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 _____**SHRAEYAA DHAIGUDE**_____ of **D15A/D15B** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2024-2025.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Project Title: **Roll No.**

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

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

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers : Mrs. Kajal Joseph.

Email : kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: **Roll No.**

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			

Project Title: **Roll No.**

MAD & PWA Lab
Journal

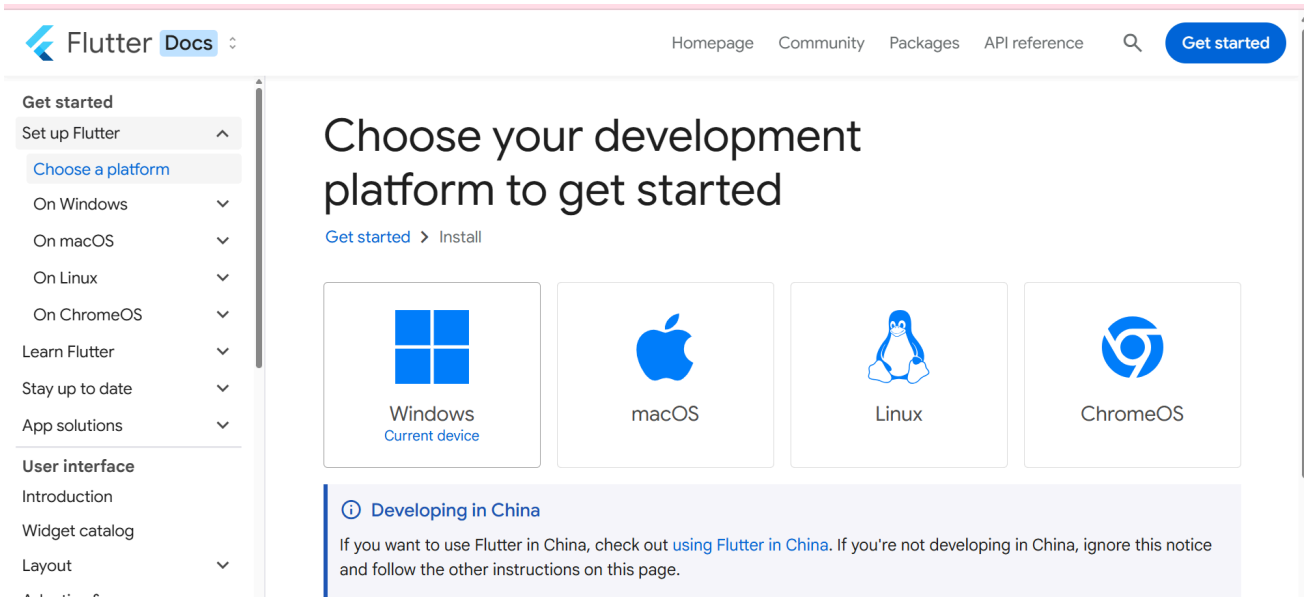
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

EXP 1

AIM: Installation and Configuration of Flutter Environment.

Install the Flutter SDK

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

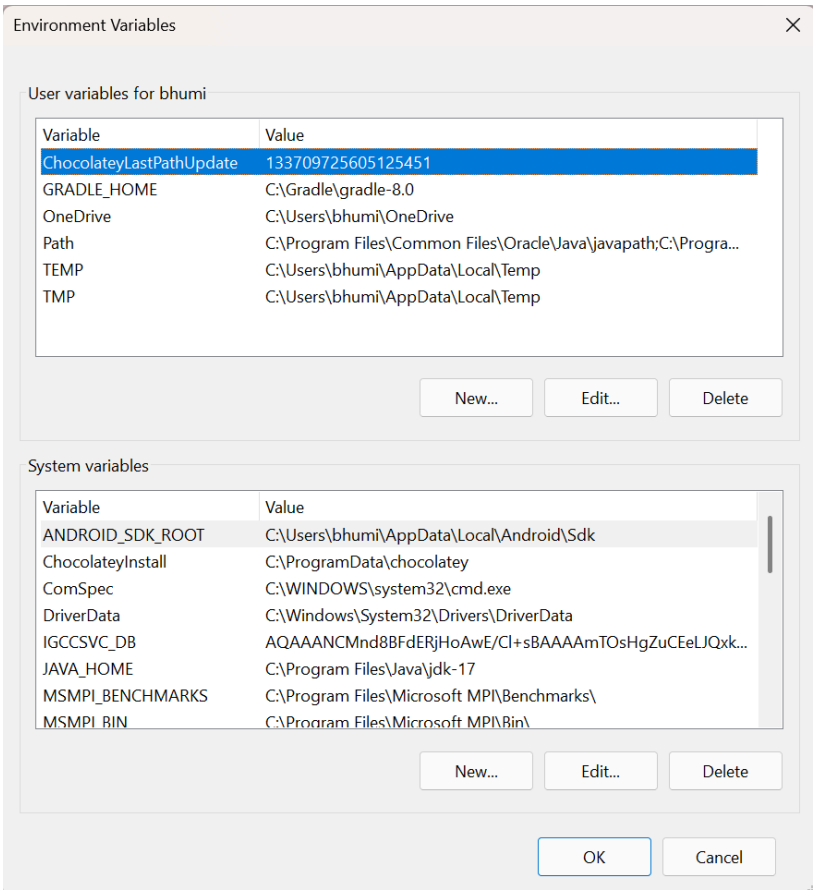


Step 2: Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will find the download link for SDK.

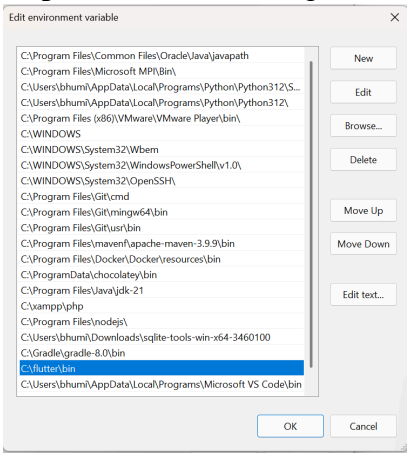
Step 3: When your download is complete, extract the **zip** file and place it in the desired installation folder or location, for example, C: /Flutter.

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

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



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



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

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

```
C:\Windows\System32>flutter
Manage your Flutter app development.

Common commands:

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

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

Usage: flutter <command> [arguments]

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

Available commands:
```

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

```
C:\Windows\System32>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.29.2, on Microsoft Windows [Version 10.0.26100.3476], locale en-IN)
[√] Windows Version (11 Home Single Language 64-bit, 24H2, 2009)
[√] Android toolchain - develop for Android devices (Android SDK version 35.0.1)
[√] Chrome - develop for the web
[√] Visual Studio - develop Windows apps (Visual Studio Build Tools 2019 16.11.44)
[√] Android Studio (version 2024.2)
[√] VS Code (version 1.98.2)
[√] Connected device (3 available)
[√] Network resources

• No issues found!
```

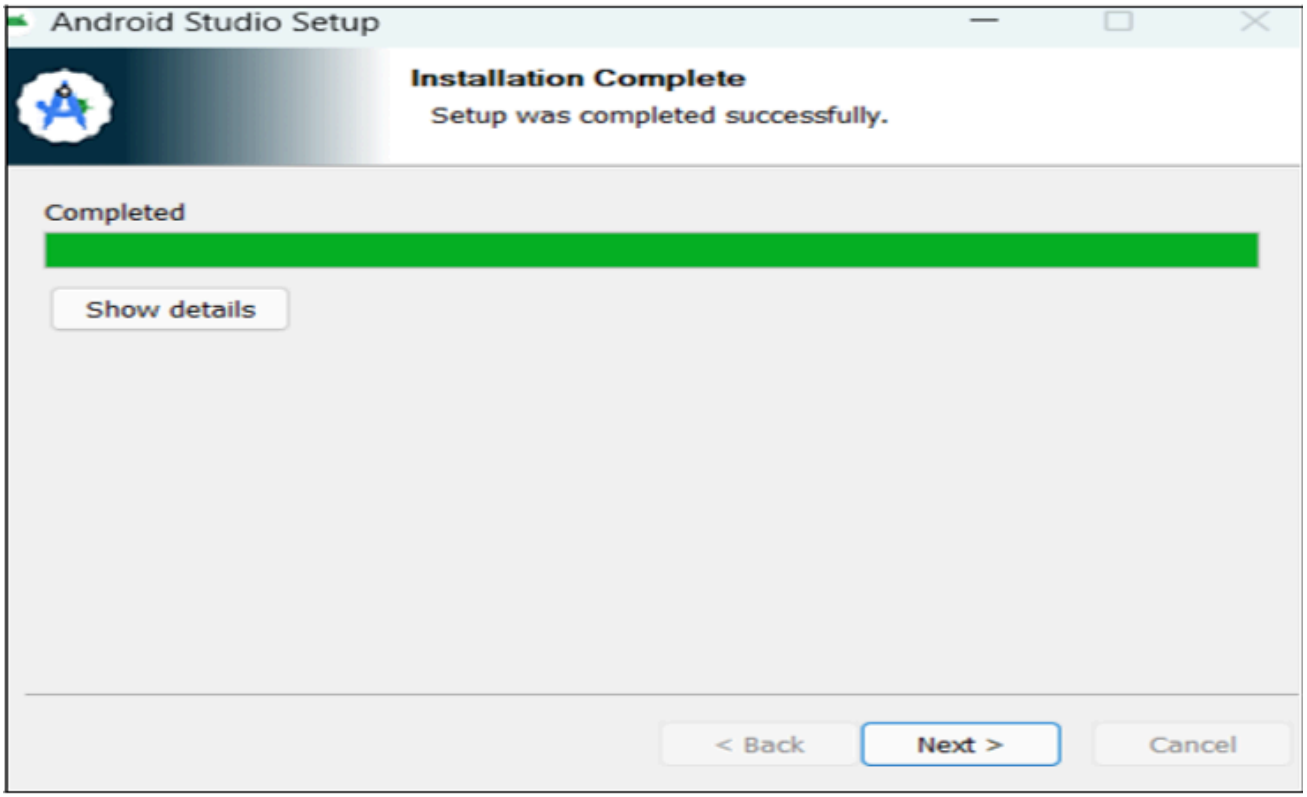
Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

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

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

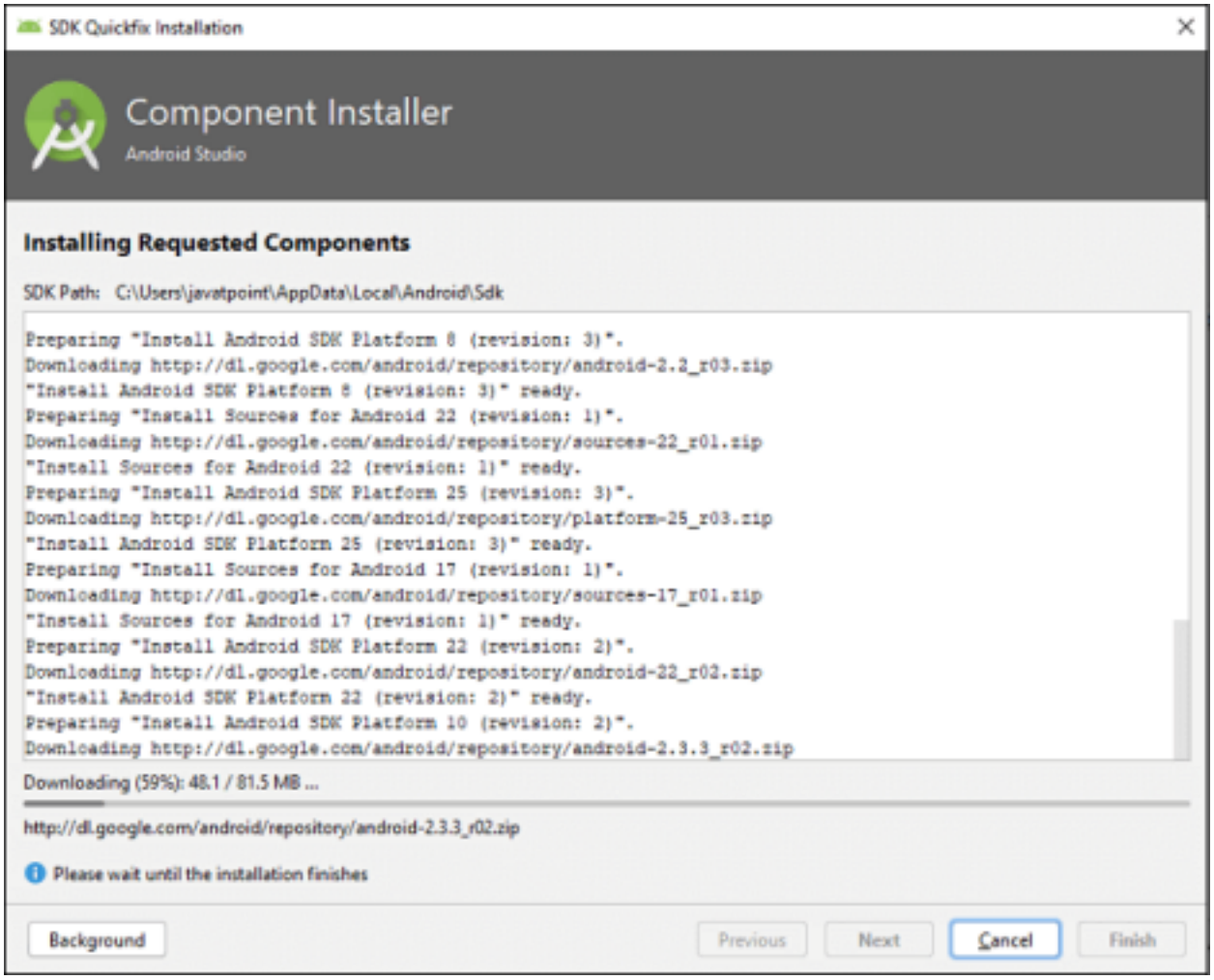
Step 7.2: When the download is complete, open the **.exe** file and run it. You will get the following dialog box.

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



Step 7.4: In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to

choose the 'Don't import Settings option' and click OK. It will start the Android Studio.



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

```
C:\Windows\System32>flutter doctor --android-licenses
Warning: Observed package id 'platform-tools' in inconsistent location 'C:\Users\bhumi\AppData\Local\Android\Sdk\platfor
m-tools.backup' (Expected 'C:\Users\bhumi\AppData\Local\Android\Sdk\platform-tools')
Warning: Already observed package id 'platform-tools' in 'C:\Users\bhumi\AppData\Local\Android\Sdk\platform-tools'. Skip
ping duplicate at 'C:\Users\bhumi\AppData\Local\Android\Sdk\platform-tools.backup'
Warning: Observed package id 'platform-tools' in inconsistent location 'C:\Users\bhumi\AppData\Local\Android\Sdk\platfor
m-tools.backup' (Expected 'C:\Users\bhumi\AppData\Local\Android\Sdk\platform-tools')
Warning: Already observed package id 'platform-tools' in 'C:\Users\bhumi\AppData\Local\Android\Sdk\platform-tools'. Skip
ping duplicate at 'C:\Users\bhumi\AppData\Local\Android\Sdk\platform-tools.backup'
[=======================================] 100% Computing updates...
All SDK package licenses accepted.
```

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

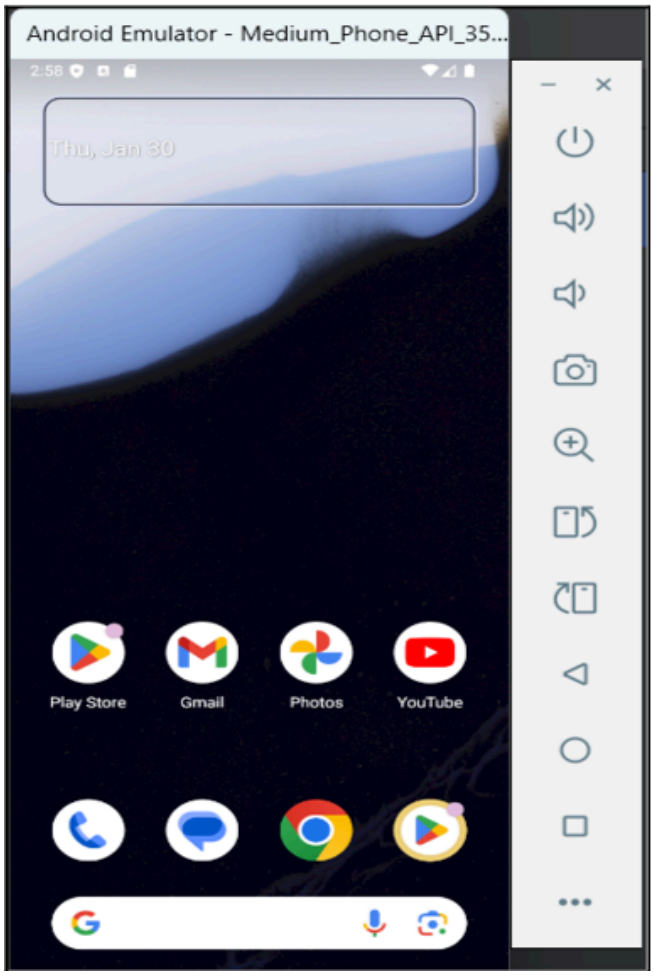
Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.

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

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

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

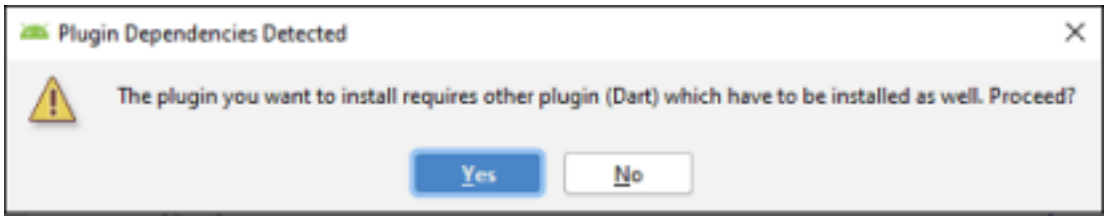
Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as shown below screen.



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

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

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



Step 9.3: Restart the Android Studio.

Conclusion: This experiment demonstrates the complete process of setting up the Flutter development environment. The installation involves multiple components including the Flutter SDK, Android Studio IDE, and plugins that work together to create a functional development environment. The Flutter doctor tool helps identify and fix any missing dependencies. Once properly configured, developers can create Flutter projects and run them on emulators or physical devices, providing a foundation for mobile application development using Flutter's cross-platform capabilities.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
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**
  Text: The Text widget is used to display static or dynamic text. It can be styled with custom fonts, sizes, colors, and more.
- **Icon**
  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**

- **Button Widgets**
  Flutter provides multiple button widgets like ElevatedButton, TextButton, and IconButton to handle user interaction. These widgets can trigger actions when tapped.

- **TextField**
  TextField: Used for user input. You can configure it to accept different types of text, such as email or password.
- **Checkbox, Radio, and Switch**
  Checkbox, Radio, and Switch: Used for boolean selections, allowing users to choose options in forms or settings.

5. **Navigation**
- **Navigator**
  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**
  Routes define the pages of an app, and you can pass data between them using arguments.

6. **Displaying Lists and Grids**
- **ListView**
  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**
  GridView: This widget allows you to display items in a grid format, with configurable row and column layouts.

7. **State Management**
- **setState and Advanced Solutions**
  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.
- **Why State Management Matters**
  Proper state management ensures your UI stays in sync with the underlying data, especially in interactive or dynamic applications.

8. **Theming and Styling**
- **ThemeData**
  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**

- **AnimatedContainer**
  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.
- **Custom Animations**
  You can also create custom animations using AnimationController and Tween.

Code Snippets

Popularmoviespage.dart:

```
import 'package:flutter/material.dart';
import '../services/popularmoviesapi.dart';
import '../screens/movieDetailsPage.dart';
import '../screens/journal.dart';
import '../screens/cinetubepage.dart';

class MovieScreen extends StatefulWidget {
  @override
  _MovieScreenState createState() => _MovieScreenState();
}

class _MovieScreenState extends State<MovieScreen> {
  Future<List<Movie>>? _moviesFuture;

  @override
  void initState() {
    super.initState();
    _moviesFuture = ImdbApiService.fetchMostPopularMovies();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xFF181B20), // Set background color to #181B20
      appBar: AppBar(
        backgroundColor: Colors.grey[900],
        title: const Text(
          'Popular',
          style: TextStyle(color: Colors.white),
        ),
        leading: IconButton(
          icon: const Icon(Icons.menu, color: Colors.white),
          onPressed: () {},
        ),
        actions: [
          IconButton(
            icon: const Icon(Icons.search, color: Colors.white),
            onPressed: () {},
          ),
        ],
        bottom: PreferredSize(
          preferredSize: const Size.fromHeight(48.0),
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceAround,
            children: [
              TextButton(
                onPressed: () {},
                child: const Text('FILMS', style: TextStyle(color: Colors.green)),
              ),
              TextButton(
                onPressed: () {
```

```
              // Navigate to Cinetube page
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => CinetubePage()), // Navigate to CinetubePage
              );
            },
            child: const Text('CINETUBE', style: TextStyle(color: Colors.white)),
          ),
          TextButton(
            onPressed: () {},
            child: const Text('LISTS', style: TextStyle(color: Colors.white)),
          ),
          TextButton(
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => MovieArticlesPage()), // Navigate to MovieArticlesPage
              );
            },
            child: const Text('JOURNAL', style: TextStyle(color: Colors.white)),
          ),

        ],
      ),
    ),
  ),
  body: SingleChildScrollView(
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Container(
          padding: const EdgeInsets.all(16.0),
          color: Colors.grey[850],
          child: Row(
            children: [
              Container(
                padding: const EdgeInsets.all(8.0),
                decoration: BoxDecoration(
                  color: Colors.orange,
                  borderRadius: BorderRadius.circular(5.0),
                ),
                child: const Text(
                  'PRO',
                  style: TextStyle(color: Colors.white),
                ),
              ),
              const SizedBox(width: 16.0),
              const Expanded(
                child: Text(
                  'Remove ads, add profile stats, activity and service filters, favorite streaming services, watchlist alerts and more by
upgrading to Pro.',
                  style: TextStyle(color: Colors.white),
                ),
              ),
            ],
          ),
        ),
        Padding(
          padding: const EdgeInsets.all(16.0),
          child: Text(
            'Popular this week',
            style: TextStyle(color: Colors.white, fontSize: 20.0, fontWeight: FontWeight.bold),
          ),
        ),
        FutureBuilder(
          future: _moviesFuture,
          builder: (context, snapshot) {
            if (snapshot.hasData) {
              return GridView.builder(
                shrinkWrap: true,
```

```
                physics: const NeverScrollableScrollPhysics(),
                gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
                  crossAxisCount: 4, // Number of columns
                  childAspectRatio: 0.6, // Adjust the aspect ratio as needed
                  mainAxisSpacing: 8.0, // Spacing between rows
                  crossAxisSpacing: 8.0, // Spacing between columns
                ),
                itemCount: snapshot.data!.length,
                itemBuilder: (context, index) {
                  return GestureDetector(
                    onTap: () {
                      Navigator.push(
                        context,
                        MaterialPageRoute(
                          builder: (context) => MovieDetailsPage(
                            movieId: snapshot.data![index].id ?? '',
                          ),
                        ),
                      );
                    },
                    child: ClipRRect(
                      borderRadius: BorderRadius.circular(8.0),
                      child: Image.network(
                        snapshot.data![index].primaryImage ?? '',
                        fit: BoxFit.cover,
                      ),
                    ),
                  );
                },
              );
            } else if (snapshot.hasError) {
              return Text('Error: ${snapshot.error}', style: const TextStyle(color: Colors.white));
            } else {
              return const Center(child: CircularProgressIndicator(color: Colors.white));
            }
          },
        ),
      ],
    ),
  ),
  floatingActionButton: FloatingActionButton(
    backgroundColor: Colors.green,
    onPressed: () {},
    child: const Icon(Icons.add, color: Colors.white),
  ),
 );
 }
}
```

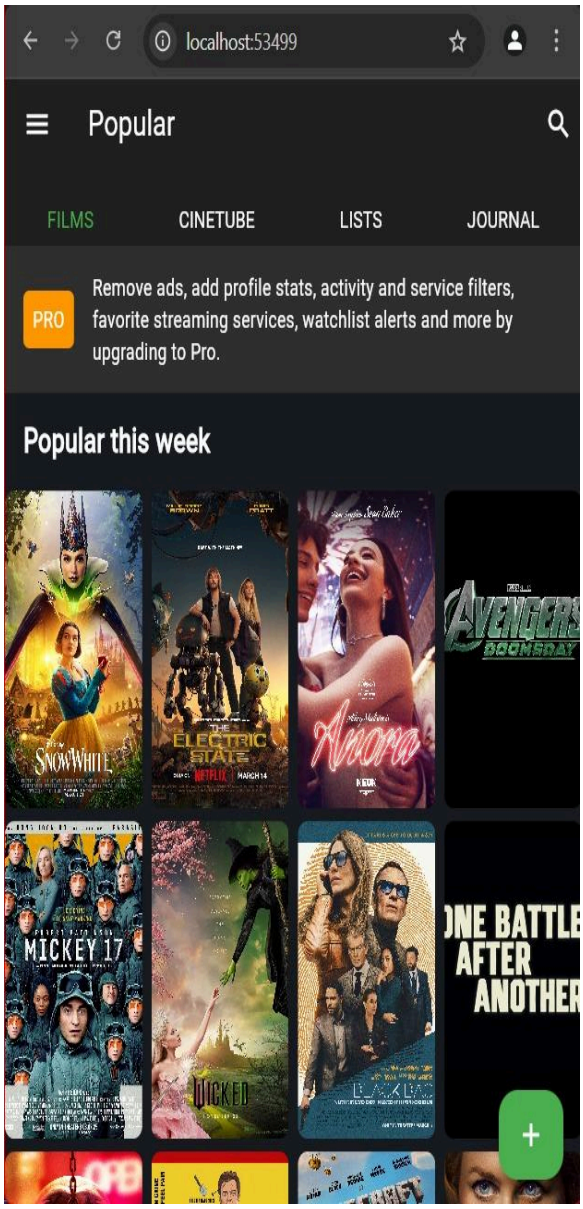
Journal.dart:

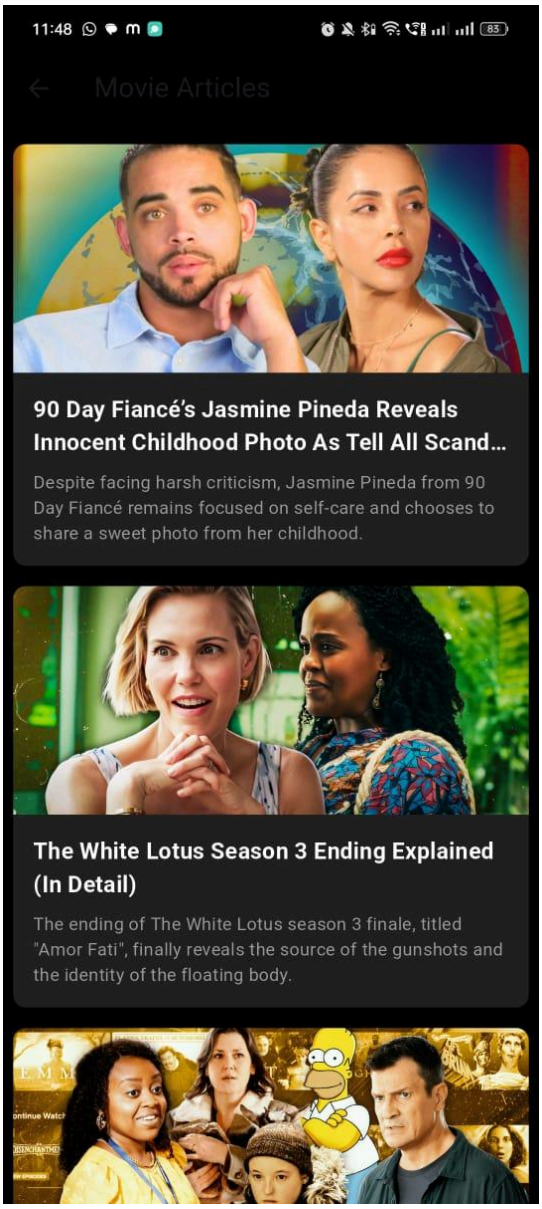
```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'package:dart_rss/dart_rss.dart';
import 'package:html/parser.dart';
import '../../widgets/moviecard.dart';

class MovieArticlesPage extends StatefulWidget {
 @override
 _MovieArticlesPageState createState() => _MovieArticlesPageState();
}

class _MovieArticlesPageState extends State<MovieArticlesPage> {
 List<RssItem> _articles = [];

 @override
 void initState() {
  super.initState();
  fetchArticles();
 }
```

```
  // Fetch RSS feed from Screen Rant
  Future<void> fetchArticles() async {
    try {
      final response = await http.get(Uri.parse('https://screenrant.com/feed/'));
      if (response.statusCode == 200) {
        final feed = RssFeed.parse(response.body);
        setState(() {
          _articles = feed.items;
        });
      } else {
        print("Failed to load RSS feed: ${response.statusCode}");
      }
    } catch (e) {
      print("Error fetching articles: $e");
    }
  }

  // Extract the first image from the article's description
  String extractImage(RssItem article) {
    if (article.media?.contents?.isNotEmpty ?? false) {
      return article.media!.contents!.first.url ?? '';
    } else if (article.enclosure?.url != null) {
      return article.enclosure!.url!;
    }

    // Try to extract image from description as fallback
    var document = parse(article.description ?? '');
    var imgTag = document.getElementsByTagName('img');
    return imgTag.isNotEmpty ? imgTag.first.attributes['src'] ?? '' : '';
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Movie Articles"),
        backgroundColor: Colors.black,
      ),
      backgroundColor: Colors.black,
      body: _articles.isEmpty
          ? const Center(child: CircularProgressIndicator(color: Colors.white))
          : ListView.builder(
        padding: const EdgeInsets.all(8),
        itemCount: _articles.length,
        itemBuilder: (context, index) {
          var article = _articles[index];
          String imageUrl = extractImage(article);

          // Extract clean text description
          String cleanDescription = "";
          try {
            var document = parse(article.description ?? "");
            cleanDescription = document.body?.text ?? "No Description";
          } catch (e) {
            cleanDescription = "No Description";
          }

          return MovieArticleCard(
            title: article.title ?? "No Title",
            description: cleanDescription,
            imageUrl: imageUrl.isNotEmpty ? imageUrl : "https://via.placeholder.com/200",
            url: article.link ?? "#",
          );
        },
      ),
    );
  }
}
```

Shraeyaa dhaigude 15

Screenshots:





Conclusion:

In conclusion, Flutter offers a comprehensive set of widgets that are crucial for building intuitive and responsive user interfaces. Common widgets such as TextField, IconButton, ElevatedButton, Card, and Checkbox play a vital role in shaping the user interface of mobile applications. These widgets enable users to input data, perform actions, and view content in a structured way, all while improving the overall user experience. By understanding and effectively utilizing these widgets, developers can create mobile applications that are clean, efficient, and user-friendly.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
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/images/my_image.png
```

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

Example of using an asset icon:
Image.asset('assets/icons/my_icon.png')

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

Example of adding custom fonts in pubspec.yaml:

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

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')
```

Code Snippets:

Moviedetailspage.dart:

```
import 'package:flutter/material.dart';
import 'package:cached_network_image/cached_network_image.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import '../../utils/api_constants.dart';
import '../../models/movie.dart';
import '../models/review.dart';
import '../../services/movieDetails.dart';
import '../../widgets/image.dart';

class MovieDetailsPage extends StatefulWidget {
  final String movieId;
  const MovieDetailsPage({Key? key, required this.movieId}) : super(key: key);
  @override
  _MovieDetailsPageState createState() => _MovieDetailsPageState();
}
class _MovieDetailsPageState extends State<MovieDetailsPage> with SingleTickerProviderStateMixin {
  late Future<Movie?> _movieFuture;
  final MoviedetailsService _apiService = MoviedetailsService();
  final _reviewController = TextEditingController();
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  late TabController _tabController;
  @override
  void initState() {
    super.initState();
    _movieFuture = _apiService.getMovieDetails(widget.movieId);
    _tabController = TabController(length: 5, vsync: this);
  }
  @override
  void dispose() {
    _tabController.dispose();
    super.dispose();
  }
  Widget _buildRatingBar(double rating, {int maxRating = 5}) {
    return Row(
      mainAxisSize: MainAxisSize.min,
      children: [
        Row(
          children: List.generate(maxRating, (index) {
            double fillPercentage = (rating - index) > 1 ? 1 : (rating - index) < 0 ? 0 : (rating - index);
            return Container(
```

```
            width: 12,
            height: 8,
            margin: const EdgeInsets.symmetric(horizontal: 1),
            decoration: BoxDecoration(
              color: Colors.greenAccent.withOpacity(fillPercentage),
              borderRadius: BorderRadius.circular(1),
            ),
          );
        }),
      ),
      const SizedBox(width: 8),
      Text(
        rating.toStringAsFixed(1),
        style: const TextStyle(
          color: Colors.white,
          fontWeight: FontWeight.bold,
        ),
      ),
    ],
  );
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.grey[900],
    body: FutureBuilder<Movie?>(
      future: _movieFuture,
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return const Center(child: CircularProgressIndicator());
        } else if (snapshot.hasError) {
          return Center(child: Text('Error: ${snapshot.error}', style: const TextStyle(color: Colors.white)));
        } else if (snapshot.hasData && snapshot.data != null) {
          final movie = snapshot.data!;
          return CustomScrollView(
            slivers: [
              // App Bar with Movie Poster
              SliverAppBar(
                expandedHeight: 220.0,
                floating: false,
                pinned: true,
                backgroundColor: Colors.grey[950],
                flexibleSpace: FlexibleSpaceBar(
                  title: Text(
                    movie.primaryTitle,
                    style: const TextStyle(
                      color: Colors.white,
                      fontWeight: FontWeight.bold,
                      fontSize: 16,
                    ),
                  ),
                  background: Stack(
                    fit: StackFit.expand,
                    children: [
                      movie.primaryImage.isNotEmpty
                          ? CachedNetworkImage(
                        imageUrl: movie.primaryImage,
                        fit: BoxFit.cover,
                        placeholder: (context, url) => const Center(child: CircularProgressIndicator()),
                        errorWidget: (context, url, error) => const Icon(Icons.error),
```

```
          )
              : Container(color: Colors.grey.shade800),
          // Gradient overlay for better text visibility
          const DecoratedBox(
            decoration: BoxDecoration(
              gradient: LinearGradient(
                begin: Alignment.topCenter,
                end: Alignment.bottomCenter,
                colors: [Colors.transparent, Colors.black87],
              ),
            ),
          ),
        ],
      ),
    ),
  ),

  // Movie Info
  SliverToBoxAdapter(
    child: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          // Director info
          Text(
            'DIRECTED BY',
            style: TextStyle(
              color: Colors.grey.shade400,
              fontSize: 12,
            ),
          ),
          Text(
            movie.directors.isNotEmpty ? movie.directors.map((d) => d.fullName).join(", ") : 'Unknown',
            style: const TextStyle(
              color: Colors.white,
              fontSize: 14,
              fontWeight: FontWeight.bold,
            ),
          ),

          const SizedBox(height: 12),

          // Year, runtime
          Row(
            children: [
              Text(
                '${movie.startYear}',
                style: const TextStyle(color: Colors.white),
              ),
              const SizedBox(width: 16),
              Text(
                '${movie.runtimeMinutes} mins',
                style: const TextStyle(color: Colors.white),
              ),
              const Spacer(),
              TextButton(
                onPressed: () {},
                child: const Text('TRAILER', style: TextStyle(color: Colors.white)),
              ),
            ],
```

```
),

const SizedBox(height: 16),

// Movie description
Text(
  movie.description,
  style: TextStyle(
    color: Colors.grey.shade300,
    fontSize: 14,
  ),
),

const SizedBox(height: 24),

// Remove ads button
Center(
  child: OutlinedButton(
    onPressed: () {},
    style: OutlinedButton.styleFrom(
      side: BorderSide(color: Colors.grey.shade600),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(4),
      ),
    ),
    child: Text(
      'REMOVE ADS',
      style: TextStyle(color: Colors.grey.shade400),
    ),
  ),
),

const SizedBox(height: 24),

// Ratings
const Text(
  'RATINGS',
  style: TextStyle(
    color: Colors.white,
    fontSize: 14,
    fontWeight: FontWeight.bold,
  ),
),

const SizedBox(height: 8),

Row(
  children: [
    Expanded(
      child: Container(
        height: 40,
        decoration: BoxDecoration(
          color: Colors.grey.shade900,
          borderRadius: BorderRadius.circular(4),
        ),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: List.generate(5, (index) {
            return Container(
              width: 6,
              height: 20,
```

```
                color: Colors.grey.shade700,
              );
            }),
          ),
        ),
      ),
      const SizedBox(width: 12),
      _buildRatingBar(movie.averageRating / 2), // Converting to 5-star scale
    ],
  ),

  const SizedBox(height: 16),

  // Rating & Review Button
  OutlinedButton.icon(
    onPressed: () {},
    icon: const Icon(Icons.star_border, color: Colors.white),
    label: const Text('Rate, log, review, add to list + more', style: TextStyle(color: Colors.white)),
    style: OutlinedButton.styleFrom(
      side: const BorderSide(color: Colors.white),
      minimumSize: const Size(double.infinity, 40),
    ),
  ),

  const SizedBox(height: 24),

  // Where to watch
  const Text(
    'Where to watch',
    style: TextStyle(
      color: Colors.white,
      fontSize: 14,
      fontWeight: FontWeight.bold,
    ),
  ),

  const SizedBox(height: 12),

  // Streaming services
  Row(
    children: [
      Container(
        padding: const EdgeInsets.all(8),
        decoration: BoxDecoration(
          color: Colors.blue,
          borderRadius: BorderRadius.circular(4),
        ),
        child: Image.network(
          'https://api.dicebear.com/7.x/initials/svg?seed=PV',
          width: 20,
          height: 20,
          color: Colors.white,
        ),
      ),
      const SizedBox(width: 8),
      Container(
        padding: const EdgeInsets.all(8),
        decoration: BoxDecoration(
          color: Colors.blue.shade900,
          borderRadius: BorderRadius.circular(4),
        ),
```

```
      child: Image.network(
       'https://api.dicebear.com/7.x/initials/svg?seed=MAX',
       width: 20,
       height: 20,
       color: Colors.white,
      ),
     ),
   ],
),

const SizedBox(height: 24),

// Members, Reviews, Lists buttons
Row(
  children: [
    Expanded(
      child: Container(
        height: 60,
        decoration: BoxDecoration(
          color: Colors.green,
          borderRadius: BorderRadius.circular(4),
        ),
        child: const Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(Icons.person, color: Colors.white),
            Text('Members', style: TextStyle(color: Colors.white, fontSize: 12)),
            Text('463k', style: TextStyle(color: Colors.white, fontSize: 12)),
          ],
        ),
      ),
    ),
    const SizedBox(width: 8),
    Expanded(
      child: Container(
        height: 60,
        decoration: BoxDecoration(
          color: Colors.grey.shade800,
          borderRadius: BorderRadius.circular(4),
        ),
        child: const Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(Icons.comment, color: Colors.white),
            Text('Reviews', style: TextStyle(color: Colors.white, fontSize: 12)),
            Text('2.8k', style: TextStyle(color: Colors.white, fontSize: 12)),
          ],
        ),
      ),
    ),
    const SizedBox(width: 8),
    Expanded(
      child: Container(
        height: 60,
        decoration: BoxDecoration(
          color: Colors.blue,
          borderRadius: BorderRadius.circular(4),
        ),
        child: const Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
```

```
            Icon(Icons.list, color: Colors.white),
            Text('Lists', style: TextStyle(color: Colors.white, fontSize: 12)),
            Text('804', style: TextStyle(color: Colors.white, fontSize: 12)),
          ],
        ),
      ),
    ),
  ],
),

const SizedBox(height: 24),

// Related news
Padding(
  padding: const EdgeInsets.only(bottom: 8.0),
  child: Text(
    'RELATED NEWS (1 OF 2)',
    style: TextStyle(color: Colors.grey.shade400, fontSize: 12),
  ),
),

// News card
Container(
  decoration: BoxDecoration(
    color: Colors.grey.shade900,
    borderRadius: BorderRadius.circular(8),
  ),
  child: Row(
    children: [
      ClipRRect(
        borderRadius: const BorderRadius.only(
          topLeft: Radius.circular(8),
          bottomLeft: Radius.circular(8),
        ),
        child: Image.network(
          'https://api.dicebear.com/7.x/shapes/svg?seed=news',
          width: 80,
          height: 80,
        ),
      ),
      const SizedBox(width: 12),
      Expanded(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            const Text(
              'When Jack Met Letterboxd',
              style: TextStyle(
                color: Colors.white,
                fontWeight: FontWeight.bold,
              ),
            ),
            const SizedBox(height: 4),
            Text(
              'Four favorites with the director and cast of Companion.',
              style: TextStyle(color: Colors.grey.shade400, fontSize: 12),
            ),
          ],
        ),
      ),
    ],
```

```
            ),
          ),
        ],
      ),
    ),
  ),

  // Tabs
  SliverPersistentHeader(
    delegate: _SliverAppBarDelegate(
      TabBar(
        controller: _tabController,
        labelColor: Colors.white,
        unselectedLabelColor: Colors.grey,
        indicatorColor: Colors.white,
        tabs: const [
          Tab(text: 'CAST + CREW'),
          Tab(text: 'DETAILS'),
          Tab(text: 'GENRE'),
          Tab(text: 'RELEASES'),
          Tab(text: ''),
        ],
      ),
    ),
    pinned: true,
  ),

  // Tab Content - Cast
  SliverToBoxAdapter(
    child: SizedBox(
      height: 150,
      child: ListView.builder(
        scrollDirection: Axis.horizontal,
        itemCount: movie.cast.length,
        itemBuilder: (context, index) {
          final actor = movie.cast[index];
          return Container(
            width: 80,
            margin: const EdgeInsets.all(8),
            child: Column(
              children: [
                CircleAvatar(
                  radius: 30,
                  backgroundColor: Colors.grey.shade800,
                  child: Text(
                    actor.fullName.isNotEmpty ? actor.fullName[0] : '?',
                    style: const TextStyle(color: Colors.white),
                  ),
                ),
                const SizedBox(height: 4),
                Text(
                  actor.fullName,
                  style: const TextStyle(color: Colors.white, fontSize: 12),
                  maxLines: 2,
                  textAlign: TextAlign.center,
                  overflow: TextOverflow.ellipsis,
                ),
                Text(
                  actor.characters?.join(', ') ?? 'Role',
                  style: TextStyle(color: Colors.grey.shade400, fontSize: 10),
                  maxLines: 1,
```

```
                overflow: TextOverflow.ellipsis,
              ),
            ],
          ),
        );
      },
    ),
  ),
),

// Crew Section
SliverToBoxAdapter(
  child: Padding(
    padding: const EdgeInsets.only(left: 16.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        const Padding(
          padding: EdgeInsets.symmetric(vertical: 8.0),
          child: Text(
            'CREW',
            style: TextStyle(
              color: Colors.white,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
        SizedBox(
          height: 100,
          child: ListView.builder(
            scrollDirection: Axis.horizontal,
            itemCount: movie.directors.length,
            itemBuilder: (context, index) {
              final director = movie.directors[index];
              return Container(
                width: 80,
                margin: const EdgeInsets.all(8),
                child: Column(
                  children: [
                    CircleAvatar(
                      radius: 25,
                      backgroundColor: Colors.redAccent,
                      child: Text(
                        director.fullName.isNotEmpty ? director.fullName[0] : '?',
                        style: const TextStyle(color: Colors.white),
                      ),
                    ),
                    const SizedBox(height: 4),
                    Text(
                      director.fullName,
                      style: const TextStyle(color: Colors.white, fontSize: 12),
                      maxLines: 2,
                      textAlign: TextAlign.center,
                      overflow: TextOverflow.ellipsis,
                    ),
                    Text(
                      'Director',
                      style: TextStyle(color: Colors.grey.shade400, fontSize: 10),
                      maxLines: 1,
                      overflow: TextOverflow.ellipsis,
                    ),
```

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

// Reviews Section
SliverToBoxAdapter(
 child: Padding(
  padding: const EdgeInsets.all(16.0),
  child: Column(
   crossAxisAlignment: CrossAxisAlignment.start,
   children: [
    const Padding(
     padding: EdgeInsets.symmetric(vertical: 8.0),
     child: Text(
      'REVIEWS',
      style: TextStyle(
       color: Colors.white,
       fontWeight: FontWeight.bold,
      ),
     ),
    ),

    // Reviews from Firebase
    StreamBuilder<QuerySnapshot>(
     stream: _firestore.collection('reviews').where('movieId', isEqualTo: widget.movieId).snapshots(),
     builder: (context, snapshot) {
      if (snapshot.hasError) {
       return Text('Error: ${snapshot.error}', style: const TextStyle(color: Colors.white));
      }

      if (snapshot.connectionState == ConnectionState.waiting) {
       return const Center(child: CircularProgressIndicator());
      }

      final reviews = snapshot.data?.docs ?? [];

      if (reviews.isEmpty) {
       return const Text(
        'No reviews yet. Be the first to review!',
        style: TextStyle(color: Colors.grey),
       );
      }

      return Column(
       children: reviews.map((doc) {
        final data = doc.data() as Map<String, dynamic>;
        return Container(
         margin: const EdgeInsets.symmetric(vertical: 8),
         child: Row(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
           CircleAvatar(
            backgroundColor: Colors.grey.shade800,
            child: Text(
```

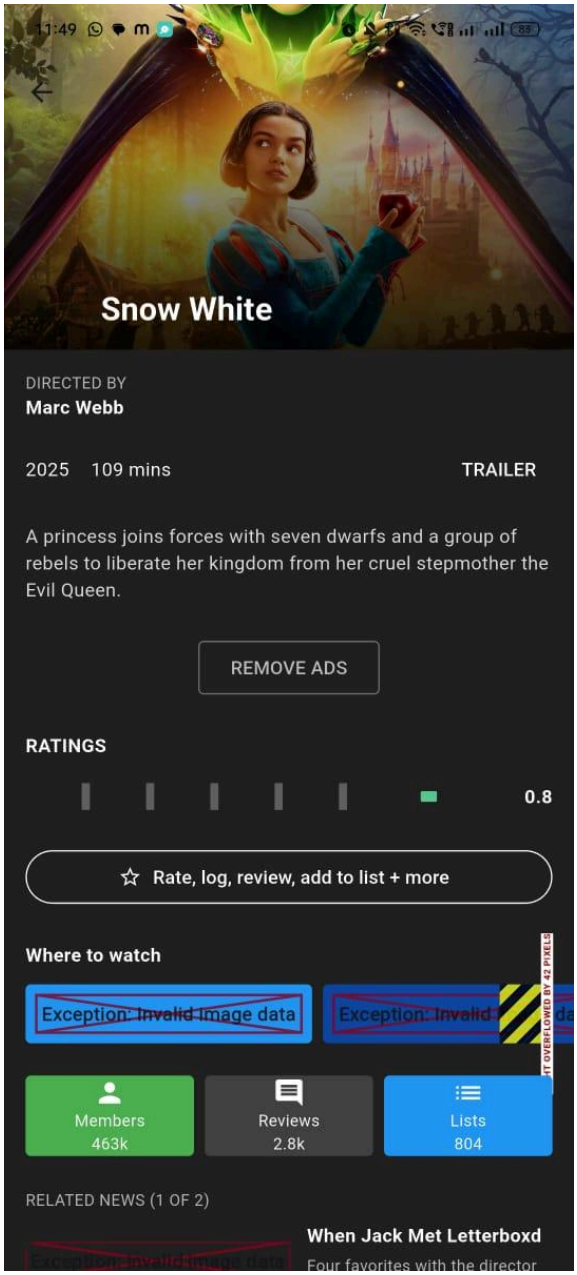
```
                (data['user'] as String).isNotEmpty ? (data['user'] as String)[0] : '?',
                style: const TextStyle(color: Colors.white),
              ),
            ),
            const SizedBox(width: 12),
            Expanded(
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    data['user'] ?? 'Anonymous',
                    style: const TextStyle(
                      color: Colors.white,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                  const SizedBox(height: 4),
                  Text(
                    data['text'] ?? '',
                    style: TextStyle(color: Colors.grey.shade300),
                  ),
                ],
              ),
            ),
            _buildRatingBar(4.5),
          ],
        ),
      );
    }).toList(),
  );
 },
),

const SizedBox(height: 16),

// Add Review
TextField(
 controller: _reviewController,
 style: const TextStyle(color: Colors.white),
 decoration: InputDecoration(
  labelText: 'Add your review',
  labelStyle: TextStyle(color: Colors.grey.shade400),
  enabledBorder: OutlineInputBorder(
   borderSide: BorderSide(color: Colors.grey.shade700),
  ),
  focusedBorder: const OutlineInputBorder(
   borderSide: BorderSide(color: Colors.white),
  ),
 ),
 maxLines: 3,
),
const SizedBox(height: 8),
ElevatedButton(
 onPressed: () {
  _addReview(movie.id);
 },
 style: ElevatedButton.styleFrom(
  backgroundColor: Colors.green,
  minimumSize: const Size(double.infinity, 40),
 ),
 child: const Text('SUBMIT REVIEW'),
```

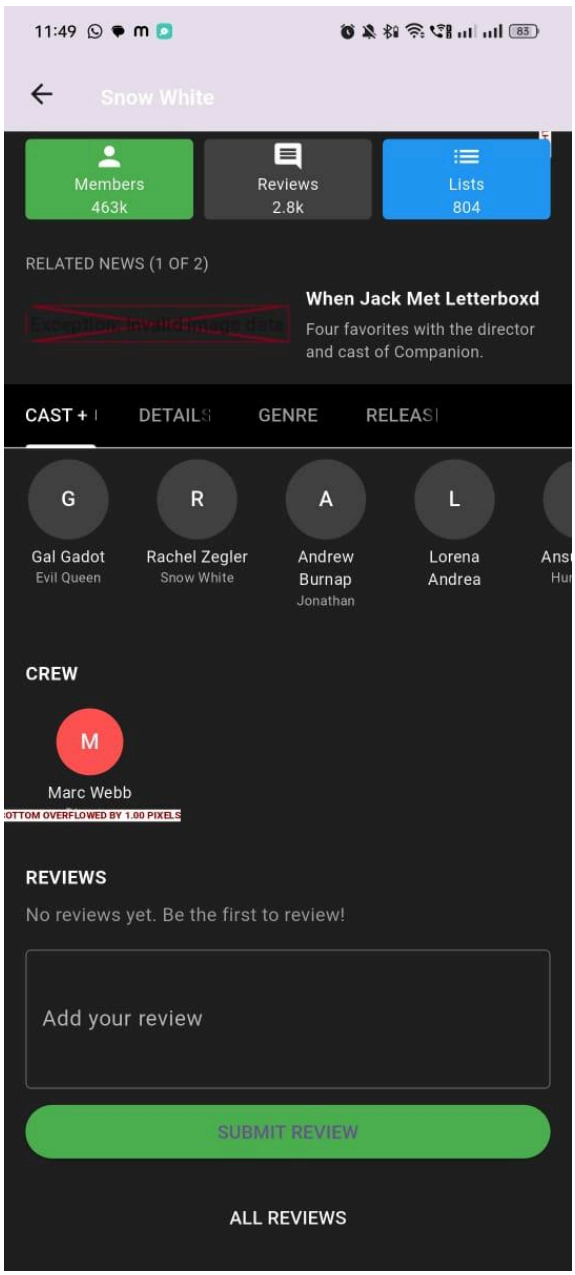
```
                    ),

                    const SizedBox(height: 16),

                    // View All Reviews
                    Center(
                      child: TextButton(
                        onPressed: () {},
                        child: const Text(
                          'ALL REVIEWS',
                          style: TextStyle(color: Colors.white),
                        ),
                      ),
                    ),
                  ],
                ),
              ),
            ),
          ],
        );
      } else {
        return const Center(
          child: Text(
            'Movie not found',
            style: TextStyle(color: Colors.white),
          ),
        );
      }
    },
  ),
 );
}

 Future<void> _addReview(String movieId) async {
  if (_reviewController.text.isNotEmpty) {
   await _firestore.collection('reviews').add({
    'movieId': movieId,
    'text': _reviewController.text,
    'user': 'Anonymous', // Replace with actual user authentication
    'timestamp': FieldValue.serverTimestamp(),
    'rating': 4.5, // This would ideally come from a rating widget
   });
   _reviewController.clear();
   setState(() {}); // Refresh the UI
  }
 }
}

// Helper class for the persistent header
class _SliverAppBarDelegate extends SliverPersistentHeaderDelegate {
 final TabBar _tabBar;

 _SliverAppBarDelegate(this._tabBar);

 @override
 double get minExtent => _tabBar.preferredSize.height;

 @override
 double get maxExtent => _tabBar.preferredSize.height;

 @override
```

SHRAEYAA DHAIGUDE D15A 15

```
  Widget build(BuildContext context, double shrinkOffset, bool overlapsContent) {
    return Container(
      color: Colors.black,
      child: _tabBar,
    );
  }

  @override
  bool shouldRebuild(_SliverAppBarDelegate oldDelegate) {
    return false;
  }
}
```

Screenshots:





Conclusion:

In conclusion, incorporating icons, images, and fonts into a Flutter app significantly enhances the user interface and user experience. By correctly managing assets through the pubspec.yaml file and understanding how to use built-in and custom resources, developers can build visually appealing and functional applications. Whether using asset images, network images, Material icons, SVGs, or custom fonts, Flutter provides flexible and efficient ways to integrate these elements. Mastery of these features empowers developers to design clean, modern, and engaging mobile applications that align with branding and usability standards.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

EXPERIMENT 4

AIM : To create an interactive Form using a form widget.

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.

Code:

LOGIN.DART:

```
import 'package:flutter/material.dart';
import '../services/auth_service.dart';
import '../widgets/custom_text_field.dart';
import '../widgets/small_button.dart';

class LoginPage extends StatelessWidget {
  final TextEditingController _usernameController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  final AuthService _authService = AuthService();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        children: [
          // Background Image
          Container(
            decoration: BoxDecoration(
              image: DecorationImage(
                image: AssetImage('assets/Connected.jpg'),
                fit: BoxFit.cover,
              ),
            ),
          ),
          // Dark overlay
          Container(
            color: Colors.black.withOpacity(0.3),
          ),
          // Login Form Positioned at Bottom
          Align(
            alignment: Alignment.bottomCenter,
            child: Container(
              padding: EdgeInsets.fromLTRB(16, 12, 16, 8),
              width: double.infinity,
              decoration: BoxDecoration(
                color: Color(0xFF1B2228),
                borderRadius: BorderRadius.only(
                  topLeft: Radius.circular(12),
                  topRight: Radius.circular(12),
                ),
              ),
              child: Column(
                mainAxisSize: MainAxisSize.min,
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    "Sign in to Letterboxd",
                    style: TextStyle(
                      fontSize: 20,
                      fontWeight: FontWeight.w500,
                      color: Colors.white,
                    ),
                  ),
                  SizedBox(height: 12),
```

```
            CustomTextField(hintText: "Username", controller: _usernameController),
            CustomTextField(hintText: "Password", isPassword: true, controller: _passwordController),
            SizedBox(height: 8),
            Row(
              children: [
                SmallButton(
                  text: "JOIN",
                  color: Color(0xFF38444E),
                  onPressed: () {
                    Navigator.pushNamed(context, '/signup');
                  },
                ),
                SizedBox(width: 8),
                SmallButton(
                  text: "RESET PASSWORD",
                  color: Color(0xFF38444E),
                  onPressed: () {
                    // Add reset password functionality
                  },
                ),
                SizedBox(width: 8),
                SmallButton(
                  text: "GO",
                  color: Colors.green,
                  onPressed: () async {
                    final user = await _authService.signIn(
                      _usernameController.text,
                      _passwordController.text,
                    );
                    if (user != null) {
                      // Fetch the username from Firestore
                      final username = await _authService.getUsername(user.uid);
                      if (username != null) {
                        Navigator.pushReplacementNamed(context, '/home');
                      } else {
                        ScaffoldMessenger.of(context).showSnackBar(
                          SnackBar(content: Text('Failed to fetch user data.')),
                        );
                      }
                    } else {
                      ScaffoldMessenger.of(context).showSnackBar(
                        SnackBar(content: Text('Login failed. Please try again.')),
                      );
                    }
                  },
                ),
              ],
            ),
            SizedBox(height: 8),
            Center(
              child: Text(
                "Artwork from Mitchell vs the Machines",
                style: TextStyle(
                  color: Colors.grey[400],
                  fontSize: 11,
                ),
              ),
            ),
          ],
        ),
      ),
    ),
```

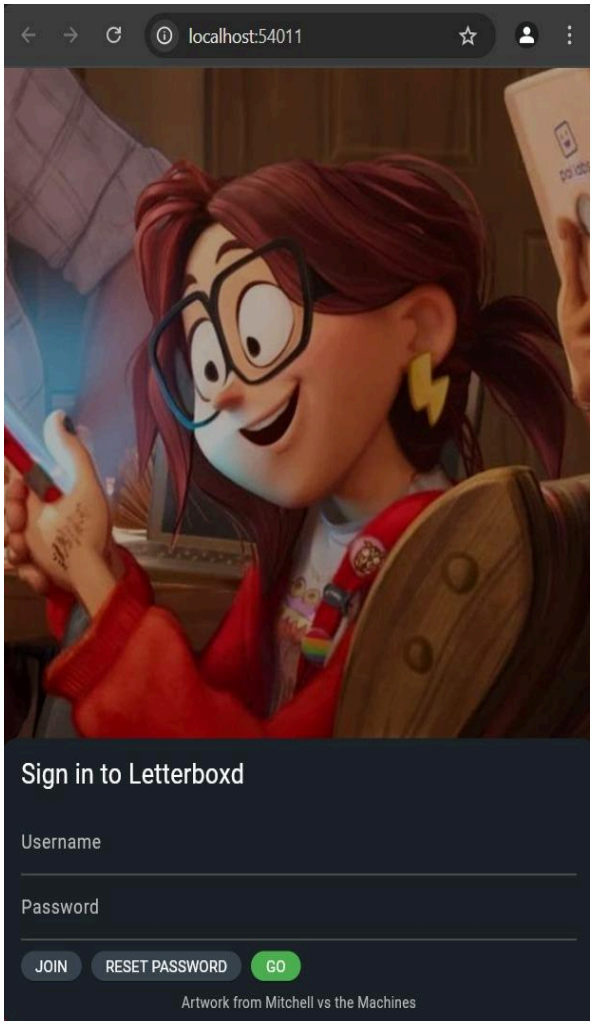
```
        ],
      ),
    );
  }
}
```

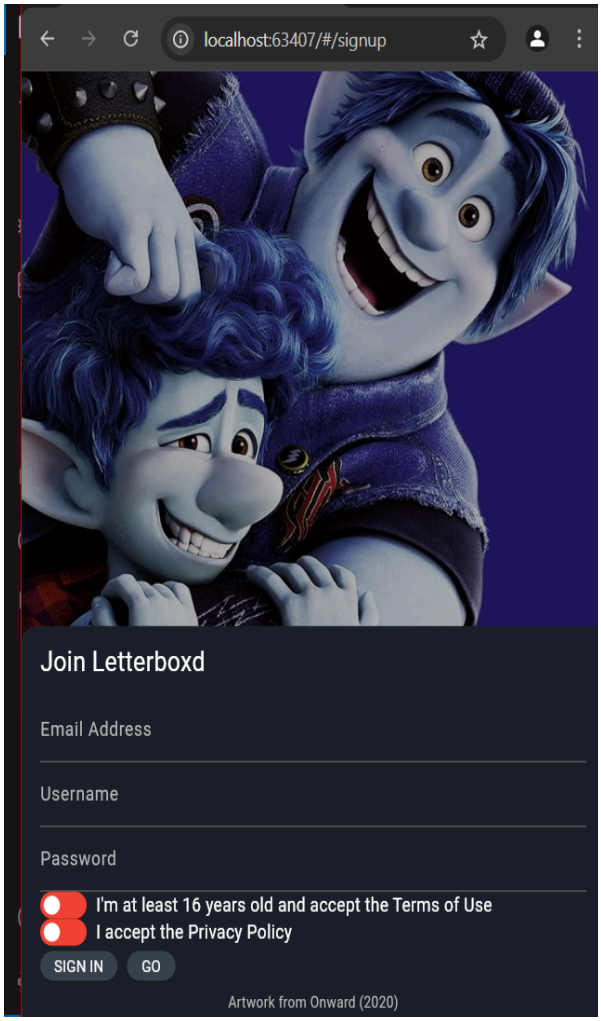
SIGNUP.DART:

```
import 'package:flutter/material.dart';
import '../services/auth_service.dart';
import '../widgets/custom_text_field.dart';
import '../widgets/small_button.dart';
import '../widgets/toggle_switch.dart';

class SignupPage extends StatelessWidget {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _usernameController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  final AuthService _authService = AuthService();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        children: [
          // Background Image
          Container(
            decoration: BoxDecoration(
              image: DecorationImage(
                image: AssetImage('assets/onward.jpg'),
                fit: BoxFit.cover,
              ),
            ),
          ),
          // Dark overlay
          Container(
            color: Colors.black.withOpacity(0.3),
          ),
          // Signup Form Positioned at Bottom
          Align(
            alignment: Alignment.bottomCenter,
            child: Container(
              padding: EdgeInsets.fromLTRB(16, 12, 16, 8),
              decoration: BoxDecoration(
                color: Color(0xFF1B2228),
                borderRadius: BorderRadius.only(
                  topLeft: Radius.circular(12),
                  topRight: Radius.circular(12),
                ),
              ),
              child: Column(
                mainAxisSize: MainAxisSize.min,
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    "Join Letterboxd",
                    style: TextStyle(
                      fontSize: 20,
                      fontWeight: FontWeight.w500,
                      color: Colors.white,
                    ),
                  ),
                  SizedBox(height: 12),
```

```
                CustomTextField(hintText: "Email Address", controller: _emailController),
                CustomTextField(hintText: "Username", controller: _usernameController),
                CustomTextField(hintText: "Password", isPassword: true, controller: _passwordController),
                ToggleSwitch(text: "I'm at least 16 years old and accept the Terms of Use"),
                ToggleSwitch(
                   text: "I accept the Privacy Policy "
                ),
                SizedBox(height: 4),
                Row(
                  children: [
                    SmallButton(
                      text: "SIGN IN",
                      color: Color(0xFF38444E),
                      onPressed: () {
                        Navigator.pushNamed(context, '/login');
                      },
                    ),
                    SizedBox(width: 8),
                    SmallButton(
                      text: "GO",
                      color: Color(0xFF38444E),
                      onPressed: () async {
                        final user = await _authService.signUp(
                          _emailController.text,
                          _usernameController.text,
                          _passwordController.text,
                        );
                        if (user != null) {
                          Navigator.pushReplacementNamed(context, '/home');
                        } else {
                          ScaffoldMessenger.of(context).showSnackBar(
                            SnackBar(content: Text('Sign-up failed. Please try again.')),
                          );
                        }
                      },
                    ),
                  ],
                ),
                SizedBox(height: 8),
                Center(
                  child: Text(
                    "Artwork from Onward (2020)",
                    style: TextStyle(
                      color: Colors.grey[400],
                      fontSize: 11,
                    ),
                  ),
                ),
              ],
            ),
          ),
        ),
      ],
    ),
  );
 }
}
```

Screenshots:





Conclusion:

In this experiment, we successfully created an interactive form using Flutter's Form and TextFormField widgets. The form included multiple input fields such as full name, email, password, and confirm password. We used a GlobalKey<FormState> to manage the form's state and implemented custom validators for each field to ensure data integrity and proper user input.

The experiment demonstrated how Flutter allows developers to build responsive and user-friendly forms with built-in validation, easy data handling, and clean UI design. This approach is highly efficient for applications that require user registration, login, or any kind of data input.

Overall, this experiment enhanced our understanding of form validation, form state management, and user interaction handling in Flutter.

SHRAEYAA DHAIGUDE D15A 15

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

EXPERIMENT 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).

SHRAEYAA DHAIGUDE D15A 15

Code:

Cinetube.dart:

```
import 'package:flutter/material.dart';
import 'package:youtube_player_iframe/youtube_player_iframe.dart';
import 'package:url_launcher/url_launcher.dart';
import 'package:youtube_explode_dart/youtube_explode_dart.dart';

class CinetubePage extends StatefulWidget {
  @override
  _CinetubePageState createState() => _CinetubePageState();
}

class _CinetubePageState extends State<CinetubePage> {
  Future<List<Video>>? _videosFuture;
  final YoutubeExplode _yt = YoutubeExplode();
  bool _isLoading = false;

  @override
  void initState() {
    super.initState();
    _fetchVideos();
  }

  @override
  void dispose() {
    _yt.close();
    super.dispose();
  }

  Future<void> _fetchVideos() async {
    if (_isLoading) return;

    setState(() {
      _isLoading = true;
    });

    try {
      final results = await _yt.search.search(
          "movie film cinema cinematography directors interviews cinephile review documentary storytelling"
      );

      setState(() {
        _videosFuture = Future.value(results);
        _isLoading = false;
      });
    } catch (e) {
      setState(() {
        _videosFuture = Future.error(e);
        _isLoading = false;
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Cinetube'),
        actions: [
          IconButton(
            icon: const Icon(Icons.refresh),
            onPressed: _fetchVideos,
          ),
        ],
      ),
      body: FutureBuilder<List<Video>>(
```

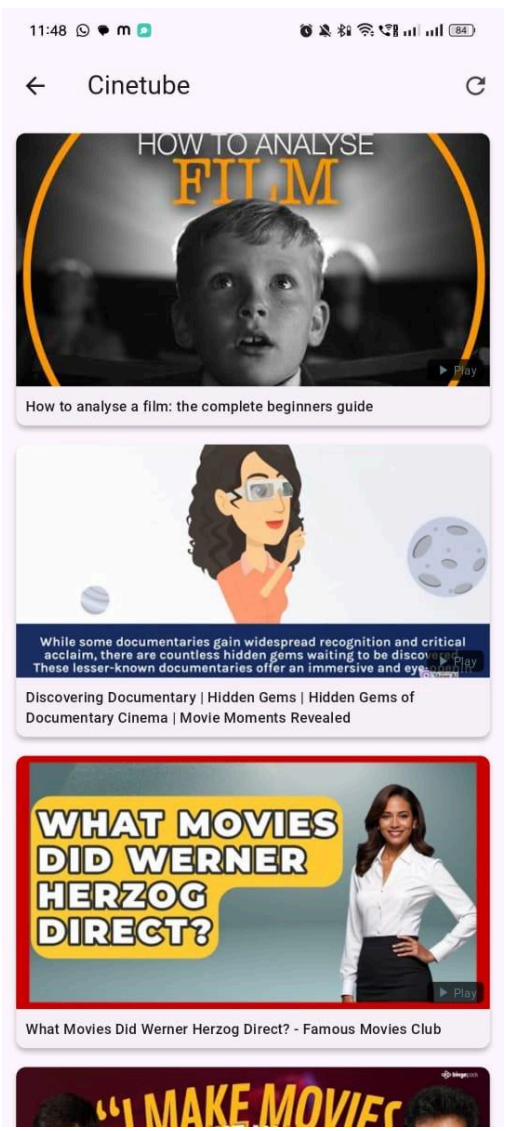
```
future: _videosFuture,
builder: (context, snapshot) {
  if (snapshot.hasData) {
    final videos = snapshot.data!;

    if (videos.isEmpty) {
      return const Center(
        child: Text('No videos found. Try a different search.'),
      );
    }

    return GridView.builder(
      padding: const EdgeInsets.all(8.0),
      gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 1,
        childAspectRatio: 1.6,
        mainAxisSpacing: 8.0,
        crossAxisSpacing: 8.0,
      ),
      itemCount: videos.length,
      itemBuilder: (context, index) {
        final video = videos[index];
        final videoId = video.id.value;

        // Get the highest resolution thumbnail available
        final thumbnailUrl = video.thumbnails.highResUrl;

        return GestureDetector(
          onTap: () async {
            final url = 'https://www.youtube.com/watch?v=$videoId';
            if (await canLaunchUrl(Uri.parse(url))) {
              await launchUrl(Uri.parse(url));
            } else {
              // If launching URL fails, open in player page
              Navigator.push(
                context,
                _createRoute(videoId),
              );
            }
          },
          child: Card(
            elevation: 3,
            clipBehavior: Clip.antiAlias,
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(8),
            ),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Expanded(
                  child: Stack(
                    fit: StackFit.expand,
                    children: [
                      thumbnailUrl.isNotEmpty
                          ? Image.network(
                        thumbnailUrl,
                        fit: BoxFit.cover,
                        errorBuilder: (context, error, stackTrace) =>
                            Container(
                              color: Colors.grey[900],
                              child: const Icon(Icons.movie, size: 50),
                            ),
                      )
                          : Container(
                        color: Colors.grey[950],
                        child: const Icon(Icons.movie, size: 50),
                      ),
                      Positioned(
                        right: 5,
                        bottom: 5,
```

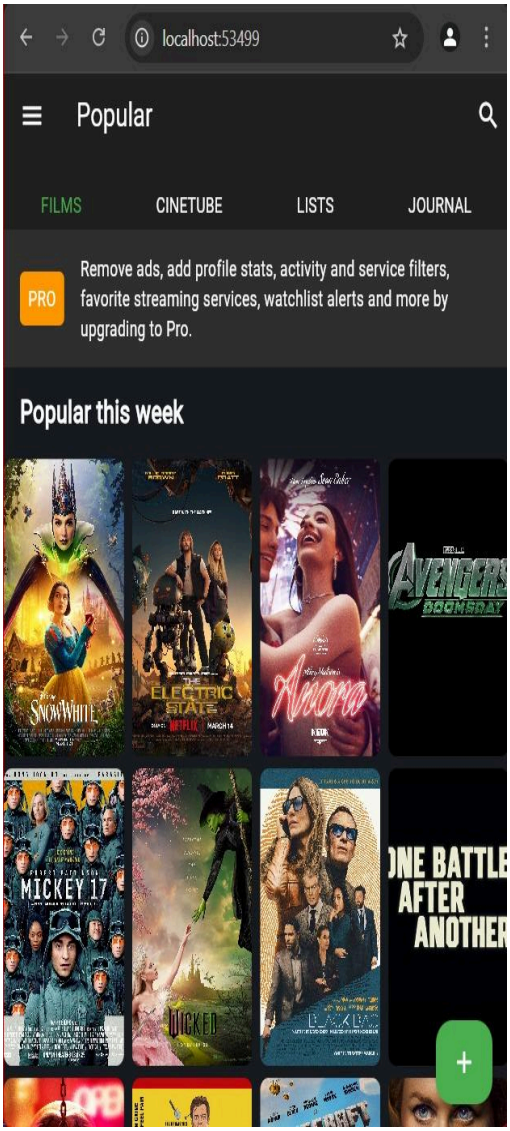
```
                    child: Container(
                      padding: const EdgeInsets.symmetric(
                        horizontal: 6,
                        vertical: 2,
                      ),
                      decoration: BoxDecoration(
                        color: Colors.black.withOpacity(0.7),
                        borderRadius: BorderRadius.circular(4),
                      ),
                      child: Row(
                        mainAxisSize: MainAxisSize.min,
                        children: const [
                          Icon(
                            Icons.play_arrow,
                            color: Colors.blueGrey,
                            size: 14,
                          ),
                          SizedBox(width: 2),
                          Text(
                            'Play',
                            style: TextStyle(
                              color: Colors.blueGrey,
                              fontSize: 10,
                            ),
                          ),
                        ],
                      ),
                    ),
                  ),
                ],
              ),
            ),
            Padding(
              padding: const EdgeInsets.all(8.0),
              child: Text(
                video.title,
                maxLines: 2,
                overflow: TextOverflow.ellipsis,
                style: const TextStyle(
                  fontSize: 12,
                  fontWeight: FontWeight.w500,
                ),
              ),
            ),
          ],
        ),
      ),
    );
  },
 );
} else if (snapshot.hasError) {
  return Center(
    child: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          const Icon(
            Icons.error_outline,
            color: Colors.red,
            size: 48,
          ),
          const SizedBox(height: 16),
          Text(
            'Error: ${snapshot.error}',
            style: const TextStyle(fontSize: 16),
            textAlign: TextAlign.center,
          ),
          const SizedBox(height: 16),
          ElevatedButton(
```

```
                  onPressed: _fetchVideos,
                  child: const Text('Retry'),
                ),
              ],
            ),
          ),
        );
      } else {
        return const Center(
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              CircularProgressIndicator(),
              SizedBox(height: 16),
              Text('Loading videos...'),
            ],
          ),
        );
      }
    },
  ),
 );
}

Route _createRoute(String videoId) {
 return PageRouteBuilder(
  pageBuilder: (context, animation, secondaryAnimation) => VideoPlayerPage(videoId: videoId),
  transitionsBuilder: (context, animation, secondaryAnimation, child) {
   const begin = Offset(0.0, 1.0);
   const end = Offset.zero;
   const curve = Curves.ease;

   var tween = Tween(begin: begin, end: end).chain(CurveTween(curve: curve));

   return SlideTransition(position: animation.drive(tween), child: child);
  },
 );
}
}

class VideoPlayerPage extends StatefulWidget {
 final String videoId;

 const VideoPlayerPage({Key? key, required this.videoId}) : super(key: key);

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

class _VideoPlayerPageState extends State<VideoPlayerPage> {
 late YoutubePlayerController _controller;

 @override
 void initState() {
  super.initState();
  _controller = YoutubePlayerController.fromVideoId(
   videoId: widget.videoId,
   autoPlay: true,
   params: const YoutubePlayerParams(
    showControls: true,
    showFullscreenButton: true,
    mute: false,
   ),
  );
 }

 @override
 void dispose() {
  _controller.close();
  super.dispose();
```

```
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Video Player'),
      actions: [
        IconButton(
          icon: const Icon(Icons.open_in_browser),
          onPressed: () async {
            final url = 'https://www.youtube.com/watch?v=${widget.videoId}';
            if (await canLaunchUrl(Uri.parse(url))) {
              await launchUrl(Uri.parse(url));
            } },  ), ], ),
    body: Center(
      child: YoutubePlayer(
        controller: _controller,
        aspectRatio: 16 / 9,
      ),),);  }}
```

Screenshots:





Conclusion:

This experiment provided a comprehensive understanding of navigation and routing in Flutter, which is essential for creating multi-screen applications. We learned how to implement smooth transitions between different screens using both direct and named routes. The use of the Navigator widget helped us manage route stacks effectively, improving the overall navigation flow of the app.

Additionally, the integration of gesture detection using widgets like GestureDetector enabled us to build more interactive and responsive user interfaces. By responding to user actions such as taps and swipes, we were able to make the application more dynamic, intuitive, and user-friendly. These features are crucial for enhancing the user experience and building polished, production-ready apps.

Overall, this experiment strengthened our understanding of managing screen navigation and incorporating user gestures, both of which are fundamental for modern mobile app development in Flutter.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

EXPERIMENT 6

AIM: To Set Up Firebase with Flutter for iOS and Android Apps.

Theory

Introduction to Firebase and Flutter Integration

Firebase is a comprehensive platform developed by Google, designed to help developers build high-quality applications for both mobile and web. It provides essential services such as real-time databases, authentication, cloud storage, hosting, and much more. One of the most widely used Firebase services is the Firebase Realtime Database, which is a NoSQL cloud database that allows data to be stored and synced in real-time across all connected devices.

Flutter, on the other hand, is an open-source UI software development kit created by Google, which allows developers to build natively compiled applications for mobile, web, and desktop from a single codebase. Its rich set of pre-designed widgets and powerful tools makes Flutter an attractive option for developing visually appealing and performant applications.

Integrating Firebase with Flutter allows developers to leverage the full potential of Firebase services in their applications. By using Firebase's Realtime Database, Flutter apps can achieve features such as real-time data synchronization, secure authentication, and cloud-based storage. This combination enables developers to create powerful, scalable, and feature-rich mobile and web applications.

Firebase Realtime Database Overview

Firebase Realtime Database is a cloud-hosted NoSQL database that stores data in a JSON-like format. The key characteristic of this database is its real-time synchronization feature, meaning that any changes made to the database are instantly reflected on all clients (i.e., devices) connected to it.

This makes it an ideal solution for applications that require frequent updates and need to maintain synchronized data across multiple users or devices, such as messaging apps, social media platforms, or collaborative tools.

The Firebase Realtime Database is structured as a tree of data, where each node in the tree can contain key-value pairs. This structure allows for easy data retrieval and modification. Firebase's real-time capabilities enable apps to immediately receive updates to the data whenever it changes, without the need to refresh or reload the page.

Additionally, the database supports offline data persistence, meaning that even if the user's device loses its internet connection, the app can still function by using the locally cached data.

Setting Up Firebase in Flutter

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

1. Creating a Firebase Project:

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

2. Integrating Firebase SDK in Flutter:

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

3. Initializing Firebase:

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

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

Code:

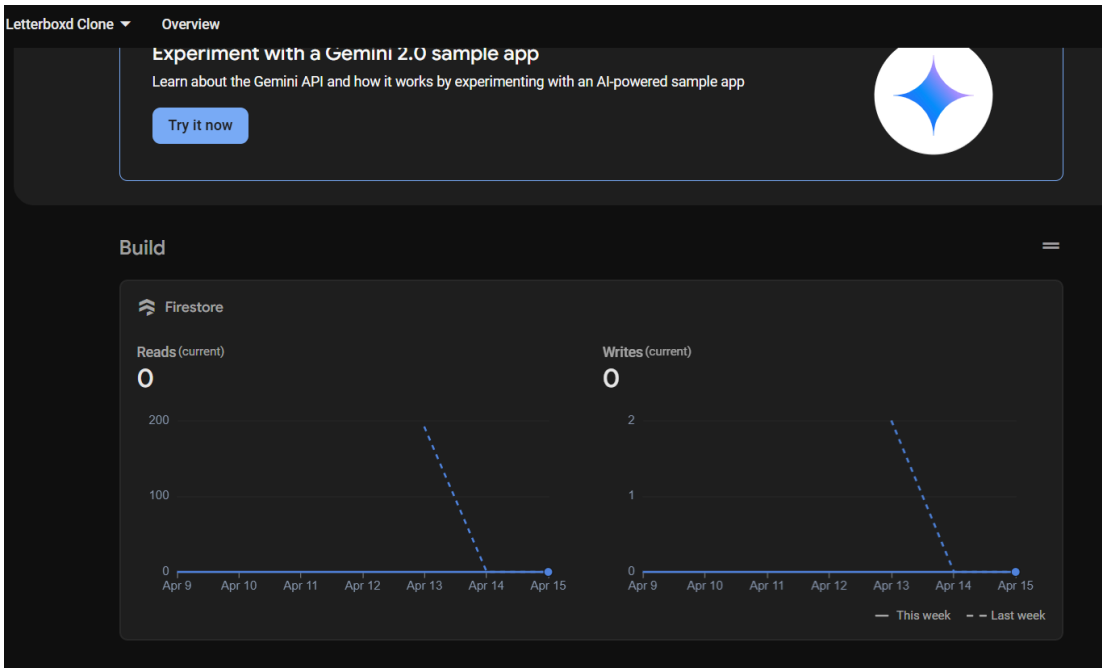
AUTHSERVICE.DART:

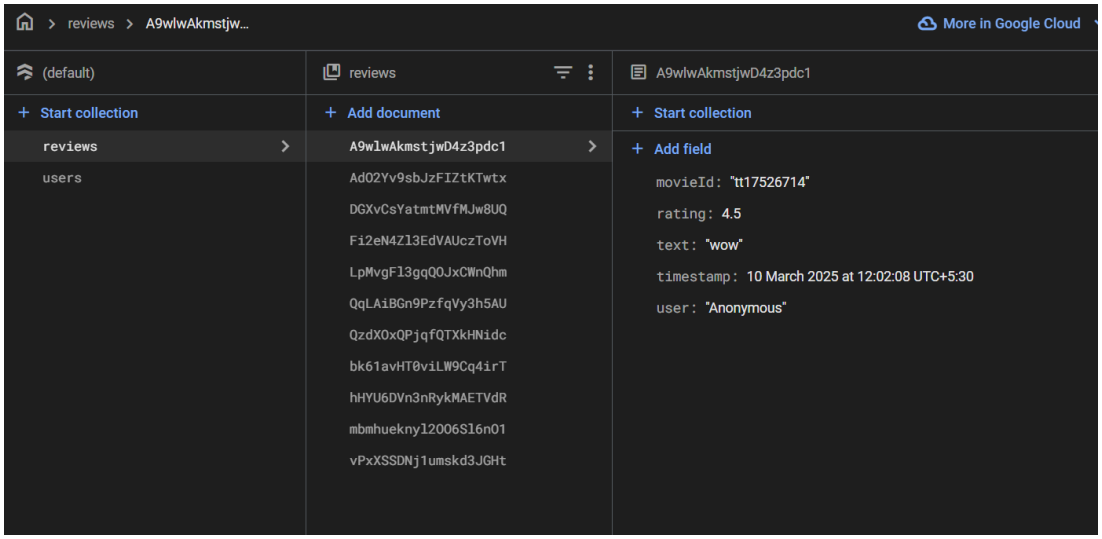
```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
class AuthService {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  // Sign up with email, username, and password
  Future<User?> signUp(String email, String username, String password) async {
    try {
      // Step 1: Create user with email and password
      UserCredential userCredential = await _auth.createUserWithEmailAndPassword(
        email: email,
        password: password,
      );

      // Step 2: Save additional user data (username) to Firestore
      await _firestore.collection('users').doc(userCredential.user!.uid).set({
        'email': email,
        'username': username,
        'createdAt': Timestamp.now(),
      });

      return userCredential.user;
    } catch (e) {
      print("Error during sign-up: $e");
      return null;
    }
  }

  // Sign in with username and password
  Future<User?> signIn(String username, String password) async {
    try {
      // Step 1: Retrieve email associated with the username from Firestore
      QuerySnapshot userQuery = await _firestore
          .collection('users')
          .where('username', isEqualTo: username)
          .limit(1)
          .get();

      if (userQuery.docs.isEmpty) {
        print("No user found with this username.");
        return null;
      }
```

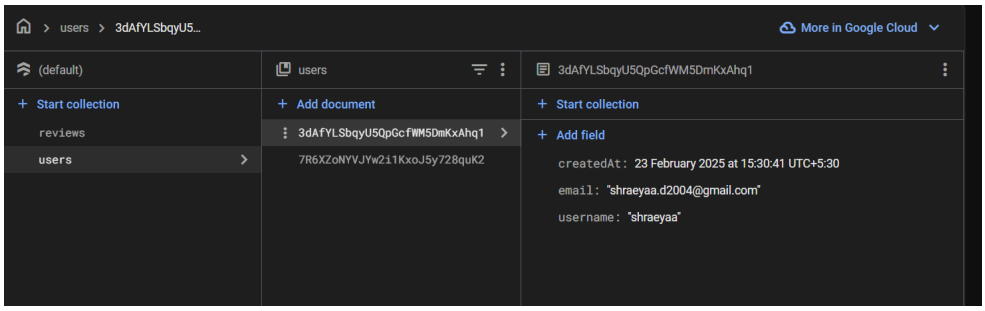
```
      String email = userQuery.docs.first['email'];

      // Step 2: Sign in with email and password
      UserCredential userCredential = await _auth.signInWithEmailAndPassword(
        email: email,
        password: password,
      );

      return userCredential.user;
    } catch (e) {
      print("Error during sign-in: $e");
      return null;
    }
  }

  // Get the current user's username from Firestore
  Future<String?> getUsername(String uid) async {
    try {
      DocumentSnapshot userDoc = await _firestore.collection('users').doc(uid).get();
      return userDoc['username'];
    } catch (e) {
      print("Error fetching username: $e");
      return null;
    }
  }

  // Sign out
  Future<void> signOut() async {
    await _auth.signOut();
  }

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

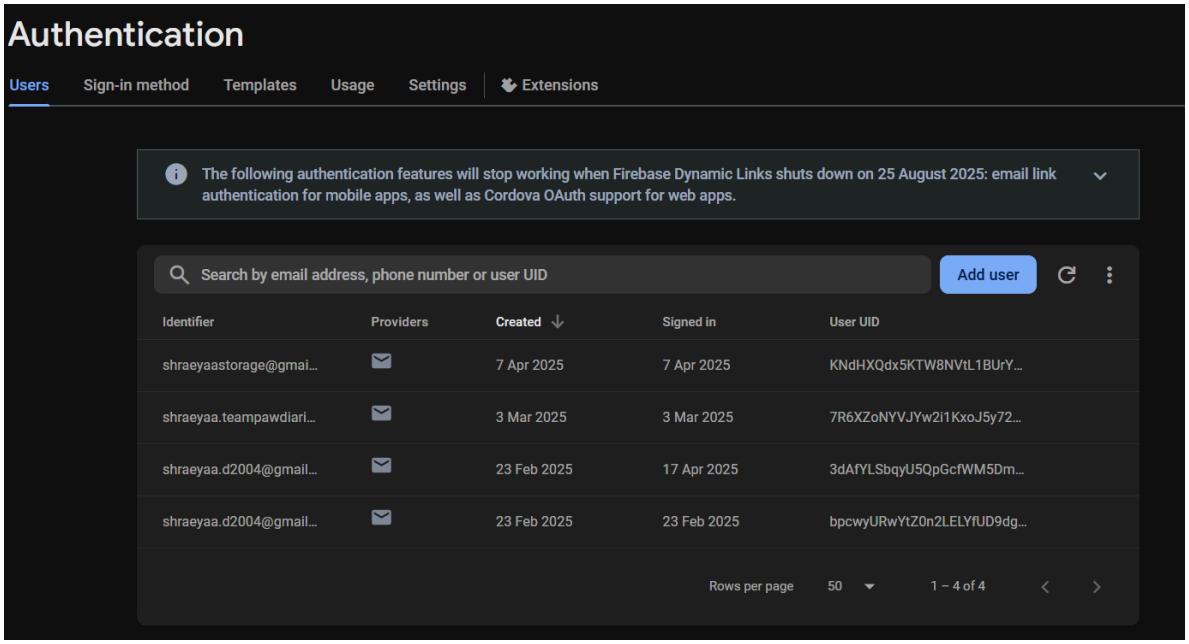
SHRAEYAA DHAIGUDE D15A 15

Screenshots:









Conclusion:
This experiment provided comprehensive, hands-on experience in integrating Firebase services into a Flutter application for both Android and iOS platforms. By incorporating Firebase Authentication, Firestore, and other cloud-based services, it enabled the development of a robust backend system that supports secure user registration, data storage, and real-time data updates. The integration allowed the app to benefit from Firebase's powerful and scalable infrastructure, which is essential for building modern, responsive, and data-driven mobile applications. Additionally, working with Firebase helped in understanding how cloud-based services can enhance the overall functionality, reliability, and user experience of cross-platform mobile apps.

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.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

Ria Chaudhari D15A07
Eesha Chavan D15A08
Shraeyaa Dhaigude D15A15

PWA EXP7

Aim: To write meta data of your 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:

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

1. Native Experience

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

2. Ease of Access

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

3. Faster Services
PWAs can cache the data and serve the user with text stylesheets, images, and other web content even before the page loads completely. This lowers the waiting time for the end-users and helps the brands improve the user engagement and retention rate, which eventually adds value to their business.

Features of the Progressive Web App
The main features are:
● Progressive
They work for every user, regardless of the browser chosen because they are built at the
base with progressive improvement principles.
● Responsive
They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can
later become available.
● App-like
They behave with the user as if they were native apps, in terms of interaction and navigation.
● Updated
Information is always up-to-date thanks to the data update process offered by service
workers.
● Secure
Exposed over HTTPS protocol to prevent the connection from displaying information or
altering the contents.
● Searchable
They are identified as “applications” and are indexed by search engines.
● Reactivable
Make it easy to reactivate the application thanks to capabilities such as web notifications.
● Installable
They allow the user to “save” the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to
face all the steps and problems related to the use of the app store.
● Linkable
Easily shared via URL without complex installations.

Code:

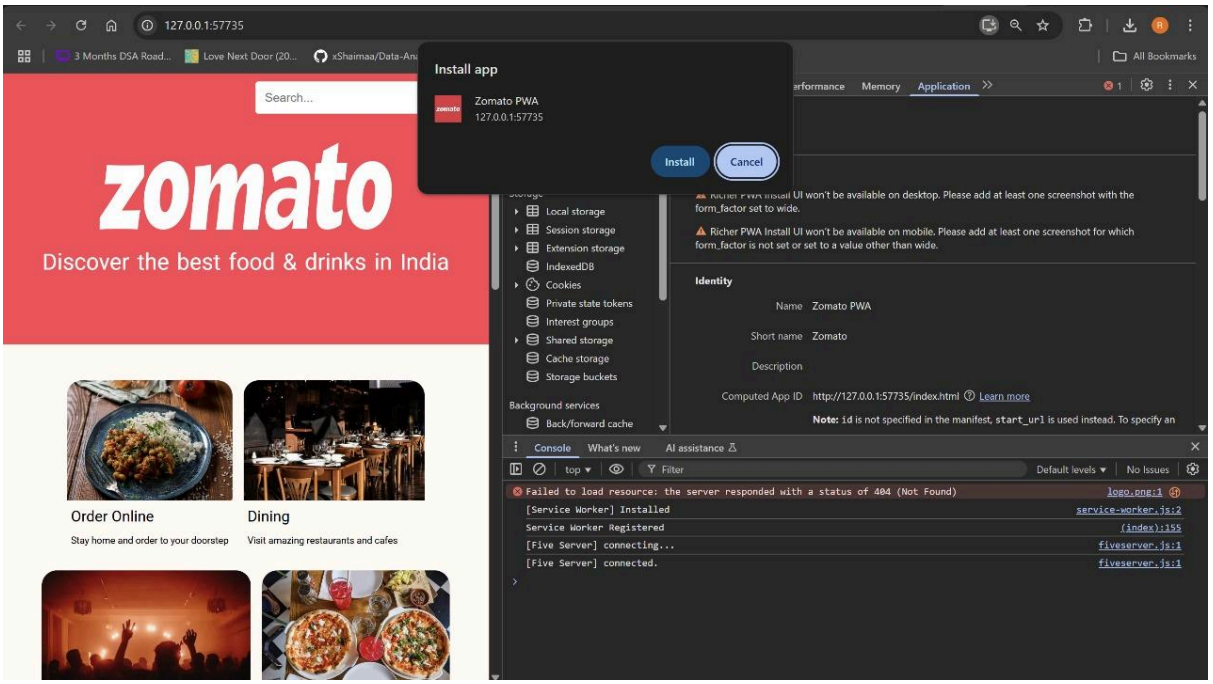
//manifest.json

```
{
    "name": "Zomato PWA",
    "short_name": "Zomato",
    "start_url": "./index.html",
    "display": "standalone",
    "background_color": "#ffffff",
    "theme_color": "#ff5a5f",
    "orientation": "portrait",
    "scope": "./",
    "icons": [
      {
        "src": "images/icon-192x192.png",
        "sizes": "192x192",
        "type": "image/png"
      },
      {
        "src": "images/icon-512x512.png",
        "sizes": "512x512",
        "type": "image/png"
      }
    ]
  }
```

//service-worker.js

```
self.addEventListener('install', function(e) {
    console.log('[Service Worker] Installed');
  });

  self.addEventListener('fetch', function(e) {
    console.log('[Service Worker] Fetch intercepted:', e.request.url);
  });
```

Output:



Shortcut added to home screen

Conclusion:

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

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.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Ria Chaudhari D15A07
Eesha Chavan D15A08
Shraeyaa Dhaigude D15A15

PWA EXP8

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 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.

Codes:

```
//Serviceworker.js
// sw.js - Complete Service Worker for E-commerce PWA
const CACHE_NAME = "ecommerce-pwa-v2";
const urlsToCache = ["/", "/index.html", "/offline.html", "/css/main.min.css", "/js/app.min.js"];

self.addEventListener("install", (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
      console.log("[Service Worker] Caching files");
      return cache.addAll(urlsToCache);
    })
  );
});

self.addEventListener("activate", (event) => {
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cache) => {
          if (cache !== CACHE_NAME) {
            console.log("[Service Worker] Deleting old cache:", cache);
            return caches.delete(cache);
          }
        })
      );
    })
  );
});
```

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

```
// Fetch Event
// ======================
self.addEventListener('fetch', (event) => {
const { request } = event;
const url = new URL(request.url);
// 1. Skip non-GET requests and chrome-extension
if (request.method !== 'GET' || url.protocol ===
'chrome-extension:') {
return;
}
```

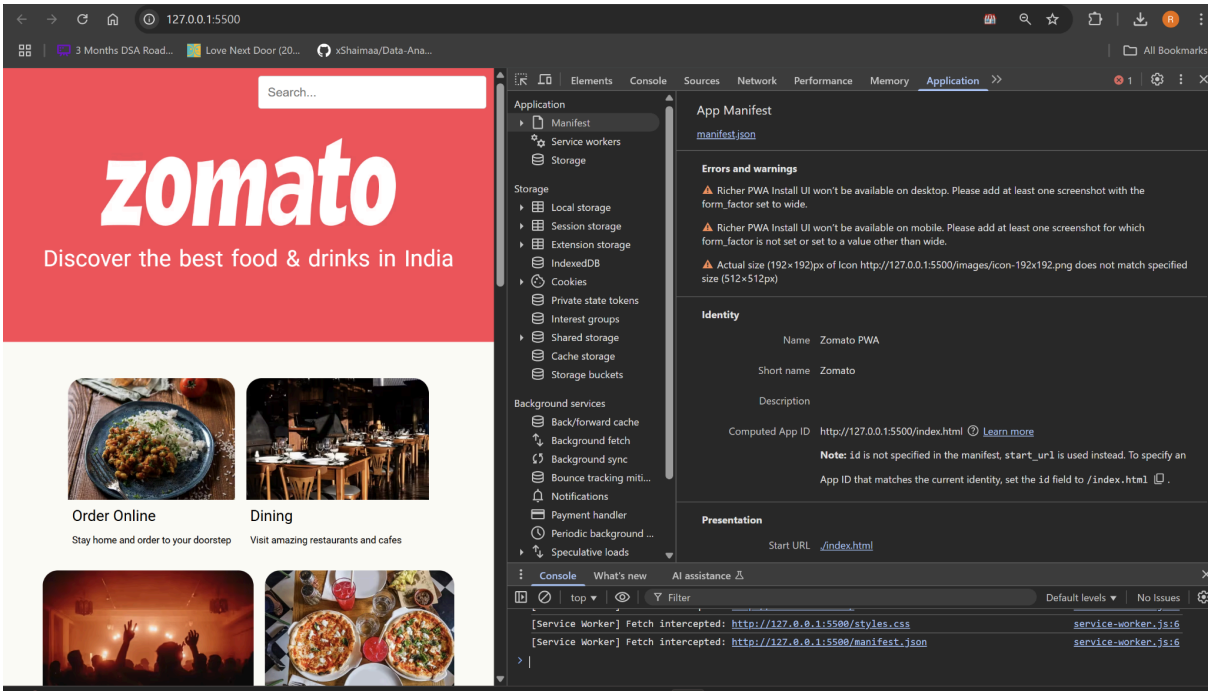
```
// 2. API Requests (Network First with Cache Fallback)
if (url.pathname.startsWith('/api/')) {
event.respondWith(
fetch(request)
.then(networkResponse => {
// Cache successful API responses
if (networkResponse.ok) {
const clone = networkResponse.clone();
caches.open(API_CACHE)
.then(cache => cache.put(request, clone));
}
return networkResponse;
})
.catch(() => {
// Return cached version if available
return caches.match(request)
.then(cachedResponse => cachedResponse || Response.json(
{ error: 'Network error' },
{ status: 503 }
));
})
);
```

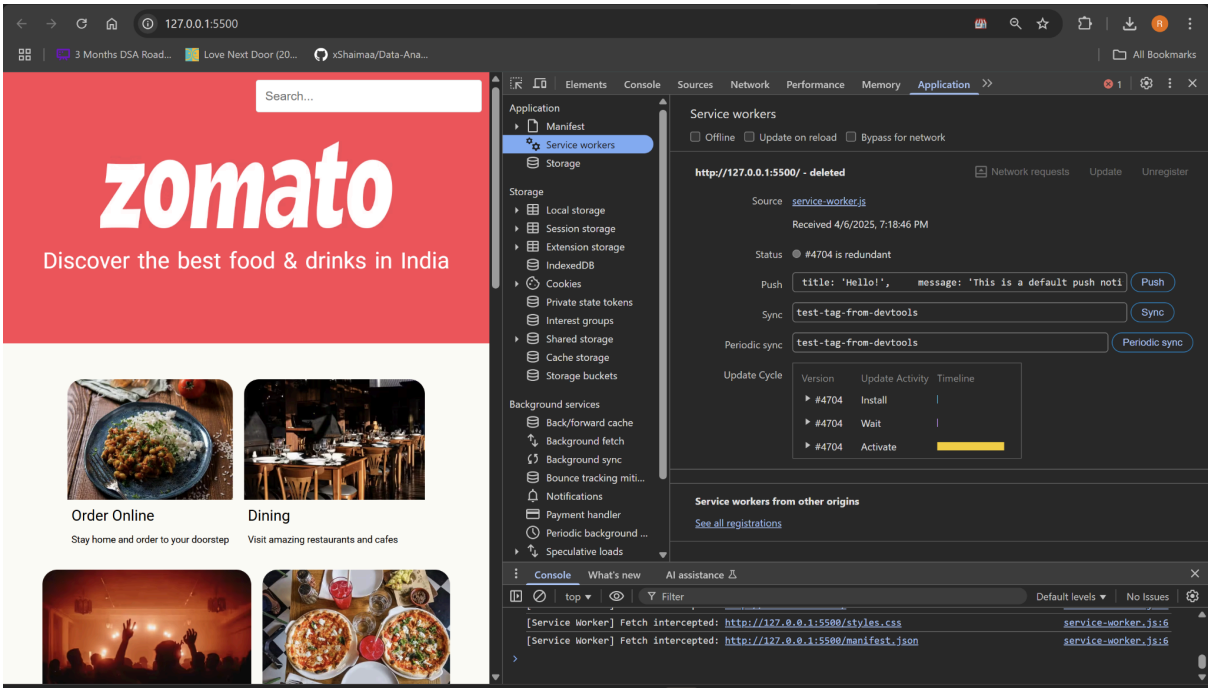
```
return;
}

// 3. Static Assets (Cache First with Network Fallback)
event.respondWith(
caches.match(request)
.then(cachedResponse => {
// Return cached version if found
if (cachedResponse) {
return cachedResponse;
}
// Otherwise fetch from network
return fetch(request)
.then(networkResponse => {
// Cache successful responses
if (networkResponse.ok) {
const clone = networkResponse.clone();
caches.open(CACHE_NAME)
.then(cache => cache.put(request, clone));
}
return networkResponse;
})
.catch(() => {
// Special handling for HTML pages
if (request.headers.get('accept').includes('text/html')) {
return caches.match('/offline.html');
}
// Return placeholder for images
if (request.headers.get('accept').includes('image')) {
return caches.match('/images/placeholder-product.jpg');
}
});
})
);
});
// ======================


 // Background Sync
// ======================
self.addEventListener('sync', (event) => {
if (event.tag === 'sync-cart') {
event.waitUntil(
// Get cart data from IndexedDB
```

```
getCartData()
.then(cartItems => {
return fetch('/api/cart/sync', {
method: 'POST',
headers: { 'Content-Type': 'application/json' },
body: JSON.stringify(cartItems)
});
})
.then(() => {
return showNotification('Cart Synced', 'Your cart has been
updated');
})
.catch(err => {
console.error('Sync failed:', err);
})
);
}
});
// ======================
// Push Notifications
// ======================
self.addEventListener('push', (event) => {
let data = {};
try {
data = event.data.json();
} catch (e) {
data = {
title: 'New Update',
body: 'Check out our latest products!',
icon: '/icons/icon-192x192.png',
url: '/'
};
}
const options = {
body: data.body,
icon: data.icon || '/icons/icon-192x192.png',
badge: '/icons/icon-96x96.png',
data: {
url: data.url || '/'
}
};
event.waitUntil(
self.registration.showNotification(data.title, options)
);
```

```
});
self.addEventListener('notificationclick', (event) => {
event.notification.close();
event.waitUntil(
clients.matchAll({ type: 'window' })
.then(clientList => {
for (const client of clientList) {
if (client.url === event.notification.data.url && 'focus' in
client) {
return client.focus();
}
}
if (clients.openWindow) {
return clients.openWindow(event.notification.data.url);
}
})
);
});
// ======================
// Helper Functions
// ======================
async function getCartData() {
// In a real app, you would use IndexedDB
return new Promise(resolve => {
resolve([]);
});
}
async function showNotification(title, body) {
return self.registration.showNotification(title, { body });
}
```

Output:


127.0.0.1:5500
Search...
zomato
Discover the best food & drinks in India
Order Online
Stay home and order to your doorstep
Dining
Visit amazing restaurants and cafes
Elements
Console
Sources
Network
Performance
Memory
Application
App Manifest
manifest.json
Errors and warnings
Identity
Name Zomato PWA
Short name Zomato
Description
Computed App ID http://127.0.0.1:5500/index.html
Presentation
Start URL ./index.html
[Service Worker] Fetch intercepted: http://127.0.0.1:5500/styles.css
[Service Worker] Fetch intercepted: http://127.0.0.1:5500/manifest.json


127.0.0.1:5500
Search...
zomato
Discover the best food & drinks in India
Order Online
Stay home and order to your doorstep
Dining
Visit amazing restaurants and cafes
Elements
Console
Sources
Network
Performance
Memory
Application
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5500/ - deleted
Source service-worker.js
Received 4/6/2025, 7:18:46 PM
Status #4704 is redundant
Push
Sync
Periodic sync
Update Cycle
Service workers from other origins
See all registrations
[Service Worker] Fetch intercepted: http://127.0.0.1:5500/styles.css
[Service Worker] Fetch intercepted: http://127.0.0.1:5500/manifest.json

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.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Ria Chaudhari D15A07
Eesha Chavan D15A08
Shraeyaa Dhaigude D15A15

PWA EXP9

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

Theory:

Service Worker

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

Things to note about Service Worker:

● A service worker is a programmable network proxy that lets you control how network requests from your page are handled.

● Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.

● The service worker becomes idle when not in use and restarts when it's next needed.

You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.

● Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out Promises, an introduction.

Fetch Event

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

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

● CacheFirst - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.

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

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

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:

```
//Serviceworker.js
// sw.js - Complete Service Worker for E-commerce PWA
const CACHE_NAME = 'ecommerce-pwa-v2';
const API_CACHE = 'ecommerce-api-v1';
const ASSETS_TO_CACHE = [
'/',
'/index.html',
'/manifest.json',
'/offline.html',
'/css/main.min.css',
'/js/app.min.js',
'/icons/icon-192x192.png',
'/icons/icon-512x512.png',
'/images/placeholder-product.jpg'
];
// ======================
// Install Event
// ======================
self.addEventListener('install', (event) => {
event.waitUntil(
caches.open(CACHE_NAME)
.then((cache) => {
console.log('[Service Worker] Cache opened');
```

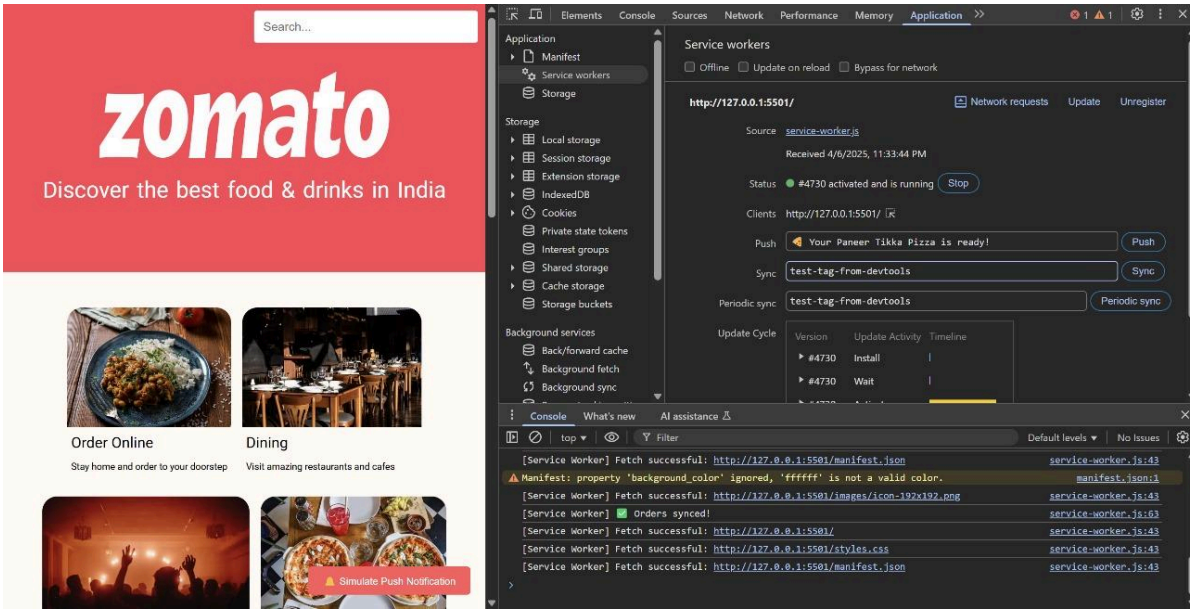
```
return cache.addAll(ASSETS_TO_CACHE);
})
.then(() => self.skipWaiting())
);
});
// ======================
// Activate Event
// ======================
self.addEventListener('activate', (event) => {
event.waitUntil(
caches.keys().then((cacheNames) => {
return Promise.all(
cacheNames.map((cacheName) => {
if (cacheName !== CACHE_NAME && cacheName !== API_CACHE) {
console.log('[Service Worker] Deleting old cache:',
cacheName);
return caches.delete(cacheName);
}
})
);
})
.then(() => self.clients.claim())
);
});
// ======================
// Fetch Event
// ======================
self.addEventListener('fetch', (event) => {
const { request } = event;
const url = new URL(request.url);
// 1. Skip non-GET requests and chrome-extension
if (request.method !== 'GET' || url.protocol ===
'chrome-extension:') {
return;
}
// 2. API Requests (Network First with Cache Fallback)
if (url.pathname.startsWith('/api/')) {
event.respondWith(
fetch(request)
.then(networkResponse => {
// Cache successful API responses
if (networkResponse.ok) {
const clone = networkResponse.clone();
caches.open(API_CACHE)
```

```
.then(cache => cache.put(request, clone));
}
return networkResponse;
})
.catch(() => {
// Return cached version if available
return caches.match(request)
.then(cachedResponse => cachedResponse || Response.json(
{ error: 'Network error' },
{ status: 503 }
));
})
);
return;
}
// 3. Static Assets (Cache First with Network Fallback)
event.respondWith(
caches.match(request)
.then(cachedResponse => {
// Return cached version if found
if (cachedResponse) {
return cachedResponse;
}
// Otherwise fetch from network
return fetch(request)
.then(networkResponse => {
// Cache successful responses
if (networkResponse.ok) {
const clone = networkResponse.clone();
caches.open(CACHE_NAME)
.then(cache => cache.put(request, clone));
}
return networkResponse;
})
.catch(() => {
// Special handling for HTML pages
if (request.headers.get('accept').includes('text/html')) {
return caches.match('/offline.html');
}
// Return placeholder for images
if (request.headers.get('accept').includes('image')) {
return caches.match('/images/placeholder-product.jpg');
}
});
```

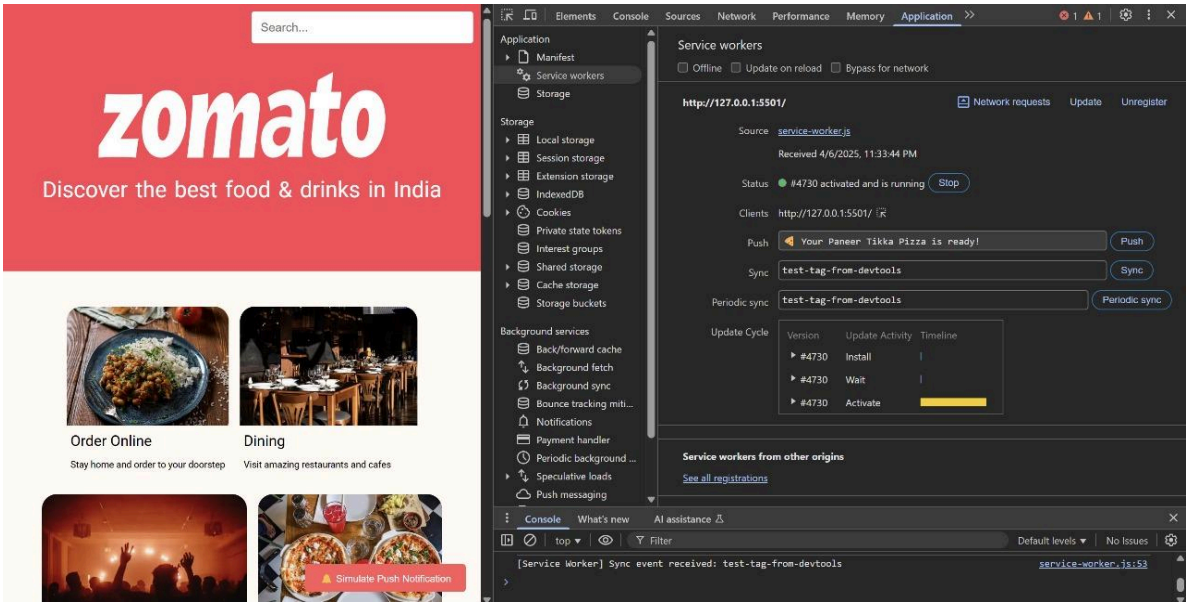
```
})
);
});
// ======================
// Background Sync
// ======================
self.addEventListener('sync', (event) => {
if (event.tag === 'sync-cart') {
event.waitUntil(
// Get cart data from IndexedDB
getCartData()
.then(cartItems => {
return fetch('/api/cart/sync', {
method: 'POST',
headers: { 'Content-Type': 'application/json' },
body: JSON.stringify(cartItems)
});
})
.then(() => {
return showNotification('Cart Synced', 'Your cart has been
updated');
})
.catch(err => {
console.error('Sync failed:', err);
})
);
}
});
// ======================
// Push Notifications
// ======================
self.addEventListener('push', (event) => {
let data = {};
try {
data = event.data.json();
} catch (e) {
data = {
title: 'New Update',
body: 'Check out our latest products!',
icon: '/icons/icon-192x192.png',
url: '/'
};
}
const options = {
```

```
body: data.body,
icon: data.icon || '/icons/icon-192x192.png',
badge: '/icons/icon-96x96.png',
data: {
url: data.url || '/'
}
};
event.waitUntil(
self.registration.showNotification(data.title, options)
);
});
self.addEventListener('notificationclick', (event) => {
event.notification.close();
event.waitUntil(
clients.matchAll({ type: 'window' })
.then(clientList => {
for (const client of clientList) {
if (client.url === event.notification.data.url && 'focus' in
client) {
return client.focus();
}
}
if (clients.openWindow) {
return clients.openWindow(event.notification.data.url);
}
})
);
});
// ======================
// Helper Functions
// ======================
async function getCartData() {
// In a real app, you would use IndexedDB
return new Promise(resolve => {
resolve([]);
});
}
async function showNotification(title, body) {
return self.registration.showNotification(title, { body });
}
```

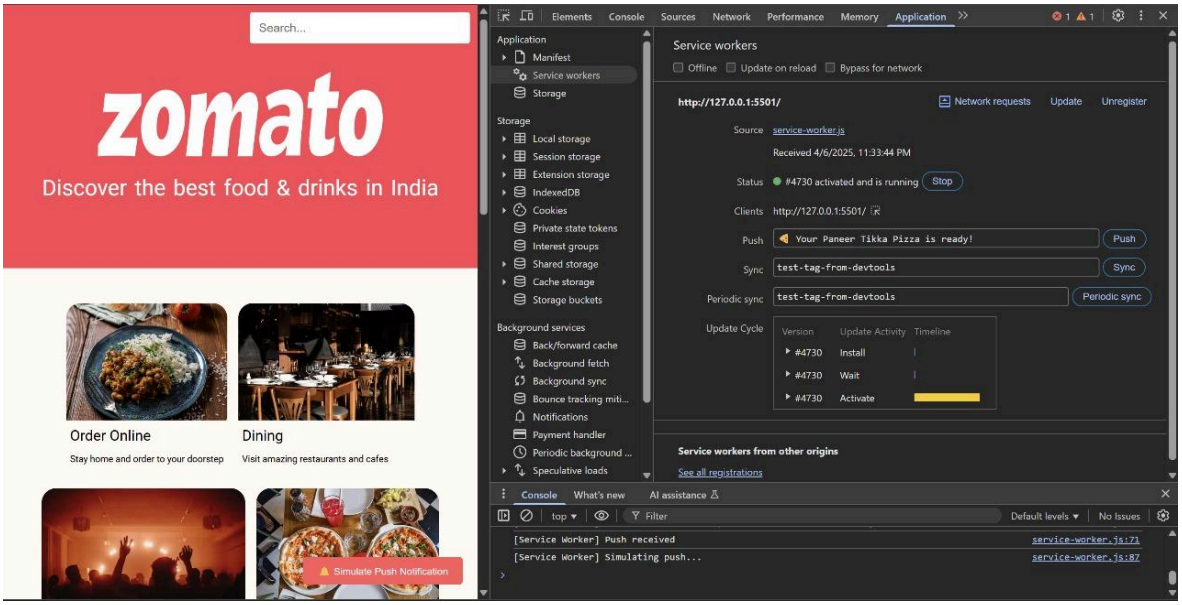
Output:
Fetch event:



Sync event:



Push:


Search...
zomato
Discover the best food & drinks in India
Order Online
Stay home and order to your doorstep
Dining
Visit amazing restaurants and cafes
Simulate Push Notification
Elements
Console
Sources
Network
Performance
Memory
Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
Back/forward cache
Background fetch
Background sync
Bounce tracking miti...
Notifications
Payment handler
Periodic background ...
Speculative loads
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5501/
Network requests
Update
Unregister
Source
service-worker.js
Received 4/6/2025, 11:33:44 PM
Status
#4730 activated and is running
Stop
Clients
http://127.0.0.1:5501/
Push
Your Paneer Tikka Pizza is ready!
Push
Sync
test-tag-from-devtools
Sync
Periodic sync
test-tag-from-devtools
Periodic sync
Update Cycle
Version
Update Activity
Timeline
#4730
Install
#4730
Wait
#4730
Activate
Service workers from other origins
See all registrations
Console
What's new
AI assistance
top
Filter
Default levels
No Issues
[Service Worker] Push received
service-worker.js:71
[Service Worker] Simulating push...
service-worker.js:87

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.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Ria Chaudhari D15A07
Eesha Chavan D15A08
Shraeyaa Dhaigude D15A15

PWA EXP10

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

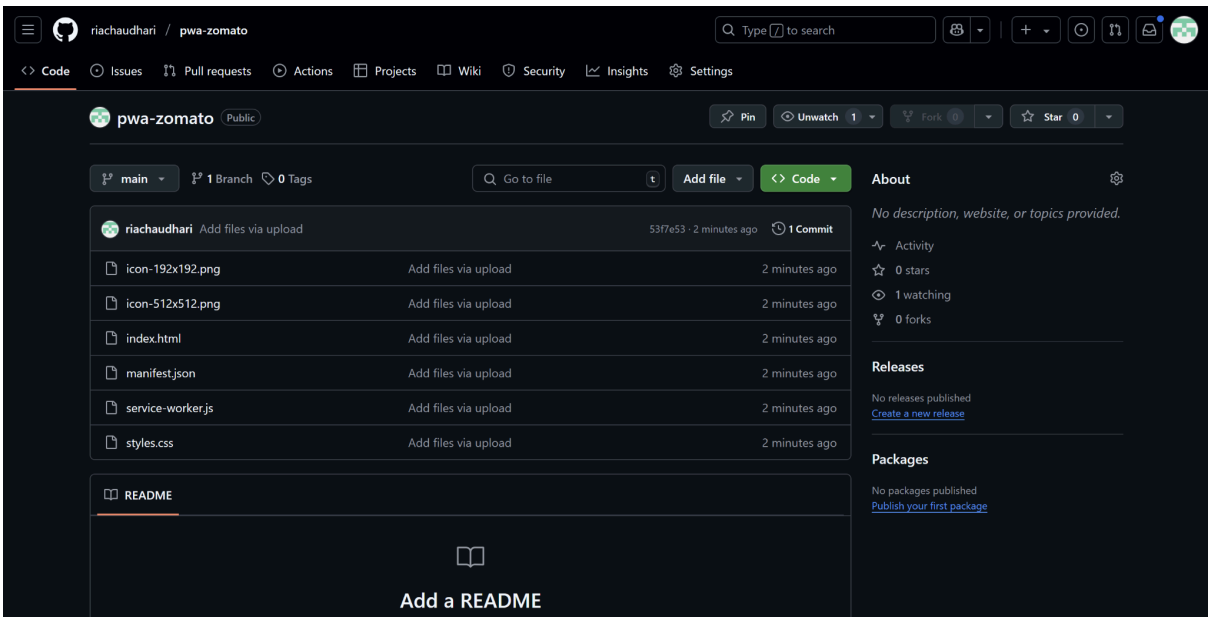
1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5.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

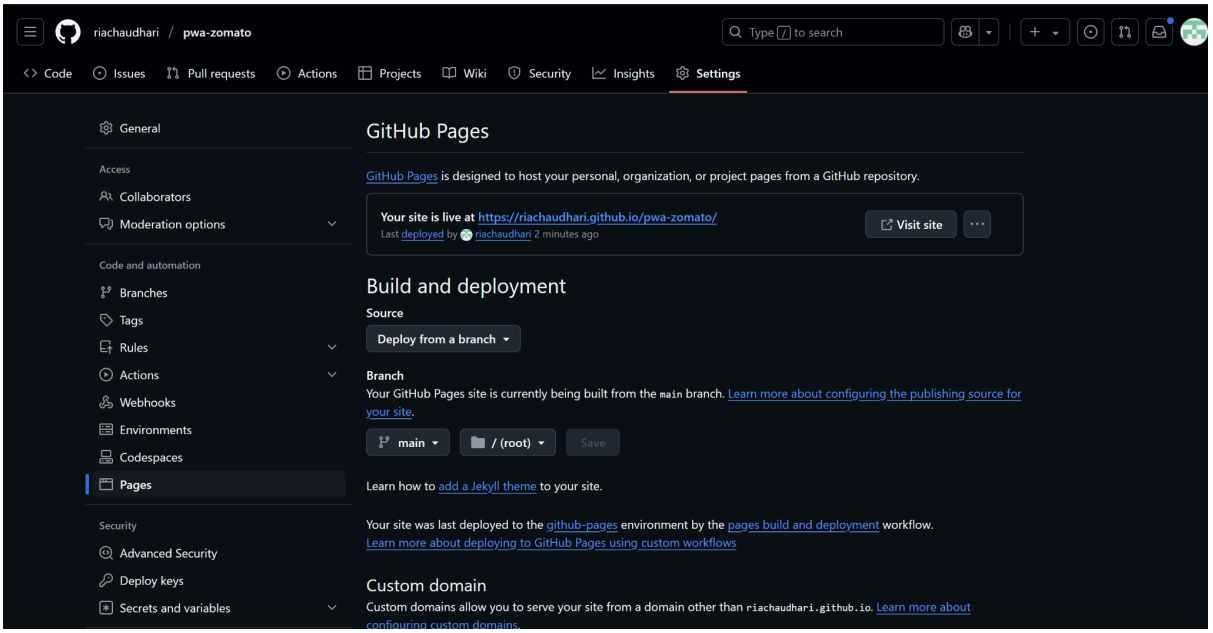
Cons

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/riachaudhari/pwa-zomato

Github Screenshot:





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.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

Ria Chaudhari D15A07
Eesha Chavan D15A08
Shraeyaa Dhaigude D15A15

PWA EXP11

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

Theory:

Google Lighthouse :

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

Key Features and Audit Metrics:

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

● Performance:
Measures loading speed, content display time, and overall site efficiency (score out of 100).

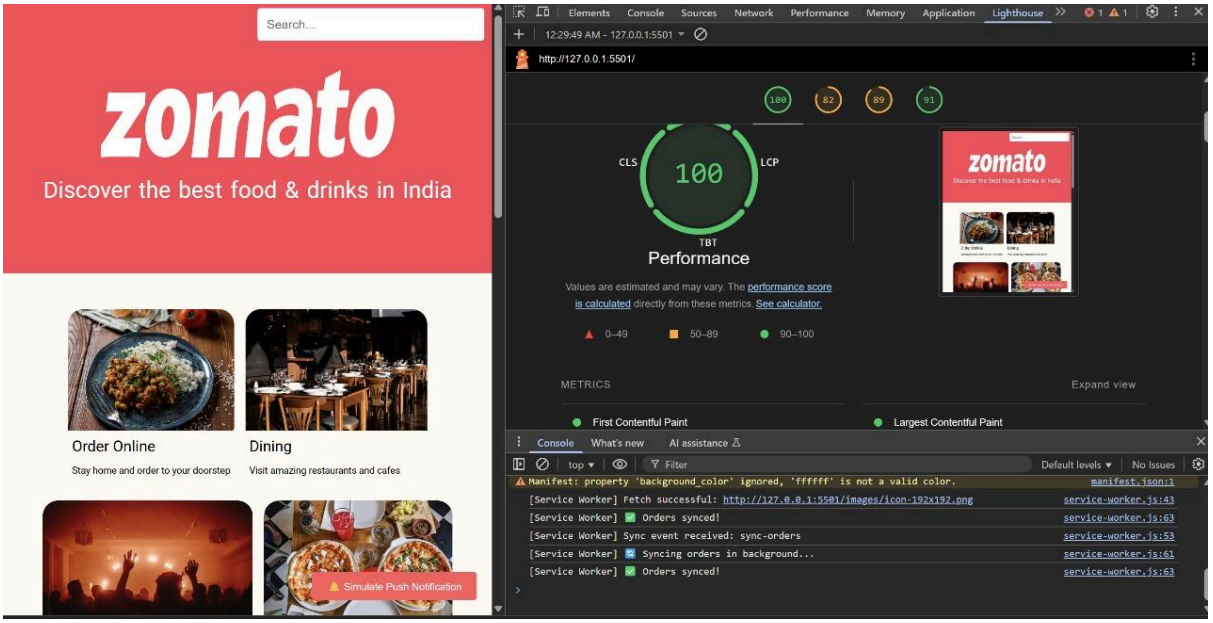
● PWA Score (Mobile):
Evaluates adherence to Google's Baseline PWA checklist, including Service Worker implementation, offline functionality, and script-disabled performance.

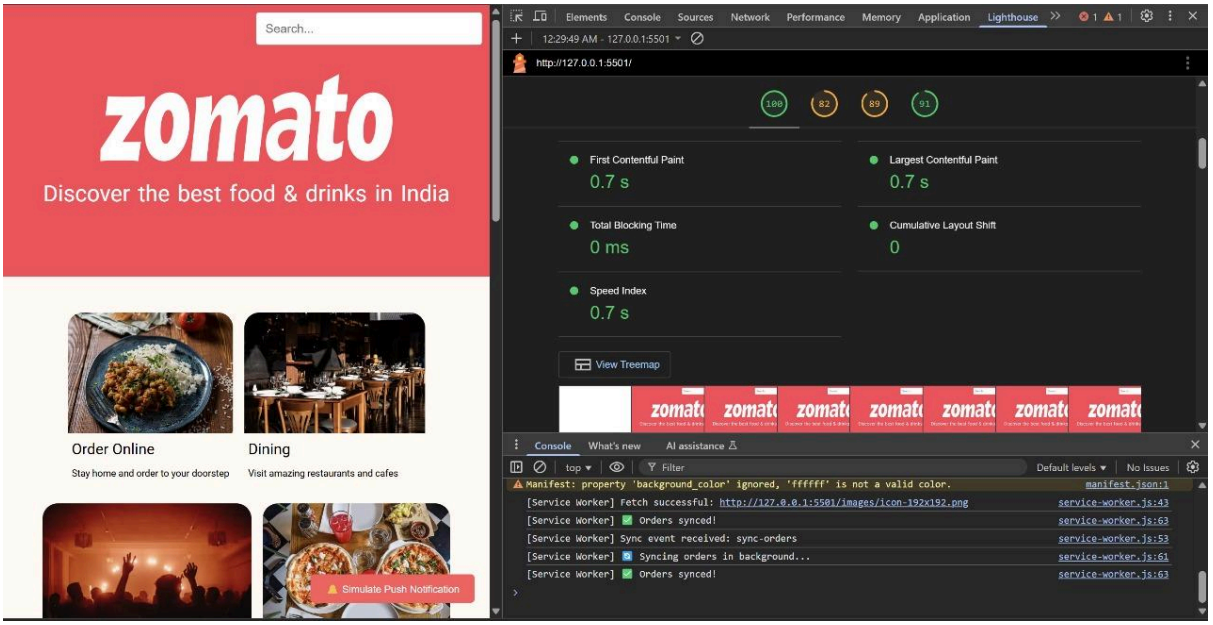
● Accessibility:
Assesses website accessibility features like aria- attributes, screen-reader compatibility, and semantic HTML tags. Scores are pass/fail.

● Best Practices:
Checks adherence to industry standards, including HTTPS usage and security measures.

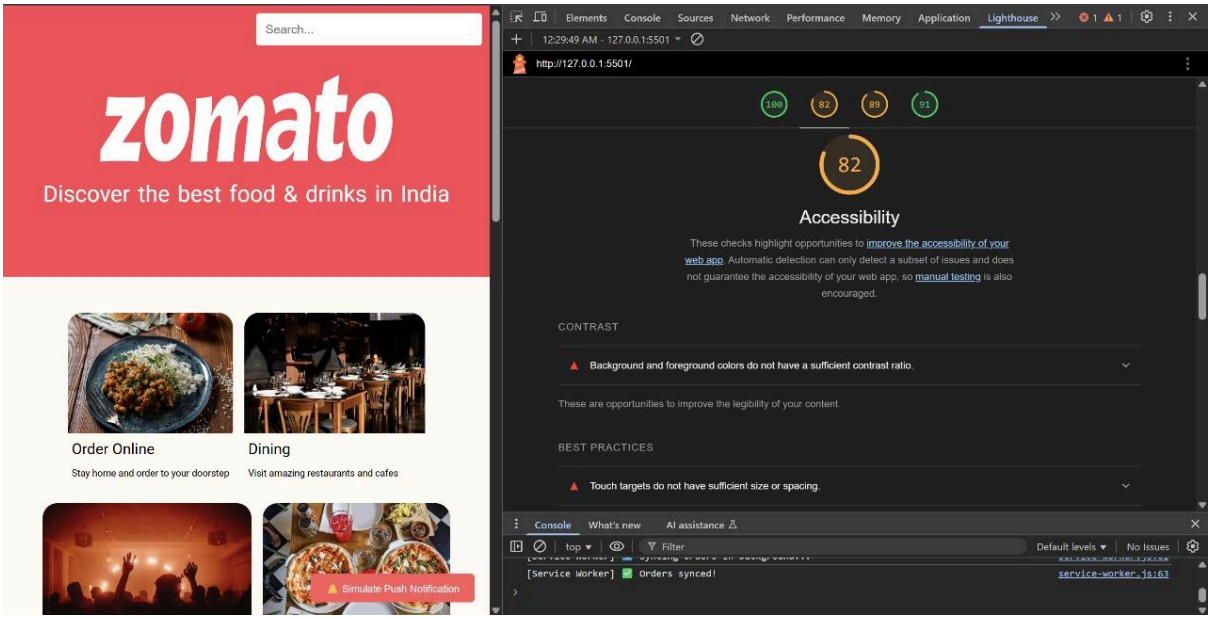
Output:

a) Performance

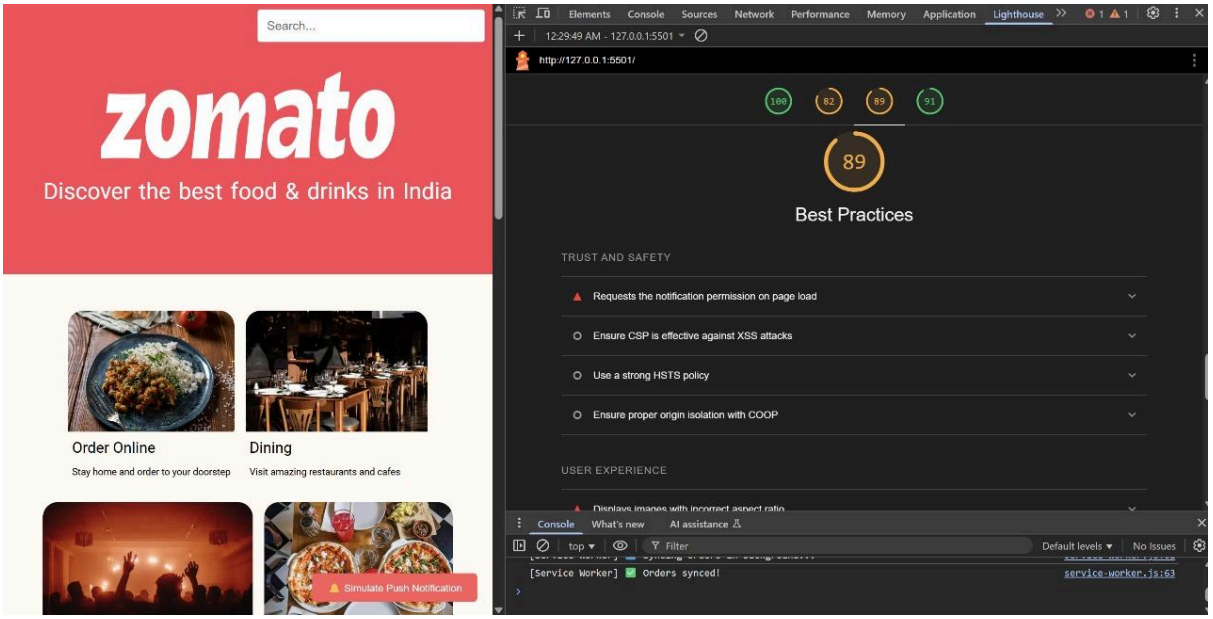




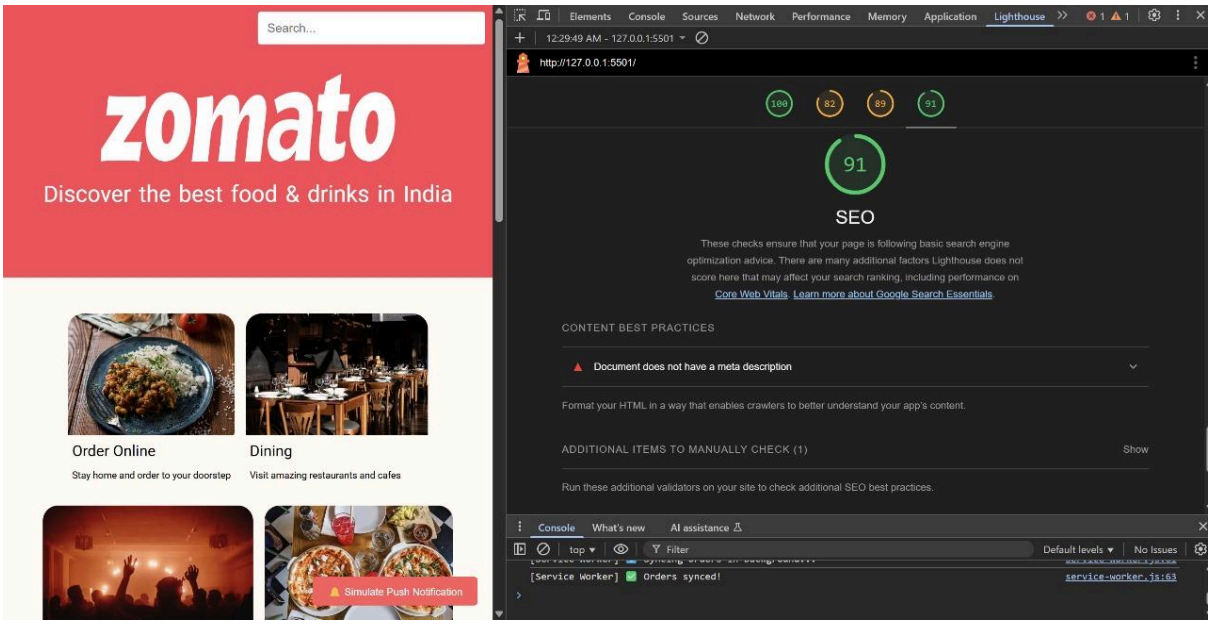
b)Accessibility



c) Best Practices



d) Search Engine Optimization (SEO)



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.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
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.	15
Name	Shraeyaa Dhaigude
Class	D15A/D15B
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:	