# Vivekanand Education Society's Institute of Technology

**An Autonomous Institute Affiliated to University of Mumbai**
**Hashu Advani Memorial Complex, Collector Colony, Chembur East, Mumbai - 400074.**


## Department of Information Technology

# CERTIFICATE

This is to certify that **Mahvish Liyaqatullah Siddiqui** of **D15A** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2024-2025.**

Lab Assistant

Subject Teacher

**Mrs. Kajal Joseph**

Principal

Head of Department

**Dr. Mrs. Shalu Chopra**

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

**Course Code :** ITL604

**Project Title:** **Roll No. 56**

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

**Faculty Incharge** : Mrs. Kajal Joseph.

**Lab Teachers** **:** Mrs. Kajal Joseph.

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

**Programme Outcomes**: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

**Program specific Outcomes**

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

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

**Project Title:** **Roll No. 56**

**Lab Objectives:**

| Sr. No. | Lab Objectives |
|---|---|
| **The Lab experiments aims:** | |
| **1** | Learn the basics of the Flutter framework. |
| **2** | Develop the App UI by incorporating widgets, layouts, gestures and animation |
| **3** | Create a production ready Flutter App by including files and firebase backend service. |
| **4** | Learn the Essential technologies, and Concepts of PWAs to get started as quickly and efficiently as possible |
| **5** | Develop responsive web applications by combining AJAX development techniques with the jQuery JavaScript library. |
| **6** | Understand how service workers operate and also learn to Test and Deploy PWA. |

**Lab Outcomes:**

| Sr. No. | Lab Outcomes | Cognitive levels of attainment as per Bloom's Taxonomy |
|---|---|---|
| **On Completion of the course the learner/student should be able to:** | | |
| **1** | Understand cross platform mobile application development using Flutter framework | L1, L2 |
| **2** | Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation | L3 |
| **3** | Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS | L3, L4 |
| **4** | Understand various PWA frameworks and their requirements | L1, L2 |
| **5** | Design and Develop a responsive User Interface by applying PWA Design techniques | L3 |
| **6** | Develop and Analyse PWA Features and deploy it over app hosting solutions | L3, L4 |

**Project Title:** **Roll No. 56**

# Index

| Sr. No | Experiment Title | LO | Grade |
|---|---|---|---|
| **1.** | To install and configure the Flutter Environment | LO1 | |
| **2.** | To design Flutter UI by including common widgets. | LO2 | |
| **3.** | To include icons, images, fonts in Flutter app | LO2 | |
| **4.** | To create an interactive Form using form widget | LO2 | |
| **5.** | To apply navigation, routing and gestures in Flutter App | LO2 | |
| **6.** | To Connect Flutter UI with fireBase database | LO3 | |
| **7.** | To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”. | LO4 | |
| **8.** | To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA | LO5 | |
| **9.** | To implement Service worker events like fetch, sync and push for E-commerce PWA | LO5 | |
| **10.** | To study and implement deployment of Ecommerce PWA to GitHub Pages. | LO5 | |
| **11.** | To use google Lighthouse PWA Analysis Tool to test the PWA functioning. | LO6 | |
| **12.** | Assignment-1 | LO1,LO2 ,LO3 | |
| **13.** | Assignment-2 | LO4,LO5 ,LO6 | |

# MAD & PWA Lab
# Journal

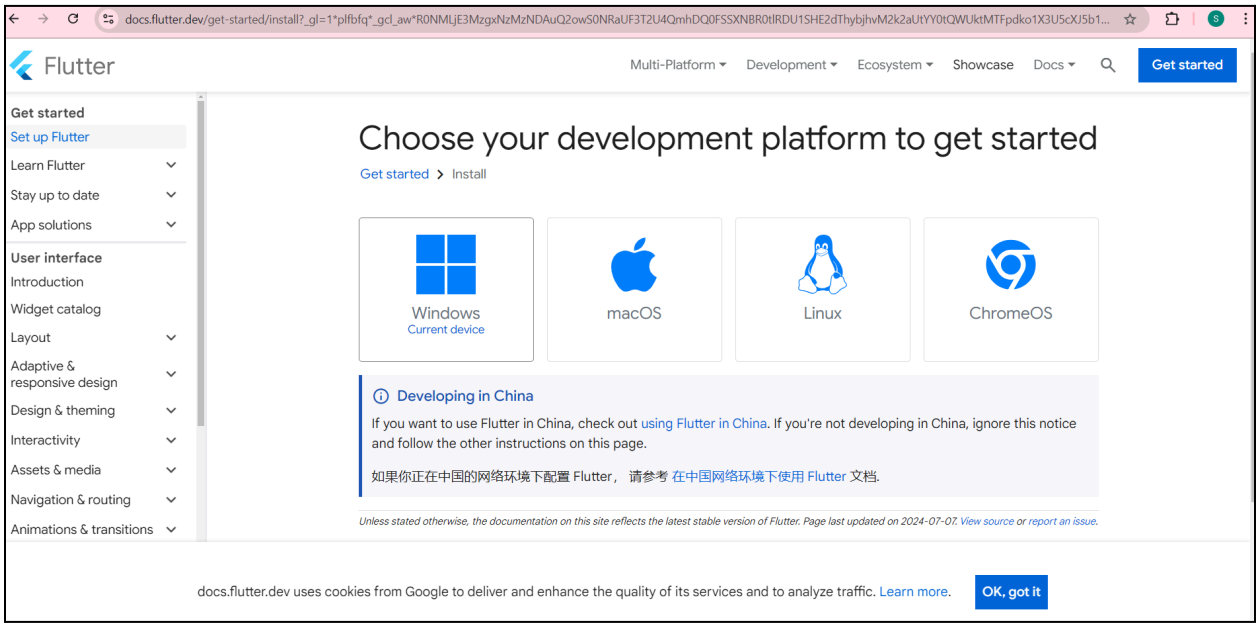
| | |
|---|---|
| Experiment No. | 01 |
| Experiment Title. | To install and configure the Flutter Environment |
| Roll No. | 56 |
| Name | Mahvish Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO1: Understand cross platform mobile application development using Flutter framework |
| Grade: | |

**Project Title:** **Roll No. 56**

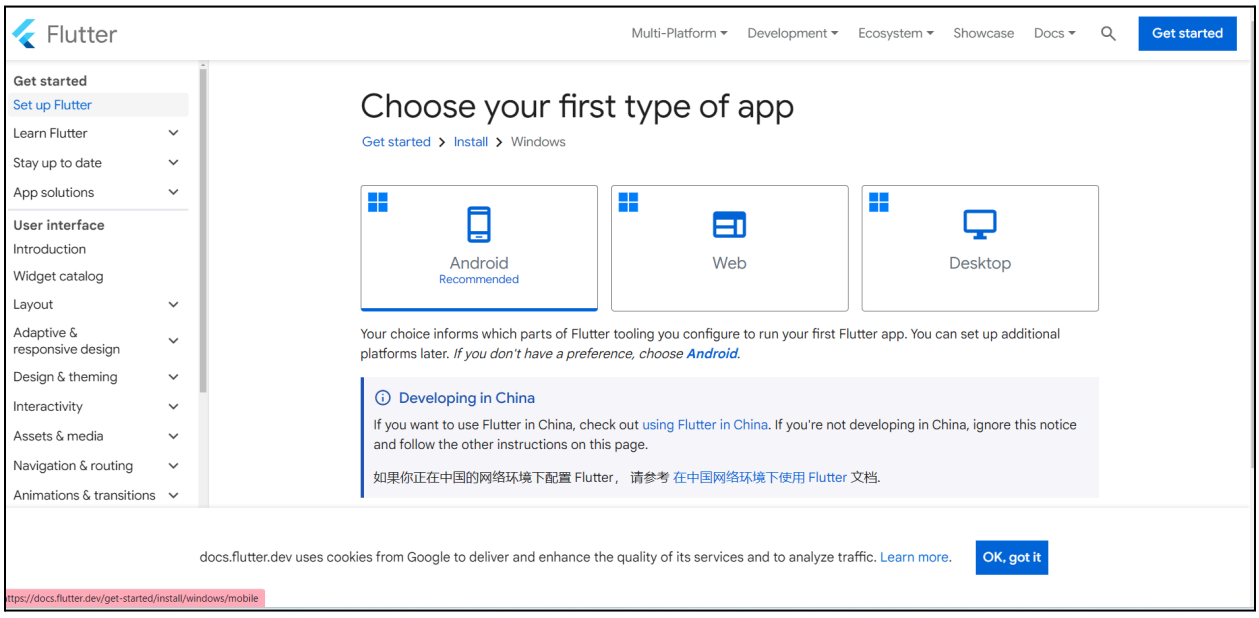
# Experiment No. 1

**AIM** : Installation and Configuration of Flutter Environment.

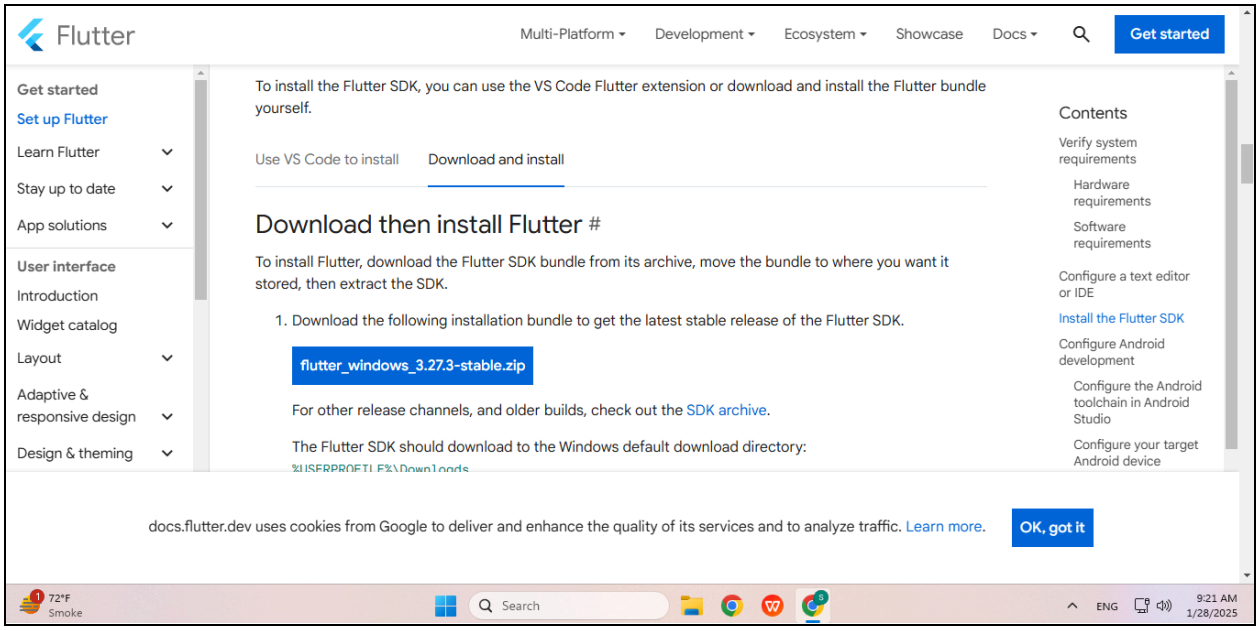
Step 1: Download the installation bundle of the Flutter Software Development Kit for windows. To download Flutter SDK, Go to its official website https://docs.flutter.dev/get-started/install , you will get the following screen.

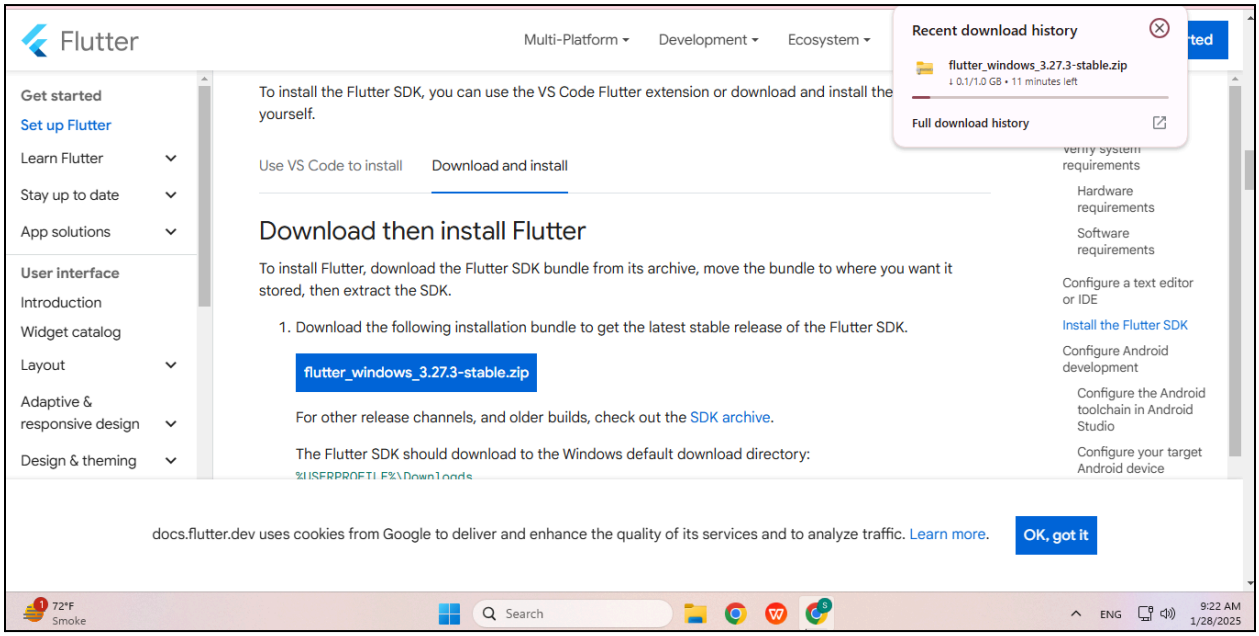



Step 2: Next, to download the latest Flutter SDK, click on the Windows icon and then select Android. Here, you will find system requirements and the download link for SDK.



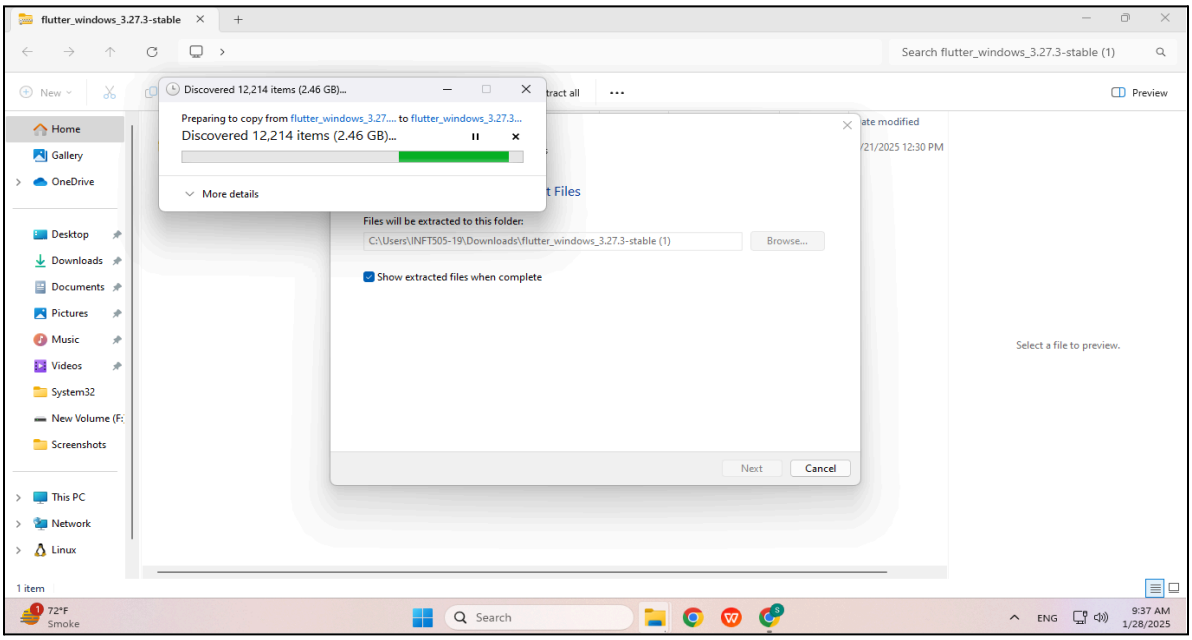
**Project Title:** **Roll No. 56**



**Project Title:** **Roll No. 56**

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. (Here I have created Flutter folder in C drive and inside that created src folder and extracted in it.)

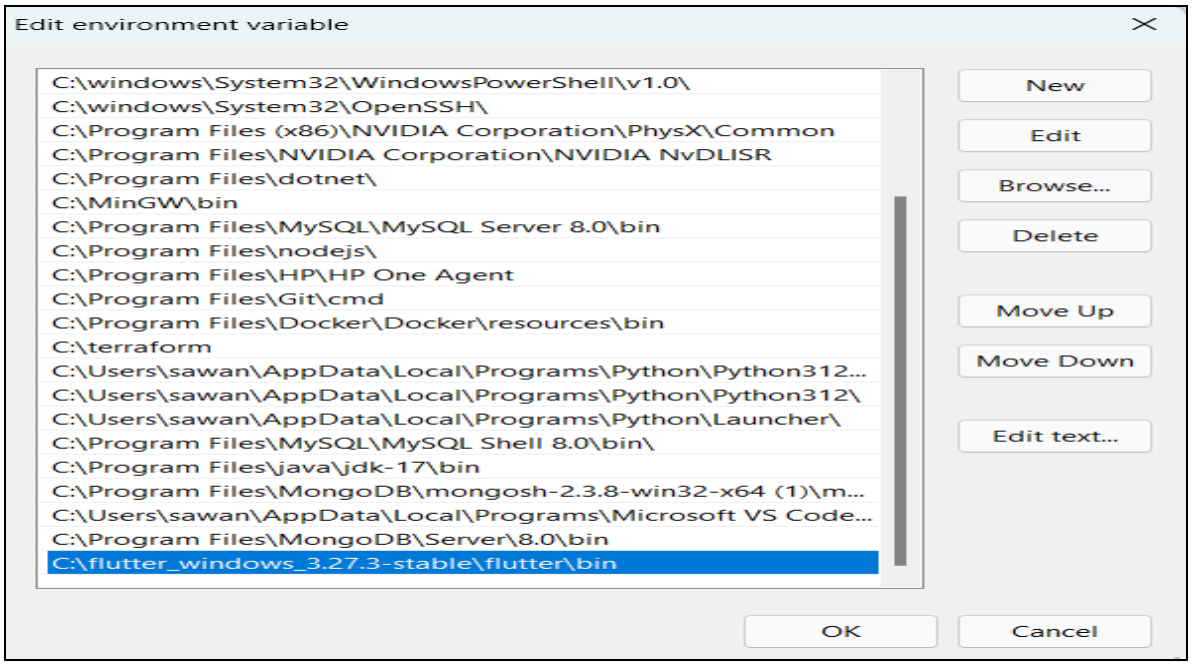



Step 4: Now run flutter doctor command in that folder bin directory we will get to know the status and what is remaining to install

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

C:\flutter_windows_3.27.3-stable\flutter\bin>flutter doctor

    Welcome to Flutter! - https://flutter.dev

  The Flutter tool uses Google Analytics to anonymously report feature usage
  statistics and basic crash reports. This data is used to help improve
  Flutter tools over time.

  Flutter tool analytics are not sent on the very first run. To disable
  reporting, type 'flutter config --no-analytics'. To display the current
  setting, type 'flutter config'. If you opt out of analytics, an opt-out
  event will be sent, and then no further information will be sent by the
  Flutter tool.

  By downloading the Flutter SDK, you agree to the Google Terms of Service.
  The Google Privacy Policy describes how data is handled in this service.

  Moreover, Flutter includes the Dart SDK, which may send usage metrics and
  crash reports to Google.

  Read about data we send with crash reports:
  https://flutter.dev/to/crash-reporting

  See Google's privacy policy:
  https://policies.google.com/privacy

  To disable animations in this tool, use
  'flutter config --no-cli-animations'.
```

```
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.27.3, on Microsoft Windows [Version 10.0.22631.4751], locale en-IN)
[✓] 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.
The Flutter CLI developer tool uses Google Analytics to report usage and diagnostic
data along with package dependencies, and crash reporting to send basic crash
reports. This data is used to help improve the Dart platform, Flutter framework,
and related tools.

Telemetry is not sent on the very first run. To disable reporting of telemetry,
run this terminal command:

    flutter --disable-analytics

If you opt out of telemetry, an opt-out event will be sent, and then no further
information will be sent. This data is collected in accordance with the Google
Privacy Policy (https://policies.google.com/privacy).
```

Step 5: 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 5.1: Search for environment variables in search bar -> advanced tab -> environment variables. You will get the following screen.



Step 6: Now, run the $ flutter command in command prompt.

```
C:\Users\siddi>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
                            diagnostic information. (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.
```

Step 7:Now, run the $ flutter doctor command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation.

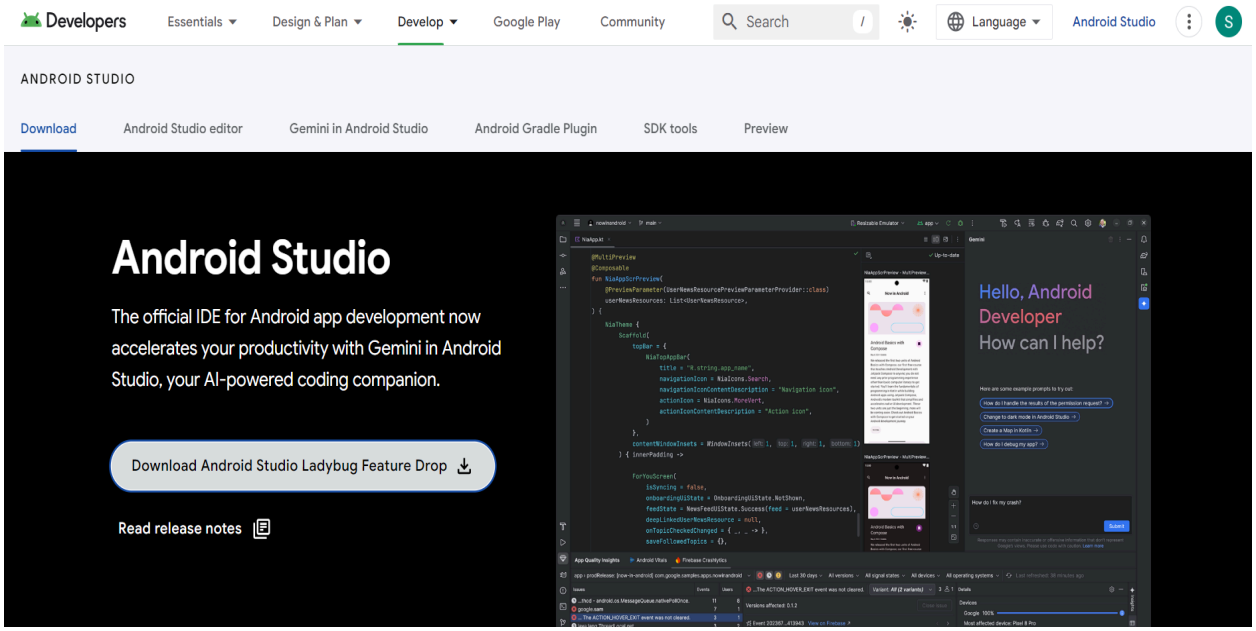
Step 8: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

```
C:\Users\siddi>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.24.0, on Microsoft Windows [Version 10.0.22631.4751], locale en-IN)
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[✓] Android toolchain - develop for Android devices (Android SDK version 35.0.0)
[✓] Chrome - develop for the web
[!] Visual Studio - develop Windows apps (Visual Studio Build Tools 2019 16.11.38)
    X The current Visual Studio installation is incomplete.
      Please use Visual Studio Installer to complete the installation or reinstall Visual Studio.
[✓] Android Studio (version 2024.1)
[✓] VS Code (version 1.96.4)
[✓] Connected device (3 available)
[✓] Network resources

! Doctor found issues in 1 category.

C:\Users\siddi>
```

Step 9: 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.

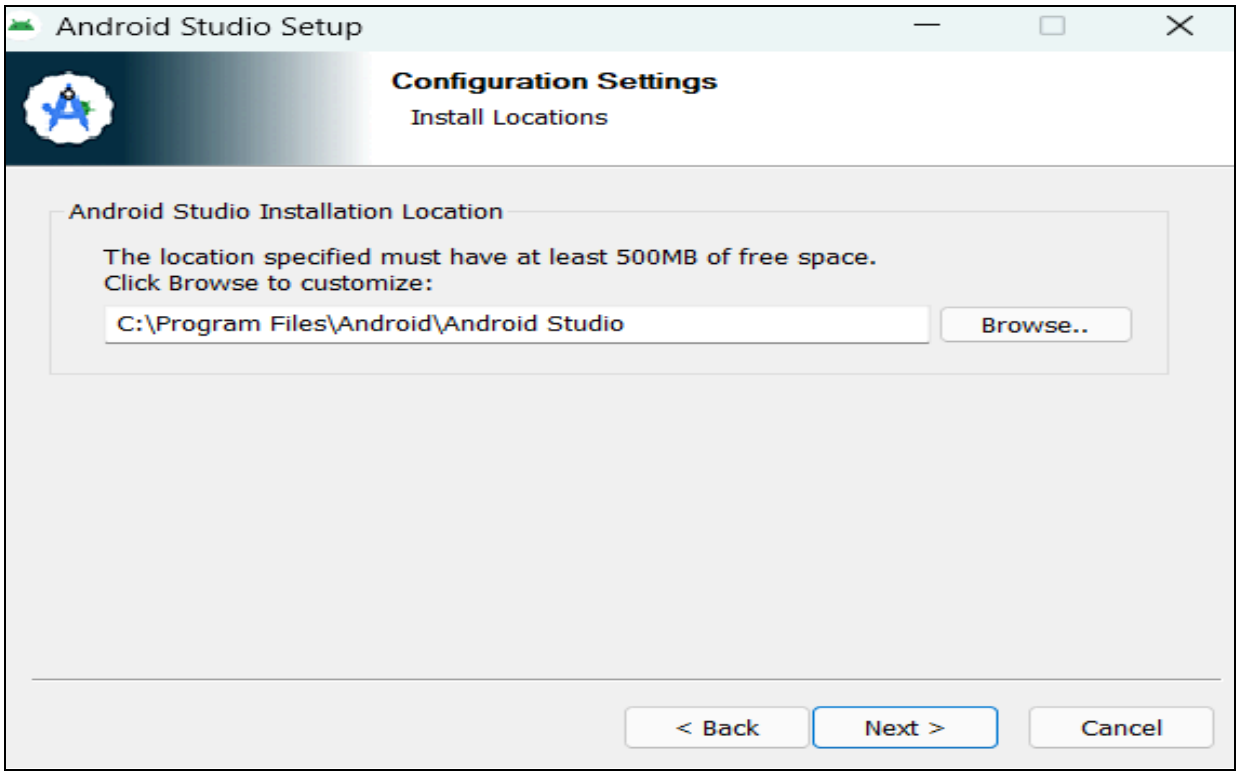
**Project Title:** **Roll No. 56**

Step 9.1: Download the latest Android Studio executable or zip file from the official site by accepting terms and conditions.

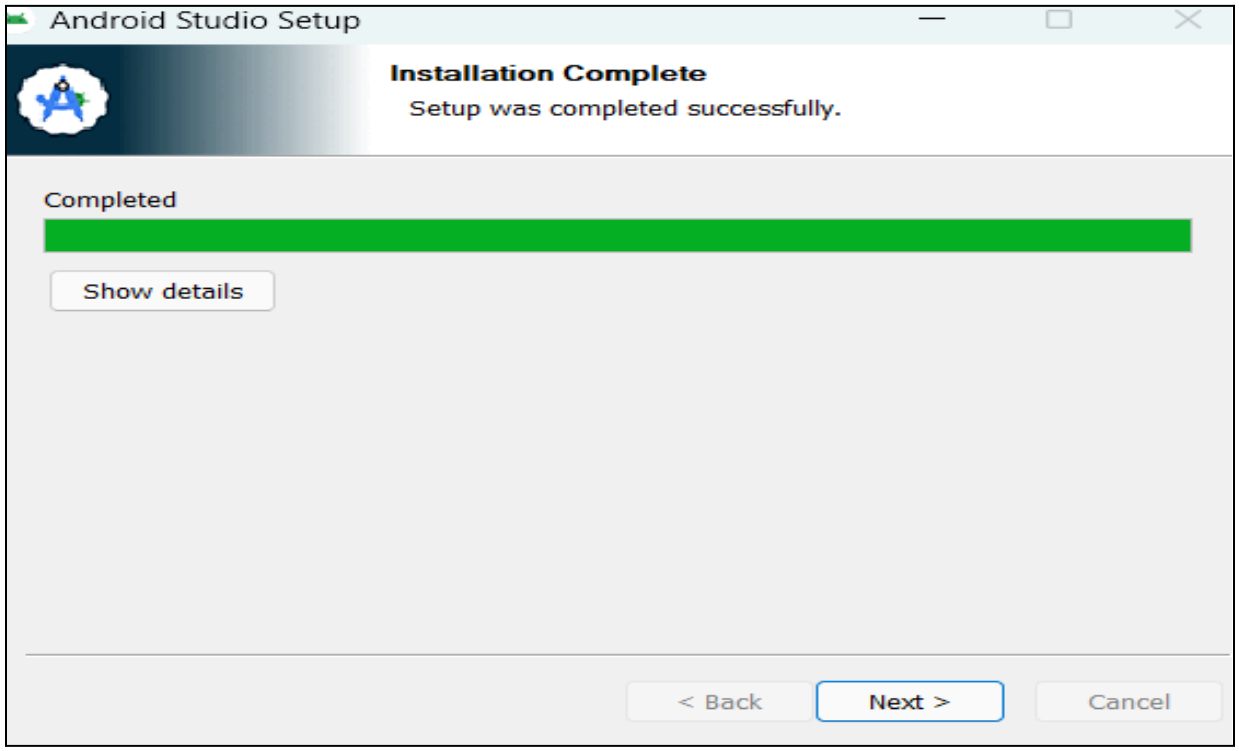



Step 9.2: When the download is complete, open the .exe file and run it. You will get the following dialog box.

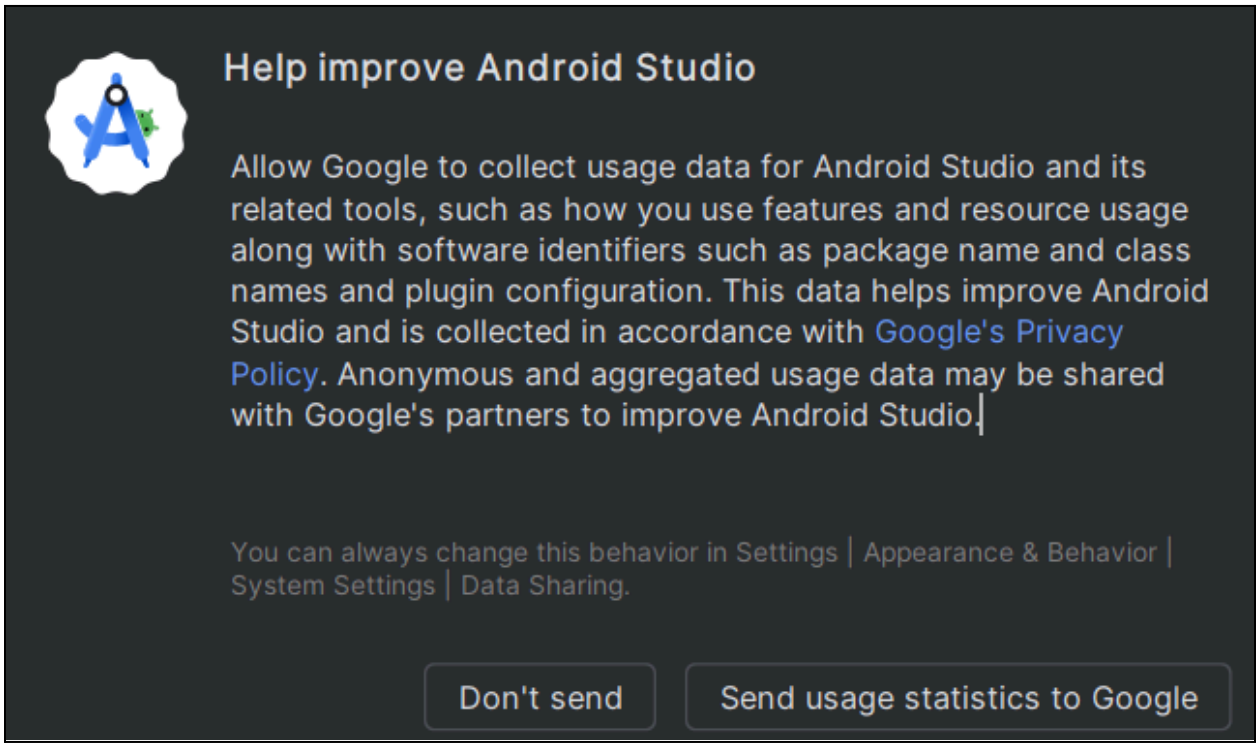

Step 9.3: Follow the steps of the installation wizard and also select installation location.



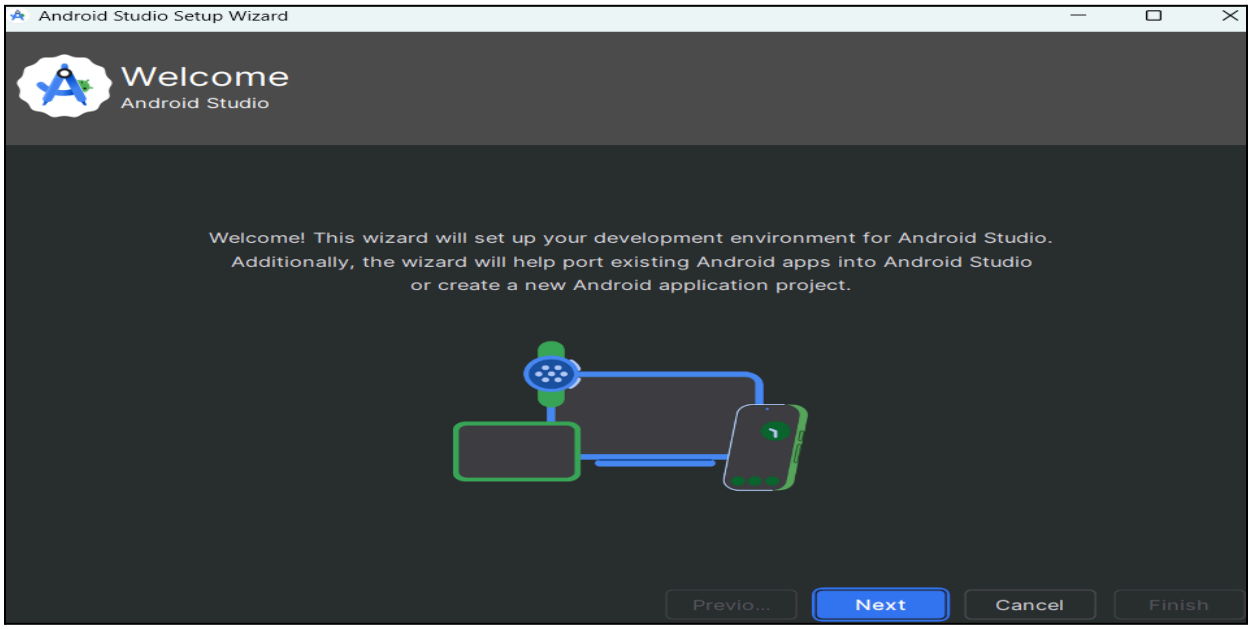

Step 9.4:Once the installation wizard completes, you will get the following screen.



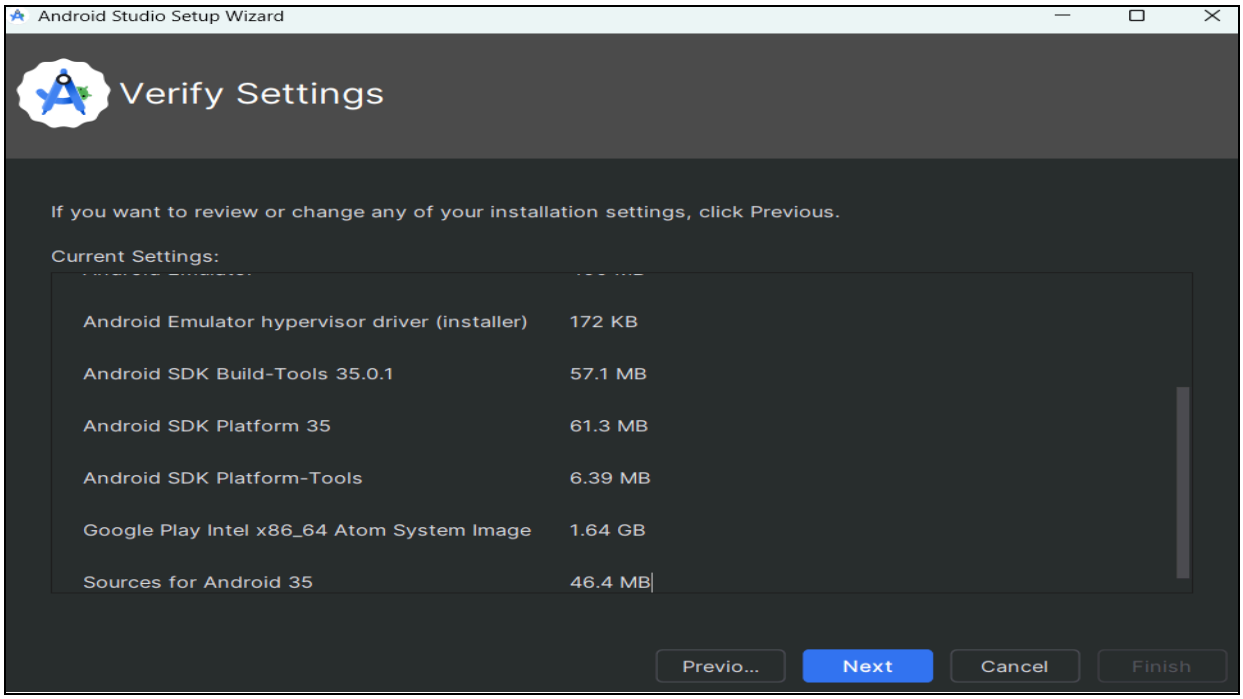
Step 9.5: In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to choose the 'Don't import Settings option' and click OK. It will start the Android Studio. Also click on Don't send so that your data will not get shared with android studio.



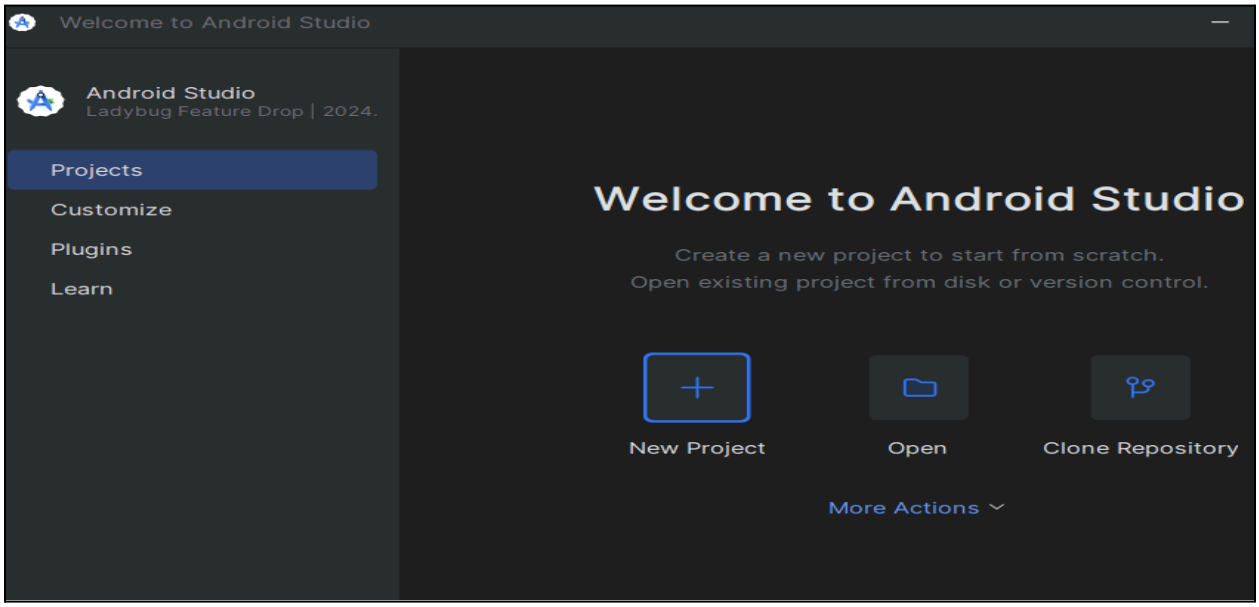

Step 9.6: Now you will see android studio setup wizard. Click next



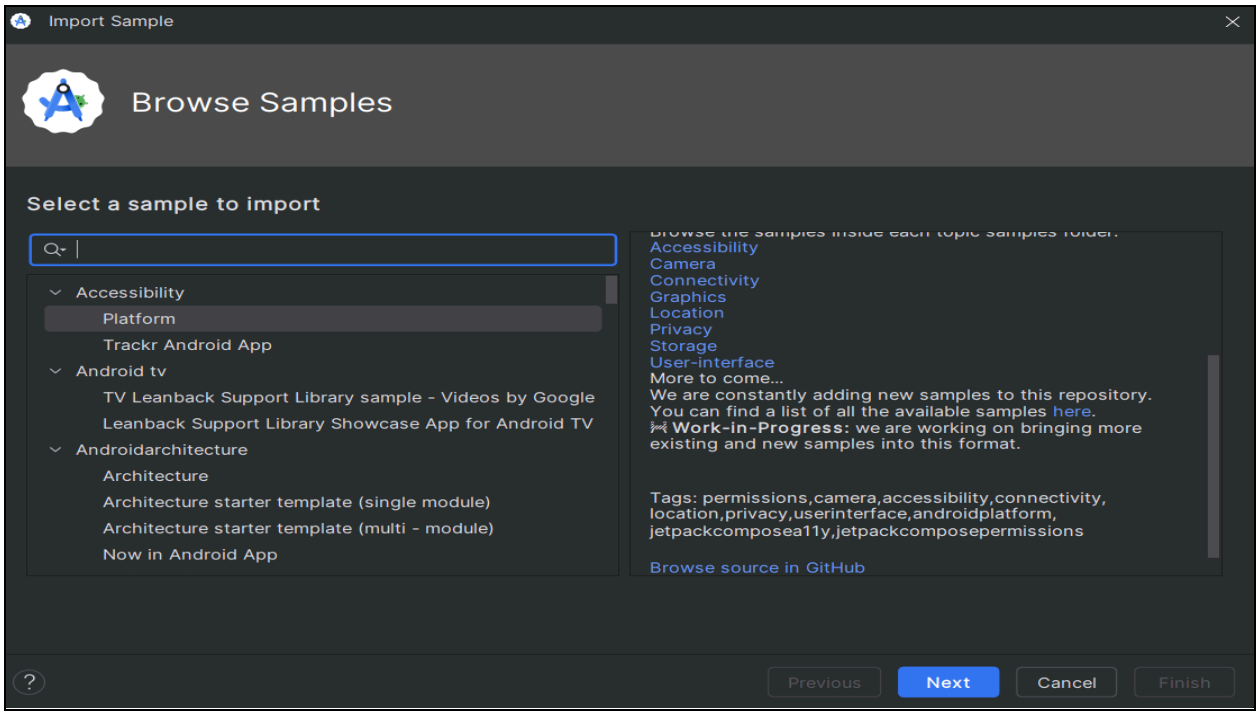

Step 9.7: Select Standard then Next -> Next -> Accept ->Finish.

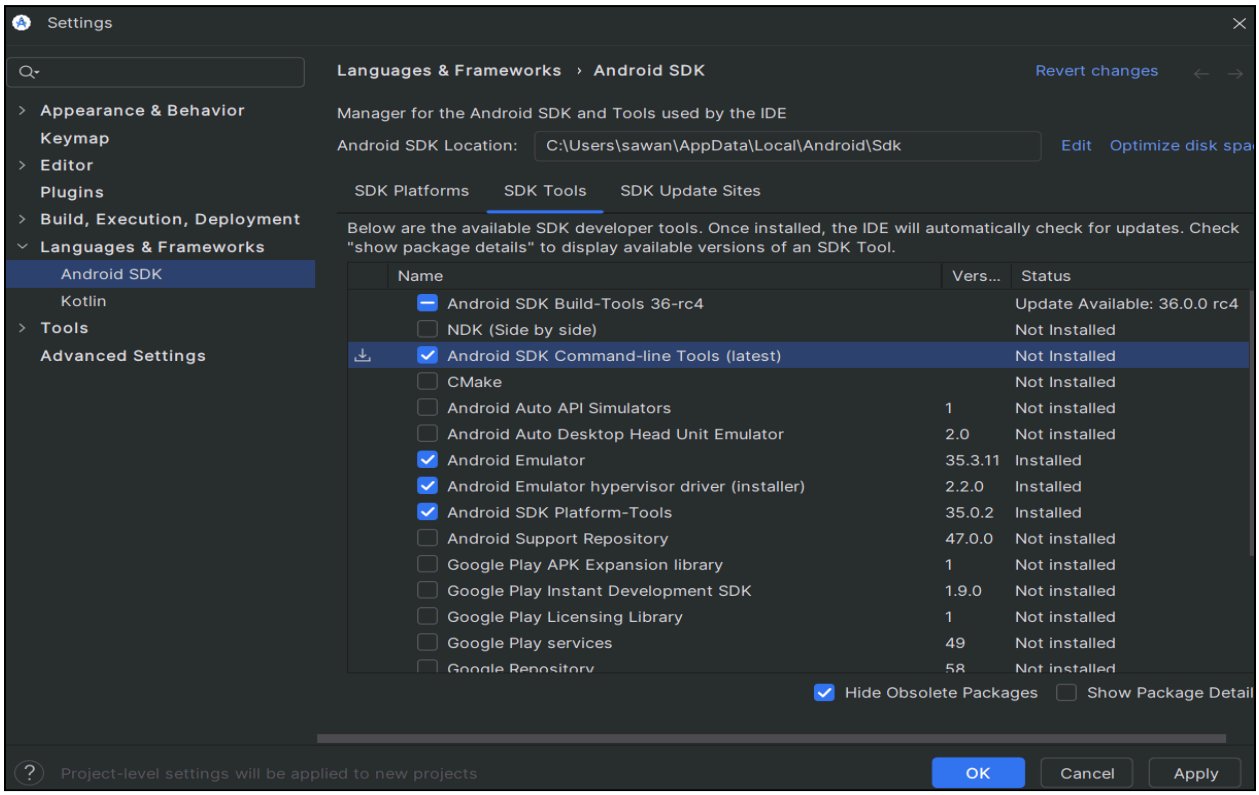


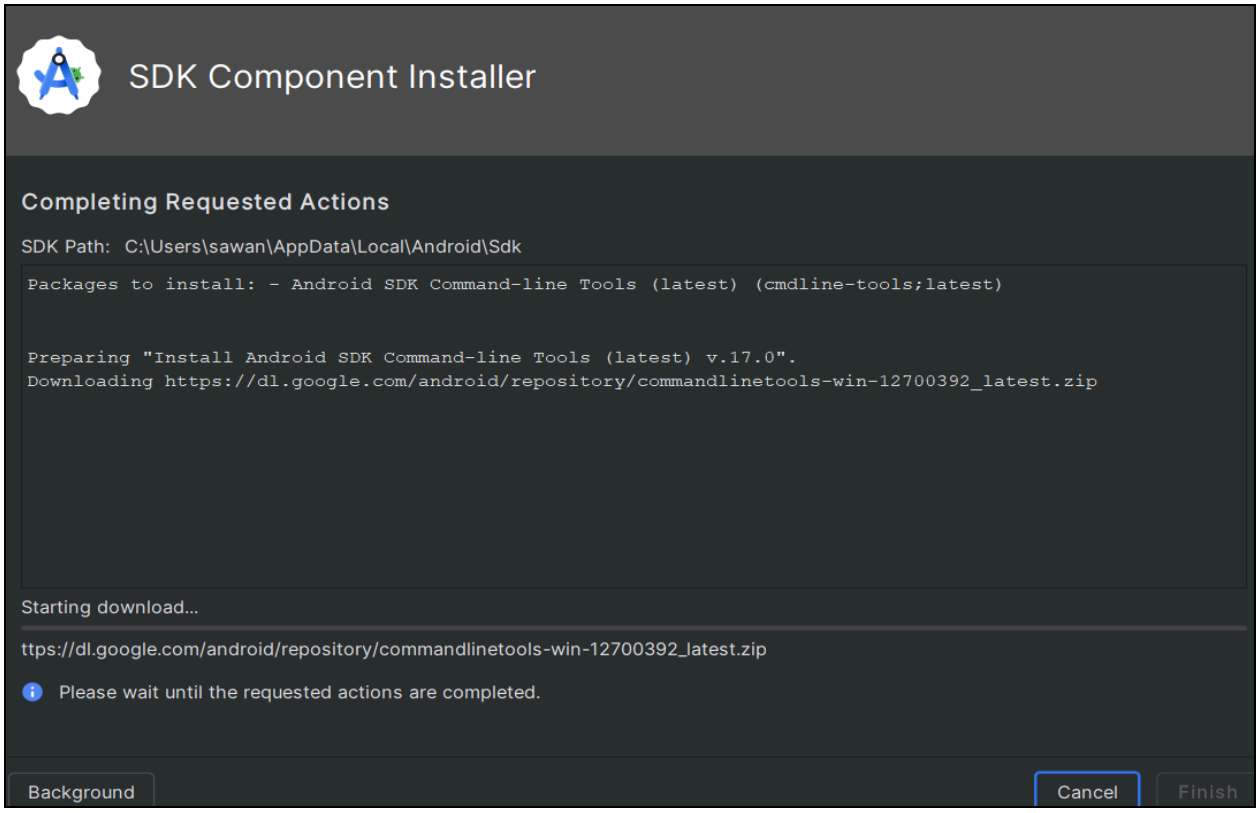

Step 9.8: Now you will see following downloading components wizard. After finishing download click on Finish.
Step 9.9: run the $ flutter doctor command and Run flutter doctor --android-licenses command.
Step 9.10: Now open android studio you will see the following window. Click on more actions-> Import an android code Sample -> select Android SDK command-line tools (latest) this will download command-line tools.



**Project Title:** **Roll No. 56**



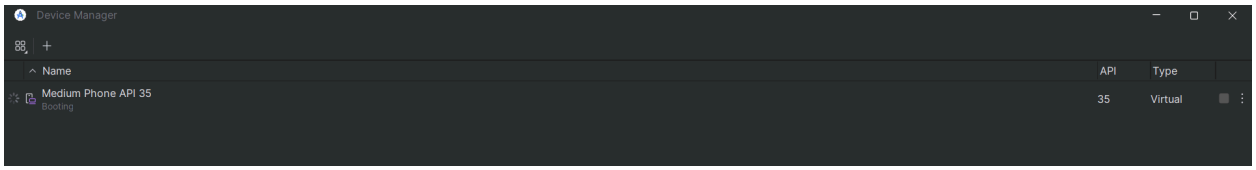





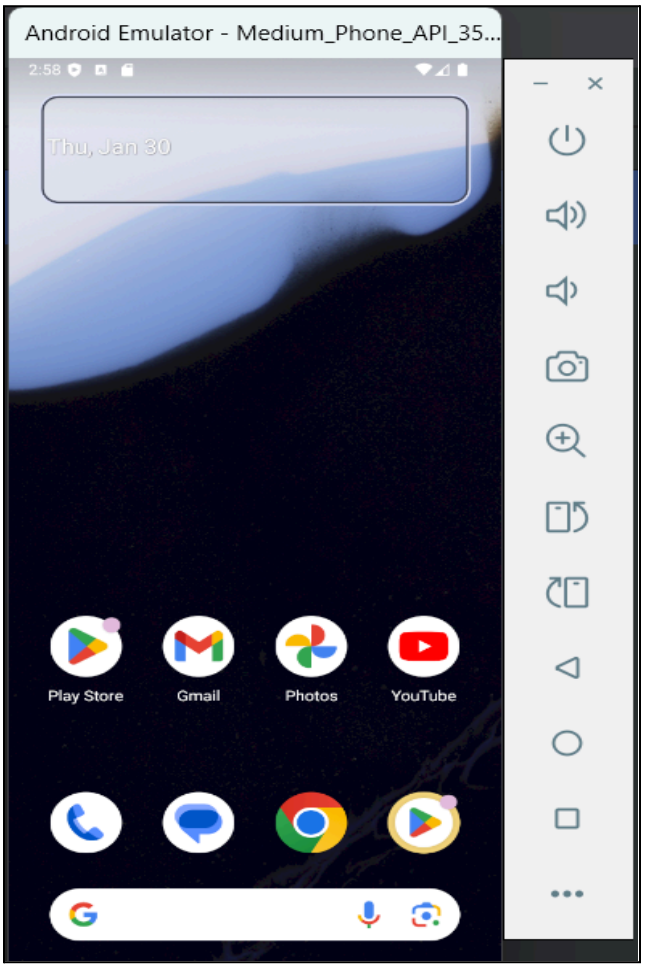

Step 10: Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.
Step 10.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.
Step 10.2: Choose your device definition and click on Next.
Step 10.3: Select the system image for the latest Android version and click on Next.

```
C:\Users\sawan> flutter doctor --android-licenses
Warning: Errors during XML parse:
Warning: Additionally, the fallback loader failed to parse the XML.ry...
Warning: Errors during XML parse:
Warning: Additionally, the fallback loader failed to parse the XML.ry...
[======================================] 100% Computing updates...
6 of 7 SDK package licenses not accepted.
Review licenses that have not been accepted (y/N)? y

1/6: License android-googletv-license:
---------------------------------------
Terms and Conditions

This is the Google TV Add-on for the Android Software Development Kit License Agreement.

1. Introduction

1.1 The Google TV Add-on for the Android Software Development Kit (referred to in this License Agreement as the "Google
TV Add-on" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to y
ou subject to the terms of this License Agreement. This License Agreement forms a legally binding contract between you a
nd Google in relation to your use of the Google TV Add-on.

1.2 "Google" means Google Inc., a Delaware corporation with principal place of business at 1600 Amphitheatre Parkway, Mo
untain View, CA 94043, United States.

2. Accepting this License Agreement

2.1 In order to use the Google TV Add-on, you must first agree to this License Agreement. You may not use the Google TV
Add-on if you do not accept this License Agreement.
```

Step 10.4: Now, verify the all AVD configuration.Select Hardware in graphics. If it is correct, click on Finish

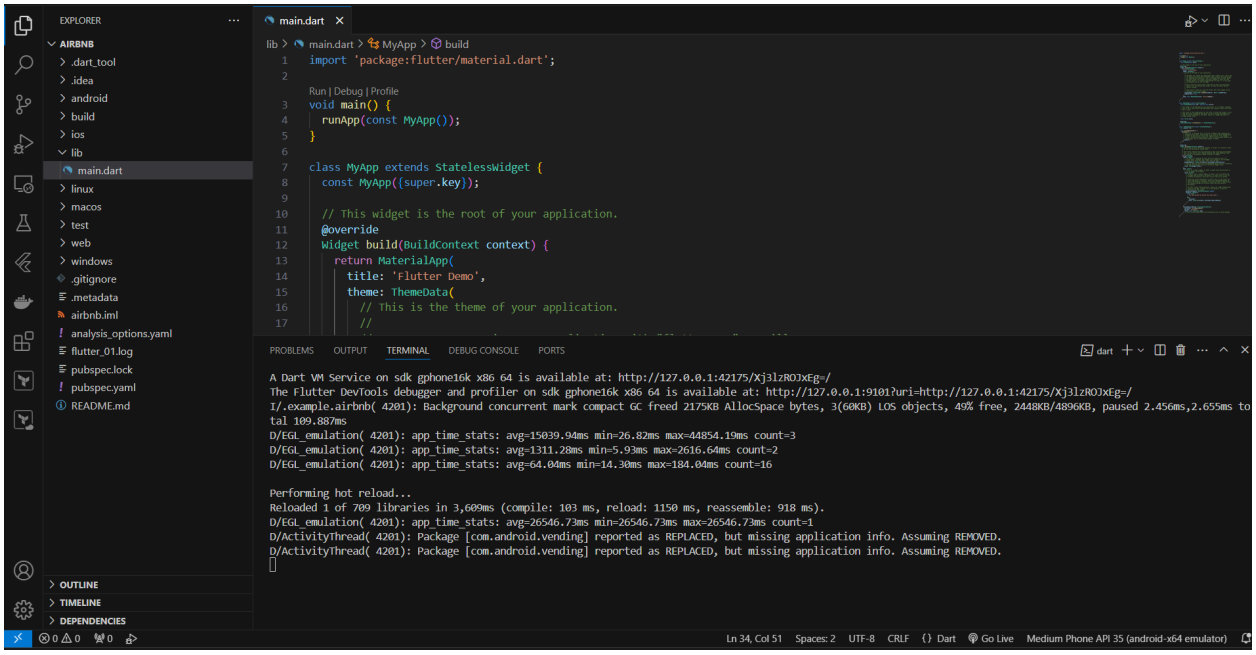


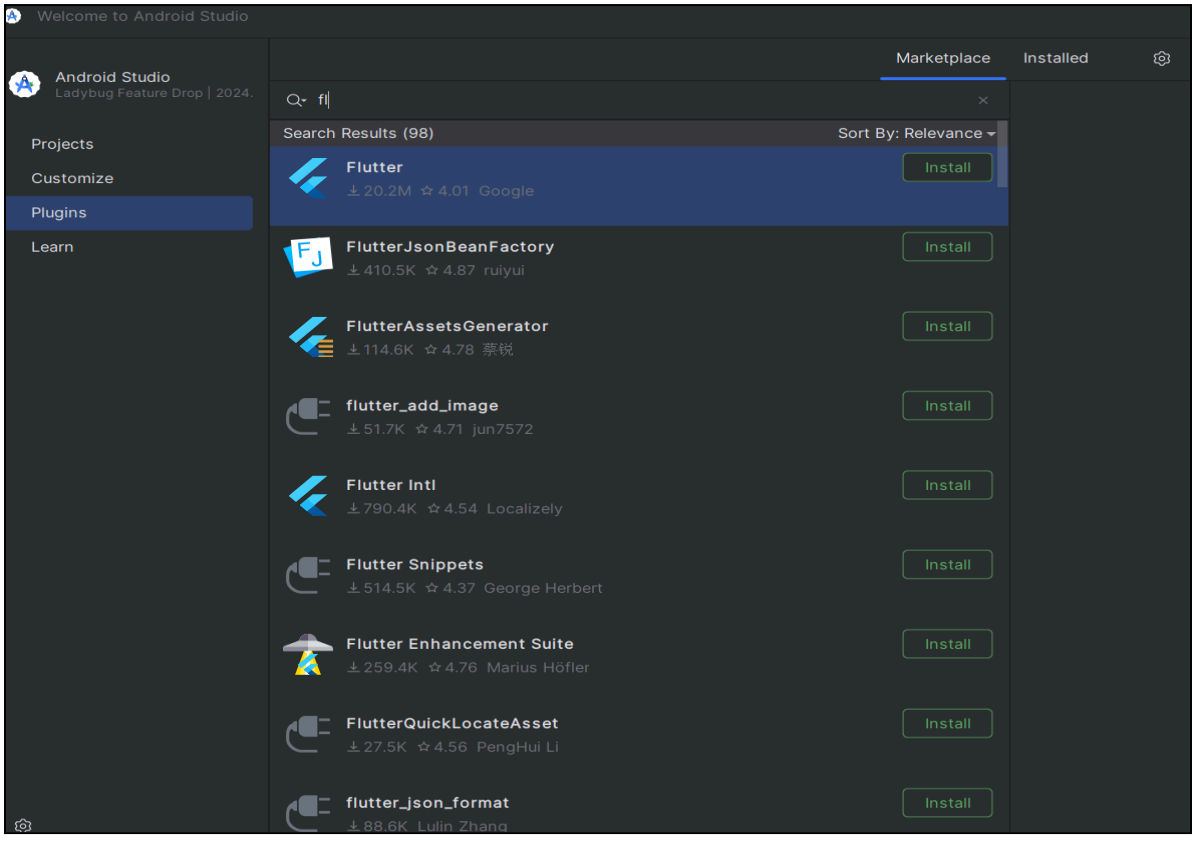

Step 10.5: Last, click on the icon pointed into the rectangle. The Android emulator displayed as below screen.




Step 11: Next, you need to install flutter plugin . Go to plugins ->Install Flutter plugin( it will automatically install dart ) and then restart 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

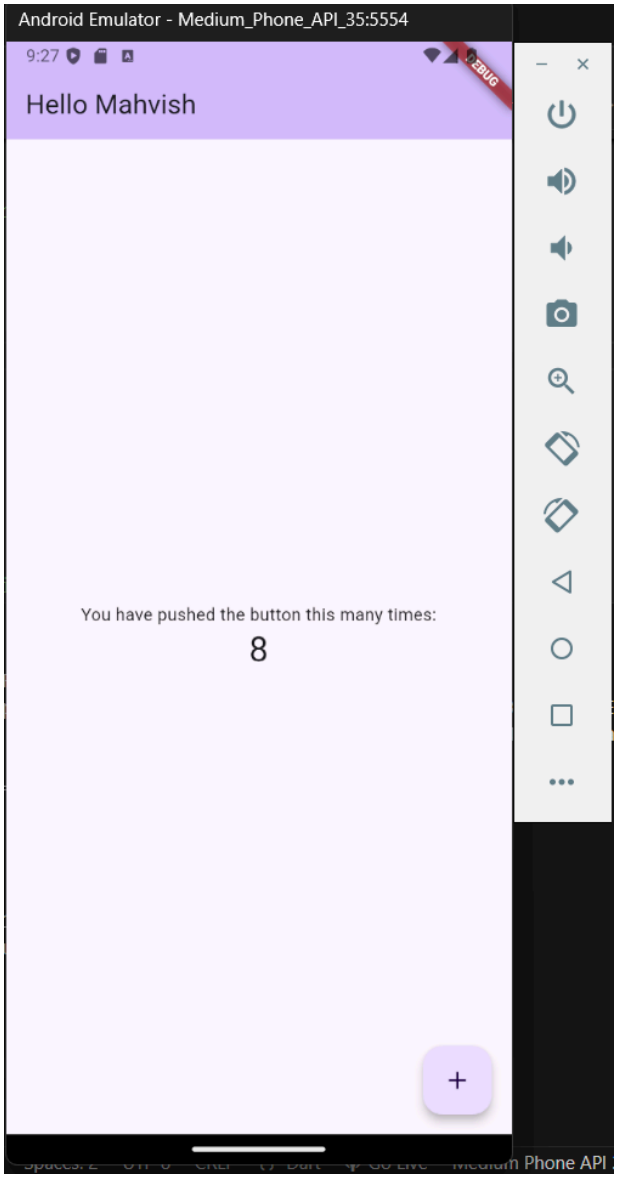
**Project Title:** **Roll No. 56**




Step 12: Next, click on new Flutter project and provide SDK path ->Next.
Step 12.1: Next, provide name of the project -> select project location -> create.

**Project Title:** **Roll No. 56**



# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 02 |
| Experiment Title. | To design Flutter UI by including common widgets. |
| Roll No. | 56 |
| Name | Mahvish Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation |
| Grade: | |

# Experiment No. 2

**AIM** : To design Flutter UI by including common widgets

## Theory:

In Flutter, designing UIs involves combining various widgets to build interactive and visually appealing applications. Here's a more detailed overview of key concepts:

1. **Widgets in Flutter**: Everything in Flutter is a widget. Widgets are the building blocks of the UI. There are two main types:
    - **Stateless Widgets**: These are immutable and don't change over time. They are responsible for displaying UI based on fixed data or input.
    - **Stateful Widgets**: These can change their state over time. They are dynamic and are used when the UI needs to update in response to user interaction or other factors.

2. **Layout Widgets**: The layout of your UI is primarily constructed using widgets like:
    - **Container**: A versatile widget used to hold other widgets and apply styling such as padding, margin, colors, and shapes.
    - **Column**: A widget that arranges its children vertically. It's useful for stacking widgets in a vertical list.
    - **Row**: A widget that arranges its children horizontally. It's useful for placing widgets side by side.
    - **Expanded**: A widget that can be used inside a Column, Row, or Flex to make child widgets flexible and fill available space.

3. **Text and Icons**:
    - **Text**: The Text widget is used to display static or dynamic text. It can be styled with custom fonts, sizes, colors, and more.
    - **Icon**: Flutter provides a large set of material design icons, and the Icon widget lets you display them in various sizes and colors.

4. **Buttons and User Interactions**:

    - Flutter provides multiple button widgets like **ElevatedButton**, **TextButton**, and **IconButton** to handle user interaction. These widgets can trigger actions when tapped.
    - **TextField**: Used for user input. You can configure it to accept different types of text, such as email or password.

- **Checkbox**, **Radio**, and **Switch**: Used for boolean selections, allowing users to choose options in forms or settings.

5. **Navigation**:
   - Flutter's **Navigator** widget is responsible for managing routes or screens. You use Navigator.push to navigate to a new screen, and Navigator.pop to return to the previous one.
   - **Routes** define the pages of an app, and you can pass data between them using arguments.

6. **Displaying Lists and Grids**:
   - **ListView**: The ListView widget is used to display a list of items that can scroll. It's perfect for long lists that need to be dynamically generated.
   - **GridView**: This widget allows you to display items in a grid format, with configurable row and column layouts.

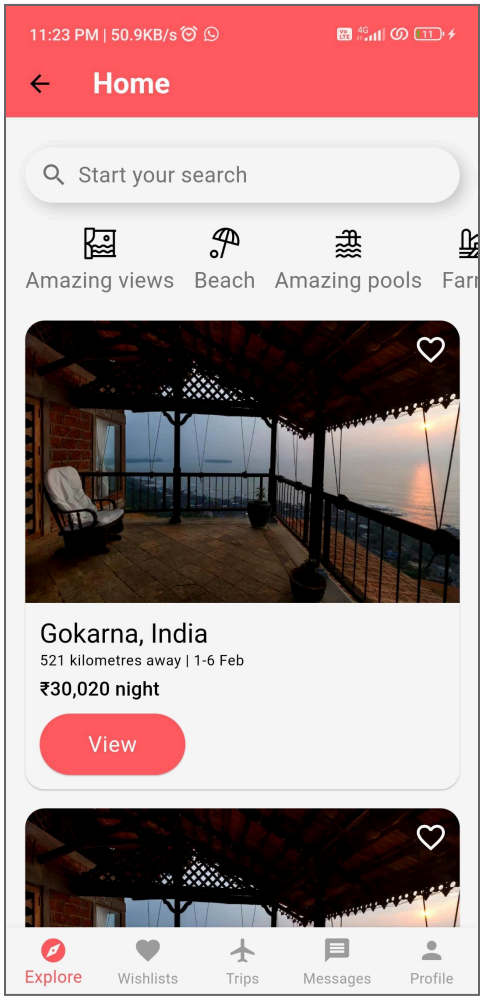
7. **State Management**:
   - Flutter provides a variety of ways to manage state. The simplest approach is using setState() to update the UI. For more complex apps, you can use state management solutions like **Provider**, **Riverpod**, or **Bloc** to separate business logic from UI code.
   - Proper state management ensures your UI stays in sync with the underlying data, especially in interactive or dynamic applications.

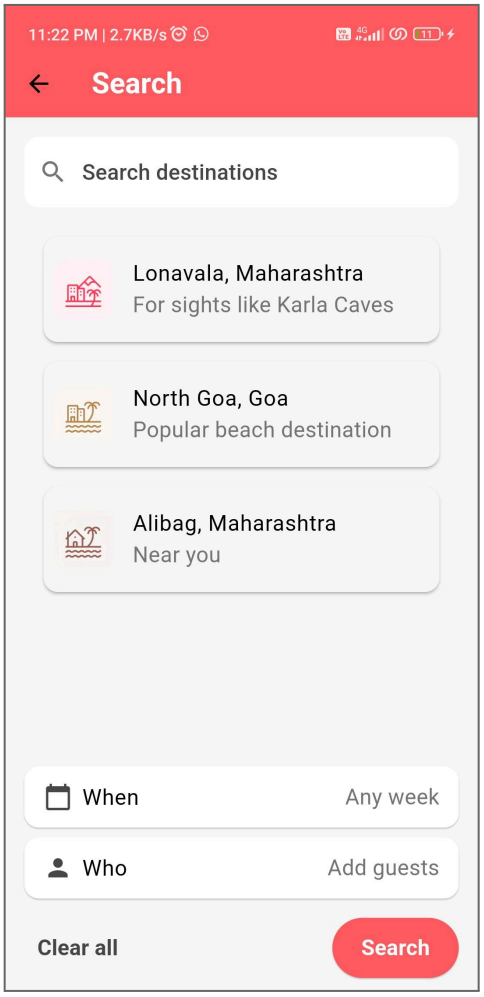
8. **Theming and Styling**:
   - Flutter allows you to define a global **Theme** for your app using ThemeData, which ensures consistent styling across the entire app. You can customize colors, typography, and button styles.

9. **Animations and Transitions**:
   - Flutter provides powerful animation support to create smooth and visually appealing transitions between UI states.
   - **AnimatedContainer**: A widget that animates changes in properties like width, height, or color over a given duration.
   - You can also create custom animations using **AnimationController** and **Tween**.

**Project Title:** **Roll No. 56**

## Screenshots:



## Code Snippets:

**Scaffold & Column Widget**

```
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(
          "Home",
          style: TextStyle(color: Colors.white),
        ),
        backgroundColor: primaryColor,
      ),
      body: Padding(
        padding: const EdgeInsets.only(top: 25.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 16.0),
              child: GestureDetector(
                onTap: () => Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => SearchPage()),
                ),
                child: SearchBar(),
              ),
            ),
            const SizedBox(height: 20),
            CategoryTabs(),
            Expanded(child: PropertyList()),
          ],
        ),
      ),
      bottomNavigationBar: BottomNavBar(),
    );
  }
}
```

**Project Title:** **Roll No. 56**

**Row Widget**

```
Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            TextButton(
              onPressed: () {},
              child: Text("Clear all", style: TextStyle(color: darkgrey)),
            ),
            ElevatedButton(
              style: ElevatedButton.styleFrom(
                backgroundColor: primaryColor,
                padding: EdgeInsets.symmetric(horizontal: 24, vertical: 12),
              ),
              onPressed: () {},
              child: Text("Search", style: TextStyle(color: Colors.white)),
            ),
          ],
        ),
```

**Project Title:** **Roll No. 56**

**Search Bar - Container widget**

```
class SearchBar extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.symmetric(horizontal: 12, vertical: 10),
      decoration: BoxDecoration(
        color: Theme.of(context).colorScheme.surface,
        borderRadius: BorderRadius.circular(30),
        boxShadow: [
          BoxShadow(
            color: Colors.black.withOpacity(0.2),
            blurRadius: 10,
            offset: Offset(4, 4),
          ),
        ],
      ),
      child: Row(
        children: [
          Icon(Icons.search, color: Colors.black54),
          SizedBox(width: 8),
          Text("Start your search",
              style: Theme.of(context).textTheme.bodyMedium),
        ],
      ),
    );
  }
}
```

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 03 |
| Experiment Title. | To include icons, images, fonts in Flutter app |
| Roll No. | 56 |
| Name | Mahvish Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation |
| Grade: | |

**Project Title:** **Roll No. 56**

# Experiment No. 3

**AIM** : To include icons, images, fonts in Flutter app

## Theory:

To include icons, images, and fonts in a Flutter app, you need to understand the following core concepts related to asset management in Flutter. Here's the theory behind including these resources:

### 1. Assets in Flutter:

Assets are files or resources such as images, fonts, icons, or sounds that you include in your app and bundle within the app package. In Flutter, you can include these assets in your project and then use them in your app.

### 2. Adding Assets to pubspec.yaml:

In Flutter, you declare assets in the pubspec.yaml file. This is where you specify which assets should be bundled with your app during the build process.

**Example for adding assets:**

```
flutter:

  assets:

    - assets/images/

    - assets/icons/
```

In this example, the images are stored in the assets/images directory, and icons in the assets/icons directory. You can also specify specific files instead of directories.

### 3. Including Images:

Flutter provides several ways to include images in your app, including network images, asset images, and file images. To use asset images, you reference them by their file path relative to the assets directory.

**Example of including an asset image:**

```
Image.asset('assets/images/my_image.png')
```

For this to work, the image (my_image.png) must be listed in the pubspec.yaml file under the flutter section, like this:

```
flutter:
  assets:
    - assets/images/my_image.png
```

## 4. Including Icons:

Flutter allows you to use custom icons in your app. You can add icon files (e.g., .png or .svg) to your assets folder and use them in the app. Alternatively, Flutter provides built-in icons via the Icons class.

**Example of using an asset icon:**

```
Image.asset('assets/icons/my_icon.png')
```

## 5. Including Fonts:

To include custom fonts, you place your font files (e.g., .ttf or .otf files) in a folder inside your assets directory. Then, you declare these fonts in the pubspec.yaml file and use them in your app.

**Example of adding custom fonts in pubspec.yaml:**

```
flutter:
  fonts:
    - family: CustomFont
      fonts:
        - asset: assets/fonts/CustomFont-Regular.ttf
        - asset: assets/fonts/CustomFont-Bold.ttf
```

**Example of using the custom font in Flutter:**

```
Text(
  'Hello, World!',
  style: TextStyle(fontFamily: 'CustomFont', fontSize: 20),
)
```

## 6. Font Weight and Style:

When specifying fonts, you can also define specific font weights and styles (like bold, italic) to support different text styles in your app.

## 7. Working with Icon Libraries:

While you can use custom icon files, Flutter also supports popular icon libraries like **FontAwesome**, **MaterialIcons**, etc. For example, Flutter's built-in Icons class provides access to the Material Design icons.

**Example of using a Material icon:**

```
Icon(Icons.home)
```

## 8. Caching and Optimization:

- **Images**: Flutter caches images, but you might want to use libraries like cached_network_image for better image loading and caching.
- **Fonts**: Custom fonts are loaded from assets when the app is first started, and they remain available for the lifecycle of the app.

## 9. SVG Images:

If you want to use vector-based images (like SVG), you can include them using packages like flutter_svg, which allows you to display scalable vector graphics (SVG files) in Flutter.

**Example of including an SVG:**

```
import 'package:flutter_svg/flutter_svg.dart';

SvgPicture.asset('assets/icons/my_icon.svg')
```

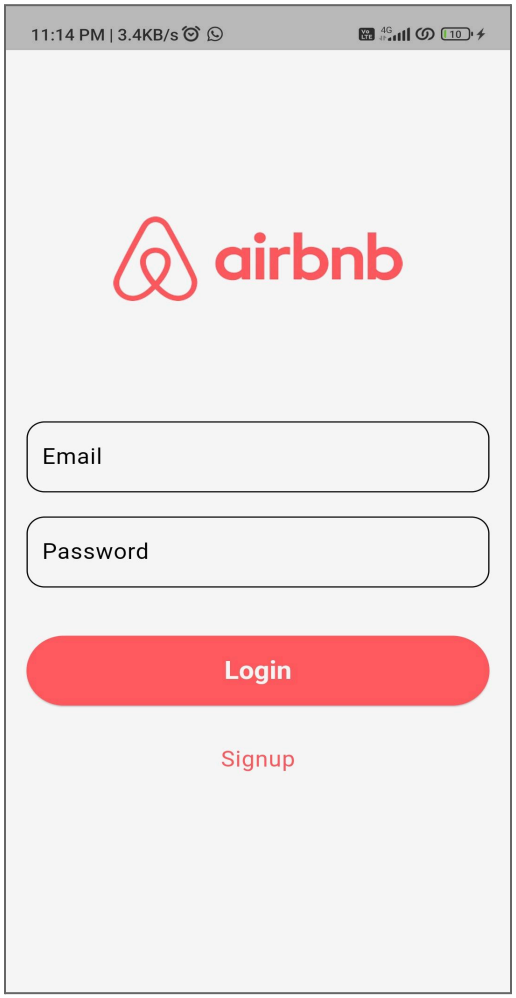
**Project Title:** **Roll No. 56**

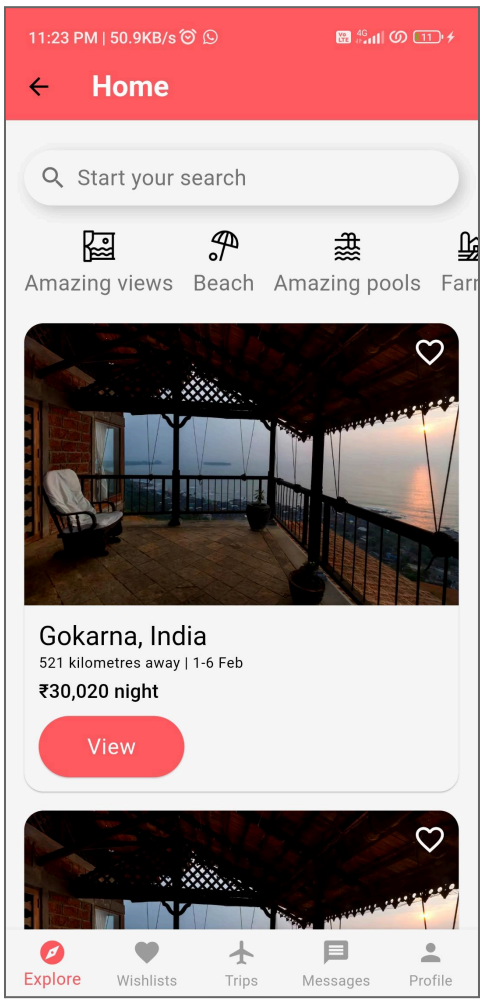
## Summary of Key Steps:

1. **Declare assets in pubspec.yaml.**
2. **For images and icons**, use Image.asset('path/to/image') and Image.asset('path/to/icon').
3. **For fonts**, declare them under the flutter section in pubspec.yaml, then use them via the TextStyle class.
4. **Use Icon Libraries**: Use Flutter's built-in Icons class or third-party icon libraries.
5. **For SVGs**, use packages like flutter_svg.

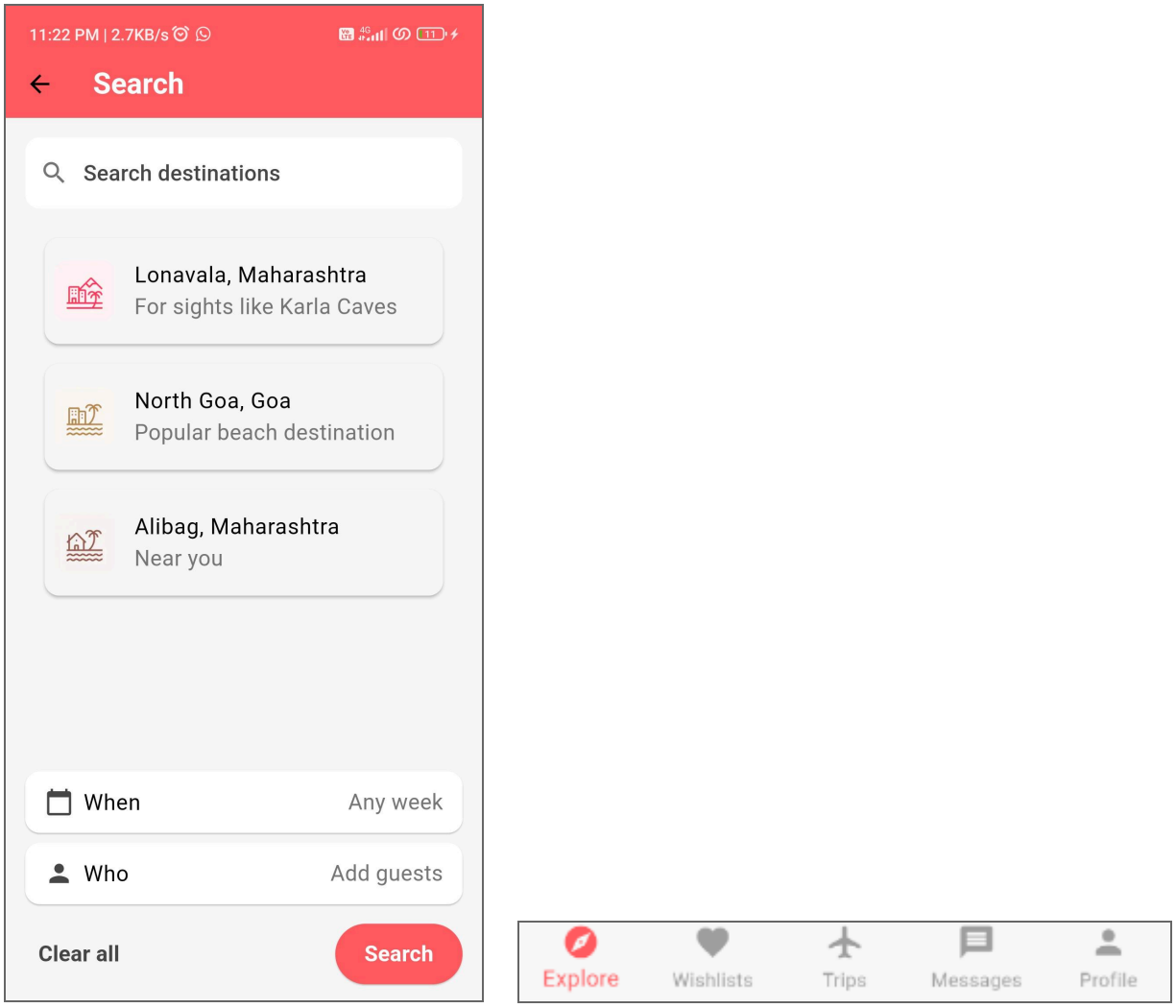
Understanding these concepts will help you effectively use images, icons, and fonts in your Flutter app.

## Screenshots:



**Project Title:** **Roll No. 56**




## Code Snippets:

Icons

1. Search Icon

```
child: Row(
      children: [
        Icon(Icons.search, color: Colors.black54),
        SizedBox(width: 8),
        Text("Start your search",
           style: Theme.of(context).textTheme.bodyMedium),
      ])
```

2. Icons for bottom Navigation

```
BottomNavigationBarItem(icon: Icon(Icons.explore), label: "Explore"),
BottomNavigationBarItem(icon: Icon(Icons.favorite), label: "Wishlists"),
BottomNavigationBarItem(icon: Icon(Icons.airplanemode_active), label: "Trips"),
BottomNavigationBarItem(icon: Icon(Icons.message), label: "Messages"),
BottomNavigationBarItem(icon: Icon(Icons.person), label: "Profile"),
```

Images

1. 
```
Image.asset(
  "assets/images/logo-airbnb.png",
  height: 70,
  fit: BoxFit.cover,
),
```

2. 
```
Image.asset(
    "assets/images/property.png",
    width: double.infinity,
    height: 235,
    fit: BoxFit.cover,
  ),
```

3. 
```
final List<Map<String, String>> categories = [
{"name": "Amazing views", "icon": "assets/icons/airbnb_views.jpg"},
{"name": "Beach", "icon": "assets/icons/airbnb_beach.jpg"},
{"name": "Amazing pools", "icon": "assets/icons/airbnb_pool.jpg"},
{"name": "Farms", "icon": "assets/icons/airbnb_farm.jpg"}];

Image.asset(
    category["icon"]!,
    width: 28,
    height: 28,
    fit: BoxFit.cover),
```

Fonts

1. Add google fonts in pubspec.yaml

```
dependencies:
  flutter:
    sdk: flutter
  google_fonts: ^6.2.1
```

2. Import google fonts in themes

```
import 'package:google_fonts/google_fonts.dart';
```

3. Create a theme and add it to textTheme

```
final TextStyle montserratBodyTextStyle = GoogleFonts.montserrat(
  fontSize: 18,
  color: Colors.black,
);

 displaySmall: montserratBodyTextStyle.copyWith(fontSize: 21),
```

4. Use it wherever required

```
child: Text(
            'Login',
            style: Theme.of(context).textTheme.displaySmall?.copyWith(
                color: Theme.of(context).colorScheme.onPrimary,
              ),
          ),
```

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 04 |
| Experiment Title. | To create an interactive Form using form widget |
| Roll No. | 56 |
| Name | Mahvish Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation |
| Grade: | |

**Project Title:** **Roll No. 56**

# Experiment No. 4

**AIM** : To create an interactive Form using form widget

## Theory:

1. **Form Widget**:
    - The Form widget is a container for managing form-related interactions. It allows for validation and saving the form data.
    - It keeps track of the state of all the fields within the form (like TextFormField widgets) through a GlobalKey<FormState>.
2. **TextFormField**:
    - This is the main widget used for collecting user input (such as text). It integrates easily with form validation and submission.
    - You can apply validators to the TextFormField to ensure the input meets specific criteria (e.g., required fields, correct format).
3. **GlobalKey**:
    - A GlobalKey<FormState> is essential for managing the form's state (e.g., validating fields, saving data). It's assigned to the Form widget and can be used to trigger actions like form validation or saving the data.
4. **Validation**:
    - You can define validation rules on each form field. The TextFormField widget has a validator property, which allows you to write logic that will run whenever the form is validated.
    - A validator checks whether the input meets the required format (such as checking for valid email format or a non-empty field).
5. **Form Submission**:
    - When you're ready to submit the form, you can call formKey.currentState?.save() to trigger the save method for all fields or formKey.currentState?.validate() to check if all the fields pass the validation checks.
6. **Saving Data**:
    - After validation, data entered in the form fields can be saved to variables or used for further processing, such as sending it to a server.

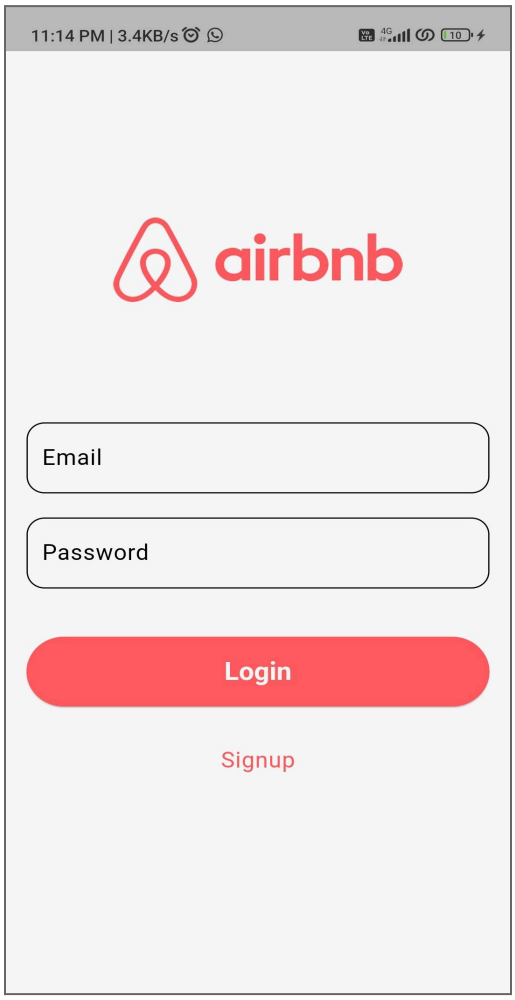
## Form Workflow:

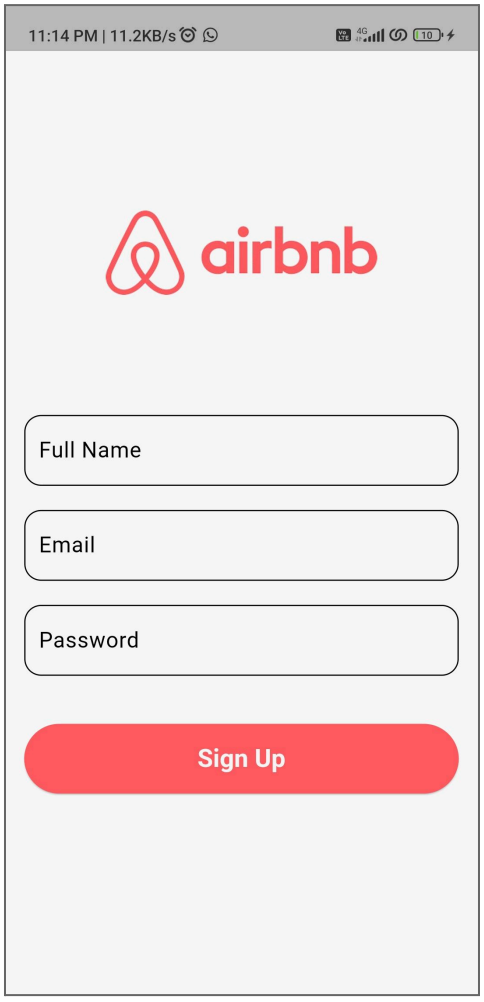
1. **Create a form**: Use the Form widget to wrap all input fields.
2. **Add form fields**: Use widgets like TextFormField to collect data.
3. **Set up validation**: Add validation logic to the form fields.

4. **Submit the form**: Trigger validation and handle the form submission.
5. **Save data**: Capture the entered data once validation passes.

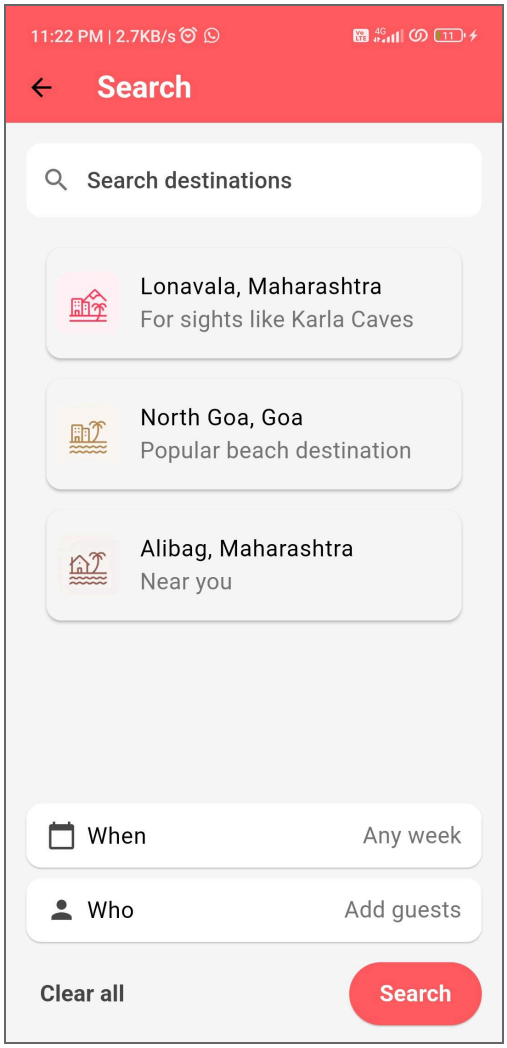
Flutter's form management tools are powerful, enabling you to handle complex forms with various input types, validations, and submissions effectively.

## Screenshots:



**Project Title:** **Roll No. 56**




## Code Snippets:

```
import 'package:airbnb/screens/signup.dart';
import 'package:flutter/material.dart';
import 'package:airbnb/theme.dart';
import 'home_page.dart';

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

  @override
  State<LoginPage> createState() => _LoginPageState();
}
```

```
class _LoginPageState extends State<LoginPage> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();

  String? _errorText;

  void _handleLogin() {
     }

  @override
  Widget build(BuildContext context) {
   return Scaffold(
     backgroundColor: backgroundColor,
     body: Padding(
       padding: const EdgeInsets.all(16.0),
       child: Column(
         mainAxisAlignment: MainAxisAlignment.center,
         children: [
           Image.asset(
             "assets/images/logo-airbnb.png",
             height: 70,
             fit: BoxFit.cover,
           ),
           const SizedBox(
             height: 100,
           ),
           TextField(
             controller: _emailController,
             decoration: InputDecoration(
               labelText: 'Email',
               labelStyle: myTheme.inputDecorationTheme.labelStyle,
               hintText: 'Enter your email',
               hintStyle: myTheme.inputDecorationTheme.hintStyle,
               border: myTheme.inputDecorationTheme.border,
               focusedBorder:
                   Theme.of(context).inputDecorationTheme.focusedBorder,
             ),
             keyboardType: TextInputType.emailAddress,
           ),
           const SizedBox(height: 20),
           TextField(
             controller: _passwordController,
             decoration: InputDecoration(
               labelText: 'Password',
```

```
    labelStyle: myTheme.inputDecorationTheme.labelStyle,
    hintText: 'Password',
    hintStyle: myTheme.inputDecorationTheme.hintStyle,
    border: myTheme.inputDecorationTheme.border,
    focusedBorder:
        Theme.of(context).inputDecorationTheme.focusedBorder,
  ),
  obscureText: true,
),
if (_errorText != null) ...[
  const SizedBox(height: 10),
  Text(_errorText!, style: TextStyle(color: Colors.red)),
],
const SizedBox(height: 40),
SizedBox(
  width: double.infinity,
  child: ElevatedButton(
    onPressed: _handleLogin,
    style: ElevatedButton.styleFrom(
      backgroundColor: primaryColor,
      padding:
          const EdgeInsets.symmetric(vertical: 16, horizontal: 40),
    ),
    child: Text(
      'Login',
      style: Theme.of(context).textTheme.displaySmall?.copyWith(
            color: Theme.of(context).colorScheme.onPrimary,
          ),
    ),
  ),
),
const SizedBox(height: 20),
SizedBox(
  width: double.infinity,
  child: TextButton(
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => const SignUpPage()),
      );
    },
    child: Text(
      'Signup',
      style: Theme.of(context).textTheme.bodyLarge?.copyWith(
```

```
                    color: primaryColor,
                  ),
                ),
              ),
            )
          ],
        ),
      ),
    );
  }
}

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

  @override
  State<SignUp> createState() => _SignUpState();
}

class _SignUpState extends State<SignUp> {
  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 05 |
| Experiment Title. | To apply navigation, routing and gestures in Flutter App |
| Roll No. | 56 |
| Name | Mahvish Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation |
| Grade: | |

# Experiment No. 5

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

## Theory:

### 1. Navigation and Routing

In Flutter, navigation refers to moving from one screen (or "route") to another. There are several key concepts:

- **Routes**: These are the different screens or pages in your app. Every route is typically represented by a Widget in Flutter. The default route is usually the home screen of the app, but you can define multiple routes for different screens.
- **Navigator**: This is a widget that manages a stack of routes. You can "push" a new route onto the stack to navigate to another screen, or "pop" the top route off to go back.
- **Named Routes**: These are routes that are identified by a string. Instead of pushing or popping routes directly, you can refer to routes by their name (e.g., /home, /settings).
- **Custom Route Transitions**: Flutter allows you to define custom animations and transitions when navigating between routes. You can create smooth, custom page transitions using PageRouteBuilder.
- **Route Arguments**: You can pass data between routes using arguments. This is particularly useful when navigating to a screen that requires specific data (e.g., opening a product page with product details).

### 2. Gestures in Flutter

Gestures are interactions that a user performs with the screen, such as taps, swipes, or long presses. Flutter provides a flexible way to detect these gestures.

- **GestureDetector**: This is the most commonly used widget for detecting gestures. You can wrap it around any widget to detect gestures like tap, double tap, long press, swipe, and others.
- **Tap Gesture**: A simple touch interaction, typically detected using onTap or onLongPress callbacks.
- **Swipe Gestures**: Swiping is usually detected via onHorizontalDragUpdate, onVerticalDragUpdate, or onPanUpdate. These allow you to track the user’s finger movement and respond accordingly.

- **Custom Gesture Detection**: Flutter also allows you to implement more complex gestures. For example, you can detect drag gestures to create features like a sliding menu or draggable elements.

- **Dismissible Widget**: This widget enables swipe-to-dismiss behavior, commonly used for items in a list that users can swipe left or right to remove.

## 3. Managing Navigation and Gestures Together

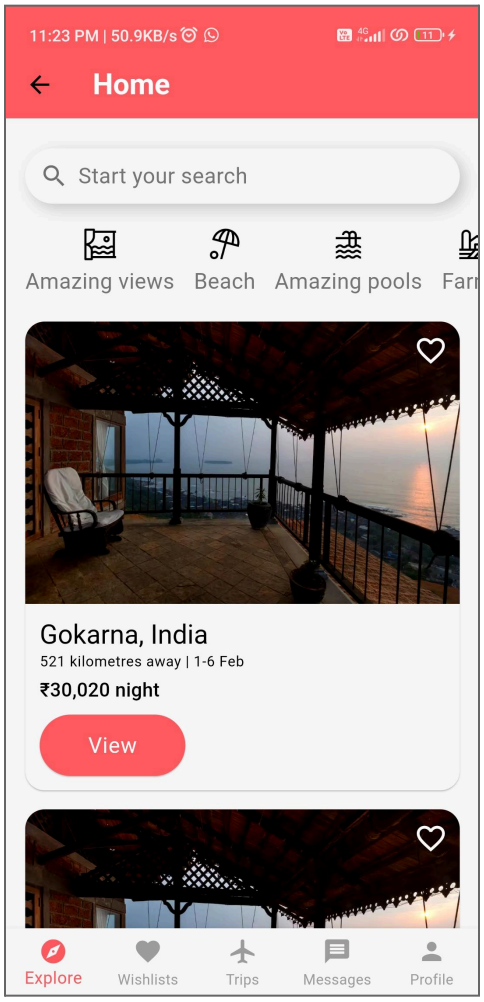
When you combine navigation with gestures, you can create more interactive and dynamic UIs. For instance, a user could swipe to navigate between screens, or tap a button that triggers navigation while performing a gesture on a different part of the screen.

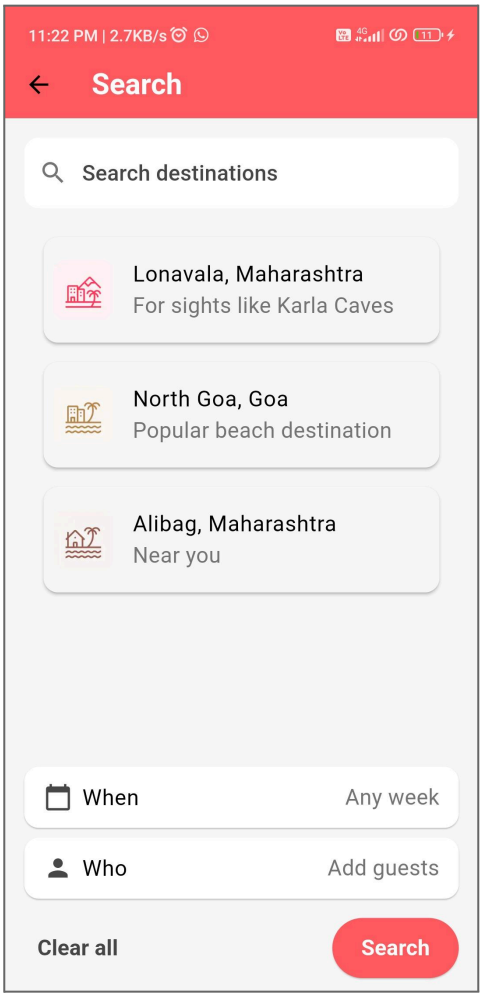
## 4. Back Button Handling

On Android devices, there is a system-wide back button that users can press to navigate backward. Flutter provides a way to intercept and customize this behavior using WillPopScope, allowing you to decide what happens when the user tries to go back (e.g., prevent the user from leaving the current screen, show a confirmation dialog, or allow normal back navigation).

**Project Title:** **Roll No. 56**

## Screenshots:



## Code Snippets:

1. Gesture Detector

```
child: GestureDetector(
            onTap: () => Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => SearchPage()),
            ),
            child: SearchBar(),
          ),
```

2. Using Navigator.push

```
Navigator.pushReplacement(
      context,
      MaterialPageRoute(builder: (context) => HomeScreen()),
    );

Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => const LoginPage()),
            );
```

```
import 'package:airbnb/screens/login.dart';
import 'package:flutter/material.dart';
import 'package:airbnb/theme.dart';

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

  @override
  State<SignUpPage> createState() => _SignUpPageState();
}

class _SignUpPageState extends State<SignUpPage> {
  final TextEditingController _nameController = TextEditingController();
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();

  String? _errorText;

  @override
  Widget build(BuildContext context) {
   return Scaffold(
     backgroundColor: backgroundColor,
     body: Padding(
       padding: const EdgeInsets.all(16.0),
       child: Column(
         mainAxisAlignment: MainAxisAlignment.center,
         children: [
           Image.asset(
             "assets/images/logo-airbnb.png",
             height: 70,
             fit: BoxFit.cover,
           ),
           const SizedBox(height: 100),
           TextField(
             controller: _nameController,
             decoration: InputDecoration(
               labelText: 'Full Name',
               labelStyle: myTheme.inputDecorationTheme.labelStyle,
               hintText: 'Enter your full name',
               hintStyle: myTheme.inputDecorationTheme.hintStyle,
               border: myTheme.inputDecorationTheme.border,
```

```
    focusedBorder:
        Theme.of(context).inputDecorationTheme.focusedBorder,
  ),
),
const SizedBox(height: 20),
TextField(
  controller: _emailController,
  decoration: InputDecoration(
    labelText: 'Email',
    labelStyle: myTheme.inputDecorationTheme.labelStyle,
    hintText: 'Enter your email',
    hintStyle: myTheme.inputDecorationTheme.hintStyle,
    border: myTheme.inputDecorationTheme.border,
    focusedBorder:
        Theme.of(context).inputDecorationTheme.focusedBorder,
  ),
  keyboardType: TextInputType.emailAddress,
),
const SizedBox(height: 20),
TextField(
  controller: _passwordController,
  decoration: InputDecoration(
    labelText: 'Password',
    labelStyle: myTheme.inputDecorationTheme.labelStyle,
    hintText: 'Enter your password',
    hintStyle: myTheme.inputDecorationTheme.hintStyle,
    border: myTheme.inputDecorationTheme.border,
    focusedBorder:
        Theme.of(context).inputDecorationTheme.focusedBorder,
  ),
  obscureText: true,
),
if (_errorText != null) ...[
  const SizedBox(height: 10),
  Text(_errorText!, style: TextStyle(color: Colors.red)),
],
const SizedBox(height: 40),
SizedBox(
  width: double.infinity,
  child: ElevatedButton(
    onPressed: () {
```

```
                  Navigator.push(
                    context,
                    MaterialPageRoute(builder: (context) => const LoginPage()),
                  );
                },
                style: ElevatedButton.styleFrom(
                  backgroundColor: primaryColor,
                  padding:
                      const EdgeInsets.symmetric(vertical: 16, horizontal: 40),
                ),
                child: Text(
                  'Sign Up',
                  style: Theme.of(context).textTheme.displaySmall?.copyWith(
                        color: Theme.of(context).colorScheme.onPrimary,
                      ),
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 06 |
| Experiment Title. | To Connect Flutter UI with fireBase database |
| Roll No. | 56 |
| Name | Mahvish Siddiqui |
| Class | D15A/D15B |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS |
| Grade: | |

# Experiment No. 6

**AIM** : To connect flutter UI with firebase database

## Theory:

Introduction to Firebase and Flutter Integration

Firebase is a comprehensive platform developed by Google, designed to help developers build high-quality applications for both mobile and web. It provides essential services such as real-time databases, authentication, cloud storage, hosting, and much more. One of the most widely used Firebase services is the Firebase Realtime Database, which is a NoSQL cloud database that allows data to be stored and synced in real-time across all connected devices. Flutter, on the other hand, is an open-source UI software development kit created by Google, which allows developers to build natively compiled applications for mobile, web, and desktop from a single codebase. Its rich set of pre-designed widgets and powerful tools makes Flutter an attractive option for developing visually appealing and performant applications. Integrating Firebase with Flutter allows developers to leverage the full potential of Firebase services in their applications. By using Firebase's Realtime Database, Flutter apps can achieve features such as real-time data synchronization, secure authentication, and cloud-based storage. This combination enables developers to create powerful, scalable, and feature-rich mobile and web applications.

**Firebase Realtime Database Overview**

Firebase Realtime Database is a cloud-hosted NoSQL database that stores data in a JSON-like format. The key characteristic of this database is its real-time synchronization feature, meaning that any changes made to the database are instantly reflected on all clients (i.e., devices) connected to it. This makes it an ideal solution for applications that require frequent updates and need to maintain synchronized data across multiple users or devices, such as messaging apps, social media platforms, or collaborative tools. The Firebase Realtime Database is structured as a tree of data, where each node in the tree can contain key value pairs. This structure allows for easy data retrieval and modification. Firebase's real-time capabilities enable apps to immediately receive updates to the data whenever it changes, without the need to refresh or reload the page. Additionally, the database supports offline data persistence, meaning that even if the user's device loses its internet connection, the app can still function by using the locally cached data.

**Setting Up Firebase in Flutter :**

To connect a Flutter app with Firebase, the following steps are typically followed:

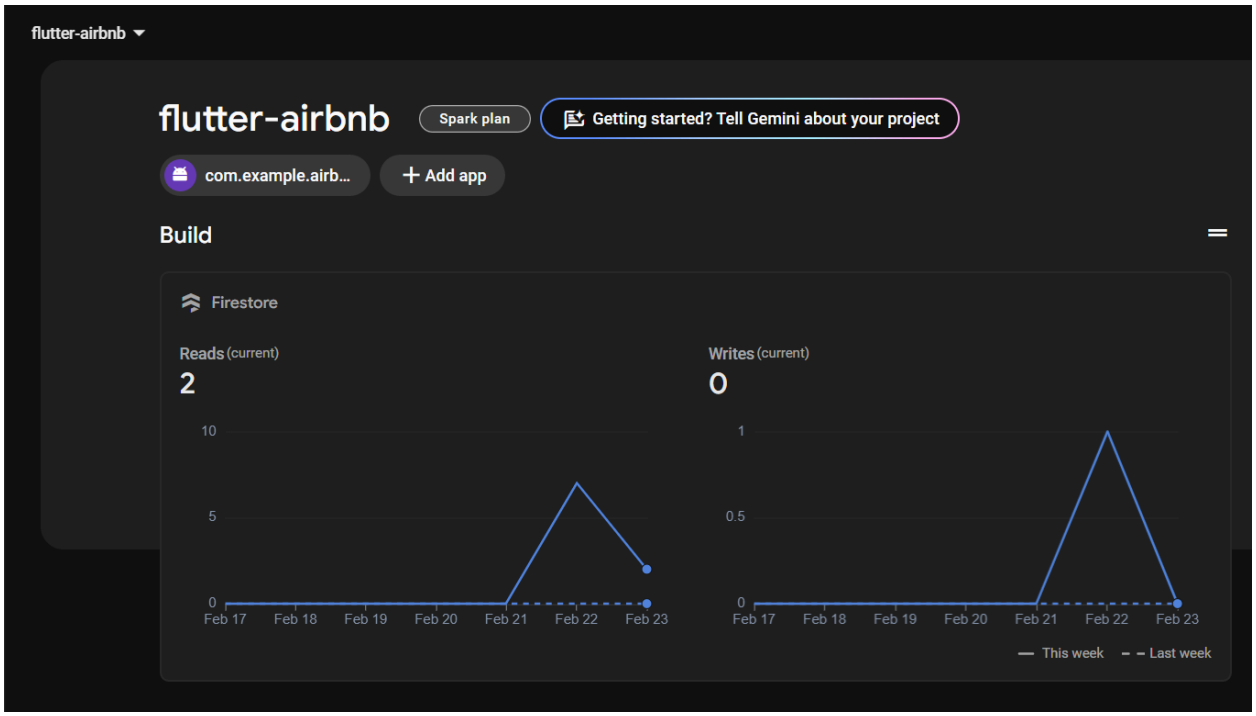
1. Creating a Firebase Project: To start using Firebase with Flutter, the first step is to create a Firebase project in the Firebase Console. Once the project is created, developers can associate their Flutter app with the Firebase project by following the platform-specific instructions for Android or iOS. This usually involves configuring API keys, downloading configuration files, and adding them to the Flutter project.

2. Integrating Firebase SDK in Flutter: After the Firebase project is set up, developers need to integrate Firebase's SDK into the Flutter app. This involves adding the necessary dependencies to the Flutter project's pubspec.yaml file. For Firebase's Realtime Database, the package firebase_database is used. Additionally, Firebase's core SDK (firebase_core) must also be included to initialize Firebase services.

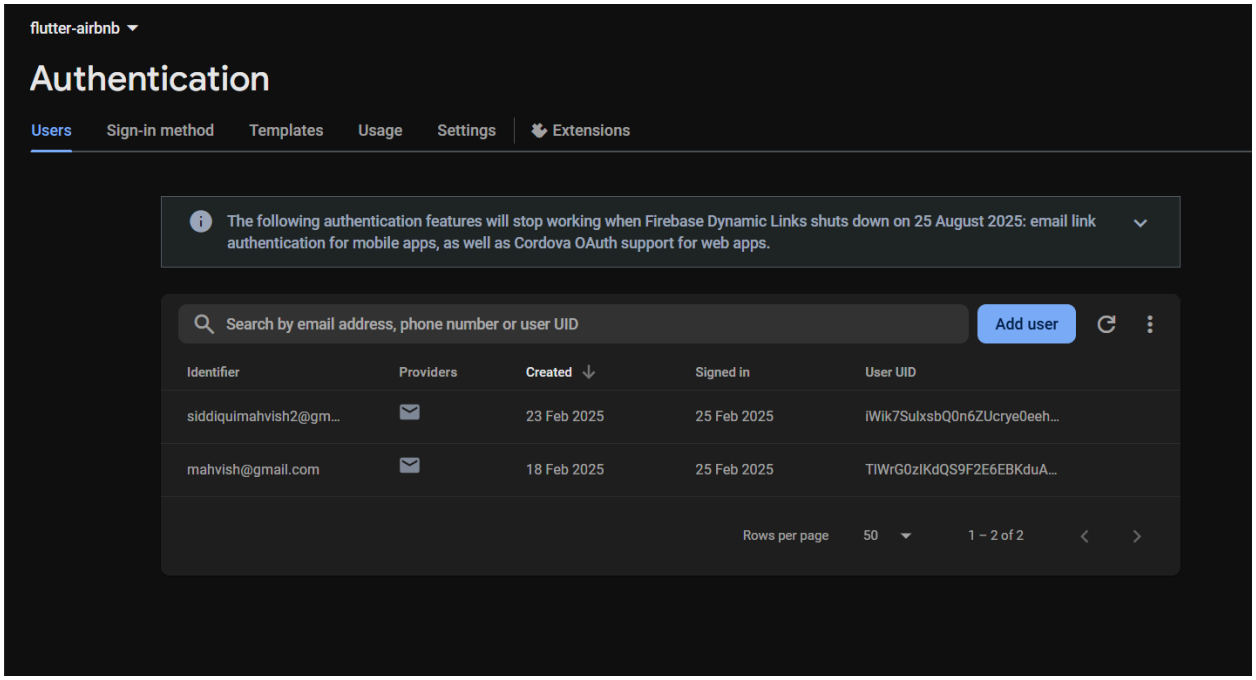
3. Initializing Firebase: Before any Firebase functionality can be used, it is essential to initialize Firebase in the Flutter app. This is done by calling Firebase.initializeApp() in the main entry point of the app (usually in the main.dart file). Firebase needs to be initialized before interacting with any Firebase services, such as the Realtime Database, Cloud Firestore, or Authentication.

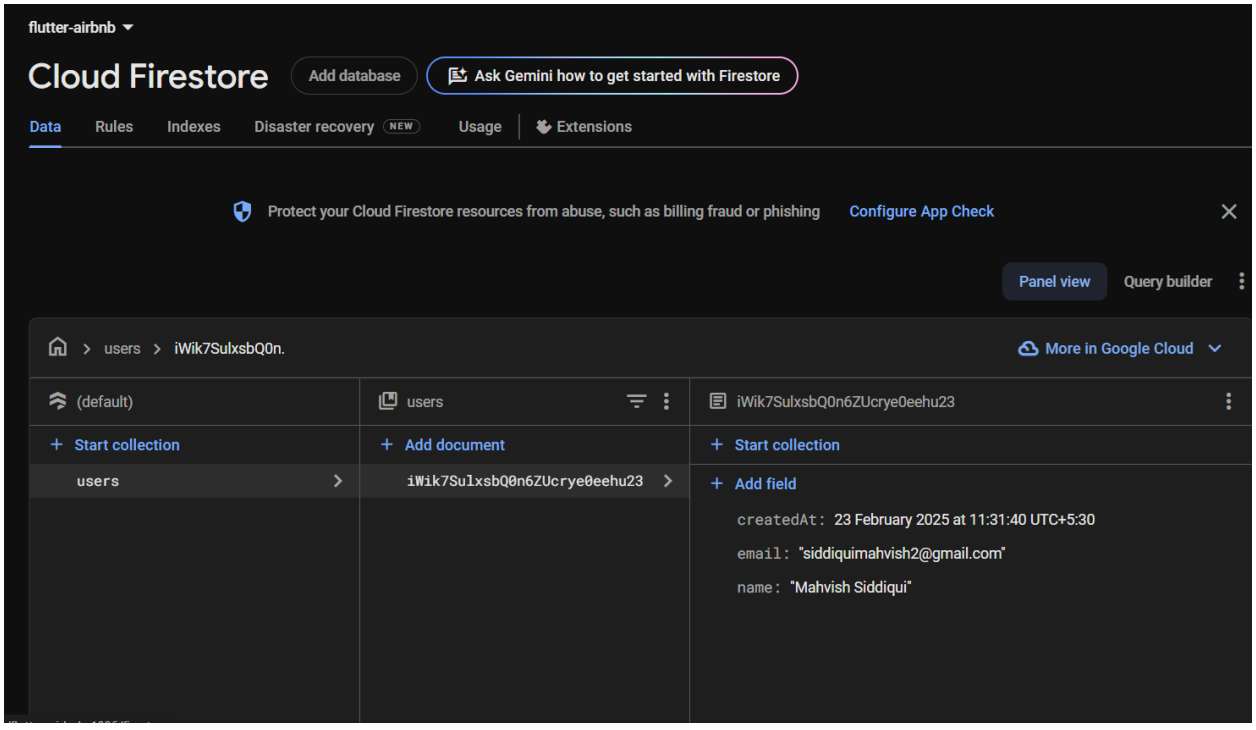
Connecting Firebase to a Flutter app enables developers to create robust, scalable, and real-time applications with ease. Firebase's Realtime Database offers a powerful, cloud-based solution for managing data in realtime, while Firebase Authentication ensures secure access control. By integrating Firebase with Flutter, developers can take advantage of real-time data synchronization, offline support, and a wide range of other Firebase features, allowing them to build feature-rich apps that meet modern user expectations.

## Screenshots:

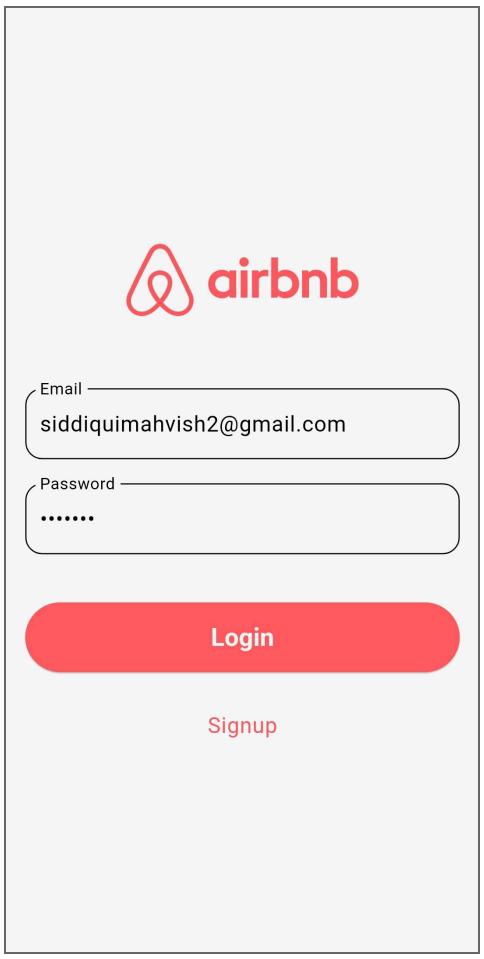


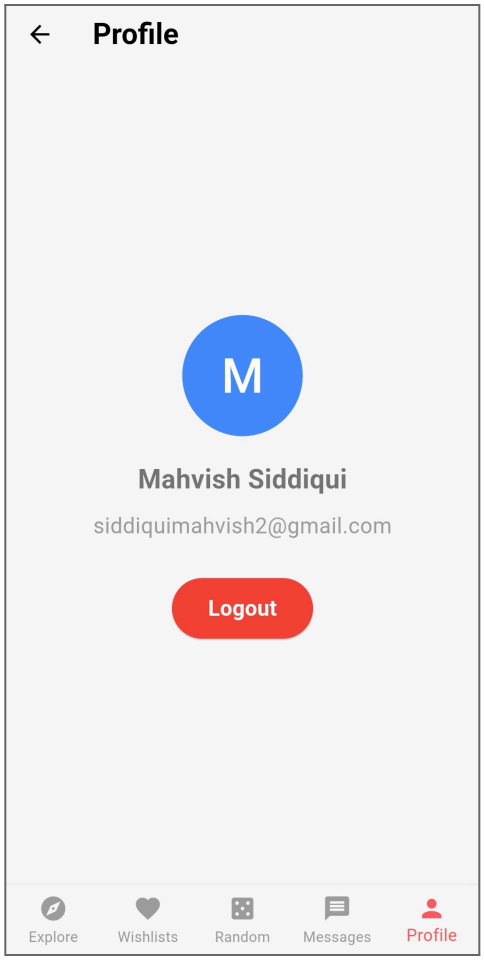
**Project Title:** **Roll No. 56**



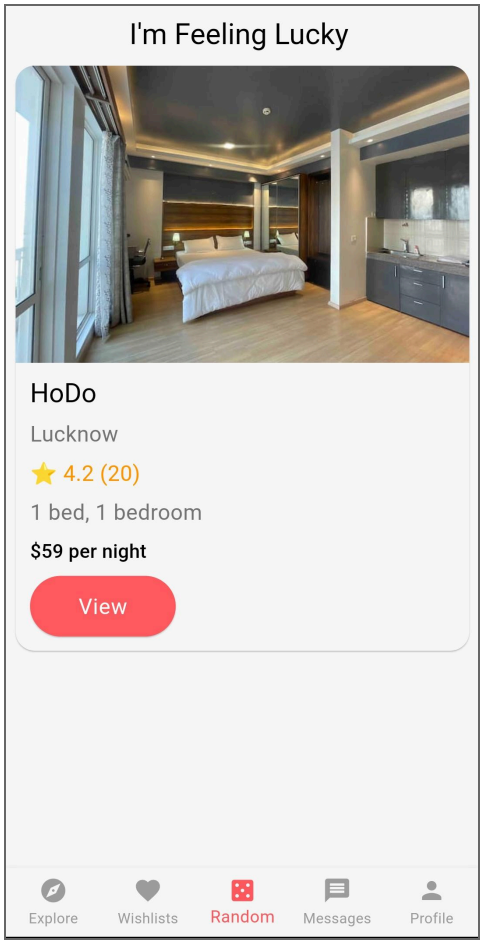
**Project Title:** **Roll No. 56**



**Project Title:**

**Roll No. 56**



## Code Snippets:

**Setup in main.dart**

```
import 'package:airbnb/firebase_options.dart';
import 'package:firebase_core/firebase_core.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(const MyApp());
}
```

**Login Function**

```
Future<void> _handleLogin() async {
    String email = _emailController.text.trim();
    String password = _passwordController.text.trim();

    try {
      await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: email,
        password: password,
      );

      Navigator.pushReplacement(
      context,
      MaterialPageRoute(builder: (context) => BottomNavBar()),
    );
    } catch (e) {

      setState(() {
        _errorText = "Invalid email or password. Please try again.";
      });
    }
  }
```

**Signup Function**

```
Future<void> _handleLogin() async {
   String email = _emailController.text.trim();
   String password = _passwordController.text.trim();

   try {
     await FirebaseAuth.instance.signInWithEmailAndPassword(
       email: email,
       password: password,
     );

     Navigator.pushReplacement(
     context,
     MaterialPageRoute(builder: (context) => BottomNavBar()),
   );
   } catch (e) {

     setState(() {
       _errorText = "Invalid email or password. Please try again.";
     });
   }
 }
```

**Profile Page - Logout**

```
import 'package:airbnb/screens/login.dart';
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

class ProfileScreen extends StatelessWidget {
  Future<String?> _fetchUserName(String uid) async {
   try {
     DocumentSnapshot userDoc =
         await FirebaseFirestore.instance.collection('users').doc(uid).get();

     return userDoc.exists ? userDoc['name'] : null;
   } catch (e) {
     print("Error fetching name: $e");
```

```
    return null;
  }
 }

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

  return Scaffold(
   appBar: AppBar(
    title: const Text("Profile"),
   ),
   body: Padding(
    padding: const EdgeInsets.all(16.0),
    child: Center(
     child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
       CircleAvatar(
        radius: 50,
        backgroundColor: Colors.blueAccent,
        child: FutureBuilder<String?>(
         future: _fetchUserName(user!.uid),
         builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
           return const CircularProgressIndicator(color: Colors.white);
          }
          String userName = snapshot.data ?? "?";
          return Text(
           userName.isNotEmpty ? userName[0].toUpperCase() : "?",
           style: const TextStyle(fontSize: 40, color: Colors.white),
          );
         },
        ),
       ),
       const SizedBox(height: 20),
       FutureBuilder<String?>(
        future: _fetchUserName(user.uid),
        builder: (context, snapshot) {
         if (snapshot.connectionState == ConnectionState.waiting) {
          return const CircularProgressIndicator();
```

```
            }
            return Text(
              snapshot.data ?? "No Name",
              style: const TextStyle(fontSize: 22, fontWeight: FontWeight.bold),
            );
          },
        ),
        const SizedBox(height: 10),
        Text(
          user.email ?? "No Email",
          style: const TextStyle(fontSize: 18, color: Colors.grey),
        ),
        const SizedBox(height: 30),
        ElevatedButton(
          onPressed: () async {
            await FirebaseAuth.instance.signOut();
            Navigator.of(context).pushAndRemoveUntil(
              MaterialPageRoute(builder: (context) => LoginPage()),
              (Route<dynamic> route) => false,
            );
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.red,
            padding: const EdgeInsets.symmetric(horizontal: 30, vertical: 12),
          ),
          child: const Text("Logout",
              style: TextStyle(fontSize: 18, color: Colors.white)),
        ),
      ],
    ),
  ),
 ),
 );
 }
}
```

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 07 |
| Experiment Title. | To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”. |
| Roll No. | 56 |
| Name | Mahvish Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO4: Understand various PWA frameworks and their requirements |
| Grade: | |

**Project Title:** **Roll No. 56**

# Experiment No. 7

**Title**: To write meta data of your Ecommerce PWA

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

**Theory:**

Regular Web App

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

Progressive Web App

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

Difference between PWAs vs. Regular Web Apps:

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

**1. Native Experience**

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

**2. Ease of Access**

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

**3. Faster Services**

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

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

**4. Engaging Approach**
As already shared, the PWAs can employ push notifications and other native device features more efficiently. Their interaction does not depend on the browser user uses. This eventually improves the chances of notifying the user regarding your services, offers, and other options related to your brand and keeping them hooked to your brand. In simpler words, PWAs let you maintain the user engagement and retention rate.

**5. Updated Real-Time Data Access**
Another plus point of PWAs is that these apps get updated on their own. They do not demand the end-users to go to the App Store or other such platforms to download the update and wait until installed.
In this app type, the web app developers can push the live update from the server, which reaches the apps residing on the user's devices automatically. Therefore, it is easier for the mobile app developer to provide the best of the updated functionalities and services to the end-users without forcing them to update their app.

**6. Discoverable**
PWAs reside in web browsers. This implies higher chances of optimizing them as per the Search Engine Optimization (SEO) criteria and improving the Google rankings like that in websites and other web apps.

**7. Lower Development Cost**
Progressive web apps can be installed on the user device like a native device, but it does not demand submission on an App Store. This makes it far more cost-effective than native mobile applications while offering the same set of functionalities.

Pros and cons of the Progressive Web App
The main features are:

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

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

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

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

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

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

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

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

**Linkable** — Easily shared via URL without complex installations.

**Offline** — Once more it is about putting the user before everything, avoiding the usual error message in case of weak or no connection. The PWA are based on two particularities: first of all the ‘skeleton’ of the app, which recalls the page structure, even if its contents do not respond and its elements include the header, the page layout, as well as an illustration that signals that the page is loading.

Weaknesses:

- IOS support from version 11.3 onwards
- Greater use of the device battery
- Not all devices support the full range of PWA features (same speech for iOS and Android operating systems)
- It is not possible to establish a strong re-engagement for iOS users (URL scheme, standard web notifications)
- Support for offline execution is however limited
- Lack of presence on the stores (there is no possibility to acquire traffic from that channel)
- There is no “body” of control (like the stores) and an approval process
- Limited access to some hardware components of the devices
- Little flexibility regarding “special” content for users (eg loyalty programs, loyalty, etc.)

**Project Title:** **Roll No. 56**

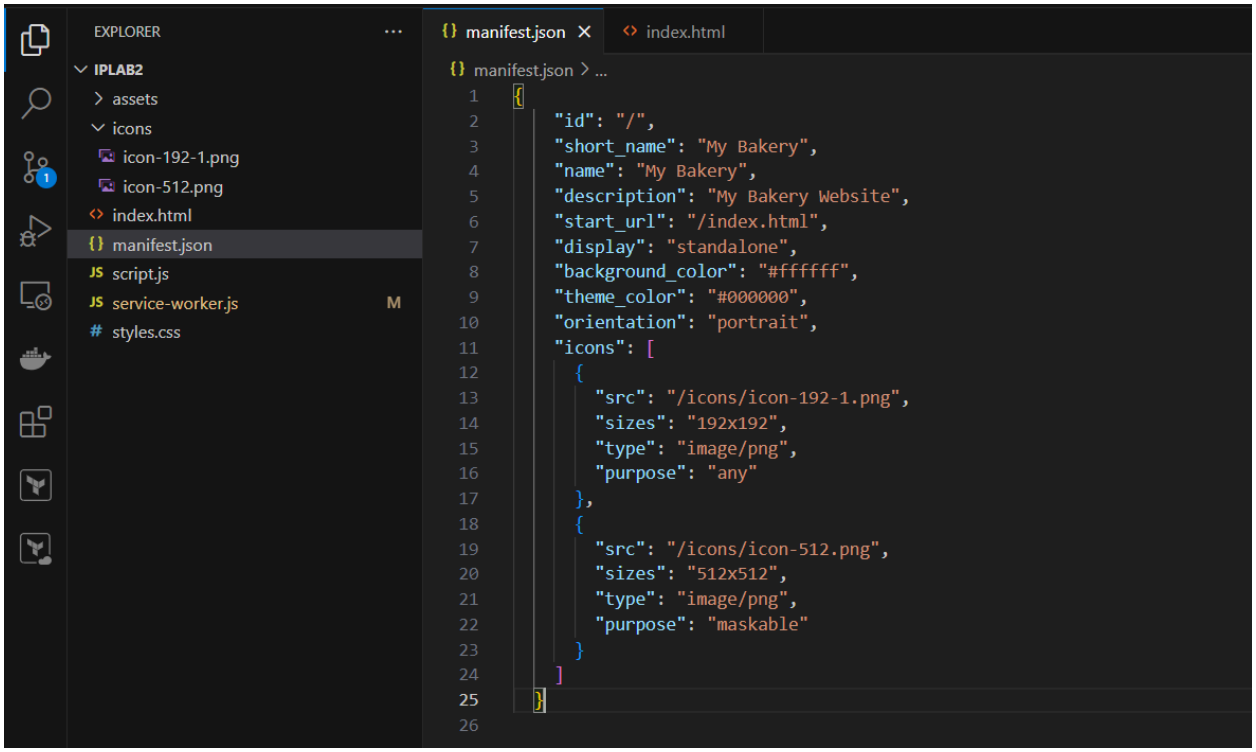
**Code:**

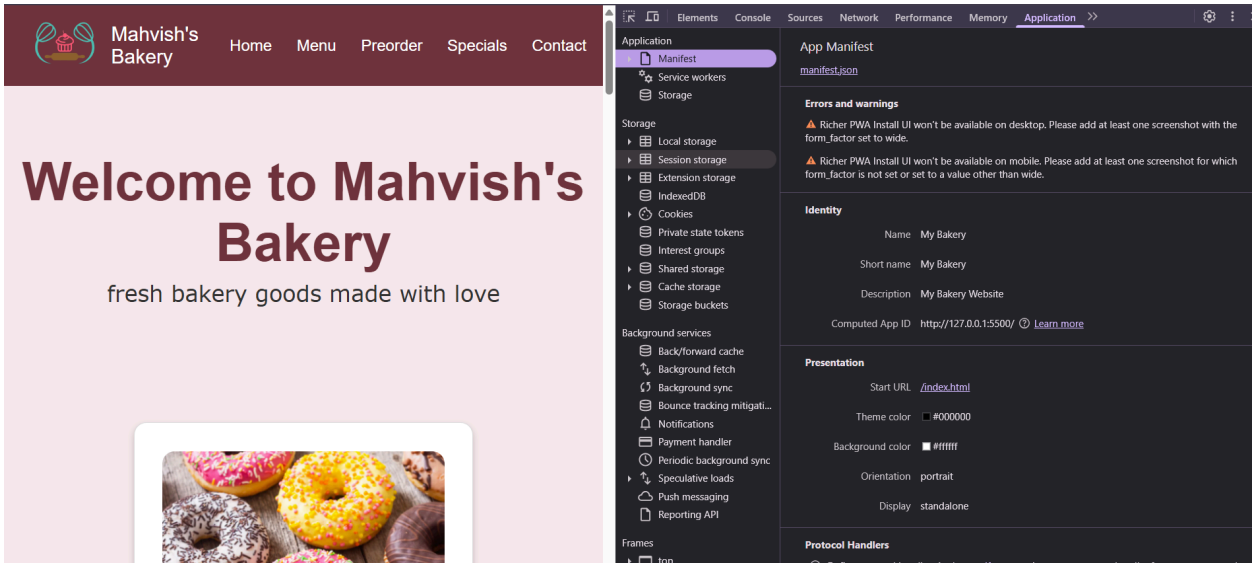
manifest.json:

```
{
    "id": "/",
    "short_name": "My Bakery",
    "name": "My Bakery",
    "description": "My Bakery Website",
    "start_url": "/index.html",
    "display": "standalone",
    "background_color": "#ffffff",
    "theme_color": "#000000",
    "orientation": "portrait",
    "icons": [
      {
        "src": "/icons/icon-192-1.png",
        "sizes": "192x192",
        "type": "image/png",
        "purpose": "any"
      },
      {
        "src": "/icons/icon-512.png",
        "sizes": "512x512",
        "type": "image/png",
        "purpose": "maskable"
      }
    ]
  }
```

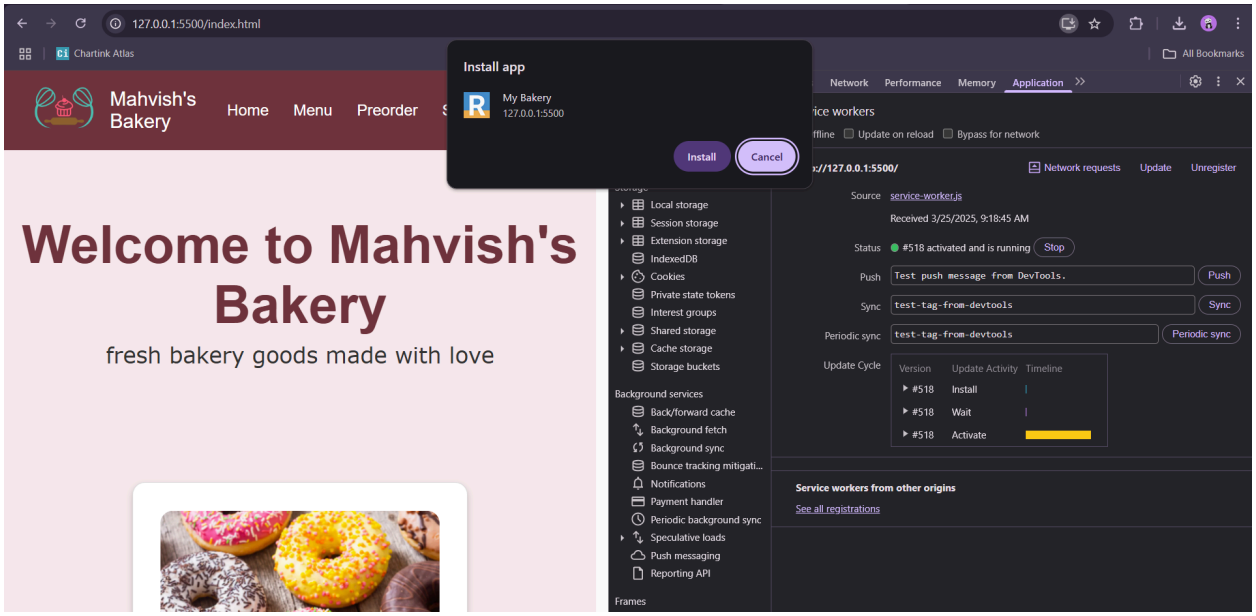
Add the link tag to link to the manifest.json file

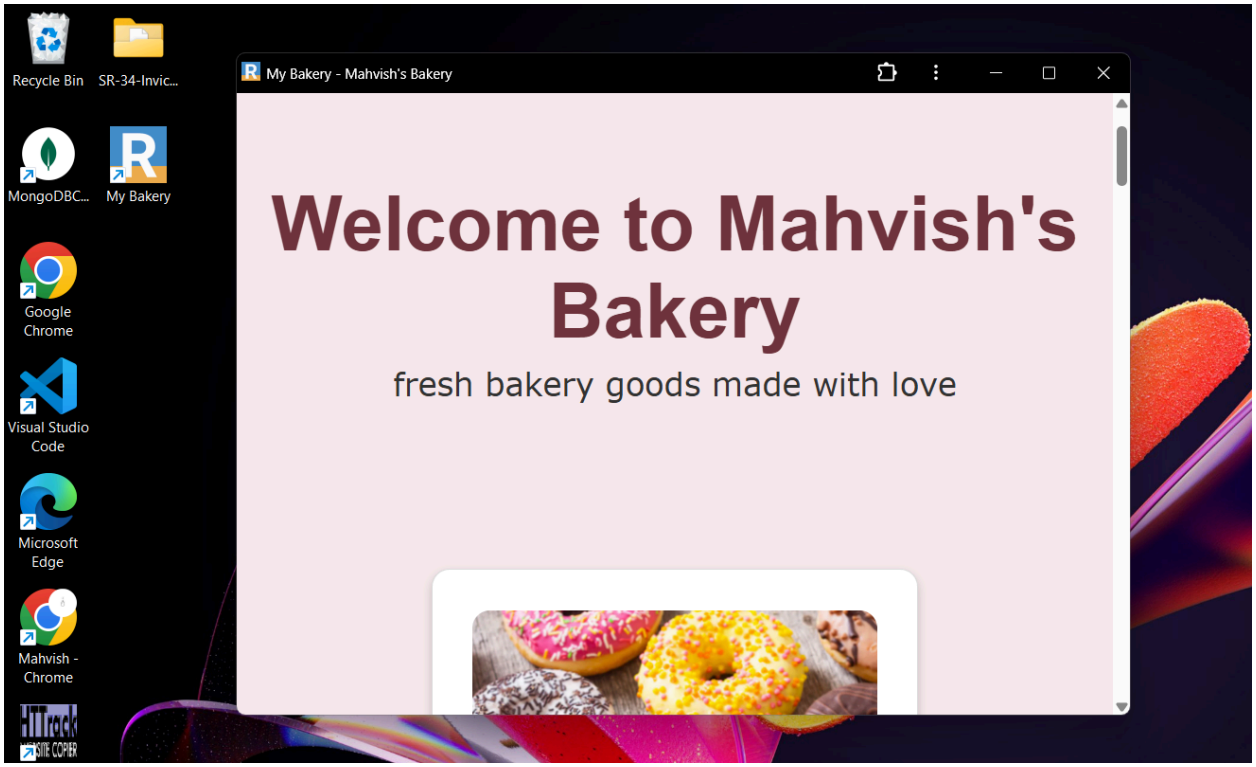
**Project Title:** **Roll No. 56**



**Project Title:** **Roll No. 56**




**Conclusion:**

Hence, we learnt how to write a metadata of our website PWA in a Web App Manifest File to enable add to homescreen feature.

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 08 |
| Experiment Title. | To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA |
| Roll No. | 56 |
| Name | Mahvish Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO5: Design and Develop a responsive User Interface by applying PWA Design techniques |
| Grade: | |

**Project Title:** **Roll No. 56**

# Experiment No. 8

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

**Theory**:

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

Things to note about Service Worker:

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

What can we do with Service Workers?

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

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

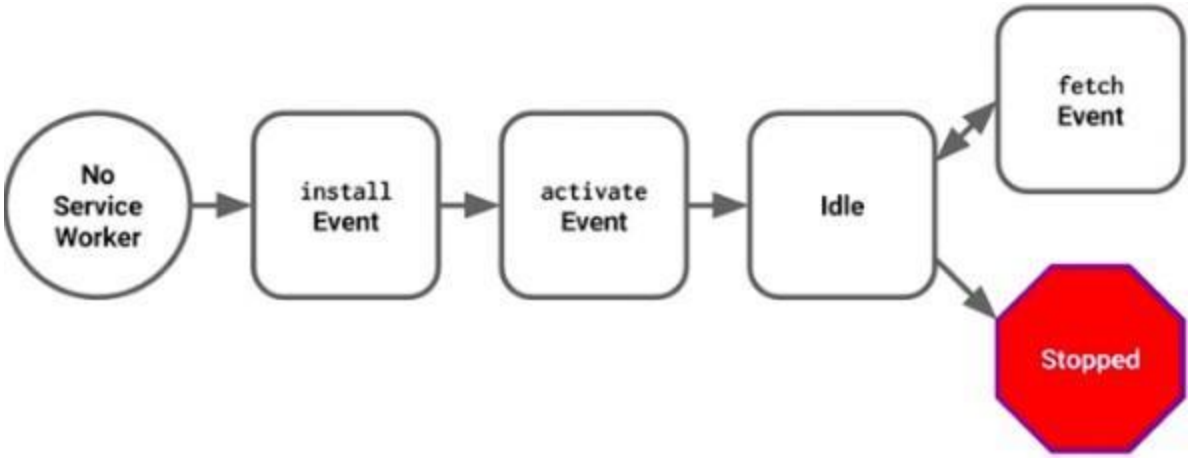
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

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:

```
if ('serviceWorker' in navigator) { navigator.serviceWorker.register('/service-worker.js')
  .then(function(registration) {
    console.log('Registration successful, scope is:', registration.scope);
  })
  .catch(function(error) {
    console.log('Service worker registration failed, error:', error);
  });
}
```

This code starts by checking for browser support by examining **navigator.serviceWorker**. The service worker is then registered with navigator.serviceWorker.register, which returns a promise that resolves when the service worker has been successfully registered. The scope of the service worker is then logged with registration.scope. If the service worker is already installed, navigator.serviceWorker.register returns the registration object of the currently active service worker.

The scope of the service worker determines which files the service worker controls, in other words, from which path the service worker will intercept requests. The default scope is the location of the service worker file, and extends to all directories below. So if service-worker.js is located in the root directory, the service worker will control requests from all files at this domain.

You can also set an arbitrary scope by passing in an additional parameter when registering. For

example: main.js

```
navigator.serviceWorker.register('/service-worker.js', { scope: '/app/'
});
```

In this case we are setting the scope of the service worker to /app/, which means the service worker will control requests from pages like /app/, /app/lower/ and /app/lower/lower, but not from pages like /app or /, which are higher.

If you want the service worker to control higher pages e.g. /app (without the trailing slash) you can indeed change the scope option, but you'll also need to set the Service-Worker-Allowed HTTP Header in your server config for the request serving the service worker script.

```
navigator.serviceWorker.register('/app/service-worker.js', { scope: '/app'
});
```

### Installation

Once the browser registers a service worker, installation can be attempted. This occurs if the service worker is considered to be new by the browser, either because the site currently doesn't have a registered service worker, or because there is a byte difference between the new service worker and the previously installed one.

A service worker installation triggers an install event in the installing service worker. We can include an install event listener in the service worker to perform some task when the service worker installs. For instance, during the install, service workers can precache parts of a web app so that it loads instantly the next time a user opens it (see caching the application shell). So, after that first load, you're going to benefit from instant repeat loads and your time to interactivity is going to be even better in those cases. An example of an installation event listener looks like this:

service-worker.js

```
self.addEventListener('install', function(event) {
   // Perform some task
});
```

### Activation

Once a service worker has successfully installed, it transitions into the activation stage. If there are any open pages controlled by the previous service worker, the new service worker enters a waiting state. The new service worker only activates when there are no longer any pages loaded that are still using the old service worker. This ensures that only one version of the service worker is running at any given time.

When the new service worker activates, an activate event is triggered in the activating service worker. This event listener is a good place to clean up outdated caches (see the Offline Cookbook for an example).

service-worker.js

```
self.addEventListener('activate', function(event) {
  // Perform some task
});
```

Once activated, the service worker controls all pages that load within its scope, and starts listening for events from those pages. However, pages in your app that were loaded before the service worker activation will not be under service worker control. The new service worker will only take over when you close and reopen your app, or if the service worker calls **clients.claim()**. Until then, requests from this page will not be intercepted by the new service worker. This is intentional as a way to ensure consistency in your site.

**Project Title:** **Roll No. 56**

**Code:**

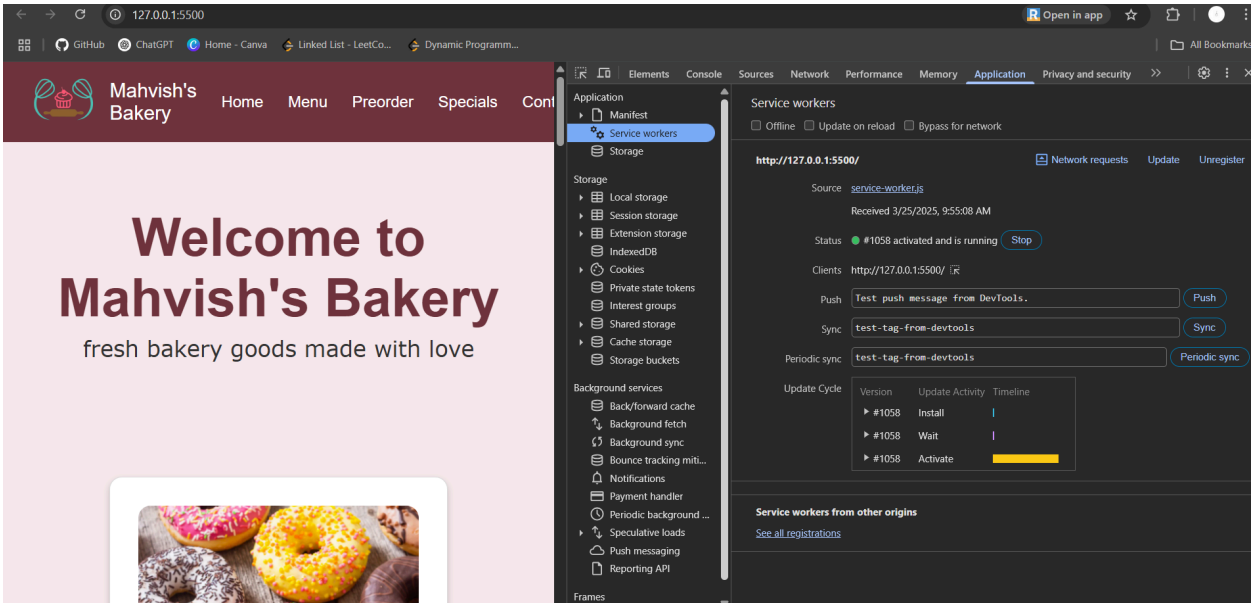
```
<script>
      if ('serviceWorker' in navigator) {
        window.addEventListener('load', () => {
          navigator.serviceWorker.register('/service-worker.js')
            .then((registration) => {
              console.log('Service Worker registered with scope: ', registration.scope);
            })
            .catch((error) => {
              console.log('Service Worker registration failed: ', error);
            });
        });
      }
    </script>
```

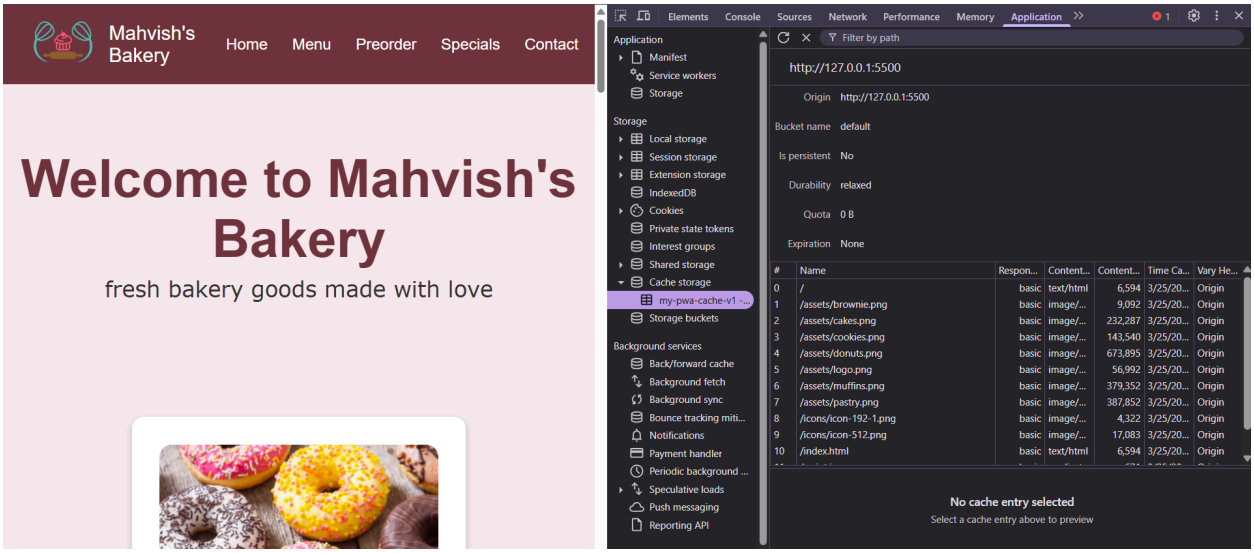
**service-worker.js**

```
const CACHE_NAME = 'my-pwa-cache-v1';
const urlsToCache = [
  '/',
  '/index.html',
  '/styles.css',
  '/icons/icon-192-1.png',
  '/icons/icon-512.png',
  '/assets/brownie.png',
  '/assets/cakes.png',
  '/assets/cookies.png',
  '/assets/donuts.png',
  '/assets/logo.png',
  '/assets/muffins.png',
  '/assets/pastry.png',
];

// Install service worker
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
      console.log('Opened cache');
      return cache.addAll(urlsToCache);
```

```
    })
  );
});

// Activate service worker
self.addEventListener('activate', (event) => {
  const cacheWhitelist = [CACHE_NAME];
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cacheName) => {
          if (!cacheWhitelist.includes(cacheName)) {
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});

// Fetch content from the cache or network
self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request).then((cachedResponse) => {
      return cachedResponse || fetch(event.request);
    })
  );
});
```

**Project Title:** **Roll No. 56**



# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 09 |
| Experiment Title. | To implement Service worker events like fetch, sync and push for E-commerce PWA |
| Roll No. | 56 |
| Name | Mahvish Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO5: Design and Develop a responsive User Interface by applying PWA Design techniques |
| Grade: | |

**Project Title:** **Roll No. 56**

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

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

Things to note about Service Worker:

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

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

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

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

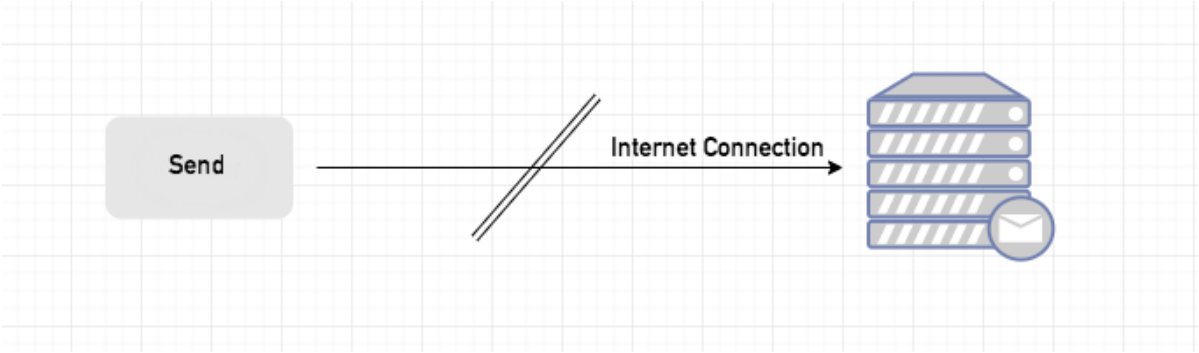
information messages to the page.

**Sync Event**

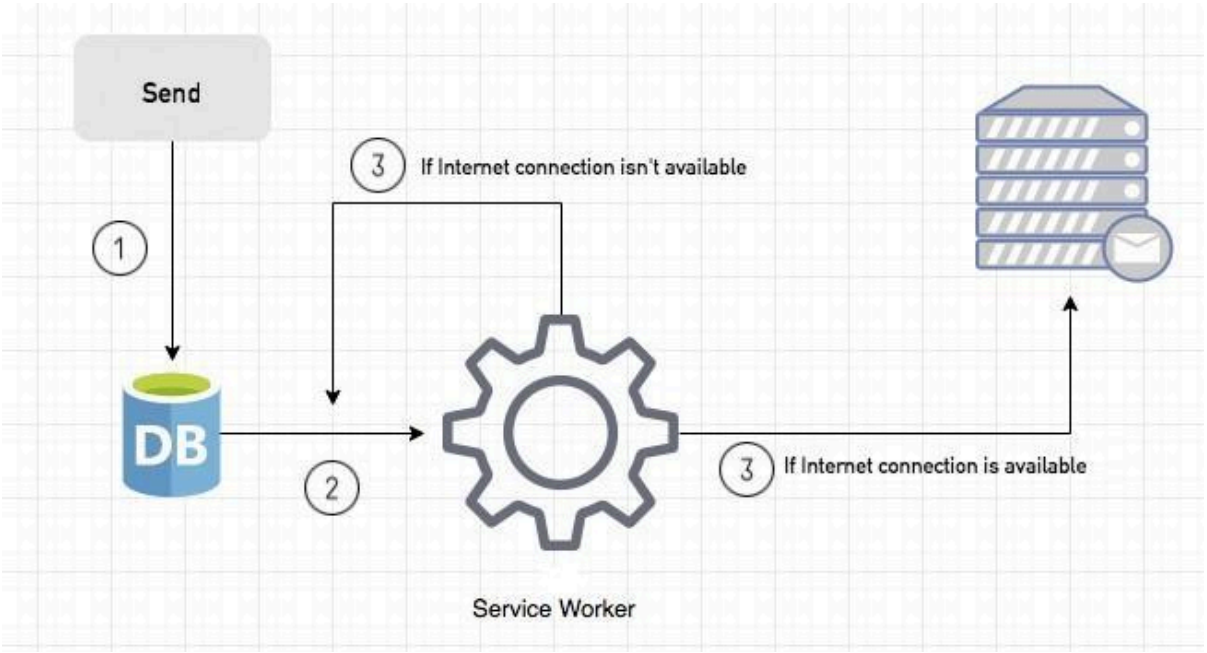
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are writing e-mail content and we didn't realize it. When completing the writing, we click the send button.

Here is a job for the Background Sync.

The following view shows the classical process of sending email to us. If the Internet Connection is broken, we can't send any content to Mail Server.




Here, you can create any scenario for yourself. A sample is in the following for this case.



1. When we click the “send” button, email content will be saved to IndexedDB.
2. Background Sync registration.
3. **If the Internet connection is available**, all email content will be read and sent to Mail Server. **If the Internet connection is unavailable**, the service worker waits until the connection is available even though the window is closed. When it is available, email content will be sent to Mail Server.

   You can see the working process within the following code block.

Event Listener for Background Sync Registration

```
document.querySelector("button").addEventListener("click", async () => {
    var swRegistration = await navigator.serviceWorker.register("sw.js");
    swRegistration.sync.register("helloSync").then(function () {
        console.log("helloSync success [main.js]");
    });
});
```

Event Listener for sw.js

```
self.addEventListener('sync', event => {
    if (event.tag == 'helloSync') {
        console.log("helloSync [sw.js]");
    }
});
```

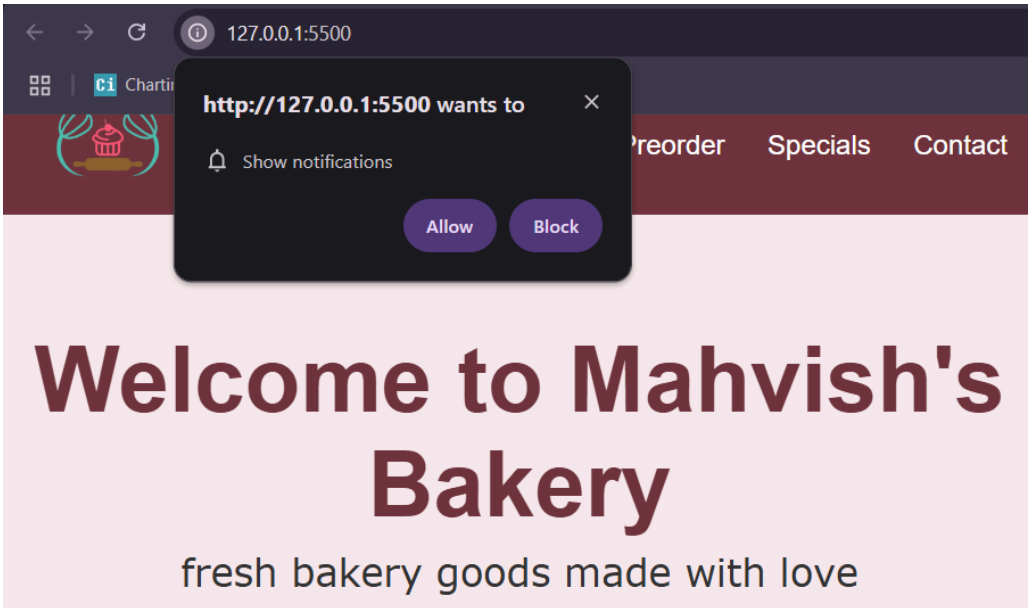
**Push Event**

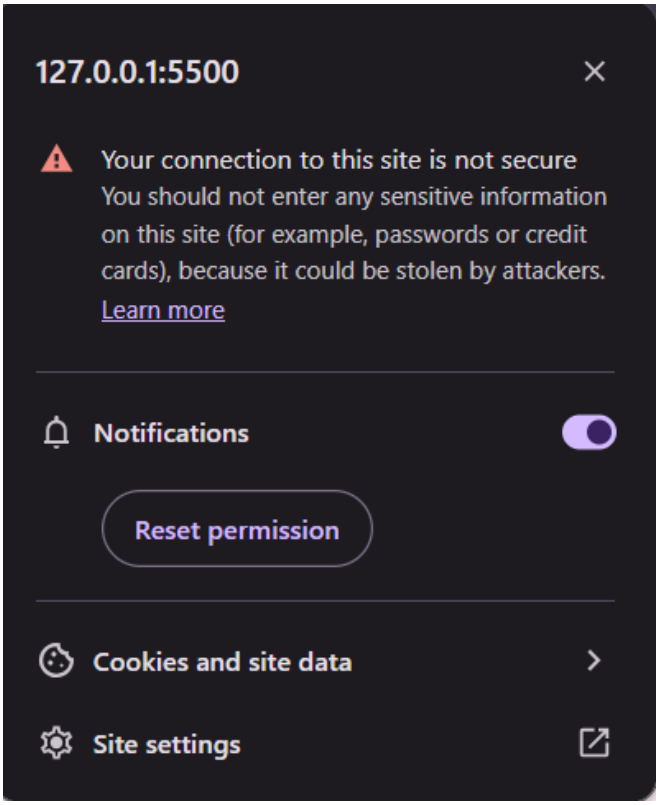
This is the event that handles push notifications that are received from the server. You can apply any method with received data.

We can check in the following example.
“Notification.requestPermission();” is the necessary line to show notification to the user. If you don’t want to show any notification, you don’t need this line.

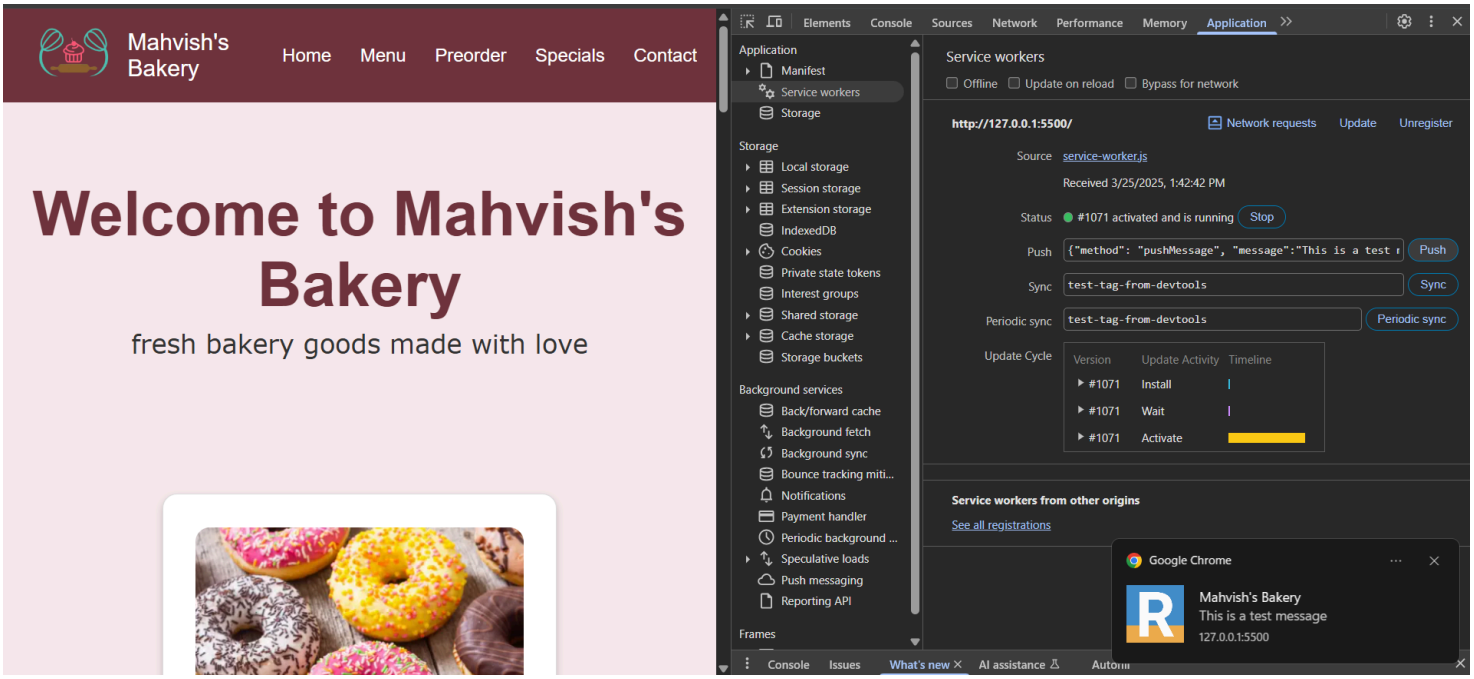
In the following code block is in sw.js file. You can handle push notifications with this event. In this example, I kept it simple. We send an object that has “method” and “message” properties. If the method value is “pushMessage”, we open the information notification with the “message” proper

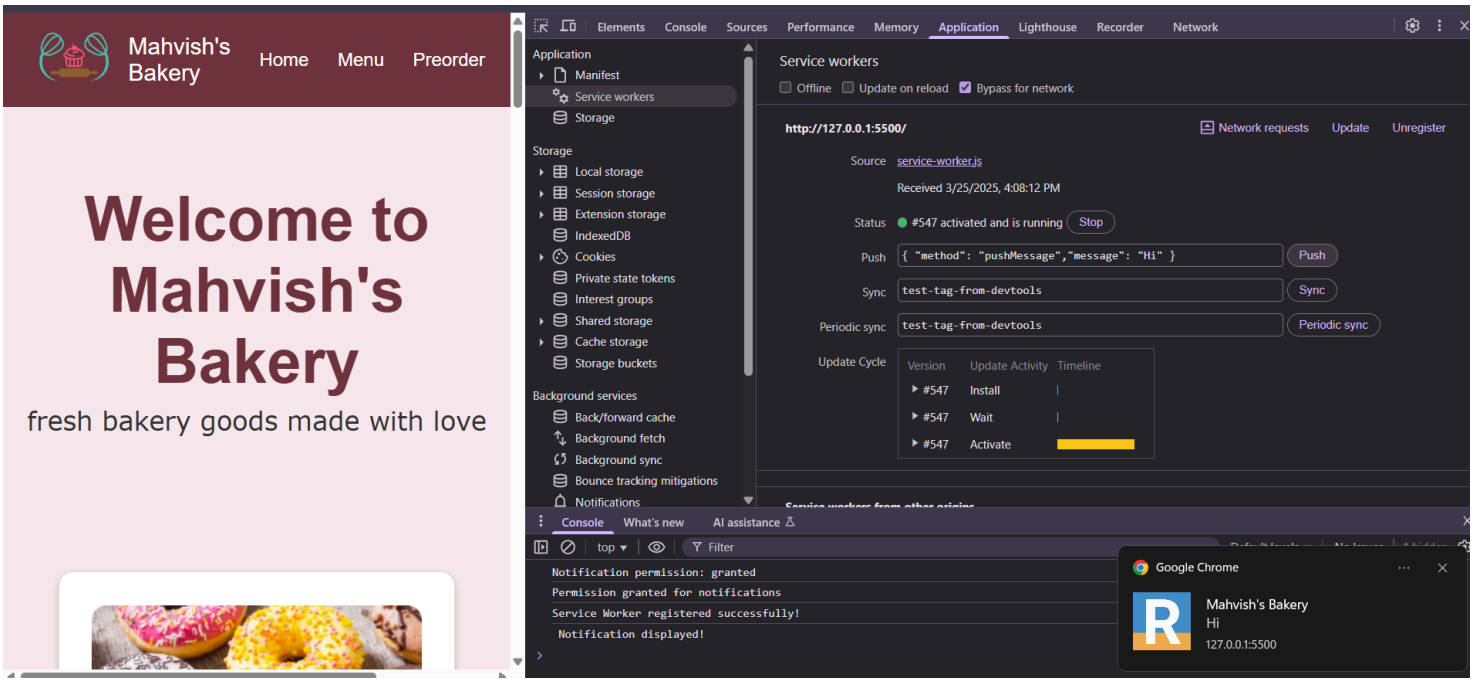
**Project Title:** **Roll No. 56**



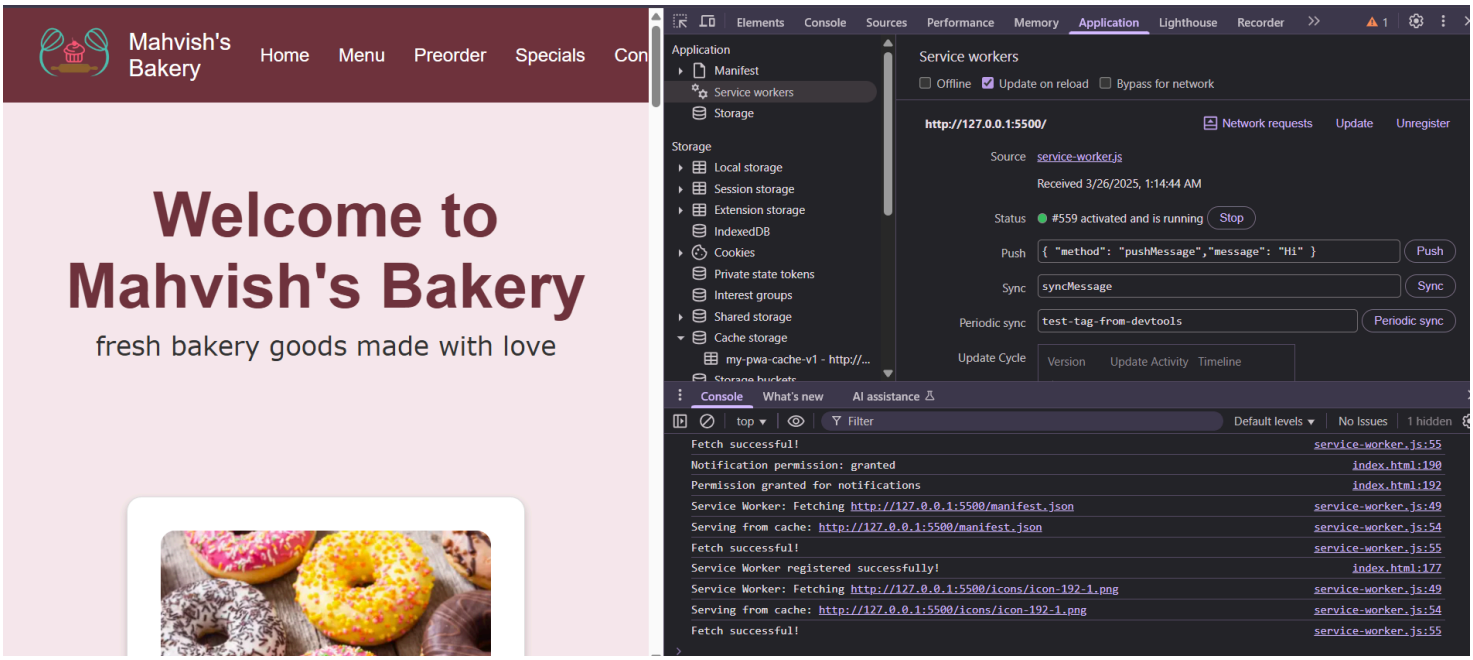



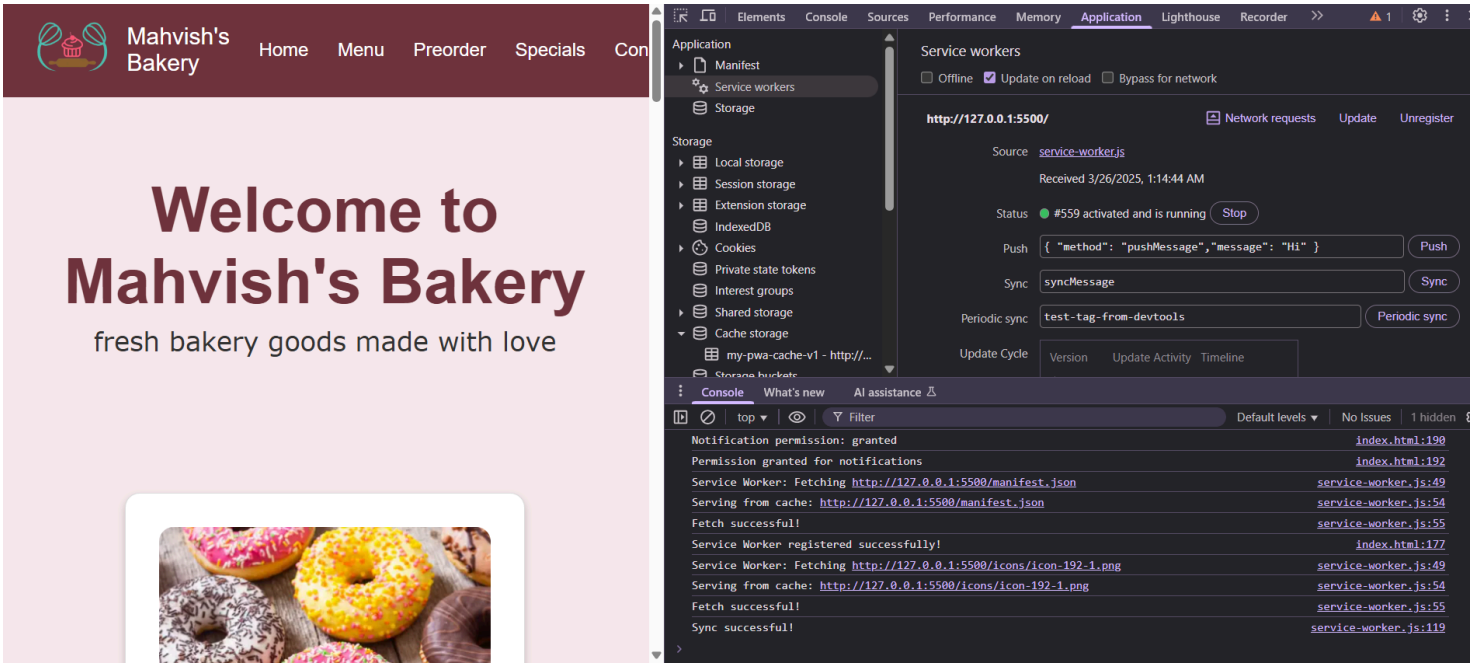
**Project Title:** **Roll No. 56**



**Project Title:** **Roll No. 56**



# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 10 |
| Experiment Title. | To study and implement deployment of Ecommerce PWA to GitHub Pages. |
| Roll No. | 56 |
| Name | Mahvish Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO5: Design and Develop a responsive User Interface by applying PWA Design techniques |
| Grade: | |

# 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

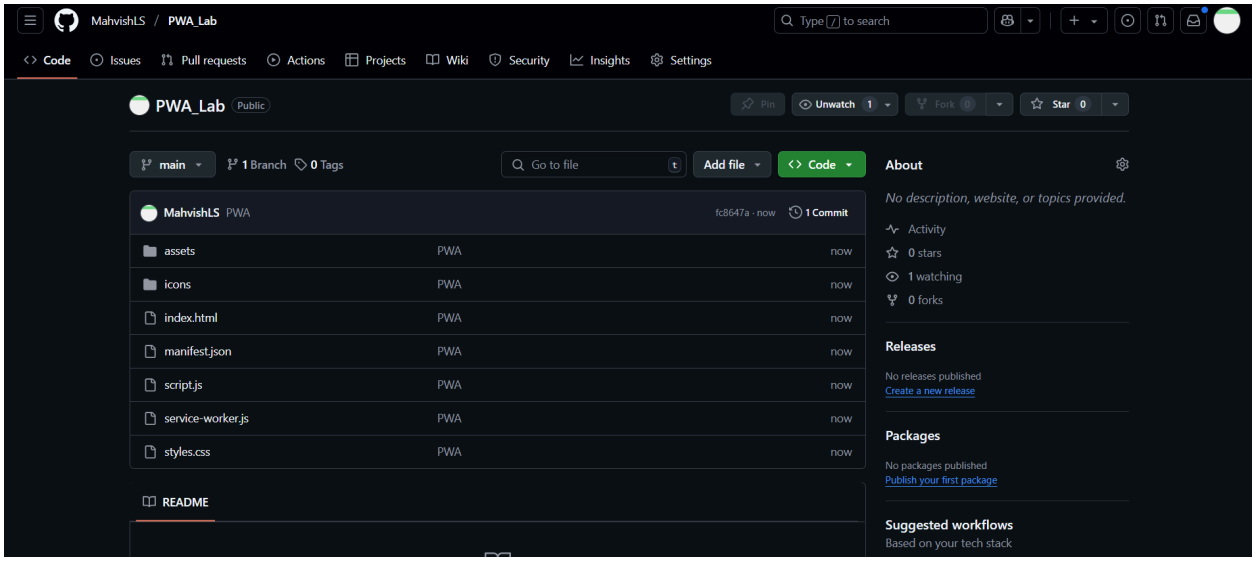
1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5.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

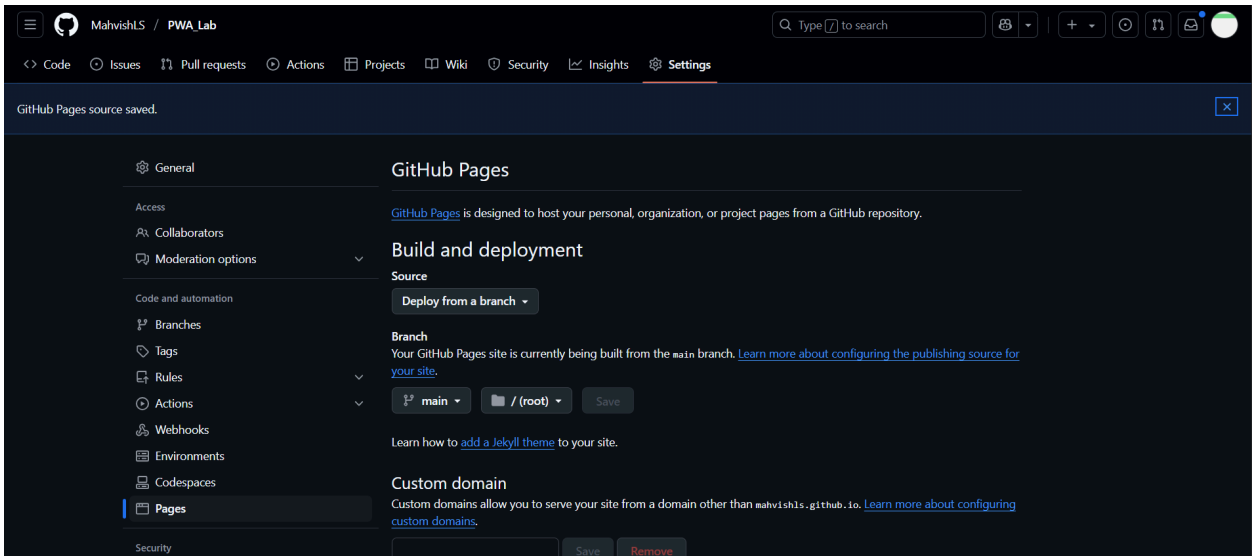
Cons

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.

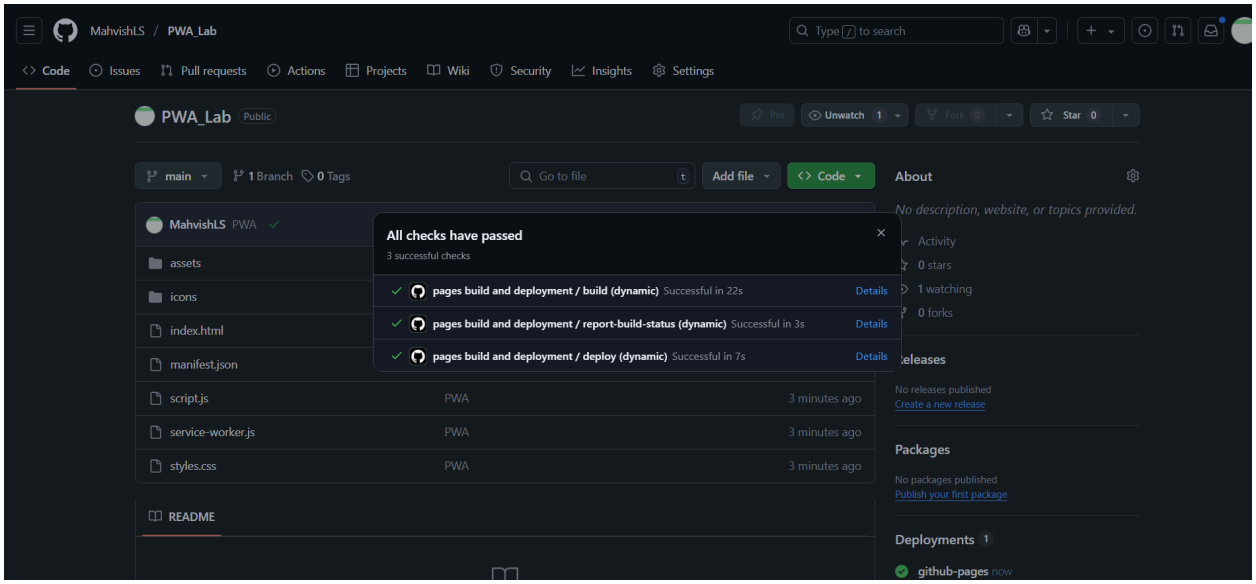
## Link to GitHub repository: https://github.com/MahvishLS/PWA_Lab

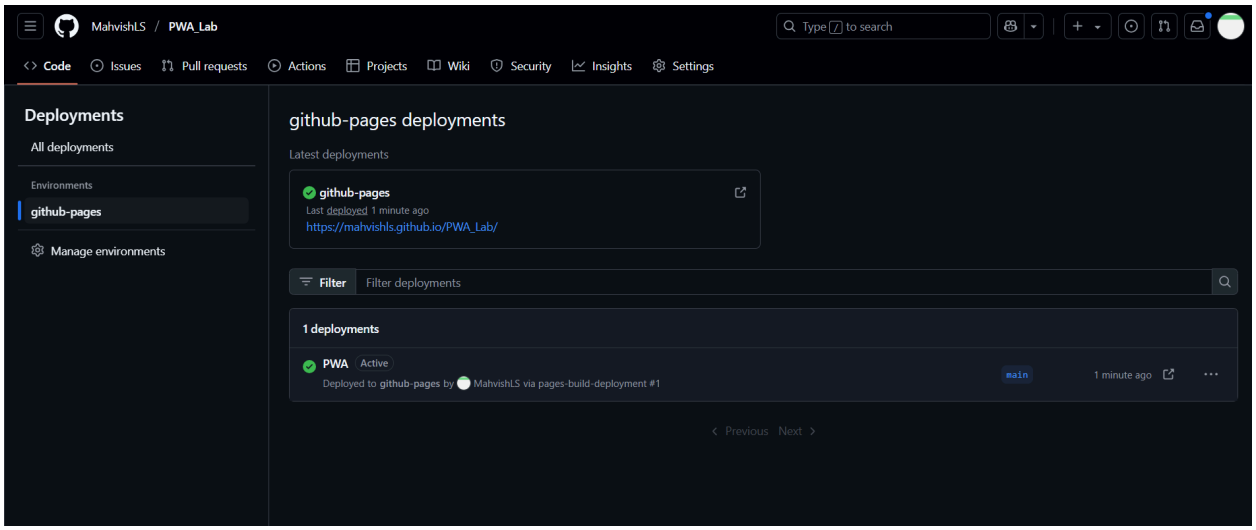
## Github Screenshot:



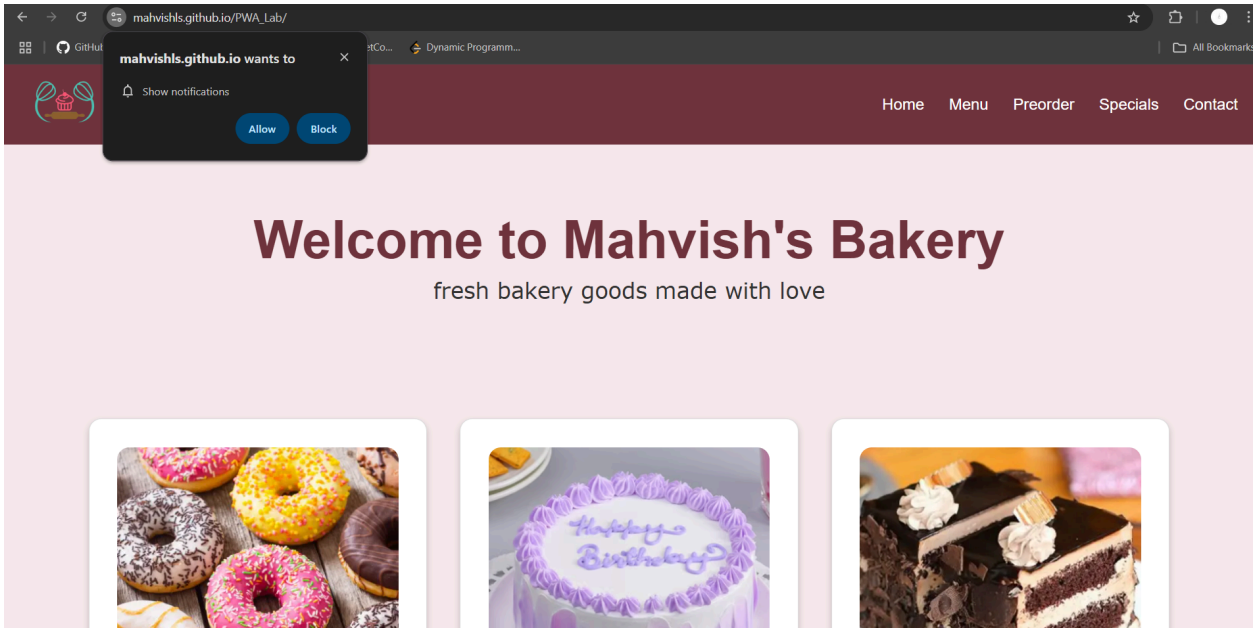



**Project Title:** **Roll No. 56**



**Project Title:** **Roll No. 56**




**Deployed Link**: https://mahvishls.github.io/PWA_Lab/

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 11 |
| Experiment Title. | To use google Lighthouse PWA Analysis Tool to test the PWA functioning. |
| Roll No. | 56 |
| Name | Mahvish Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO6: Develop and Analyze PWA Features and deploy it over app hosting solution |
| Grade: | |

# Experiment No. 11

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

**Theory** :

**Google Lighthouse :**

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

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

**Key Features and Audit Metrics**

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

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

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

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

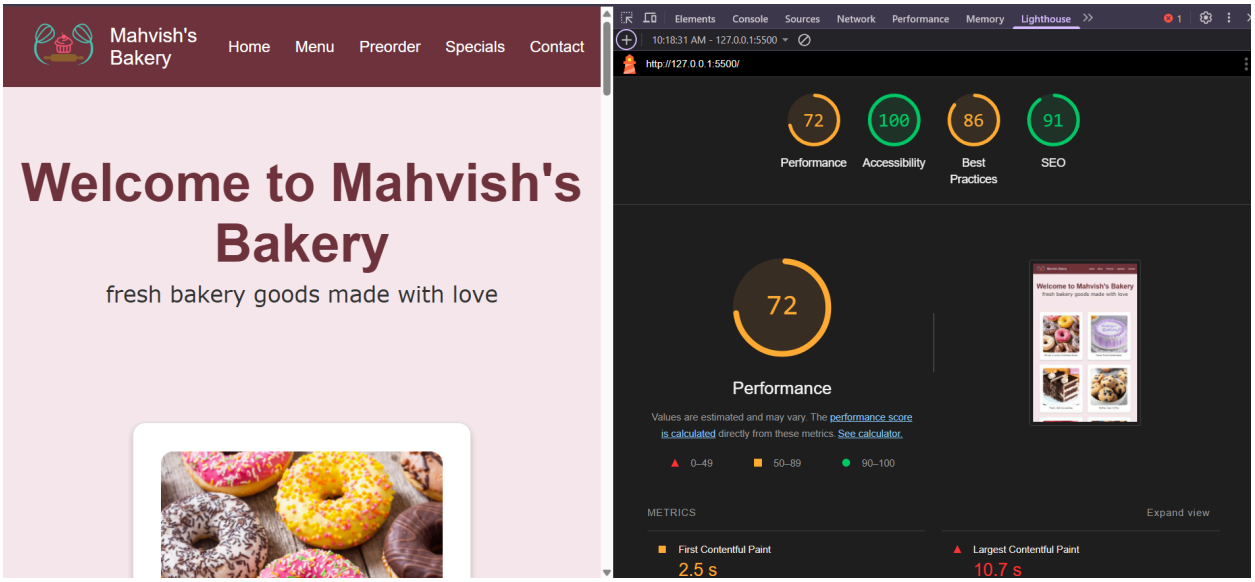
   Avoiding the use of deprecated code elements like tags, directives, libraries, etc. Password input with paste-into disabled

   Geo-Location and cookie usage alerts on load, etc.

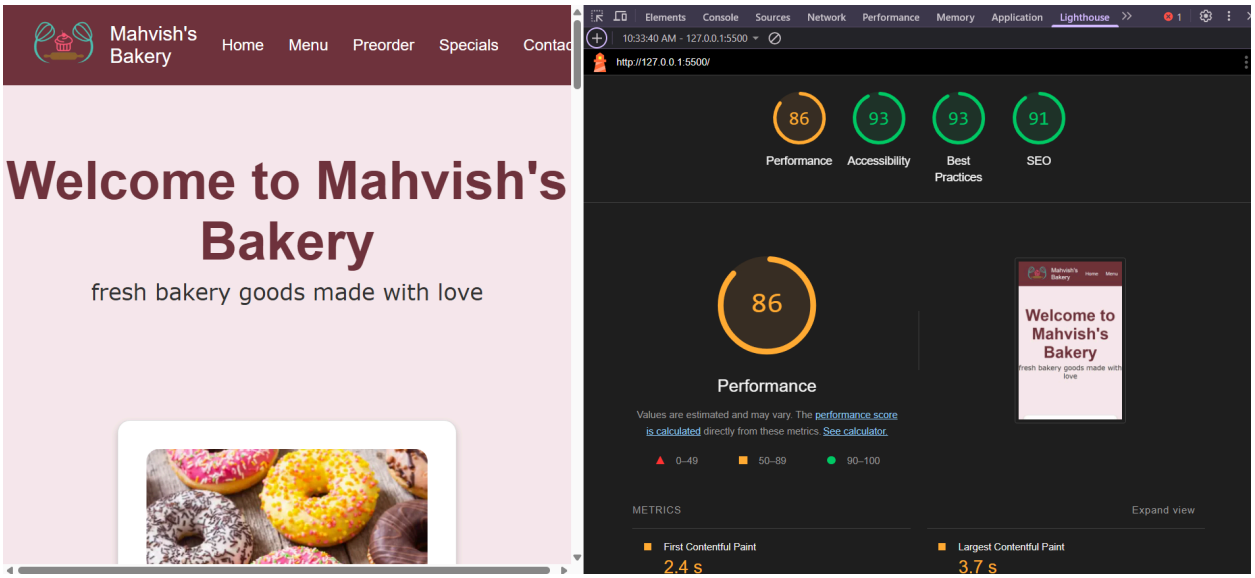
**Screenshots:**

<u>Mobile: initial</u>

**Project Title:** **Roll No. 56**

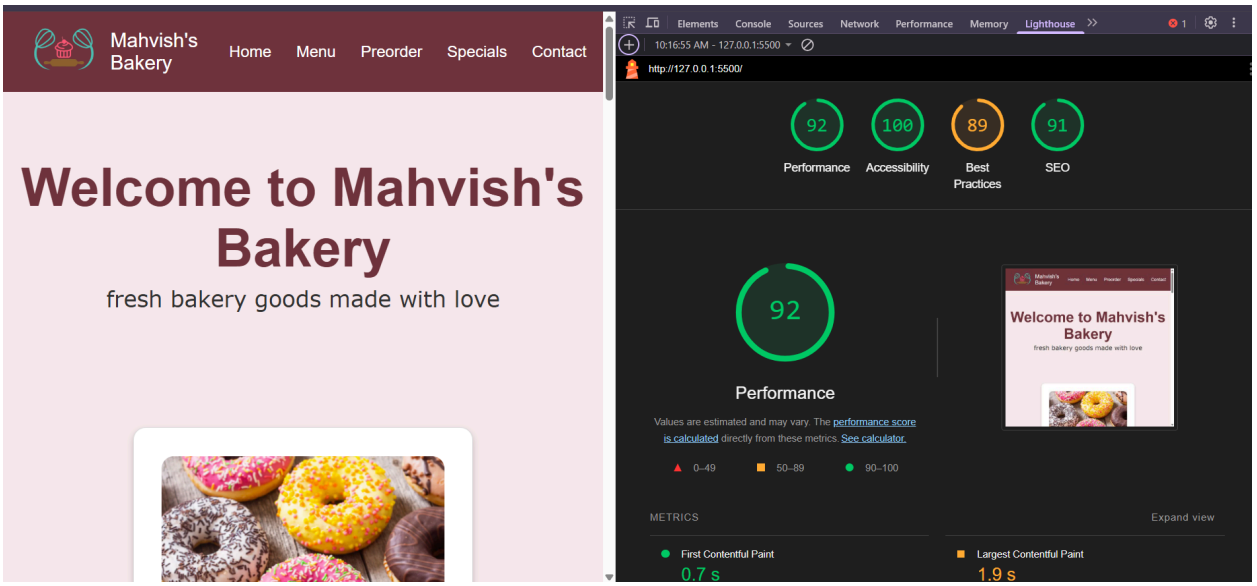



Mobile: After making recommended changes




Desktop

**Project Title:** **Roll No. 56**




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