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 Jai Talreja of **D15A** semester **VI,** has successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2024-2025.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Project Title: **Roll No.**

MAD & PWA Lab File **59**

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

Year/Sem/Class **:** TE/Sem VI/D15A **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.**

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.**

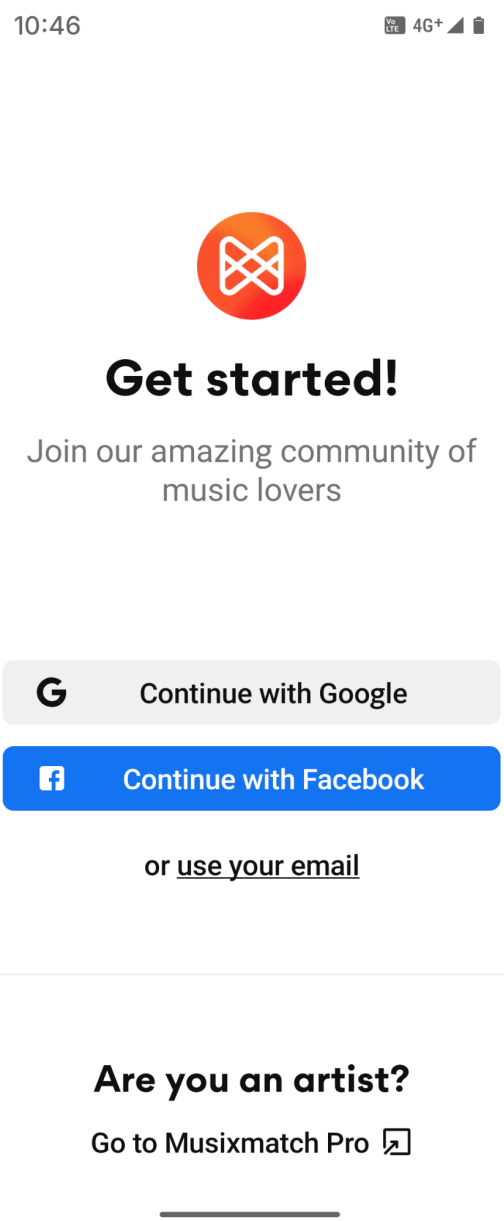
Index

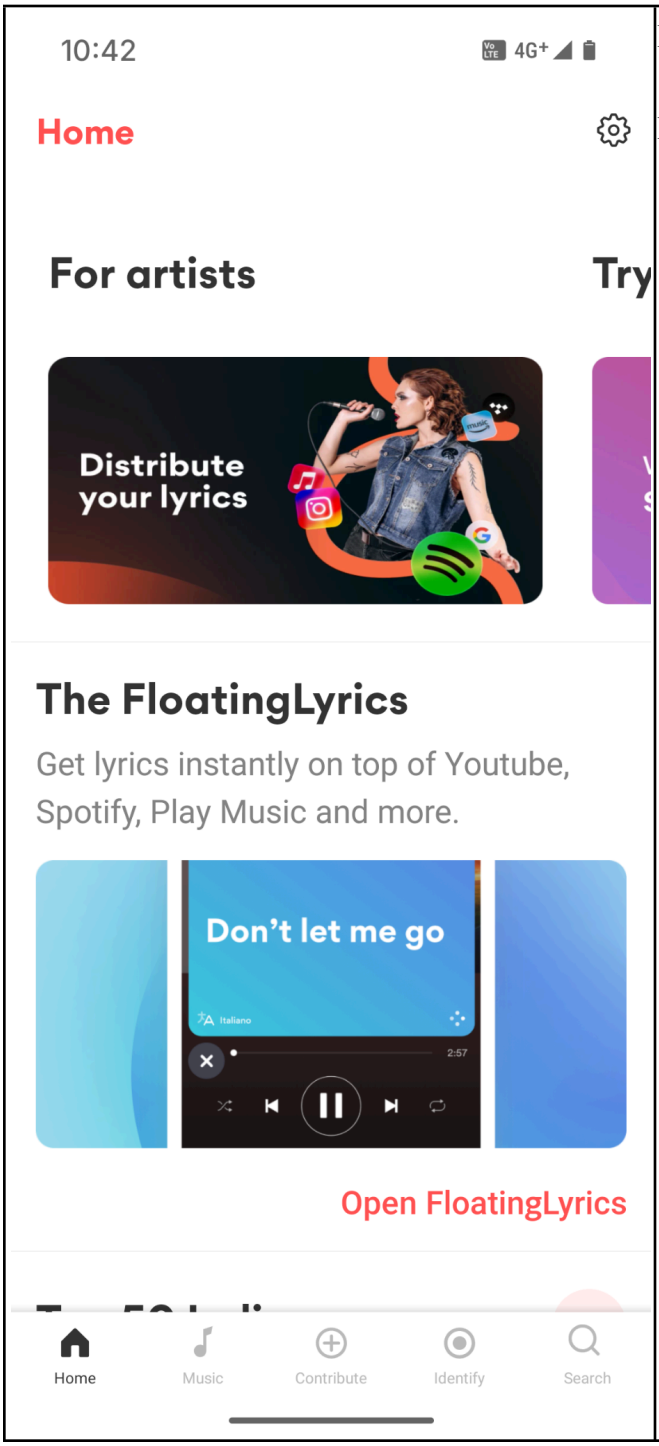
Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1			
2.	To design Flutter UI by including common widgets.	LO2			
3.	To include icons, images, fonts in Flutter app	LO2			
4.	To create an interactive Form using form widget	LO2			
5.	To apply navigation, routing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

MPL & PWA LAB PREREQUISITE

AIM: - Selecting features for application development, the features should comprise of:

1. Common Widgets
2. Should include icons, images, charts etc.
3. Should have an interactive form.
4. Should apply navigation, routing and gestures
5. Should connect with FireBase database

Screen Shot	Features
	**Login Page** • **Purpose**: Enables users to register or log in by Google Account, or optionally by Facebook login or another email ID. • **Features**: 1. **"Continue with Google" Button** 2. **Option to use other email** • **Firebase Integration and Google Sign In** Ensures secure login by storing user details in Firebase Authentication. GoogleSignInClient API can validate user identity based on Google Account credentials. • **Facebook account sign in** The LoginManager in Facebook SDK and handle sign in requests with Facebook account.



HomePage

Banner Carousel:

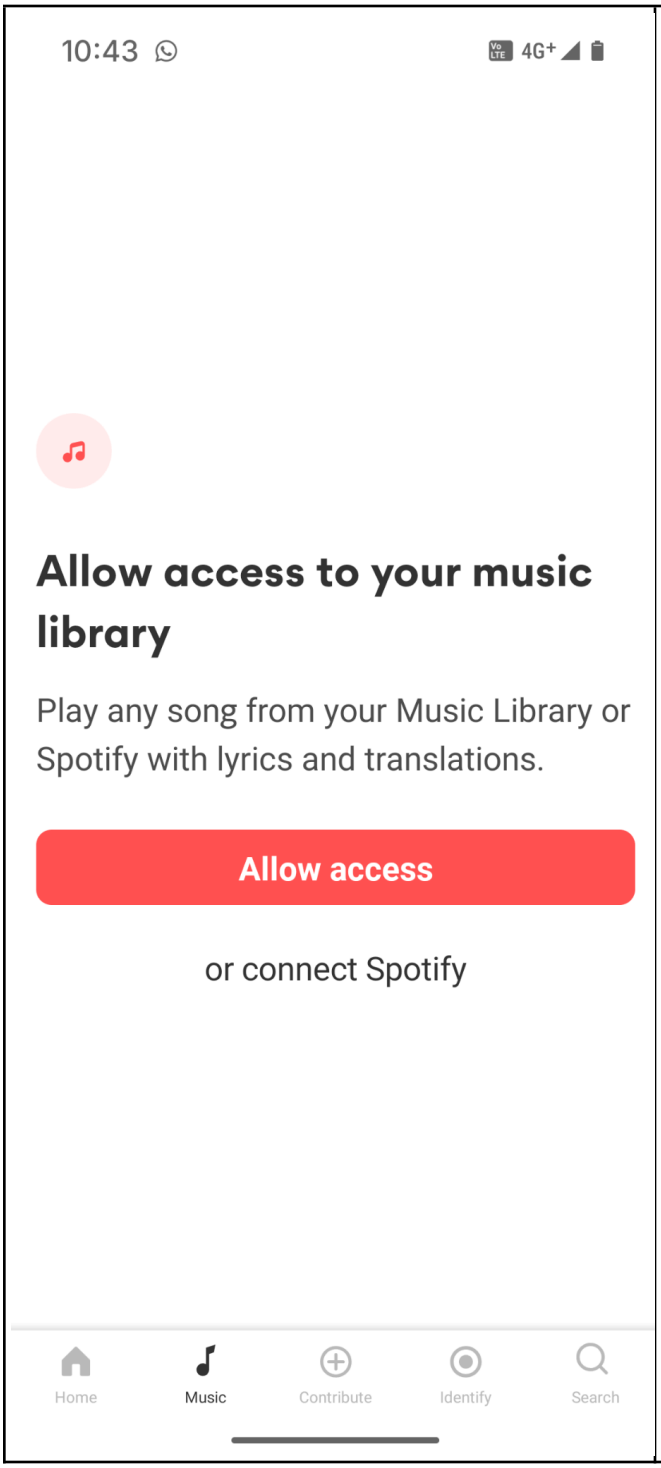
- **Purpose**: Displays the capabilities of the app and available features.
- **Widgets Used**: Carousel widget with images and animations.

Feature Icons:

- **Purpose**: Interactive icons representing categories like For Artists, Floating Lyrics, Translated languages.
- **Widgets Used**: Icon Buttons with labels for clear representation.

Navigation Bar:

- **Purpose**: A bottom navigation bar for quick access to app sections like Home, Music, Contribute, Identify, Search.
- **Widgets Used**: Navigation Widget with routing and gesture handling.

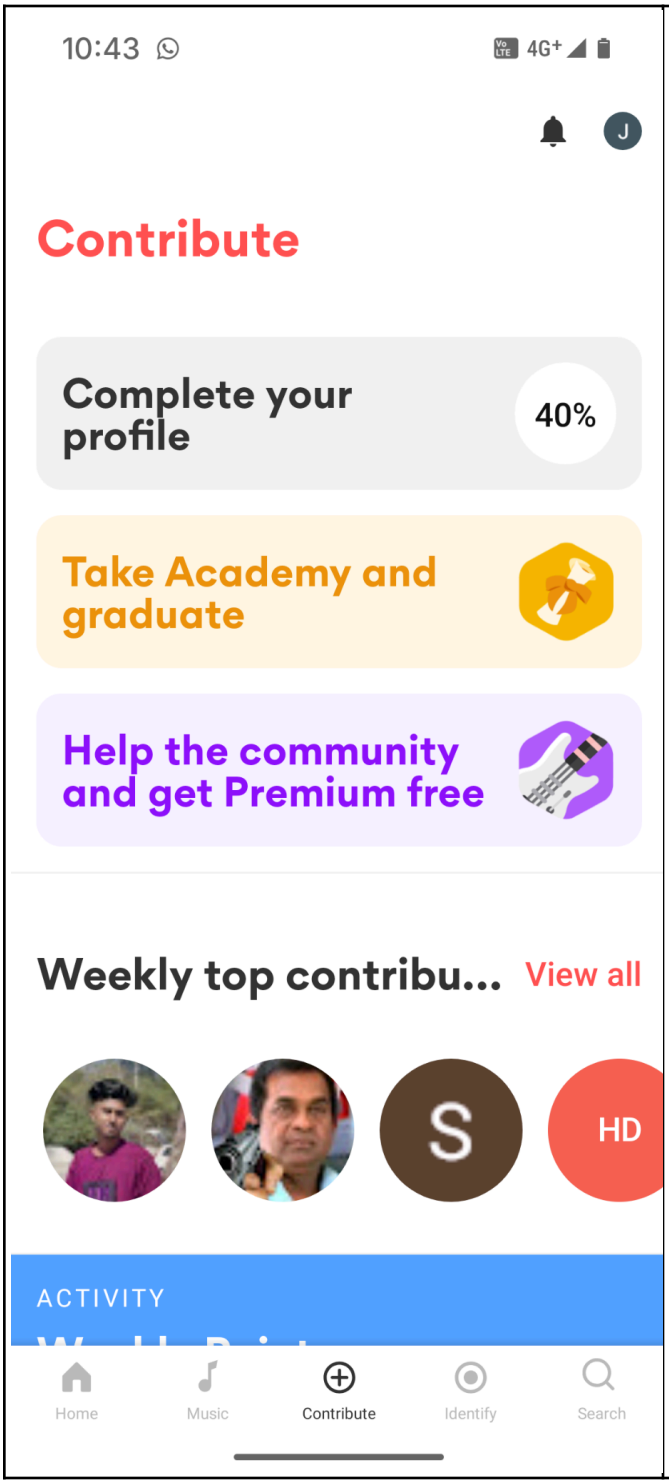


Music Page

- **Purpose :**
  To access locally stored music or optionally, Spotify account music.

- **Requirements :**
  1. Access to local storage,
  2. Spotify Integration using Spotify SDK

- **Features :**
  1. Button to bring local music access permission pop-up,
  2. Button to bring Spotify login window



Contribute Page

- **Purpose :**
  To encourage the user to contribute to the Musixmatch community and to the App's purpose.

- **Features :**
  Widgets to take the user to the profile page and to other pages and one that redirects to a website,
  A scrolling info field to display the profiles of the top contributors.

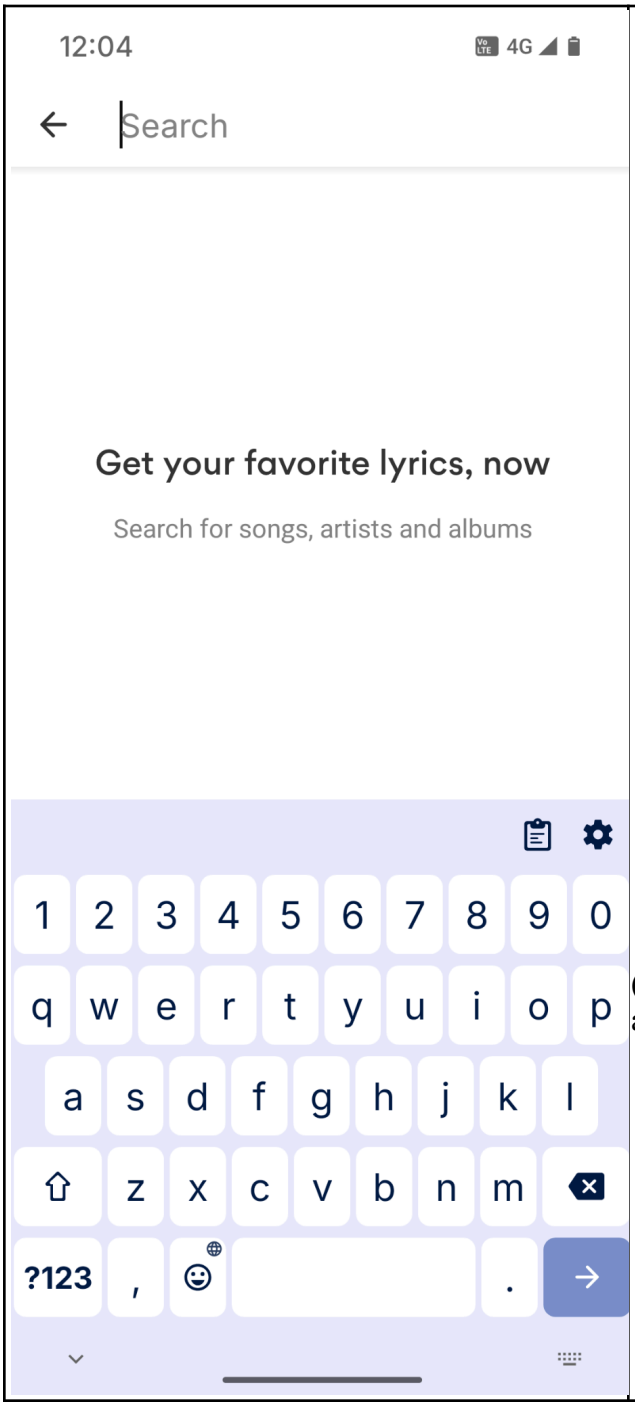
Song Identification Page

- **Purpose :**
  To identify the song being played using the microphone.

- **UI Features :**
  An animated button for the user to start recording audio and to display that audio is being recorded.

- **Functional Features :**
  1. ACR Cloud SDK for the song detection API,
  2. Audio recording permission

- **UI requirements :**
  1. Button,
  2. ImageView,
  3. MaterialButton,
  4. ObjectAnimator,
  5. ValueAnimator,
  6. LottieAnimationView



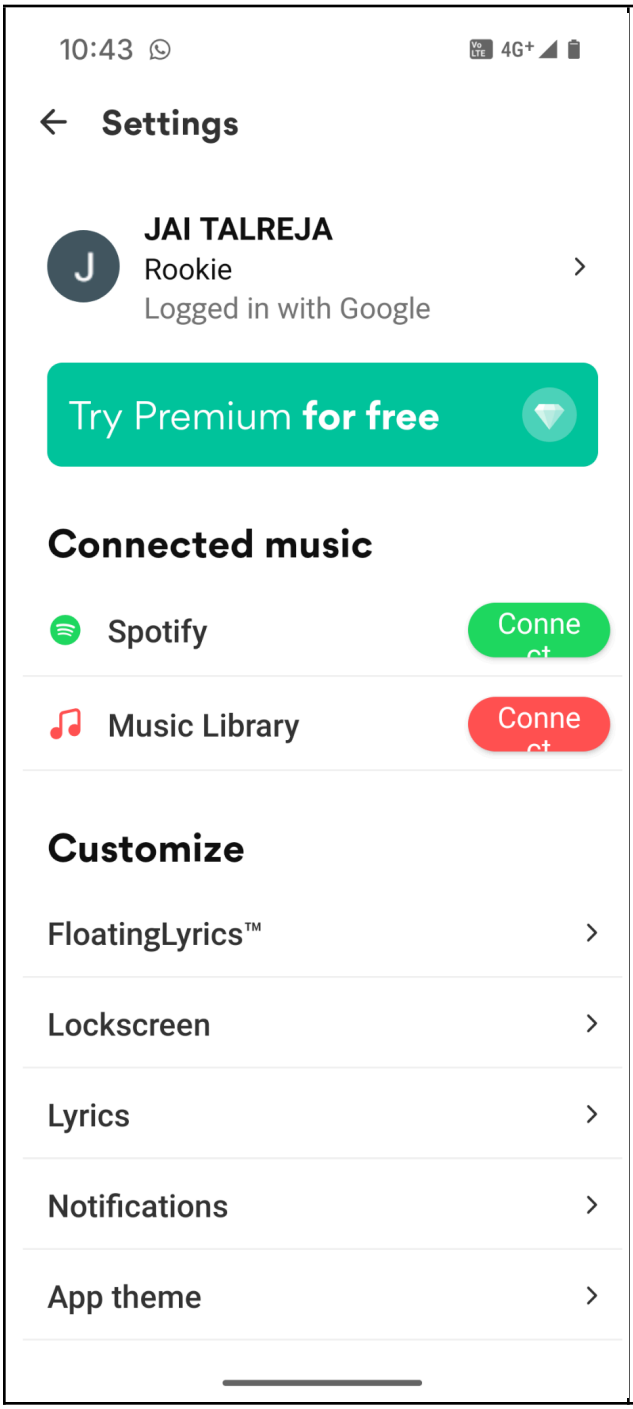
Search Page

- **Purpose :**
  To let the user search for music using the name of the track, of the artist, or the lyrics within the track.

- **Functional Features :**
  1. Pops up the default system keyboard when the page is visited,
  2. Text input field

- **UI Components :**
  1. EditText,
  2. AutoCompleteTextView,
  3. TextWatcher,
  4. Button

(Keyboard not a part of the UI, it is another app)



Settings Page

- **Purpose :**
  Allows the user to Change the logged in account, Customize app functionality, change the app tier (Free or Premium), And connect or disconnect local storage and Spotify Music library.

- **UI Components :**
  1. LinearLayout,
  2. RecyclerView,
  3. MaterialButton,
  4. android:DrawableEnd

- **Functional Feature requirements :**
  1. Firebase Authentication,
  2. Spotify Authentication,
  3. GoogleSignInClient API,
  4. Local Music access permission

Project Title: **Roll No.**

MAD & PWA Lab
Journal

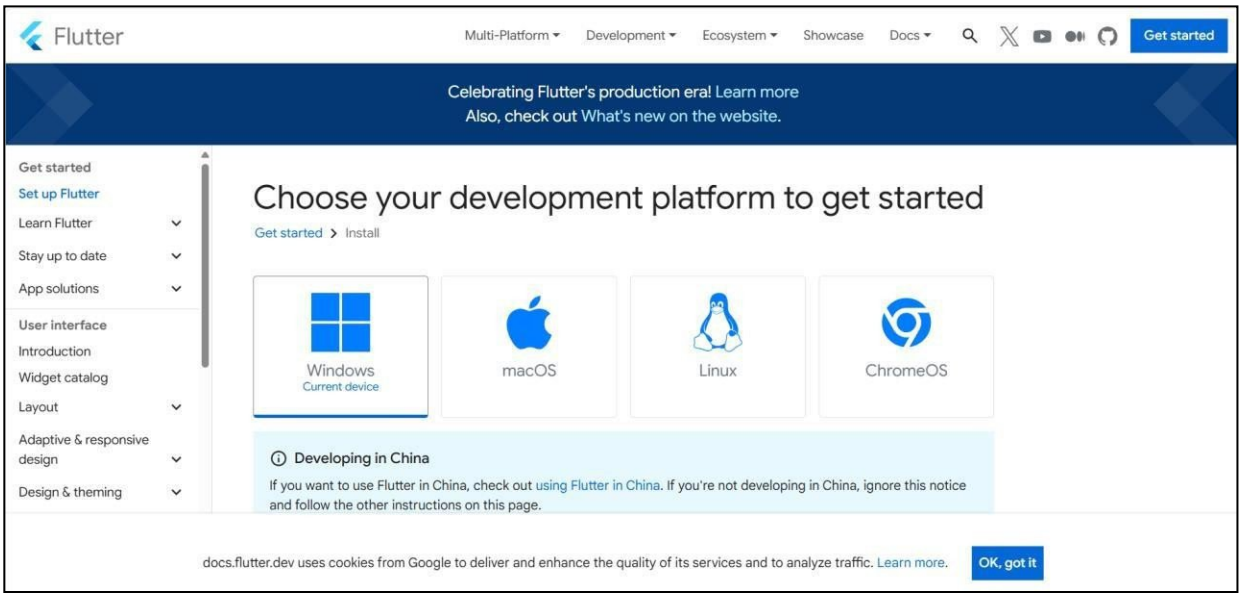
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	59
Name	Jai Talreja
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

EXPERIMENT NO: - 01

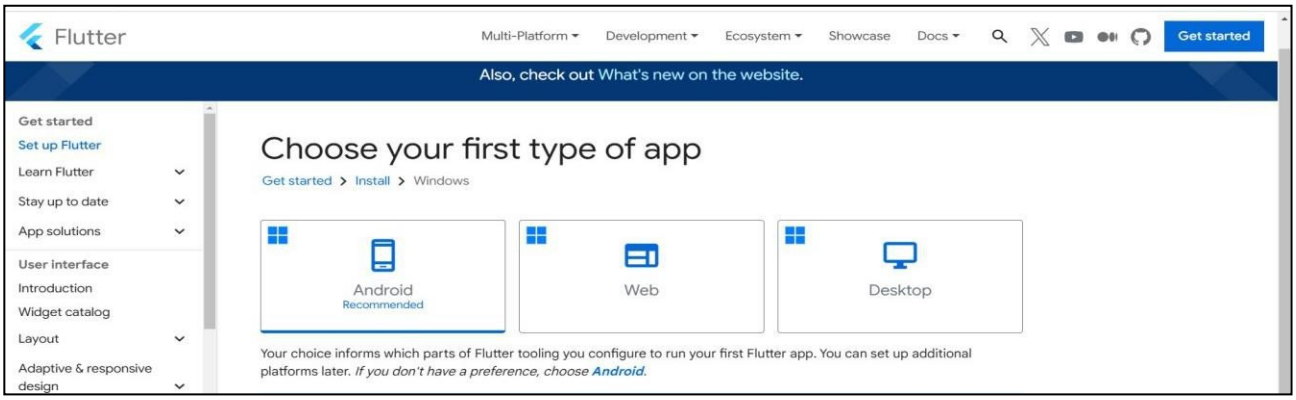
Name:- Jai Talreja **Class:-** D15A **Roll:No: -** 59

AIM: - Installation and Configuration of Flutter Environment.

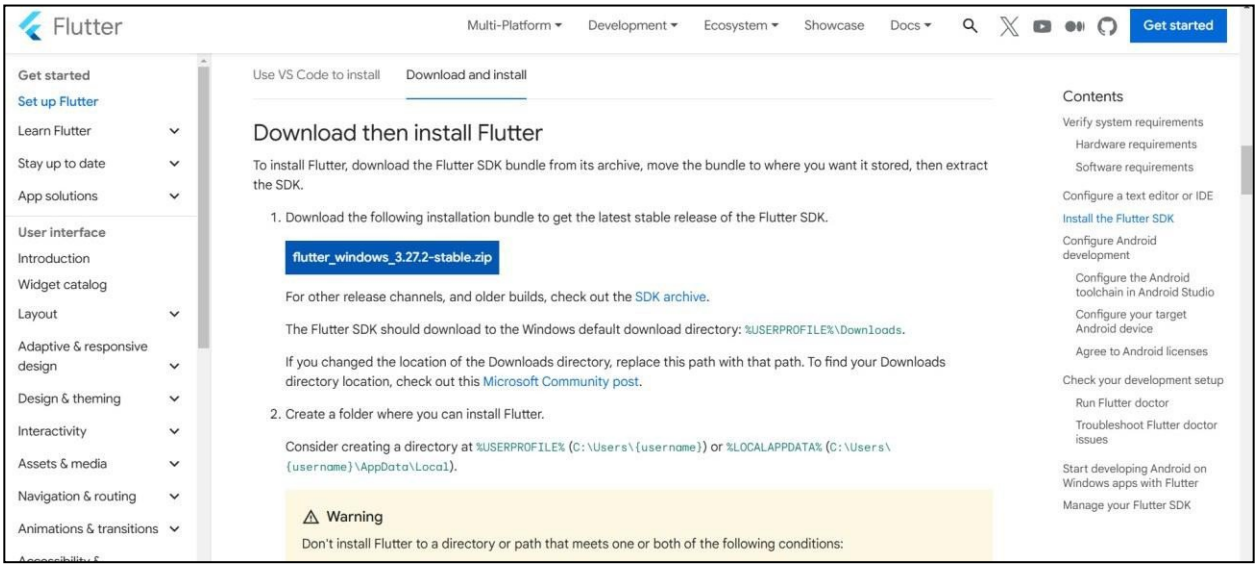
Step 1: Go to the official Flutter website: https://docs.flutter.dev/get-started/install



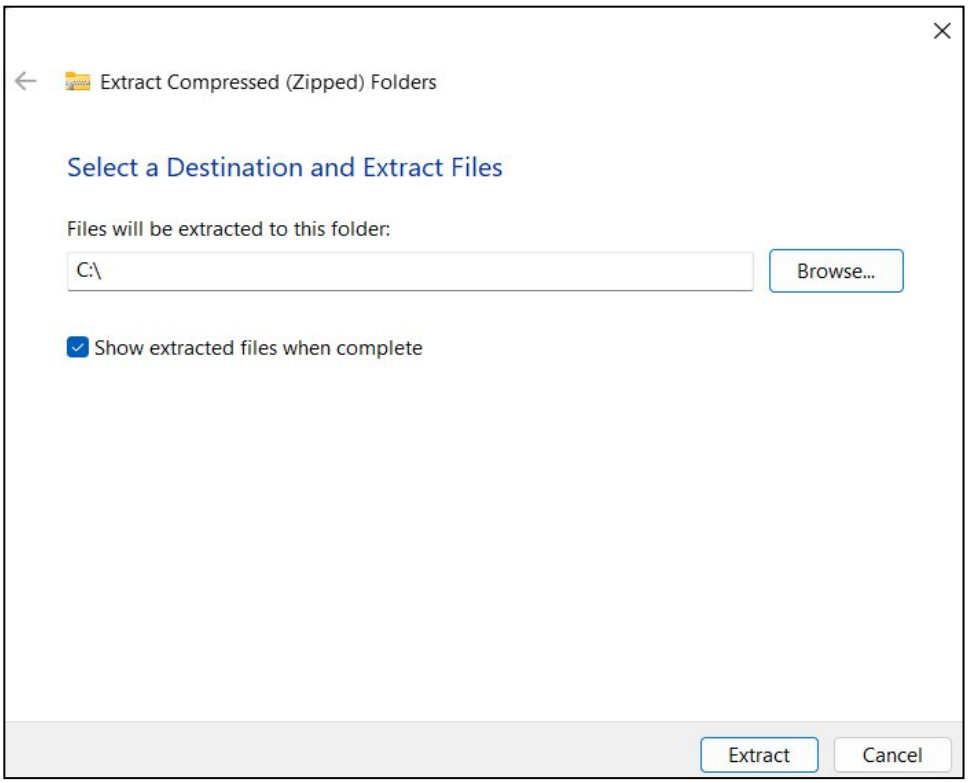
Step 2: To download the latest Flutter SDK, click on the Windows icon > Android



Step 3: For Windows, download the stable release (a .zip file).



Step 4: Extract the ZIP file to a folder (e.g., C:\flutter).

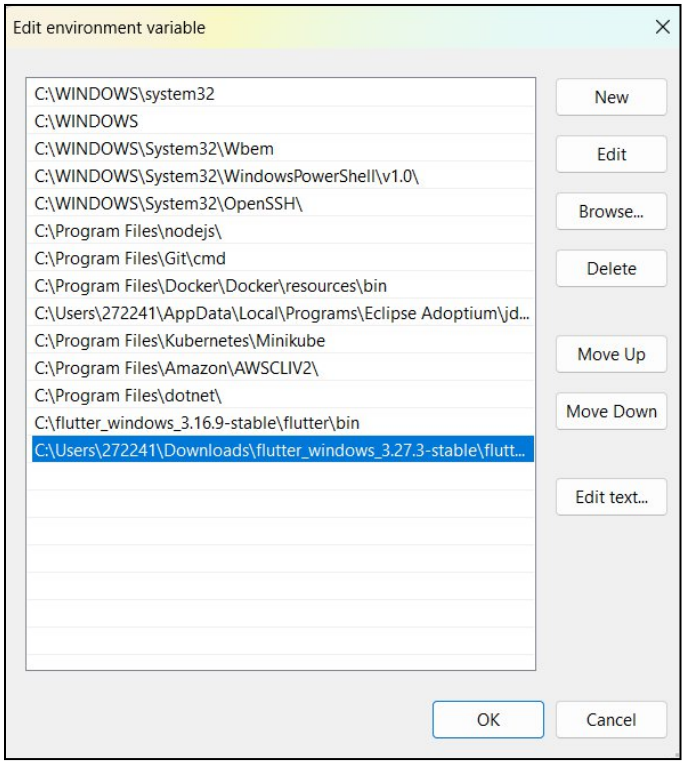


Step 5 :- Add Flutter to System PATH

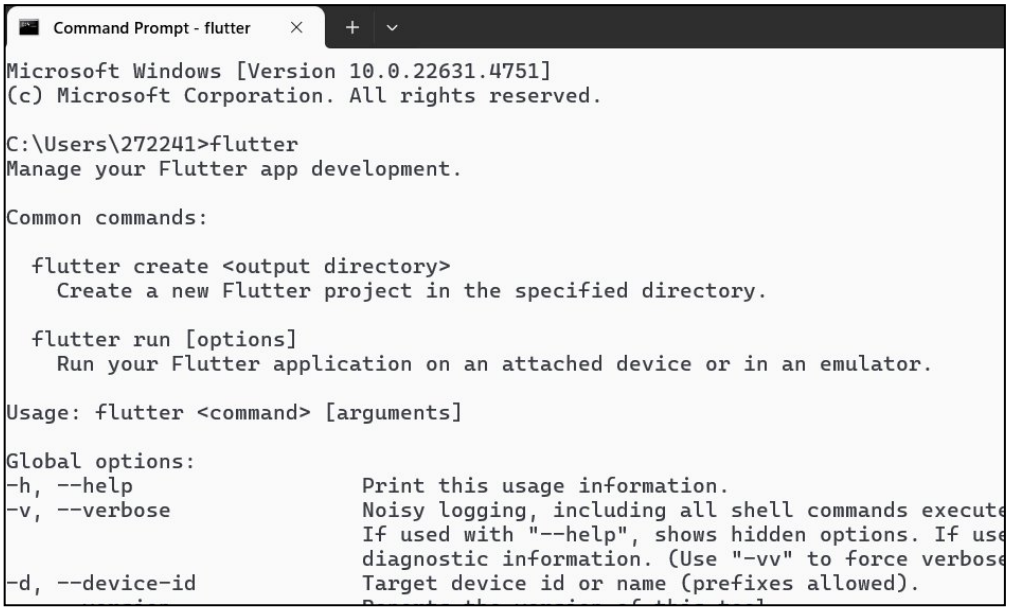
Right-click on the Start Menu > System > Advanced system settings > Environment Variables.

Under System Variables, find Path and click Edit.

Add the full path to the flutter/bin directory (e.g., C:\flutter\bin).



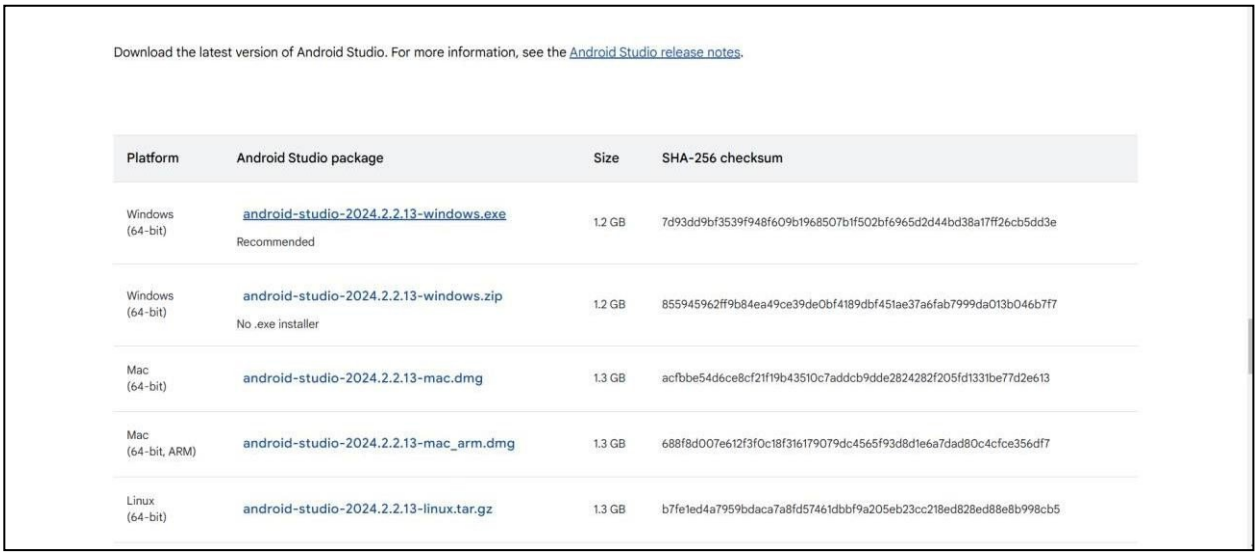
Step 6 : - Now, run the $ flutter command in command prompt.



Step 7:- 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

```
C:\Users\272241>flutter doctor
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 er
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[✓] Android toolchain - develop for Android devices (Android SDK version 34.0.0)
[✓] Chrome - develop for the web
[!] Visual Studio - develop Windows apps (Visual Studio Community 2022 17.12.4)
    X Visual Studio is missing necessary components. Please re-run the Visual Studio installer
      development with C++" workload, and include these components:
        MSVC v142 - VS 2019 C++ x64/x86 build tools
         - If there are multiple build tool versions available, install the latest
        C++ CMake tools for Windows
        Windows 10 SDK
[✓] Android Studio (version 2023.1)
[✓] VS Code (version 1.96.4)
[✓] Connected device (3 available)
[✓] Network resources

! Doctor found issues in 1 category.
```

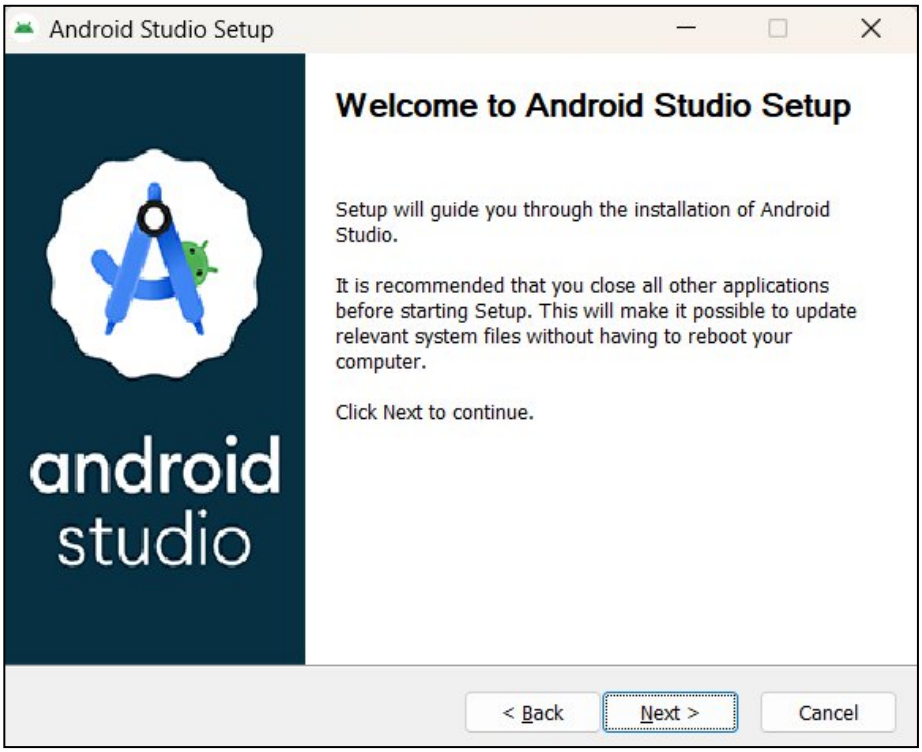
Step 8 : - Go to Android Studio and download the installer.



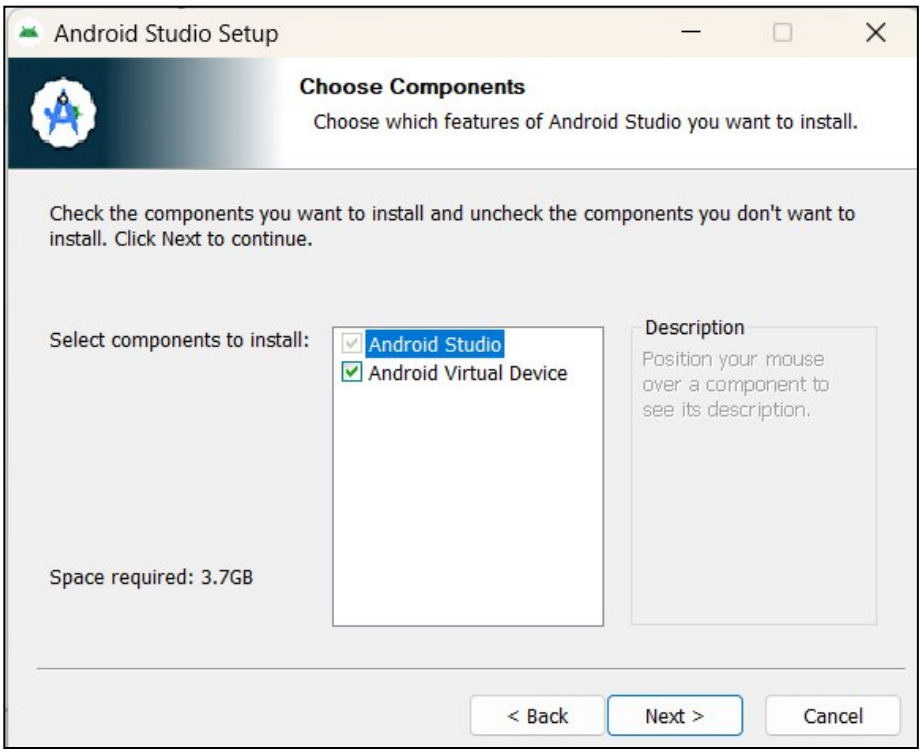
Download the latest version of Android Studio. For more information, see the Android Studio release notes.

Platform	Android Studio package	Size	SHA-256 checksum
Windows (64-bit)	android-studio-2024.2.2.13-windows.exe Recommended	1.2 GB	7d93dd9bf3539f948f609b1968507b1f502bf6965d2d44bd38a17ff26cb5dd3e
Windows (64-bit)	android-studio-2024.2.2.13-windows.zip No .exe installer	1.2 GB	855945962ff9b84ea49ce39de0bf4189dbf451ae37a6fab7999da013b046b7f7
Mac (64-bit)	android-studio-2024.2.2.13-mac.dmg	1.3 GB	acfbbe54d6ce8cf21f19b43510c7addcb9dde2824282f205fd1331be77d2e613
Mac (64-bit, ARM)	android-studio-2024.2.2.13-mac_arm.dmg	1.3 GB	688f8d007e612f3f0c18f316179079dc4565f93d8d1e6a7dad80c4cfce356df7
Linux (64-bit)	android-studio-2024.2.2.13-linux.tar.gz	1.3 GB	b7fe1ed4a7959bdaca7a8fd57461dbbf9a205eb23cc218ed828ed88e8b998cb5

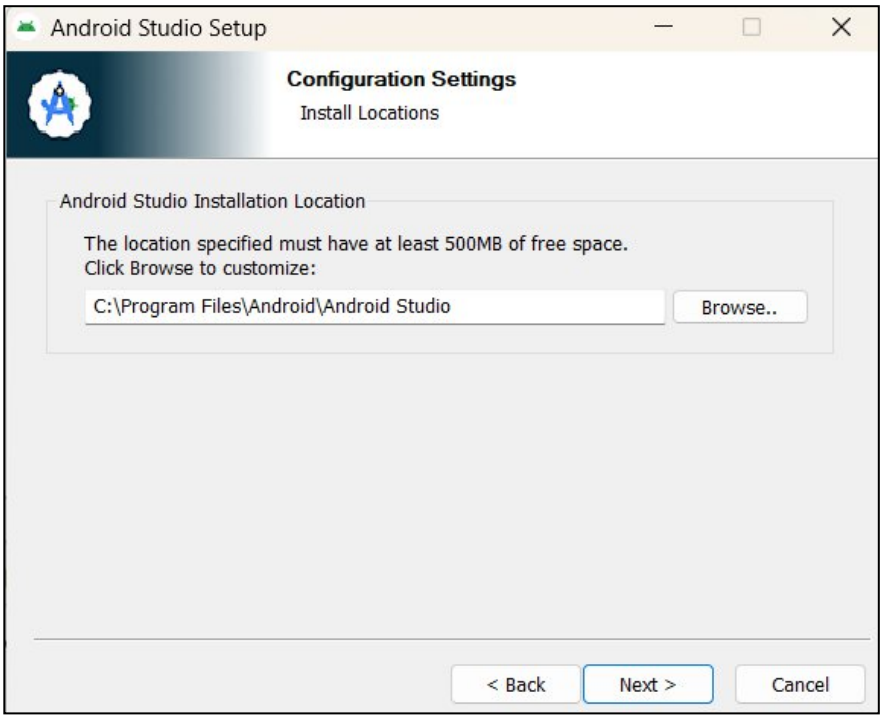
Step 8.1: - When the download is complete, open the .exe file and run it. You will get the following dialog box



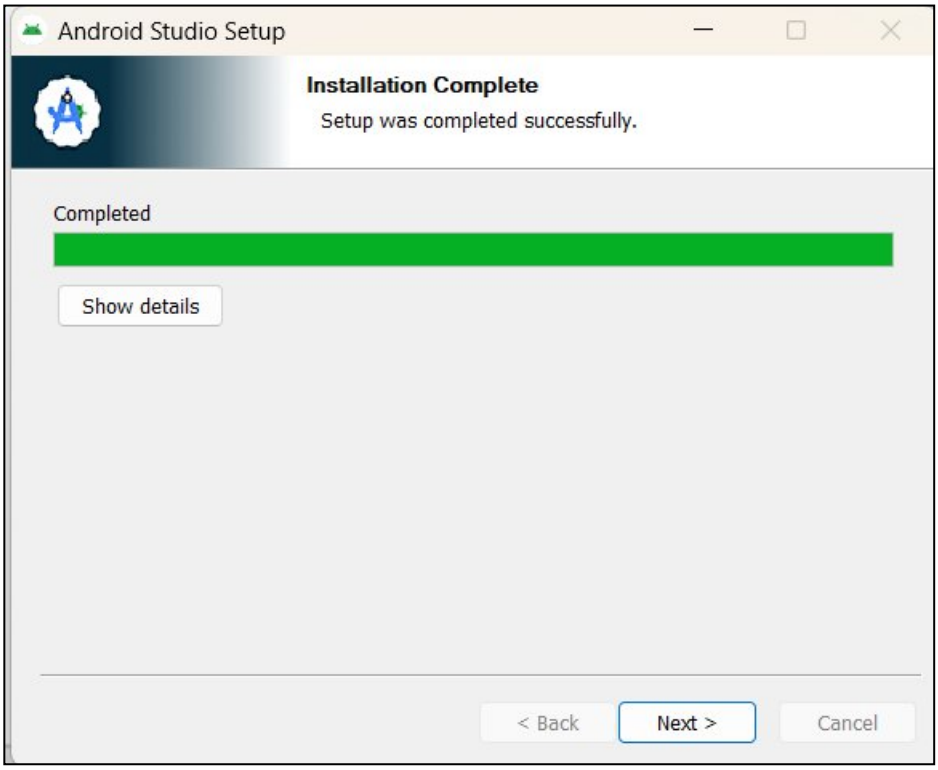
Step 8.2: - Select all the Checkboxes and Click on 'Next' Button.

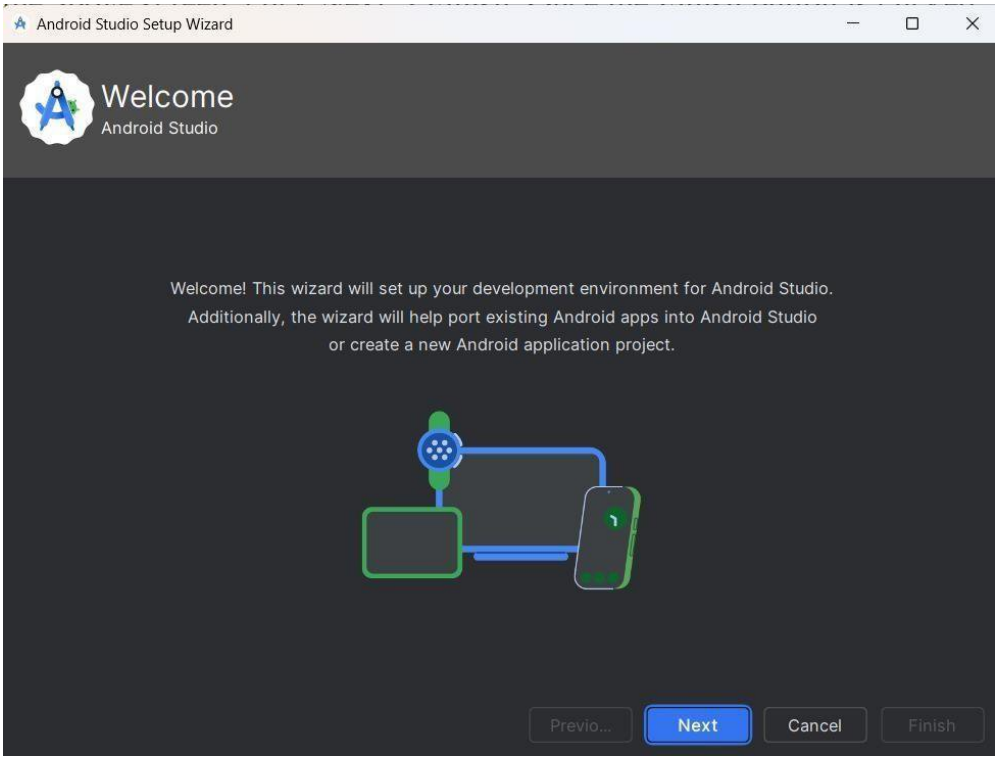


Step 8.3: - Change the destination as per your convenience and click on ‘Next’ Button.

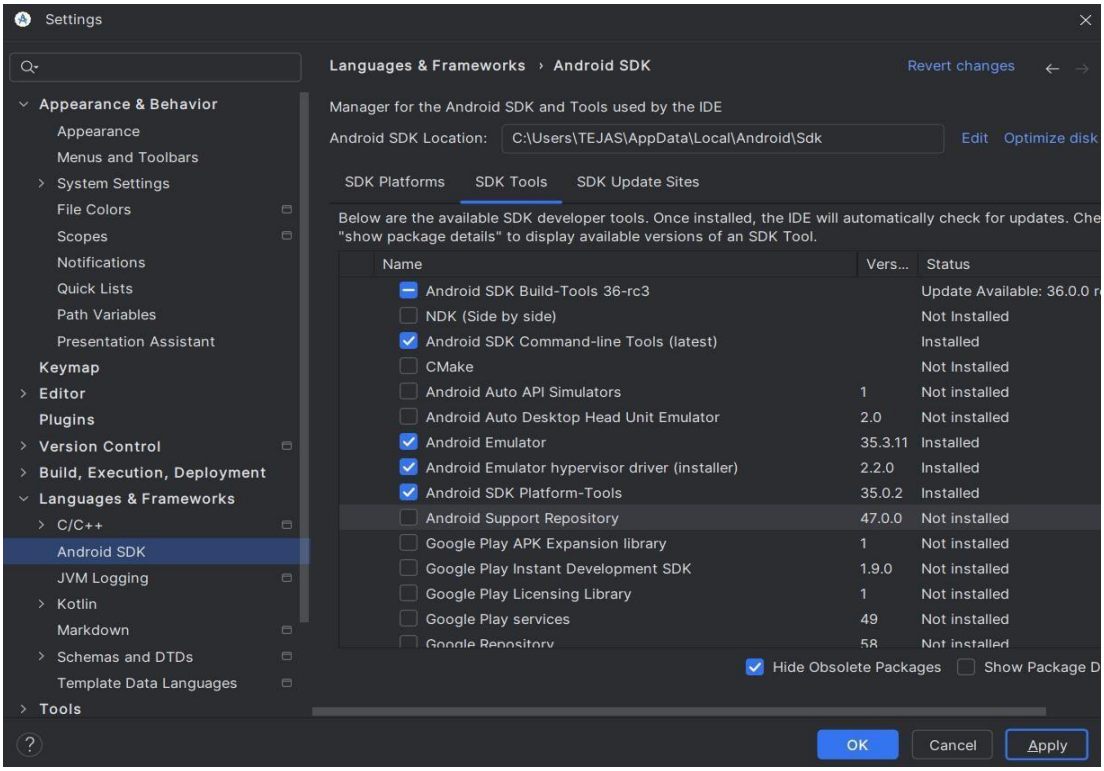


Step 8.4: - Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.





Step 8.5: - Go to Preferences > Appearance & Behavior > System Settings > Android SDK. Select the SDK Tools tab and check Android SDK Command-line Tools and Install it.

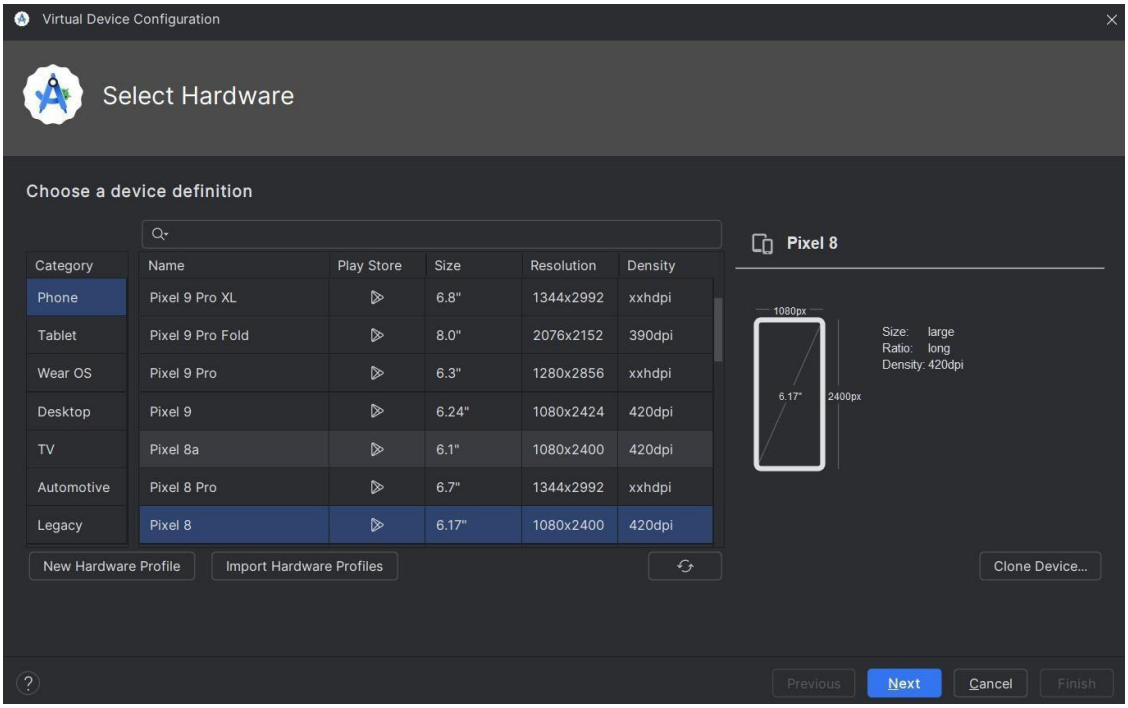


Step 9: - Open a terminal and run the following command

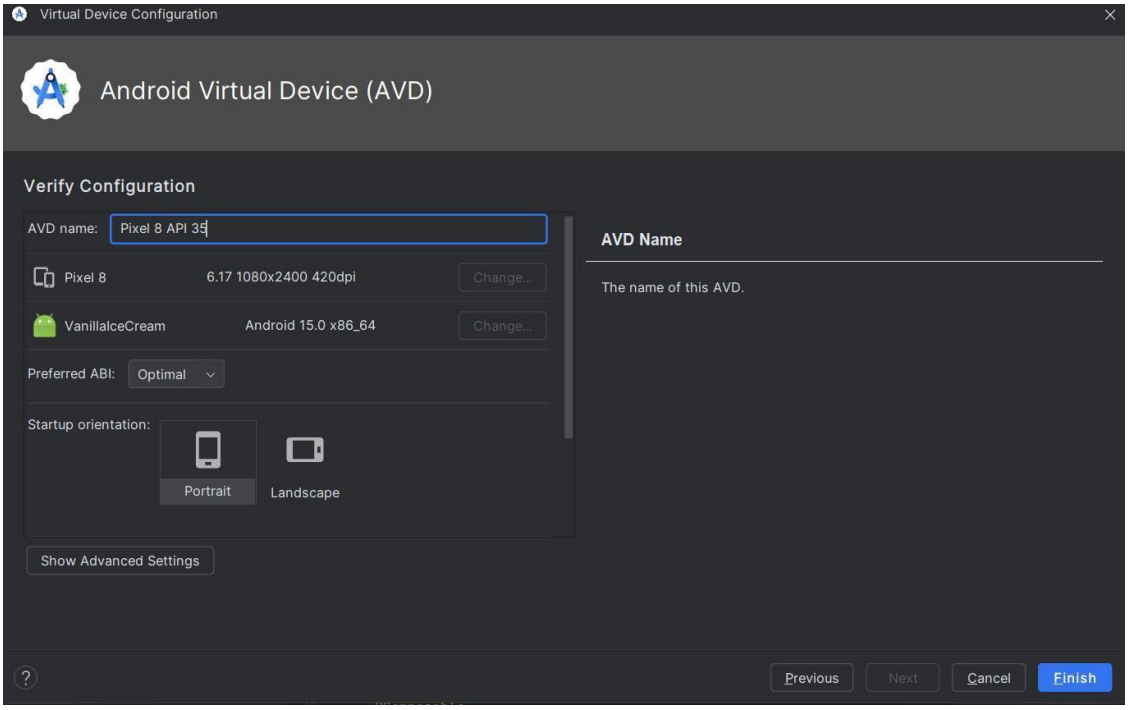
```
C:\Users\272241>flutter doctor --android-licenses
                                                                                Warning: Errors during XML parse:
Warning: Additionally, the fallback loader failed to parse the XML.
Warning: Errors during XML parse:       ] 64% Fetch remote repository...
Warning: Additionally, the fallback loader failed to parse the XML.ry...
[======================================] 100% Computing updates...
All SDK package licenses accepted.


C:\Users\272241>
```

```
C:\Users\272241>flutter doctor
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
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[✓] Android toolchain - develop for Android devices (Android SDK version 34.0.0)
[✓] Chrome - develop for the web
[!] Visual Studio - develop Windows apps (Visual Studio Community 2022 17.12.4)
    X Visual Studio is missing necessary components. Please re-run the Visual Studio installer
      development with C++" workload, and include these components:
        MSVC v142 - VS 2019 C++ x64/x86 build tools
         - If there are multiple build tool versions available, install the latest
        C++ CMake tools for Windows
        Windows 10 SDK
[✓] Android Studio (version 2023.1)
[✓] VS Code (version 1.96.4)
[✓] Connected device (3 available)
[✓] Network resources

! Doctor found issues in 1 category.
```

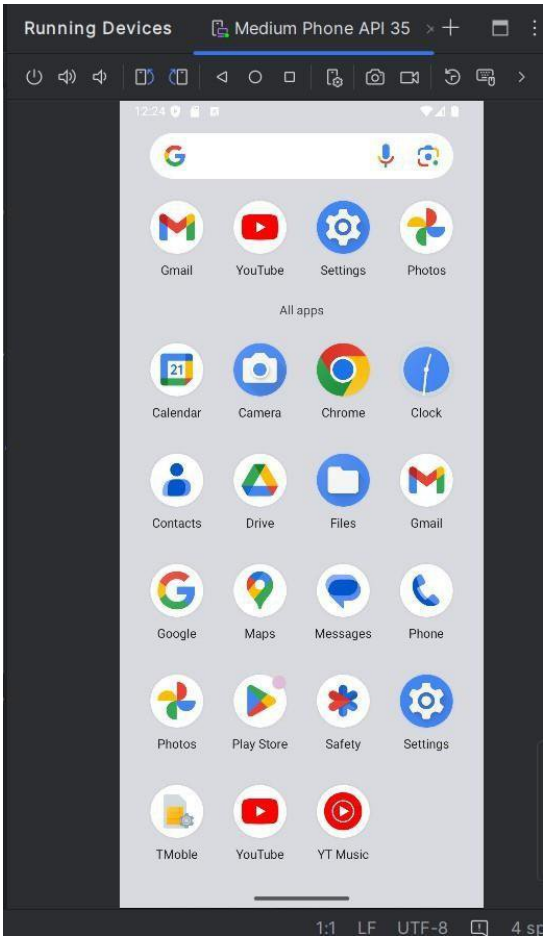
Step 10: - Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application



Step 10.1: - Open Android Studio and go to Tools > AVD Manager. Create a new virtual device.

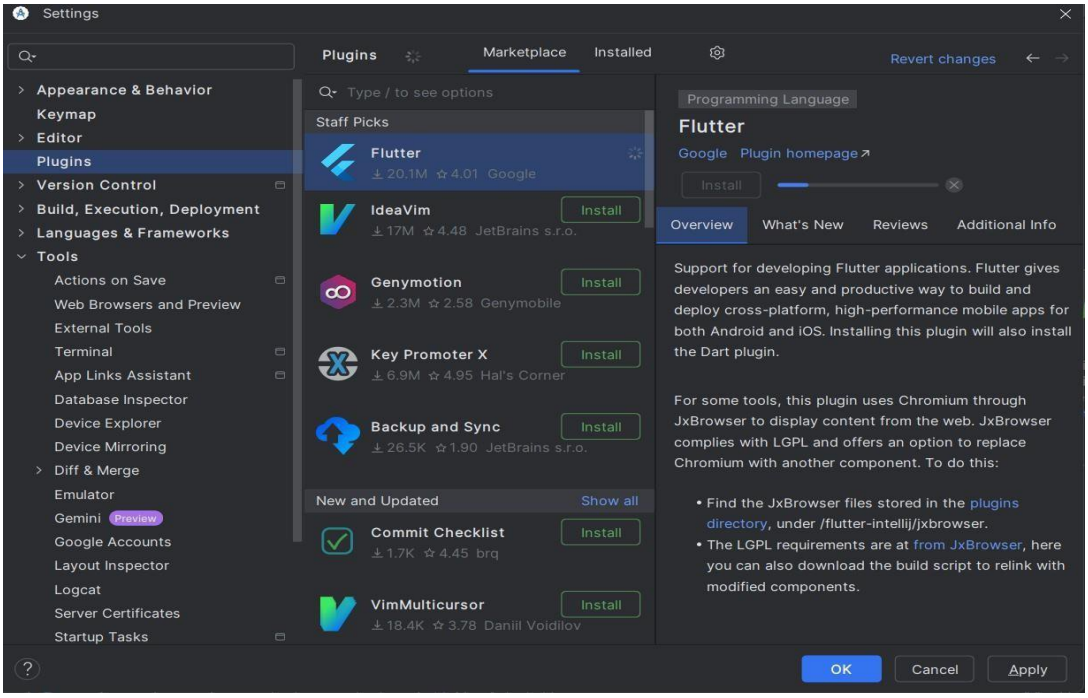


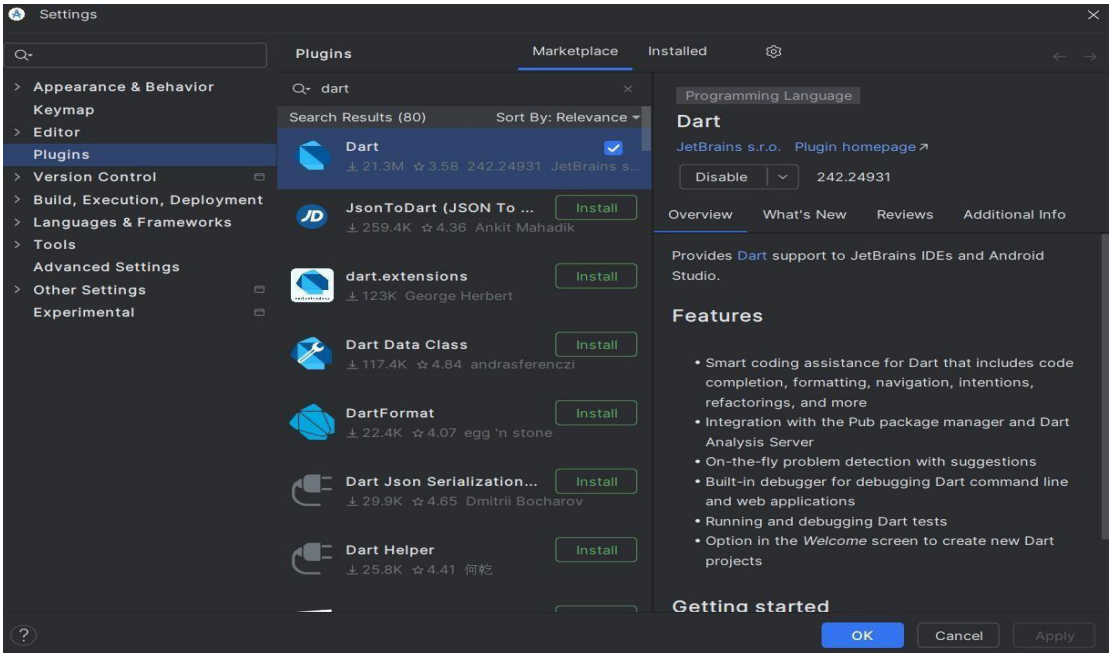
Step 10.2: - Click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen



Step 11: - Now, install Flutter and Dart plugin for building Flutter application in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself

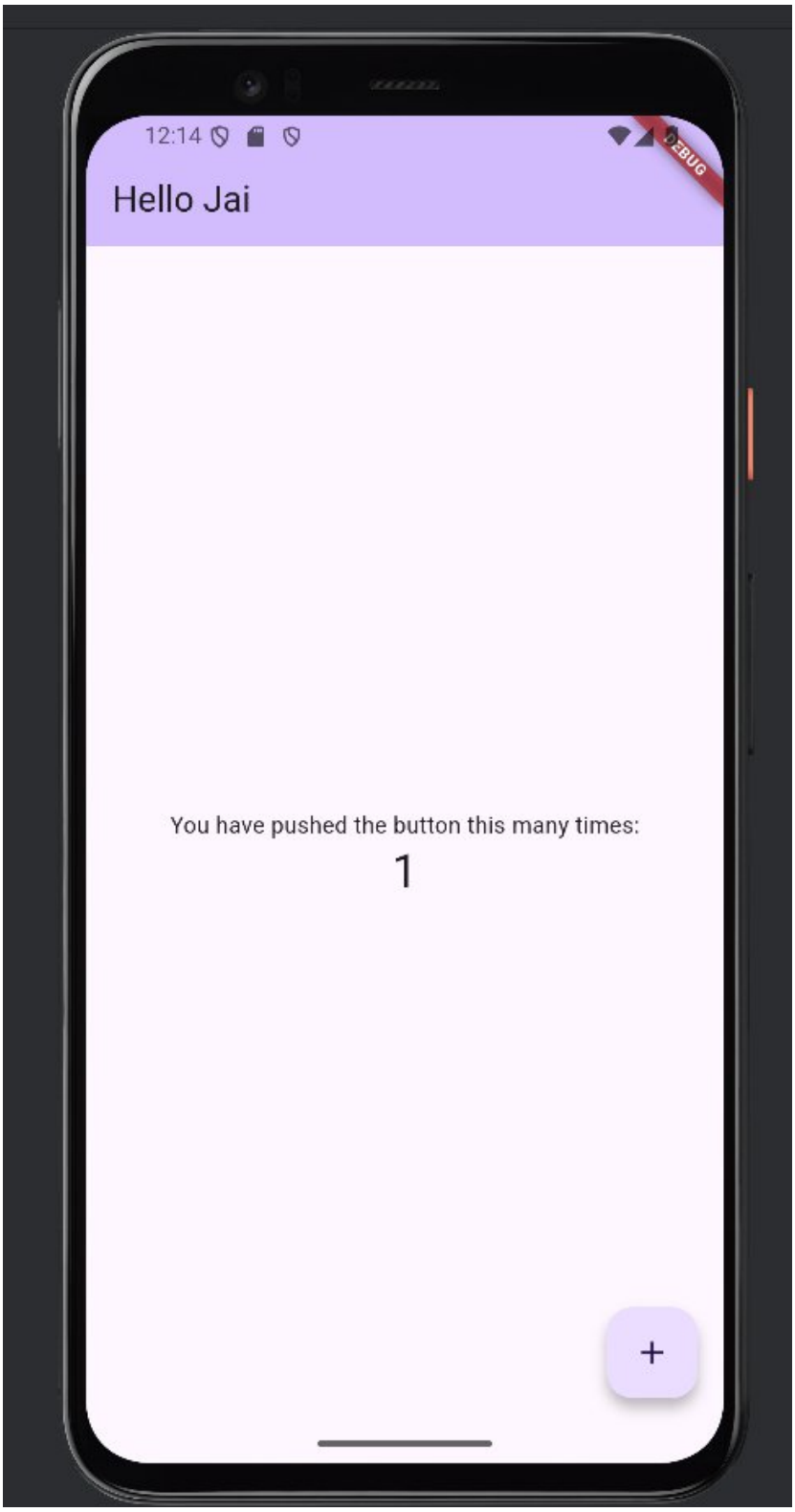
Step 11.1: - Open the Android Studio and then go to File->Settings->Plugins. Now, search the Flutter plugin. If found, select Flutter plugin and click install





Step 11.2: - Restart the Android Studio

Step 12: - Go to File > New Project > Create Flutter Project, then select the project name and location, and click Next to proceed.



Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	59
Name	Jai Talreja
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.

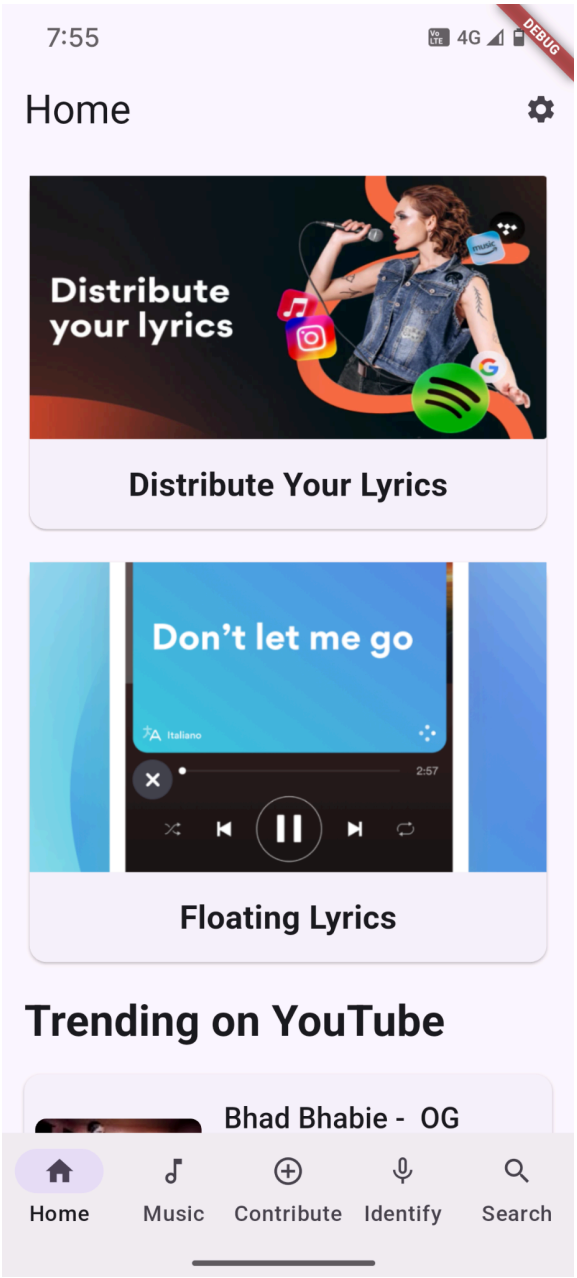
6. **Displaying Lists and Grids**:
   - **ListView**: The ListView widget is used to display a list of items that can scroll. It's perfect for long lists that need to be dynamically generated.
   - **GridView**: This widget allows you to display items in a grid format, with configurable row and column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

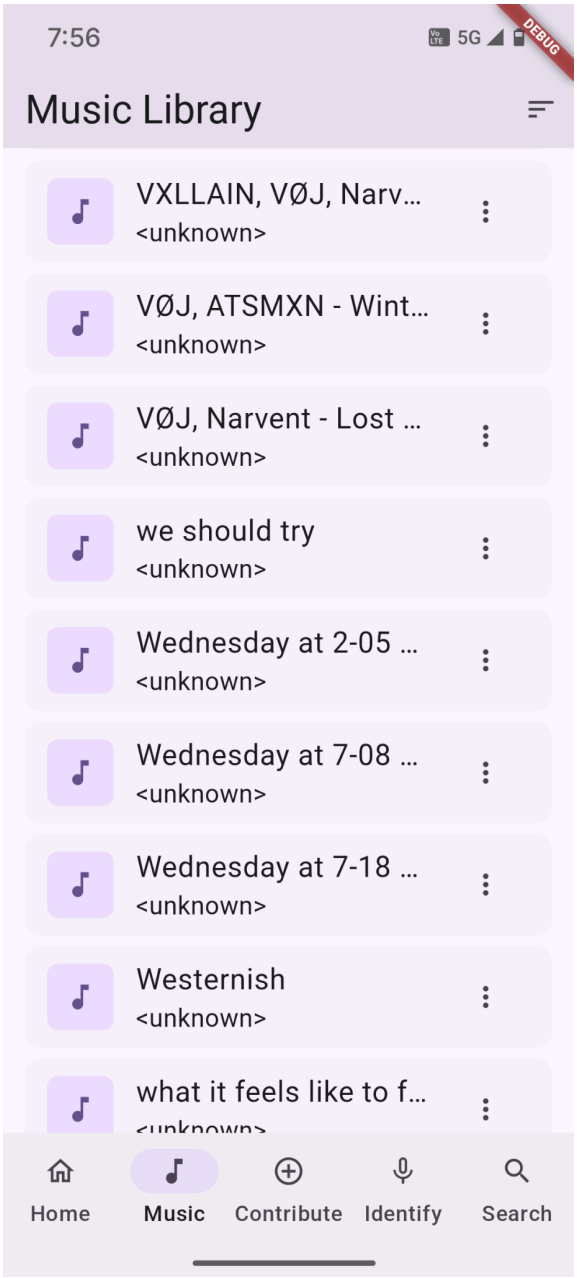
7. **State Management**:
   - Flutter provides a variety of ways to manage state. The simplest approach is using setState() to update the UI. For more complex apps, you can use state management solutions like **Provider**, **Riverpod**, or **Bloc** to separate business logic from UI code.
   - Proper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

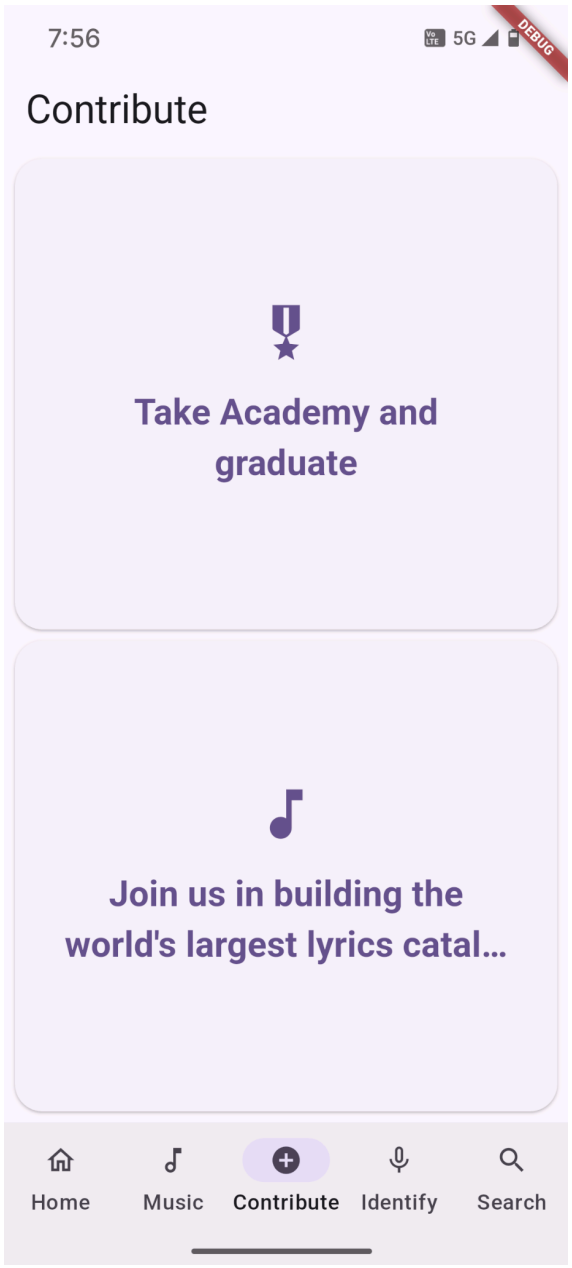
8. **Theming and Styling**:
   - Flutter allows you to define a global **Theme** for your app using ThemeData, which ensures consistent styling across the entire app. You can customize colors, typography, and button styles.

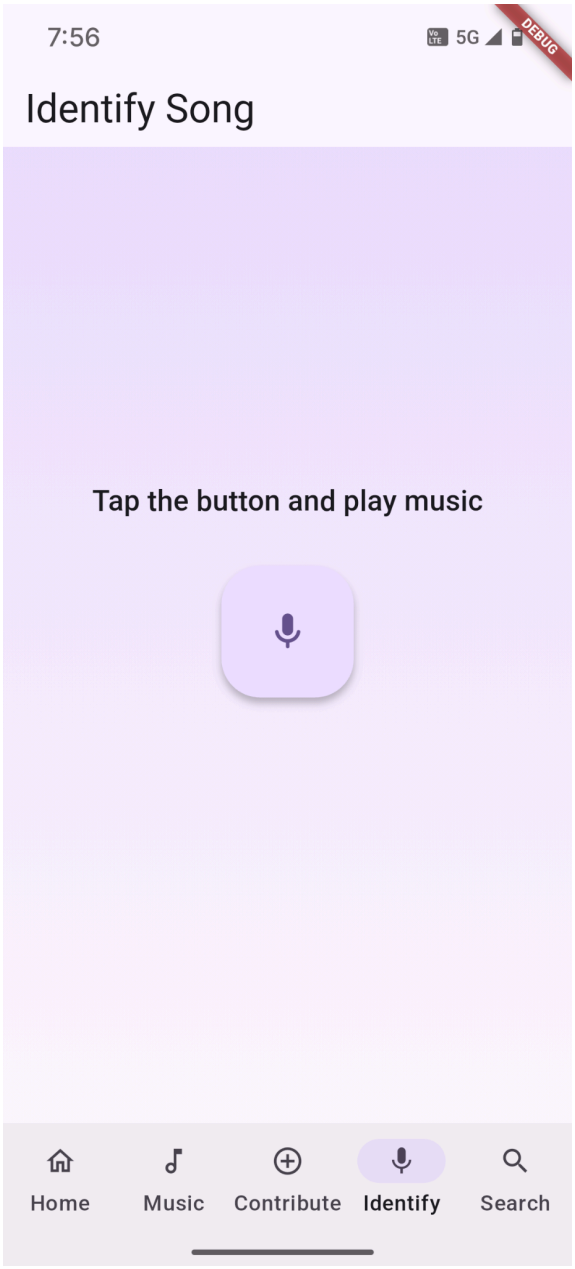
9. **Animations and Transitions**:
   - Flutter provides powerful animation support to create smooth and visually appealing transitions between UI states.
   - **AnimatedContainer**: A widget that animates changes in properties like width, height, or color over a given duration.
   - You can also create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

Screenshots:

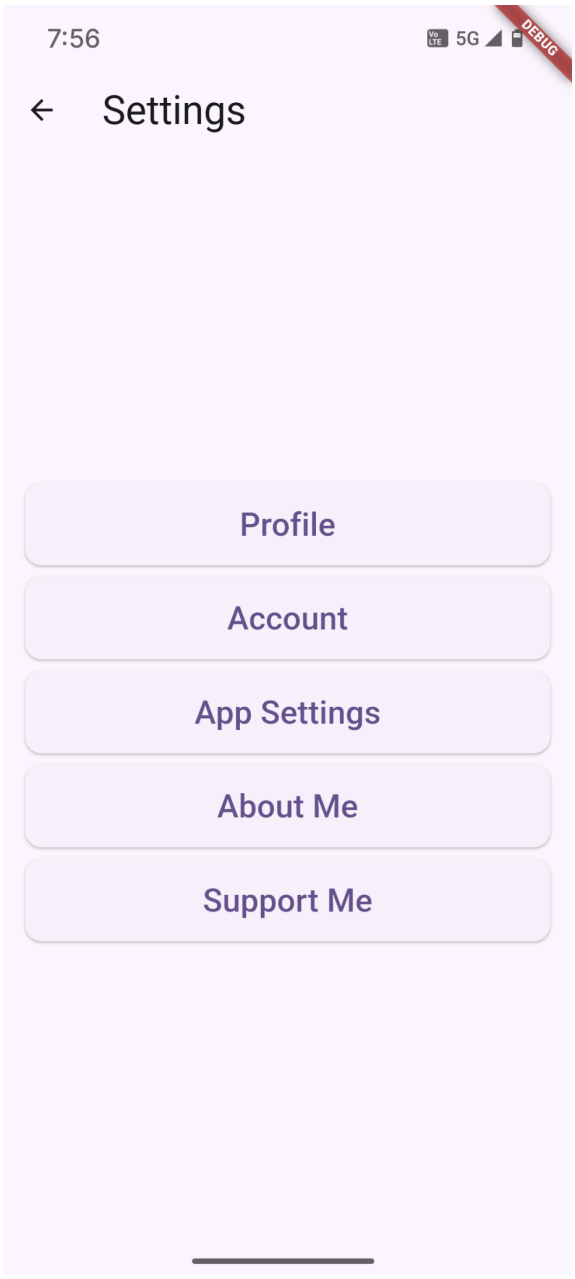


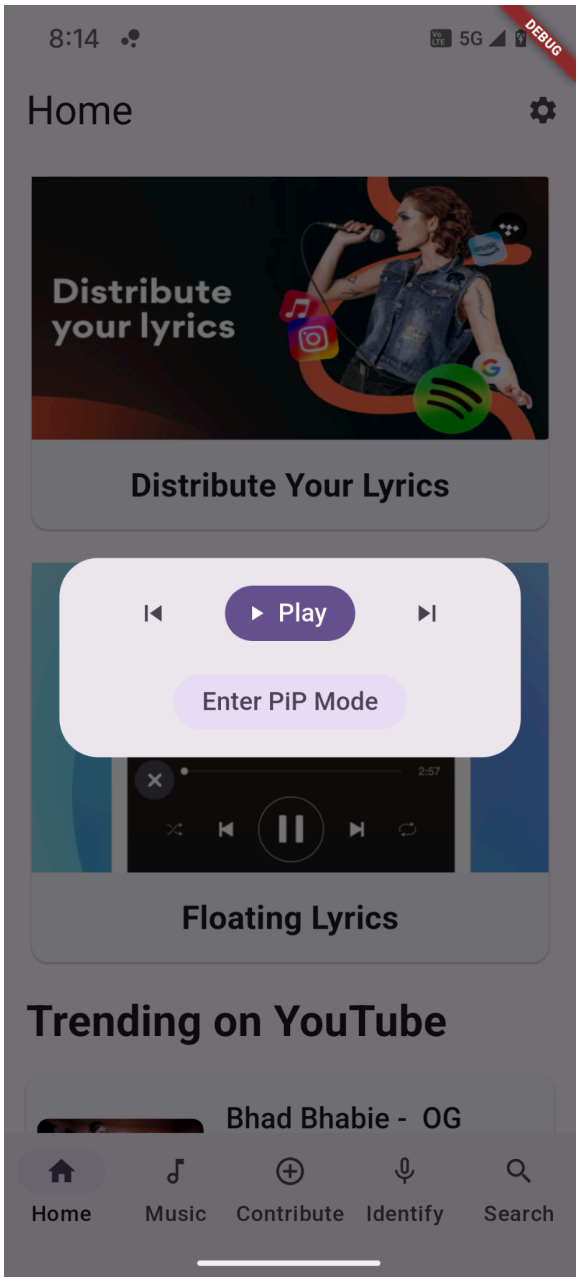


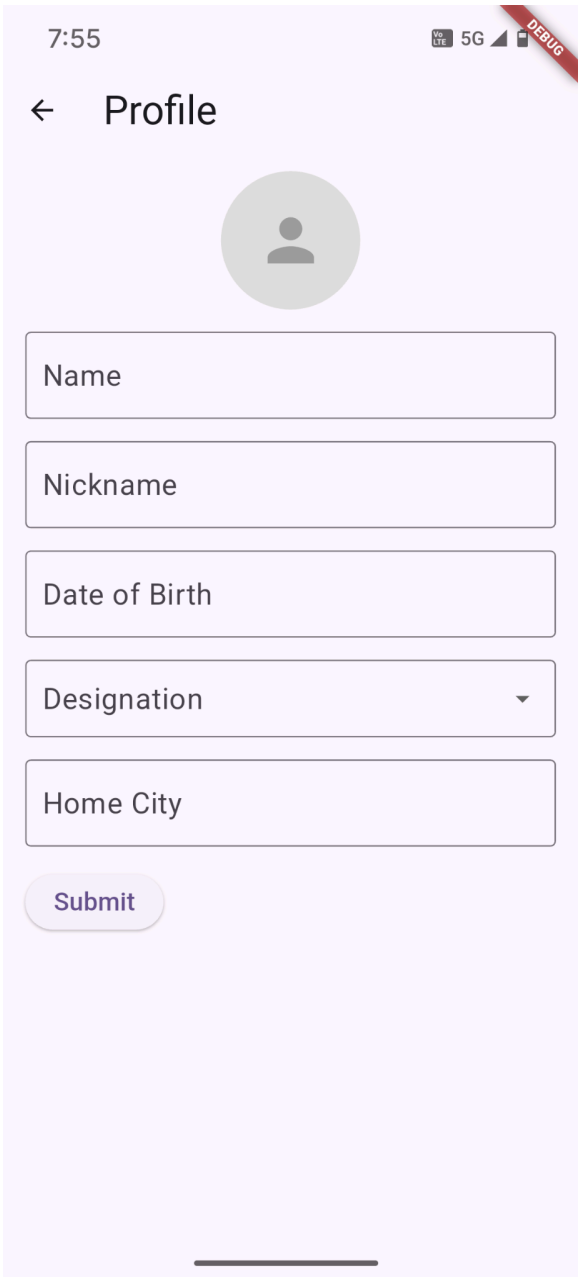

7:56
5G
DEBUG
Contribute
Take Academy and graduate
Join us in building the world's largest lyrics catal...
Home
Music
Contribute
Identify
Search


7:56
5G
DEBUG
Identify Song
Tap the button and play music
Home
Music
Contribute
Identify
Search


7:57
5G
DEBUG
Search
Search...
Home
Music
Contribute
Identify
Search


7:56
5G
DEBUG
Settings
Profile
Account
App Settings
About Me
Support Me


8:14
5G
DEBUG
Home
Distribute your lyrics
Distribute Your Lyrics
Play
Enter PiP Mode
Floating Lyrics
Trending on YouTube
Bhad Bhabie - OG
Home
Music
Contribute
Identify
Search


7:55
5G
DEBUG
Profile
Name
Nickname
Date of Birth
Designation
Home City
Submit

Code Snippets:

Container :

```
Container(
  width: 200.0,
  height: 100.0,
  color: Colors.blue,
  child: Center(
    child: Text(
      'Hello, Flutter!',
      style: TextStyle(
        color: Colors.white,
        fontSize: 20.0,
      ),
    ),
  ),
)
```

Column :

```
Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    Text('Item 1'),
    Text('Item 2'),
    Text('Item 3'),
  ],
)
```

Row :

```
Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: <Widget>[
    Icon(Icons.home),
    Icon(Icons.search),
    Icon(Icons.settings),
  ],
)
```

Text :

```
Text(
  'Flutter is awesome!',
  style: TextStyle(
    color: Colors.blue,
    fontSize: 24.0,
    fontWeight: FontWeight.bold,
  ),
)
```

Image :

```
Image.asset(
  'assets/images/flutter_logo.png',
  width: 100.0,
  height: 100.0,
)

Image.network(
  'https://www.example.com/flutter_logo.png',
  width: 100.0,
  height: 100.0,
)
```

ElevatedButton :

```
ElevatedButton(
  onPressed: () {
    print('ElevatedButton pressed');
  },
  child: Text('Click Me'),
  style: ElevatedButton.styleFrom(
    primary: Colors.blue,
    onPrimary: Colors.white,
  ),
)
```

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	59
Name	Jai Talreja
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. 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/distribute-your-music.png
    - assets/floating-lyrics.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:

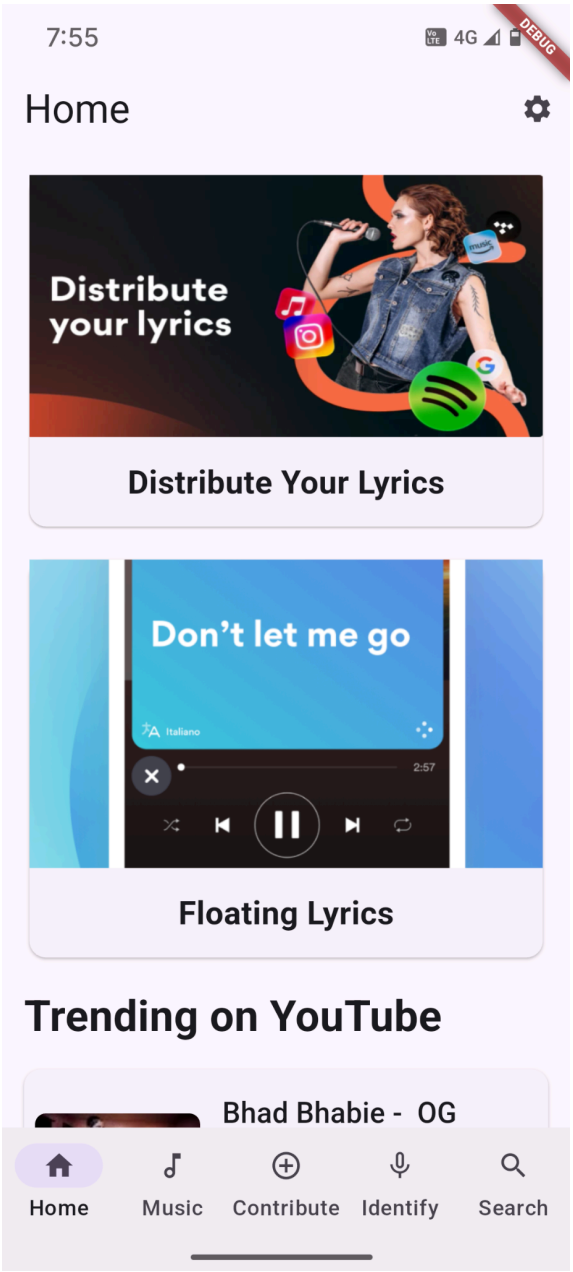
```
import 'package:flutter_svg/flutter_svg.dart';

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

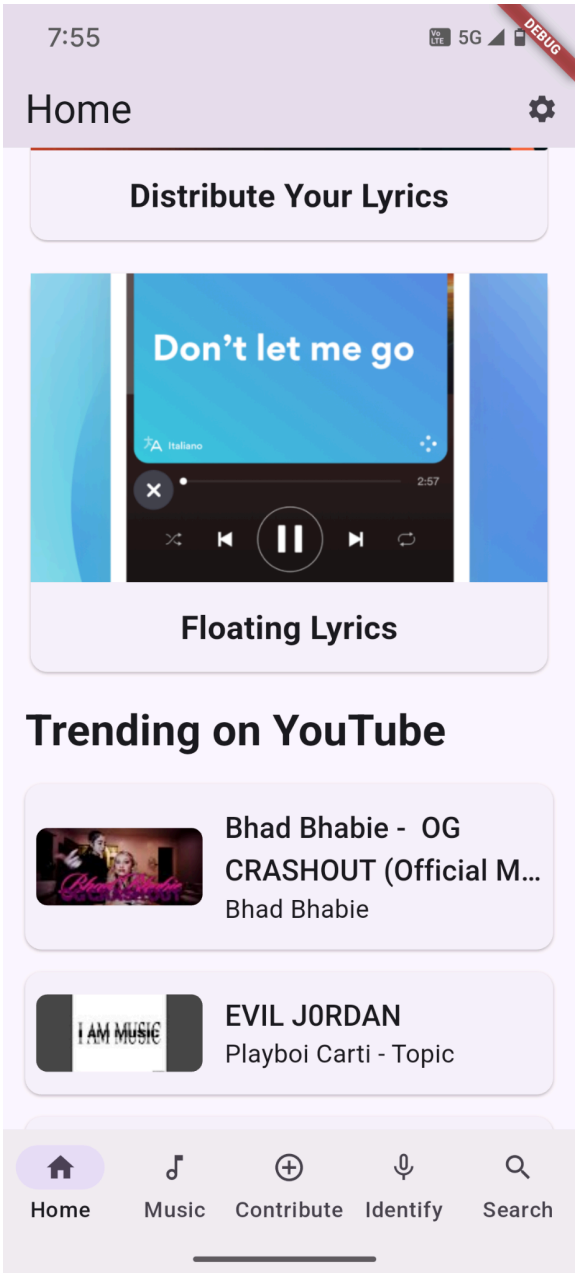
Summary of Key Steps:

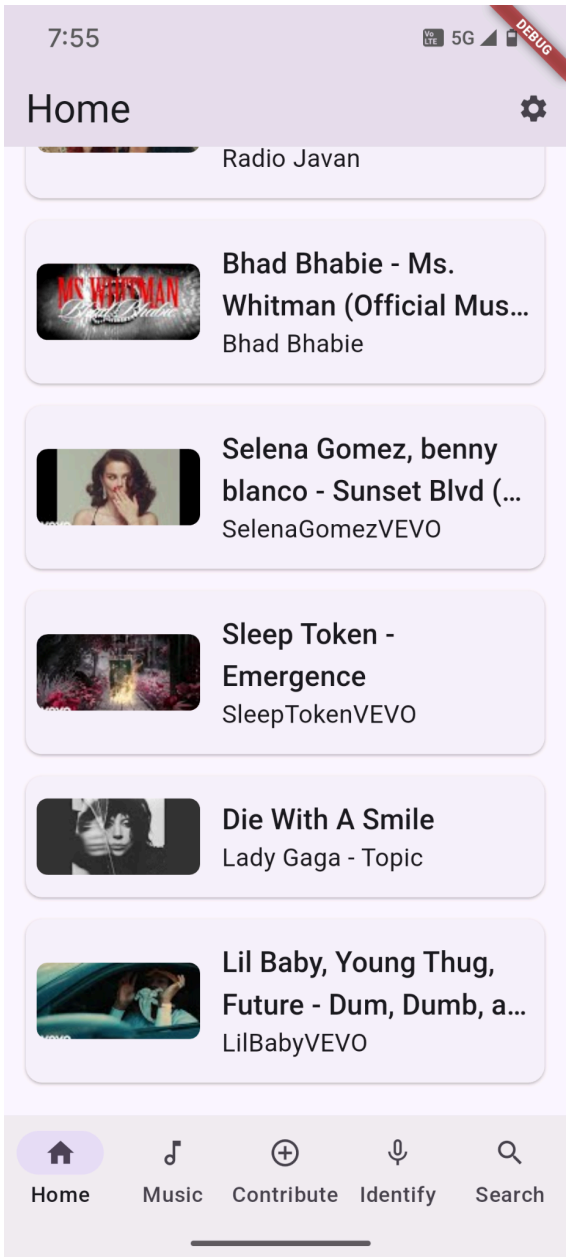
1. **Declare assets in pubspec.yaml.**
2. **For images and icons**, use Image.asset('path/to/image') and Image.asset('path/to/icon').
3. **For fonts**, declare them under the flutter section in pubspec.yaml, then use them via the TextStyle class.
4. **Use Icon Libraries**: Use Flutter's built-in Icons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

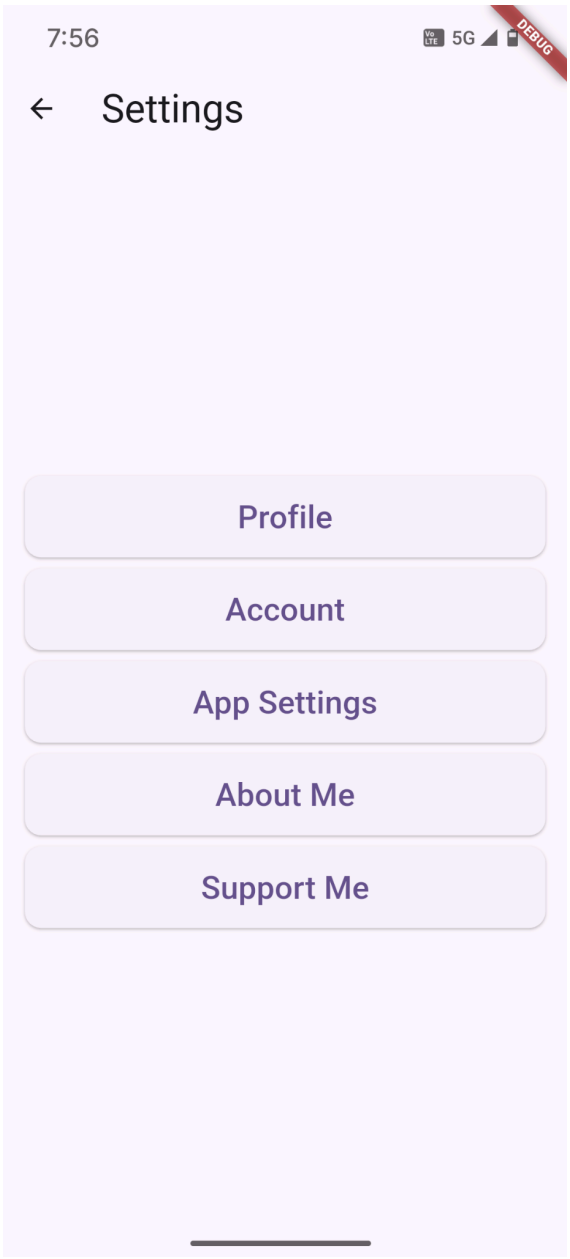
Understanding these concepts will help you effectively use images, icons, and fonts in your Flutter app.

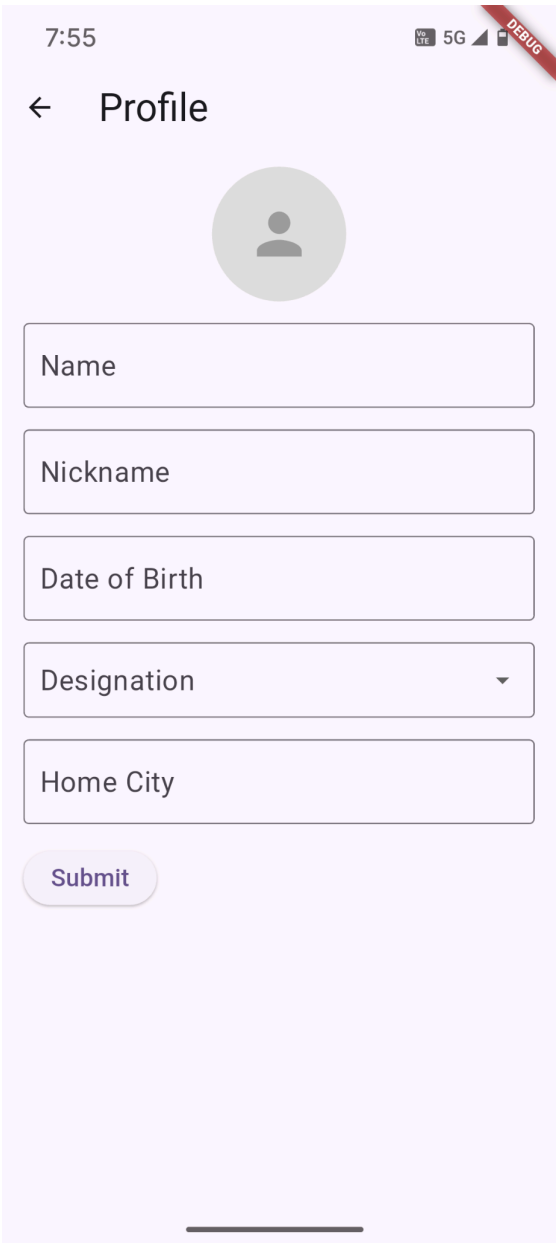
Screenshots:

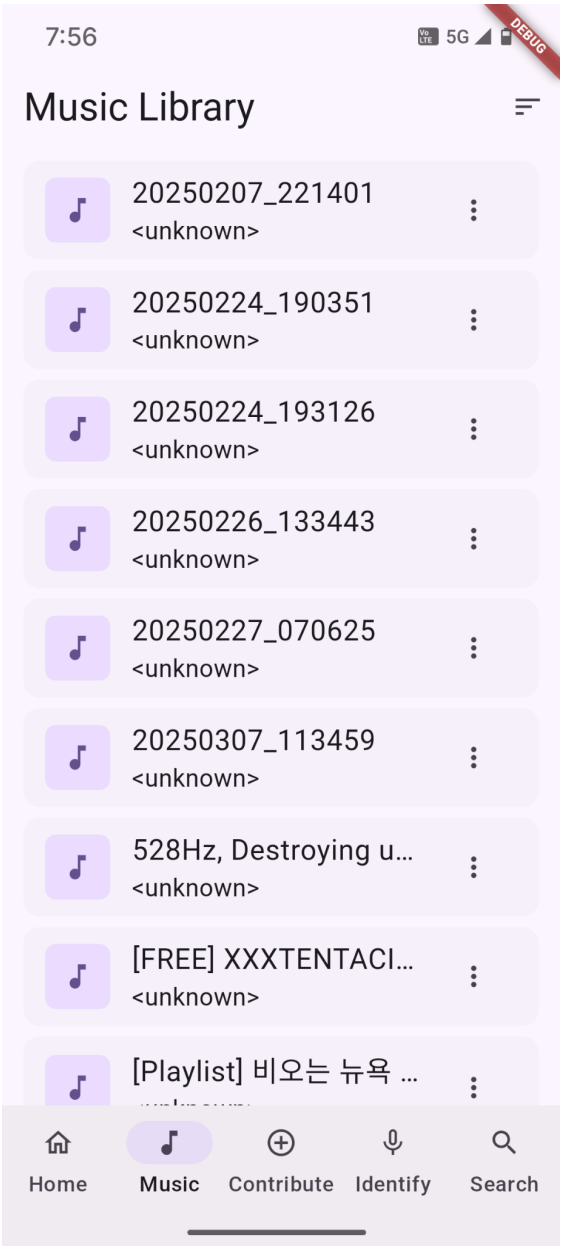


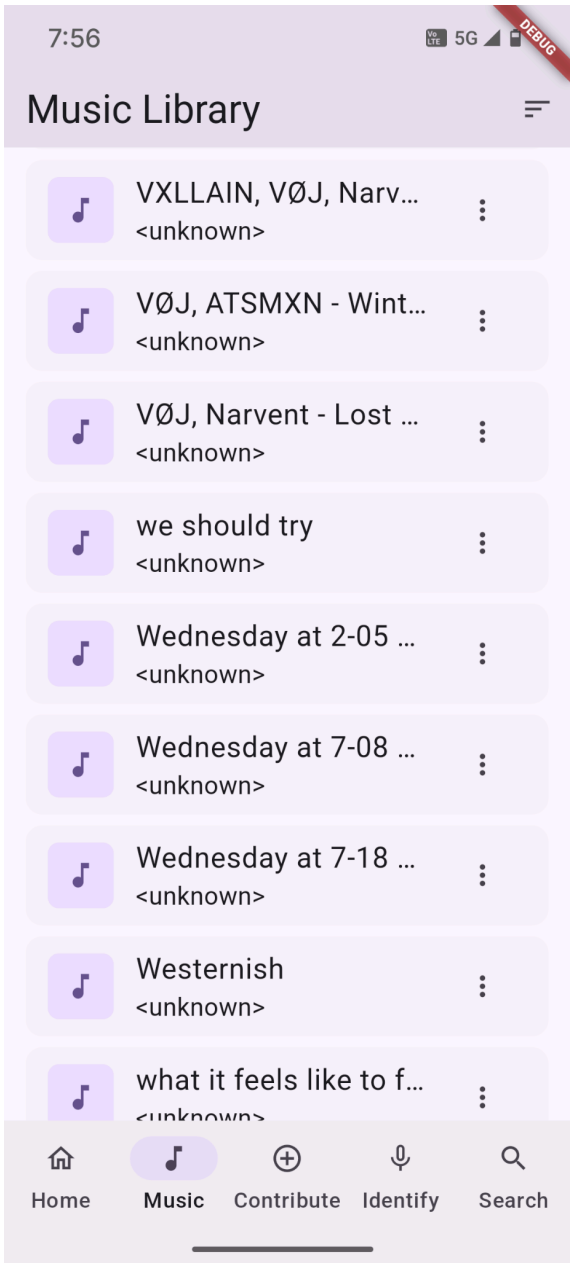


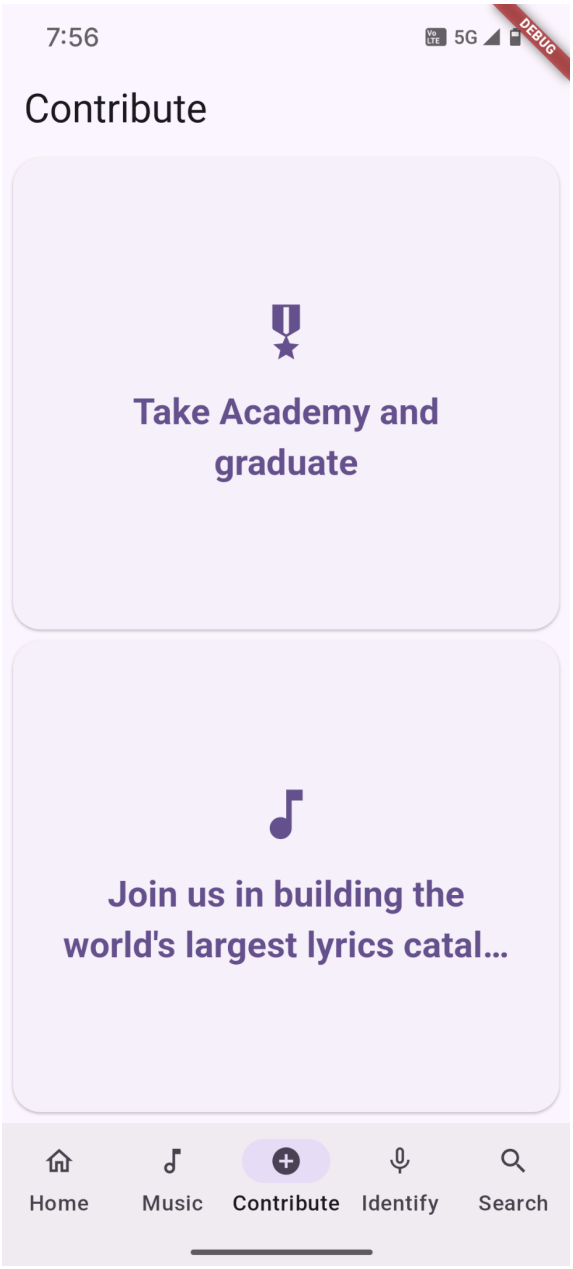

7:55
5G
DEBUG
Home
Radio Javan
Bhad Bhabie - Ms. Whitman (Official Mus...
Bhad Bhabie
Selena Gomez, benny blanco - Sunset Blvd (...
SelenaGomezVEVO
Sleep Token - Emergence
SleepTokenVEVO
Die With A Smile
Lady Gaga - Topic
Lil Baby, Young Thug, Future - Dum, Dumb, a...
LilBabyVEVO
Home
Music
Contribute
Identify
Search

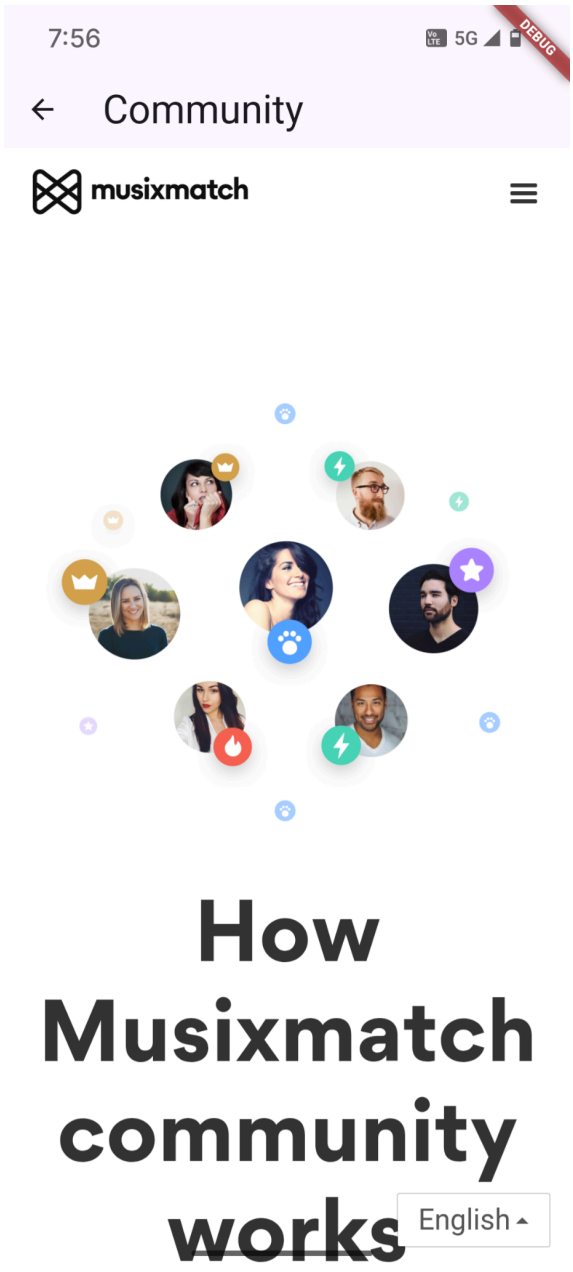

7:56
5G
DEBUG
Settings
Profile
Account
App Settings
About Me
Support Me

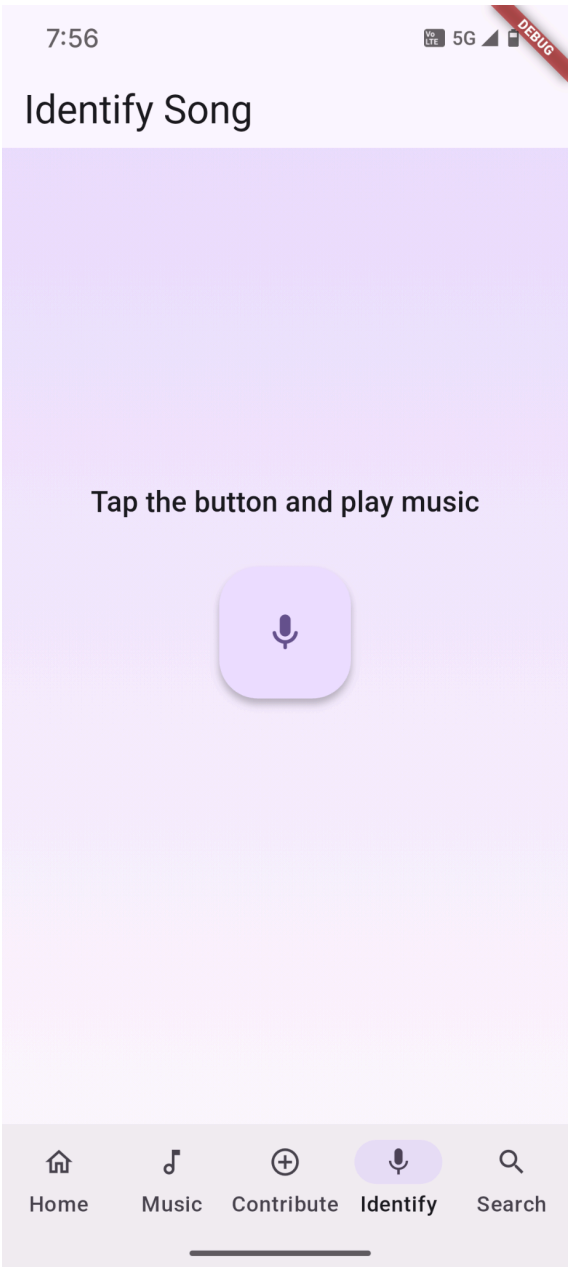

7:55
5G
DEBUG
Profile
Name
Nickname
Date of Birth
Designation
Home City
Submit


7:56
5G
DEBUG
Music Library
20250207_221401
<unknown>
20250224_190351
<unknown>
20250224_193126
<unknown>
20250226_133443
<unknown>
20250227_070625
<unknown>
20250307_113459
<unknown>
528Hz, Destroying u...
<unknown>
[FREE] XXXTENTACI...
<unknown>
[Playlist] 비오는 뉴욕 ...
Home
Music
Contribute
Identify
Search

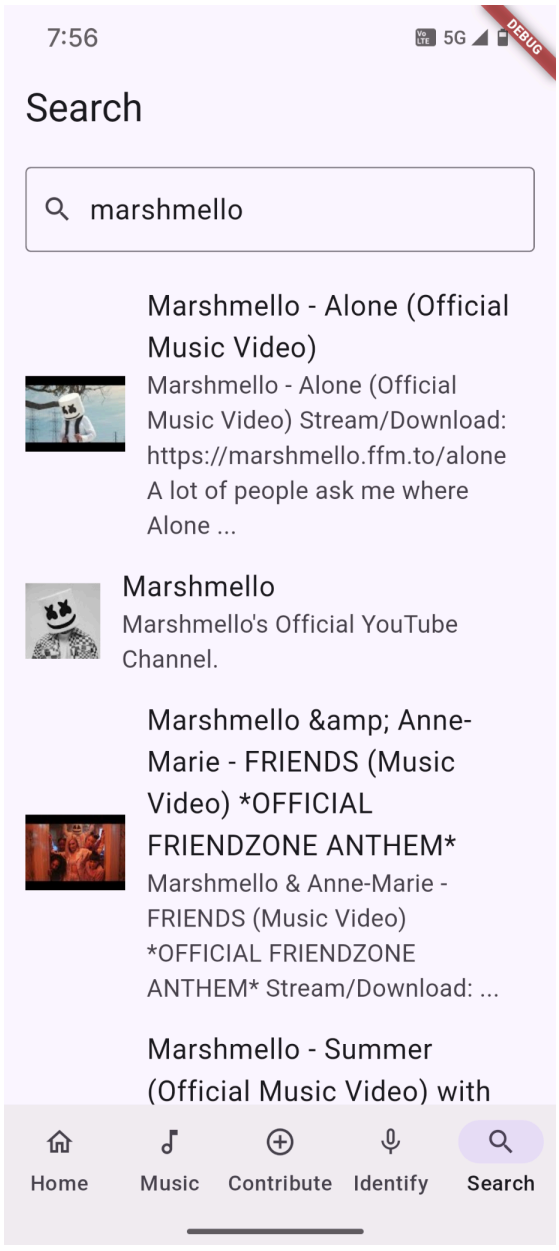

7:56
5G
DEBUG
Music Library
VXLLAIN, VØJ, Narv...
<unknown>
VØJ, ATSMXN - Wint...
<unknown>
VØJ, Narvent - Lost ...
<unknown>
we should try
<unknown>
Wednesday at 2-05 ...
<unknown>
Wednesday at 7-08 ...
<unknown>
Wednesday at 7-18 ...
<unknown>
Westernish
<unknown>
what it feels like to f...
Home
Music
Contribute
Identify
Search

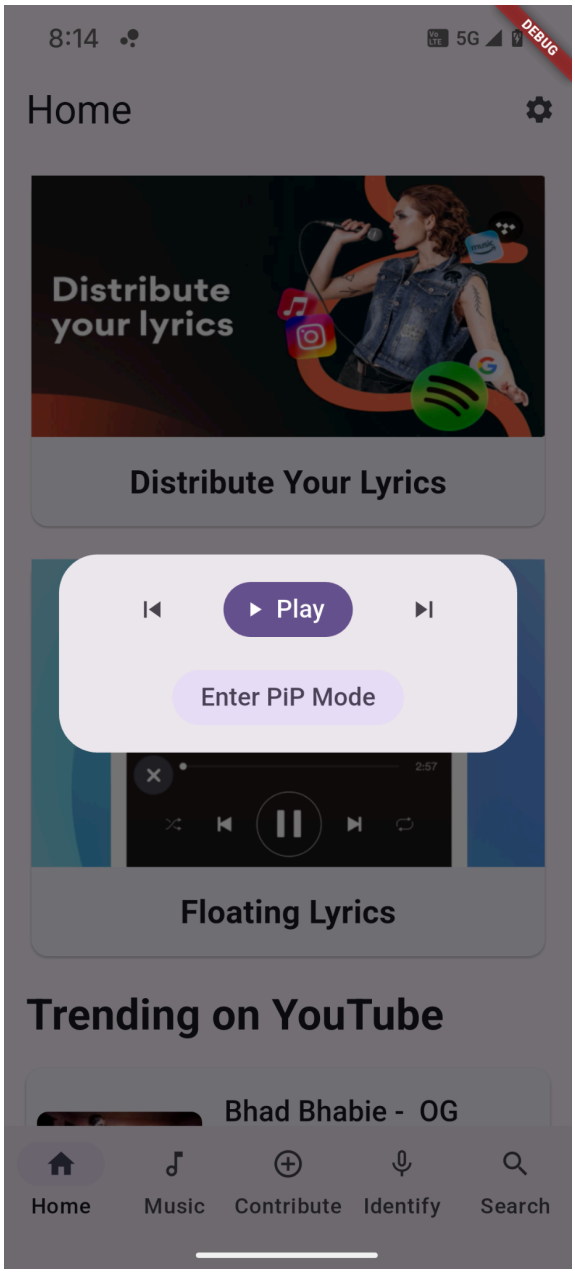

7:56
5G
DEBUG
Contribute
Take Academy and graduate
Join us in building the world's largest lyrics catal...
Home
Music
Contribute
Identify
Search

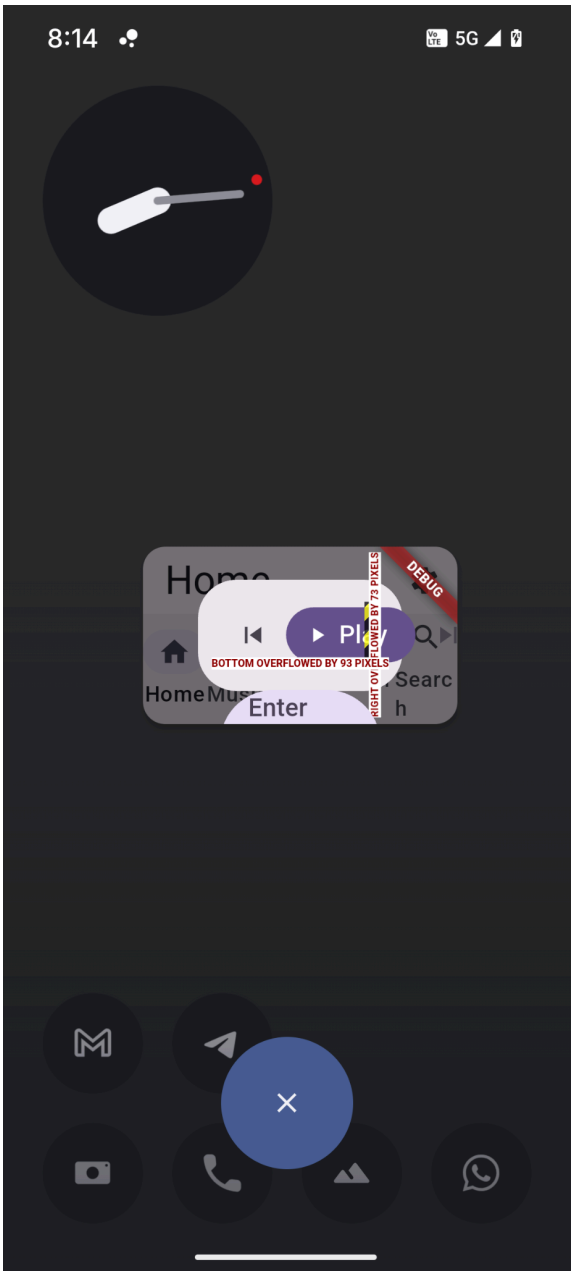

7:56
5G
DEBUG
Community
musixmatch
How Musixmatch community works
English


7:56
5G
DEBUG
Identify Song
Tap the button and play music
Home
Music
Contribute
Identify
Search


7:57
5G
DEBUG
Search
Search...
Home
Music
Contribute
Identify
Search


7:56
5G
DEBUG
Search
marshmello
Marshmello - Alone (Official Music Video)
Marshmello - Alone (Official Music Video) Stream/Download: https://marshmello.ffm.to/alone A lot of people ask me where Alone ...
Marshmello
Marshmello's Official YouTube Channel.
Marshmello & Anne-Marie - FRIENDS (Music Video) *OFFICIAL FRIENDZONE ANTHEM*
Marshmello & Anne-Marie - FRIENDS (Music Video) *OFFICIAL FRIENDZONE ANTHEM* Stream/Download: ...
Marshmello - Summer (Official Music Video) with
Home
Music
Contribute
Identify
Search


8:14
5G
DEBUG
Home
Distribute your lyrics
Distribute Your Lyrics
Play
Enter PiP Mode
Floating Lyrics
Trending on YouTube
Bhad Bhabie - OG
Home
Music
Contribute
Identify
Search


8:14
5G
DEBUG
BOTTOM OVERFLOWED BY 93 PIXELS
RIGHT OVERFLOWED BY 73 PIXELS
Enter

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,
              ),
          ),
```

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	59
Name	Jai Talreja
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. 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.

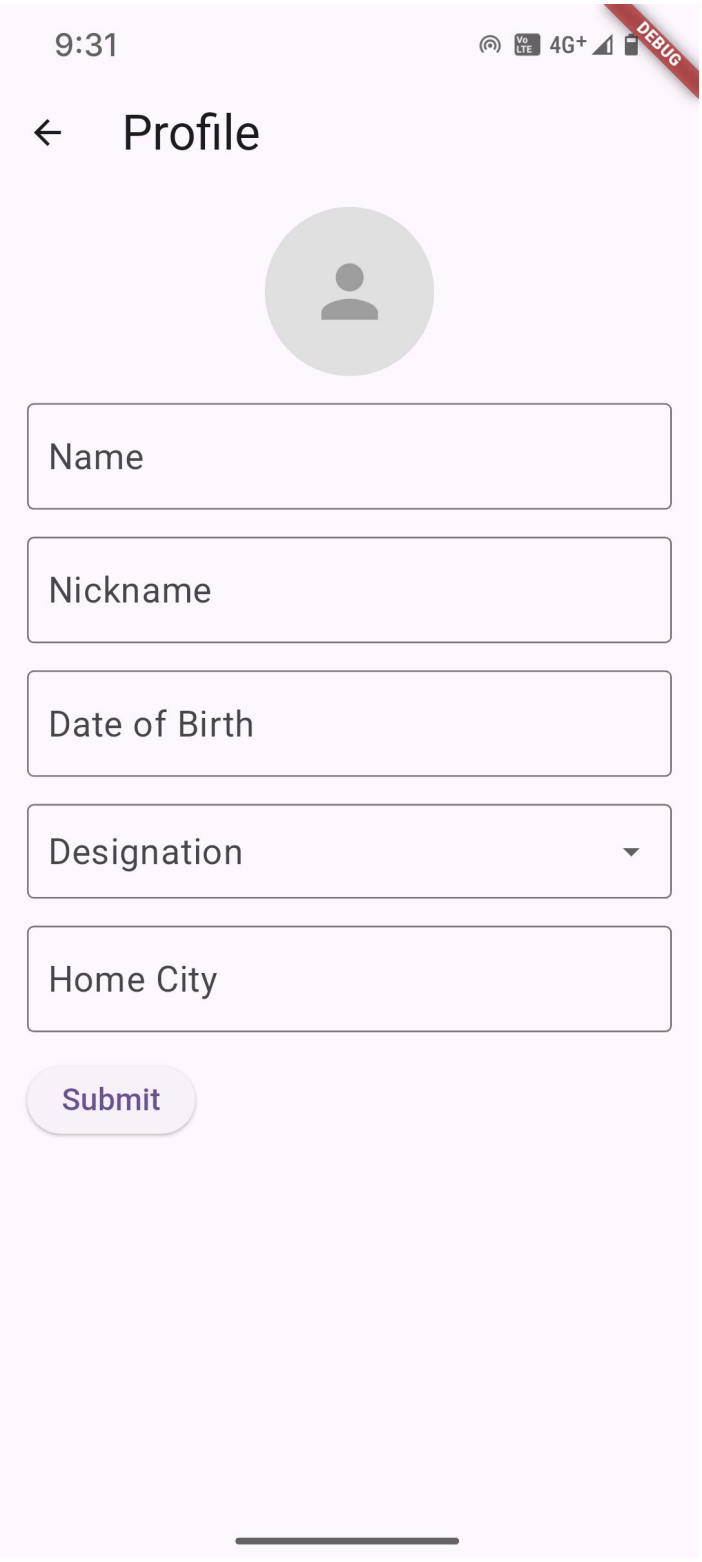
Form Workflow:

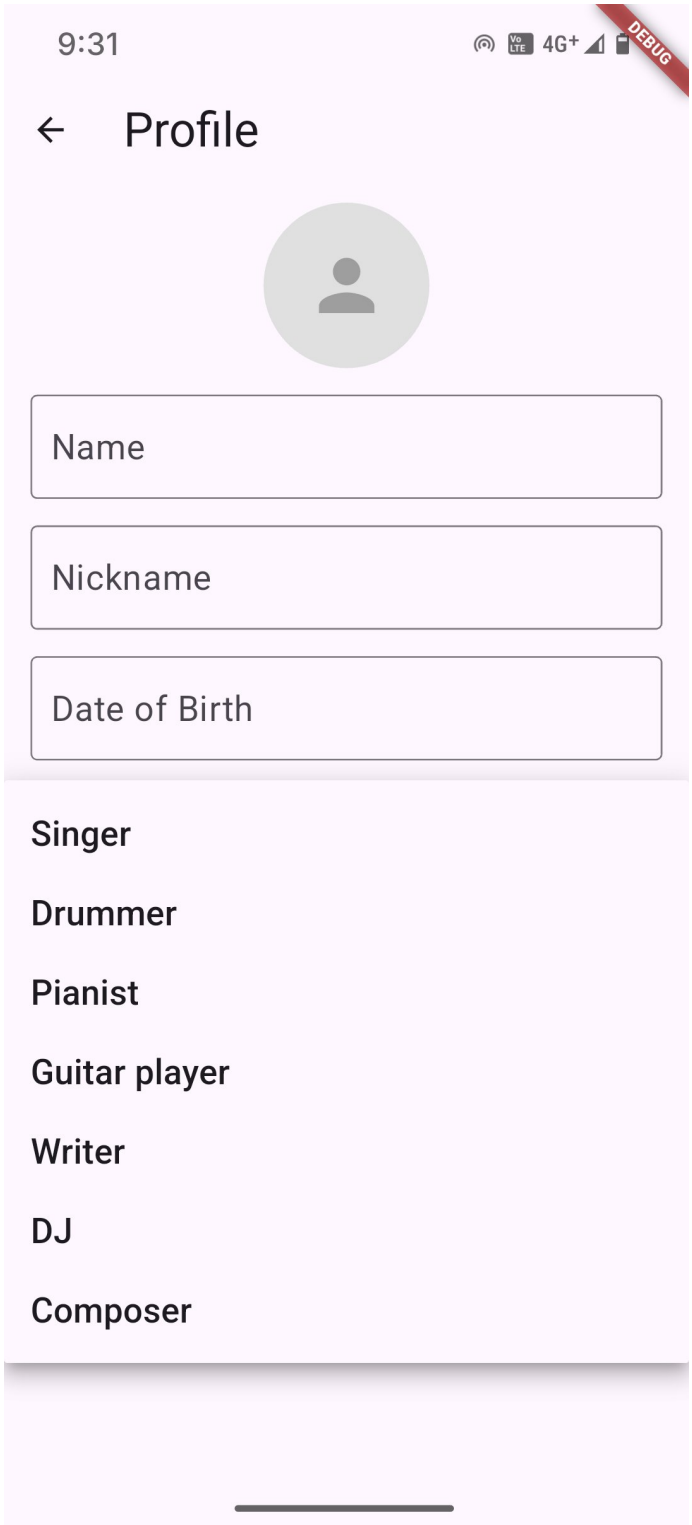
1. **Create a form**: Use the Form widget to wrap all input fields.
2. **Add form fields**: Use widgets like TextFormField to collect data.
3. **Set up validation**: Add validation logic to the form fields.

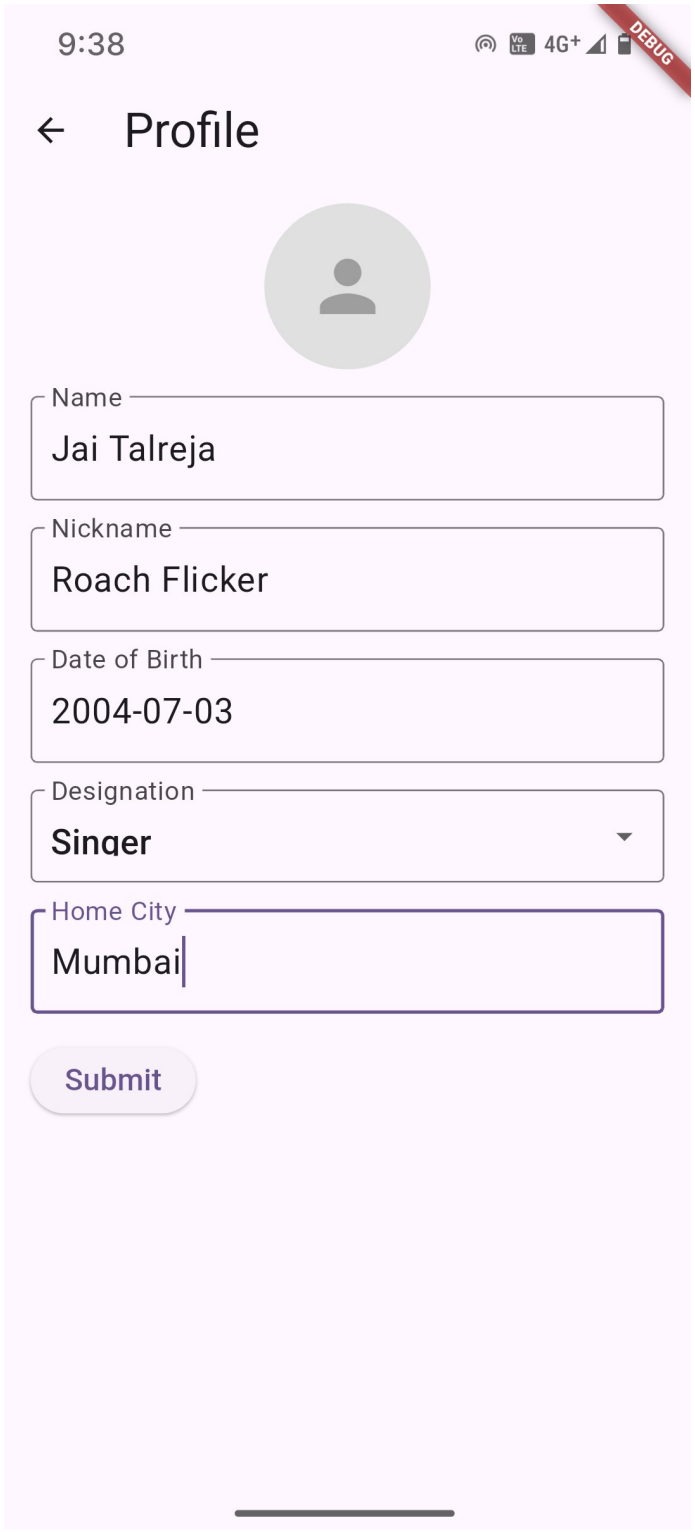
4. **Submit the form**: Trigger validation and handle the form submission.
5. **Save data**: Capture the entered data once validation passes.

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

Screenshots:




9:31
4G+
DEBUG
Profile
Name
Nickname
Date of Birth
Singer
Drummer
Pianist
Guitar player
Writer
DJ
Composer


9:38
4G+
DEBUG
Profile
Name
Jai Talreja
Nickname
Roach Flicker
Date of Birth
2004-07-03
Designation
Singer
Home City
Mumbai
Submit

Code Snippets:

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

class ProfilePage extends StatelessWidget {
  const ProfilePage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Profile'),
        leading: IconButton(
          icon: const Icon(Icons.arrow_back),
          onPressed: () => Navigator.pop(context),
        ),
      ),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            // Image Placeholder
            Center(
              child: Container(
                width: 100,
                height: 100,
                decoration: BoxDecoration(
                  color: Colors.grey[300],
                  borderRadius: BorderRadius.circular(50),
                ),
                child: const Icon(
                  Icons.person,
                  size: 50,
                  color: Colors.grey,
                ),
              ),
```

```
),
const SizedBox(height: 16),

// Name Field
TextField(
  decoration: InputDecoration(
    labelText: 'Name',
    border: OutlineInputBorder(),
  ),
),
const SizedBox(height: 16),

// Nickname Field
TextField(
  decoration: InputDecoration(
    labelText: 'Nickname',
    border: OutlineInputBorder(),
  ),
),
const SizedBox(height: 16),

// Date of Birth Field
TextField(
  decoration: InputDecoration(
    labelText: 'Date of Birth',
    border: OutlineInputBorder(),
    hintText: 'YYYY-MM-DD',
  ),
  keyboardType: TextInputType.datetime,
),
const SizedBox(height: 16),

// Designation Dropdown
DropdownButtonFormField<String>(
  decoration: InputDecoration(
    labelText: 'Designation',
    border: OutlineInputBorder(),
```

```
            ),
            items: const [
              DropdownMenuItem(value: 'Singer', child: Text('Singer')),
              DropdownMenuItem(value: 'Drummer', child: Text('Drummer')),
              DropdownMenuItem(value: 'Pianist', child: Text('Pianist')),
              DropdownMenuItem(value: 'Guitar player', child: Text('Guitar player')),
              DropdownMenuItem(value: 'Writer', child: Text('Writer')),
              DropdownMenuItem(value: 'DJ', child: Text('DJ')),
              DropdownMenuItem(value: 'Composer', child: Text('Composer')),
            ],
            onChanged: (value) {},
          ),
          const SizedBox(height: 16),

          // Home City Field
          TextField(
            decoration: InputDecoration(
              labelText: 'Home City',
              border: OutlineInputBorder(),
            ),
          ),
          const SizedBox(height: 16),

          // Submit Button
          ElevatedButton(
            onPressed: () {
              // Handle form submission
            },
            child: const Text('Submit'),
          ),
        ],
      ),
    ),
  );
 }
}
```

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	59
Name	Jai Talreja
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.

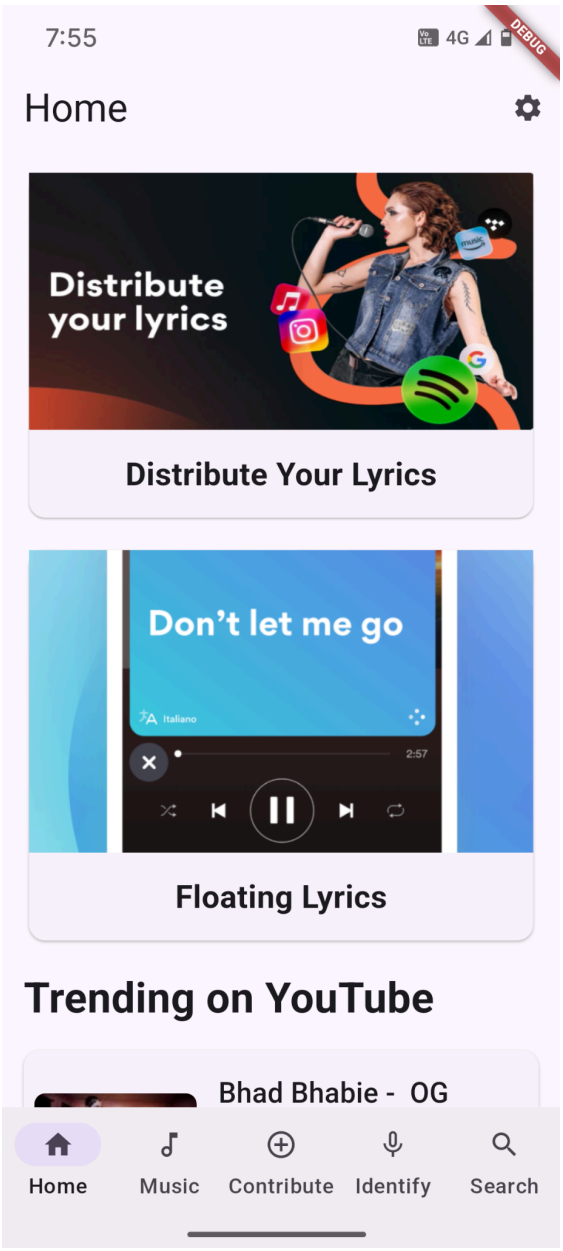
3. Managing Navigation and Gestures Together

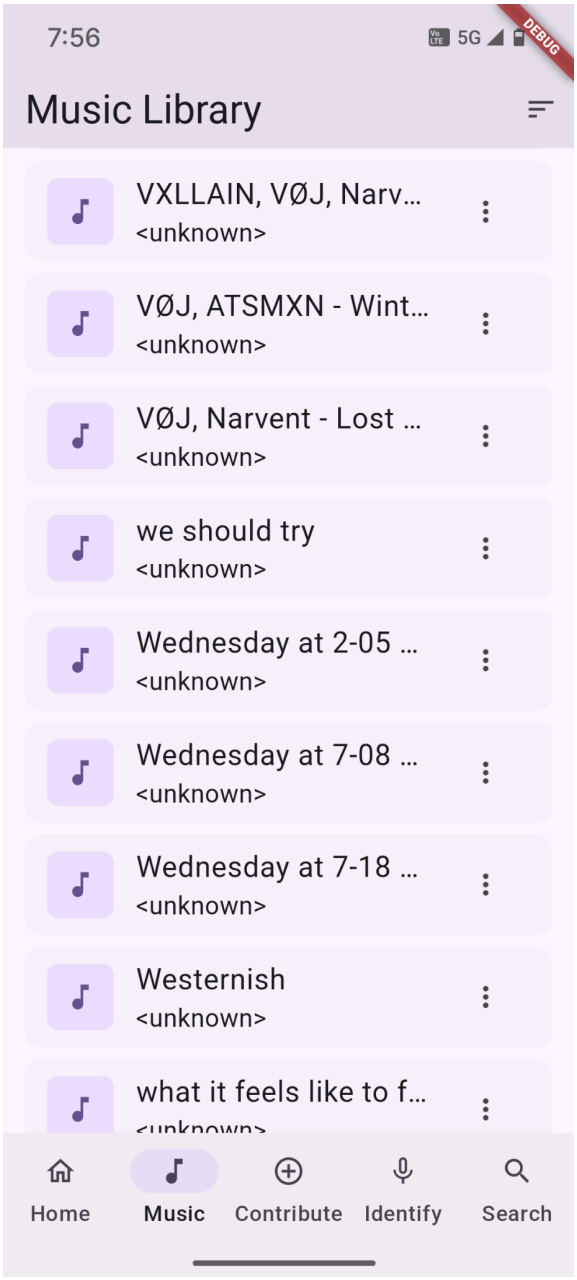
When you combine navigation with gestures, you can create more interactive and dynamic UIs. For instance, a user could swipe to navigate between screens, or tap a button that triggers navigation while performing a gesture on a different part of the screen.

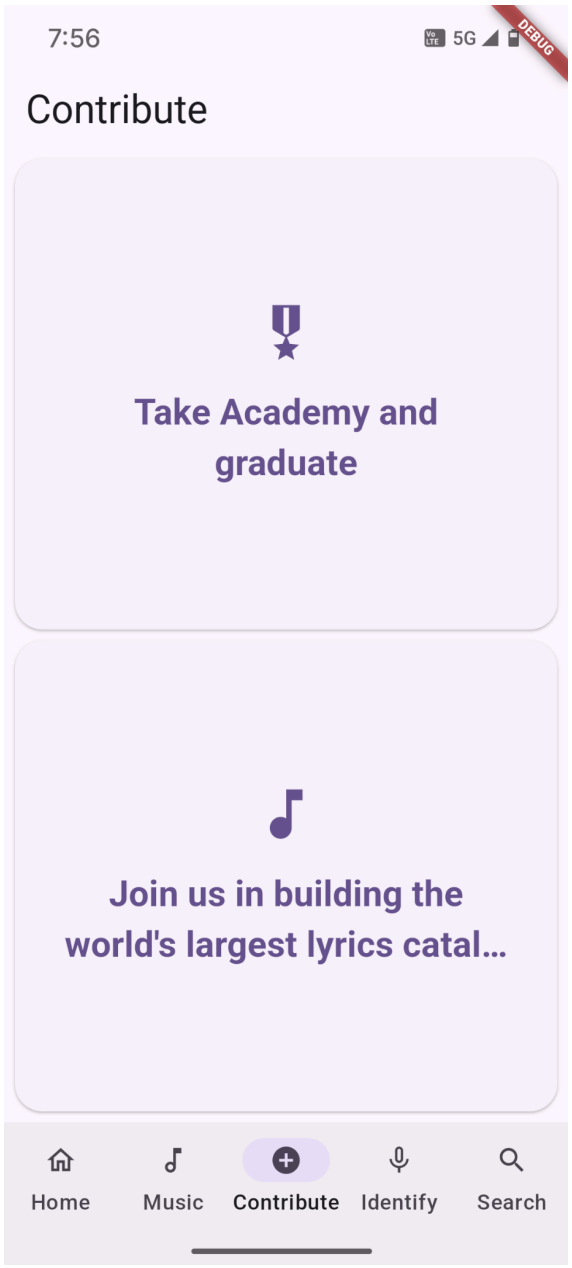
4. Back Button Handling

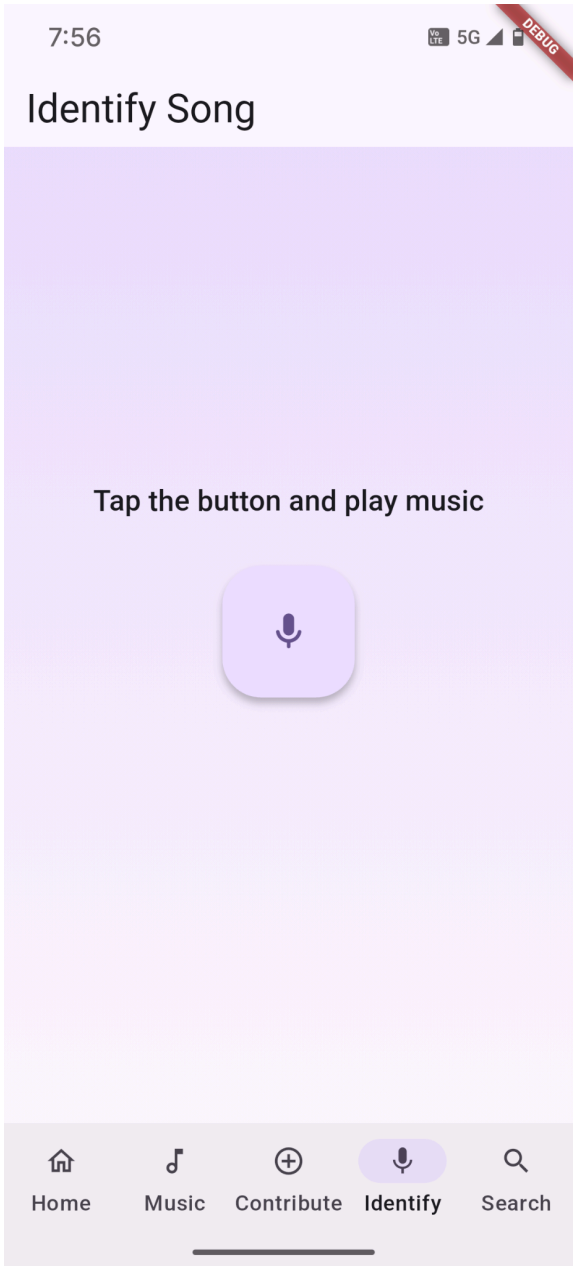
On Android devices, there is a system-wide back button that users can press to navigate backward. Flutter provides a way to intercept and customize this behavior using WillPopScope, allowing you to decide what happens when the user tries to go back (e.g., prevent the user from leaving the current screen,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

Screenshots:

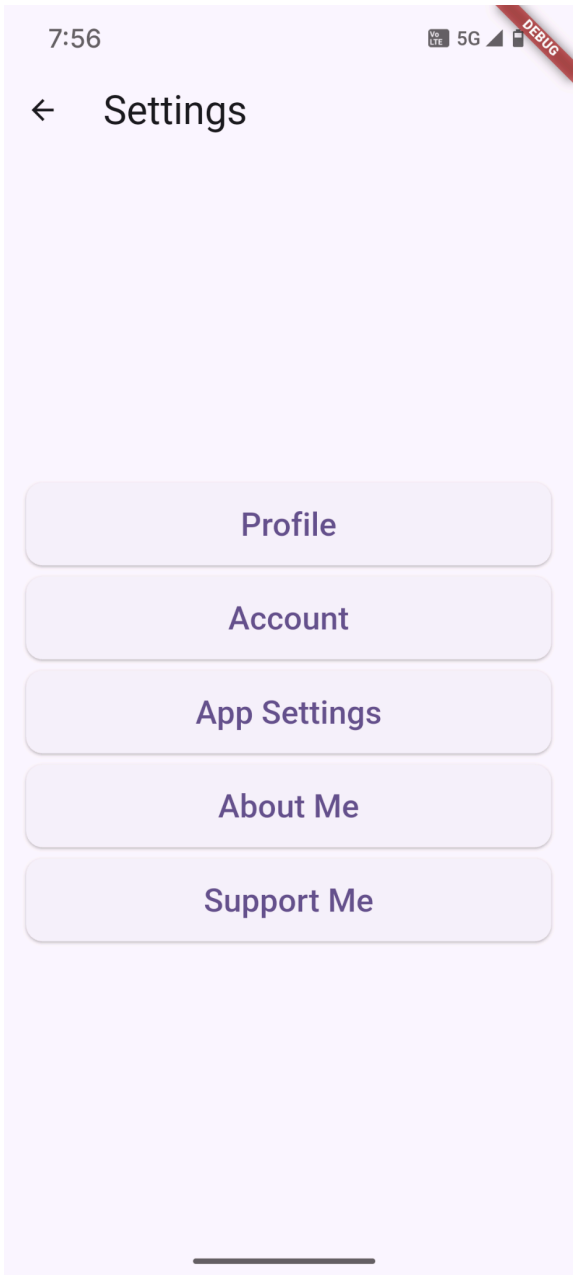


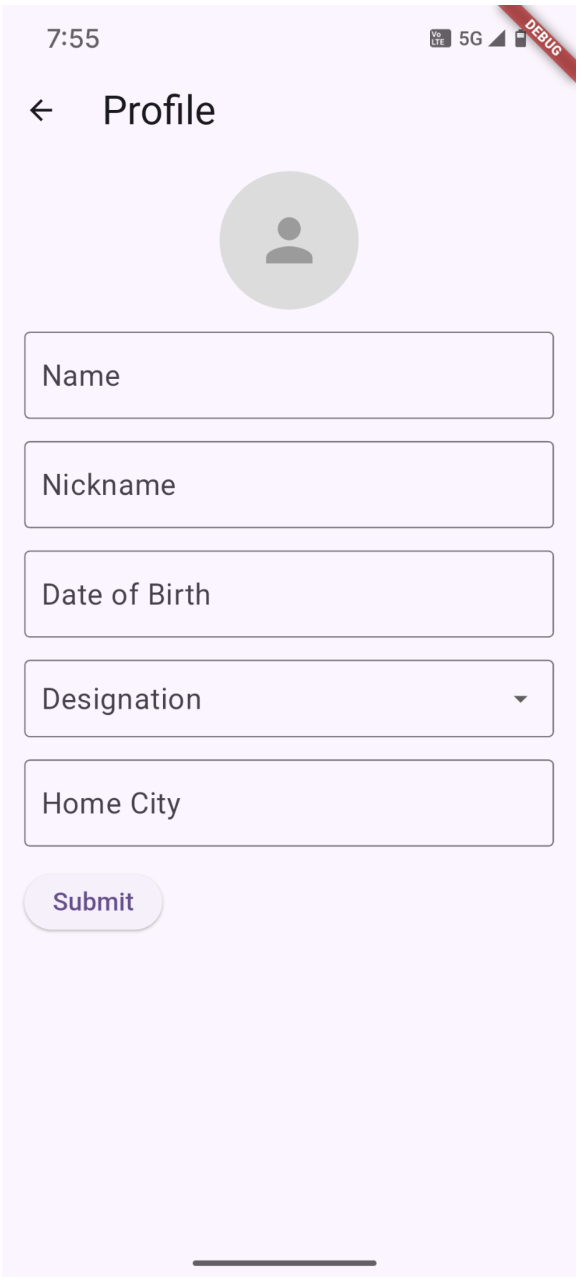


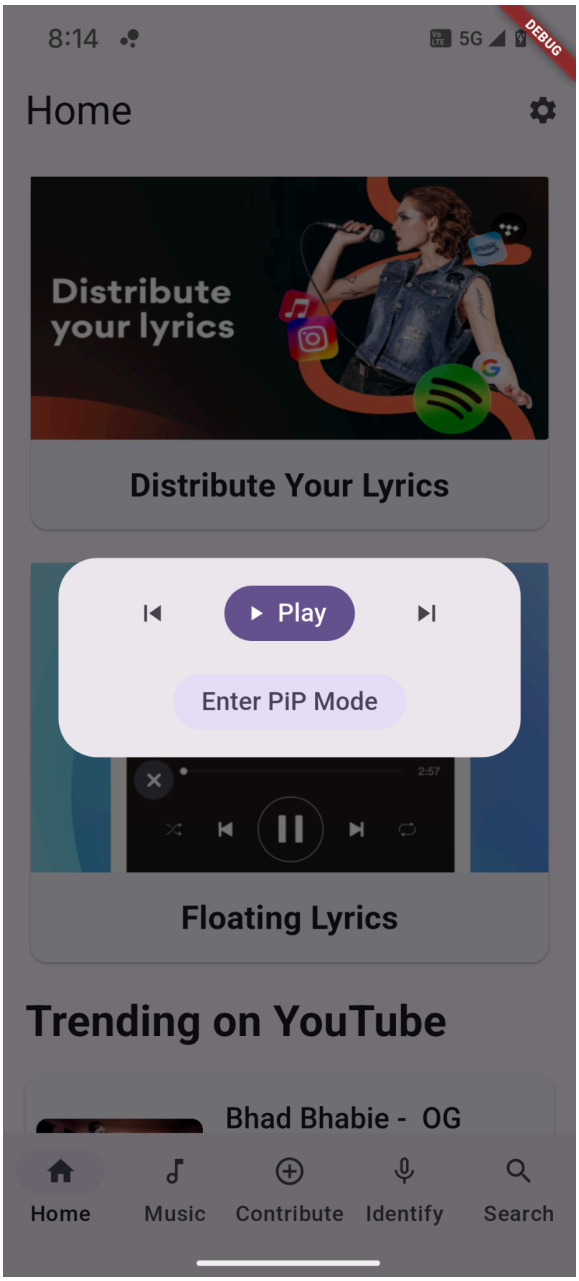

7:56
5G
DEBUG
Contribute
Take Academy and graduate
Join us in building the world's largest lyrics catal...
Home
Music
Contribute
Identify
Search

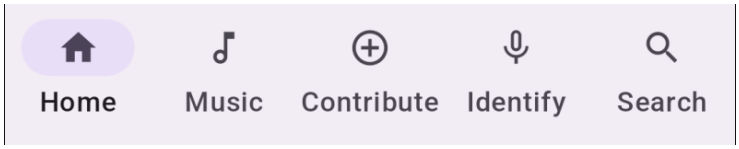

7:56
5G
DEBUG
Identify Song
Tap the button and play music
Home
Music
Contribute
Identify
Search


7:57
5G
DEBUG
Search
Search...
Home
Music
Contribute
Identify
Search


7:56
5G
DEBUG
Settings
Profile
Account
App Settings
About Me
Support Me


7:55
5G
DEBUG
Profile
Name
Nickname
Date of Birth
Designation
Home City
Submit


8:14
5G
DEBUG
Home
Distribute your lyrics
Distribute Your Lyrics
Play
Enter PiP Mode
Floating Lyrics
Trending on YouTube
Bhad Bhabie - OG
Home
Music
Contribute
Identify
Search


Home
Music
Contribute
Identify
Search

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

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	59
Name	Jai Talreja
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Jai Talreja 59 D15A

Exp 6

Aim :- To connect Flutter UI with Firebase Database

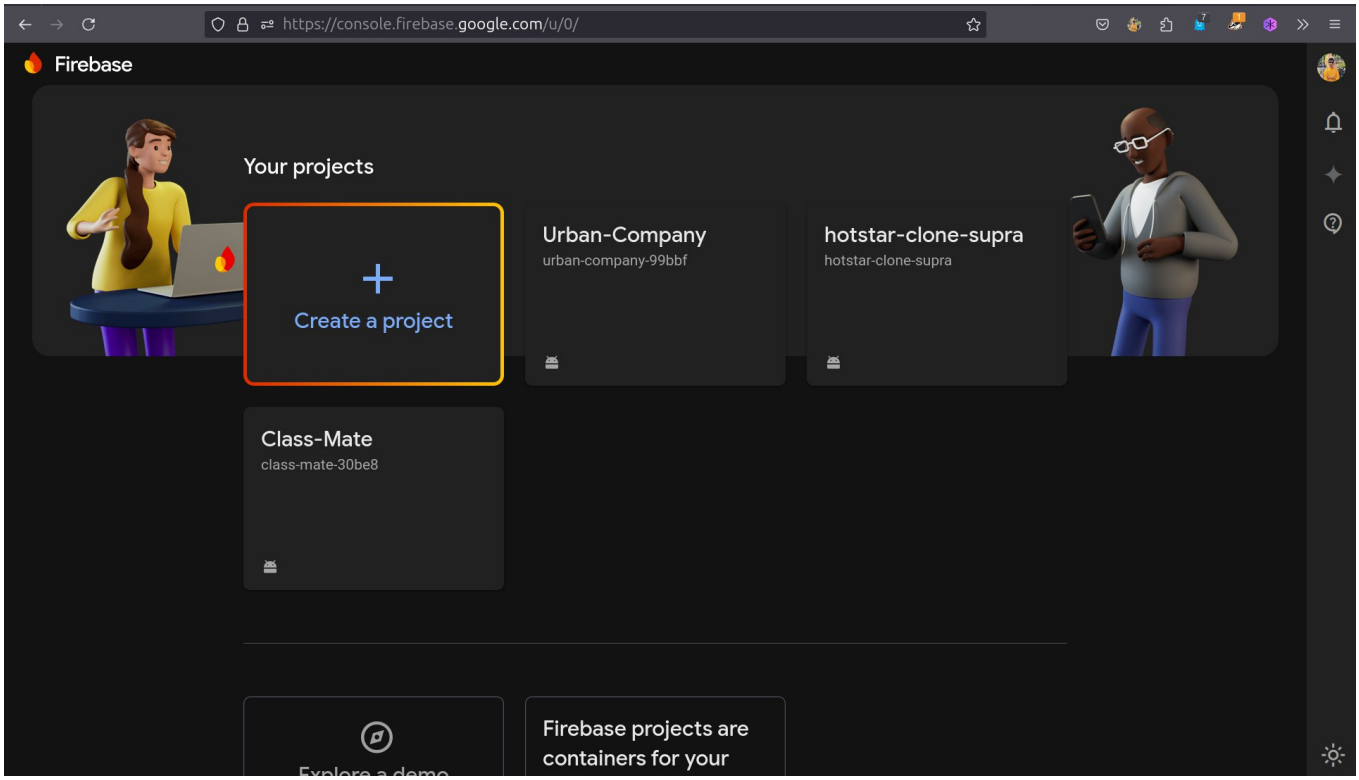
Theory :-

Flutter is an open-source UI toolkit developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. Firebase, a Backend-as-a-Service (BaaS) platform, provides real-time database, authentication, and cloud storage services, making it a powerful backend solution for Flutter applications.

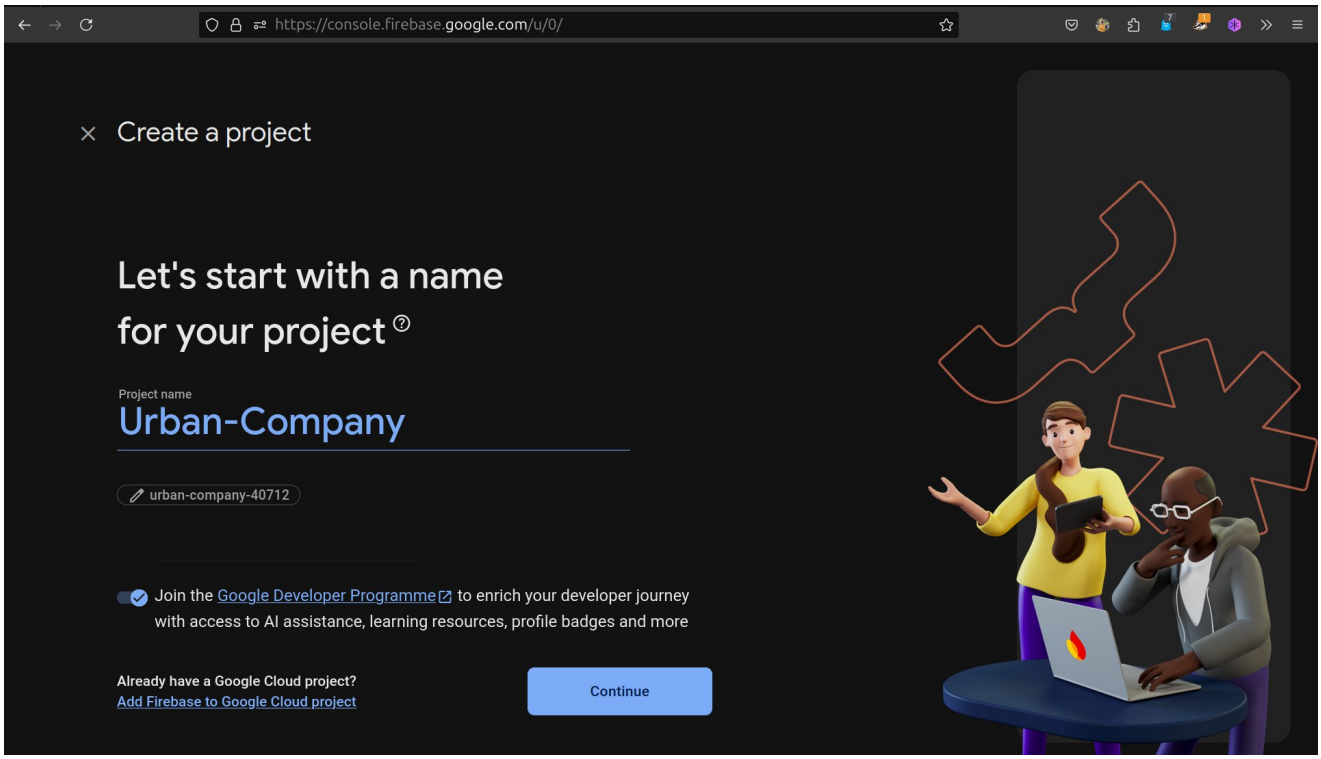
By integrating Firebase with Flutter, developers can store and retrieve data in real time, authenticate users, and manage cloud-based data efficiently. This is particularly useful for applications requiring dynamic content updates and user interactions.

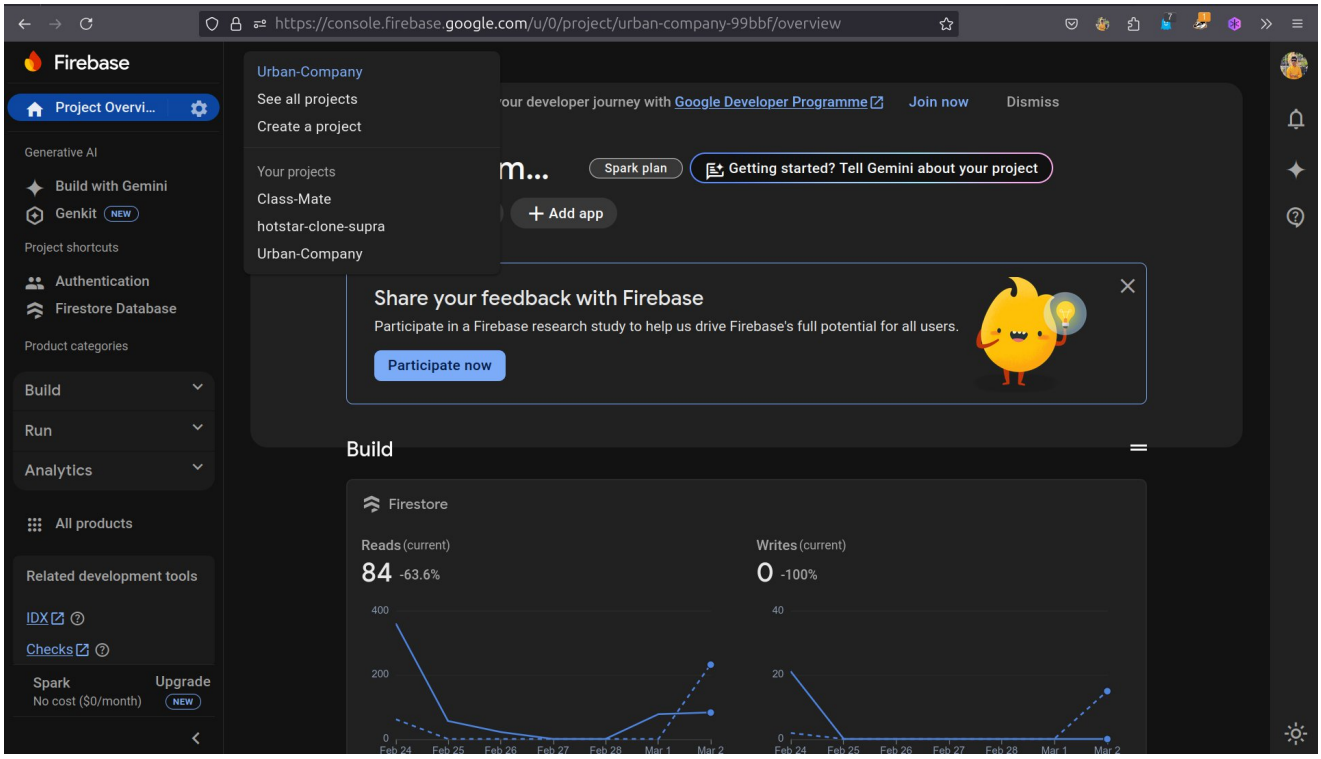
Practical :-

1. **Creating new app in Firebase console**

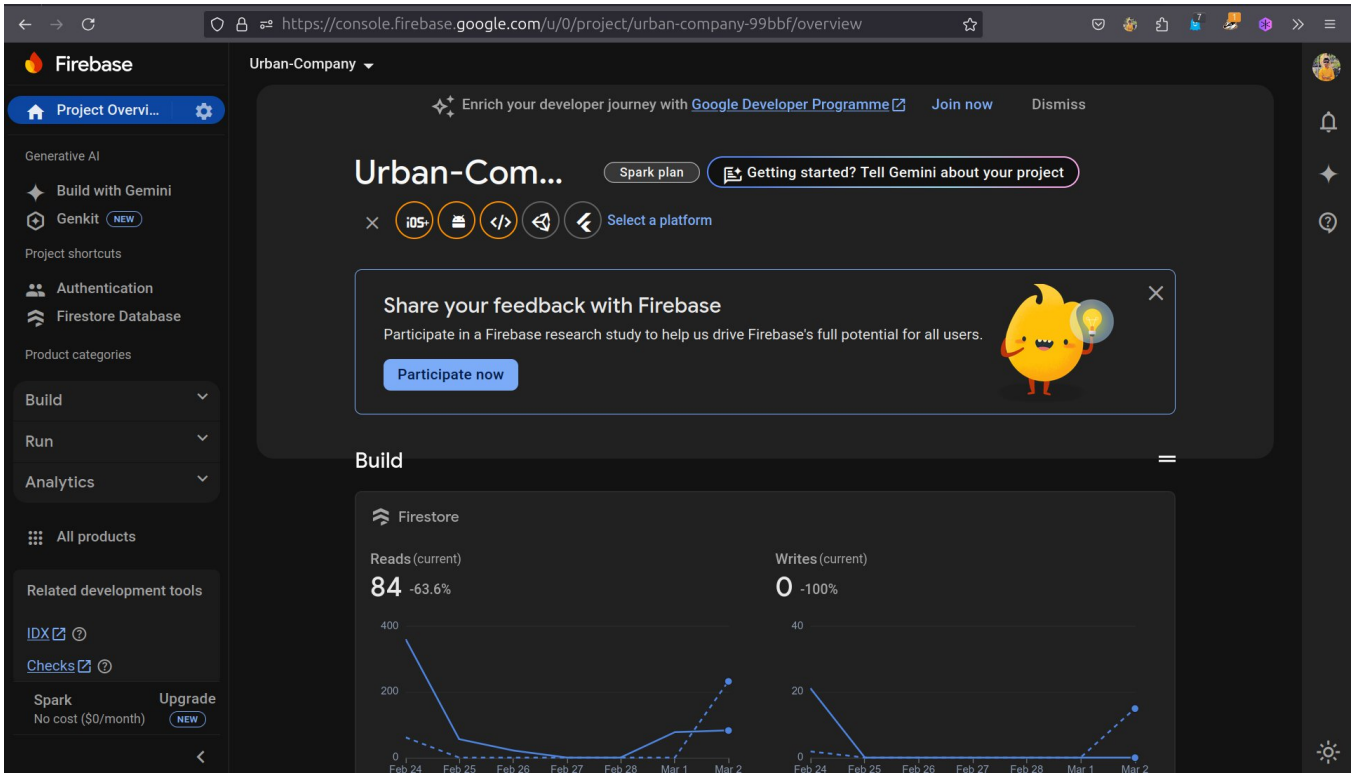


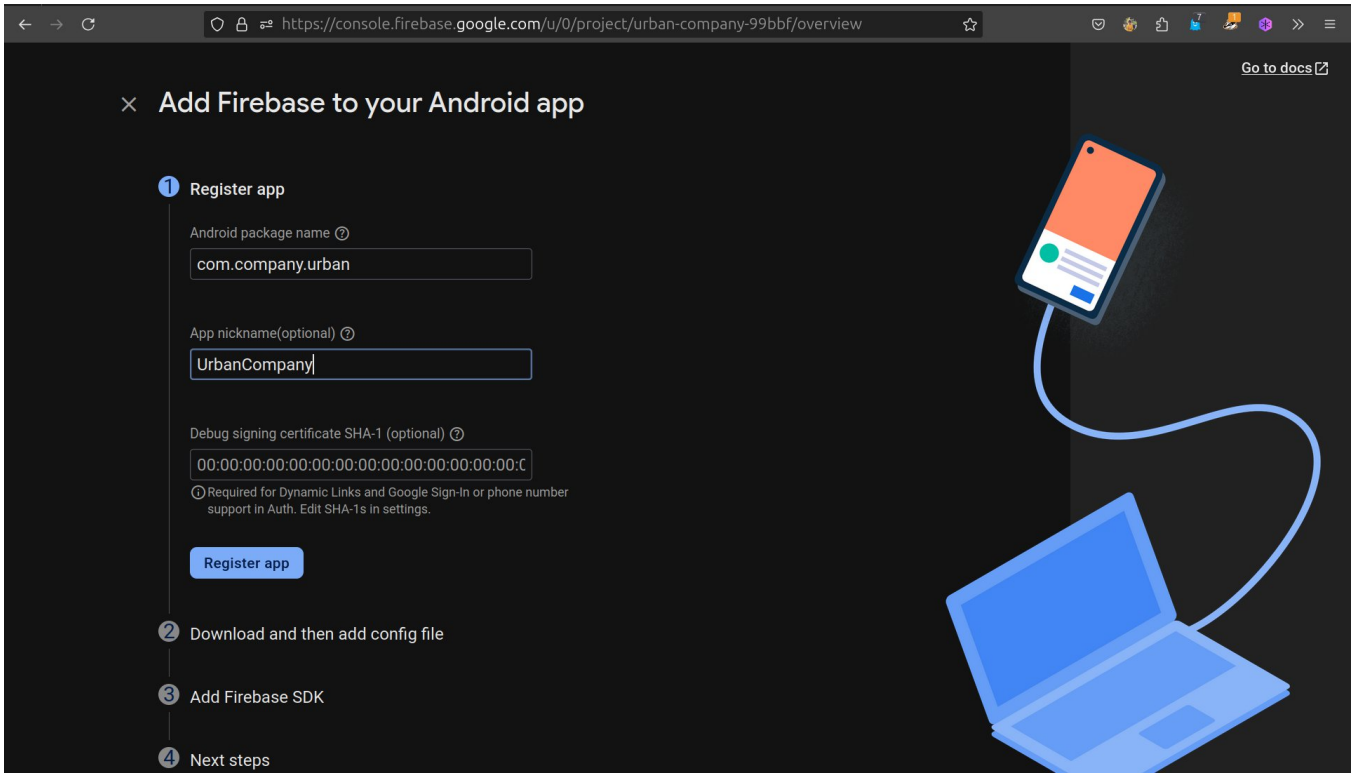
Jai Talreja 59 D15A



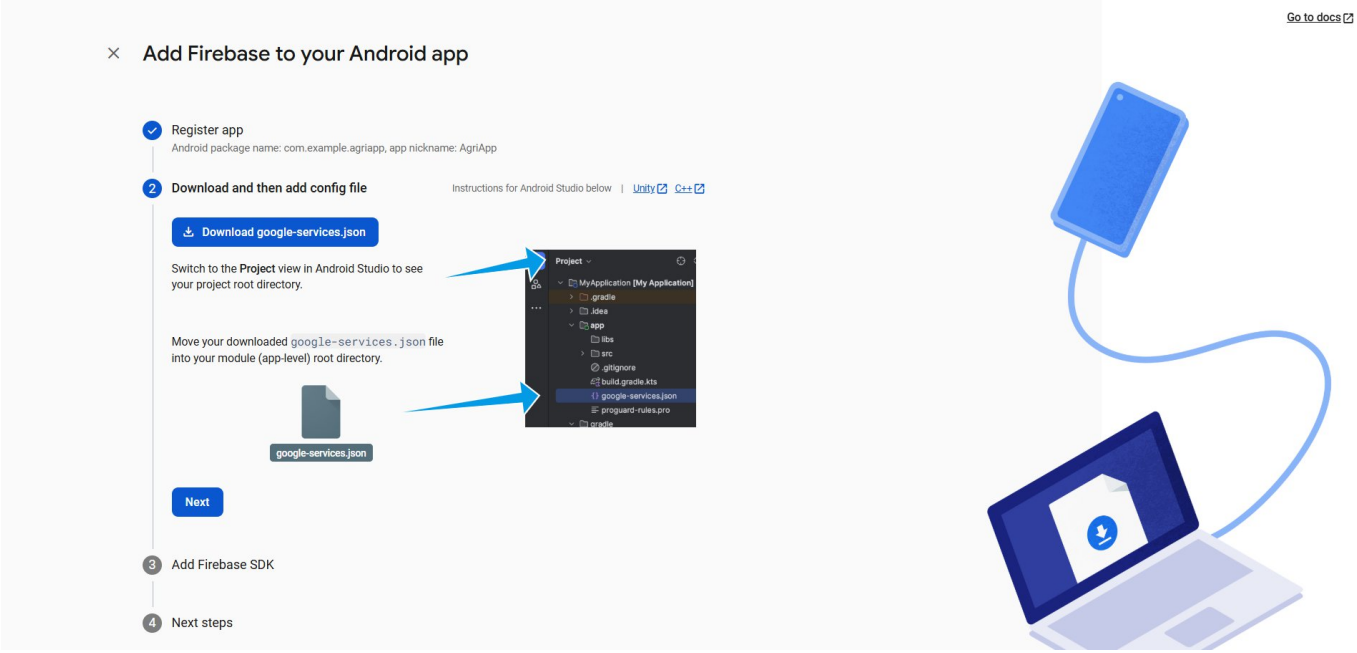


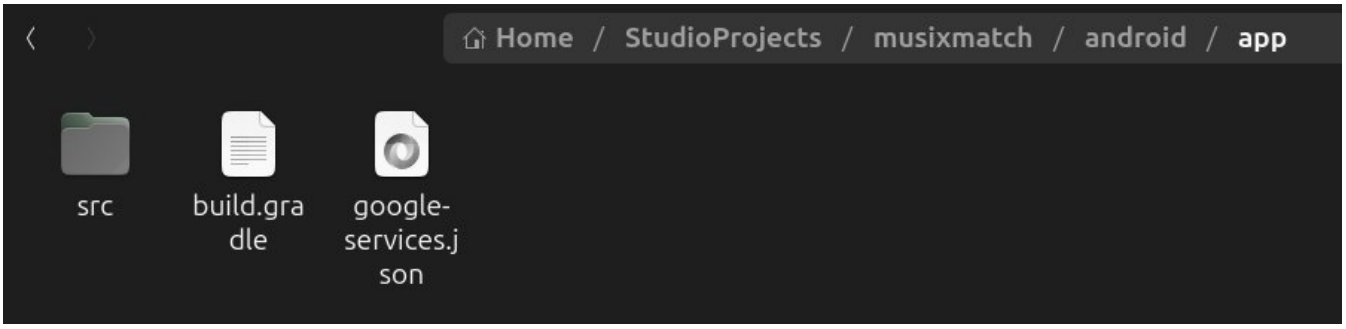
2. Adding Firebase dependencies and SDK to Flutter app





Jai Talreja 59 D15A

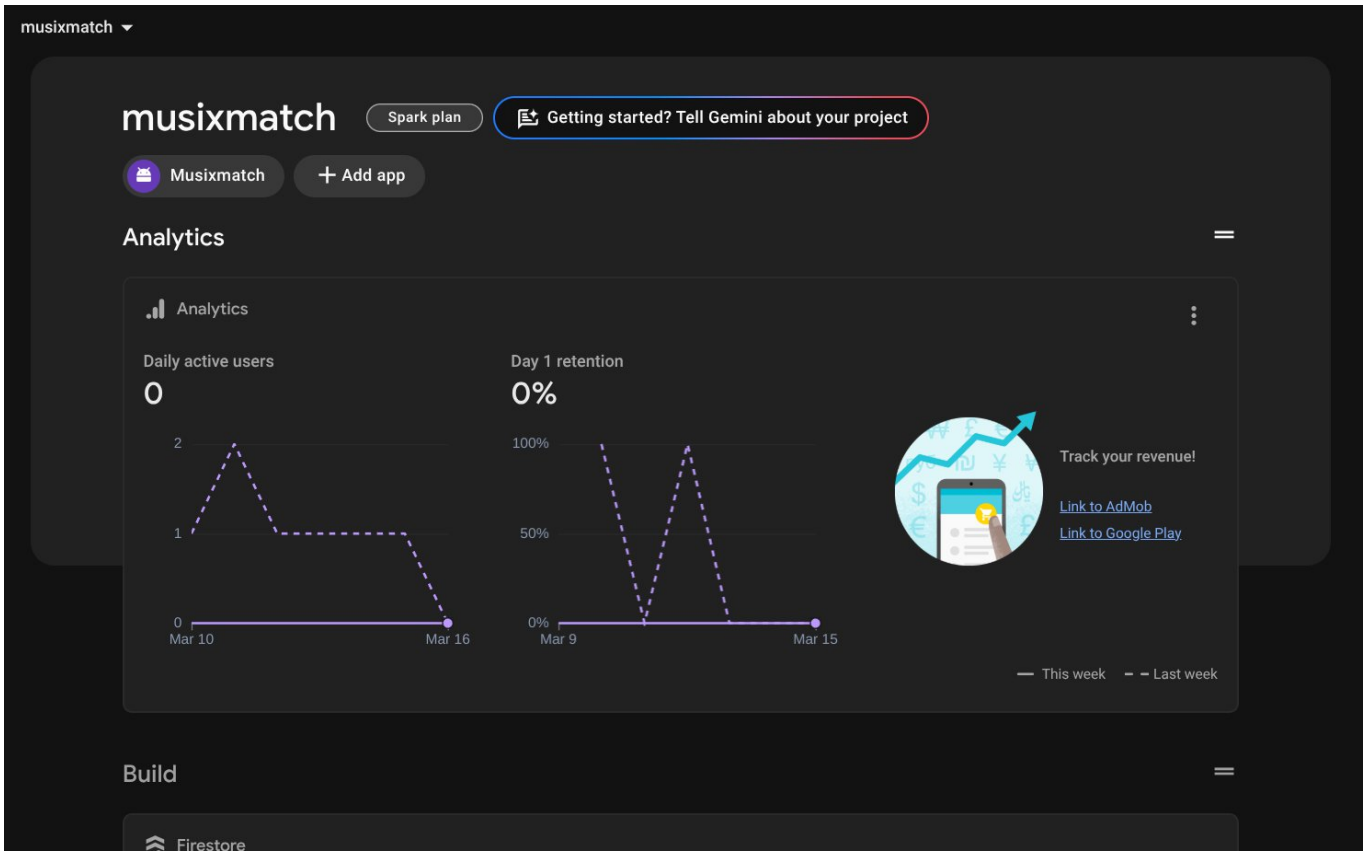




Open

build.gradle
~/StudioProjects/musixmatch/android/app

```
plugins {
    id "com.android.application"
    id "kotlin-android"
    // The Flutter Gradle Plugin must be applied after the Android and Kotlin Gradle plugins.
    id "dev.flutter.flutter-gradle-plugin"
    id 'com.google.gms.google-services'
}

dependencies
    {
        implementation platform('com.google.firebase:firebase-bom:33.10.0')
        implementation 'com.google.firebase:firebase-analytics'
    }

android {
    namespace "com.example.musixmatch"
    compileSdkVersion flutter.compileSdkVersion
    ndkVersion flutter.ndkVersion
```



3. Configuring Firebase real-time Database

main.dart :-

```
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'navigation_bar.dart'; // Update this import

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Musixmatch',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const MainScreen(), // Change HomePage to MainScreen
    );
```

```
  }
}
```

Profile_page.dart :-

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

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

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

class _ProfilePageState extends State<ProfilePage> {
  // Controllers to capture form input
  final TextEditingController nameController = TextEditingController();
  final TextEditingController nicknameController = TextEditingController();
  final TextEditingController dobController = TextEditingController();
  final TextEditingController cityController = TextEditingController();
  String? designation;

  // Submit form data to Firestore
  Future<void> submitForm() async {
    // Gather all input data
    final profileData = {
```

```
      'name': nameController.text,
      'nickname': nicknameController.text,
      'dob': dobController.text,
      'designation': designation,
      'city': cityController.text,
    };

    try {
      // Save data to Firestore under the 'profiles' collection
      await FirebaseFirestore.instance.collection('profiles').add(profileData);

      // Show a confirmation message
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Profile saved successfully')),
      );
    } catch (e) {
      // Handle error (e.g., show an error message)
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Error saving profile: $e')),
      );
    }
  }

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

```
      appBar: AppBar(
        title: const Text('Profile'),
        leading: IconButton(
          icon: const Icon(Icons.arrow_back),
          onPressed: () => Navigator.pop(context),
        ),
      ),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            // Image Placeholder
            Center(
              child: Container(
                width: 100,
                height: 100,
                decoration: BoxDecoration(
                  color: Colors.grey[300],
                  borderRadius: BorderRadius.circular(50),
                ),
                child: const Icon(
                  Icons.person,
                  size: 50,
                  color: Colors.grey,
                ),
```

```
      ),
    ),
    const SizedBox(height: 16),

    // Name Field
    TextField(
      controller: nameController,
      decoration: const InputDecoration(
        labelText: 'Name',
        border: OutlineInputBorder(),
      ),
    ),
    const SizedBox(height: 16),

    // Nickname Field
    TextField(
      controller: nicknameController,
      decoration: const InputDecoration(
        labelText: 'Nickname',
        border: OutlineInputBorder(),
      ),
    ),
    const SizedBox(height: 16),

    // Date of Birth Field
    TextField(
```

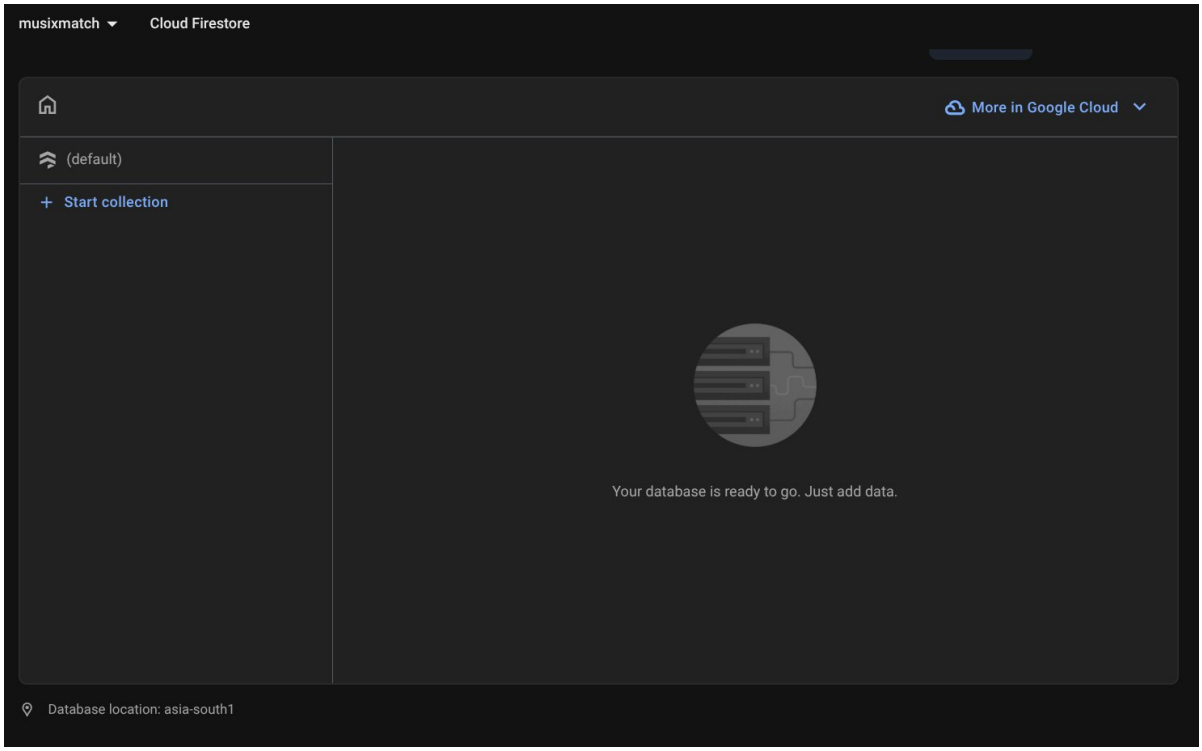
```
            controller: dobController,
            decoration: const InputDecoration(
              labelText: 'Date of Birth',
              border: OutlineInputBorder(),
              hintText: 'YYYY-MM-DD',
            ),
            keyboardType: TextInputType.datetime,
          ),
          const SizedBox(height: 16),

          // Designation Dropdown
          DropdownButtonFormField<String>(
            value: designation,
            decoration: const InputDecoration(
              labelText: 'Designation',
              border: OutlineInputBorder(),
            ),
            items: const [
              DropdownMenuItem(value: 'Singer', child: Text('Singer')),
              DropdownMenuItem(value: 'Drummer', child: Text('Drummer')),
              DropdownMenuItem(value: 'Pianist', child: Text('Pianist')),
              DropdownMenuItem(value: 'Guitar player', child: Text('Guitar player')),
              DropdownMenuItem(value: 'Writer', child: Text('Writer')),
              DropdownMenuItem(value: 'DJ', child: Text('DJ')),
              DropdownMenuItem(value: 'Composer', child: Text('Composer')),
            ],
```

```
            onChanged: (value) {
              setState(() {
                designation = value;
              });
            },
          ),
          const SizedBox(height: 16),

          // Home City Field
          TextField(
            controller: cityController,
            decoration: const InputDecoration(
              labelText: 'Home City',
              border: OutlineInputBorder(),
            ),
          ),
          const SizedBox(height: 16),

          // Submit Button
          ElevatedButton(
            onPressed: submitForm,
            child: const Text('Submit'),
          ),
        ],
      ),
    ),
```

```
    );
  }
}
```

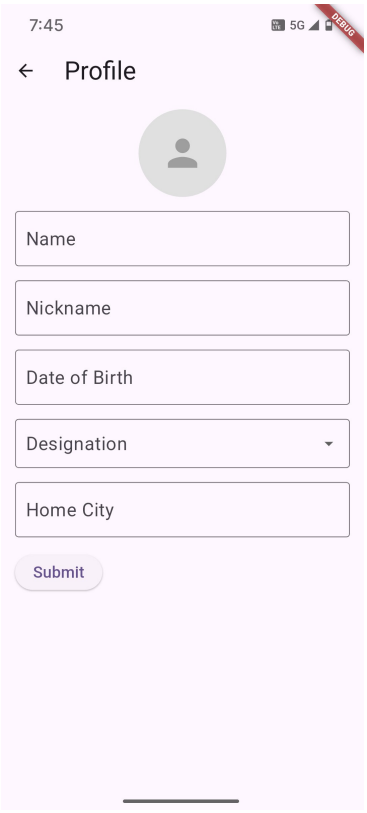
4. Final result

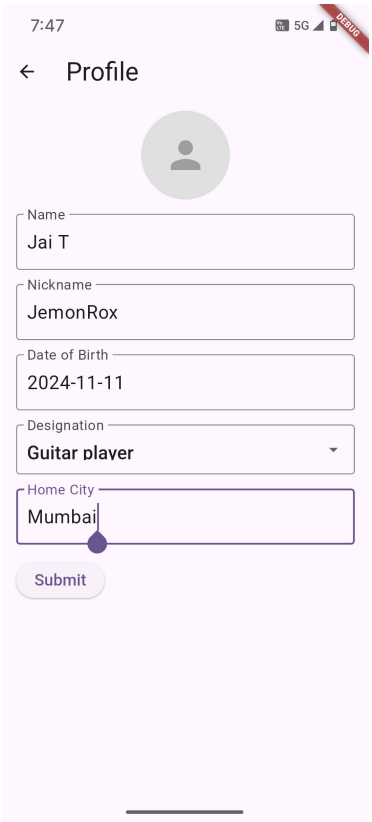
Before filling data in the app

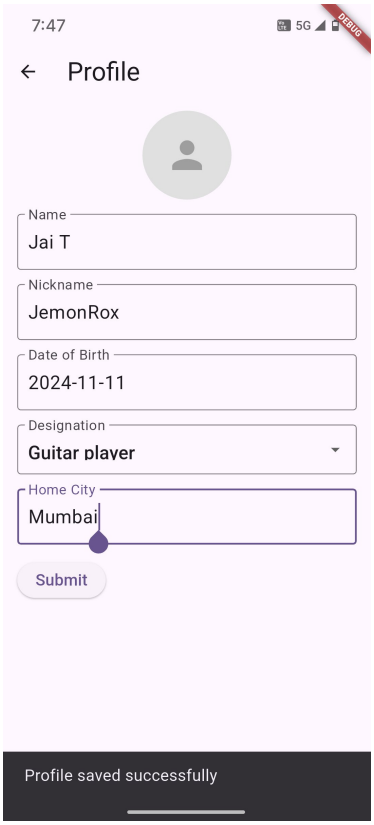


Jai Talreja 59 D15A

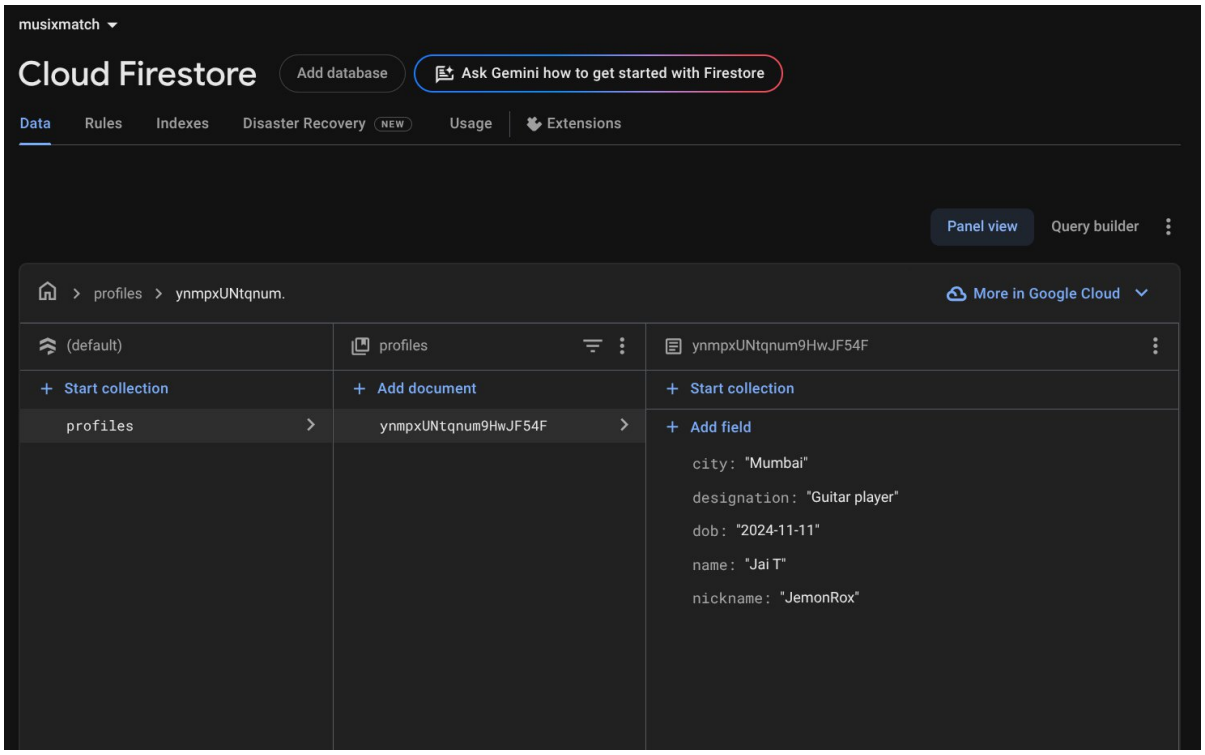
Filling data in the app







After filling data in the app



Project Title: **Roll No.**

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.	59
Name	Jai Talreja
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

Experiment No. 7

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:**

  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.

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.

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.

- **The main features are:**

1. Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.
2. Responsive — They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can later become available.
3. Updated — Information is always up-to-date thanks to the data update process offered by service workers.
4. Secure — Exposed over HTTPS protocol to prevent the connection from displaying information or altering the contents.
5. Searchable — They are identified as “applications” and are indexed by search engines.
6. Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
7. 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.
8. Linkable — Easily shared via URL without complex installations.
9. 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.

Code:

1. **Manifest.json**

```
{
    "name": "Simple PWA",
    "short_name": "PWA",
    "start_url": "index.html",
    "display": "standalone",
    "background_color": "#ffffff",
    "theme_color": "#000000",
    "description": "A simple Progressive Web App that explains what a PWA is.",
    "icons": [
        {
            "src": "pwa-banner.png",
            "sizes": "192x192",
            "type": "image/png"
        }
    ]
}
```

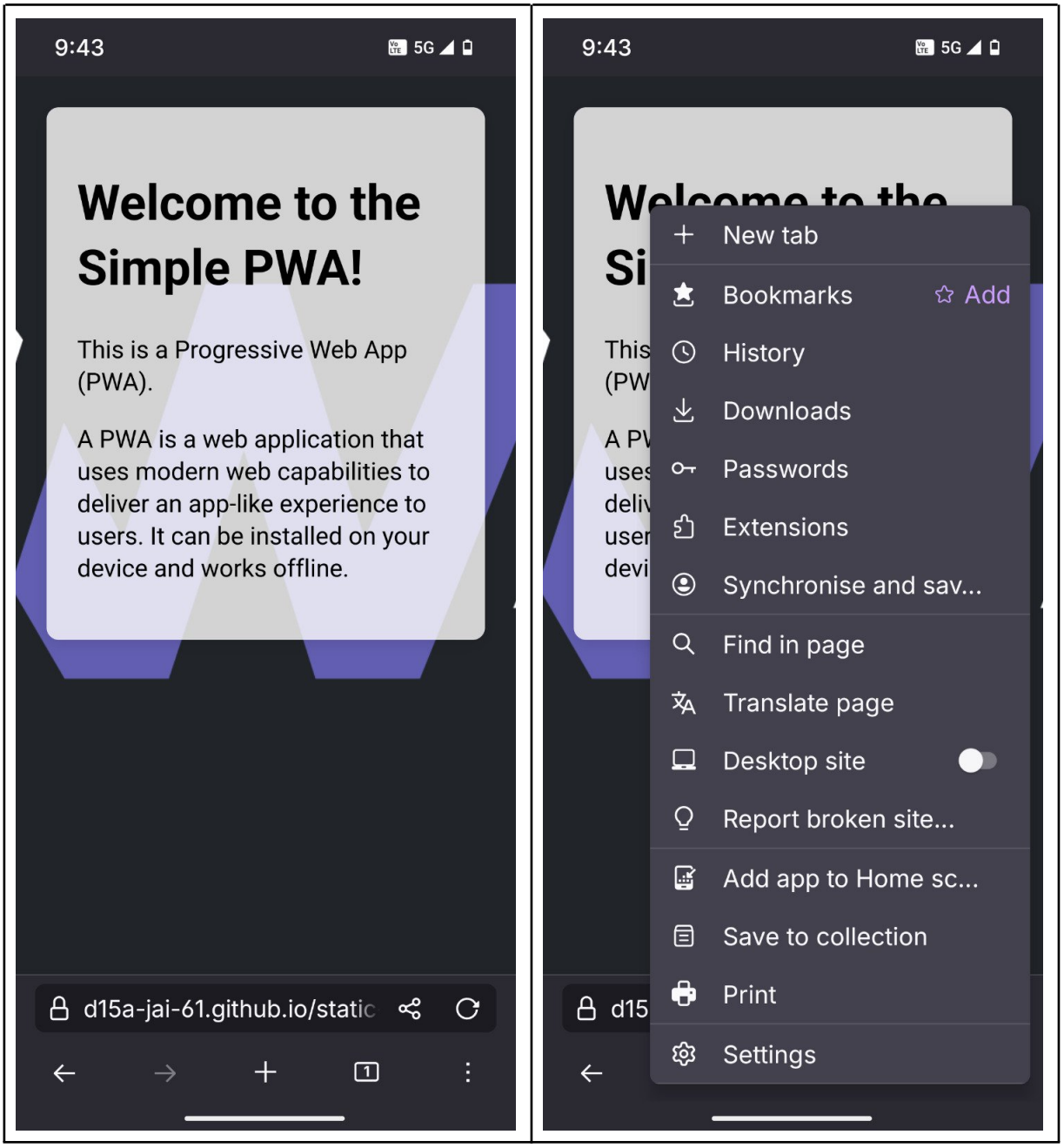
2. **Service-worker.js**

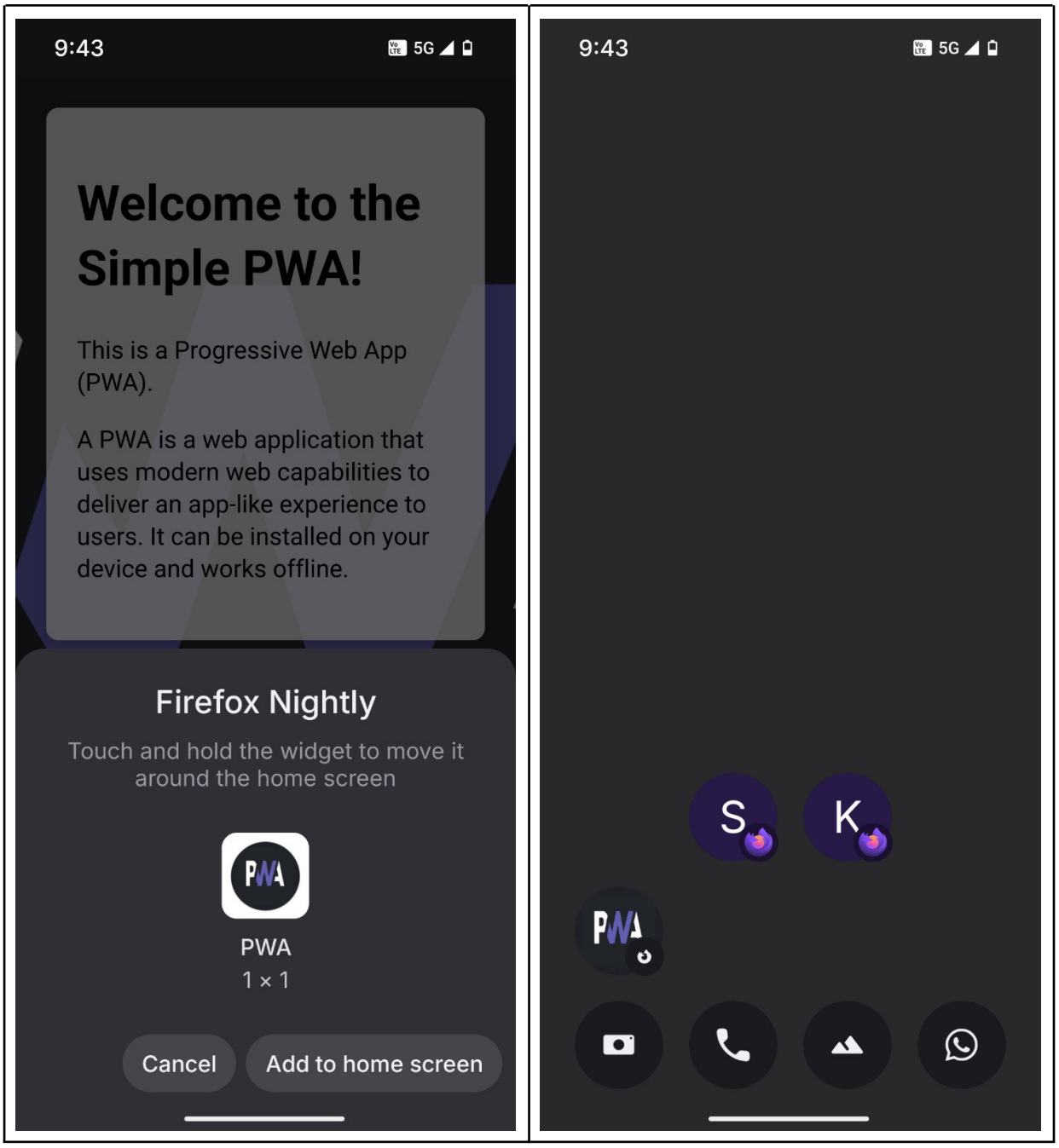
```
const CACHE_NAME = 'simple-pwa-cache-v1';
const urlsToCache = [
    'index.html',
    'offline-form.html',
    'styles.css',
    'manifest.json',
    'pwa-banner.png',
];

// Install the service worker and cache assets
self.addEventListener('install', (event) => {
    event.waitUntil(
        caches.open(CACHE_NAME).then((cache) => {
            console.log('[Service Worker] Caching essential assets');
            return cache.addAll(urlsToCache);
        })
    );
});

// Fetch event - serve cached assets or fetch from network
self.addEventListener('fetch', (event) => {
    event.respondWith(
        caches.match(event.request).then((response) => {
            if (response) {
                console.log('[Service Worker] Returning cached resource:',
```

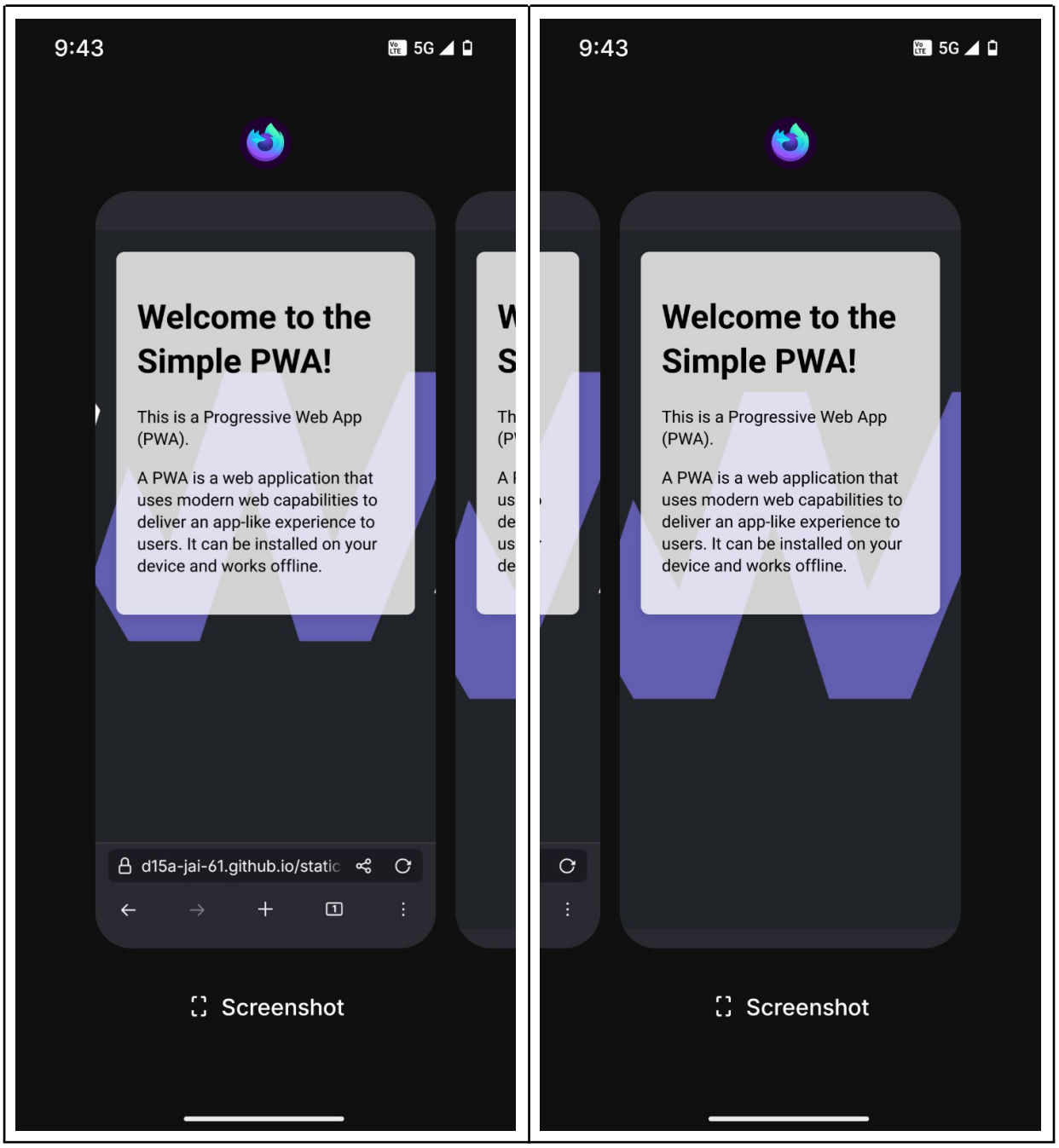
```
event.request.url);
                return response;
            }
            console.log('[Service Worker] Fetching from network:', event.request.url);
            return fetch(event.request);
        })
    );
});

// Sync event - retry offline form submissions
self.addEventListener('sync', (event) => {
    if (event.tag === 'sync-form') {
        console.log('[Service Worker] Sync event triggered: Submitting offline form
data...');
        event.waitUntil(syncFormData());
    }
});

function syncFormData() {
    return getFormDataFromIndexedDB().then((formData) => {
        if (formData) {
            console.log('[Service Worker] Syncing form data...');
            console.log('[Service Worker] Form Data:', formData);

            // Instead of sending to an API, let's store it back in IndexedDB or localStorage
for now
            saveFormDataLocally(formData);

            // Clear the form data from IndexedDB after "syncing"
            clearFormDataFromIndexedDB();
        }
    });
}

function getFormDataFromIndexedDB() {
    return new Promise((resolve, reject) => {
        const request = indexedDB.open('offlineFormData', 1);
        request.onsuccess = function() {
            const db = request.result;
            const tx = db.transaction('formData', 'readonly');
            const store = tx.objectStore('formData');
            const data = store.getAll();  // Get all stored form data
            data.onsuccess = function() {
                if (data.result.length > 0) {
                    resolve(data.result[0]);  // Return the first form data entry
                } else {
                    resolve(null);  // No data found
                }
            };
            data.onerror = reject;
        };
        request.onerror = reject;
    });
}
```

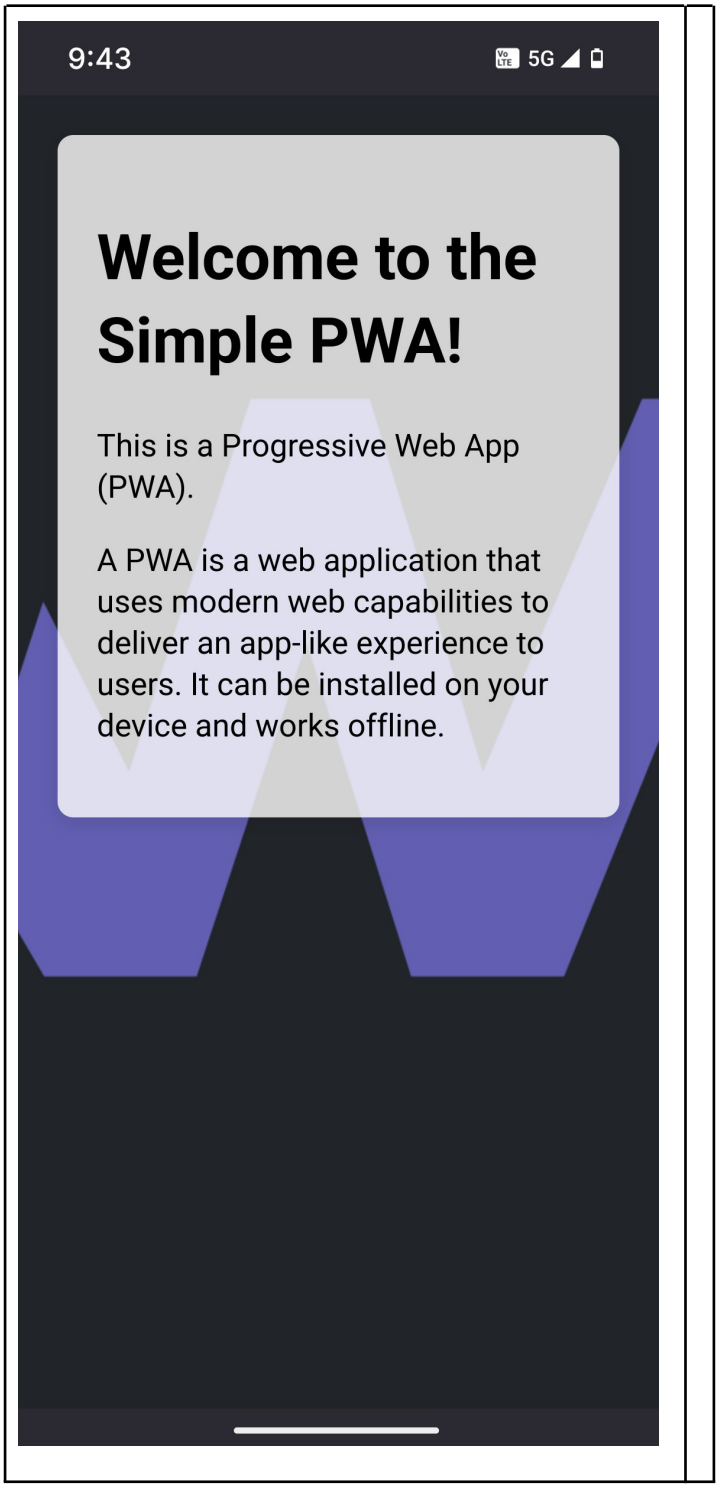
```
function saveFormDataLocally(formData) {
    // Save form data back into IndexedDB (simulating pushing data back into the app)
    const request = indexedDB.open('offlineFormData', 1);
    request.onsuccess = function() {
        const db = request.result;
        const tx = db.transaction('formData', 'readwrite');
        const store = tx.objectStore('formData');
        store.add(formData);  // Push data back into IndexedDB
        tx.oncomplete = function() {
            console.log('[Service Worker] Form data saved locally to IndexedDB:',
formData);
        };
    };
}

function clearFormDataFromIndexedDB() {
    const request = indexedDB.open('offlineFormData', 1);
    request.onsuccess = function() {
        const db = request.result;
        const tx = db.transaction('formData', 'readwrite');
        const store = tx.objectStore('formData');
        store.clear();  // Clear the form data from IndexedDB after syncing
        console.log('[Service Worker] Form data cleared from IndexedDB.');
    };
}

// Push event - handle push notifications (you can modify this part as per your needs)
self.addEventListener('push', (event) => {
    const options = {
        body: event.data.text(),
        icon: 'pwa-banner.png',  // Use pwa-banner.png for the notification icon
        badge: 'badge.png',      // Optional: You can use a badge if needed
    };

    event.waitUntil(
        self.registration.showNotification('New Content Available!', options)
    );
});

// Handle notification click
self.addEventListener('notificationclick', (event) => {
    event.notification.close();
    event.waitUntil(
        clients.openWindow('https://d15a-jai-61.github.io/static-pwa/') // Open your app's
homepage or a specific page
    );
});
```

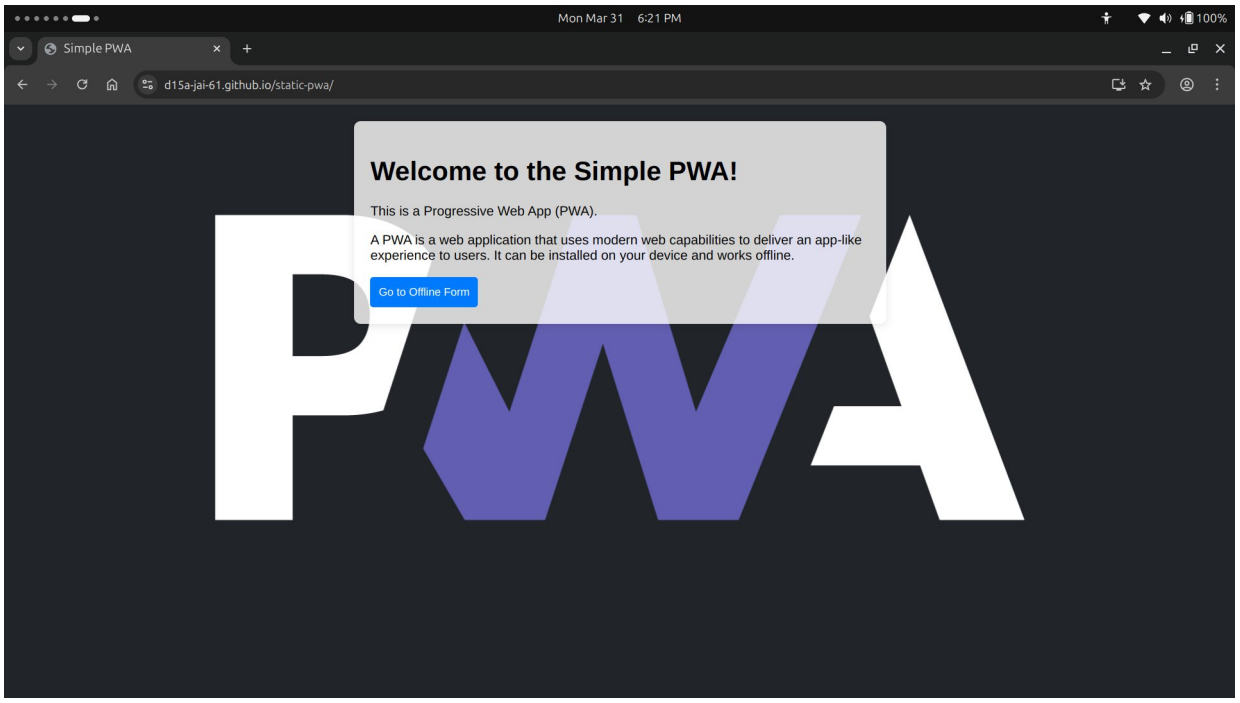
Output:

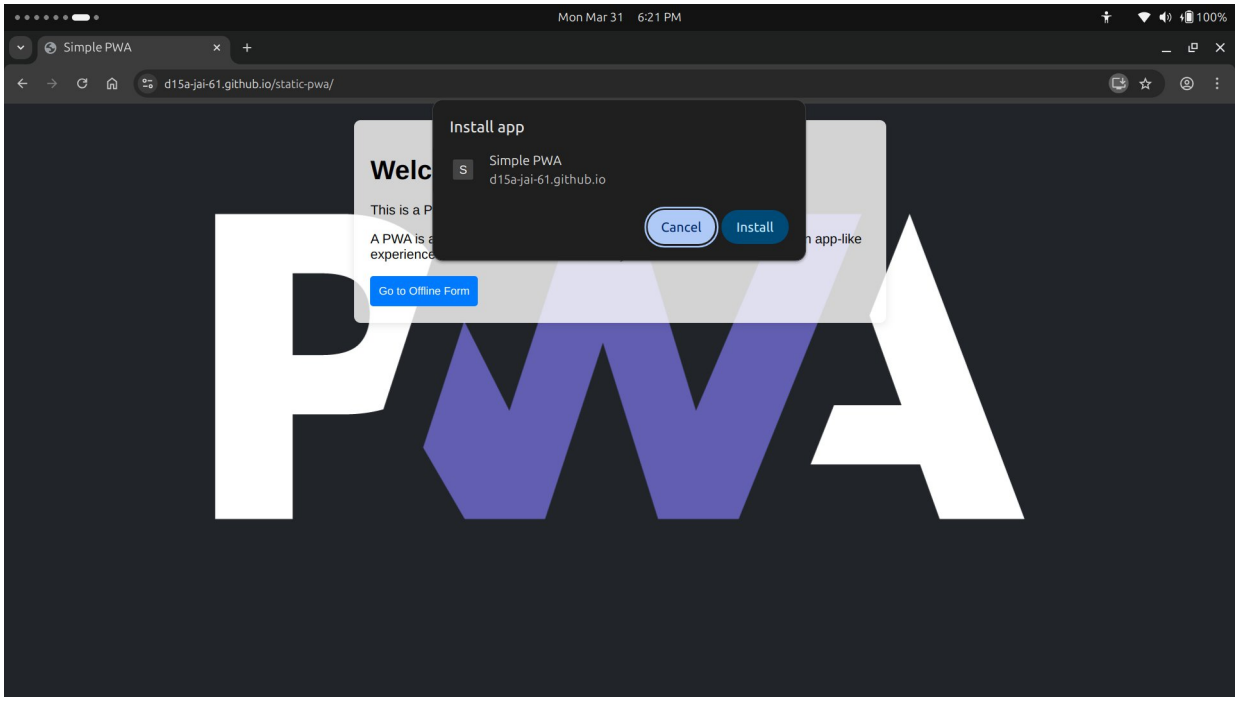

9:43
5G
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
d15a-jai-61.github.io/static
9:43
5G
New tab
Bookmarks
Add
History
Downloads
Passwords
Extensions
Synchronise and sav...
Find in page
Translate page
Desktop site
Report broken site...
Add app to Home sc...
Save to collection
Print
Settings

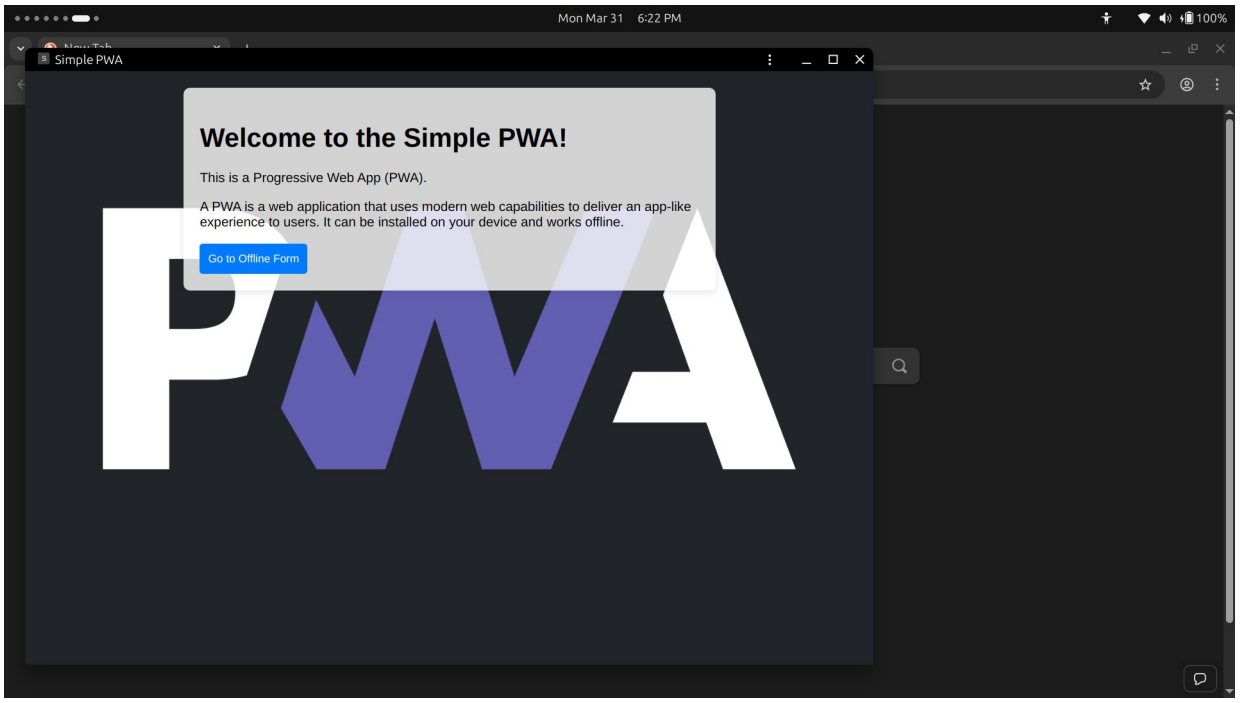

9:43
5G
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Firefox Nightly
Touch and hold the widget to move it around the home screen
PWA
1 × 1
Cancel
Add to home screen
9:43
5G
S
K
PWA

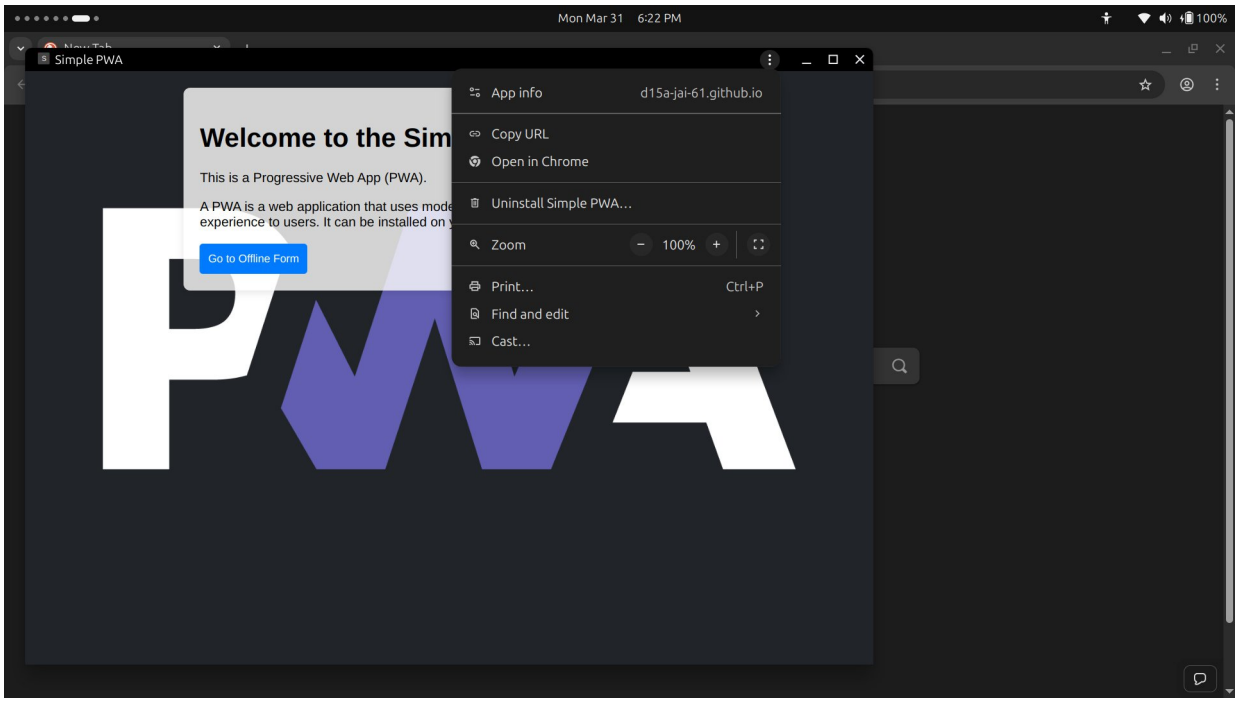

9:43
5G
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
d15a-jai-61.github.io/static
Screenshot
9:43
5G
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Screenshot

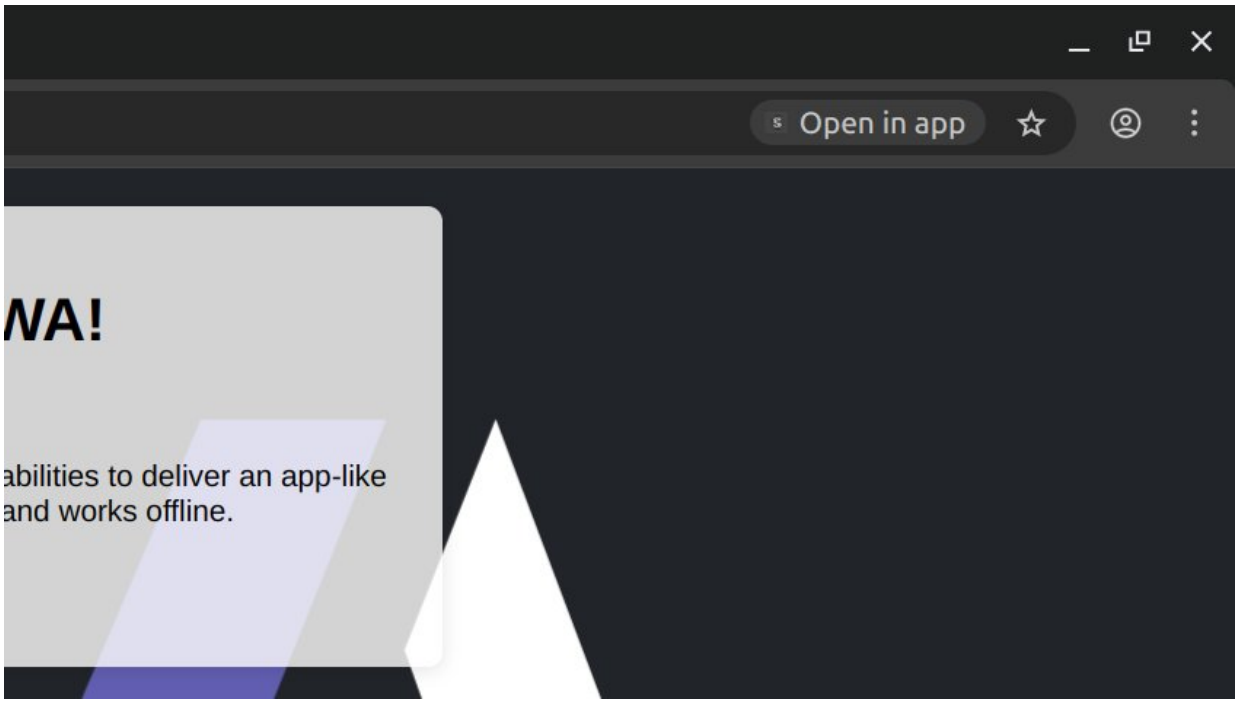

9:43
5G
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.

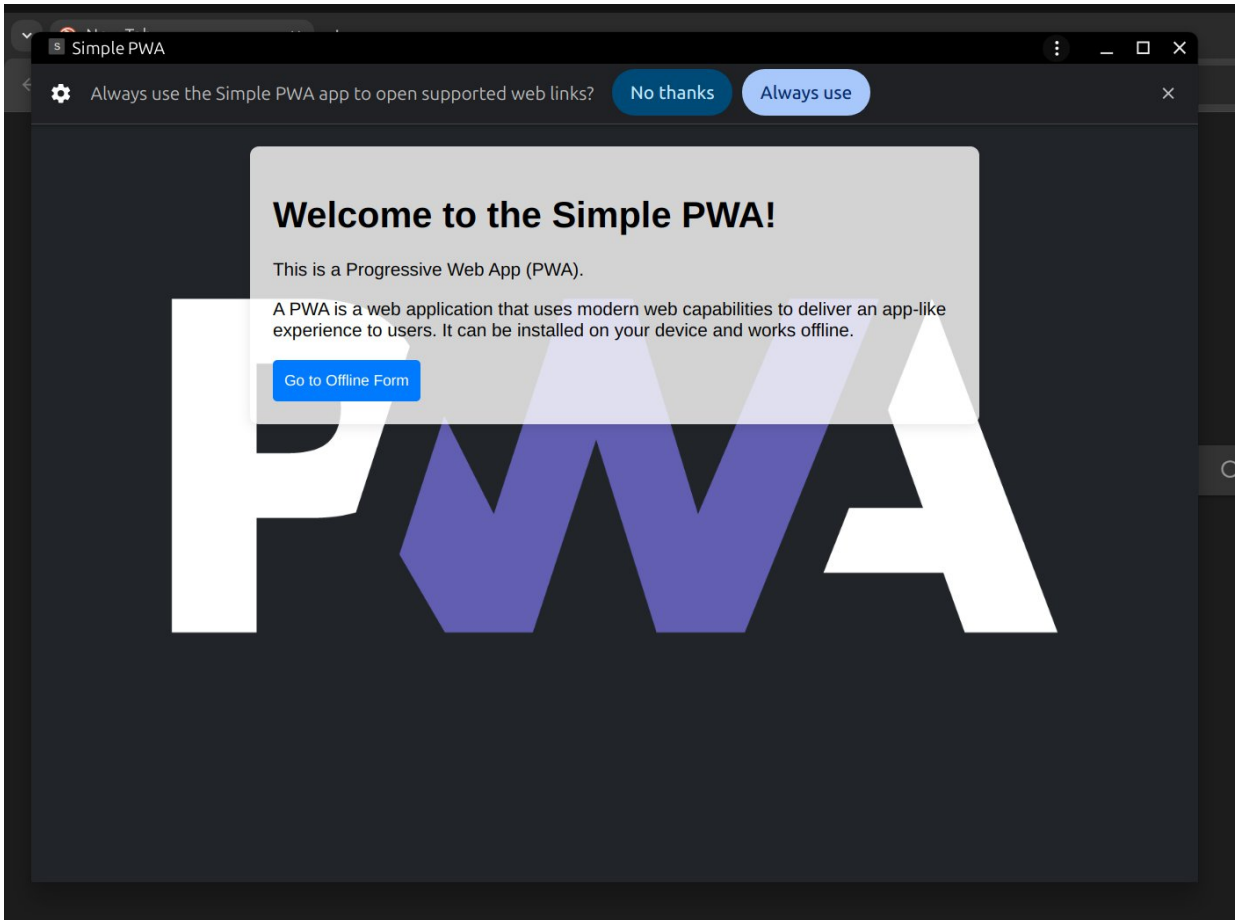

Mon Mar 31 6:21 PM
100%
Simple PWA
d15a-jai-61.github.io/static-pwa/
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Go to Offline Form
PWA


Mon Mar 31 6:21 PM
100%
Simple PWA
d15a-jai-61.github.io/static-pwa/
Install app
S
Simple PWA
d15a-jai-61.github.io
Cancel
Install
Welc
This is a P
A PWA is a
experience
app-like
Go to Offline Form
PWA


Mon Mar 31 6:22 PM
100%
Simple PWA
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Go to Offline Form


Mon Mar 31 6:22 PM
100%
Simple PWA
Welcome to the Sim
This is a Progressive Web App (PWA).
Go to Offline Form
App info
d15a-jai-61.github.io
Copy URL
Open in Chrome
Uninstall Simple PWA...
Zoom
100%
Print...
Ctrl+P
Find and edit
Cast...


Open in app
NA!
abilities to deliver an app-like
and works offline.


Simple PWA
Always use the Simple PWA app to open supported web links?
No thanks
Always use
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Go to Offline Form
PWA

Project Title: **Roll No.**

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.	59
Name	Jai Talreja
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Name: Diksha Utekar Jai Talreja D15A- 63/59

Experiment 8

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

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.

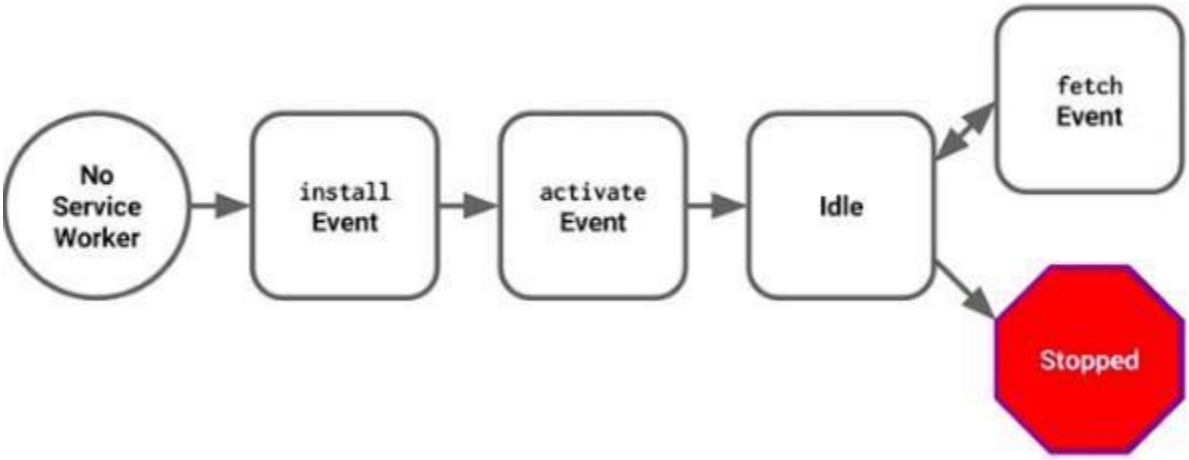
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

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:

main.js

```
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.

main.js

```
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

```
// Listen for install event, set callback
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
});
```

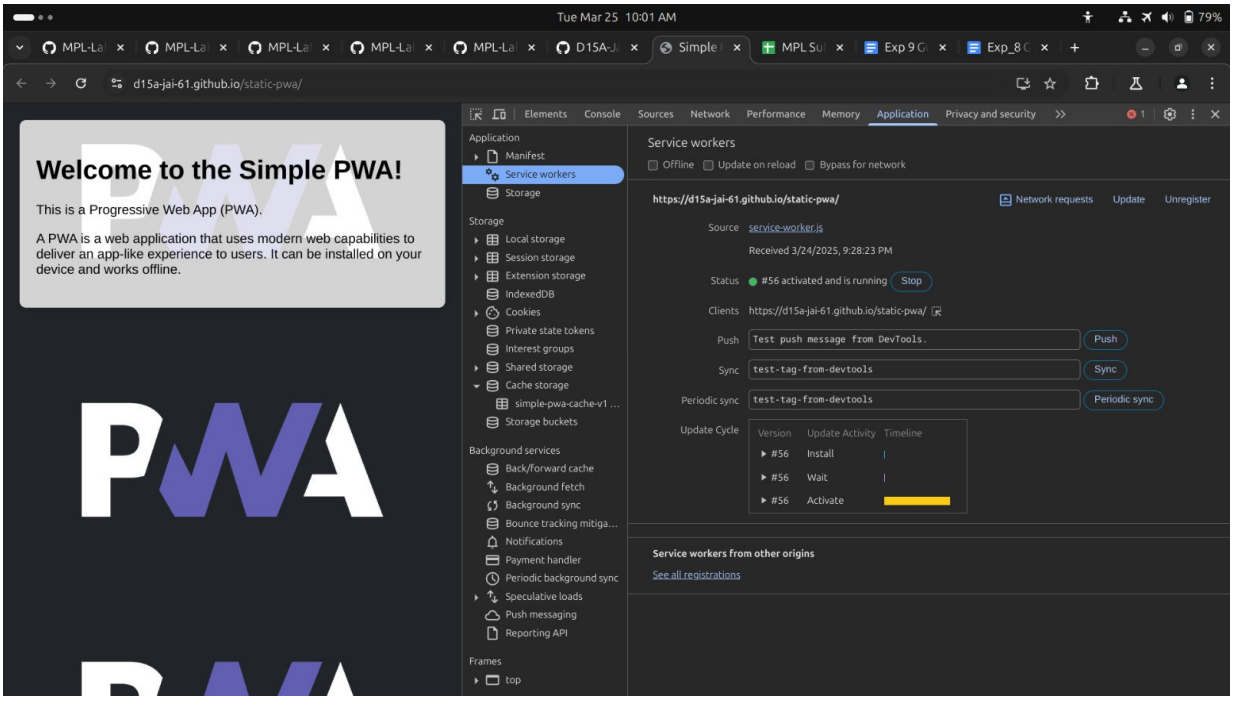
Once activated, the service worker controls all pages that load within its scope, and starts listening for events from those pages. However, pages in your app that were loaded before the service worker activation will not be under service worker control. The new service worker will only take over when 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.

Code under index.html:

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

Name: Diksha Utekar Jai Talreja D15A- 63/59

Output:



Project Title: **Roll No.**

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.	59
Name	Jai Talreja
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment 9

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.

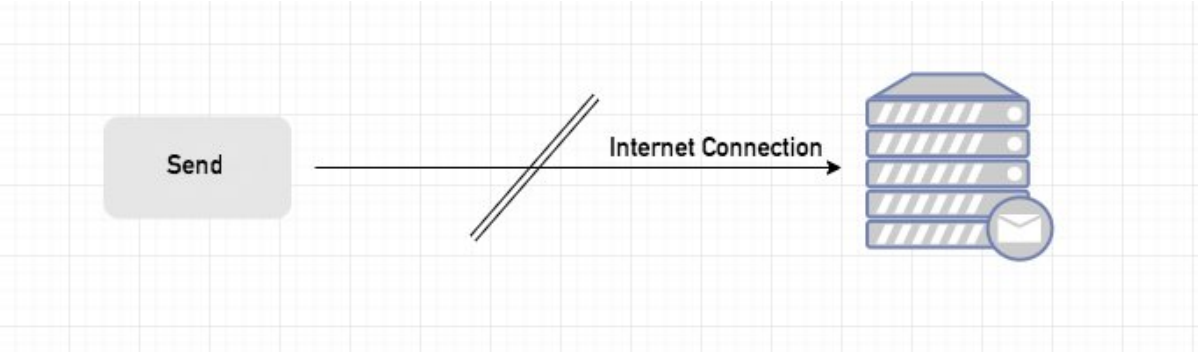
```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);

    if (url.origin === location.origin) {
        event.respondWith(cacheFirst(req));
    }
    else {
        event.respondWith(networkFirst(req));
    }
});

async function cacheFirst(req) {
    return await caches.match(req) || fetch(req);
}

async function networkFirst(req) {
    const cache = await caches.open("pwa-dynamic");
    try {
        const res = await fetch(req);
        cache.put(req, res.clone());
        return res;
    } catch (error) {
        const cachedResponse = await cache.match(req);
        return cachedResponse || await caches.match("./noconnection.json");
    }
}
```

Sync Event

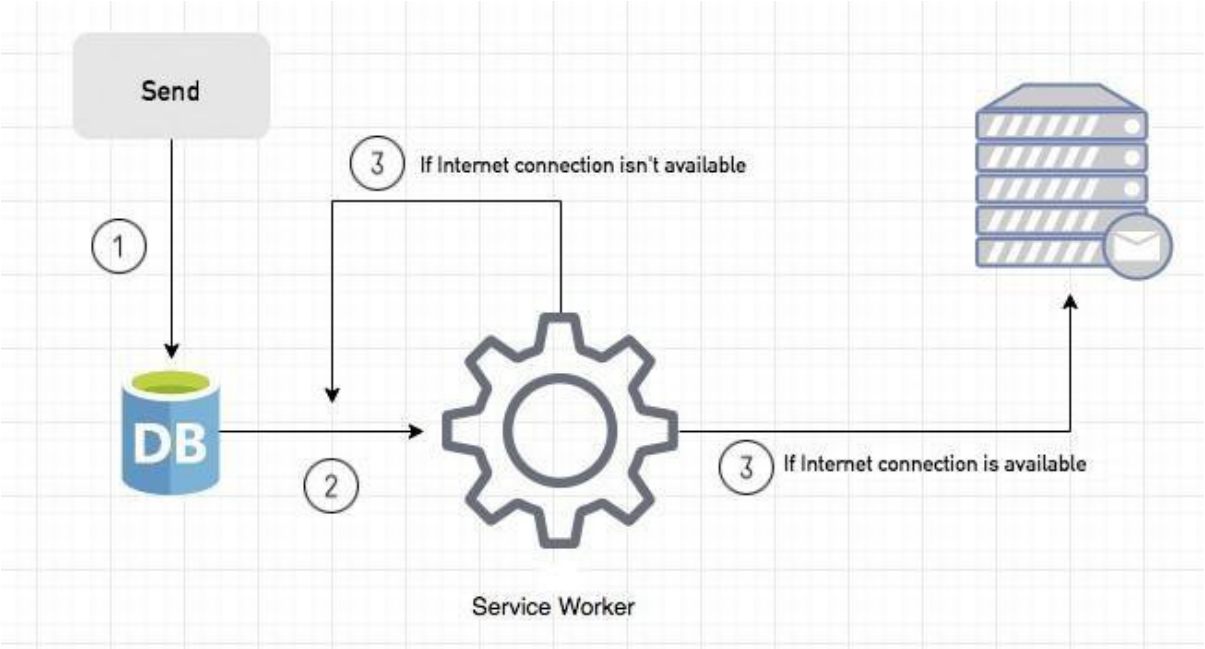
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are 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.

Push Event

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

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

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

Code:

service-worker.js :

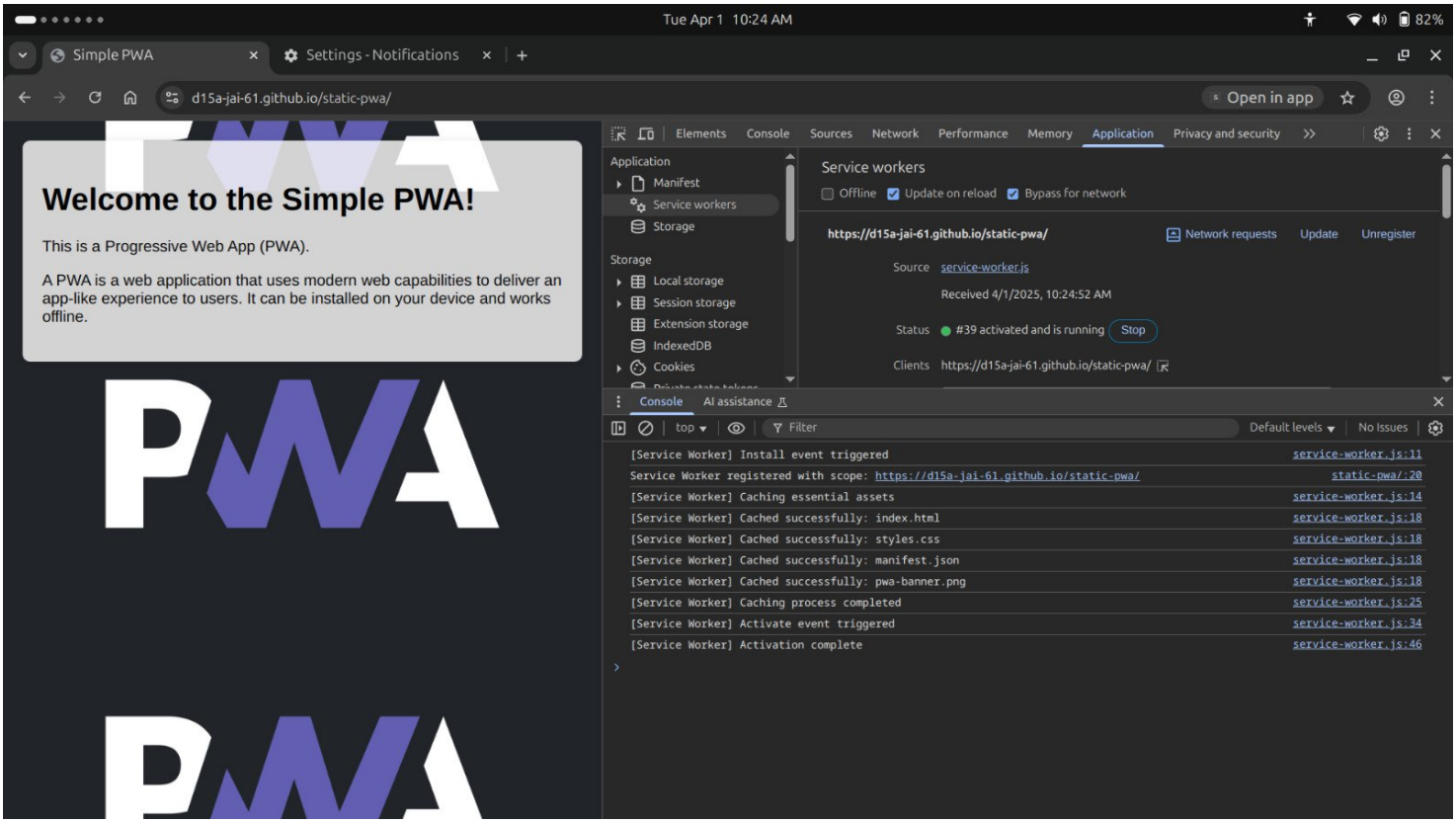
```
const CACHE_NAME = 'static-pwa-cache-v1';
const urlsToCache = [
    'index.html',
    'styles.css',
    'manifest.json',
    'pwa-banner.png',
];

// Install event - Cache essential assets
self.addEventListener('install', (event) => {
    console.log('[Service Worker] Install event triggered');
    event.waitUntil(
        caches.open(CACHE_NAME).then(cache => {
            console.log('[Service Worker] Caching essential assets');
            return Promise.all(
                urlsToCache.map(url =>
                    cache.add(url).then(() =>
                        console.log([Service Worker] Cached successfully: ${url})
                    ).catch(error =>
                        console.error([Service Worker] Failed to cache: ${url}, error)
                    )
                )
            );
        }).then(() => {
            console.log('[Service Worker] Caching process completed');
            return self.skipWaiting();
        })
        .catch(error => console.error('[Service Worker] Caching failed:', error))
    );
});

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

// Fetch event - Serve cached assets or fetch from network
```

```
self.addEventListener('fetch', (event) => {
    console.log([Service Worker] Fetch event triggered for: ${event.request.url});
    if (event.request.method !== 'GET') return;

    event.respondWith(
        fetch(event.request).then(networkResponse => {
            console.log([Service Worker] Network response for ${event.request.url});
            return caches.open(CACHE_NAME).then(cache => {
                cache.put(event.request, networkResponse.clone());
                return networkResponse;
            });
        }).catch(error => {
            console.log([Service Worker] Network failed, trying cache for ${event.request.url});
            return caches.match(event.request).then(cachedResponse => {
                if (cachedResponse) {
                    console.log([Service Worker] Serving from cache: ${event.request.url});
                    return cachedResponse;
                }
                if (event.request.mode === 'navigate') {
                    return caches.match(OFFLINE_URL);
                }
                return new Response('Offline', { status: 503, statusText: 'Service Unavailable' });
            });
        })
    );
});

// Sync event - Handle background sync
self.addEventListener('sync', (event) => {
    console.log([Service Worker] Sync event triggered: ${event.tag});
    if (event.tag === 'sync-demo') {
        event.waitUntil(
            handleSync().catch(error => {
                console.error('[Service Worker] Sync failed:', error);
            })
        );
    }
});

async function handleSync() {
    console.log('[Service Worker] Starting sync...');
    await new Promise(resolve => setTimeout(resolve, 2000));
    console.log('[Service Worker] Sync task completed successfully!');
}

// Push event - Handle push notifications
self.addEventListener('push', (event) => {
    console.log('[Service Worker] Push event received');
    const options = {
        body: event.data ? event.data.text() : 'New update available!',
        icon: 'pwa-banner.png',
        badge: 'badge.png',
        actions: [
            { action: 'open', title: 'View Now' },
            { action: 'dismiss', title: 'Dismiss' }
        ]
```

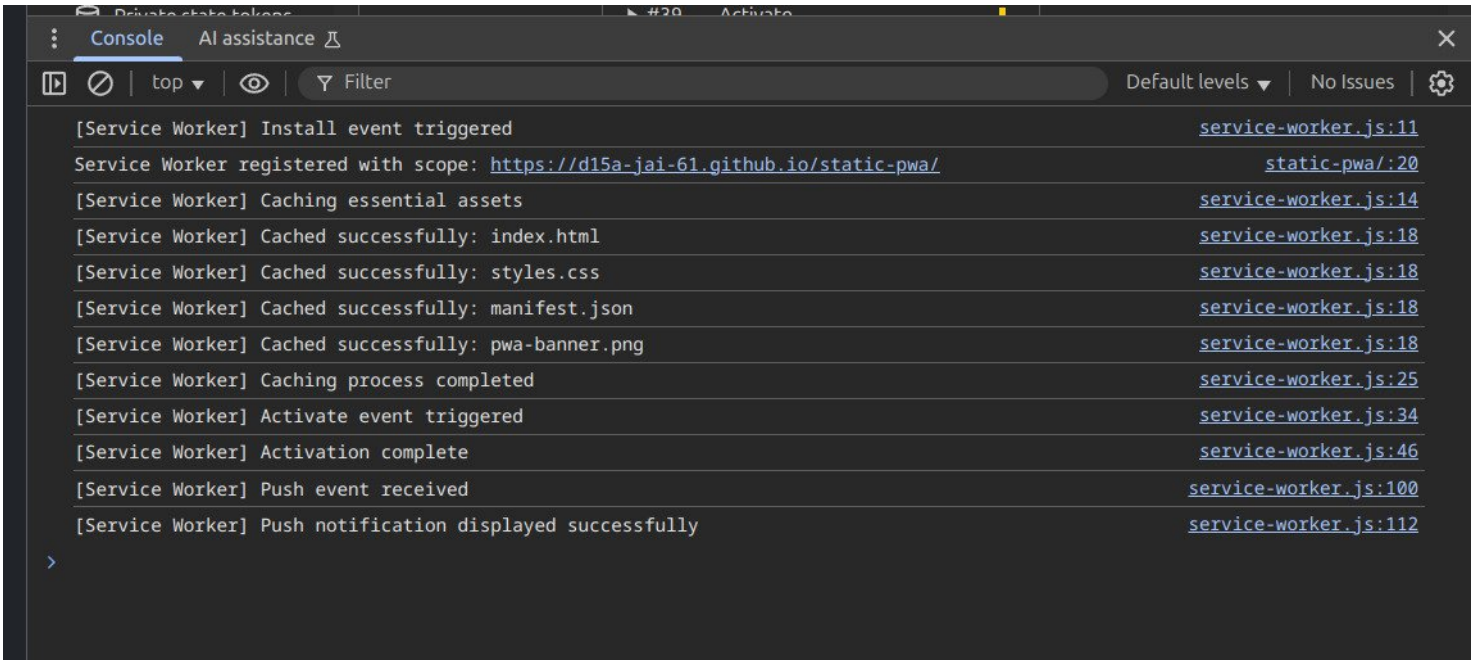
```
    };
    event.waitUntil(
        self.registration.showNotification('New Notification', options)
        .then(() => console.log('[Service Worker] Push notification displayed successfully'))
        .catch((error) => console.error('[Service Worker] Failed to display push notification:', error))
    );
});

// Handle notification click
self.addEventListener('notificationclick', (event) => {
    console.log([Service Worker] Notification clicked: ${event.notification.title});
    event.notification.close();
    if (event.action === 'open') {
        console.log('[Service Worker] Opening application');
        event.waitUntil(clients.openWindow('https://your-static-site.com'));
    } else {
        console.log('[Service Worker] Notification dismissed');
    }
});
```

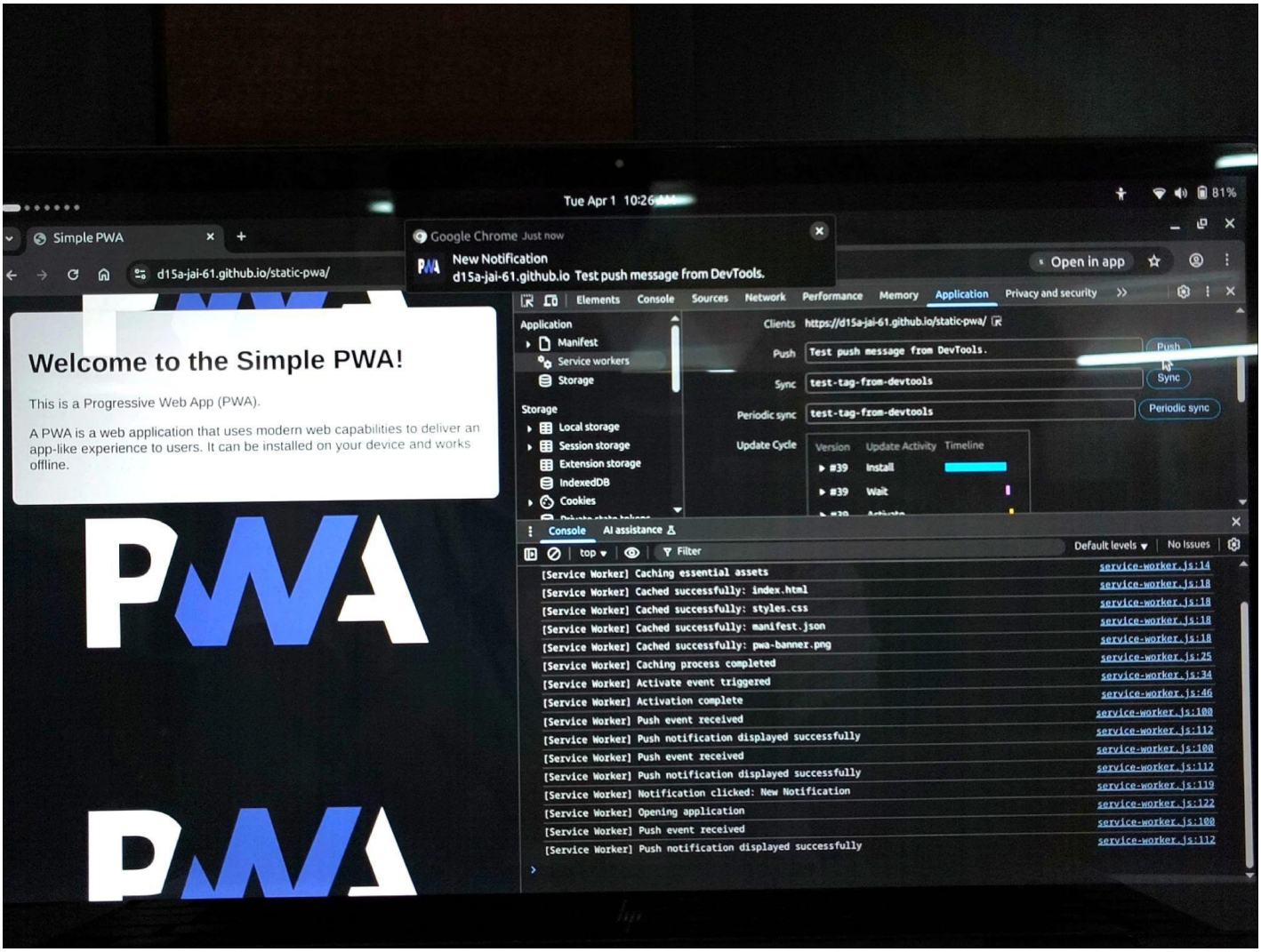
OUTPUT:

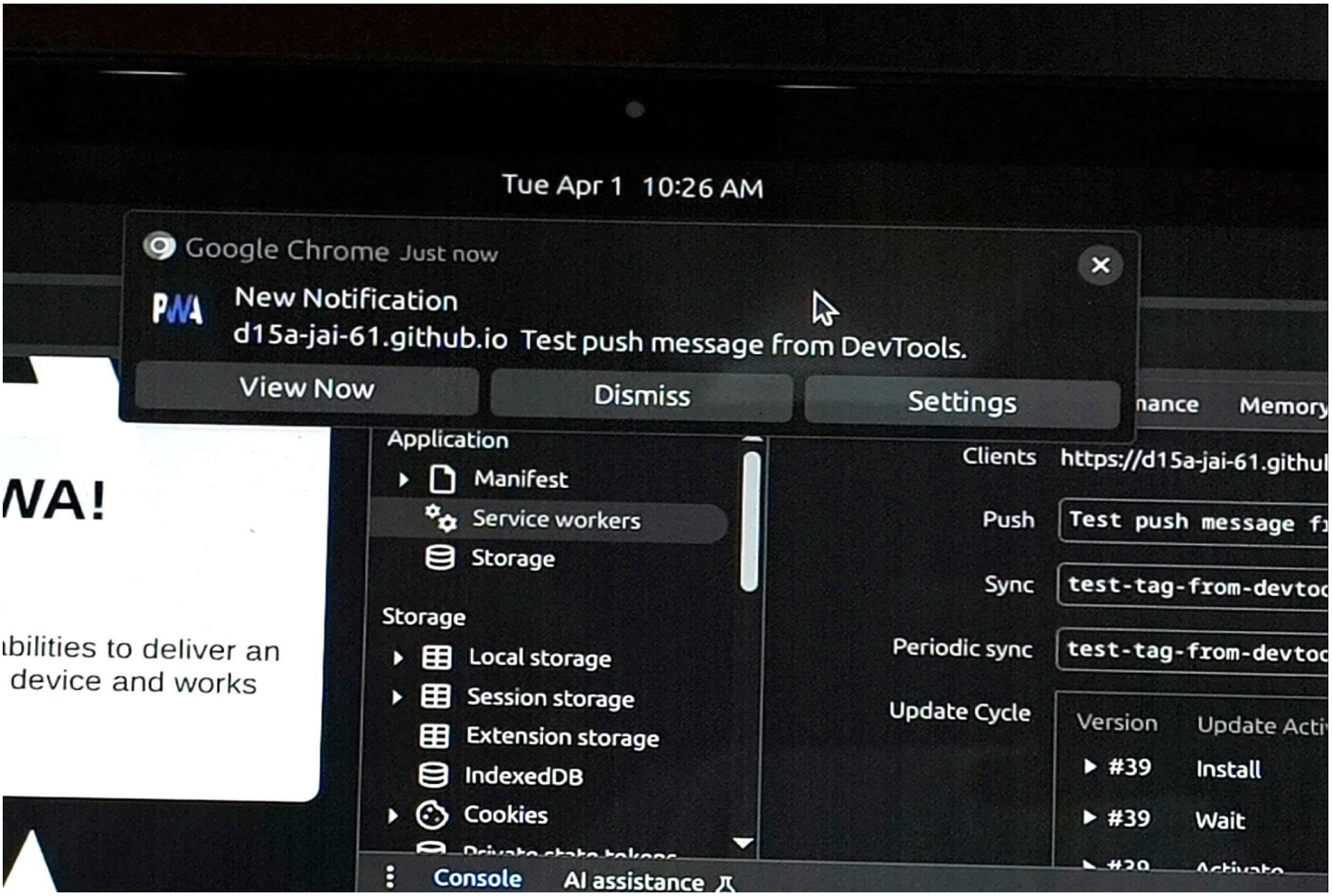
Fetch event



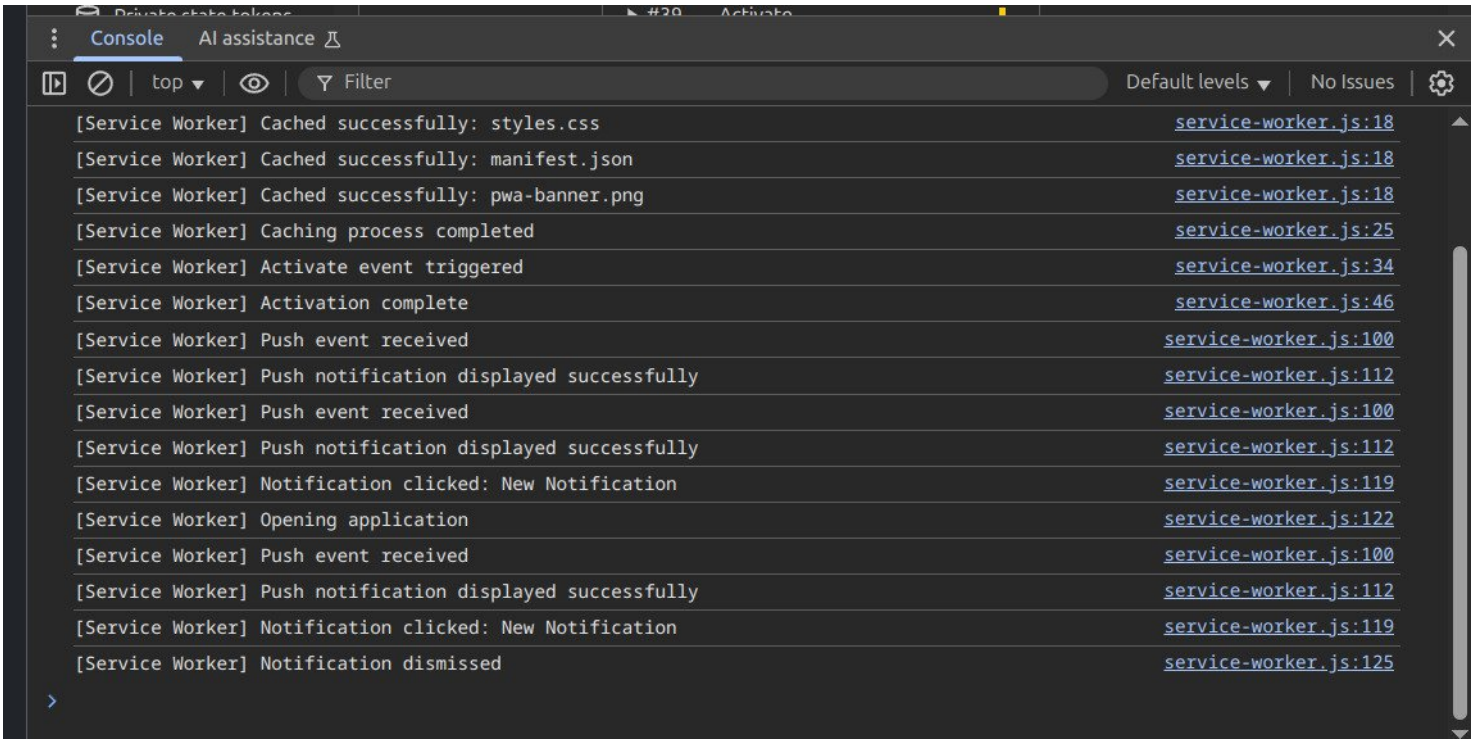
Push event


Console
AI assistance
top
Filter
Default levels
No Issues
[Service Worker] Install event triggered service-worker.js:11
Service Worker registered with scope: https://d15a-jai-61.github.io/static-pwa/ static-pwa/:20
[Service Worker] Caching essential assets service-worker.js:14
[Service Worker] Cached successfully: index.html service-worker.js:18
[Service Worker] Cached successfully: styles.css service-worker.js:18
[Service Worker] Cached successfully: manifest.json service-worker.js:18
[Service Worker] Cached successfully: pwa-banner.png service-worker.js:18
[Service Worker] Caching process completed service-worker.js:25
[Service Worker] Activate event triggered service-worker.js:34
[Service Worker] Activation complete service-worker.js:46
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112


Simple PWA
d15a-jai-61.github.io/static-pwa/
Google Chrome Just now
New Notification
d15a-jai-61.github.io Test push message from DevTools.
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
PWA
PWA
Open in app
Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Clients https://d15a-jai-61.github.io/static-pwa/
Push Test push message from DevTools.
Sync test-tag-from-devtools
Periodic sync test-tag-from-devtools
Update Cycle
Version
Update Activity
Timeline
#39 Install
#39 Wait
Console
AI assistance
top
Filter
Default levels
No Issues
[Service Worker] Caching essential assets service-worker.js:14
[Service Worker] Cached successfully: index.html service-worker.js:18
[Service Worker] Cached successfully: styles.css service-worker.js:18
[Service Worker] Cached successfully: manifest.json service-worker.js:18
[Service Worker] Cached successfully: pwa-banner.png service-worker.js:18
[Service Worker] Caching process completed service-worker.js:25
[Service Worker] Activate event triggered service-worker.js:34
[Service Worker] Activation complete service-worker.js:46
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112
[Service Worker] Notification clicked: New Notification service-worker.js:119
[Service Worker] Opening application service-worker.js:122
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112



After dismissing the notification



Sync event


[Service Worker] Opening application service-worker.js:122
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112
[Service Worker] Notification clicked: New Notification service-worker.js:119
[Service Worker] Notification dismissed service-worker.js:125
[Service Worker] Sync event triggered: test-tag-from-devtools service-worker.js:82

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	59
Name	Jai Talreja
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment 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.

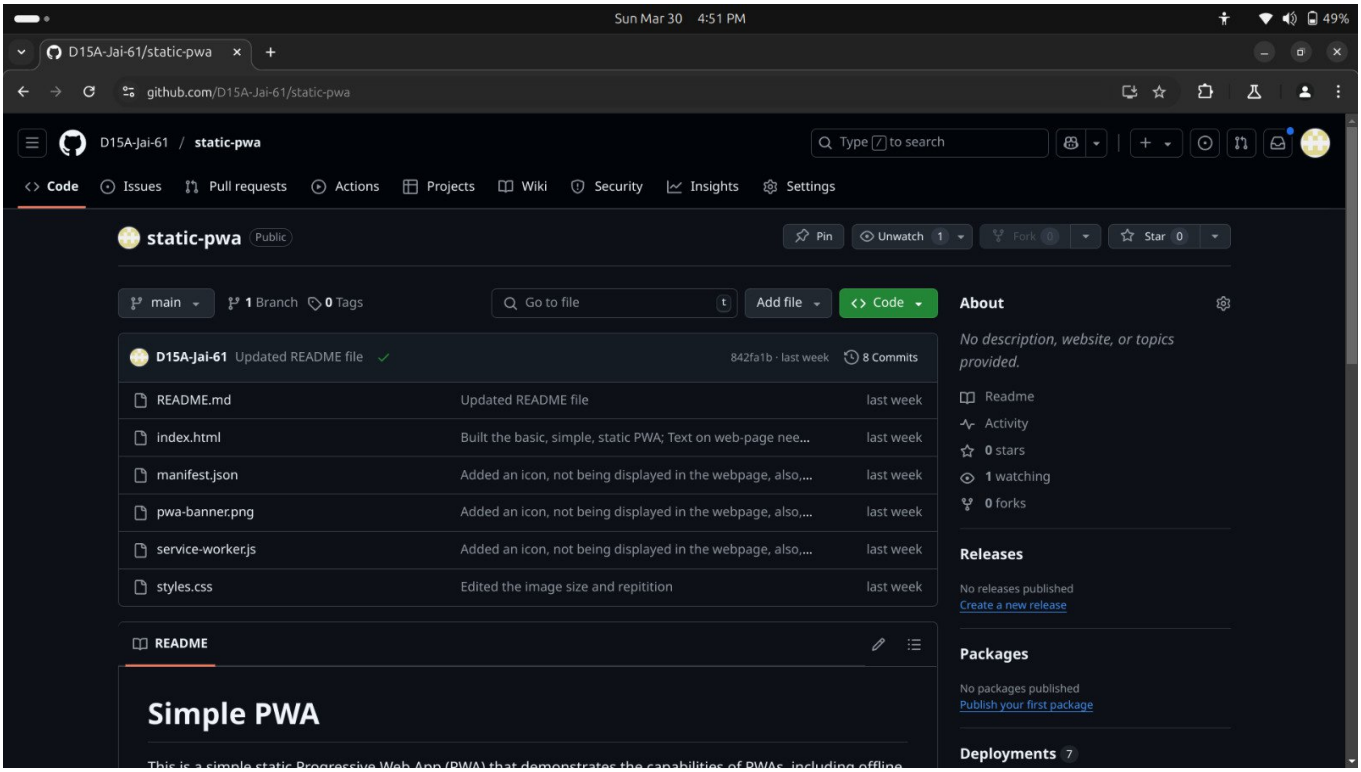
Cons

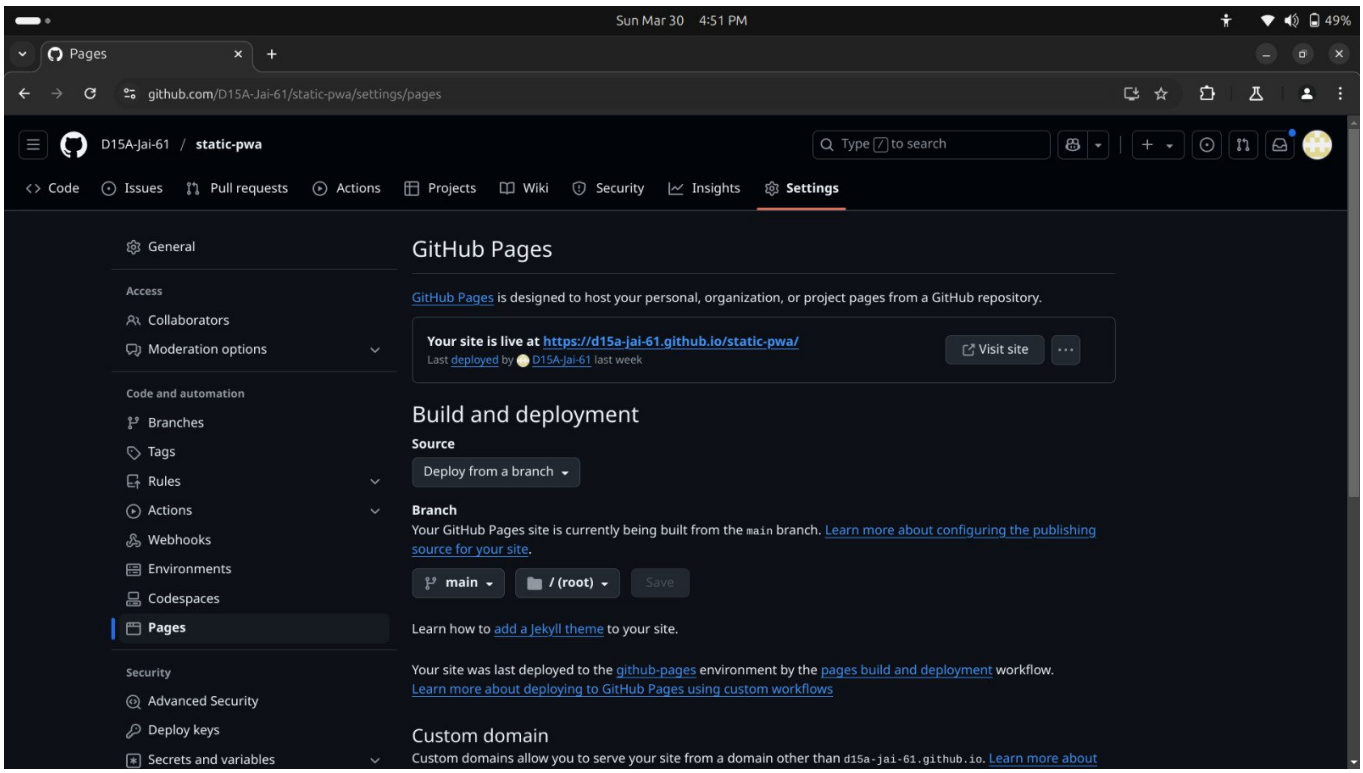
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

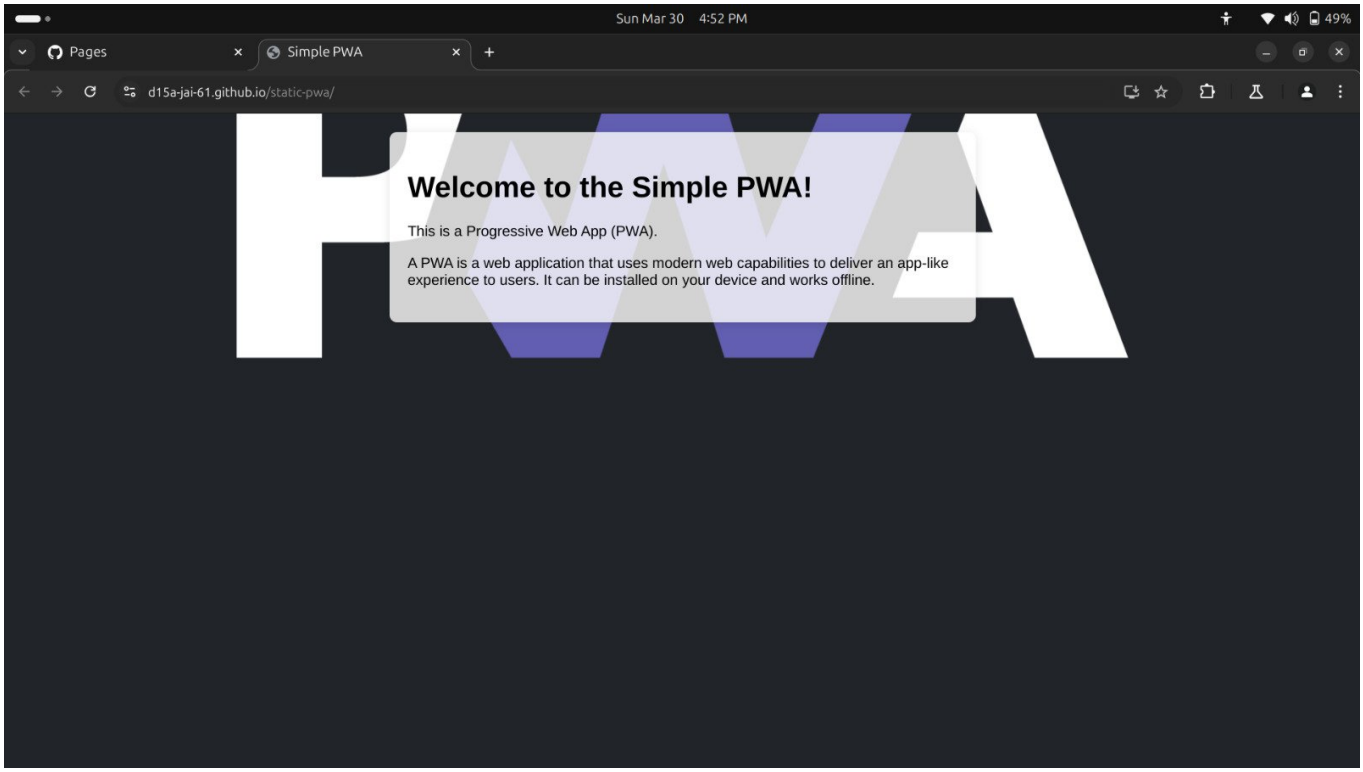
Link to our GitHub repository:

https://github.com/D15A-Jai-61/static-pwa.git

Github Screenshot:


Sun Mar 30 4:51 PM
49%
D15A-Jai-61/static-pwa
github.com/D15A-Jai-61/static-pwa
D15A-Jai-61 / static-pwa
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
static-pwa
Public
Pin
Unwatch 1
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
Add file
Code
D15A-Jai-61 Updated README file
842fa1b · last week
8 Commits
README.md Updated README file last week
index.html Built the basic, simple, static PWA; Text on web-page nee... last week
manifest.json Added an icon, not being displayed in the webpage, also,... last week
pwa-banner.png Added an icon, not being displayed in the webpage, also,... last week
service-worker.js Added an icon, not being displayed in the webpage, also,... last week
styles.css Edited the image size and repitition last week
README
Simple PWA
About
No description, website, or topics provided.
Readme
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Deployments 7


Sun Mar 30 4:51 PM
49%
Pages
github.com/D15A-Jai-61/static-pwa/settings/pages
D15A-Jai-61 / static-pwa
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Security
Advanced Security
Deploy keys
Secrets and variables
GitHub Pages
GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.
Your site is live at https://d15a-jai-61.github.io/static-pwa/
Last deployed by D15A-Jai-61 last week
Visit site
Build and deployment
Source
Deploy from a branch
Branch
Your GitHub Pages site is currently being built from the main branch. Learn more about configuring the publishing source for your site.
main
/ (root)
Save
Learn how to add a Jekyll theme to your site.
Your site was last deployed to the github-pages environment by the pages build and deployment workflow.
Learn more about deploying to GitHub Pages using custom workflows
Custom domain
Custom domains allow you to serve your site from a domain other than d15a-jai-61.github.io. Learn more about


Sun Mar 30 4:52 PM
49%
Pages
Simple PWA
d15a-jai-61.github.io/static-pwa/
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	59
Name	Jai Talreja
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

Experiment 11

Aim :

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

Theory :

Reference : https://www.semrush.com/blog/google-lighthouse/

Google Lighthouse :

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

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

Key Features and Audit Metrics

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

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

PWA Score (Mobile): Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards the PWA paradigm,

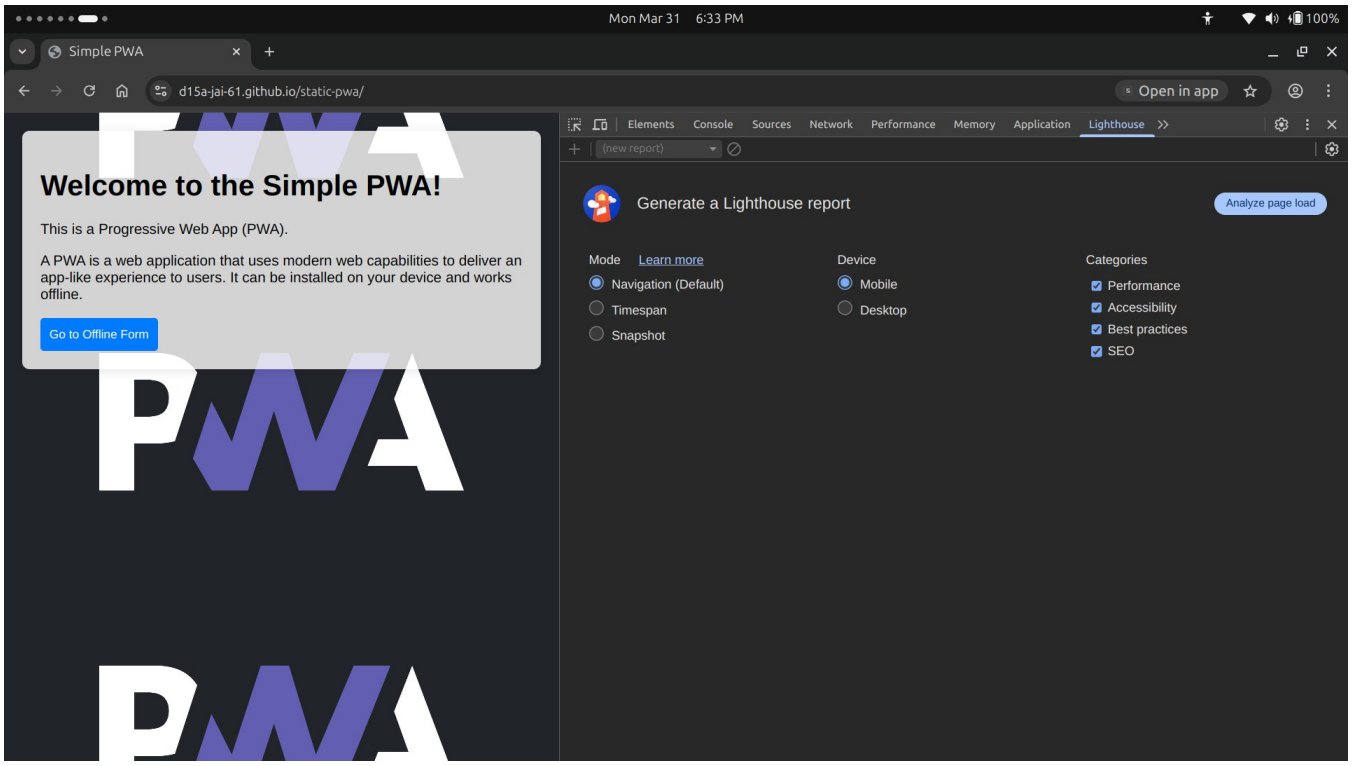
where the objective is to make the application behave as close as possible to native mobile applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled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

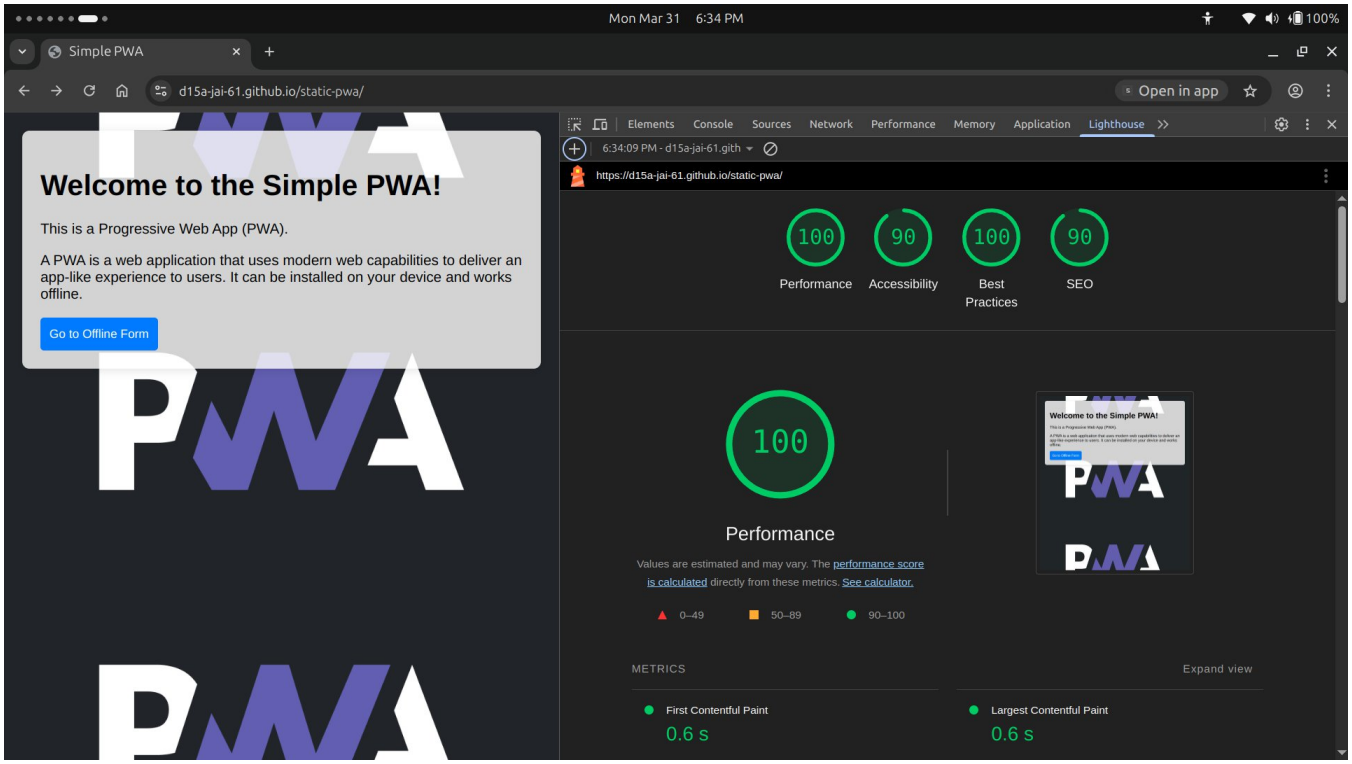
Accessibility: As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the ‘aria-’ attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though,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

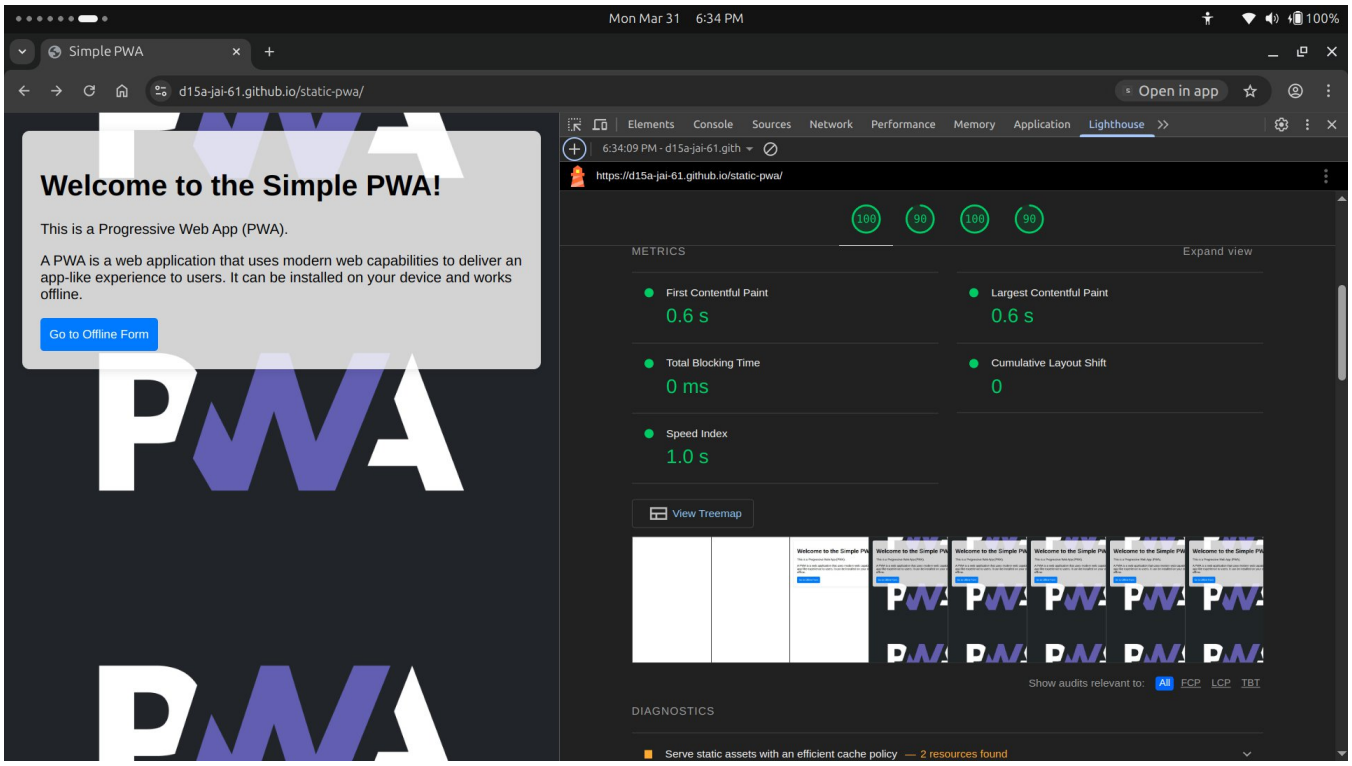
Best Practices: As any developer would know, there are a number of practices that have been deemed ‘best’ based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS

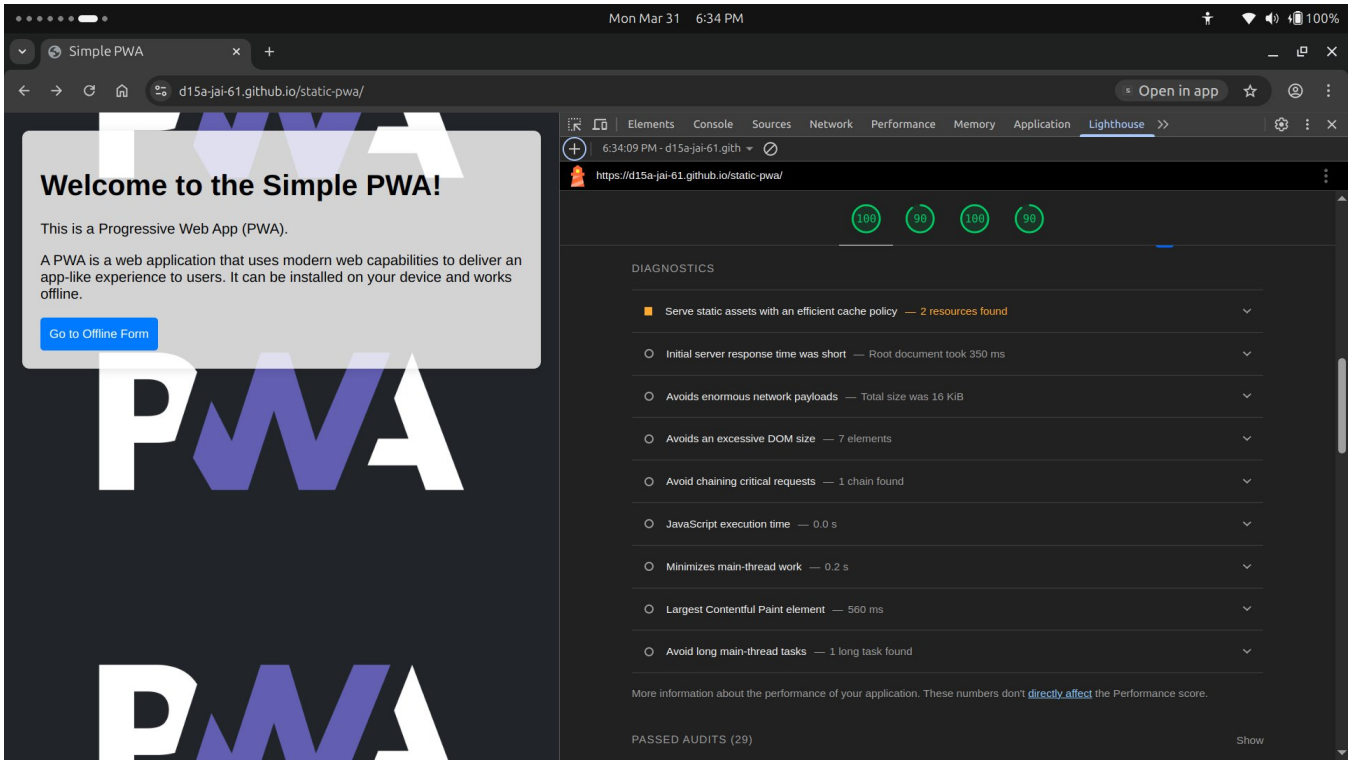
Avoiding the use of deprecated code elements like tags, directives, libraries, etc. Password input with paste-into disabled

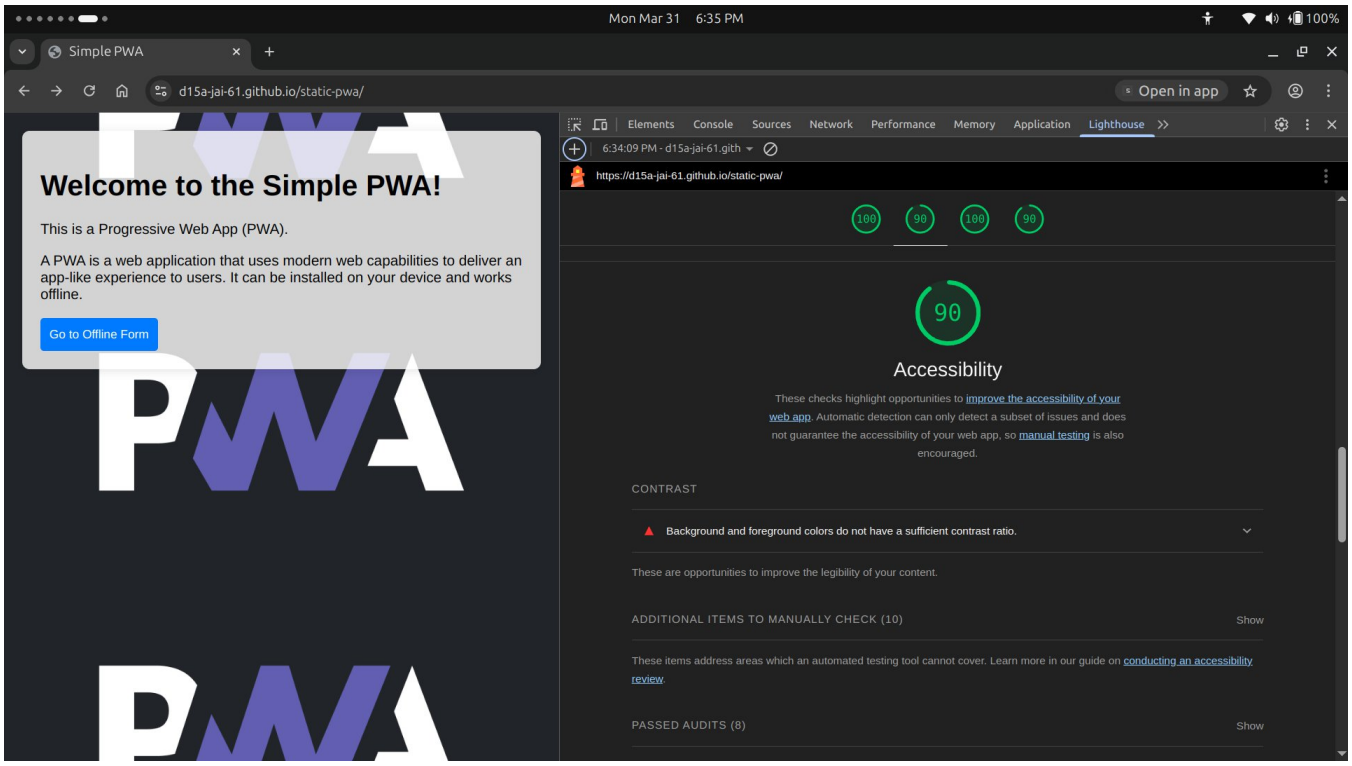
Geo-Location and cookie usage alerts on load, etc.

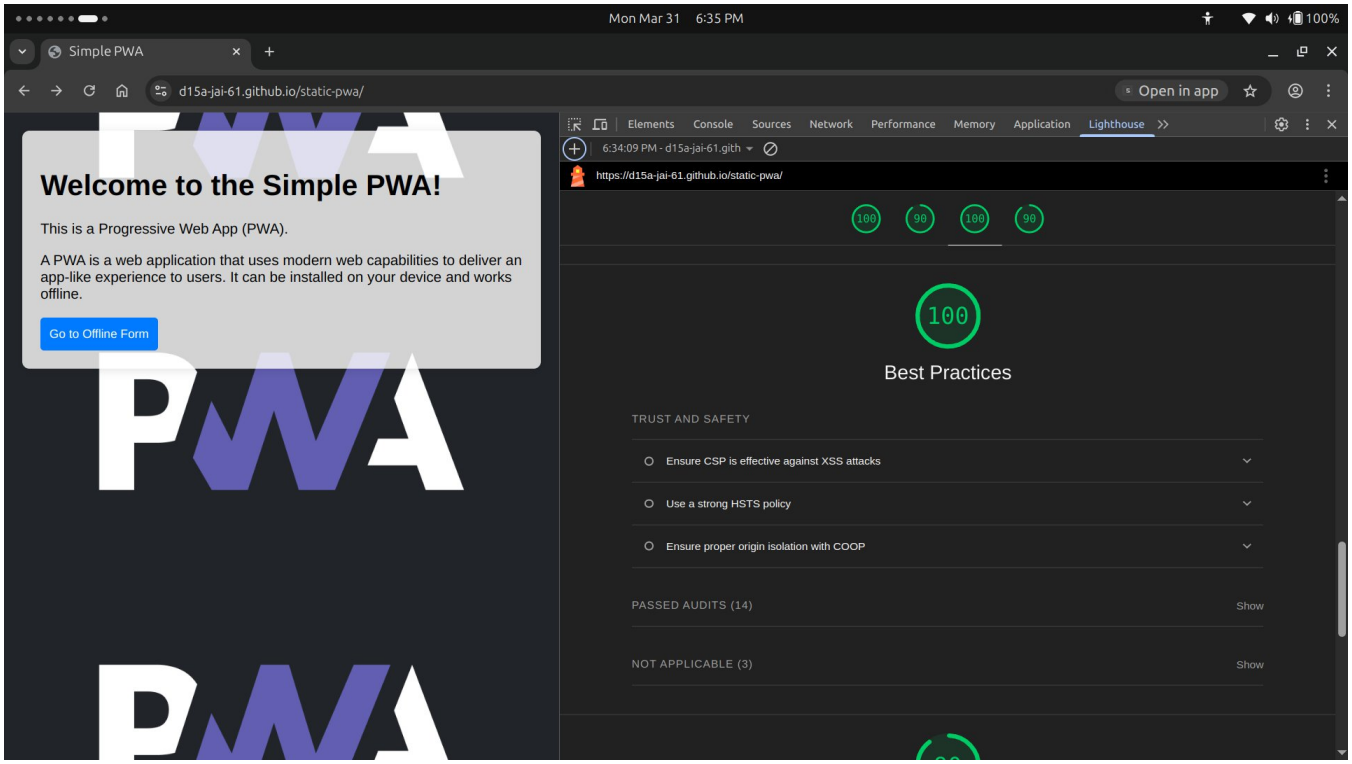

Mon Mar 31 6:33 PM
Simple PWA
d15a-jai-61.github.io/static-pwa/
Open in app
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Go to Offline Form
Elements Console Sources Network Performance Memory Application Lighthouse
Generate a Lighthouse report
Analyze page load
Mode Learn more
Navigation (Default)
Timespan
Snapshot
Device
Mobile
Desktop
Categories
Performance
Accessibility
Best practices
SEO

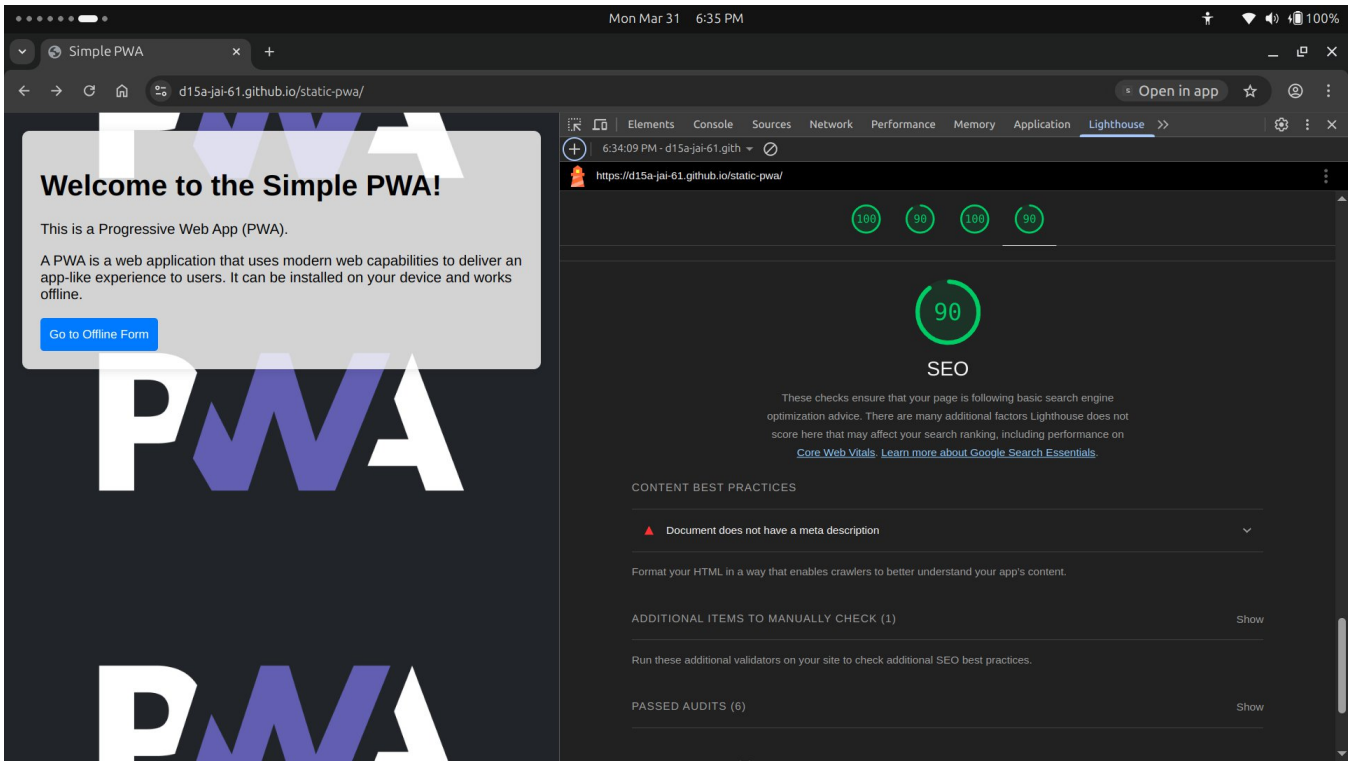

Mon Mar 31 6:34 PM
Simple PWA
d15a-jai-61.github.io/static-pwa/
Open in app
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Go to Offline Form
Elements Console Sources Network Performance Memory Application Lighthouse
https://d15a-jai-61.github.io/static-pwa/
100 Performance
90 Accessibility
100 Best Practices
90 SEO
100
Performance
Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.
0–49 50–89 90–100
METRICS
Expand view
First Contentful Paint
0.6 s
Largest Contentful Paint
0.6 s


Mon Mar 31 6:34 PM
100%
Simple PWA
d15a-jai-61.github.io/static-pwa/
Open in app
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Go to Offline Form
Elements Console Sources Network Performance Memory Application Lighthouse
https://d15a-jai-61.github.io/static-pwa/
100 90 100 90
METRICS
Expand view
First Contentful Paint
0.6 s
Largest Contentful Paint
0.6 s
Total Blocking Time
0 ms
Cumulative Layout Shift
0
Speed Index
1.0 s
View Treemap
Show audits relevant to: All FCP LCP TBT
DIAGNOSTICS
Serve static assets with an efficient cache policy — 2 resources found


Mon Mar 31 6:34 PM
100%
Simple PWA
d15a-jai-61.github.io/static-pwa/
Open in app
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Go to Offline Form
Elements Console Sources Network Performance Memory Application Lighthouse
https://d15a-jai-61.github.io/static-pwa/
100 90 100 90
DIAGNOSTICS
Serve static assets with an efficient cache policy — 2 resources found
Initial server response time was short — Root document took 350 ms
Avoids enormous network payloads — Total size was 16 KiB
Avoids an excessive DOM size — 7 elements
Avoid chaining critical requests — 1 chain found
JavaScript execution time — 0.0 s
Minimizes main-thread work — 0.2 s
Largest Contentful Paint element — 560 ms
Avoid long main-thread tasks — 1 long task found
More information about the performance of your application. These numbers don't directly affect the Performance score.
PASSED AUDITS (29)
Show


Mon Mar 31 6:35 PM
Simple PWA
d15a-jai-61.github.io/static-pwa/
Open in app
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Go to Offline Form
Elements Console Sources Network Performance Memory Application Lighthouse
https://d15a-jai-61.github.io/static-pwa/
100 90 100 90
90
Accessibility
These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.
CONTRAST
Background and foreground colors do not have a sufficient contrast ratio.
These are opportunities to improve the legibility of your content.
ADDITIONAL ITEMS TO MANUALLY CHECK (10)
Show
These items address areas which an automated testing tool cannot cover. Learn more in our guide on conducting an accessibility review.
PASSED AUDITS (8)
Show


Mon Mar 31 6:35 PM
Simple PWA
d15a-jai-61.github.io/static-pwa/
Open in app
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Go to Offline Form
Elements Console Sources Network Performance Memory Application Lighthouse
https://d15a-jai-61.github.io/static-pwa/
100 90 100 90
100
Best Practices
TRUST AND SAFETY
Ensure CSP is effective against XSS attacks
Use a strong HSTS policy
Ensure proper origin isolation with COOP
PASSED AUDITS (14)
Show
NOT APPLICABLE (3)
Show



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

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	59
Name	Jai Talreja
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	Assignment-2
Assignment 2 Questions	1. Define Progressive Web App (PWA) and explain its significance in modern web development.Discuss the key characteristics that differentiate PWAs from traditional mobile apps 2. Define responsive web design and explain its importance in the context of Progressive Web Apps.Compare and contrast responsive, fluid, and adaptive web design approaches. 3. Describe the lifecycle of Service Workers, including registration, installation, and activation phases. 4. Explain the use of IndexedDB in the Service Worker for data storage.
Roll No.	59
Name	Jai Talreja
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	