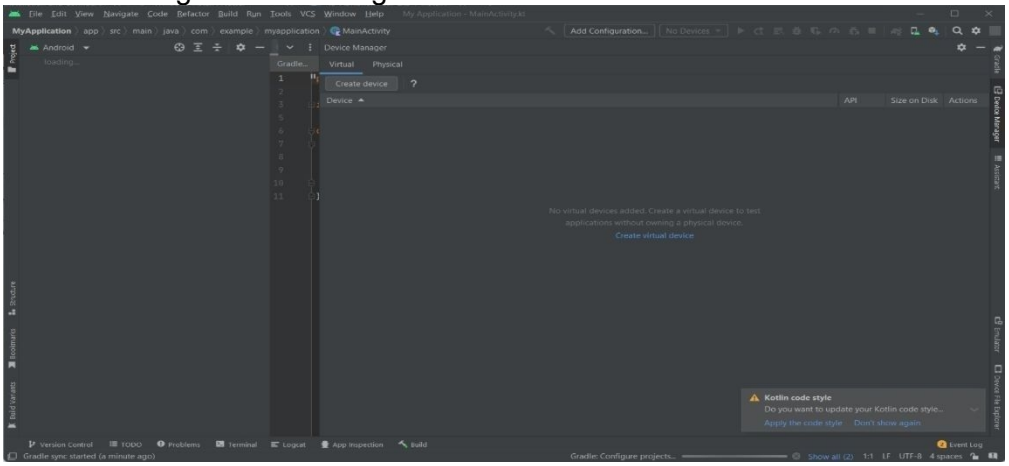
Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



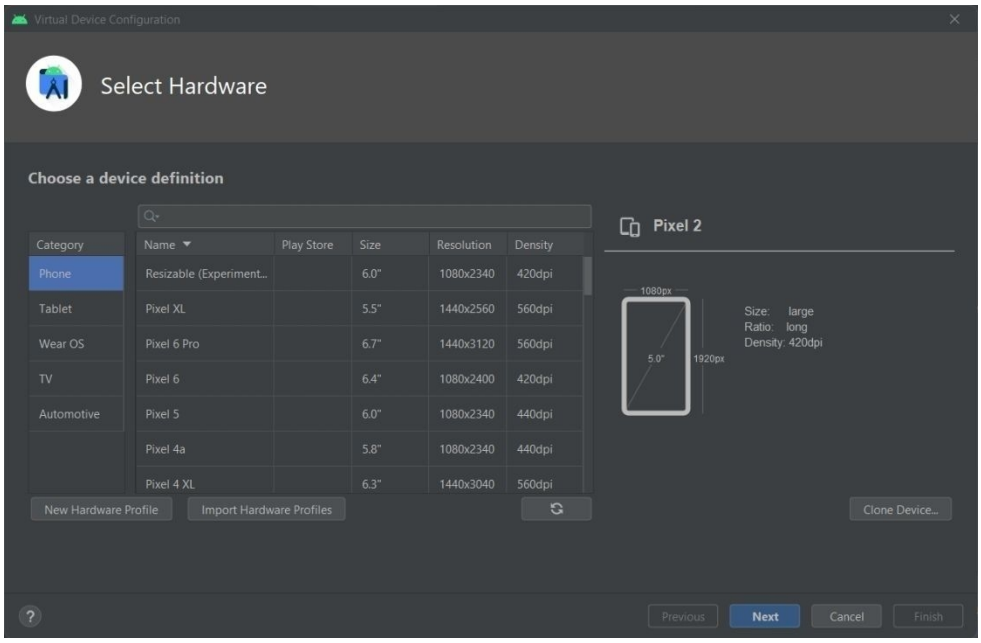
Step 7.4: In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to choose the 'Don't import Settings option' and click OK. It will start the Android Studio.
run the $ flutter doctor command and Run flutter doctor --android-licenses command

```
                                                    Warning: Errors during XML parse:
Warning: Additionally, the fallback loader failed to parse the XML.
Warning: Errors during XML parse:       ] 49% Fetch remote repository...
Warning: Additionally, the fallback loader failed to parse the XML.ry...
[======================================] 100% Computing updates...
All SDK package licenses accepted.
```

Step 8: Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.
Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.

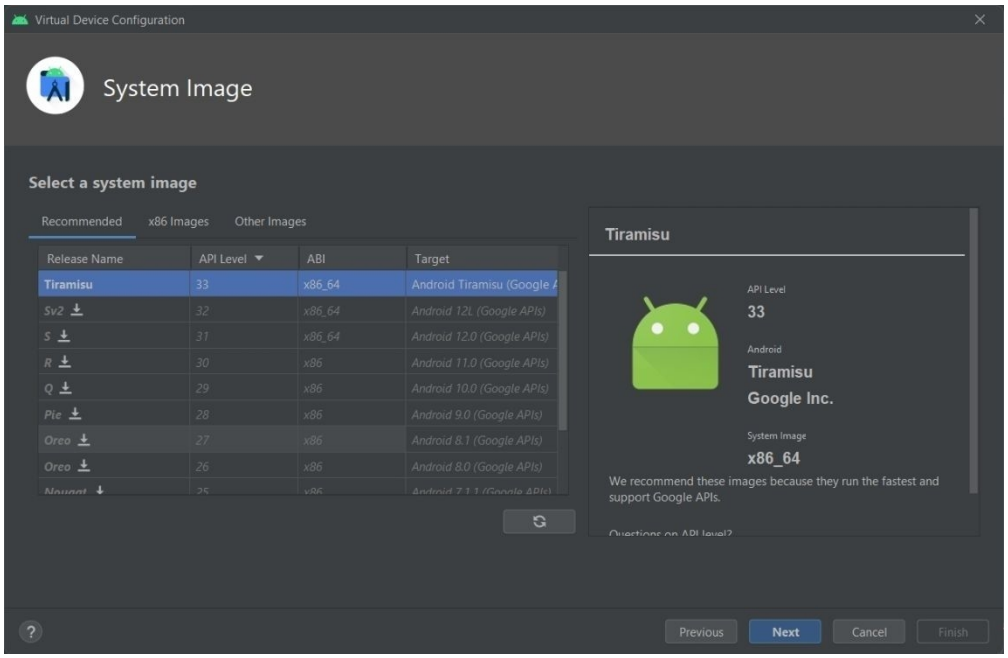


Step 8.2: Choose your device definition and click on Next.

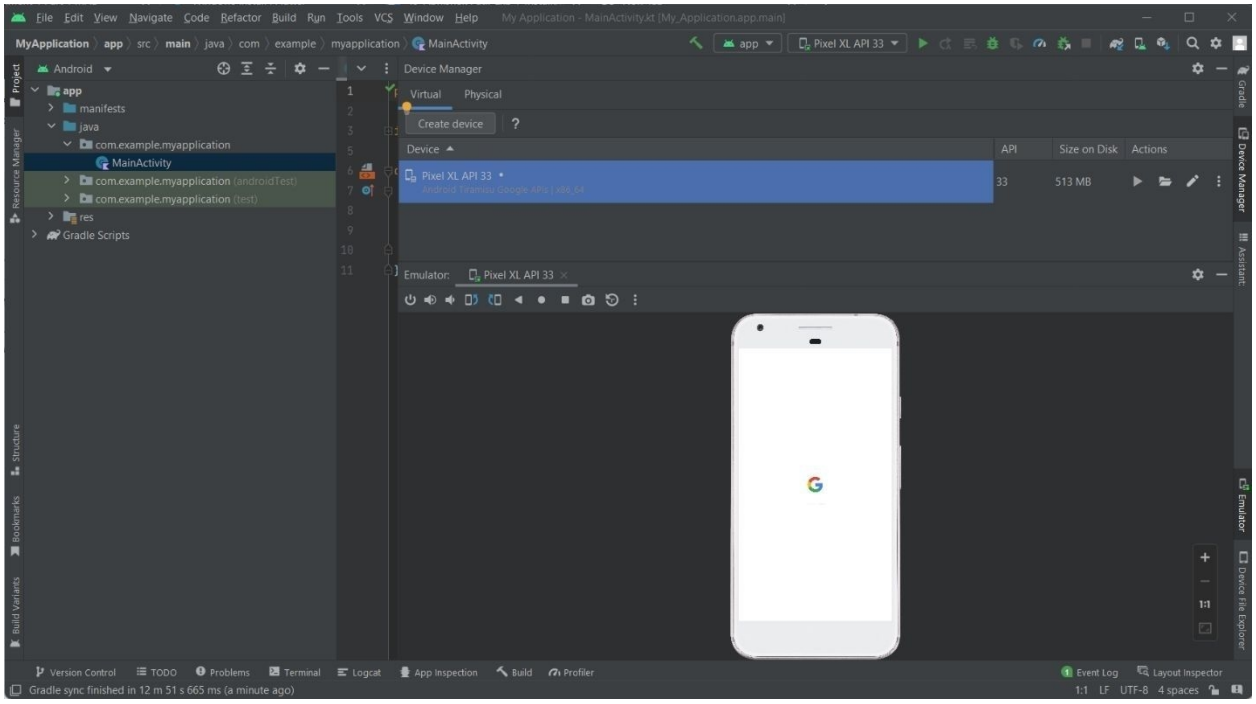


Step 8.3: Select the system image for the latest Android version and click on Next.

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

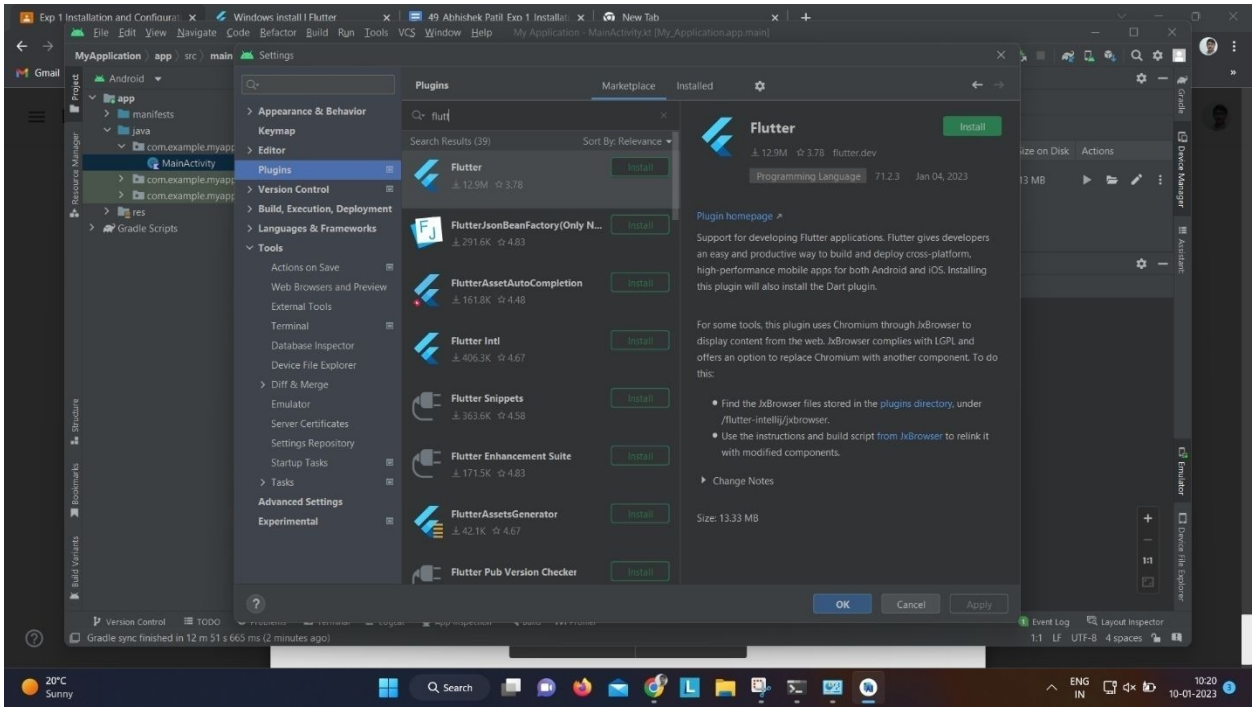


Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.



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

Conclusion:
Setting up the Flutter environment is essential for cross-platform app development. Proper installation of the Flutter SDK, IDE, and dependencies ensures a smooth workflow. A well-configured setup enables efficient development, testing, and deployment of high-performance applications across multiple platforms**.**

MAD Ex 2

Aim: To design Flutter UI by including common widgets

Theory:

Flutter is a UI toolkit by Google used to build cross-platform applications with a widget-based architecture. Widgets are the fundamental building blocks of Flutter UI, categorized into structural, interactive, display, navigation, and state management widgets.

- Structural widgets like Container, Row, and Column define layout.
- Interactive widgets like ElevatedButton and TextField handle user input.
- Display widgets such as Text, Image, and Card enhance visual appeal.
- Navigation widgets like Navigator and Drawer help in screen transitions.
- State management is crucial with StatefulWidget and providers, aiding in dynamic UI updates.

Flutter's widget-based approach ensures consistency, reusability, customizability, and responsive design, making UI development efficient and scalable.

Conclusion:

In conclusion, Flutter's widget-based approach simplifies UI development by providing a vast collection of reusable and customizable widgets. By leveraging structural, interactive, display, and navigation widgets, developers can create visually appealing and responsive applications efficiently.The flexibility and performance optimizations of Flutter widgets make it an ideal choice for building modern cross-platform user interfaces.

Code:

```
import 'package:flutter/material.dart';
void main() {
  runApp(AccountApp());
}
class AccountApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Account(),
    );
  }
}
class Account extends StatelessWidget {
  const Account({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white24,
      appBar: AppBar(
        centerTitle: true,
        backgroundColor: Colors.white24,
        leading: IconButton(
          onPressed: () {
            Navigator.pop(context);
          },
          icon: Icon(Icons.arrow_back),
        ),
        title: Text(
          "Settings",
          style: TextStyle(color: Colors.white),
        ),
      ),
      body: SafeArea(
        child: Padding(
          padding: EdgeInsets.fromLTRB(10, 20, 10, 0),
          child: SingleChildScrollView(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
children: [
                _buildSectionHeader("PROFILE"),
                SizedBox(height: 2),
                _buildProfileButton(
                  label: "VESIT\n VESIT",
```

```
                onPressed: () {
                  print("You pressed Profile Button");
                },
              ),
              SizedBox(height: 10),
              _buildSectionHeader("FEATURES"),
              SizedBox(height: 10),
              _buildFeatureButton(
                label: "Memories",
                icon: Icons.calendar_today,
                onPressed: () {
                  print("You pressed Memories Button");
                },
              ),
              SizedBox(height: 2),
              _buildFeatureButton(
                label: "Blocked Profile",
                icon: Icons.block,
                onPressed: () {
                  print("You pressed Blocked Profile Button");
},
              ),
              SizedBox(height: 10),
              _buildSectionHeader("SETTINGS"),
              SizedBox(height: 10),
              _buildFeatureButton(
                label: "Notifications",
                icon: Icons.notifications,
                onPressed: () {
                  print("You pressed Notifications Button");
                },
              ),
              SizedBox(height: 2),
              _buildFeatureButton(
                label: "Time Zone",
                icon: Icons.access_time,
                onPressed: () {
                  print("You pressed Time Zone Button");
                },
              ),
              SizedBox(height: 2),
              _buildFeatureButton(
                label: "Others",
                icon: Icons.settings_suggest,
```

```
                onPressed: () {
                  print("You pressed Others Button");
                },
              ),
              SizedBox(height: 10),
              _buildSectionHeader("ABOUT"),
              SizedBox(height: 10),
              _buildFeatureButton(
                label: "Share BeReal",
                icon: Icons.share,
                onPressed: () {
                  print("You pressed Share BeReal Button");
},
              ),
              SizedBox(height: 2),
              _buildFeatureButton(
                label: "Rate",
                icon: Icons.star_outline,
                onPressed: () {
                  print("You pressed Rate Button");
                },
              ),
              SizedBox(height: 2),
              _buildFeatureButton(
                label: "Help",
                icon: Icons.help_outline,
                onPressed: () {
                  print("You pressed Help Button");
                },
              ),
              SizedBox(height: 2),
              _buildFeatureButton(
                label: "About",
                icon: Icons.info,
                onPressed: () {
                  print("You pressed About Button");
                },
              ),
            ],
          ),
        ),
      ),
    ),
  );
```

```
  }
  Widget _buildSectionHeader(String title) {
    return Text(
      title,
      style: TextStyle(fontSize: 22, fontWeight: FontWeight.bold),
    );
  }
 Widget _buildProfileButton({required String label, required VoidCallback onPressed}) {
return ElevatedButton.icon(
      onPressed: onPressed,
      icon: Icon(Icons.account_circle, color: Colors.white),
      label: Text(label, style: TextStyle(color: Colors.white, fontSize: 16)),
      style: ElevatedButton.styleFrom(
        backgroundColor: Colors.deepOrangeAccent,
        padding: EdgeInsets.symmetric(vertical: 20, horizontal: 30),
      ),
    );
  }
Widget _buildFeatureButton({required String label, required IconData icon, required
VoidCallback onPressed}) {
    return ElevatedButton.icon(
      onPressed: onPressed,
      icon: Icon(icon, color: Colors.white),
      label: Text(label, style: TextStyle(color: Colors.white, fontSize: 18)),
      style: ElevatedButton.styleFrom(
        backgroundColor: Colors.deepOrangeAccent,
        padding: EdgeInsets.symmetric(vertical: 20, horizontal: 30),
      ),
    );
  }
}
```

←

PROFILE

VESIT
VESIT

FEATURES

Memories

Blocked Profile

SETTINGS

Notifications

Time Zone

Others

ABOUT

Share BeReal

Rate

Help

About

```
A Dart VM Service on Chrome is available at: http://127.0.0.1:51393/9cYPtot4fBY=
The Flutter DevTools debugger and profiler on Chrome is available at: http://127.0.0.1:9101?uri=http://127.0.0.1:51393/9cYPtot4fBY=
You pressed Profile Button
You pressed Profile Button
You pressed Memories Button
You pressed Blocked Profile Button
You pressed Notifications Button
You pressed Time Zone Button
You pressed Others Button
You pressed Share BeReal Button
```

Usage of Widgets:

● Widgets like Row, Column, SizedBox, ElevatedButton, and Align are used to structure the UI.

● The SafeArea widget ensures that content is displayed within the safe area of the screen.

● The SingleChildScrollView widget allows scrolling when the content overflows the screen

List of Widgets

Flutter Scaffold
Flutter Container
Flutter Row & Column
Flutter Text
Flutter TextField
Flutter Buttons
Flutter Stack
Flutter Forms
Flutter AlertDialog
Flutter Icons
Flutter Images
Flutter Card
Flutter Tabbar
Flutter Drawer
Flutter Lists
Flutter GridView
Flutter Toast
Flutter Checkbox
Flutter Radio Button
Flutter Progress Bar
Flutter Snackbar
Flutter Tooltip
Flutter Slider
Flutter Switch
Flutter Charts
Bottom Navigation Bar
Flutter Themes
Flutter Table
Flutter Calendar
Flutter Animation

MAD Lab 3

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

Theory:

Flutter allows developers to enhance the user interface by incorporating icons, images, and custom fonts seamlessly.

- Icons can be added using Flutter's built-in Icons class, which provides a collection of Material Design icons. These icons can be customized in terms of size, color, and other styling properties.
- For custom icons, developers can use vector graphics formats like SVG or icon fonts such as FontAwesome, by integrating third-party packages like flutter_svg or font_awesome_flutter.
- Images in Flutter can be included from assets, network sources, or the device's file system.
    - To add asset images, developers need to place the image files in the assets directory and declare them in the pubspec.yaml file under the flutter section.
    - Images can then be displayed using the Image.asset widget.
    - Similarly, network images (URLs) can be loaded using the Image.network() widget, making it convenient for dynamic content.

Conclusion:

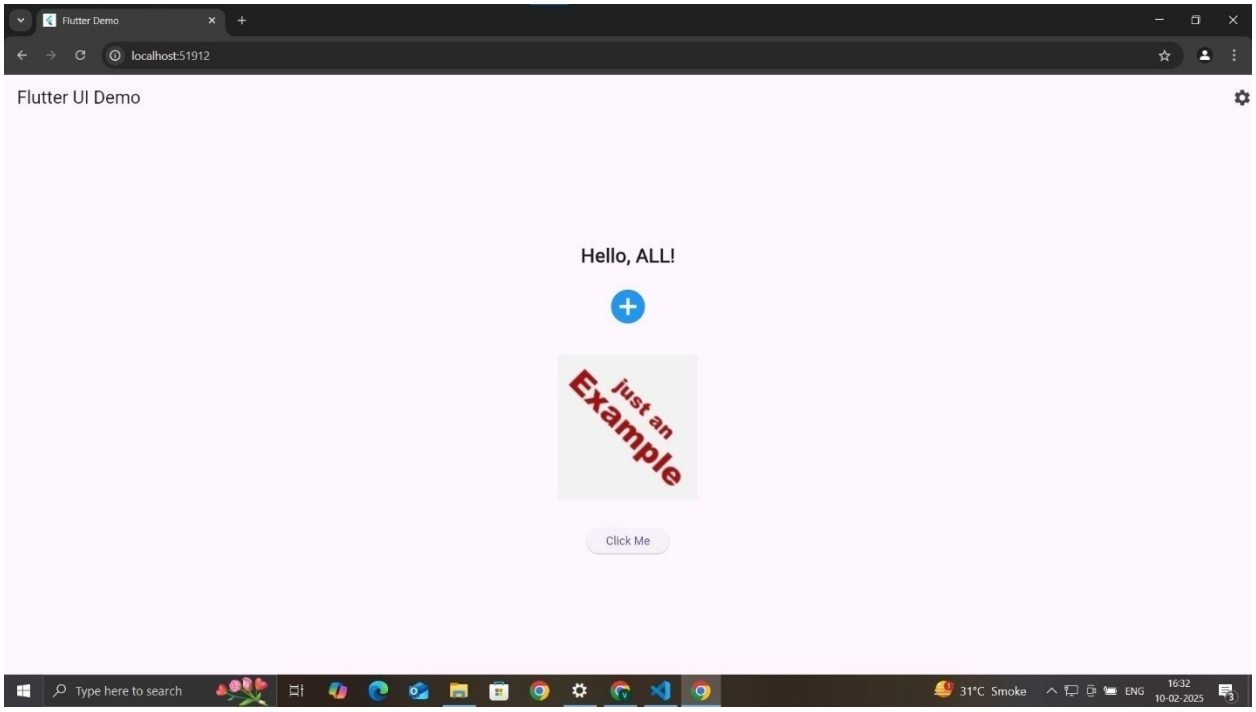
Flutter provides a seamless way to integrate icons, images, and custom fonts, allowing developers to create visually appealing and dynamic applications. By utilizing built-in and custom icons, asset and network images, and custom fonts, developers can enhance the user experience and maintain a consistent design. Proper asset management and pubspec.yaml configuration ensure smooth implementation, making Flutter a powerful framework for UI-rich applications.

Code:

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
@override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        fontFamily: 'CustomFont', // Applying custom font
),
      home: const MyHomePage(),
    );
  }
}
class MyHomePage extends StatelessWidget {
  const MyHomePage({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Flutter UI Demo'),
        actions: const [
          Icon(Icons.settings), // Built-in icon
        ],
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
children: [
            // Custom Font Text
            const Text(
              'Hello, ALL!',
              style: TextStyle(
                fontSize: 24,
                fontWeight: FontWeight.bold,
                fontFamily: 'CustomFont',
              ),
```

```
            ),
            const SizedBox(height: 20),
            // Icon Example
            const Icon(
              Icons.add_circle,
              size: 50,
              color: Colors.blue,
            ),
            const SizedBox(height: 20),
  // Image Example (Using a placeholder direct link)
            Image.network(
              'https://upload.wikimedia.org/wikipedia/commons/a/a9/Example.jpg', // Placeholder
direct image URL
              width: 200,
              errorBuilder: (context, error, stackTrace) {
                return const Text(
                  'Image failed to load',
                  style: TextStyle(color: Colors.red),
                );
              },
            ),
            const SizedBox(height: 20),
            // Button Example
            ElevatedButton(
              onPressed: () {
                // Perform an action
              },
              child: const Text('Click Me'),
            ),
          ],
        ),
      ),
    );
  }
}
```

Output


Flutter Demo
localhost:51912
Flutter UI Demo
Hello, ALL!
just an Example
Click Me
Type here to search
31°C Smoke
ENG
16:32
10-02-2025

MAD Ex 4

Aim: To create an interactive form using the form widget in Flutter.

Theory:

The Form widget in Flutter provides a structured way to handle user input with validation and state management. It contains form fields like:

- Text form fields (TextFormField)
- Dropdown buttons (DropdownButtonFormField)
- Checkbox lists

While working alongside GlobalKey FormState, form validation and submission validation are implemented using the validator property. Form actions like validate(), save(), and reset() ensure efficient input handling. By utilizing these widgets, developers can create user-friendly and responsive forms that enhance data accuracy and user experience.

Importance of using the Form widget:

1) **Ensures data validation** – Prevents incorrect user input.
2) **Manages form state** – Tracks changes and updates fields.
3) **Enhances user experience** – Provides interactive input handling.
4) **Simplifies form submission** – Enables easy data processing and handling.

Conclusion:

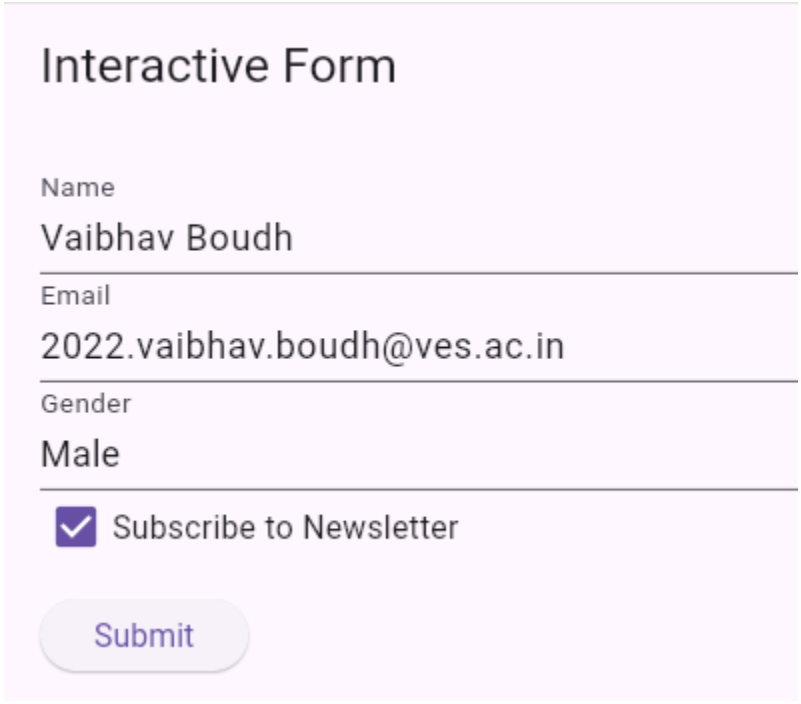
Creating an interactive form using the Form widget in Flutter improves user input handling by providing structured validation and state management. By utilizing various form elements like TextFormField, DropdownButtonFormField, and buttons, developers can design efficient and user-friendly forms for mobile applications.

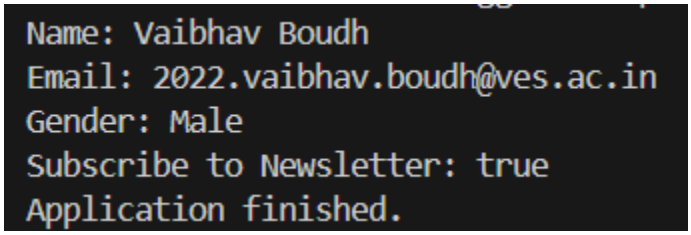
Code to create an interactive Form using form widget

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Interactive Form'),
        ),
        body: MyForm(),
      ),
    );
  }
}
class MyForm extends StatefulWidget {
  @override
  _MyFormState createState() => _MyFormState();
}
class _MyFormState extends State<MyForm> {
  final _formKey = GlobalKey<FormState>();
  String _name = '';
  String _email = '';
  String _selectedGender = 'Male';
  bool _subscribeToNewsletter = false;
 @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(16.0),
      child: Form(
        key: _formKey,
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            TextFormField(
              decoration: InputDecoration(labelText: 'Name'),
validator: (value) {
                if (value == null || value.isEmpty) {
                  return 'Please enter your name';
                }
                return null;
```

```
            },
            onSaved: (value) {
              _name = value!;
            },
          ),
          TextFormField(
            decoration: InputDecoration(labelText: 'Email'),
            validator: (value) {
              if (value == null || value.isEmpty || !value.contains('@')) {
return 'Please enter a valid email address';
              }
              return null;
            },
            onSaved: (value) {
              _email = value!;
            },
          ),
          DropdownButtonFormField<String>(
            value: _selectedGender,
            items: ['Male', 'Female', 'Other']
                .map((gender) => DropdownMenuItem(
                      value: gender,
                      child: Text(gender),
                    ))
                .toList(),
            onChanged: (value) {
              setState(() {
                _selectedGender = value!;
              });
            },
            decoration: InputDecoration(labelText: 'Gender'),
          ),
          Row(
            children: [
              Checkbox(
                value: _subscribeToNewsletter,
                onChanged: (value) {
                  setState(() {
                    _subscribeToNewsletter = value ?? false;
                  });
                },
              ),
              Text('Subscribe to Newsletter'),
            ],
```

```
          ),
          SizedBox(height: 16),
          ElevatedButton(
            onPressed: () {
              if (_formKey.currentState!.validate()) {
                _formKey.currentState!.save();
                // Process the form data, e.g., send it to a server
                print('Name: $_name');
                print('Email: $_email');
                print('Gender: $_selectedGender');
                print('Subscribe to Newsletter: $_subscribeToNewsletter');
}
            },
            child: Text('Submit'),
          ),
        ],
      ),
    ),
  );
 }
}
```





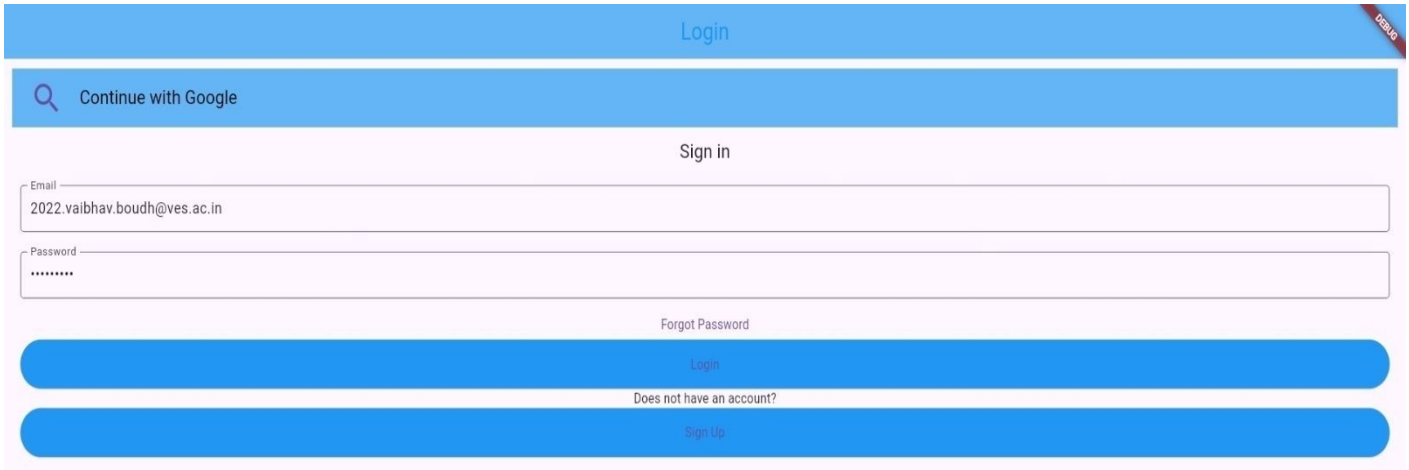
Code to create an interactive Login and sign in page

```
import 'package:flutter/material.dart';
void main() => runApp(const MyApp());
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  static const String _title = 'Sample App';
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: _title,
      home: Scaffold(
        appBar: AppBar(
          elevation: 0,
          backgroundColor: Colors.blue[300],
          centerTitle: true,
          title: Text(
            'Login',
            style: TextStyle(
              color: Colors.blue[500],
              fontFamily: 'Roboto',
              fontSize: 24,
            ),
          ),
        ),
        body: const MyStatefulWidget(),
      ),
    );
  }
}
class MyStatefulWidget extends StatefulWidget {
  const MyStatefulWidget({Key? key}) : super(key: key);
  @override
  State<MyStatefulWidget> createState() => _MyStatefulWidgetState();
}
class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  TextEditingController emailController = TextEditingController();
  TextEditingController passwordController = TextEditingController();
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(10),
      child: ListView(
        children: <Widget>[
          Container(
```

```
        width: double.infinity,
    alignment: Alignment.centerLeft,
        padding: const EdgeInsets.all(10),
        color: Colors.blue[300],
        child: Column(
          children: [
            TextButton(
              onPressed: () {},
              child: Row(
                children: [
                  Icon(
                    Icons.search, // Replace with the appropriate Google icon
size: 40,
                  ),
                  SizedBox(width: 20),
                  Text(
                    'Continue with Google',
                    style: TextStyle(
                      color: Colors.black,
                      fontFamily: 'Roboto',
                      fontSize: 20,
                    ),
                  ),
                ],
              ),
            ),
          ],
        ),
      ),
      Container(
        alignment: Alignment.center,
        padding: const EdgeInsets.all(10),
        child: const Text(
          'Sign in',
          style: TextStyle(fontSize: 20),
        ),
      ),
      Container(
        padding: const EdgeInsets.all(10),
        child: TextField(
          controller: emailController,
          decoration: const InputDecoration(
            border: OutlineInputBorder(),
            labelText: 'Email',
```

```
          ),
        ),
      ),
      Container(
        padding: const EdgeInsets.all(10),
        child: TextField(
          obscureText: true,
          controller: passwordController,
          decoration: const InputDecoration(
            border: OutlineInputBorder(),
            labelText: 'Password',
          ),
        ),
      ),
      TextButton(
        onPressed: () {
          // Forgot password action
        },
        child: const Text('Forgot Password'),
      ),
      Container(
        height: 50,
        padding: const EdgeInsets.fromLTRB(10, 0, 10, 0),
        child: ElevatedButton(
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.blue, // Set the background color to blue
),
          child: const Text('Login'),
          onPressed: () {
            print(emailController.text);
            print(passwordController.text);
          },
        ),
      ),
      Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          const Text('Does not have an account?'),
        ],
      ),
      Container(
        height: 50,
        padding: const EdgeInsets.fromLTRB(10, 0, 10, 0),
        child: ElevatedButton(
```

```
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.blue, // Set the background color to blue
),
            child: const Text('Sign Up'),
            onPressed: () {
              print(emailController.text);
              print(passwordController.text);
            },
          ),
        ),
      ],
    ),
  );
 }
}
```

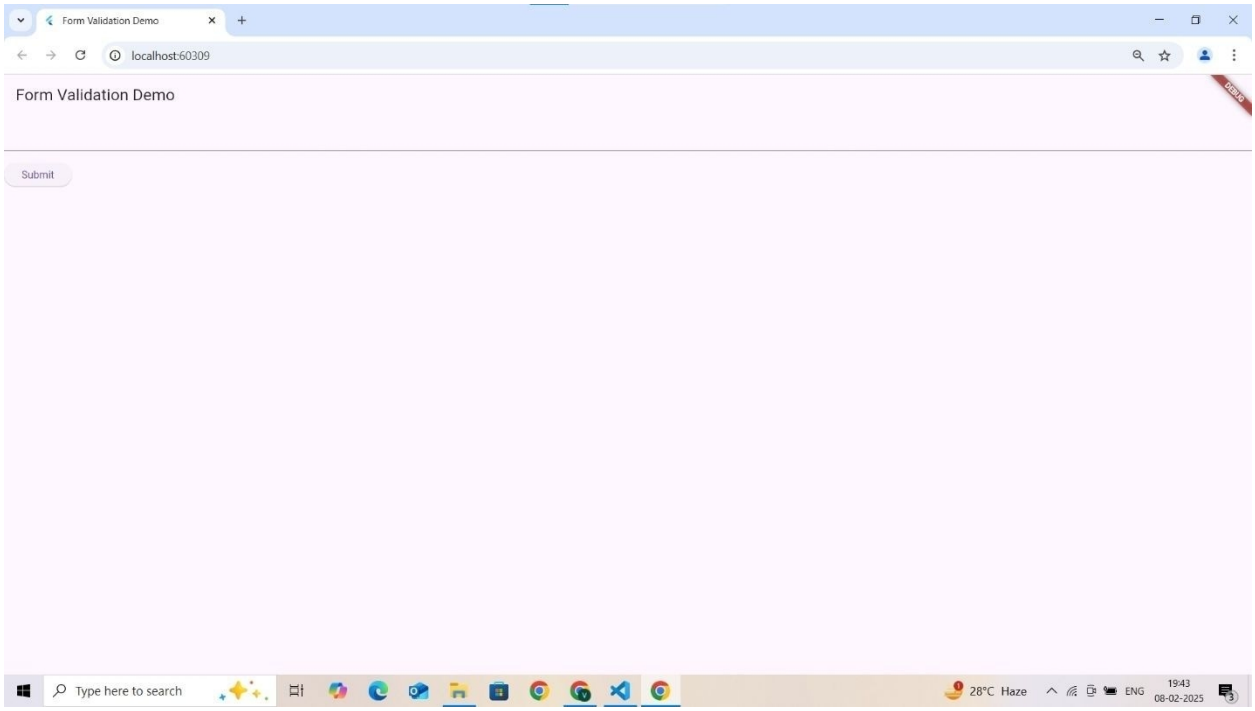


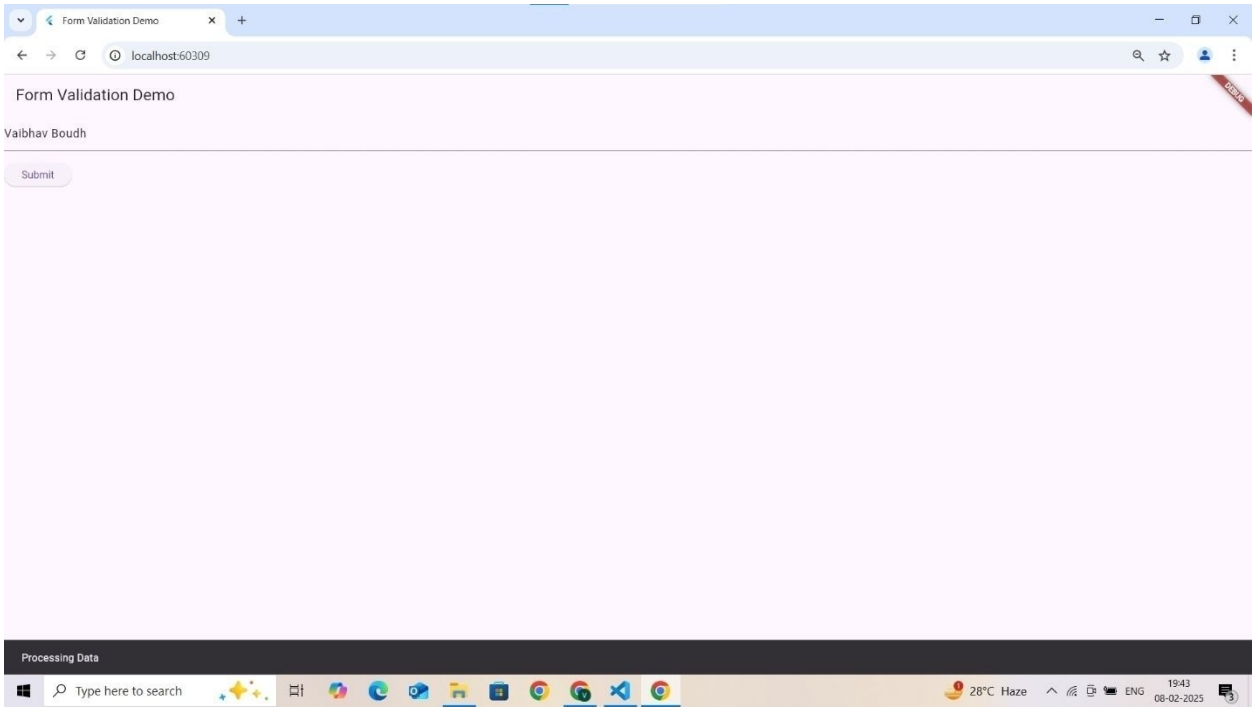

```
A Dart VM Service on Chrome is available at: http://127.0.0.1:60153/lN_NLVXpAAo=
The Flutter DevTools debugger and profiler on Chrome is available at: http://127.0.
2022.vaibhav.boudh@ves.ac.in
147852369
```

Create an interactive Validation page

```
import 'package:flutter/material.dart';
void main() => runApp(const MyApp());
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    const appTitle = 'Form Validation Demo';
    return MaterialApp(
      title: appTitle,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(appTitle),
        ),
        body: const MyCustomForm(),
      ),
    );
  }
}
// Create a Form widget.
class MyCustomForm extends StatefulWidget {
  const MyCustomForm({Key? key}) : super(key: key);
  @override
  MyCustomFormState createState() {
    return MyCustomFormState();
  }
}
// Create a corresponding State class.
class MyCustomFormState extends State<MyCustomForm> {
// Create a global key that uniquely identifies the Form widget
// and allows validation of the form.
  final _formKey = GlobalKey<FormState>();
  @override
  Widget build(BuildContext context) {
    // Build a Form widget using the _formKey created above.
    return Padding(
      padding: const EdgeInsets.all(16.0),
      child: Form(
        key: _formKey,
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            TextFormField(
              decoration: const InputDecoration(
```

```
              labelText: 'Enter some text',
              border: OutlineInputBorder(),
            ),
            // The validator receives the text that the user has entered.
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Please enter some text';
              }
              return null;
            },
          ),
          const SizedBox(height: 20),
          Center(
            child: ElevatedButton(
              onPressed: () {
                // Validate returns true if the form is valid, or false otherwise.
if (_formKey.currentState!.validate()) {
                  // If the form is valid, display a snackbar.
                  ScaffoldMessenger.of(context).showSnackBar(
                    const SnackBar(content: Text('Processing Data')),
                  );
                }
              },
              child: const Text('Submit'),
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```


Form Validation Demo
localhost:60309
Form Validation Demo
DEBUG
Submit
Type here to search
28°C Haze
ENG
19:43
08-02-2025


Form Validation Demo
localhost:60309
Form Validation Demo
DEBUG
Vaibhav Boudh
Submit
Processing Data
Type here to search
28°C Haze
ENG
19:43
08-02-2025

MAD Ex 5

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

Theory:

Navigation, routing, and gestures are essential for building interactive and user-friendly Flutter applications. Navigation allows users to move between screens using the Navigator widget, which manages a stack of routes. Routing can be implemented using named routes or direct push and pop methods. Named routes help in organizing navigation, making the app structure more manageable.

Gestures enhance user interaction by detecting taps, swipes, and drags using the GestureDetector widget. Common gestures include tapping for button clicks, swiping for navigation, and pressing for additional actions. Flutter's built-in gesture recognition makes UI interactions smooth and intuitive.

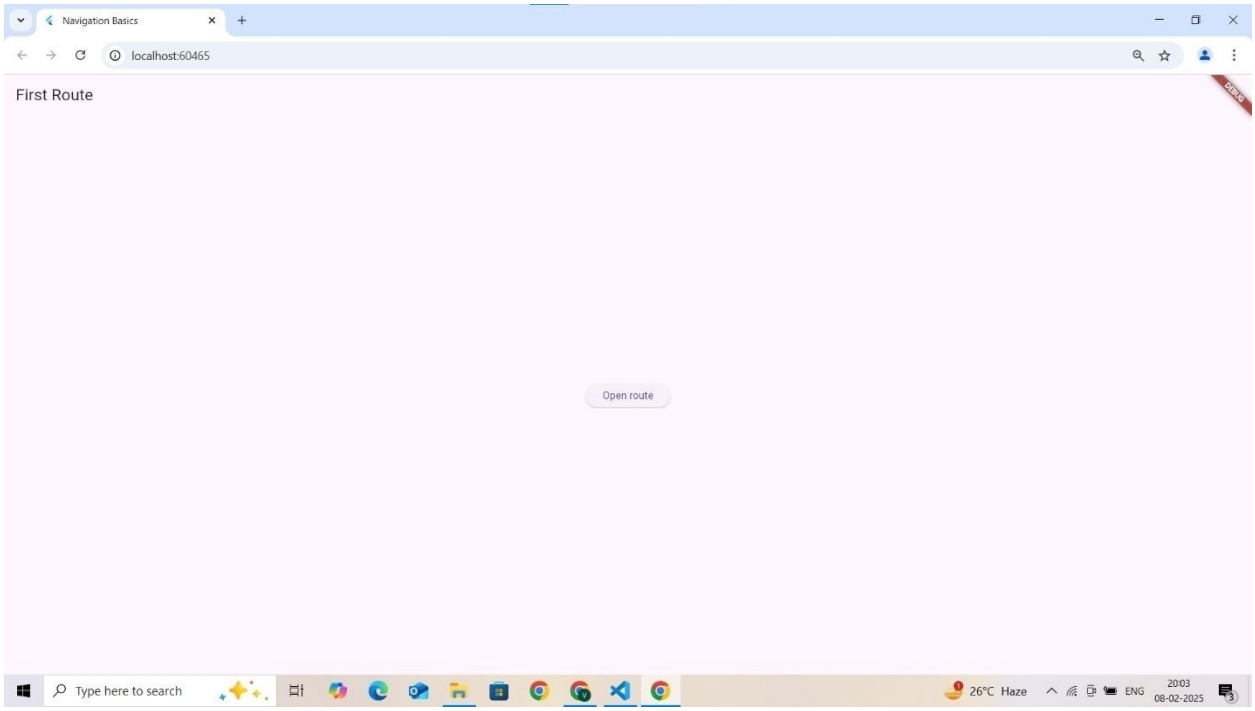
By effectively implementing navigation, routing, and gestures, Flutter apps become more dynamic, resulting in seamless transitions and an enhanced user experience.

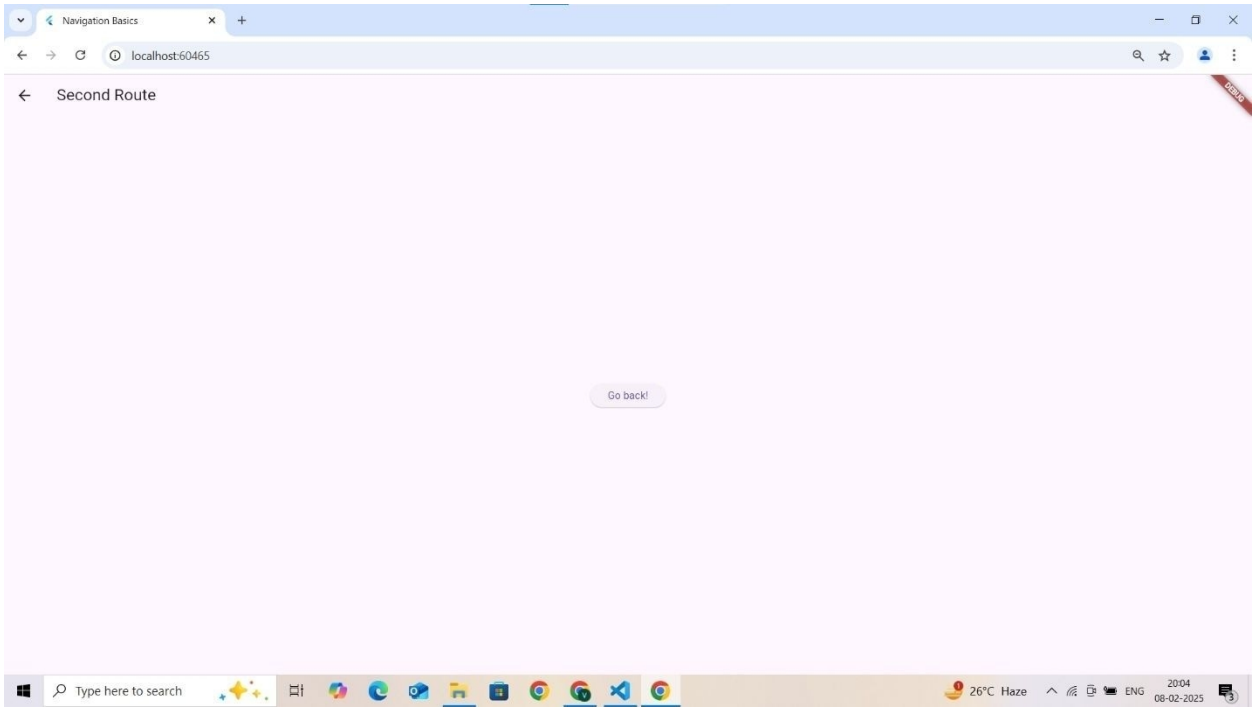
Conclusion:

Implementing navigation, routing, and gestures in a Flutter app enhances user experience by enabling smooth screen transitions and interactive controls. The Navigator and named routes simplify navigation, while the GestureDetector widget allows intuitive touch interactions. Properly integrating these features ensures a seamless, responsive, and user-friendly app.

Exp 5 To apply navigation, routing and gestures in Flutter App

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MaterialApp(
    title: 'Navigation Basics',
    home: FirstRoute(),
  ));
}
class FirstRoute extends StatelessWidget {
  const FirstRoute({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Route'),
      ),
      body: Center(
        child: ElevatedButton(
          child: const Text('Open route'),
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => const SecondRoute()),
);
          },
        ),
      ),
    );
  }
}
class SecondRoute extends StatelessWidget {
  const SecondRoute({Key? key}) : super(key: key);
 @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Second Route'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: const Text('Go back!'),
        ),
      ),
    );
  }
}
```

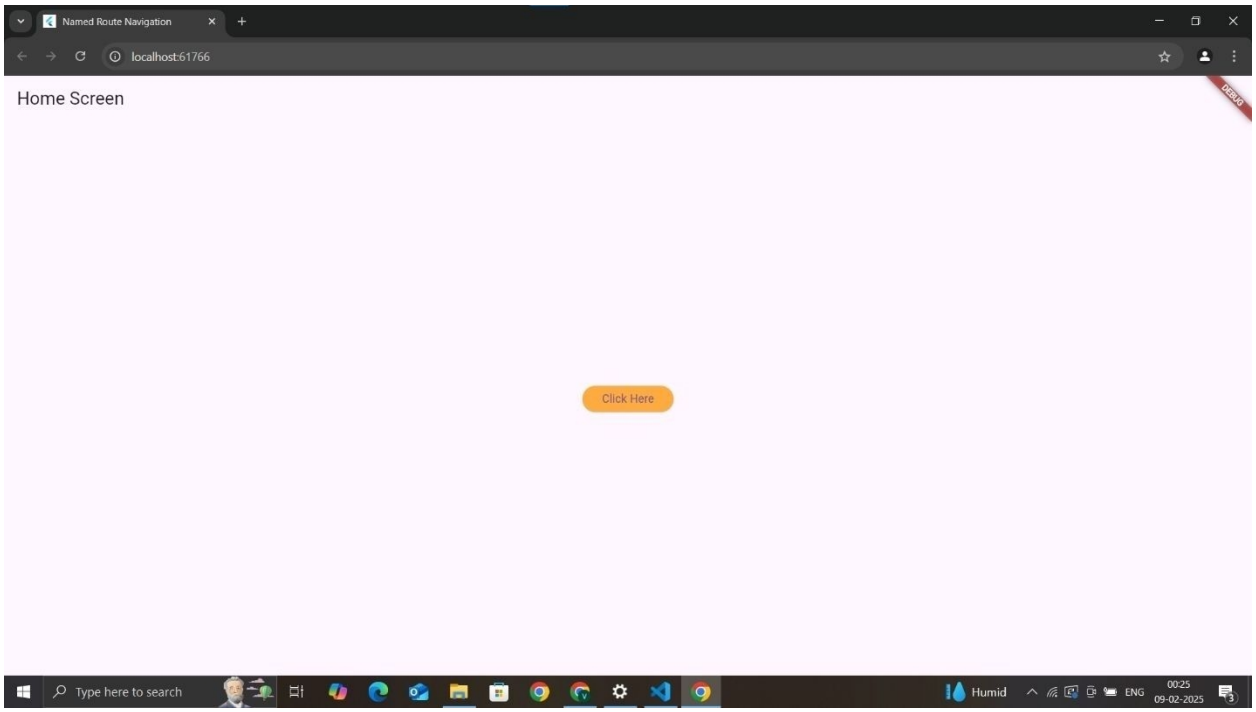

Navigation Basics
localhost:60465
First Route
DEBUG
Open route
Type here to search
26°C Haze
ENG
20:03
08-02-2025

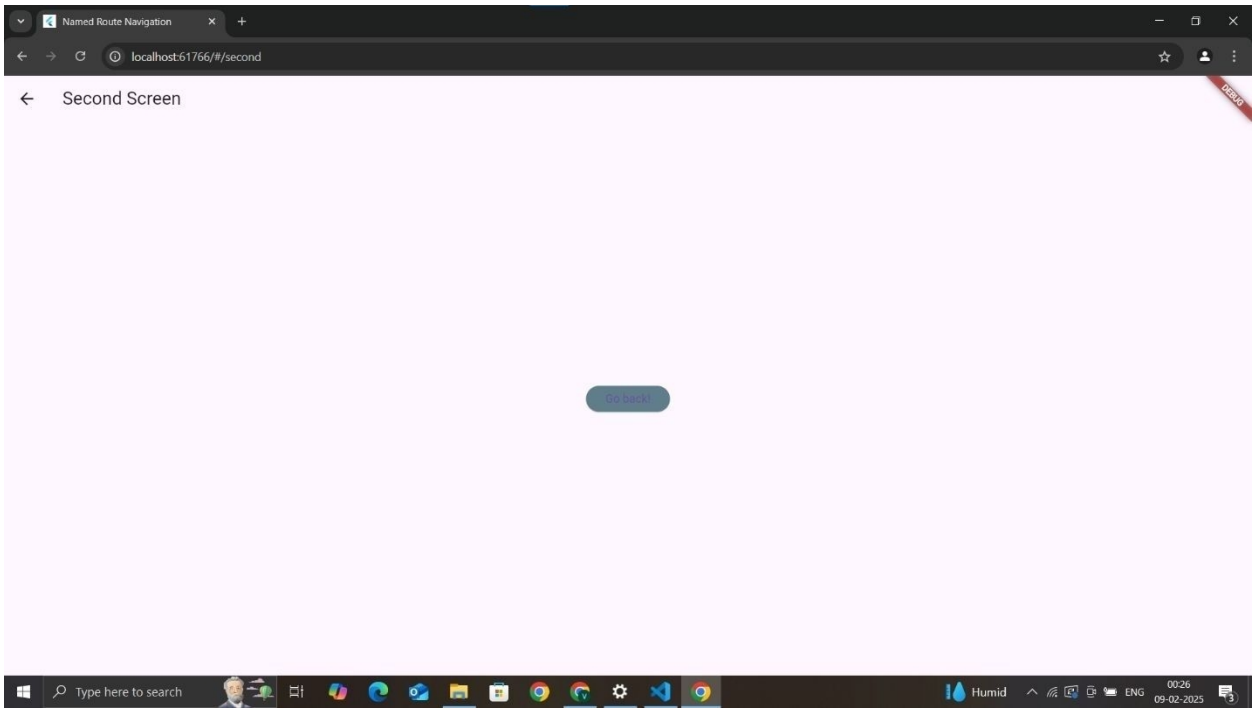

Navigation Basics
localhost:60465
Second Route
DEBUG
Go back!
Type here to search
26°C Haze
ENG
20:04
08-02-2025

//Routing Navigation with Named Routes

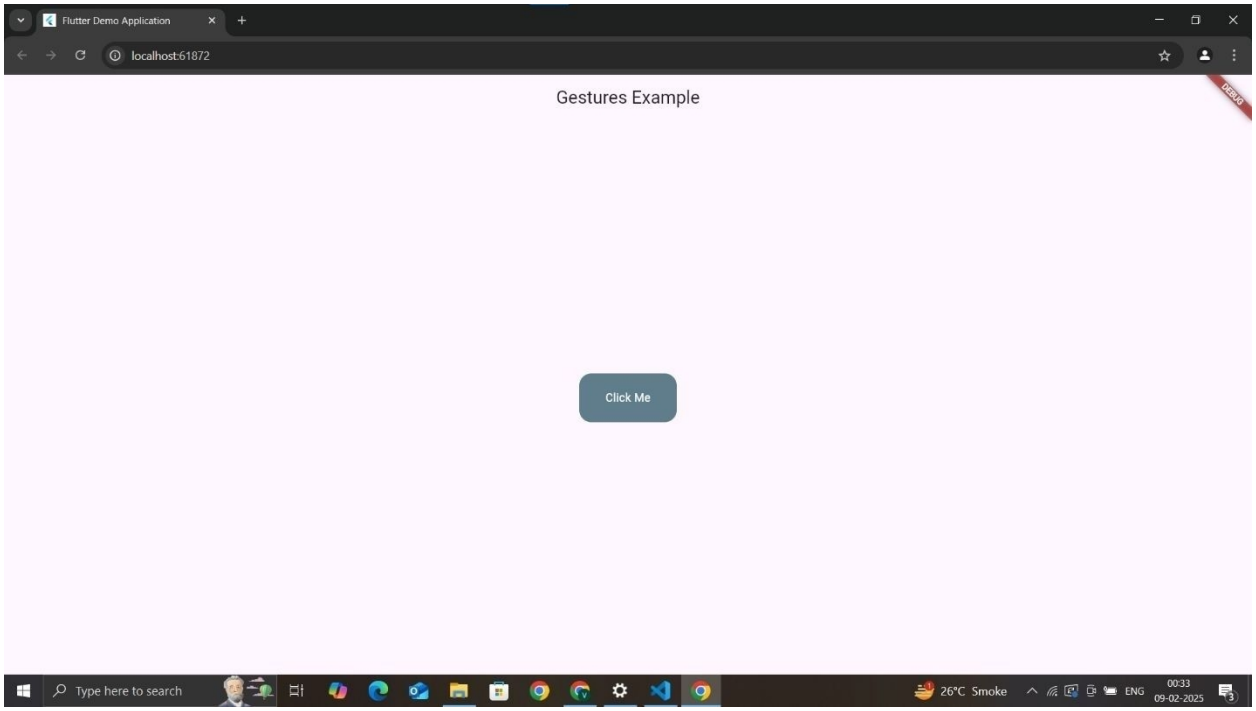
```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(
    title: 'Named Route Navigation',
    theme: ThemeData(
      primarySwatch: Colors.green,
    ),
    // Start the app with the "/" named route.
    initialRoute: '/',
    routes: {
      // When navigating to the "/" route, build the HomeScreen widget.
      '/': (context) => HomeScreen(),
      // When navigating to the "/second" route, build the SecondScreen widget.
'/second': (context) => SecondScreen(),
    },
  ));
}
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Home Screen'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pushNamed(context, '/second');
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.orangeAccent,
          ),
          child: const Text('Click Here'),
        ),
      ),
    );
  }
}
class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Second Screen"),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pop(context);
          },
```

```
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.blueGrey,
),
          child: const Text('Go back!'),
        ),
      ),
    );
  }
}
```



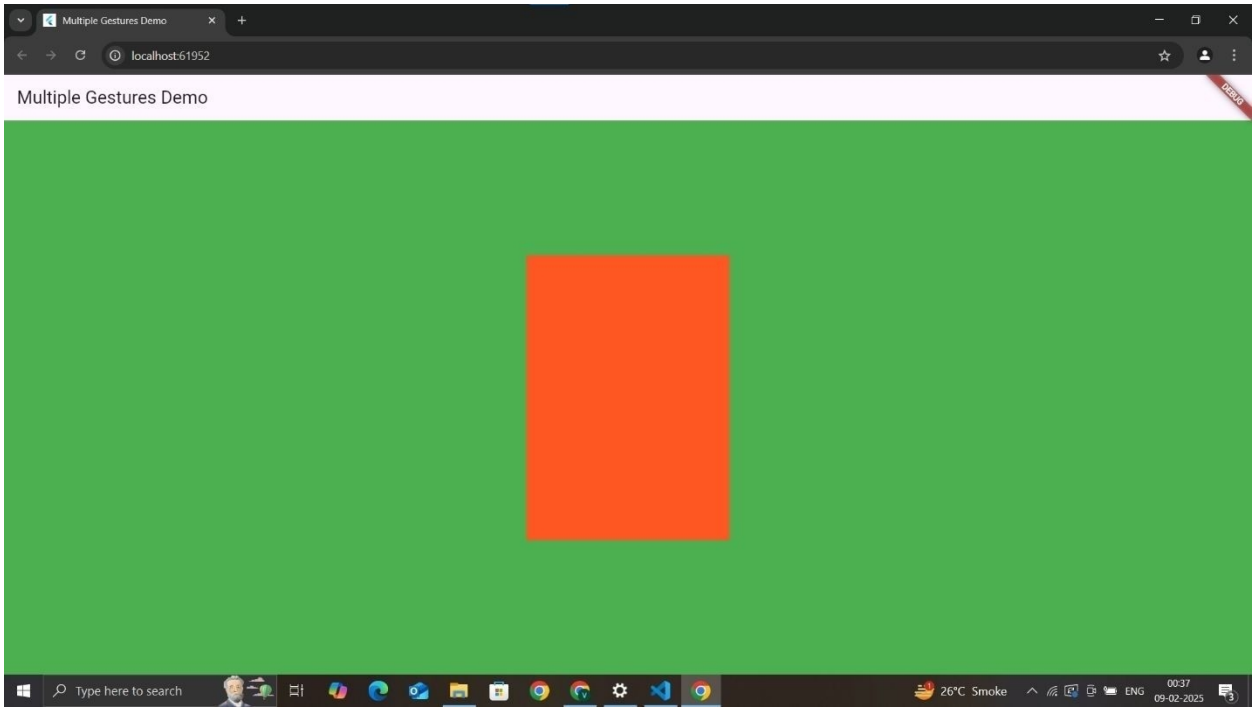


```
//Gesture Example
import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo Application',
      theme: ThemeData(
        primarySwatch: Colors.green,
      ),
      home: MyHomePage(),
    );
  }
}
class MyHomePage extends StatefulWidget {
  @override
  MyHomePageState createState() => MyHomePageState(); }
class MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Gestures Example'),
        centerTitle: true,
      ),
      body: Center(
        child: GestureDetector(
          onTap: () {
            print('Box Clicked');
          },
          child: Container(
            height: 60.0,
            width: 120.0,
            padding: EdgeInsets.all(10.0),
            decoration: BoxDecoration(
              color: Colors.blueGrey,
              borderRadius: BorderRadius.circular(15.0),
            ),
            child: Center(
              child: Text(
                'Click Me',
                style: TextStyle(color: Colors.white),
              ),
            ),
          ),
        ),
      ),
    );
  }}
```


Flutter Demo Application
localhost:61872
Gestures Example
DEBUG
Click Me
Type here to search
26°C Smoke
ENG
00:33
09-02-2025

```
// Multiple Gesture Example
import 'package:flutter/gestures.dart';
import 'package:flutter/material.dart';
// Entry point for the Flutter app.
void main() {
  runApp(
    MaterialApp(
      title: 'Multiple Gestures Demo',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Multiple Gestures Demo'),
        ),
        body: DemoApp(),
      ),
    ),
  );
}
class DemoApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return RawGestureDetector(
      gestures: {
        AllowMultipleGestureRecognizer:
            GestureRecognizerFactoryWithHandlers<
                AllowMultipleGestureRecognizer>(
          () => AllowMultipleGestureRecognizer(),
          (AllowMultipleGestureRecognizer instance) {
            instance.onTap =
                () => print('It is the parent container gesture');
          },
        ),
      },
      behavior: HitTestBehavior.opaque,
          // Parent Container
      child: Container(
        color: Colors.green,
        child: Center(
          // Now, wraps the second container in RawGestureDetector
child: RawGestureDetector(
            gestures: {
              AllowMultipleGestureRecognizer:
                  GestureRecognizerFactoryWithHandlers<
                      AllowMultipleGestureRecognizer>(
                () => AllowMultipleGestureRecognizer(), // Constructor
                (AllowMultipleGestureRecognizer instance) { // Initializer
instance.onTap =
                      () => print('It is the nested container');
                },
              ),
            },
```

```
          // Creates the nested container within the first
          child: Container(
            color: Colors.deepOrange,
            width: 250.0,
            height: 350.0,
          ),
        ),
      ),
    ),
  );
 }
}
// Custom Gesture Recognizer to allow multiple gestures
class AllowMultipleGestureRecognizer extends TapGestureRecognizer {
@override
 void rejectGesture(int pointer) {
  acceptGesture(pointer);
 }
}
```



Experiment No: 06

How To Set Up Firebase with Flutter for Android Apps

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

In this article, you will create a Firebase project for iOS and Android platforms using Flutter.

Prerequisites

To complete this tutorial, you will need:

- A Google account to use Firebase.
- Developing for iOS will require XCode.
- To download and install Flutter.
- To download and install Android Studio and Visual Studio Code.
- It is recommended to install plugins for your code editor:
    - `Flutter` and `Dart` plugins installed for Android Studio.
    - `Flutter` extension installed for Visual Studio Code.

Creating a New Flutter Project

This tutorial will require the creation of an example Flutter app.
Once you have your environment set up for Flutter, you can run the following to create a new application:

```
flutter create flutterfirebaseexample
```

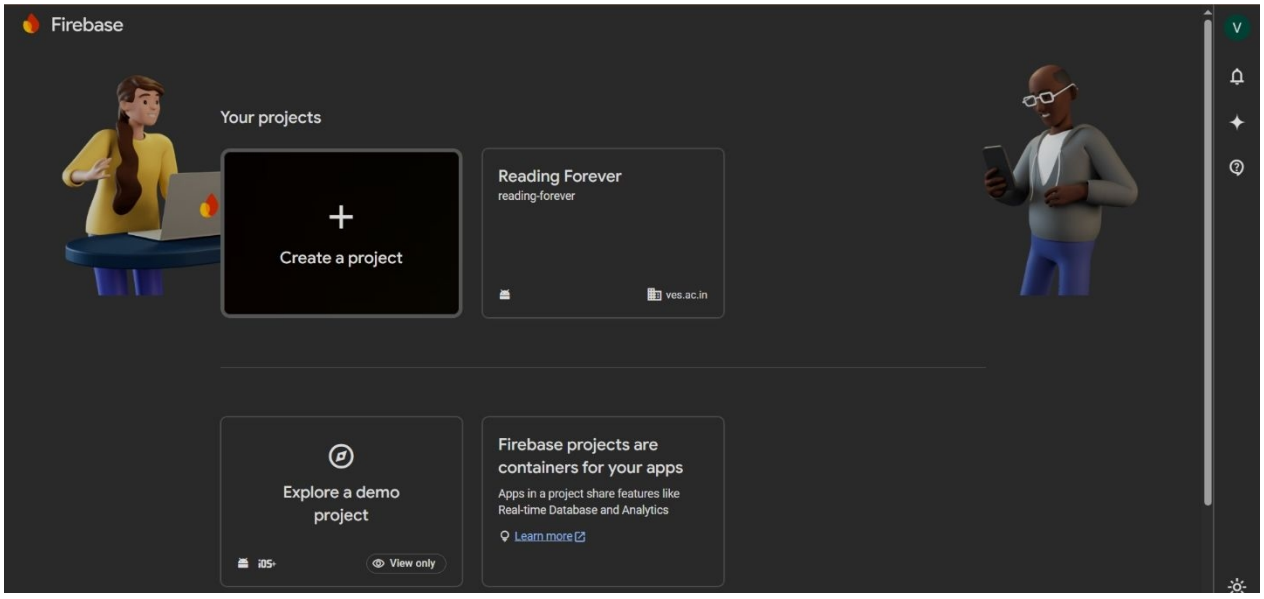
Navigate to the new project directory:

```
cd flutterfirebaseexample
```

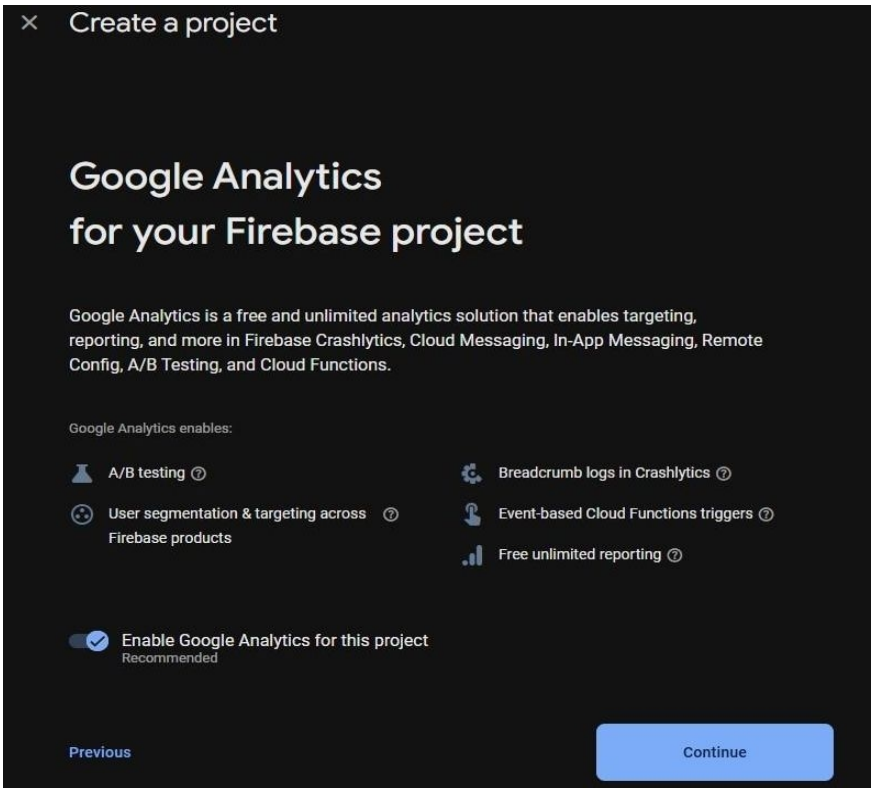
Using `flutter create` will produce a demo application that will display the number of times a button is clicked.
Now that we've got a Flutter project up and running, we can add Firebase.

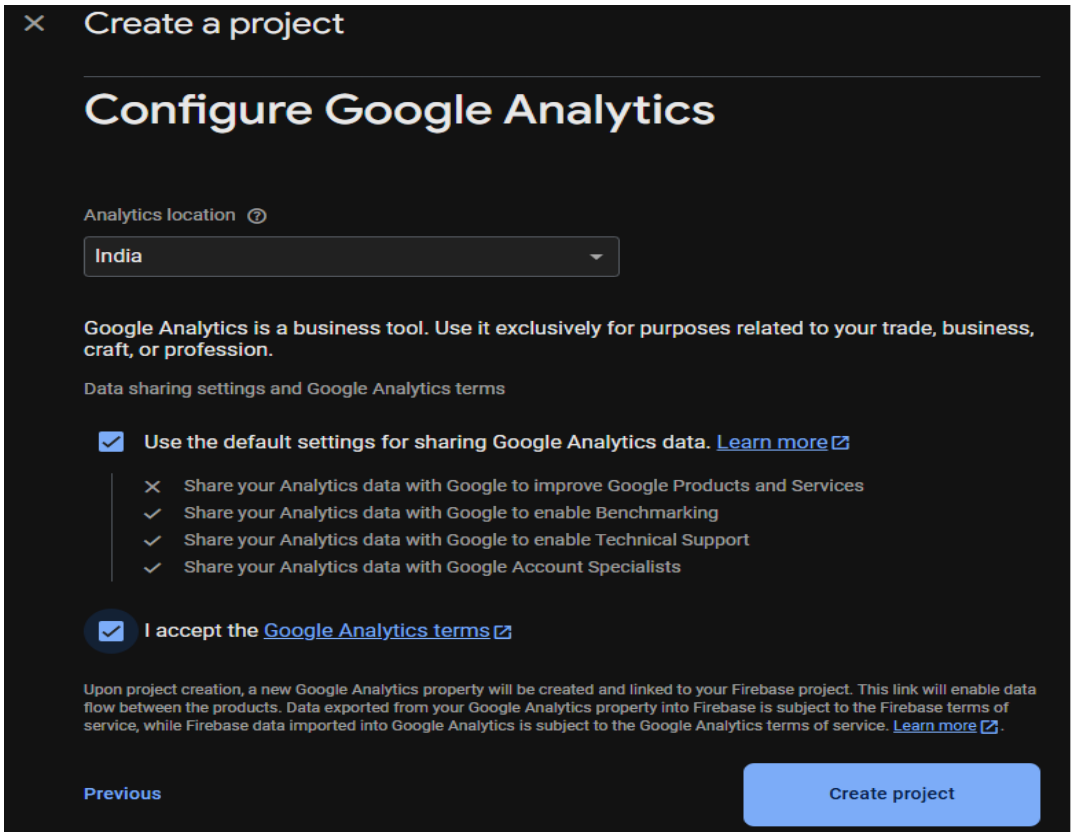
Creating a New Firebase Project

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



Next, we're given the option to enable Google Analytics. This tutorial will not require Google Analytics, but you can also choose to add it to your project.

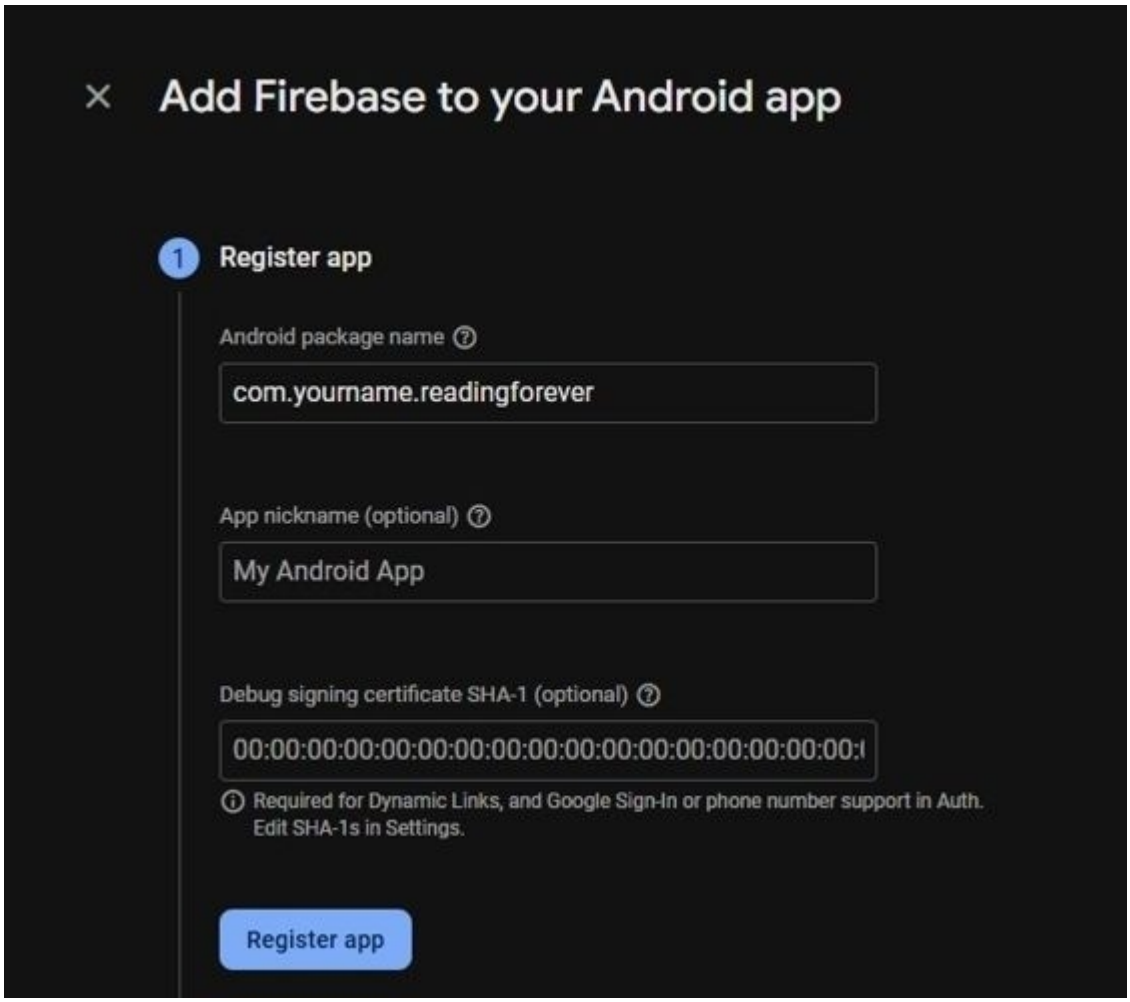




Adding Android support

Registering the App

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



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

The structure consists of at least two segments. A common pattern is to use a domain name, a company name, and the application name:

`com.example.flutterfirebaseexample`

Once you've decided on a name, open `android/app/build.gradle` in your code editor and update the `applicationId` to match the Android package name:

android/app/build.gradle

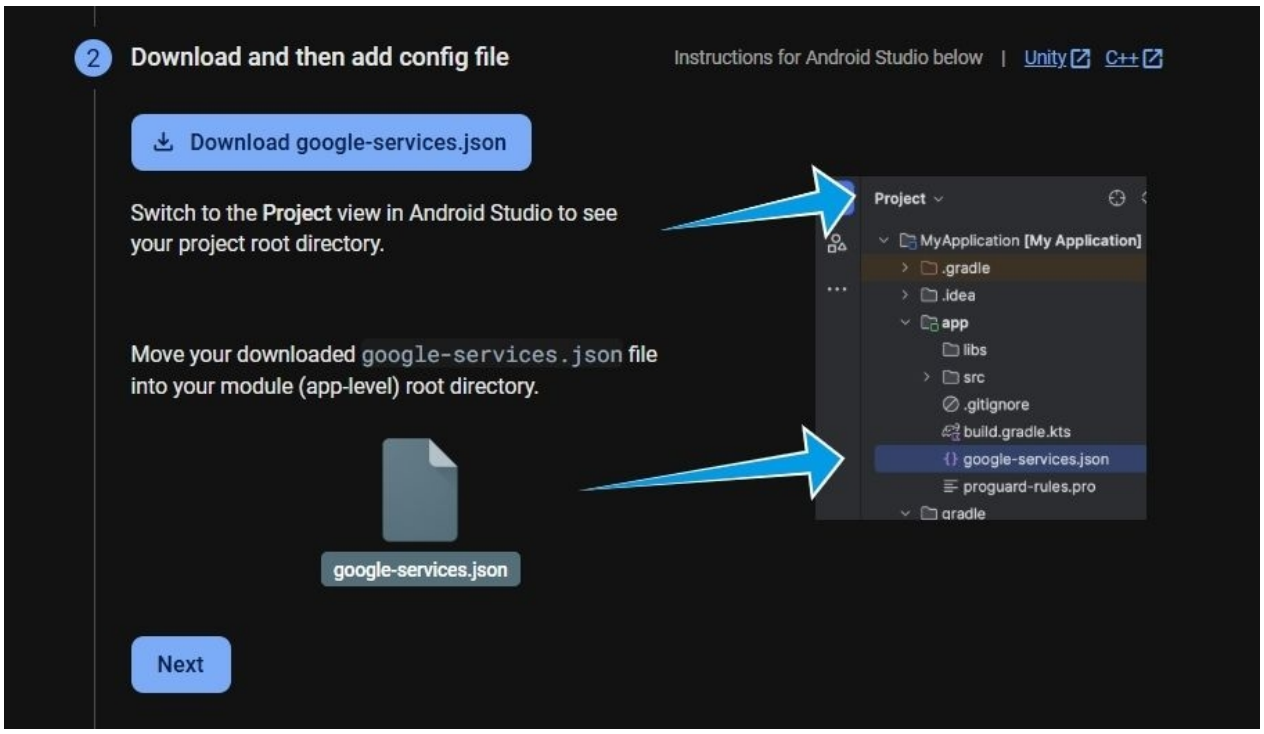
```
...
defaultConfig {
 // TODO: Specify your own unique Application ID
(https://developer.android.com/studio/build/application-id.html)
.  applicationId 'com.example.flutterfirebaseexample'   ...
}
...
```

You can skip the app nickname and debug signing keys at this stage. Select Register app to continue.

Downloading the Config File

The next step is to add the Firebase configuration file into our Flutter project. This is important as it contains the API keys and other critical information for Firebase to use.

Select Download `google-services.json` from this page:



Next, move the `google-services.json` file to the `android/app` directory within the Flutter project.

Adding the Firebase SDK

We'll now need to update our Gradle configuration to include the Google Services plugin.

Open `android/build.gradle` in your code editor and modify it to include the following:

android/build.gradle

```
buildscript {
 repositories {
 // Check that you have the following line (if not, add it):
google() // Google's Maven repository
 }
 dependencies {
 ...
 // Add this line
 classpath 'com.google.gms:google-services:4.3.6'
 }
}

allprojects {
 ...
 repositories {
 // Check that you have the following line (if not, add it):
google() // Google's Maven repository
 ...
 }
}
```

Finally, update the app level file at `android/app/build.gradle` to include the following:

android/app/build.gradle

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
 // Import the Firebase BoM
 implementation platform('com.google.firebase:firebase-bom:28.0.0')

 // Add the dependencies for any other desired Firebase products  //
https://firebase.google.com/docs/android/setup#available-libraries }
```

With this update, we're essentially applying the Google Services plugin as well as looking at how other Flutter Firebase plugins can be activated such as Analytics.

From here, run your application on an Android device or simulator. If everything has worked correctly, you should get the following message in the dashboard:

3 Add Firebase SDK

Instructions for Gradle | Unity C++

> ★ Are you still using the `buildscript` syntax to manage plugins? Learn how to add Firebase plugins using that syntax.

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

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

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

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

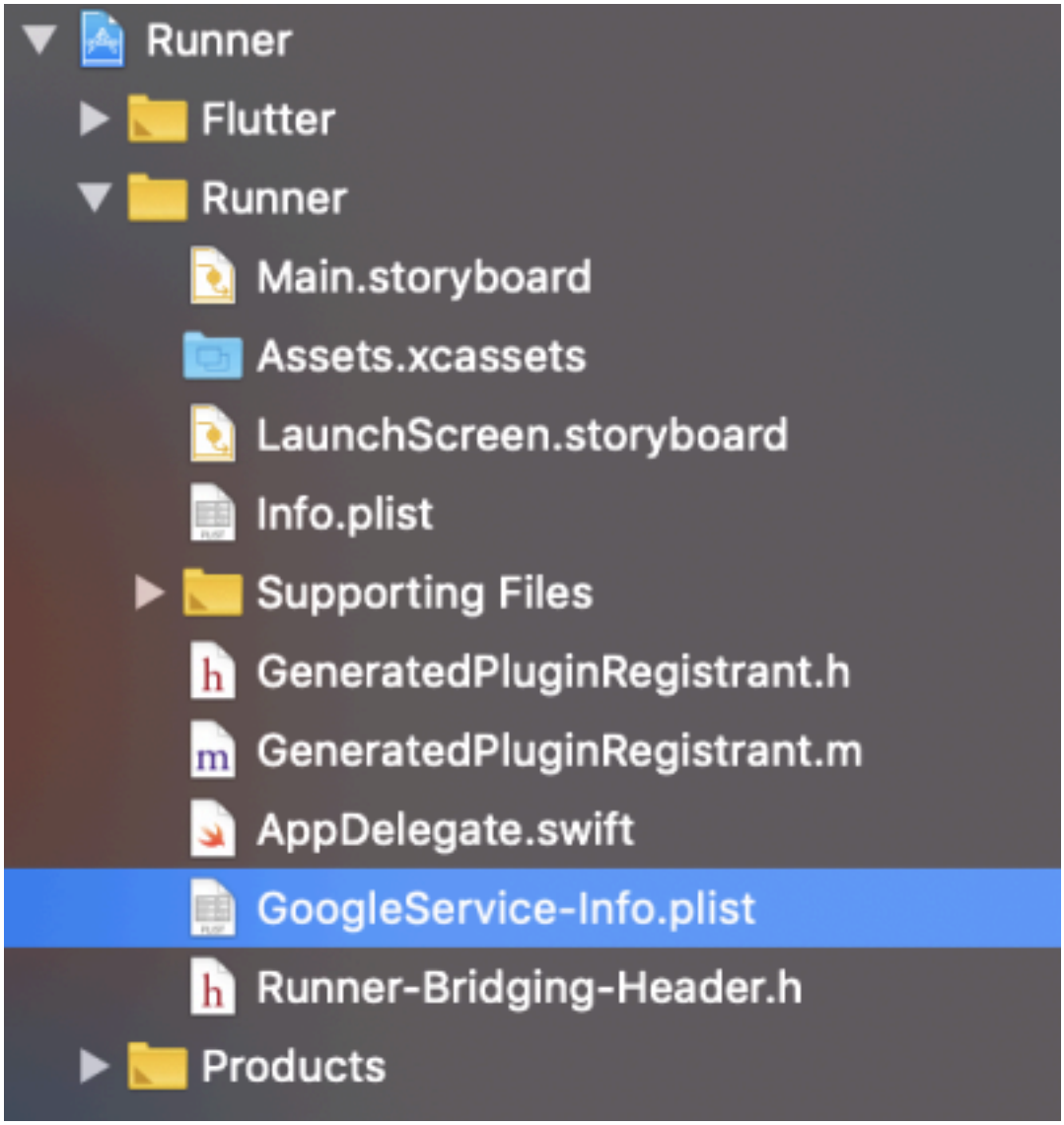
```
plugins {
  // ...

  // Add the dependency for the Google services Gradle plugin
  id 'com.google.gms.google-services' version '4.4.2' apply false
}
```

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

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

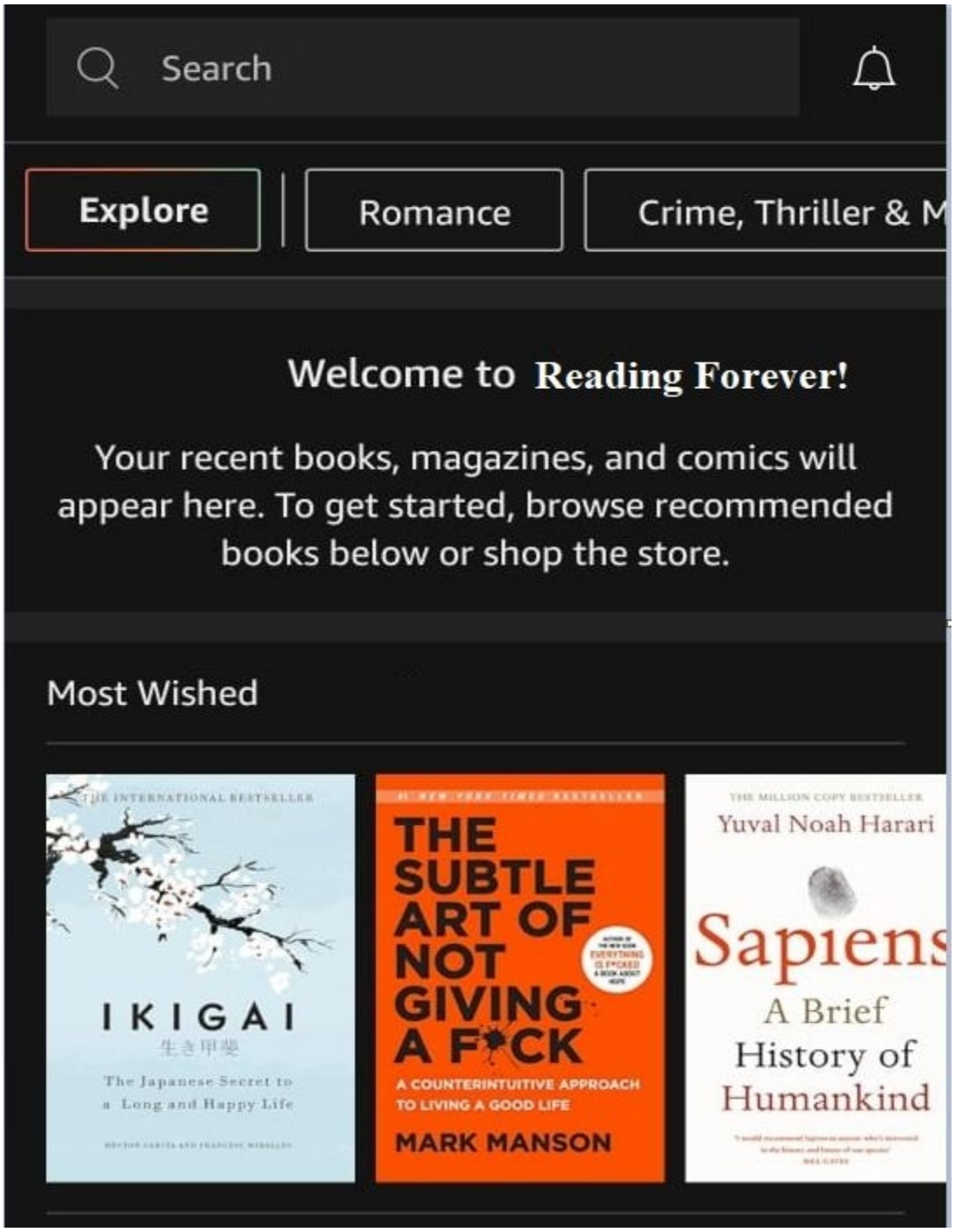
```
plugins {
  id 'com.android.application'
  // Add the Google services Gradle plugin
  id 'com.google.gms.google-services'
  ...
}

dependencies {
  // Import the Firebase BoM
  implementation platform('com.google.firebase:firebase-bom:33.10.0')

  // TODO: Add the dependencies for Firebase products you want to use
  // When using the BoM, don't specify versions in Firebase dependencies
  implementation 'com.google.firebase:firebase-analytics'

  // Add the dependencies for any other desired Firebase products
  // https://firebase.google.com/docs/android/setup#available-libraries
}
```

S.



Be sure to move this file within Xcode to create the proper file references.
There are additional steps for installing the Firebase SDK and adding initialization code, but they are not necessary for this tutorial.

After Setup



Conclusion

In this article, you learned how to set up and ready our Flutter applications to be used with Firebase.
Flutter has official support for Firebase with the FlutterFire set of libraries.

Experiment 7:-

Implementation of Add to Home Screen Feature in Progressive Web Apps (PWA)

Theory:-

What is a Progressive Web App (PWA)?

A Progressive Web App (PWA) is a type of web application that provides a native app-like experience while running in a browser. It can work offline, load fast, and be installed on a user's device just like a mobile app.

Key Features of a PWA:

1. Responsive Design – Works on different screen sizes (mobile, tablet, desktop).
2. Service Worker – A background script that enables offline functionality and caching.
3. Web App Manifest – A JSON file that defines app metadata like name, icon, and theme color.
4. Secure (HTTPS) – Ensures a secure connection for improved user trust.
5. Add to Home Screen (A2HS) – Allows users to install the app on their home screen.

Add to Home Screen (A2HS) Functionality:

The A2HS feature enables users to install a PWA on their home screen without using an app store. This enhances accessibility and user engagement. It requires:

- A valid web app manifest (manifest.json)
- A service worker (serviceworker.js)
- HTTPS hosting

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="theme-color" content="#ffffff">
    <title>Simple E-commerce PWA</title>
    <link rel="manifest" href="/manifest.json">
    <link rel="icon" href="https://via.placeholder.com/192" type="image/png">
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background: #f0f0f0;
        }
        .product-grid {
            display: grid;
            grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
            gap: 20px;
            max-width: 1200px;
            margin: 0 auto;
        }
        .product-card {
            background: white;
            padding: 15px;
```

```
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
            text-align: center;
        }
        .product-card img {
            max-width: 100%;
            height: 150px;
            object-fit: cover;
        }
        button {
            background: #007bff;
            color: white;
            border: none;
            padding: 10px 20px;
            border-radius: 5px;
            cursor: pointer;
        }
        button:hover {
            background: #0056b3;
        }
        #notification {
            position: fixed;
            bottom: 20px;
            right: 20px;
            background: #28a745;
            color: white;
            padding: 10px 20px;
            border-radius: 5px;
            display: none;
        }
    </style>
</head>
<body>
    <h1>Simple E-commerce PWA</h1>
    <div class="product-grid" id="products"></div>
    <div id="notification">Item added to cart!</div>

    <script>
        // Sample product data
        const products = [
            { id: 1, name: "T-Shirt", price: 19.99, image:
"https://via.placeholder.com/200" },
            { id: 2, name: "Jeans", price: 39.99, image:
"https://via.placeholder.com/200" },
            { id: 3, name: "Shoes", price: 59.99, image:
"https://via.placeholder.com/200" }
        ];

        // Render products
        function renderProducts() {
            const grid = document.getElementById('products');
            grid.innerHTML = products.map(product =>
                <div class="product-card">
                    <img src="${product.image}" alt="${product.name}">
```

```
                    <h3>${product.name}</h3>
                    <p>$${product.price}</p>
                    <button onclick="addToCart(${product.id})">Add to
Cart</button>
                </div>
            ).join('');
        }

        // Add to cart simulation
        function addToCart(productId) {
            const notification = document.getElementById('notification');
            notification.style.display = 'block';
            setTimeout(() => notification.style.display = 'none', 2000);
        }

        // Register service worker
        if ('serviceWorker' in navigator) {
            window.addEventListener('load', () => {
                navigator.serviceWorker.register('/sw.js')
                    .then(reg => {
                        console.log('Service Worker registered', reg);
                    })
                    .catch(err => {
                        console.log('Service Worker registration failed:', err);
                    });
            });
        }

        renderProducts();
    </script>
</body>
</html>
```

manifest.json

```
{
    "name": "Simple E-commerce PWA",
    "short_name": "E-commerce",
    "start_url": "/index.html",
    "display": "standalone",
    "background_color": "#ffffff",
    "theme_color": "#ffffff",
    "icons": [
        {
            "src": "https://via.placeholder.com/192",
            "sizes": "192x192",
            "type": "image/png"
        },
        {
            "src": "https://via.placeholder.com/512",
            "sizes": "512x512",
            "type": "image/png"
        }
    ]
```

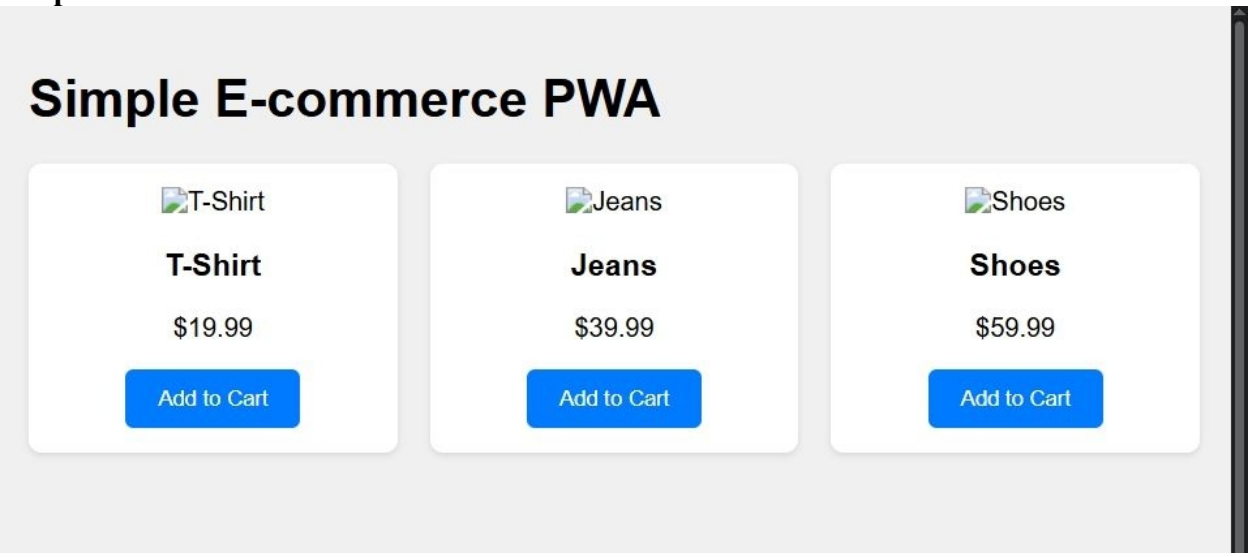
```
}
```

Serviceworker.js

```
const CACHE_NAME = 'ecommerce-pwa-v1';
const urlsToCache = [
    '/',
    '/index.html',
    '/manifest.json'
];

// Install event
self.addEventListener('install', event => {
    event.waitUntil(
        caches.open(CACHE_NAME)
            .then(cache => {
                return cache.addAll(urlsToCache);
            })
    );
});

// Activate event
self.addEventListener('activate', event => {
    event.waitUntil(
        caches.keys().then(cacheNames => {
            return Promise.all(
                cacheNames.filter(name => name !== CACHE_NAME)
                    .map(name => caches.delete(name))
            );
        })
    );
});

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

// Sync event
self.addEventListener('sync', event => {
    if (event.tag === 'sync-cart') {
        event.waitUntil(
            // Add your sync logic here
            console.log('Background sync triggered')
        );
    }
```

```
});

// Push event
self.addEventListener('push', event => {
    const data = event.data.json();
    const options = {
        body: data.body,
        icon: 'https://via.placeholder.com/192'
    };
    event.waitUntil(
        self.registration.showNotification(data.title, options)
    );
});
```

Output:-



Conclusion:-

In this experiment, we successfully developed a Progressive Web App (PWA) that includes the Add to Home Screen (A2HS) feature. The application can be installed on a user's device, providing an enhanced user experience similar to a native app. This demonstrates how PWAs can bridge the gap between web and mobile applications while remaining lightweight and accessible.

Experiment No: 08

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

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

Things to note about Service Worker:

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

What can we do with Service Workers?

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

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

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

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

What can’t we do with Service Workers?

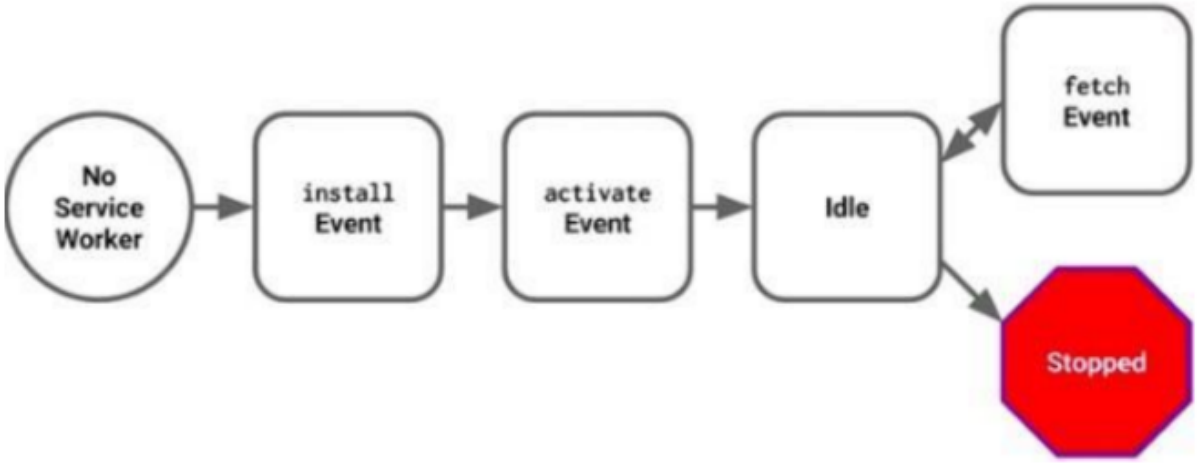
● You can’t access the **Window**

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

● You can’t work it on **80 Port**

Service Worker just can work on HTTPS protocol. But you can work on localhost during development.

Service Worker Cycle

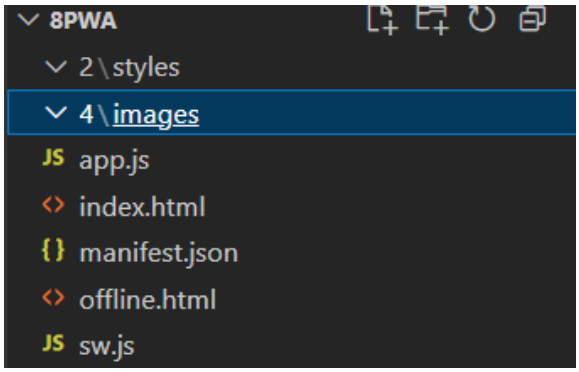


A service worker goes through three steps in its life cycle:

● Registration
● Installation
● Activation

Registration

To install a service worker, you need to register it in your main JavaScript code. Registration tells the browser
where your service worker is located, and to start installing it in the background. Let's look at an example:



app.js

```
// Service Worker Registration
if ('serviceWorker' in navigator) {
    window.addEventListener('load', () => {
      navigator.serviceWorker.register('/sw.js')
        .then(registration => {
          console.log('SW registered: ', registration);
        })
        .catch(registrationError => {
          console.log('SW registration failed: ', registrationError);
});
    });
  }
  // Listen for controller changes (updates)
  navigator.serviceWorker.addEventListener('controllerchange', () => {
console.log('New service worker activated!');
    window.location.reload();
  });
```

sw.js

```
const CACHE_NAME = 'ecommerce-pwa-v1';
const OFFLINE_URL = '/offline.html';
const ASSETS_TO_CACHE = [
  '/',
  '/index.html',
  '/styles/main.css',
  '/scripts/app.js',
  '/images/logo.png',
  OFFLINE_URL,
  // Add other critical assets for your e-commerce site
];
// Install event - caching essential assets
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        console.log('Opened cache');
        return cache.addAll(ASSETS_TO_CACHE);
      })
      .then(() => self.skipWaiting()) // Force the waiting service worker to become
active
  );
});
// Activate event - clean up old caches
self.addEventListener('activate', event => {
  const cacheWhitelist = [CACHE_NAME];
  event.waitUntil(
    caches.keys().then(cacheNames => {
      return Promise.all(
        cacheNames.map(cacheName => {
          if (cacheWhitelist.indexOf(cacheName) === -1) {
            return caches.delete(cacheName);
          }
```

```
        })
```

```
      );
    })
    .then(() => self.clients.claim()) // Take control of all clients immediately
  );
});
// Fetch event - network with cache fallback
self.addEventListener('fetch', event => {
  // Skip cross-origin requests
  if (!event.request.url.startsWith(self.location.origin)) {
    return;
  }
  // For API calls, use network-first strategy
  if (event.request.url.includes('/api/')) {
    event.respondWith(
      fetch(event.request)
        .then(response => {
          // Cache the API response if successful
          const responseToCache = response.clone();
          caches.open(CACHE_NAME)
            .then(cache => cache.put(event.request, responseToCache));
          return response;
        })
        .catch(() => {
          // If network fails, try to get from cache
          return caches.match(event.request);
        })
    );
  } else {
    // For static assets, use cache-first strategy
    event.respondWith(
      caches.match(event.request)
        .then(cachedResponse => {
          // Return cached response if found
          if (cachedResponse) {
            return cachedResponse;
          }
          // Otherwise fetch from network
          return fetch(event.request)
            .then(response => {
              // Check if we received a valid response
              if (!response || response.status !== 200 || response.type !== 'basic') {
                return response;
              }
              // Clone the response
              const responseToCache = response.clone();

              caches.open(CACHE_NAME)
                .then(cache => {
                  cache.put(event.request, responseToCache);
});

              return response;

            })
            .catch(() => {
```

```
            // If both fail, show offline page for HTML requests
            if (event.request.headers.get('accept').includes('text/html')) {
return caches.match(OFFLINE_URL);
              }
            });
        })
    );
  }
});
// Push notification event handling
self.addEventListener('push', event => {
  const data = event.data.json();
  const title = data.title || 'New update from our store!';
  const options = {
    body: data.body || 'Check out our latest products and offers!',
    icon: '/images/icon-192x192.png',
    badge: '/images/badge-72x72.png'
  };
  event.waitUntil(
    self.registration.showNotification(title, options)
  );
});
self.addEventListener('notificationclick', event => {
  event.notification.close();
  event.waitUntil(
    clients.openWindow('https://your-ecommerce-site.com/latest-offers')
  );
});
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta name="theme-color" content="#ffffff" />
  <title>Simple E-commerce PWA</title>
  <link rel="manifest" href="/manifest.json" />
  <link rel="icon" href="https://via.placeholder.com/192" type="image/png" />
<link rel="stylesheet" href="/styles/main.css" />
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 20px;
      background: #f0f0f0;
    }

    h1 {
      text-align: center;
}
```

```
    .product-grid {
      display: grid;
      grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
gap: 20px;
      max-width: 1200px;
      margin: 0 auto;
    }

    .product-card {
      background: white;
      padding: 15px;
      border-radius: 8px;
      box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
text-align: center;
    }

    .product-card img {
max-width: 100%;
height: 150px;
      object-fit: cover;
}

    button {
      background: #007bff;
color: white;
      border: none;
      padding:  10px  20px;
border-radius:         5px;
cursor: pointer;
    }

    button:hover {
      background: #0056b3;
}

    #notification {
      position: fixed;
      bottom: 20px;
      right: 20px;
      background: #28a745;
      color: white;
      padding: 10px 20px;
      border-radius: 5px;
      display: none;
    }
  </style>
</head>
<body>
  <h1>Welcome to Our E-commerce Store</h1>
  <div class="product-grid" id="products"></div>
<div id="notification">Item added to cart!</div>

  <script>
    // Sample product data
```

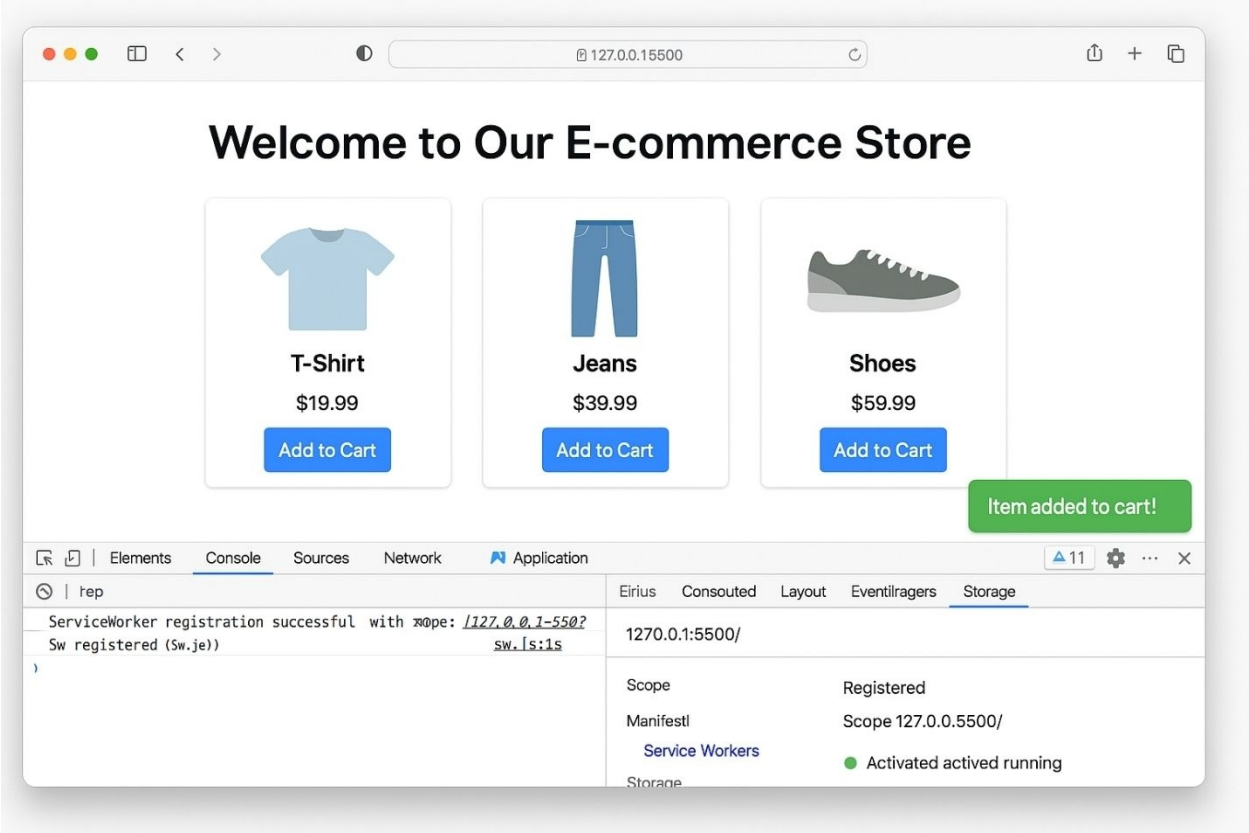
```
    const products = [
      { id: 1, name: "T-Shirt", price: 19.99, image:
"https://via.placeholder.com/200" },
      { id: 2, name: "Jeans", price: 39.99, image: "https://via.placeholder.com/200"
},
      { id: 3, name: "Shoes", price: 59.99, image: "https://via.placeholder.com/200" }
    ];

    // Render products
    function renderProducts() {
      const grid = document.getElementById('products');
      grid.innerHTML = products.map(product =>
        <div class="product-card">
          <img src="${product.image}" alt="${product.name}">
          <h3>${product.name}</h3>
          <p>$${product.price}</p>
          <button onclick="addToCart(${product.id})">Add to Cart</button>
</div>
       ).join('');
    }

    // Add to cart simulation
    function addToCart(productId) {
      const notification = document.getElementById('notification');
notification.style.display = 'block';
      setTimeout(() => notification.style.display = 'none', 2000);
}

    // Register service worker
    if ('serviceWorker' in navigator) {
      window.addEventListener('load', () => {
        navigator.serviceWorker.register('/sw.js')
          .then(reg => {
            console.log('Service Worker registered', reg);
          })
          .catch(err => {
            console.log('Service Worker registration failed:', err);
});
      });
    }

    renderProducts();
</script>
</body>
</html>
```

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 | Your E-commerce Store</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      padding: 50px;
    }
    h1 {
      color: #333;
    }
    p {
      color: #666;
    }
  </style>
</head>
<body>
  <h1>You're Offline</h1>
  <p>It looks like you've lost your internet connection. Some features may not be available.</p>
  <p>We'll automatically show you the latest version when you're back online.</p>
  <button id="reload">Try Again</button>

  <script>

    document.getElementById('reload').addEventListener('click', () => {

window.location.reload();

    });

  </script>

</body>

</html>
```

manifest.js

```
{
    "name": "E-commerce Store",
    "short_name": "EcomStore",
    "start_url": "/",
    "display": "standalone",
    "background_color": "#ffffff",
    "theme_color": "#4285f4",
    "icons": [
      {
        "src": "/images/icon-72x72.png",
"sizes": "72x72",
        "type": "image/png"
      },
      {
        "src": "/images/icon-96x96.png",
"sizes": "96x96",
        "type": "image/png"
      },
      {
        "src": "/images/icon-128x128.png",
"sizes": "128x128",
        "type": "image/png"
      },
      {
        "src": "/images/icon-144x144.png",
"sizes": "144x144",
        "type": "image/png"
      },
      {
        "src": "/images/icon-152x152.png",
"sizes": "152x152",
        "type": "image/png"
      },
      {
        "src": "/images/icon-192x192.png",
"sizes": "192x192",
        "type": "image/png"
      },
      {
        "src": "/images/icon-384x384.png",
"sizes": "384x384",
        "type": "image/png"
      },
      {
        "src": "/images/icon-512x512.png",
"sizes": "512x512",
        "type": "image/png"
      }
    ]
  }
```


127.0.0.15500
Welcome to Our E-commerce Store
T-Shirt
$19.99
Add to Cart
Jeans
$39.99
Add to Cart
Shoes
$59.99
Add to Cart
Item added to cart!
Elements
Console
Sources
Network
Application
11
rep
ServiceWorker registration successful with scope: /127.0.0.1-550?
Sw registered (Sw.je))
sw.[s:1s
Eirius
Consouted
Layout
Eventilragers
Storage
1270.0.1:5500/
Scope
Registered
Manifestl
Scope 127.0.0.5500/
Service Workers
Activated actived running
Storage

EXPERIMENT NO. 9

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

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

Things to note about Service Worker:

● A service worker is a programmable network proxy that lets you control how network requests from your page are handled.
● Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.
● The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.
● Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out Promises, an introduction.

Fetch Event
You can track and manage page network traffic with this event. You can check existing cache, manage “cache first” and “network first” requests and return a response that you want.

Of course, you can use many different methods but you can find in the following example a “cache first” and “network first” approach. In this example, if the request’s and current location’s origin are the same (Static content is requested.), this is called “cacheFirst” but if you request a targeted external URL, this is called “networkFirst”.

● **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
● **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed successfully, the new response will be cached and returned. But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

1. fetch Event – Handle Resource Requests

```
self.addEventListener('fetch', event => {
  if (event.request.method !== 'GET') return;
  if (event.request.url.includes('/api/')) {
    // Network-first strategy for dynamic API data
    event.respondWith(
      fetch(event.request)
        .then(response => {
          const clone = response.clone();
          caches.open('ecommerce-pwa-v1').then(cache => {
            cache.put(event.request, clone);
          });
          return response;
        })
        .catch(() => caches.match(event.request))
    );
  } else {
    // Cache-first strategy for static assets
    event.respondWith(
      caches.match(event.request).then(cachedResponse => {
        return cachedResponse || fetch(event.request).then(response => {
const clone = response.clone();
          caches.open('ecommerce-pwa-v1').then(cache => {
            cache.put(event.request, clone);
          });
          return response;
        }).catch(() => {
          if (event.request.headers.get('accept').includes('text/html')) {
            return caches.match('/offline.html');
          }
        });
      })
    );
  }
});
```

2. sync Event – Background Sync for Deferred Requests

Register sync in your app:

```
navigator.serviceWorker.ready.then(reg => {
  return reg.sync.register('sync-cart');
});
```

Handle it in sw.js:

```
self.addEventListener('sync', event => {
  if (event.tag === 'sync-cart') {
    event.waitUntil(syncCartData());
  }
});
function syncCartData() {
  // Example: Sync stored cart data when back online
return idbKeyval.get('pending-cart').then(cart => {
    if (!cart) return;
    return fetch('/api/sync-cart', {
      method: 'POST',
      body: JSON.stringify(cart),
      headers: {
        'Content-Type': 'application/json'
      }
    }).then(() => {
      // Clear local data after sync
      return idbKeyval.del('pending-cart');
    });
  });
}
```

3. push Event – Handle Push Notifications

Trigger from server:

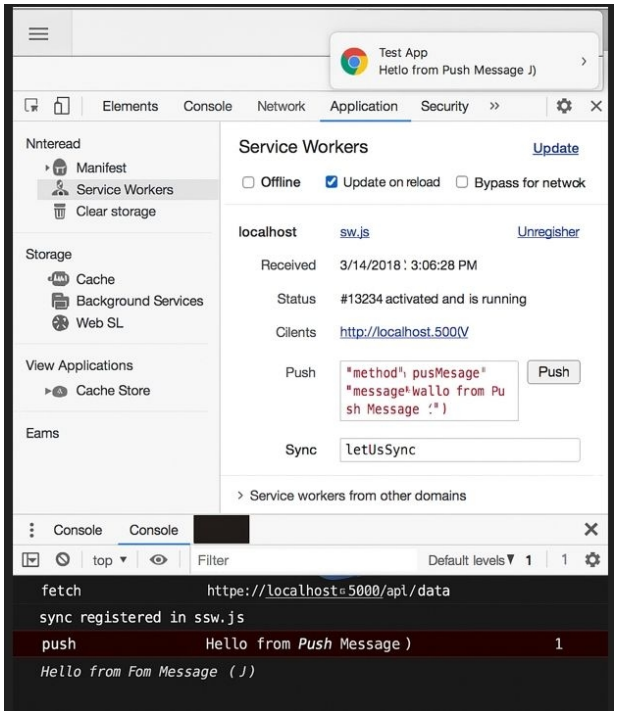
```
{
  "title": "Order Update",
  "body": "Your order #1234 has shipped!",
  "icon": "/images/icon-192x192.png",
  "badge": "/images/badge.png"
}
```

Handle in sw.js:

```
self.addEventListener('push', event => {
  const data = event.data?.json() || {};
  const title = data.title || "E-commerce Store";
  const options = {
    body: data.body || "Check out our latest updates!",
icon: data.icon || '/images/icon-192x192.png',
    badge: data.badge || '/images/badge.png'
  };
  event.waitUntil(
    self.registration.showNotification(title, options)
  );
});
```

Handle click on notification:

```
self.addEventListener('notificationclick', event => {
  event.notification.close();
  event.waitUntil(
    clients.openWindow('/orders') // Redirect to orders page or promotional link
);
});
```



Experiment No. 10

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

Theory:
GitHub Pages
Public web pages are freely hosted and easily published. Public webpages hosted directly from your GitHub repository. Just edit, push, and your changes are live.

GitHub Pages provides the following key features:
1. Blogging with Jekyll
2. Custom URL
3. Automatic Page Generator

Reasons for favoring this over Firebase:
1. Free to use
2. Right out of github
3. Quick to set up

GitHub Pages is used by Lyft, CircleCI, and HubSpot.
GitHub Pages is listed in 775 company stacks and 4401 developer stacks.

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

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

Firebase

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

Firebase cloud and with other clients within milliseconds.

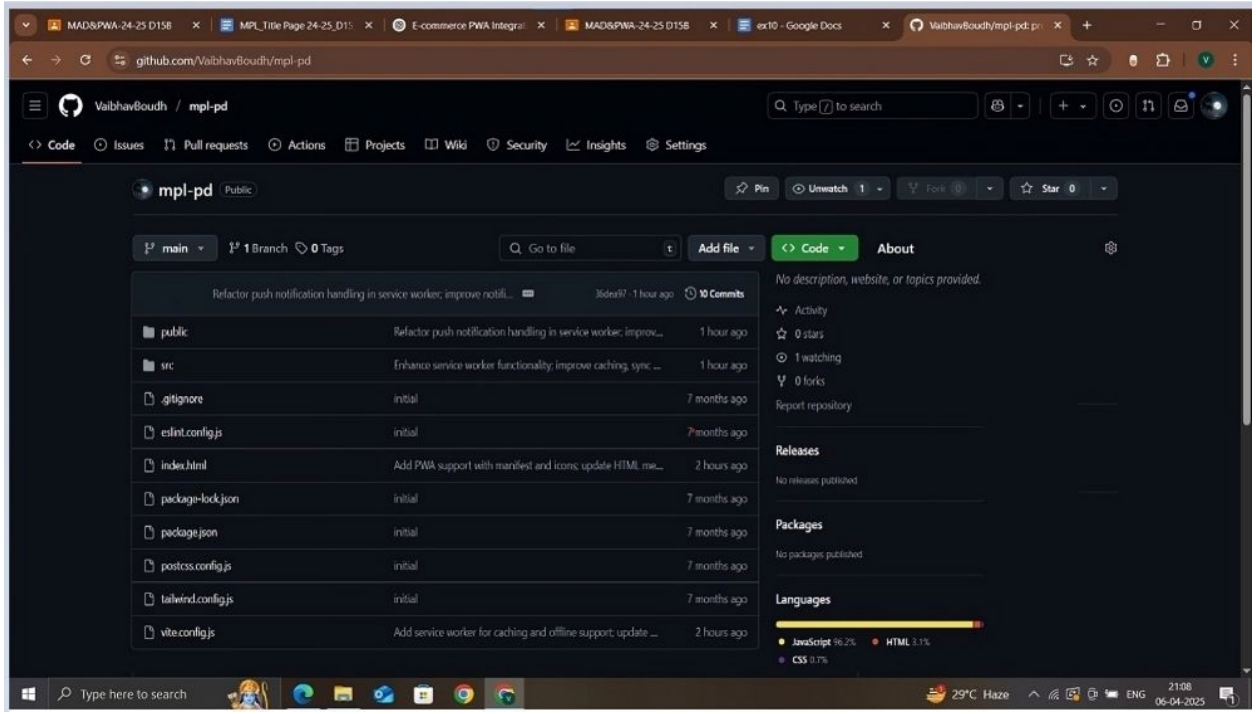
Some of the features offered by Firebase are:
1. Add the Firebase library to your app and get access to a shared data structure. Any changes made to that data are automatically synchronized with the Firebase cloud and with other clients within milliseconds.
2. Firebase apps can be written entirely with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

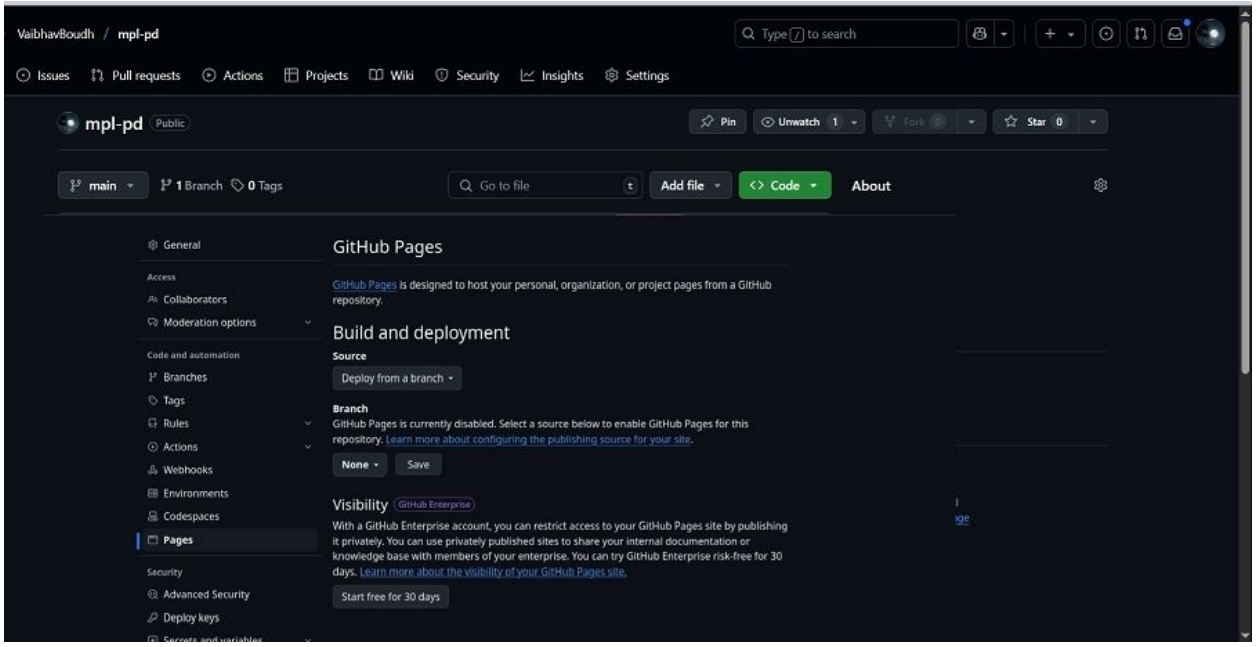
Reasons for favoring over GitHub Pages:
1. Realtime backend made easy
2. Fast and responsive

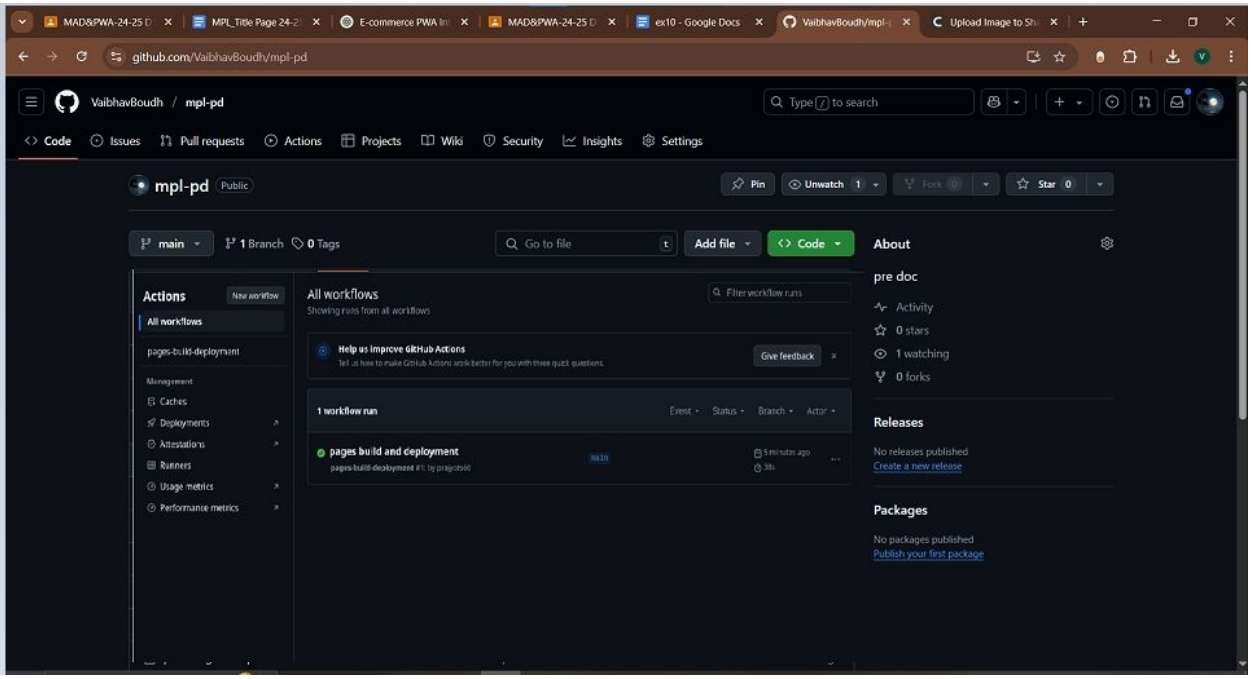
Instacart, 9GAG, and Twitch are some of the popular companies that use Firebase
Firebase has a broader approval, being mentioned in 1215 company stacks & 4651 developers stacks

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

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


github.com/VaibhavBoudh/mpl-pd
VaibhavBoudh / mpl-pd
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
mpl-pd
Public
Pin
Unwatch 1
Star 0
main
1 Branch
0 Tags
Go to file
Add file
Code
About
No description, website, or topics provided.
Activity
0 stars
1 watching
0 forks
Report repository
Releases
No releases published
Packages
No packages published
Languages
JavaScript 96.2%
HTML 3.1%
CSS 0.7%
public
src
.gitignore
eslint.config.js
index.html
package-lock.json
package.json
postcss.config.js
tailwind.config.js
vite.config.js
Type here to search
29°C Haze
ENG
21:08
06-04-2025


VaibhavBoudh / mpl-pd
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
mpl-pd
Public
Pin
Unwatch 1
Star 0
main
1 Branch
0 Tags
Go to file
Add file
Code
About
General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Security
Advanced Security
Deploy keys
GitHub Pages
GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.
Build and deployment
Source
Deploy from a branch
Branch
GitHub Pages is currently disabled. Select a source below to enable GitHub Pages for this repository. Learn more about configuring the publishing source for your site.
None
Save
Visibility
GitHub Enterprise
With a GitHub Enterprise account, you can restrict access to your GitHub Pages site by publishing it privately. You can use privately published sites to share your internal documentation or knowledge base with members of your enterprise. You can try GitHub Enterprise risk-free for 30 days. Learn more about the visibility of your GitHub Pages site.
Start free for 30 days


github.com/VaibhavBoudh/mpl-pd
VaibhavBoudh / mpl-pd
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
mpl-pd
Public
Pin
Unwatch 1
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
t
Add file
Code
About
pre doc
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Actions
New workflow
All workflows
pages-build-deployment
Management
Caches
Deployments
Attestations
Runners
Usage metrics
Performance metrics
All workflows
Showing runs from all workflows
Filter workflow runs
Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.
Give feedback
1 workflow run
Event
Status
Branch
Actor
pages build and deployment
pages-build-deployment #1: by prajotsvd
main
5 minutes ago
34s

Experiment No. 11

Theory :

Google Lighthouse :

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

Key Features and Audit Metrics

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

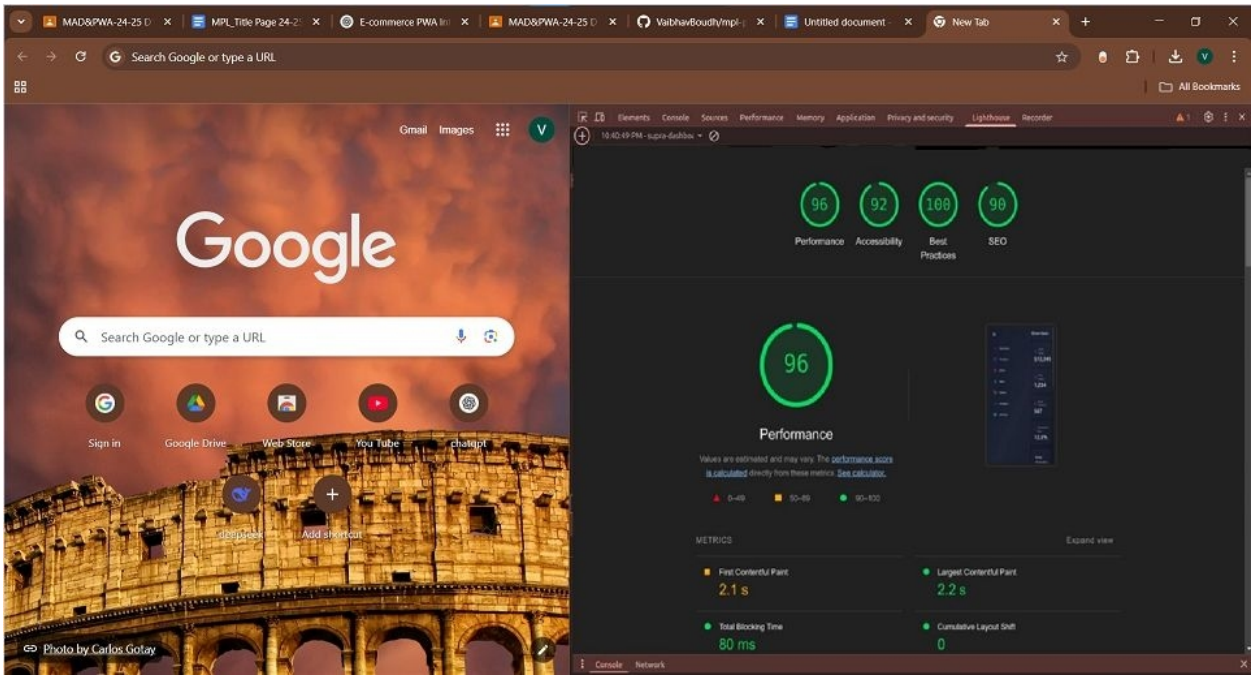
1. Performance: This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of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

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

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

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

Screenshot


New Tab
Search Google or type a URL
All Bookmarks
Gmail
Images
Google
Search Google or type a URL
Sign in
Google Drive
Web Store
You Tube
chatgpt
Add shortcut
Photo by Carlos Gotay
Performance
Accessibility
Best Practices
SEO
96
92
100
90
Performance
First Contentful Paint
2.1 s
Largest Contentful Paint
2.2 s
Total Blocking Time
80 ms
Cumulative Layout Shift
0

Vaibhav Boudh
D15B
Roll no: 06 MAD Assignment-1

Q.1 a) Explain the key features and advantages of using Flutter for mobile app development

Ans Flutter, developed by Google, is a popular open-source UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase. Its key features and advantages include:

1. Single Codebase - Flutter allows developers to write one codebase for both Android and iOS, reducing development time and effort.
2. Hot Reload - Enables real-time UI updates without restarting the application, speeding up debugging and development.
3. Rich Widget Library :- Offers a wide range of customizable widgets that help build visually appealing and consistent UIs
4. Fast Performance :- Uses the Dart programming language and a high-performance rendering engine (Skia) for smooth animations and fast execution
5. Cross-Platform Development - Apart from mobile, Flutter supports web, desktop, and embedded devices.
6. Open Source - Being free and open-source, it has strong community support and extensive documentation
7. Native-like Experience - Ensures high performance and a native look and feel due to direct compilation to native ARM code.

Teacher's Sign.: _____

Q.1. b) Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community.

Ans Traditional mobile development requires separate codebases for Android (Java/Kotlin) and iOS (Swift/Objective-C). Flutter differs by offering:

1. Declarative UI - Unlike traditional imperative UI frameworks Flutter uses a declarative UI approach, making UI development easier and predictable.
2. Faster Development - With Hot Reload, changes in the code are instantly reflected, reducing development time
3. Single Codebase :- Instead of writing separate code for Android and iOS, Flutter allowes developers to maintain a single codebase, reducing complexity and cost.
4. No Need for Native UI Component :- Traditional framework rely on native UI components, while Flutter renders UI using its own Skia rendering engine, ensuring consistency across platforms.
5. Growing Popularity :- The increasing demand for cross platform development and Google's support have contributed to its widespread adoption in the developer community.

Q.2.a) Describle the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces.

Ans In Flutter, everything is a widget - UI components, layouts, animations, and even the app itself.

- Widget Tree represents the hierarchy of widgets in a Flutter app. It defines the structure and layout of the UI.
- Composition over Inheritance - Instead of creating complex widgets from scratch, smaller widgets are composed together to form a UI.
- Stateless Widget Does not maintain any state. Example: Text, Icon, Container.
- Stateful Widget :- Maintains dynamic state and can be updated. E.g. Textfield, checkbox, slider.

Q.2.b) Provide example of commonly used widgets and their roles in creating a widget tree.

Ans 1. Basic Widgets:

- Text() - Displays text.
- Image() - Displays images.
- Icon() - Displays icons.

2. Layout Widgets:

- Row() - Arranges widgets horizontally
- Column() - Arranges widgets vertically
- Container() - used for styling, padding and margins

3. Input Widgets:

- TextField() - For user text input.
- Checkbox() - For selecting / deselecting an option.

Teacher's Sign.: _____

4. Interactive Widgets
- ElevatedButton() - A clickable button.
- GestureDetector() - Detects touch gestures.

Q3.a) Discuss the importance of state management in Flutter applications.

Ans State management is crucial for handling UI changes dynamically.

1. Efficient UI Updates - Ensures only necessary parts of the UI are re-rendered when data changes.

2. Better Performance - Reduces unnecessary rebuilds, making the app responsive and smooth.

3. Data Persistence - Helps maintain user interactions, like form inputs or authentication states.

4. Scalability - As applications grow, proper state management prevents code complexity.

Q.3.b) Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable.

Ans

Approach	Description	When to Use
setState	The simplest method that updates the UI within a StatefulWidget	Best for small apps with limited state changes (e.g., button click updates).

Teacher's Sign.: _____

Approach	Description	When to Use
Provider	A more scalable and efficient approach for managing state across multiple widgets	Suitable for medium-sized apps requiring dependency injection and efficient state sharing
Riverpod	An advanced version of Provider with better performance and less boilerplate code	Ideal for large applications with complex state management needs

Q.4.a) Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. 2

Ans Steps to integrate Firebase with Flutter:

1. Create a Firebase project on Firebase Console.
2. Add an App (Android/iOS) and download the configuration file (google-services.json) for Android, GoogleService-Info.plist for iOS).
3. Add Firebase SDK - Install dependencies in pubspec.yaml.

```
dependencies
  firebase_core: latest_version
  firebase_auth: latest_version
```

4. Initialize Firebase in main.dart:

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

5. Use Firebase Services - Authentication, Firestore, storage, etc

Teacher's Sign.: _____

Benefits of Firebase as a Backend Solution:
- No Server Management - Firebase provides Backend-as-a-Service
- Real-time Data Sync - Cloud Firestore enables real-time updates
- Authentication - Easy integration with Google, Facebook, and Email authentication
- Scalability - Firebase handles large-scale applications seamlessly.

Q.4.b) Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved

Ans Commonly used Firebase Services
1. Firebase Authentication - Manages user authentication.
2. Cloud Firestore - NoSQL database with real-time synchronization.
3. Firebase Realtime Database - Another NoSQL database focused on real time data updates.
4. Firebase Cloud Storage - Stores and serves large files like images and videos
5. Firebase Cloud Messaging (FCM) - Sends push notifications

Data Synchronization in Firebase:
- Firebase Firestore and Realtime Database sync data in real time across all connected devices.
- It uses listeners that update UI instantly when data changes

E.g.

```
Firebase.Firestore.instance
  .collection('messages')
  .snapshots()
  .listen((snapshot){
  for(var doc in snapshot.docs){
    print(doc.data()); }});
```

Teacher's Sign.: _____

Vaibhav Boudh
D15B
Roll no: 06 MPL Assignment - 2

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

ns A progressive web app (PWA) is a web application that combines the best features of both web and mobile apps to deliver a seamless, reliable, and fast user experience. PWAs work offline, load quickly, and provide an app like experience on web browsers.

Significance in Modern Web development:

- Platform Independance - Runs on any device with a web browser
- Improved Performance - Faster load times due to caching and service workers.
- Offline Functionality - Works without an internet connection
- No App Store Dependencies - Users can install PWAs directly from the browser.
- Engaging User Experience - provides push notification and background syncing.

Key characteristics of PWAs vs Traditional Mobile Apps:

Feature	PWAs	Traditional Mobile Apps
Installation	Installed from a browser	Downloaded from App store
Platform	Works across platforms	Requires seprate develop-
Dependency	with one codebase	ment for iOS and Android
Offline Support	Uses Service Workers for offline access	Usually requires native implementation
Updates	Updated automatically via the web	Requires app store updates
Performance	Faster due to caching and lightweight assets	Can be faster, but optimized for specific platforms

Teacher's Sign.: _____

2. Define responsive web design and explain its importance in the context of Progressive Web Apps. Compare and contrast responsive, fluid, and adaptive web design approaches.

Ans Responsive web Design is an approach that ensures web pages adapt to different screen sizes and orientations using flexible grids, media queries, and scalable images.

Importance in PWAS:

- Ensures a consistent user experience across different devices.
- Eliminates the need for multiple codebases for different devices.
- Enhances usability by making content readable on various screen.

Comparison of Responsive, Fluid and Adaptive web design.

Feature	Responsive	Fluid	Adaptive
Definition	Uses CSS media queries to adjust layout dynamically	Uses percentage based units for elements to scale naturally	Uses predefined layouts for different screen sizes
Flexibility	Highly Flexible	Completely flexible	Fixed at specific breakpoints
Performance	Efficient but requires more CSS adjustments	Smooth scaling	May cause layout shifts
Best use case	Websites and PWAS for all screen	Apps requiring seamless scaling	websites with predefined layouts

3. Describe the lifecycle of Service Workers, including registration, installation, and activation phases.

Ans A Service Worker is a background script that runs independently from the main browser thread, enabling features like offline caching and push notification.

Teacher's Sign.: _____

Lifecycle Phases:

1. Registration:

· The service worker is registered in JavaScript using navigator.serviceWorker.register(). e.g.

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

2. Installation:

· Occurs when the service worker is first downloaded.

· Typically used for caching assets. e.g.

```
self.addEventListener('install', event =>{
  event.waitUntil(
    caches.open('v1').then(cache =>{
return cache.addAll(['/index.html', '/style.css']);
  })
  );
});
```

3. Activation:

· Runs after installation and ensures old caches are cleared if necessary. e.g.

```
self.addEventListener('activate', event =>{
  event.waitUntil(
    caches.keys().then(keys =>{
      return Promise.all(
keys.filter(key => key !== 'v1').map(key => cache.delete(key))
    );
  })
  );
});
```

Teacher's Sign.: _____

4. Fetching and Updates:
- The service worker intercepts network requests and serves cached content e.g.

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

4. Explain the use of Indexed DB in the Service Worker for data storage.

Ans Indexed DB is a low-level NoSQL database in the browser that allows web apps to store and retrieve large amounts of structured data efficiently.

Used of Index DB in Service Workers:

1. Offline storage - Saves user data when offline and syncs it when back online
2. Persistent Data - Unlike local Storage, Indexed DB is asynchronous and handles large amounts of data.
3. Background Sync - Service Workers can use Indexed DB to store data and sync it later.

E.g.

```
Const dbRequest = indexedDB.open('myDatabase', 1);
dbRequest.onupgradeneeded = event => { const db = event.target;
db.createObjectStore('messages', { keyPath: 'id' }); };
dbRequest.onsuccess = event => { const db = event.target.result;
const transaction = db.transaction('messages', 'readwrite');
const store = transaction.objectStore('messages');
store.put({ id: 1, Text: 'Hello from IndexedDB' }); };
```

Teacher's Sign.: _____