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 **Sarvadnya Rajendra Awaghad_** of **D15A** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Project Title: Flutter : Tinder Clone / PWA : Dormdine **Roll No. 04**

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

Year/Sem/Class : D15A/D15B **A.Y.:** 23-24

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers : Mrs. Kajal Jewani.

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: Flutter : Tinder Clone / PWA : Dormdine **Roll No. 04**

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

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1			15 M
2.	To design Flutter UI by including common widgets.	LO2			15 M
3.	To include icons, images, fonts in Flutter app	LO2			15 M
4.	To create an interactive Form using form widget	LO2			15 M
5.	To apply navigation, routing and gestures in Flutter App	LO2			15 M
6.	To Connect Flutter UI with fireBase database	LO3			15 M
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4			15 M
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			15 M
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5			15 M
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5			15 M
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6			15 M
12.	Assignment-1	LO1,LO2 ,LO3			5 M
13.	Assignment-2	LO4,LO5 ,LO6			4 M

MAD & PWA Lab

Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	04
Name	Sarvadnya Rajendra Awaghad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	15 M

Experiment No : 1

Aim : To install and configure the Flutter Environment

Theory:

Flutter is an open-source UI software development toolkit created by Google. It is used for building natively compiled applications for mobile, web, and desktop from a single codebase. Flutter uses the Dart programming language and follows a reactive programming paradigm.

Key Concepts and Theoretical Aspects:

Widget Tree:

In Flutter, everything is a widget. Widgets are the basic building blocks of the user interface.
The entire UI is represented as a tree of widgets. Each widget can be a structural element (like a button or text) or a layout (like a row or column).
Declarative UI:

Flutter follows a declarative approach, where the UI is described in terms of what it should look like based on the current state. Developers declare the desired state, and Flutter takes care of updating the UI to reflect that state.
Hot Reload:

One of Flutter's key features is Hot Reload, which allows developers to instantly see the impact of the code changes without restarting the entire application. This accelerates the development process and enhances productivity.
State Management:

Flutter provides various options for managing the state of an application. Understanding and implementing effective state management is crucial for building scalable and maintainable Flutter applications.
Dart Language:

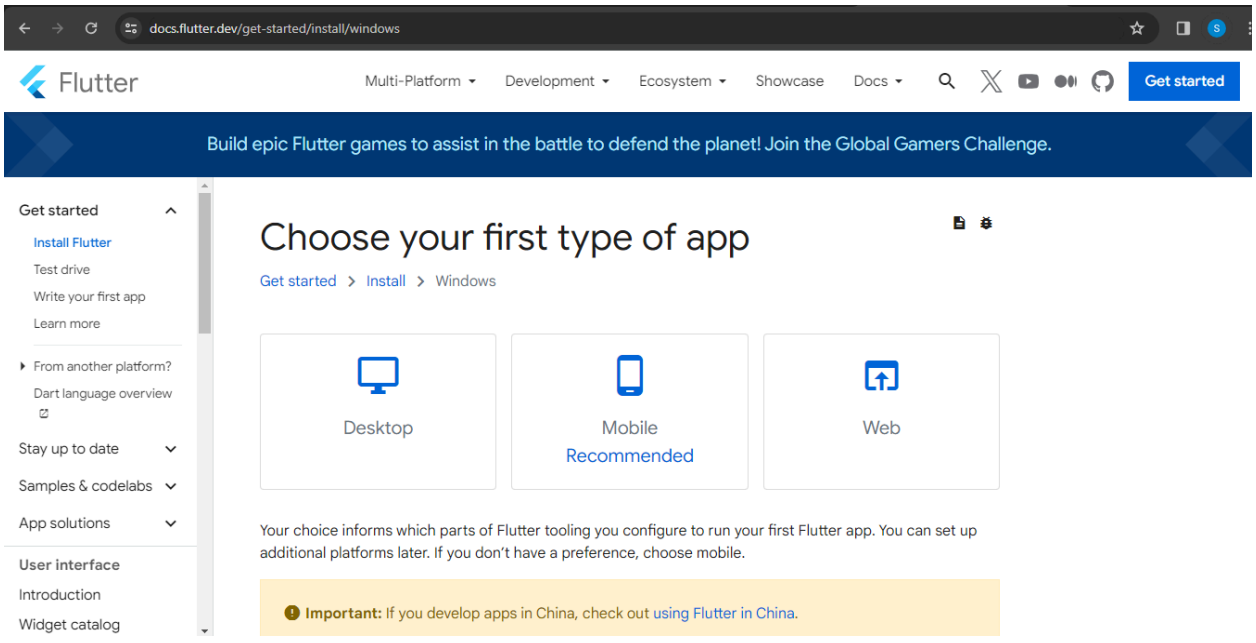
Flutter uses Dart as its programming language. Understanding the Dart language and its features is essential for efficient Flutter development.
Material Design and Cupertino Widgets:

Flutter provides a set of widgets that implement the Material Design guidelines for Android and iOS-style widgets for iOS through the Cupertino library. Understanding how to use these widgets helps in creating platform-specific user interfaces.
Asynchronous Programming:

Dart is designed to support asynchronous programming, which is important for handling tasks like network requests without blocking the UI thread. Understanding asynchronous programming concepts is crucial for developing responsive Flutter applications.

Installation of Flutter:

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.



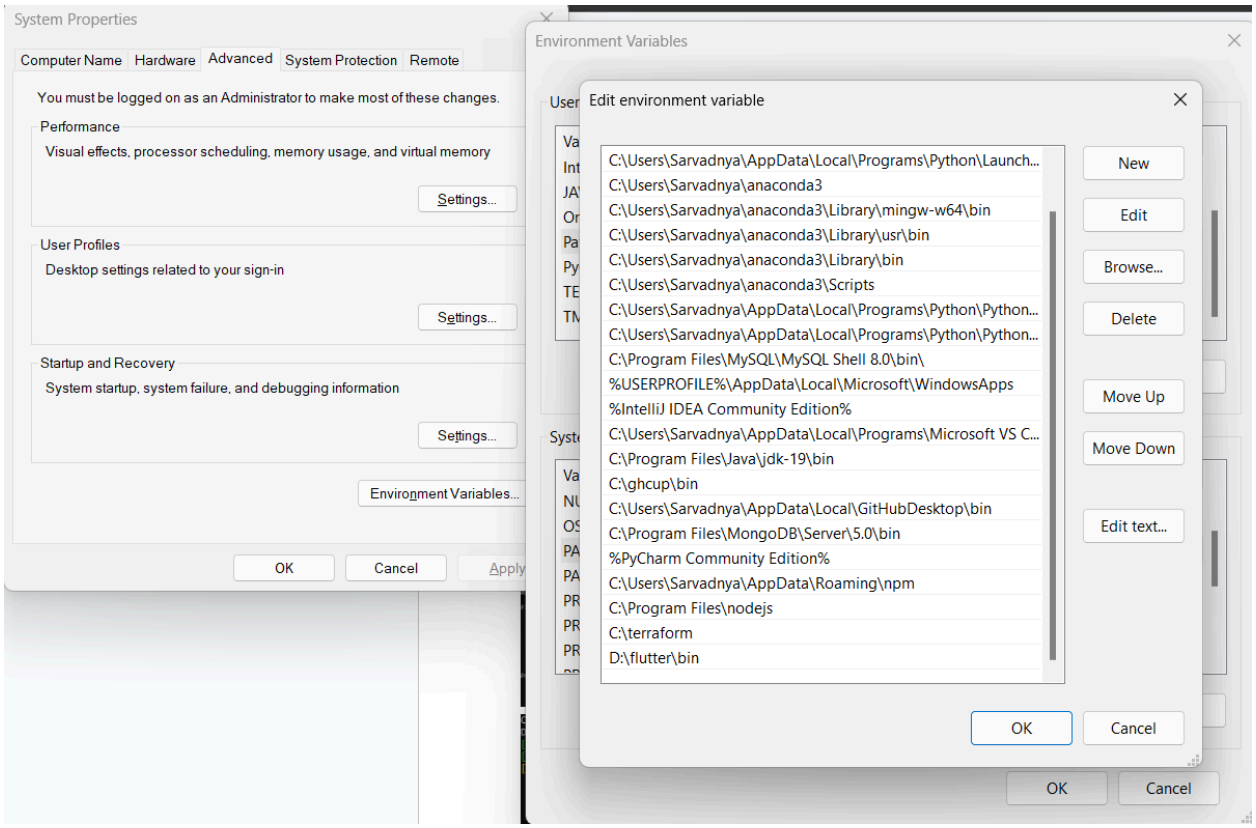
Step 2: Next, to download the latest Flutter SDK, click on the Windows icon. Here, you will
find the download link for SDK.

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:

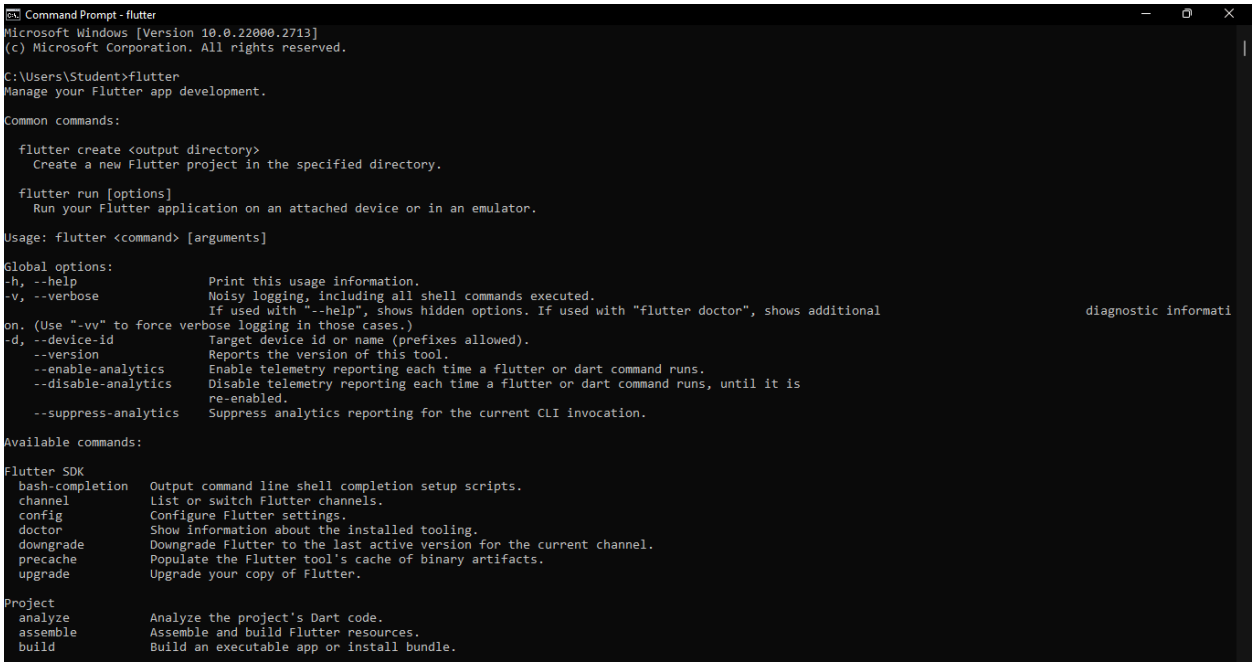
Step 4.1: Go to MyComputer properties -> advanced tab -> environment variables. You will get
the following screen.

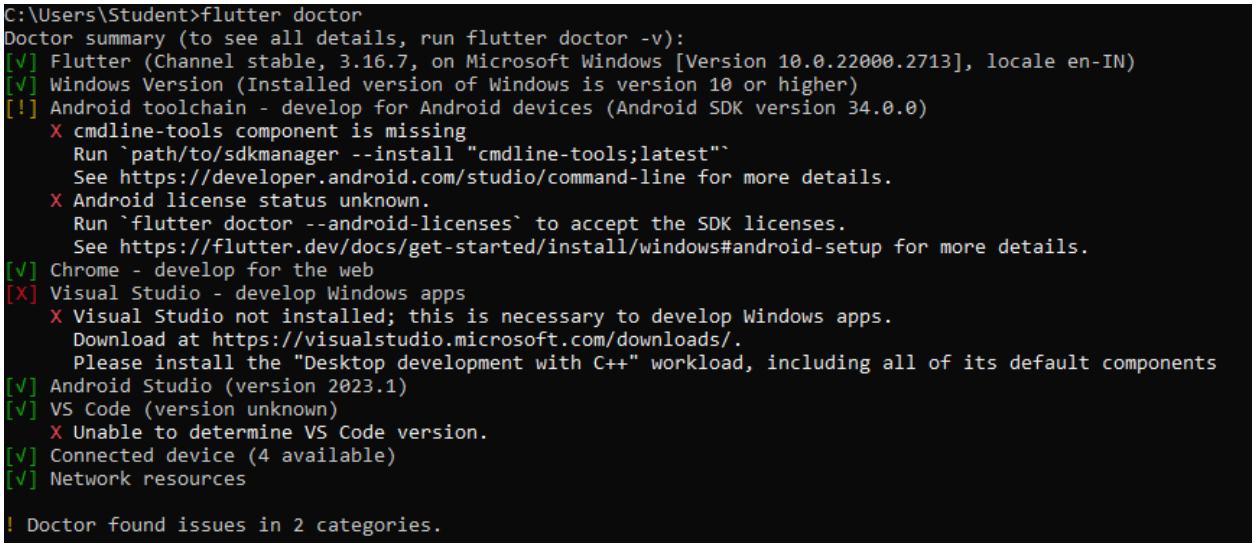
Editing Environment Variables:



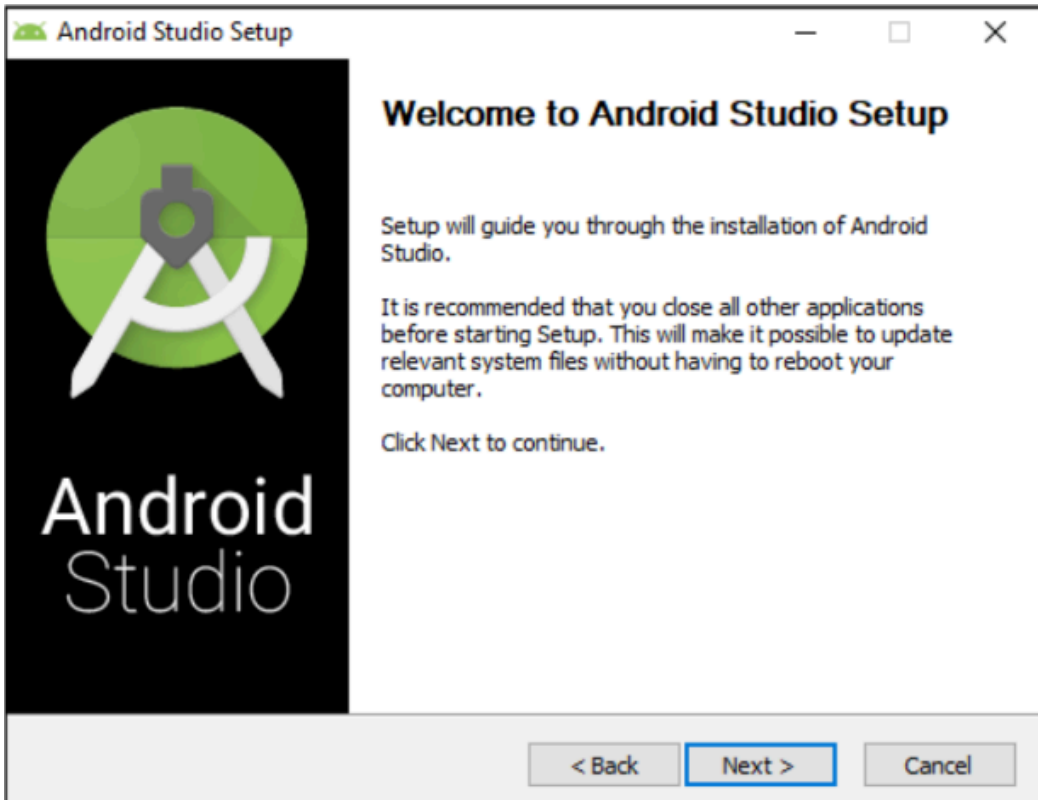
Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value -
> ok -> ok -> ok.

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

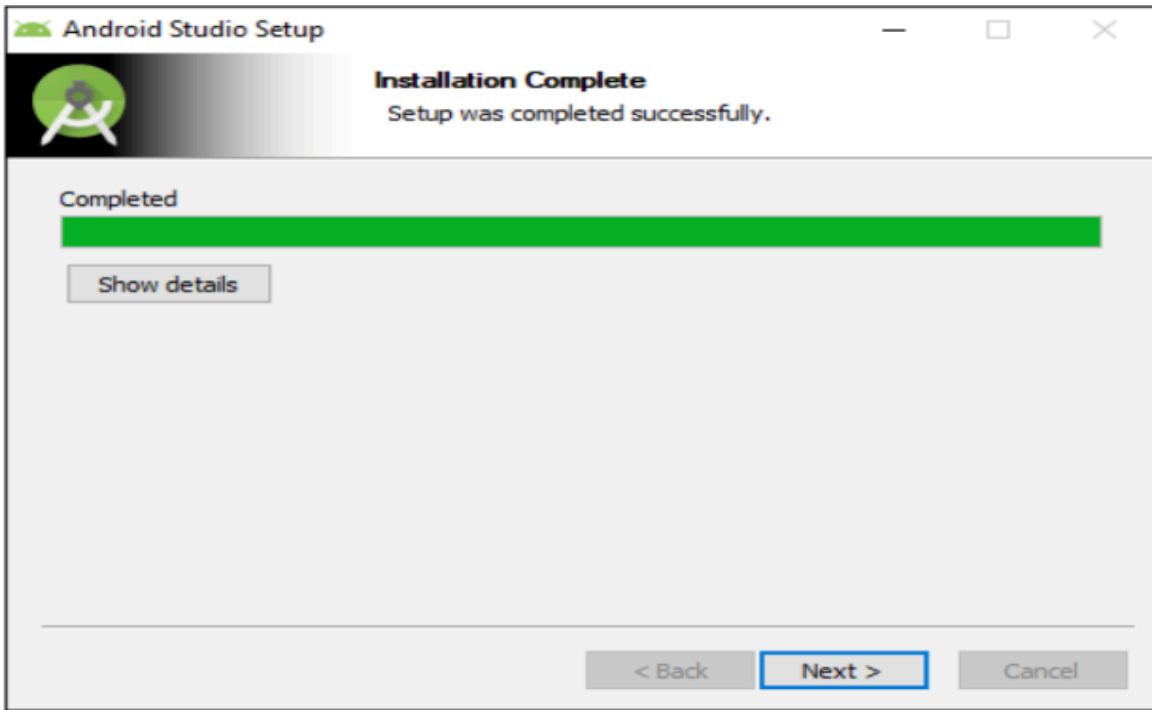




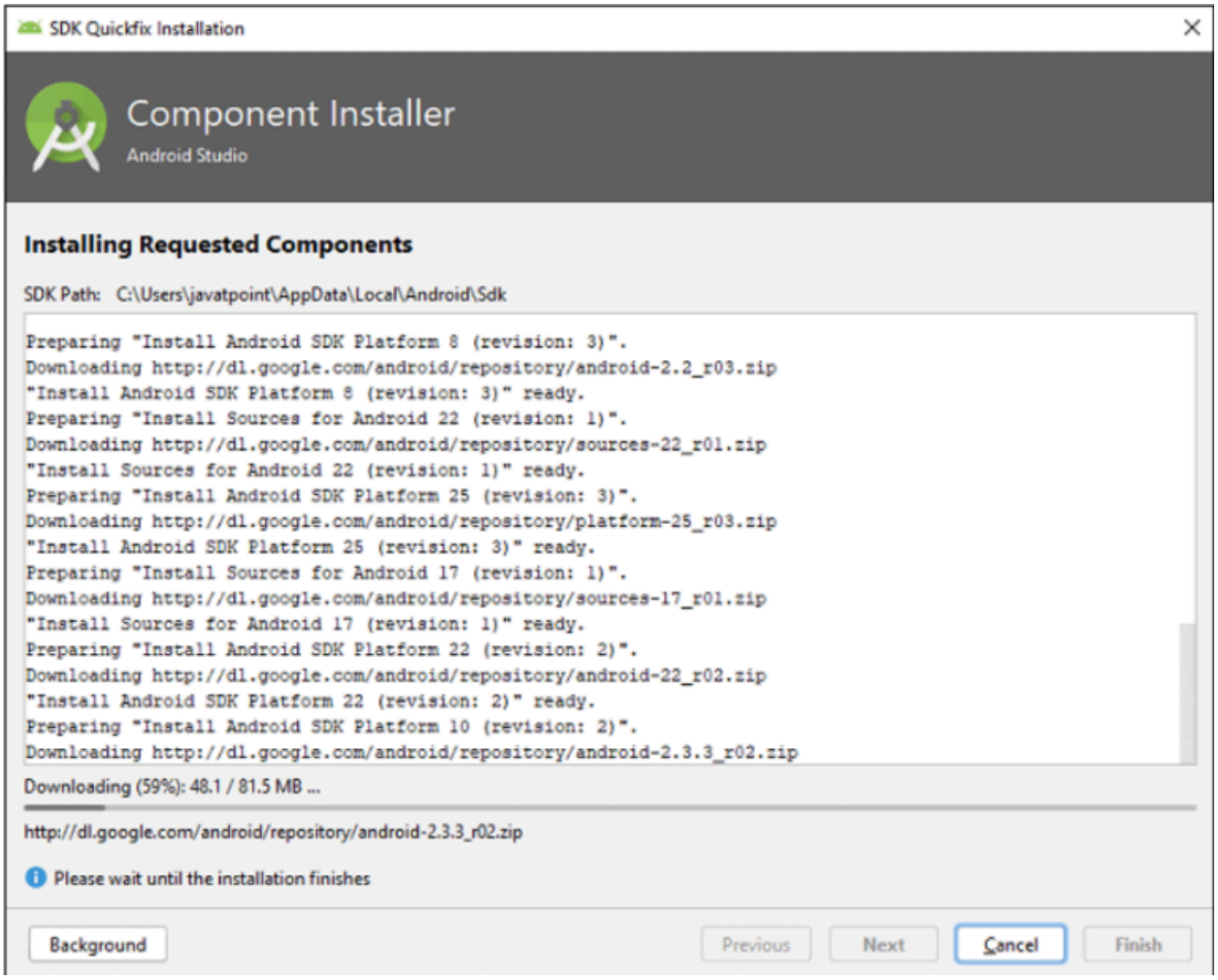
Installation of Android Studio:



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

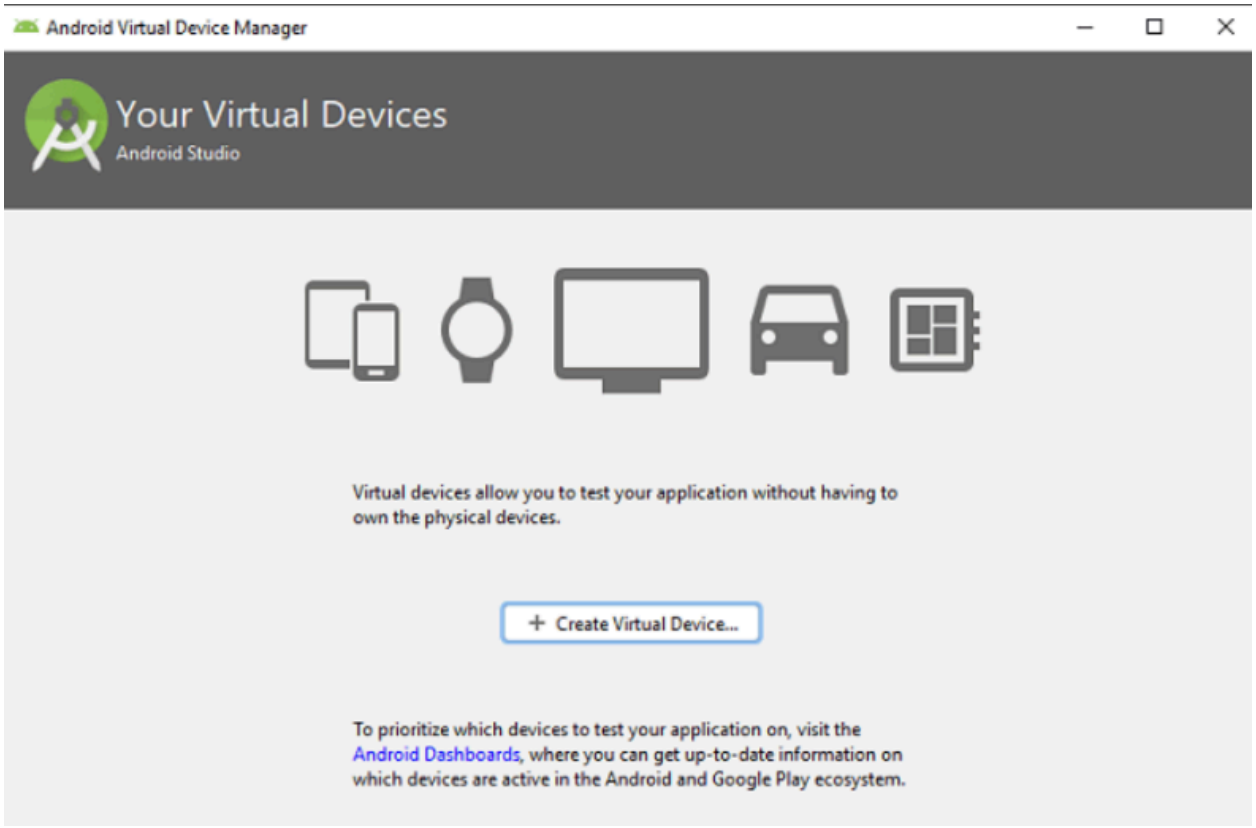


Step : In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to choose the 'Don't import Settings option' and click OK. It will start the Android Studio.

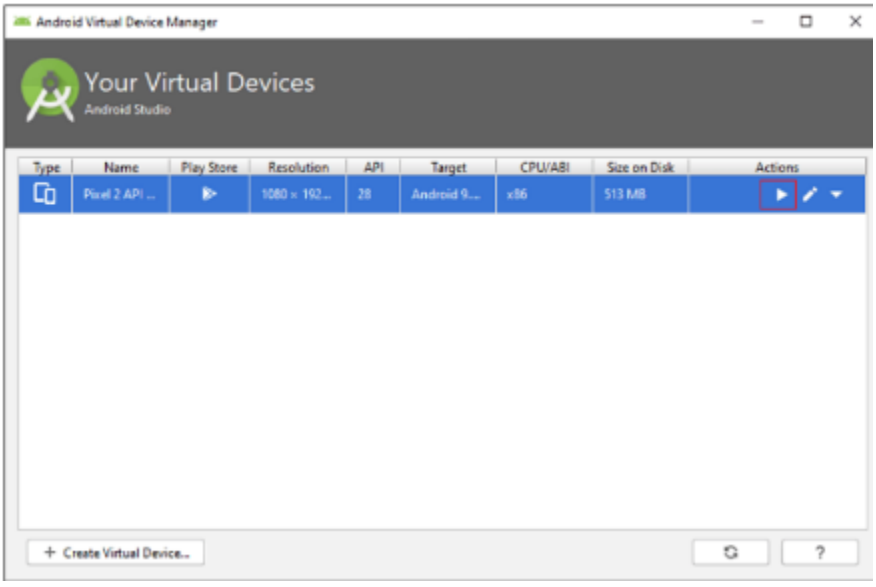


Step 8: Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.

Step 8..2: To set an Android emulator, go to Android Studio > Tools > Android > AVD Managerand select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.



Step 8.2: Choose your device definition and click on Next.
Step 8.3: Select the system image for the latest Android version and click on Next.
Step 8.4: Now, verify the all AVD configuration. If it is correct, click on Finish. The following screen appears.

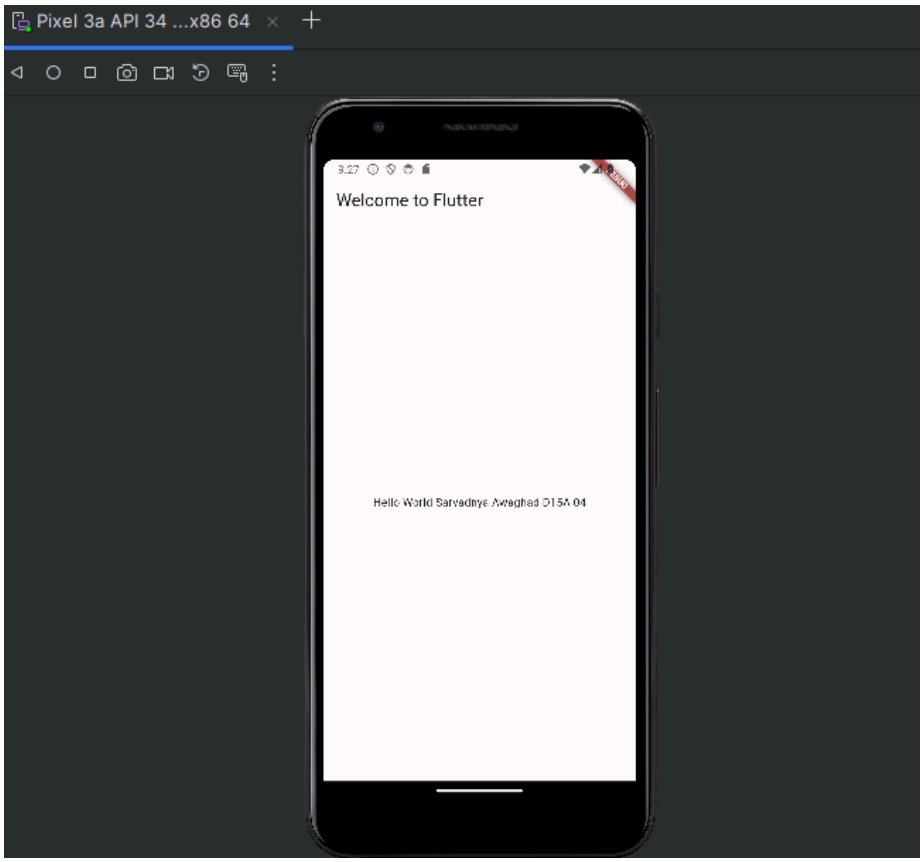


Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

Flutter Hello World App :

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
```

```
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ),
        body: const Center(
          child: Text('              Hello World \n Sarvadnya Awaghad D15A 04'),

        ),
      ),
    );
  }
}
```



Conclusion : In the above experiment , we have successfully configured and installed flutter and android studio in our desktop. AVD emulator is installed and flutter app is executed successfully.

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	04
Name	Sarvadnya Rajendra Awaghad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15 M

Aim: To design Flutter UI by including common widgets.

Theory:

Flutter widgets are the building blocks of the user interface, representing visual and interactive elements. Widgets can be categorized into two main types: Stateless and Stateful. Stateless widgets are immutable and do not store any internal state, while Stateful widgets can change and maintain their state over time.

Built-in and Custom Widgets:
Flutter comes with a rich set of built-in widgets that cover a wide range of UI elements and interactions. Additionally, developers can create their custom widgets to tailor the user interface to specific requirements. This section explores the anatomy of custom widgets and their integration into the Flutter framework.

Layout and Constraints:
Flutter's layout system is based on the concept of constraints, allowing widgets to adapt to various screen sizes and orientations. Understanding how widgets handle layout constraints is essential for creating responsive and adaptive user interfaces.

Animation with Widgets:
Flutter provides a robust animation framework that seamlessly integrates with widgets, enabling the creation of fluid and engaging user experiences. This section explores how animations can be incorporated into Flutter widgets to enhance the overall user interface.

Testing and Debugging Widgets:
As with any software development, testing and debugging are integral parts of the process. This section discusses strategies for testing and debugging Flutter widgets, ensuring the reliability and quality of the UI components.

Container:
The Container widget is a basic building block that can contain other widgets and is often used to define the dimensions, padding, margin, and decoration of a UI element.

Text:
The Text widget is used to display a paragraph or a line of text. It supports various styling options such as font size, color, and alignment.

ListView:
The ListView widget is used to create a scrollable list of widgets. It can display a large number of children efficiently.

Row and Column:
Row and Column widgets are used to arrange children in a horizontal or vertical line, respectively.

Stack:
The Stack widget allows you to overlay multiple widgets on top of each other. It's often used for complex layouts.

AppBar:
The AppBar widget represents the top app bar that usually contains the app's title, icons, and actions.

Code:

main.dart :

```
import 'dart:math';
import 'package:cached_network_image/cached_network_image.dart';
import 'package:faker/faker.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_svg/flutter_svg.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:swipe_cards/draggable_card.dart';
import 'package:swipe_cards/swipe_cards.dart';
import 'package:tinder/authenticationScreen/chat_screen.dart';
```

```
import 'package:tinder/authenticationScreen/explore_screen.dart';
import 'package:tinder/authenticationScreen/profile_screen.dart';
class MainPage extends StatefulWidget {
 const MainPage({Key? key}) : super(key: key);
 @override
 State<MainPage> createState() => _MainPageState();
}
class _MainPageState extends State<MainPage> {
 List<SwipeItem> swipeItems = <SwipeItem>[];
 List<int> imageIndexList = List.generate(10, (index) => 0);
 MatchEngine matchEngine = MatchEngine();
 bool isNope = false;
 bool isLike = false;
 bool isSuperLike = false;
 @override
 void initState() {
  swipeItems = List.generate(
    10,
    (index) => SwipeItem(
      content: TinderCard(
        name: Faker().person.firstName(),
        photos: [
          'https://i.pravatar.cc/${800 + index}',
          ...List.generate(Random().nextInt(5) + 1,
              (photoIndex) => 'https://picsum.photos/${800 + photoIndex}'),
        ],
        age: Random().nextInt(20) + 18,
        km: Random().nextInt(30) + 1,
      ),
      onSlideUpdate: (SlideRegion? region) async {
        if (region == SlideRegion.inLikeRegion) {
          setState(() {
            isLike = true;
          });
        } else if (region == SlideRegion.inNopeRegion) {
          setState(() {
            isNope = true;
          });
        } else if (region == SlideRegion.inSuperLikeRegion) {
          setState(() {
            isSuperLike = true;
          });
        } else {
          setState(() {
            isLike = false;
            isNope = false;
            isSuperLike = false;
          });
        }
      },
    ),
```

```
  );

  setState(() {
    matchEngine = MatchEngine(swipeItems: swipeItems);
  });
  super.initState();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.white,
    body: SafeArea(
      child: Column(
        children: [
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 14),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                Image.asset(
                  'images/text_logo.png',
                  width: 100,
                ),
                const Icon(
                  CupertinoIcons.bell_fill,
                  color: Color(0xFF7E858F),
                ),
              ],
            ),
          ),
          const SizedBox(
            height: 6,
          ),
          Expanded(
            child: Stack(
              children: [
                Padding(
                  padding: const EdgeInsets.symmetric(
                    horizontal: 4,
                    vertical: 10,
                  ),
                  child: SwipeCards(
                    matchEngine: matchEngine,
                    itemBuilder: (BuildContext context, int index) {
                      return GestureDetector(
                        onTapDown: (details) {
                          var position = details.globalPosition;
                          if (position.dx <
                              MediaQuery.of(context).size.width / 2) {
                            if (imageIndexList[index] > 0) {
```

```
        setState(() {
          imageIndexList[index]--;
        });
      }
    } else {
      if (imageIndexList[index] <
          swipeItems[index].content.photos.length - 1) {
        setState(() {
          imageIndexList[index]++;
        });
      }
    }
  },
  child: Container(
    height: double.infinity,
    decoration: BoxDecoration(
      image: DecorationImage(
        image: CachedNetworkImageProvider(
          swipeItems[index]
              .content
              .photos[imageIndexList[index]],
        ),
        fit: BoxFit.cover,
      ),
      borderRadius: BorderRadius.circular(10),
      boxShadow: [
        BoxShadow(
          color: Colors.grey.withOpacity(0.7),
          blurRadius: 5,
          spreadRadius: 5,
          offset: const Offset(0, 2),
        ),
      ]),
    child: Column(
      children: [
        const SizedBox(
          height: 4,
        ),
        Padding(
          padding:
              const EdgeInsets.symmetric(horizontal: 8),
          child: Row(
            children: List.generate(
              swipeItems[index].content.photos.length,
              (stepIndex) => Expanded(
                child: Padding(
                  padding:
                      const EdgeInsets.only(right: 4),
                  child: Container(
                    height: imageIndexList[index] ==
                            stepIndex
```

```
                ? 2.5
                : 2,
            decoration: BoxDecoration(
              borderRadius:
                  BorderRadius.circular(2),
              color: imageIndexList[index] ==
                      stepIndex
                  ? Colors.white
                  : Colors.black
                      .withOpacity(0.2),
            ),
          ),
        ),
      ),
    ),
  ),
),
Expanded(
  child: Container(
    decoration: BoxDecoration(
      gradient: const LinearGradient(
        begin: Alignment.topCenter,
        end: Alignment.bottomCenter,
        colors: [
          Colors.transparent,
          Colors.black
        ],
        stops: [
          0.5,
          1.0,
        ],
      ),
      borderRadius: BorderRadius.circular(10),
    ),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.end,
      crossAxisAlignment:
          CrossAxisAlignment.start,
      children: [
        Padding(
          padding:
              const EdgeInsets.only(left: 18),
          child: RichText(
            text: TextSpan(
                text: swipeItems[index]
                        .content
                        .name +
                    " ",
                style: GoogleFonts.roboto(
                  color: Colors.white,
                  fontWeight: FontWeight.bold,
```

```
                    fontSize: 28,
                  ),
                  children: [
                   TextSpan(
                    text: swipeItems[index]
                      .content
                      .age
                      .toString(),
                    style: GoogleFonts.roboto(
                     color: Colors.white,
                     fontWeight:
                       FontWeight.w400,
                     fontSize: 20,
                    ),
                   )
                  ]),
               ),
              ),
                        onStackFinished: () {},
itemChanged: (SwipeItem item, int index) {
 setState(() {
  imageIndexList[index] = 0;
 });
},
likeTag: Container(
 padding: const EdgeInsets.symmetric(
  horizontal: 6,
  vertical: 2,
 ),
 decoration: BoxDecoration(
  color: Colors.transparent,
  border: Border.all(
   color: const Color(0xFF6EE6BA),
   width: 4,
  ),
  borderRadius: BorderRadius.circular(10),
 ),
 child: Text(
  "LIKE",
  style: GoogleFonts.roboto(
   fontWeight: FontWeight.bold,
   color: const Color(0xFF6EE6BA),
   fontSize: 40,
  ),
 ),
),
nopeTag: Container(
 padding: const EdgeInsets.symmetric(
  horizontal: 6,
  vertical: 2,
 ),
```

```
          decoration: BoxDecoration(
            color: Colors.transparent,
            border: Border.all(
              color: const Color(0xFFEC5E6A),
              width: 4,
            ),
            borderRadius: BorderRadius.circular(10),
          ),
          child: Text(
            "NOPE",
            style: GoogleFonts.roboto(
              fontWeight: FontWeight.bold,
              color: const Color(0xFFEC5E6A),
              fontSize: 40,
            ),
          ),
        ),
        superLikeTag: Transform.rotate(
          angle: -15 * pi / 180,
          child: Container(
            margin: const EdgeInsets.only(bottom: 70),
            padding: const EdgeInsets.symmetric(
              horizontal: 8,
              vertical: 2,
            ),
            decoration: BoxDecoration(
              color: Colors.transparent,
              border: Border.all(
                color: const Color(0xFF66C4F2),
                width: 4,
              ),
              borderRadius: BorderRadius.circular(10),
            ),
            child: Text(
              "SUPER\nLIKE",
              style: GoogleFonts.roboto(
                fontWeight: FontWeight.bold,
                color: const Color(0xFF66C4F2),
                fontSize: 40,
              ),
              textAlign: TextAlign.center,
            ),
          ),
        ),
        upSwipeAllowed: true,
        fillSpace: true,
      ),
    ),
    Positioned.fill(
      child: Align(
        alignment: Alignment.bottomCenter,
```

```
child: Padding(
  padding: const EdgeInsets.symmetric(
    horizontal: 10,
    vertical: 18,
  ),
  child: Row(
    children: [
      Expanded(
        child: Container(
          decoration: BoxDecoration(
            shape: BoxShape.circle,
            border: Border.all(
              color: Colors.white,
            ),
          ),
          padding: const EdgeInsets.all(12),
          child: SvgPicture.asset(
            'images/return.svg',
            color: Colors.white,
            width: 14,
          ),
        ),
      ),
      Expanded(
        child: Container(
          decoration: BoxDecoration(
            shape: BoxShape.circle,
            border: Border.all(
              color: const Color(0xFFEC5E6A),
            ),
            color: isNope
                ? const Color(0xFFEC5E6A)
                : Colors.transparent,
          ),
          padding: const EdgeInsets.all(14),
          child: SvgPicture.asset(
            'images/cross.svg',
            width: 24,
            color: isNope ? Colors.white : null,
          ),
        ),
      ),
      Expanded(
        child: Container(
          decoration: BoxDecoration(
            shape: BoxShape.circle,
            border: Border.all(
              color: const Color(0xFF66C4F2),
            ),
            color: isSuperLike
                ? const Color(0xFF66C4F2)
```

```
                        : Colors.transparent,
                    ),
                    padding: const EdgeInsets.all(8),
                    child: SvgPicture.asset(
                      'images/star.svg',
                      width: 20,
                      color: isSuperLike ? Colors.white : null,
                    ),
                  ),
                ),
                Expanded(
                  child: Container(
                    decoration: BoxDecoration(
                      shape: BoxShape.circle,
                      border: Border.all(
                        color: const Color(0xFF6EE6BA),
                      ),
                      color: isLike
                          ? const Color(0xFF6EE6BA)
                          : Colors.transparent,
                    ),
                    padding: const EdgeInsets.all(14),
                    child: SvgPicture.asset(
                      'images/heart.svg',
                      width: 26,
                      color: isLike ? Colors.white : null,
                    ),
                  ),
                ),
                Expanded(
                  child: Container(
                    decoration: BoxDecoration(
                      shape: BoxShape.circle,
                      border: Border.all(
                        color: const Color(0xFFA64AE9),
                      ),
                    ),
                    padding: const EdgeInsets.all(12),
                    child: SvgPicture.asset(
                      'images/thunder.svg',
                      width: 10,
                    ),
                  ),
                ),
              ],
            ),
          ),
        ),
      ),
    ],
  ),
```

```
),
Container(
  padding: const EdgeInsets.only(top: 14),
  decoration: const BoxDecoration(
   border: Border(
    top: BorderSide(
     color: Color(0xFFD0D2D8),
    ),
   ),
  ),
  child: Row(
   children: [
    Expanded(
     child: GestureDetector(
      onTap: () {
       Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => MainPage()),
       );
      },
      child: Image.asset(
       'images/logo.png',
       height: 20,
      ),
     ),
    ),
    Expanded(
     child: GestureDetector(
      onTap: () {
       Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => ExploreScreen()),
       );
      },
      child: SvgPicture.asset(
       'images/search.svg',
       width: 22,
      ),
     ),
    ),
    Expanded(
     child: GestureDetector(
      onTap: () {
       Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => ChatPage()),
       );
      },
      child: SvgPicture.asset(
       'images/star.svg',
```

```
                    width: 24,
                  ),
                ),
              ),
              Expanded(
                child: GestureDetector(
                  onTap: () {
                    Navigator.push(
                      context,
                      MaterialPageRoute(builder: (context) => ChatPage()),
                    );
                  },
                  child: SvgPicture.asset(
                    'images/chat.svg',
                    width: 24,
                  ),
                ),
              ),
              Expanded(
                child: GestureDetector(
                  onTap: () {
                    Navigator.push(
                      context,
                      MaterialPageRoute(
                          builder: (context) => AccountPage()),
                    );
                  },
                  child: SvgPicture.asset(
                    'images/person.svg',
                    width: 16,
                  ),
                ),
              ),
            ],
          )),
        ],
      ),
    ),
  );
 }
}
class TinderCard {
 final String name;
 final List<String> photos;
 final int age;
 final int km;
 final String? aboutMe;
 final List<String>? interests;

 TinderCard({
  required this.name,
```

```
  required this.photos,
  required this.age,
  required this.km,
  this.aboutMe,
  this.interests,
 });
}
```

Output:

Conclusion:

Flutter's widget architecture offers great flexibility for building complex UIs. Understanding key widgets and concepts is essential for effective Flutter development.

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	04
Name	Sarvadnya Rajendra Awaghad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15 M

Experiment - 3

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

Theory:

Fonts:

In Flutter, the TextStyle class is used to define the styling for text within the Text widget or other widgets that involve displaying text. Here's an overview of how you can use the TextStyle class to set various font-related properties

fontSize:
You can set the size of the font using the fontSize property.

fontWeight:
The fontWeight property allows you to set the thickness of the characters in the text.

fontStyle:
The fontStyle property lets you specify whether the text should be in normal, italic, or oblique style.

fontFamily:
You can specify the font family using the fontFamily property. This refers to the specific font you want to use, and it should be available in your project.

decoration:
The decoration property allows you to add decorations to the text, such as underline or overline.

1. Text

A Text widget holds some text to display on the screen. We can align the text
widget by using textAlign property, and style property allow the customization of Text that includes font, font weight, font style, letter spacing, color, and many more.

2. Button

This widget allows you to perform some action on click. Flutter does not allow
you to use the Button widget directly; instead, it uses a type of buttons like a FlatButton and a RaisedButton.

3. Image

This widget holds the image which can fetch it from multiple sources like from the asset folder or directly from the URL. It provides many constructors for loading image, which are given below:

o Image: It is a generic image loader, which is used by ImageProvider.

o asset: It load image from your project asset folder.

o file: It loads images from the system folder.

o memory: It load image from memory.

o network: It loads images from the network.

To add an image in the project, you need first to create an assets folder

where
you keep your images and then add the below line in pubspec.yaml file.

assets:
- assets/images

```
# The following section is specific to Flutter packages.
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true

  # To add assets to your application, add an assets section, like this:
  assets:
    - images/
```

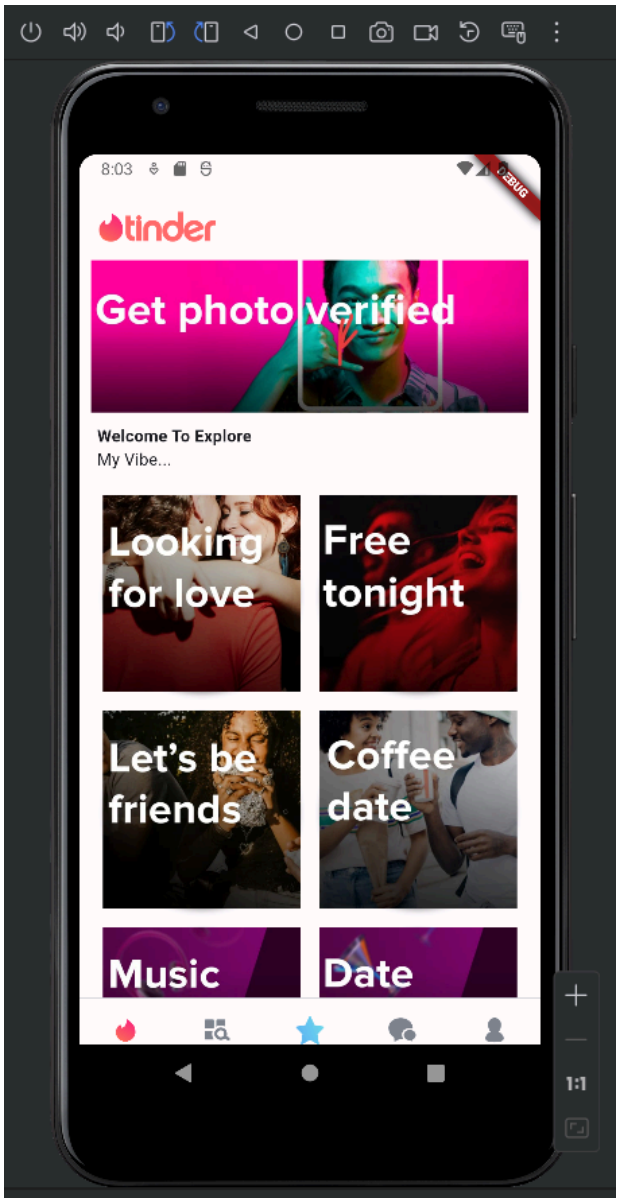
Code:
explore_screen.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_svg/flutter_svg.dart';
import 'package:get/get.dart';
import 'package:tinder/authenticationScreen/chat_screen.dart';
import 'package:tinder/authenticationScreen/explore_screen.dart';
import 'package:tinder/authenticationScreen/main_page.dart';
import 'package:tinder/authenticationScreen/profile_screen.dart';
void main() {
 runApp(ExploreScreen());
}

class ExploreScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Tinder Clone',
    theme: ThemeData(
      primarySwatch: Colors.blue,
      visualDensity: VisualDensity.adaptivePlatformDensity,
    ),
    home: InterestSelectionScreen(),
  );
```

```
 }
}

class InterestSelectionScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Column(
        mainAxisAlignment: MainAxisAlignment.start,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          const SizedBox(
            height: 20,
          ),
          Image.asset(
            'images/text_logo.png',
            width: 100,
          ),
        ],
      ),
    ),
    body: Column(
      mainAxisAlignment: MainAxisAlignment.start,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Container(
          // Add a container for the rectangular image
          padding: EdgeInsets.all(10),
          height: 150, // Set the desired height
          width: double.infinity,
          child: Image.asset(
            'images/verified.jpg',
            fit: BoxFit.cover,
          ),
        ),
        Padding(
          padding:
              const EdgeInsets.only(left: 15.0), // Adjust the left padding
          child: Text(
            'Welcome To Explore',
            style: TextStyle(
              fontSize: 14,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),

  Expanded(
        child: SingleChildScrollView(
          child: Padding(
```

```
            padding: const EdgeInsets.all(16.0),
            child: GridView.builder(
              gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
                crossAxisCount: 2,
                crossAxisSpacing: 8.0,
                mainAxisSpacing: 8.0,
              ),
              shrinkWrap: true,
              physics: NeverScrollableScrollPhysics(),
              itemCount: 8,
              itemBuilder: (context, index) {
                return InterestTile(image: 'images/image$index.jpg');
              },
            ),
          ),
        ),
      ),
    ],
  ),
 class InterestTile extends StatelessWidget {
final String image;

InterestTile({required this.image});

@override
Widget build(BuildContext context) {
 return GestureDetector(
   onTap: () {
     Navigator.push(
       context,
       MaterialPageRoute(builder: (context) => MainPage()),
     );
   },
   child: Card(
     elevation: 3,
     shape: RoundedRectangleBorder(
       borderRadius: BorderRadius.circular(105.0),
     ),
     child: Container(
       height: 200,
       width: double.infinity,
       child: Image.asset(
         image,
         fit: BoxFit.cover,
       ),
     ),
   ),
 );
}
}
```

Output:



Conclusion:

In this experiment, we have successfully imported and inserted image in the flutter and used font style to enter text and successfully created button for it. All concept of image , font are implemented successfully.

MAD & PWA Lab
Journal

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

EXP 4

Aim : To create an interactive Form using form widget

Theory:

Form Widgets:

Form widgets are essential components of interactive forms, offering a range of input elements such as text fields, checkboxes, radio buttons, dropdown menus, and more. These widgets empower developers to design forms that cater to specific data input requirements. The flexibility of form widgets allows for the creation of dynamic and user-friendly interfaces, ensuring that the form adapts to the user's needs.

Form Inputs:

Text Fields:

Purpose: Allow users to input general text information.
Attributes: May include specifications such as maximum length, placeholder text, and input type (e.g., email, password).

Checkboxes:

Purpose: Enable users to make multiple selections from a list of options.
Attributes: Each checkbox typically represents a distinct option, and users can choose multiple checkboxes simultaneously.

Radio Buttons:

Purpose: Provide users with exclusive choices within a group.
Attributes: Users can select only one option from the group, making radio buttons suitable for mutually exclusive selections.

Dropdown Menus:

Purpose: Offer a space-efficient way to present a list of options for selection.
Attributes: Users click on a dropdown menu to reveal a list of choices, selecting one option from the list.

Textareas:

Purpose: Allow users to input multiline text, suitable for longer responses or comments.
Attributes: Can include settings for the number of rows and columns to determine the size of the textarea.

Date Pickers:

Purpose: Facilitate the selection of dates.
Attributes: Users can choose a specific date from a calendar interface, helping to ensure accurate date input.

File Upload:

Purpose: Enable users to submit files (e.g., images, documents).
Attributes: May include file type restrictions, maximum file size, and a browse button for users to locate and upload files from their device.

Code

login_screen.dart

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:tinder/authenticationScreen/registration_screen.dart';
import 'package:tinder/widgets/custom_text_field_widget.dart';

class LoginScreen extends StatefulWidget {
 const LoginScreen({Key? key}) : super(key: key);
```

```
 @override
 _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
 TextEditingController emailTextEditingController = TextEditingController();
 TextEditingController passwordTextEditingController = TextEditingController();
 bool showProgressBar = false;

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    body: Container(
      decoration: BoxDecoration(
        gradient: LinearGradient(
          colors: [Colors.pink, Colors.orange],
          begin: Alignment.topLeft,
          end: Alignment.bottomRight,
        ),
      ),
      child: SingleChildScrollView(
        child: Center(
          child: Column(
            children: [
              const SizedBox(
                height: 220,
              ),
              Image.asset(
                'images/tinder_logo.png',
                width: 100,
              ),
              const SizedBox(
                height: 60,
              ),
              Text(
                "By clicking Login",
                style: TextStyle(
                  fontSize: 13,
                ),
              ),
              Text(
                " you agree with our terms and conditions.",
                style: TextStyle(
                  fontSize: 13,
                ),
```

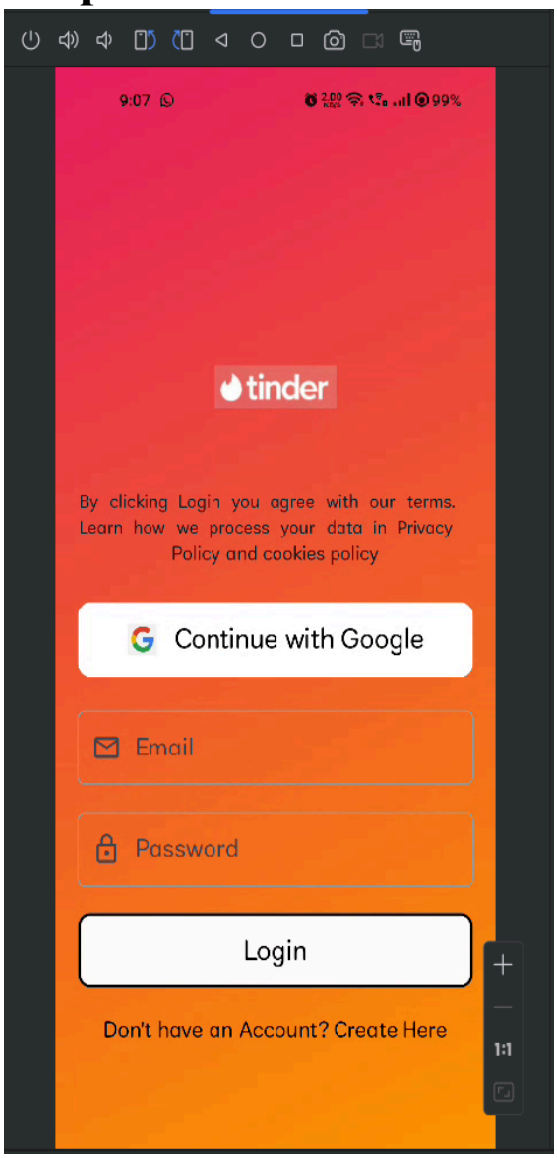
```
),
Text(
  " Learn how we process your data in Privacy Policy and cookies policy",
  style: TextStyle(
    fontSize: 12,
  ),
),
const SizedBox(
  height: 25,
),
GestureDetector(
  onTap: () {
    // Implement your Google login logic here
  },
  child: Container(
    width: MediaQuery.of(context).size.width - 36,
    height: 55,
    decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(10),
    ),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const SizedBox(width: 25),
        Image.asset(
          'images/google.jpeg', // Replace with the actual path to your Google logo image
          width: 24, // Adjust the width as needed
          height: 24, // Adjust the height as needed
        ),
        const SizedBox(width: 15),
        Text(
          "Continue with Google",
          style: TextStyle(
            fontSize: 20,
            color: Colors.black,
            fontWeight: FontWeight.bold,
          ),
        ),
      ],
    ),
  ),
),
const SizedBox(
  height: 25,
),
```

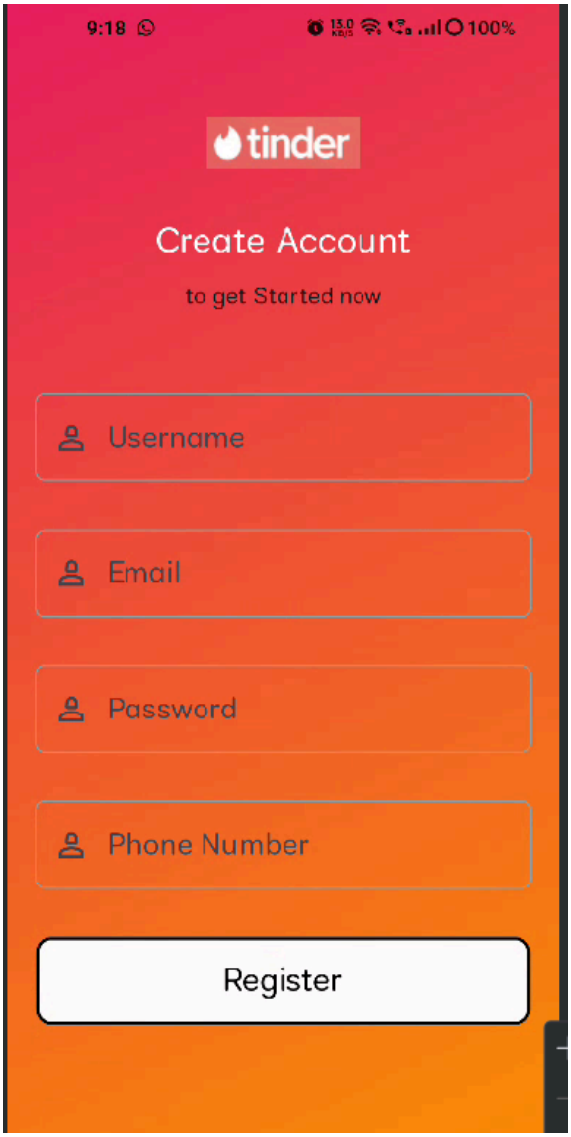
```
SizedBox(
  width: MediaQuery.of(context).size.width - 36,
  height: 55,
  child: CustomTextFieldWidget(
    editingController: emailTextEditingController,
    labelText: 'Email',
    iconData: Icons.email_outlined,
    isObscure: false,
  ),
),
const SizedBox(
  height: 20,
),
SizedBox(
  width: MediaQuery.of(context).size.width - 36,
  height: 55,
  child: CustomTextFieldWidget(
    editingController: passwordTextEditingController,
    labelText: 'Password',
    iconData: Icons.lock_outline,
    isObscure: true,
  ),
),
const SizedBox(
  height: 20,
),
GestureDetector(
  onTap: () {
    // Implement your login logic here
    // For example, you can use Get.to(HomeScreen());
  },
  child: Container(
    width: MediaQuery.of(context).size.width - 36,
    height: 55,
    decoration: BoxDecoration(
      gradient: LinearGradient(
        colors: [Colors.pink, Colors.orange],
        begin: Alignment.topLeft,
        end: Alignment.bottomRight,
      ),
      borderRadius: BorderRadius.circular(10),
    ),
    child: const Center(
      child: Text(
        "Login",
        style: TextStyle(
```

```
                fontSize: 20,
                color: Colors.white,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
        ),
      ),
      const SizedBox(
        height: 20,
      ),
      Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text(
            'Don\'t have an Account?',
            style: TextStyle(
              fontSize: 16,
              color: Colors.black,
            ),
          ),
          InkWell(
            onTap: () {
              Get.to(RegistrationScreen());
            },
            child: const Text(
              'Create Here',
              style: TextStyle(
                fontSize: 16,
                color: Colors.black,
                fontWeight: FontWeight.bold,
              ),
            ),
          )
        ],
      ),
      const SizedBox(
        height: 40,
      ),
      showProgressBar
          ? const CircularProgressIndicator(
              valueColor: AlwaysStoppedAnimation<Color>(Colors.pink),
            )
          : Container()
    ],
  ),
```

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

Output:





Conclusion: In this experiment , we have successfully created form using form widget and create a login page for my clone application, various properties of form are implemented successfully in the above experiment.

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	04
Name	Sarvadnya Rajendra Awaghad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15 M

Experiment - 5

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

Theory:

Navigation:
Navigation is a fundamental aspect of mobile app development that involves transitioning between different screens or pages. In Flutter, the Navigator class plays a central role in managing the navigation stack, allowing developers to push and pop routes as users move through the app.

Navigator Class:
The Navigator class handles the navigation stack, maintaining a history of routes.
The push method adds a new route to the stack, typically triggered by user actions.
The pop method removes the current route from the stack, enabling backward navigation.
Routing:
Routing in Flutter involves defining and organizing the paths or routes within the application. Routes represent different screens or pages, and their effective use is crucial for structuring the app's architecture.

MaterialPageRoute and CupertinoPageRoute:

MaterialPageRoute is employed in apps following Material Design principles, providing a standard Android-style transition between screens.
CupertinoPageRoute is used for iOS-style designs, ensuring a consistent and native user experience.

Named Routes:

Named routes offer a more organized and readable approach to navigation.
By assigning names to routes, such as '/details', developers can easily navigate to specific screens using Navigator.pushNamed.

Gestures in Flutter:
Gestures enhance user interaction by allowing the app to respond to touch or mouse inputs. In Flutter, the GestureDetector widget is instrumental in recognizing and handling various gestures.

GestureDetector Widget:

The GestureDetector widget is used to detect a variety of gestures, including taps, drags, and long presses.
By wrapping UI components with GestureDetector, developers can specify callback functions to execute when specific gestures are detected.
Gesture Recognizers:

Flutter provides gesture recognizers for more complex gestures, such as panning, pinching, and swiping.
These recognizers, when combined with a GestureDetector, enable the app to respond to a broader range of user inputs.
InkWell and InkWell:

The InkWell and InkResponse widgets bring a material ripple effect to touchable UI components.
Integrating these widgets enhances the visual feedback during user interactions, contributing to a more polished and intuitive user experience.

Code:
profile_screen.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_custom_clippers/flutter_custom_clippers.dart';
import 'package:flutter_svg/flutter_svg.dart';
import 'package:tinder/authenticationScreen/chat_screen.dart';
import 'package:tinder/authenticationScreen/explore_screen.dart';
import 'package:tinder/authenticationScreen/main_page.dart';
import 'package:tinder/data/account_json.dart';
import 'package:tinder/authenticationScreen/colors.dart';

class AccountPage extends StatefulWidget {
 @override
 _AccountPageState createState() => _AccountPageState();
}

class _AccountPageState extends State<AccountPage> {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.white, // Set background color to white
    body: getBody(),
    bottomNavigationBar: getBottomNavigationBar(),
  );
 }

 Widget getBody() {
  var size = MediaQuery.of(context).size;
  return Column(
    children: [
      ClipPath(
        clipper: OvalBottomBorderClipper(),
        child: Container(
          width: size.width,
          height: size.height * 0.6,
          decoration: BoxDecoration(color: white, boxShadow: [
            BoxShadow(
              color: Colors.grey,
              spreadRadius: 10,
              blurRadius: 10,
            ),
          ]),
          child: Padding(
            padding: const EdgeInsets.only(left: 30, right: 30, bottom: 40),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.end,
              children: [
                Row(
                  children: [
                    Expanded(
```

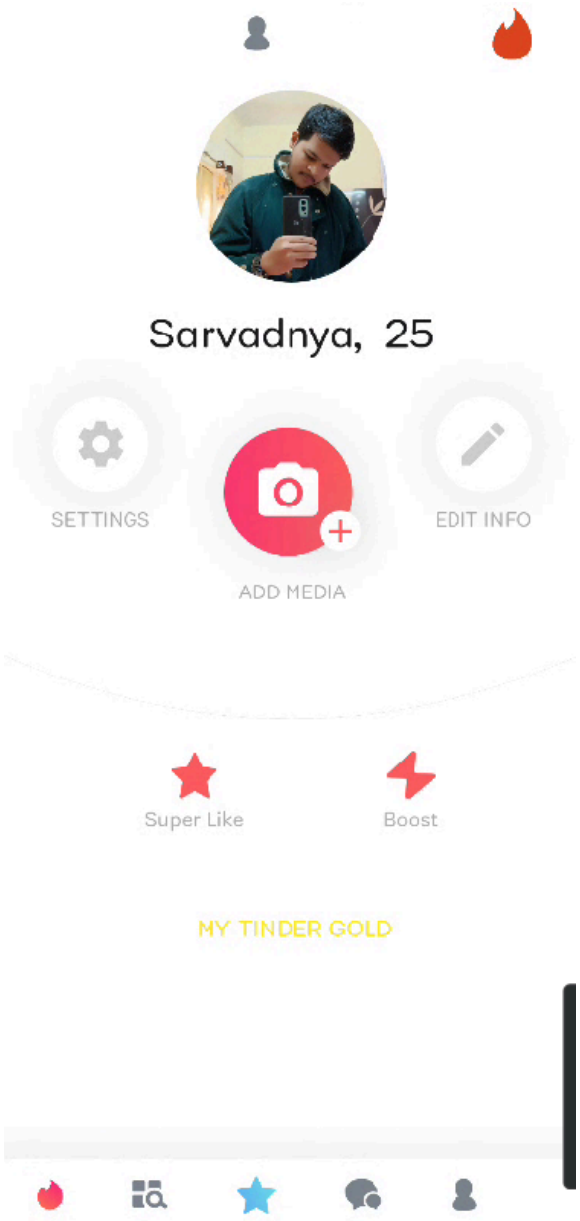
```
        child:
            Container(), // This container takes the left space
      ),
      GestureDetector(
        onTap: () {
          // Add your logic for the person SVG tap
        },
        child: SvgPicture.asset(
          'images/person.svg',
          width: 16,
        ),
      ),
      SizedBox(
          width:
              20), // Add spacing between person and tinder icons
      Expanded(
        child:
            Container(), // This container takes the remaining space
      ),
      GestureDetector(
        onTap: () {
          // Add your logic for the tinder Image tap
        },
        child: Image.asset(
          'images/tinder_logo2.png', // Replace with the actual path to your tinder Image file
          width: 24,
        ),
      ),
    ],
  ),
  SizedBox(height: 20),
  Container(
    width: 120,
    height: 120,
    decoration: BoxDecoration(
        shape: BoxShape.circle,
        image: DecorationImage(
            image: AssetImage(account_json[0]['img']),
            fit: BoxFit.cover)),
  ),
  SizedBox(
    height: 15,
  ),
  Text(
    account_json[0]['name'] + ", " + account_json[0]['age'],
    style: TextStyle(fontSize: 25, fontWeight: FontWeight.w600),
  ),
  SizedBox(
    height: 20,
  ),
  Row(
```

```
            crossAxisAlignment: CrossAxisAlignment.start,
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              Column(
                children: [
                  Container(
                    width: 60,
                    height: 60,
                    decoration: BoxDecoration(
                        shape: BoxShape.circle,
                        color: white,
                        boxShadow: [
                          BoxShadow(
                            color: grey.withOpacity(0.1),
                            spreadRadius: 10,
                            blurRadius: 15,
                          ),
                        ]),
                    child: Icon(
                      Icons.settings,
                      size: 35,
                      color: grey.withOpacity(0.5),
                    ),
                  ),
                  SizedBox(
                    height: 10,
                  ),
                  Text(
                    "SETTINGS",
                    style: TextStyle(
                        fontSize: 12,
                        fontWeight: FontWeight.w600,
                        color: grey.withOpacity(0.8)),
                  )
                ],
              ),

Widget getBottomNavigationBar() {
  return Container(
    padding: EdgeInsets.symmetric(horizontal: 20, vertical: 10),
    decoration: BoxDecoration(
      color: white,
      boxShadow: [
        BoxShadow(
          color: grey.withOpacity(0.1),
          spreadRadius: 10,
          blurRadius: 10,
        ),
      ],
    ),
    child: Row(
```

```
mainAxisAlignment: MainAxisAlignment.spaceBetween,
children: [
 GestureDetector(
  onTap: () {
   Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => MainPage()),
   );
  },
  child: Image.asset(
   'images/logo.png',
   height: 20,
  ),
 ),
 GestureDetector(
  onTap: () {
   Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => ExploreScreen()),
   );
  },
  child: SvgPicture.asset(
   'images/search.svg',
   width: 22,
  ),
 ),
 GestureDetector(
  onTap: () {
   Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => ChatPage()),
   );
  },
  child: SvgPicture.asset(
   'images/star.svg',
   width: 24,
  ),
 ),
 GestureDetector(
  onTap: () {
   Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => ChatPage()),
   );
  },
  child: SvgPicture.asset(
   'images/chat.svg',
   width: 24,
  ),
 ),
 Row(
```

```
          mainAxisAlignment: MainAxisAlignment.end,
          children: [
            GestureDetector(
              onTap: () {
                // Add your logic for the person SVG tap
              },
              child: SvgPicture.asset(
                'images/person.svg',
                width: 16,
              ),
            ),
            SizedBox(
                width: 20), // Add spacing between person and tinder icons
            GestureDetector(
              onTap: () {
                // Add your logic for the tinder SVG tap
              },
              child: SvgPicture.asset(
                'path/to/tinder.svg', // Replace with the actual path to your tinder SVG file
                width: 24,
              ),
            ),
          ],
        ),
      ],
    ),
  );
 }
}
```

Output:



Conclusion:

In this experiment, we have successfully created routing in the bottom navigation bar and connected all pages successfully using Navigator class and implemented it successfully.

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	04
Name	Sarvadnya Rajendra Awaghad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	15 M

Experiment - 6

Aim: To Connect Flutter UI with FireBase database

Theory:

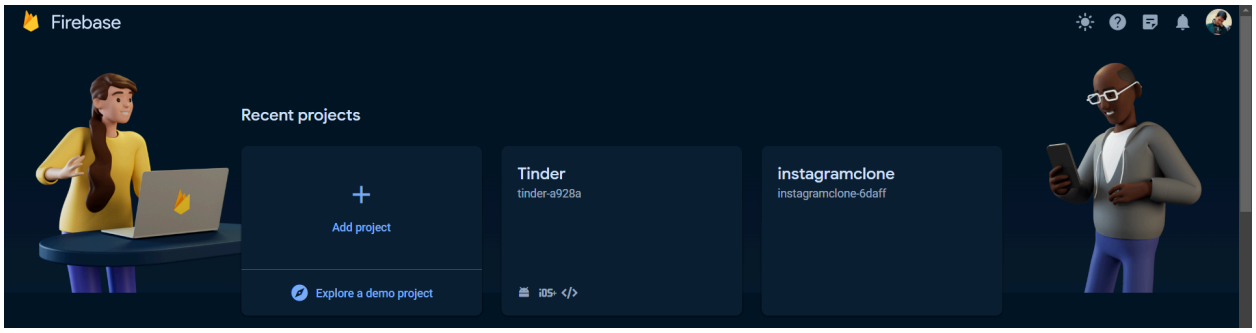
Prerequisites

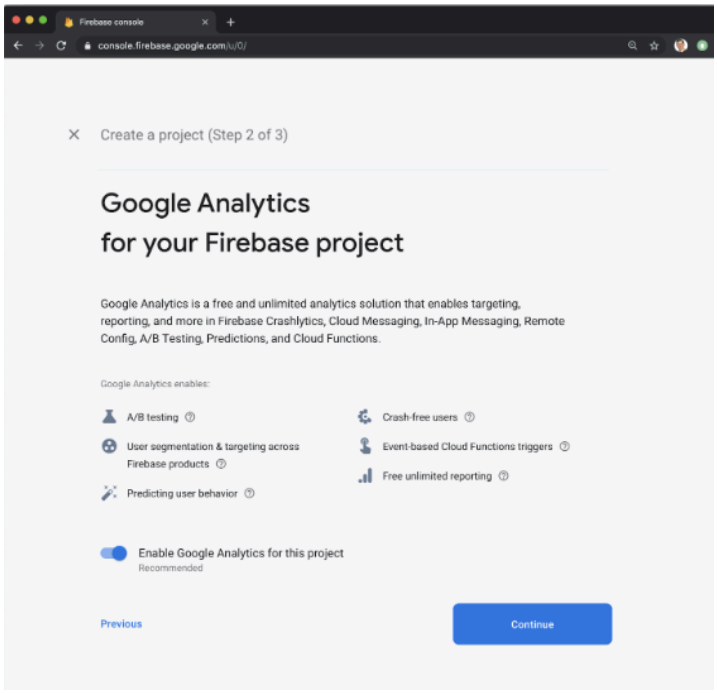
To complete this tutorial, you will need:

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

Create a Firebase Project:

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





Go to the Firebase Console and create a new project.
Add your Flutter app to the Firebase project:

Register your app in the Firebase project, and follow the instructions to download the configuration files (google-services.json for Android, GoogleService-Info.plist for iOS).

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

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

com.example.flutterfirebaseexample

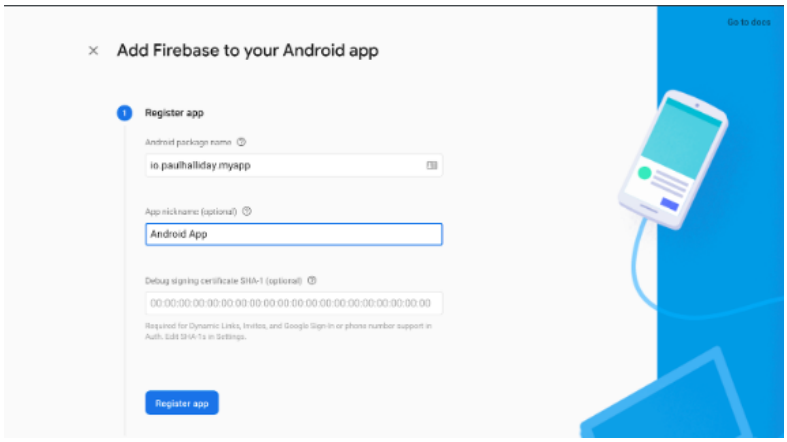
Once you've decided on a name, open android/app/build.gradle in your code editor and update the applicationId to match the Android package name:
android/app/build.gradle

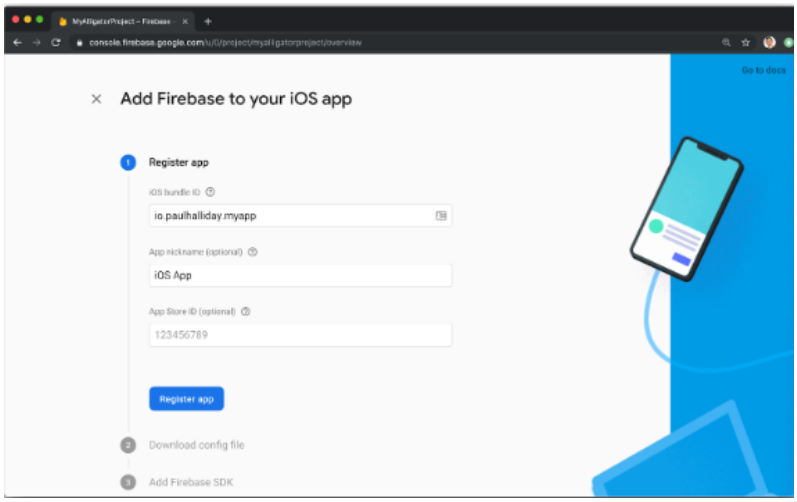
```
...
defaultConfig {
    // TODO: Specify your own unique Application ID
(https://developer.android.com/studio/build/application-id.html).
    applicationId 'com.example.flutterfirebaseexample'
    ...
}
```

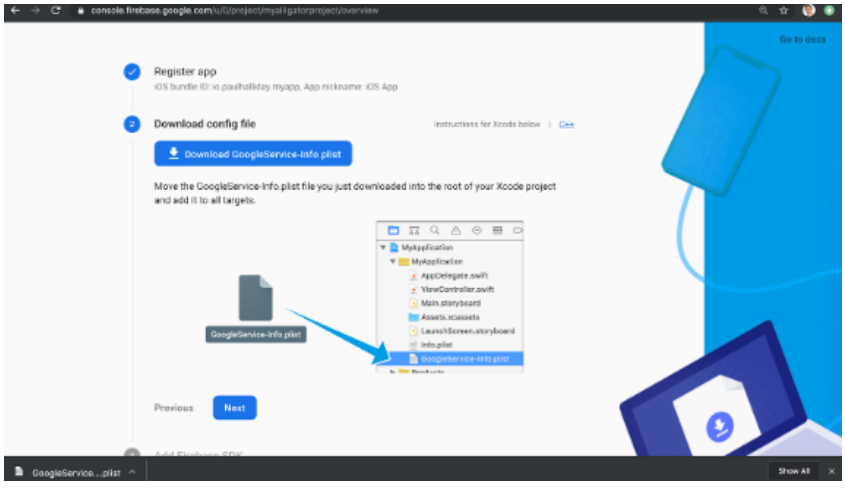
Downloading the Config File

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

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







2. Add Firebase to your Flutter project:
Add Dependencies:

Open your pubspec.yaml file and add the necessary dependencies:
yaml

```yaml
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2

  swipe_cards: ^2.0.0+1
  google_fonts: ^6.1.0
  flutter_svg: ^2.0.9
  cached_network_image: ^3.3.1
  font_awesome_flutter: ^10.7.0
  faker: ^2.1.0
  flutter_custom_clippers: ^2.1.0
  firebase_core: ^2.25.3
  firebase_auth: ^4.17.3
  cloud_firestore: ^4.15.4
  google_sign_in: ^6.2.1
```

Code:
main.dart

```
import 'package:firebase_core/firebase_core.dart';
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```

login.dart

```
import 'dart:developer';
```

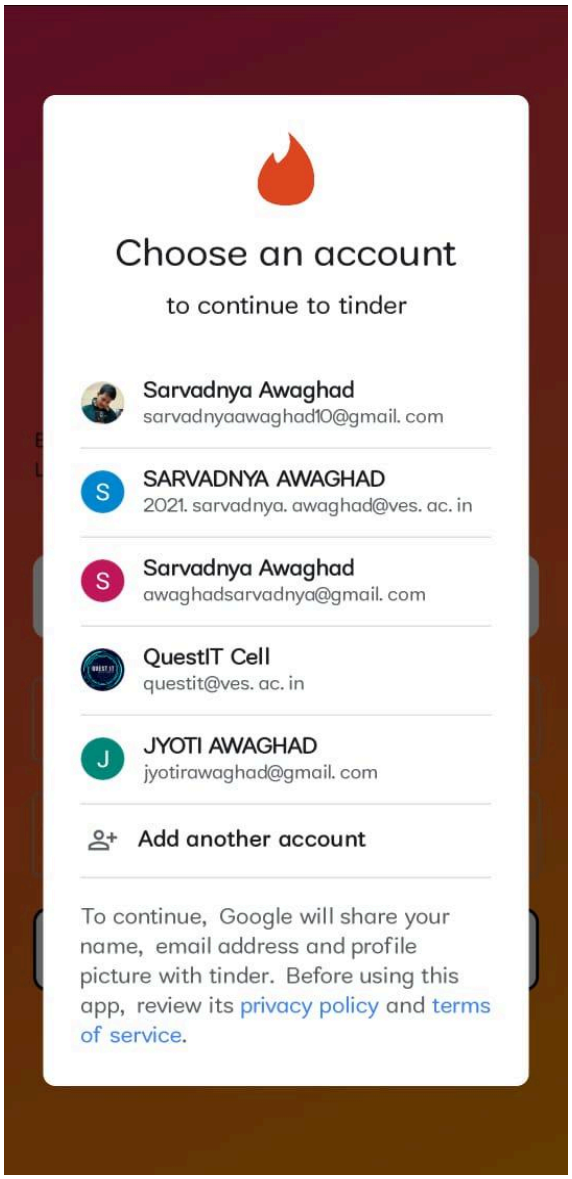
```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:tinder/authenticationScreen/main_page.dart';
import 'package:tinder/authenticationScreen/registration_screen.dart';
import 'package:tinder/widgets/custom_text_field_widget.dart';
class LoginScreen extends StatefulWidget {
  const LoginScreen({Key? key}) : super(key: key);
  @override
  _LoginScreenState createState() => _LoginScreenState();
}
class _LoginScreenState extends State<LoginScreen> {
  TextEditingController emailTextEditingController =
TextEditingController();
  TextEditingController passwordTextEditingController =
TextEditingController();
  bool showProgressBar = false;
  void login() async {
    String email = emailTextEditingController.text.trim();
    String password = passwordTextEditingController.text.trim();
    if (email == "" || password == "") {
      print('please fill all details');
    } else {
      try {
        UserCredential userCredential = await FirebaseAuth.instance
            .signInWithEmailAndPassword(email: email, password:
password);
        if (userCredential != null) {
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => MainPage()),
          );
        }
      } on FirebaseAuthException catch (ex) {
```

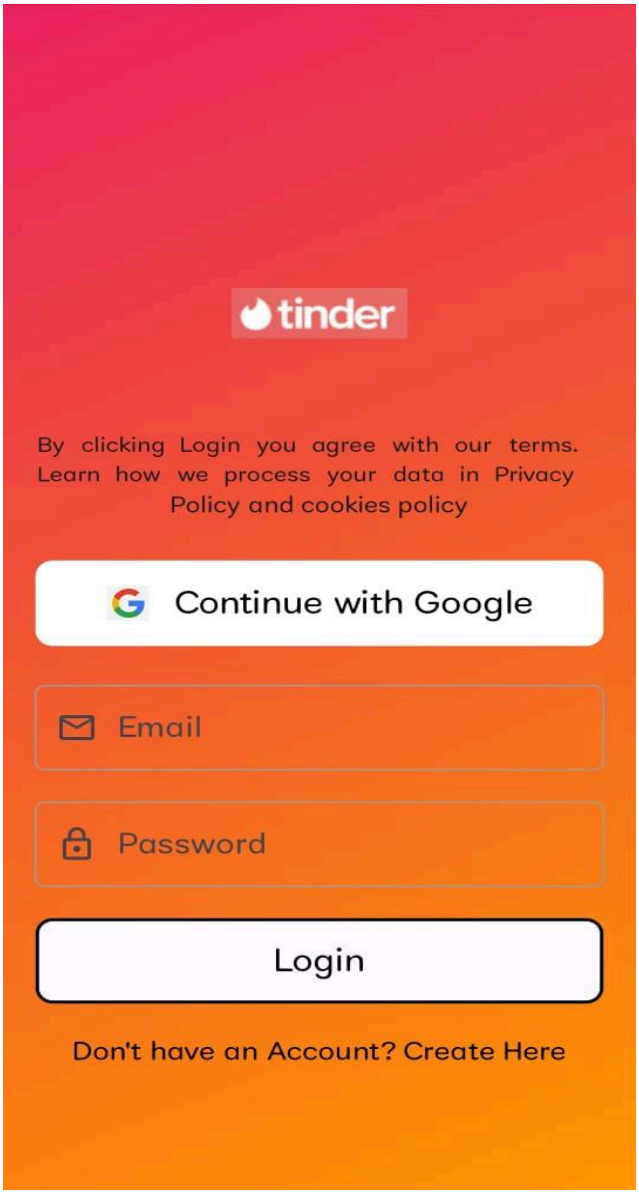
```
      log(ex.code.toString());
    }
  }
}
_signInWithGoogle() async {
  final GoogleSignIn _googleSignIn = GoogleSignIn();
  try {
    final GoogleSignInAccount? googleSignInAccount =
        await _googleSignIn.signIn();
    if (googleSignInAccount != null) {
      final GoogleSignInAuthentication googleSignInAuthentication =
          await googleSignInAccount.authentication
      final AuthCredential credential = GoogleAuthProvider.credential(
        idToken: googleSignInAuthentication.idToken,
        accessToken: googleSignInAuthentication.accessToken,
      );
      await FirebaseAuth.instance.signInWithCredential(credential);
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => MainPage()),
      );
    }
  } catch (e) {
    log("some error occurred $e");
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Container(
      decoration: BoxDecoration(
        gradient: LinearGradient(
          colors: [Colors.pink, Colors.orange],
```

```
          begin: Alignment.topLeft,
          end: Alignment.bottomRight,
        ),
      ),
      child: SingleChildScrollView(
        child: Center(
          child: Column(
            children: [
              const SizedBox(
                height: 220,
              ),
              Image.asset(
                'images/tinder_logo.png',
                width: 100,
              ),
              const SizedBox(
                height: 60,
              ),
              Center(
                child: Padding(
                  padding: EdgeInsets.symmetric(
                    horizontal: 20,
                  ),
                  child: Text(
                    "By clicking Login you agree with our terms. Learn how
we process your data in Privacy ",
                    style: TextStyle(
                      fontSize: 13,
                      wordSpacing: 5,
                    ),
                  ),
                ),
              ),
              Text('Policy and cookies policy'),
```

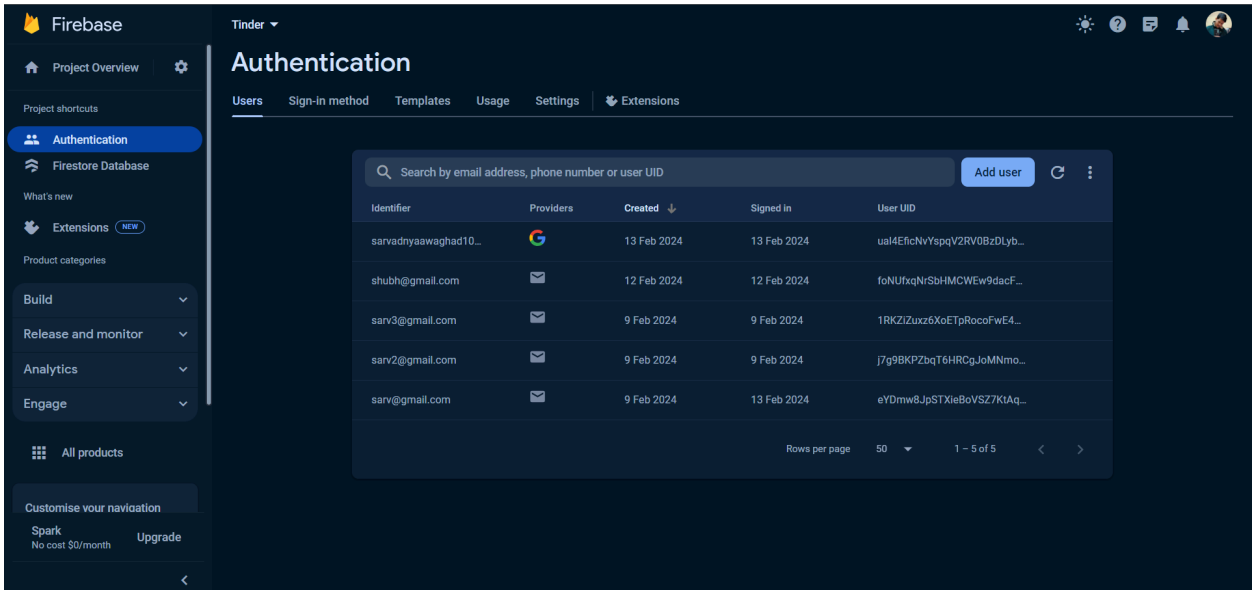
```
              const SizedBox(
                height: 25,
              ),
              ElevatedButton(
                onPressed: () {
                  // Implement your Google login logic here
                  _signInWithGoogle();
                },
                style: ElevatedButton.styleFrom(
                  primary: Colors.white,
                  onPrimary: Colors.black,
                  elevation: 0,
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(10),
                  ),
                )
              showProgressBar
                  ? const CircularProgressIndicator(
                      valueColor:
AlwaysStoppedAnimation<Color>(Colors.pink),
                    )
                  : Container()
            ],
          ),
        ),
      ),
    ),
  );
 }
}
```

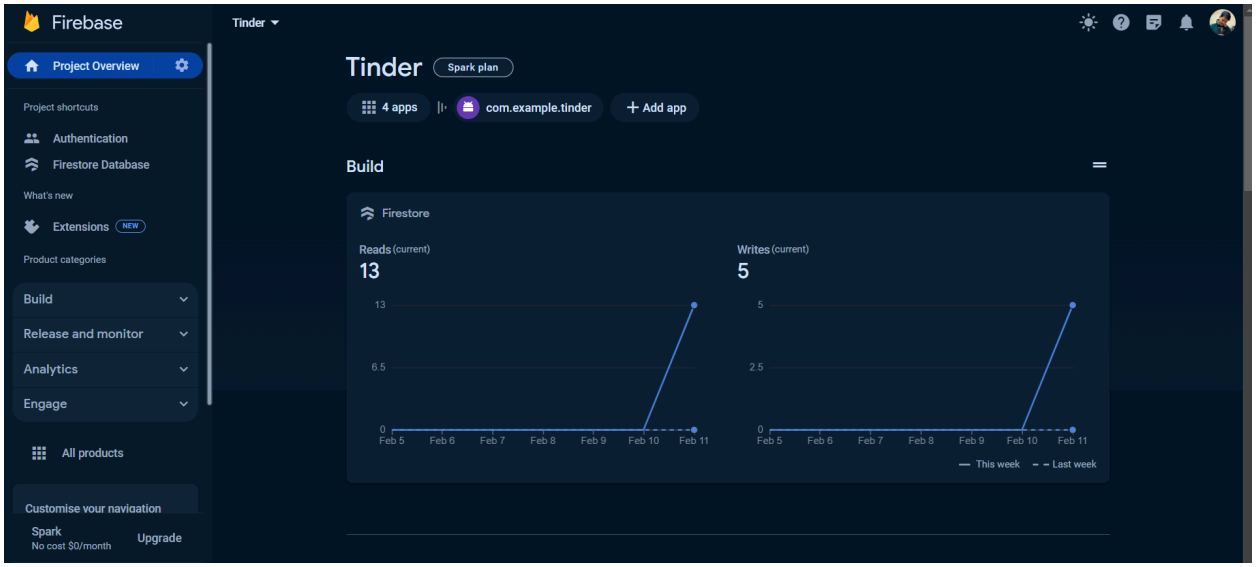
Output:


Choose an account
to continue to tinder
Sarvadnya Awaghad
sarvadnyaawaghad10@gmail. com
SARVADNYA AWAGHAD
2021. sarvadnya. awaghad@ves. ac. in
Sarvadnya Awaghad
awaghadsarvadnya@gmail. com
QuestIT Cell
questit@ves. ac. in
JYOTI AWAGHAD
jyotirawaghad@gmail. com
Add another account
To continue, Google will share your name, email address and profile picture with tinder. Before using this app, review its privacy policy and terms of service.


tinder
By clicking Login you agree with our terms. Learn how we process your data in Privacy Policy and cookies policy
Continue with Google
Email
Password
Login
Don't have an Account? Create Here

Project Title: Flutter : Tinder Clone / PWA : Dormdine **Roll No. 04**





Conclusion:

In this experiment, we have successfully connected firebase database and authenticated using google signin and email and password with out flutter application successfully.

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.	04
Name	Sarvadnya Rajendra Awaghad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15 M

Experiment - 7

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

Theory:

Regular Web App

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

Progressive Web App

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

Difference between PWAs vs. Regular Web Apps:

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

1. Native Experience

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

2. Ease of Access

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

3. Faster Services

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

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

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

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.

Code:

```
import Footer from './components/footer/Footer';
import Hero from './components/hero/Hero';
import Navbar from './components/navbar/Navbar';
import Newsletter from './components/newsletter/Newsletter';
import PopularProperties from './components/popularProperties/PopularProperties';
import Signin from './components/signin/Signin';
import Signup from './components/signup/Signup';
import Properties from './components/properties/Properties';
import PropertyDetail from './components/propertyDetail/PropertyDetail';
import { useSelector } from 'react-redux'
import { Routes, Route, Navigate, useLocation } from 'react-router-dom'
import { useEffect } from 'react';
import EditProperty from './components/editProperty/EditProperty';
import Yachts from './components/yachts/Yachts';
import YachtDetails from './components/yachtDetails/YachtDetails';
import CreateYacht from './components/createYacht/CreateYacht';
import YachtEdit from './components/yachtEdit/YachtEdit';
import MyProfile from './components/myProfile/MyProfile';
import UpdateProfile from './components/updateProfile/UpdateProfile';
import './App.css';
import NotFound from './components/notFound/NotFound';
import Layout from './components/Layout';
import Services from './components/Services/Services';


function App() {
  const { user } = useSelector((state) => state.auth)
  const url = useLocation().pathname

  useEffect(() => {
    url && window.scrollTo(0, 0)
  }, [url])

  return (
    <div>
      <Routes>
```

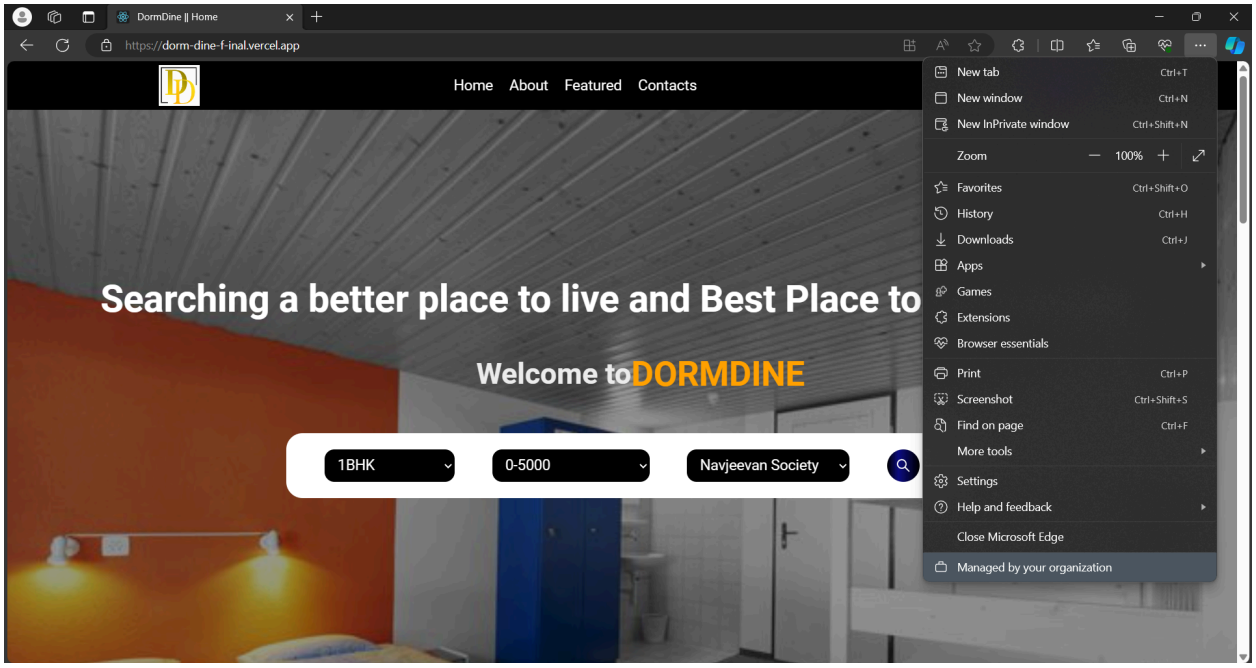
```
<Route path='/' element={
  <Layout title="DormDine || Home">
    <Navbar />
    <Hero />
    <PopularProperties />
    <Services/>
    <Newsletter />
    <Footer />
  </Layout>
} />
<Route path='/signup' element={!user ? <Signup /> : <Navigate to='/' />} />
<Route path='/signin' element={!user ? <Signin /> : <Navigate to='/' />} />
<Route path='/properties' element={
  <>
    <Navbar />
    <Properties />
    <Footer />
  </>
} />
<Route path='/yachts' element={user ?
  <>
    <Navbar />
    <Yachts />
    <Footer />
  </>
  : <Navigate to='/signin' />} />
<Route path='/yacht/:id' element={user ?
  <>
    <Navbar />
    <YachtDetails />
    <Footer />
  </>
  : <Navigate to='/signin' />} />
<Route path='/create-yacht' element={user ?
  <>
    <Navbar />
    <CreateYacht />
    <Footer />
  </>
  : <Navigate to='/signin' />} />
```

```
<Route path='/yacht-edit/:id' element={user ?
  <>
    <Navbar />
    <YachtEdit />
    <Footer />
  </>
  : <Navigate to='/signin' />} />
<Route path='/propertyDetail/:id' element={
  <>
    <Navbar />
    <PropertyDetail />
    <Footer />
  </>
} />
<Route path='/editproperty/:id' element={
  user ?
    <>
      <Navbar />
      <EditProperty />
      <Footer />
    </>
    : <Navigate to='/signin' />
} />
<Route path='/my-profile' element={
  user ?
    <>
      <Navbar />
      <MyProfile />
      <Footer />
    </>
    : <Navigate to='/signin' />
} />
<Route path='/update-profile' element={
  user ?
    <>
      <Navbar />
      <UpdateProfile />
      <Footer />
    </>
    : <Navigate to='/signin' />
```

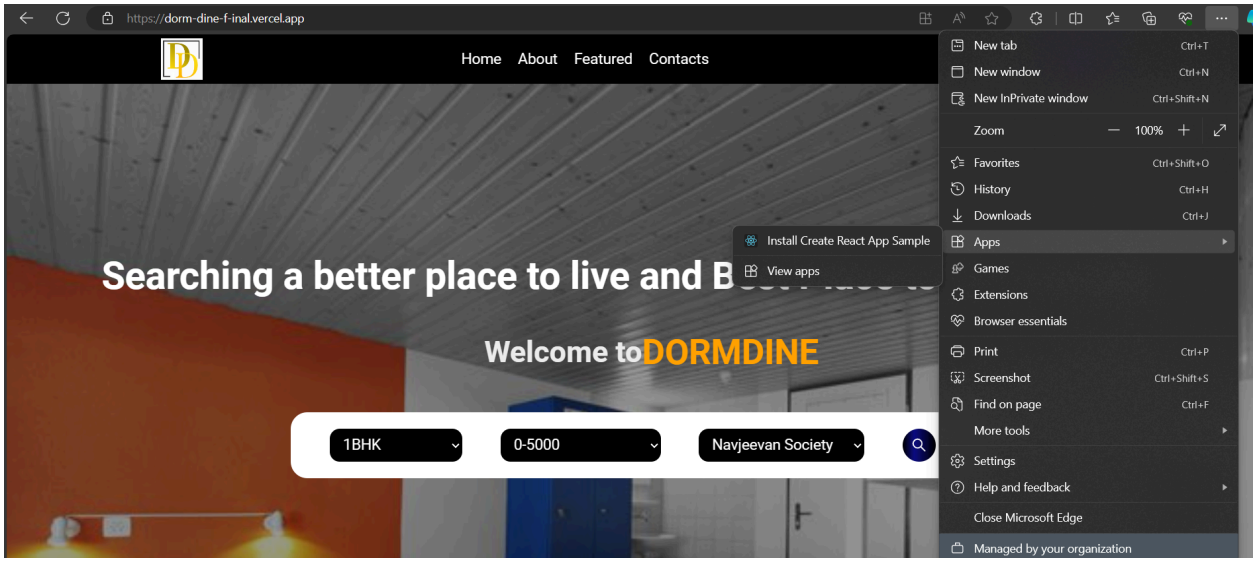
```
        } />
        <Route path='*' element={
          <>
            <Navbar />
            <NotFound />
            <Footer />
          </>
        } />
      </Routes>
    </div>
  );
}

export default App;
```

Open folder in VS code and click go live at bottom right corner

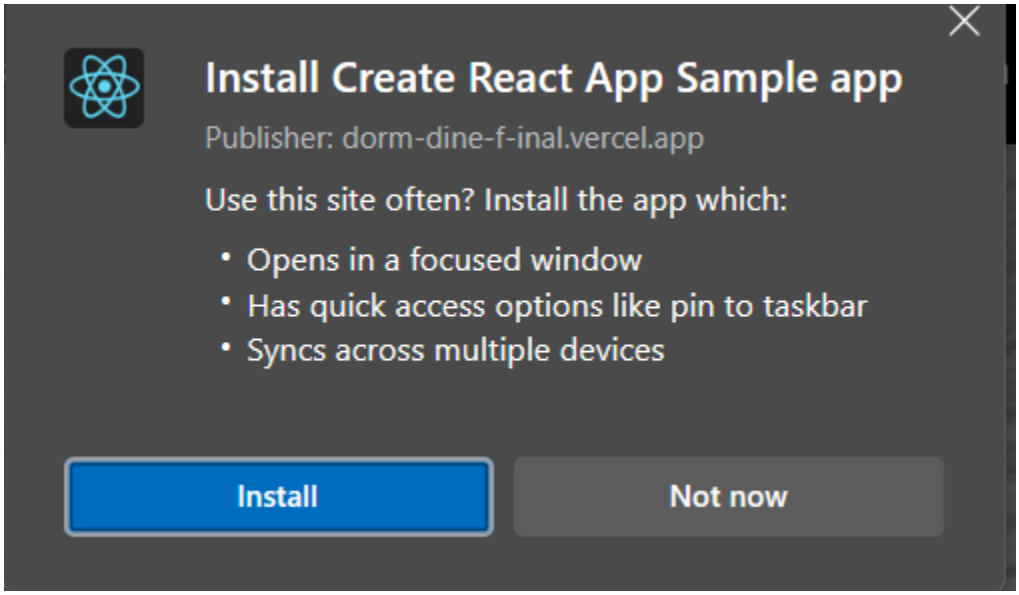
Open your hosted site on Microsoft Edge



Click on 3 dots
click on Apps
select install app option



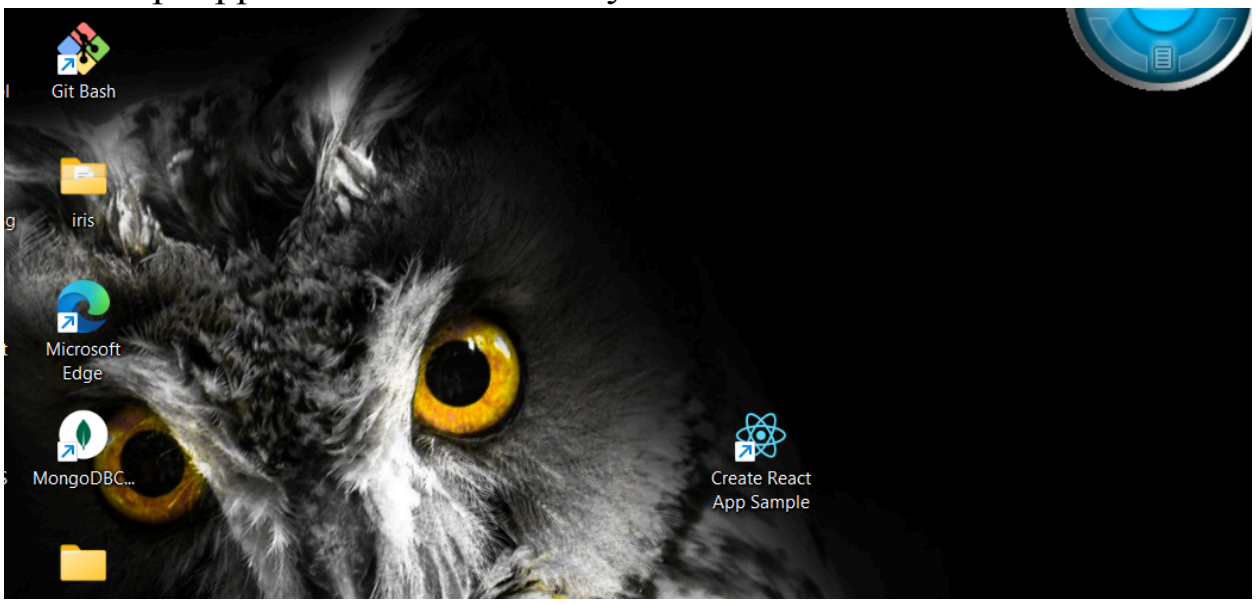
Click on Install



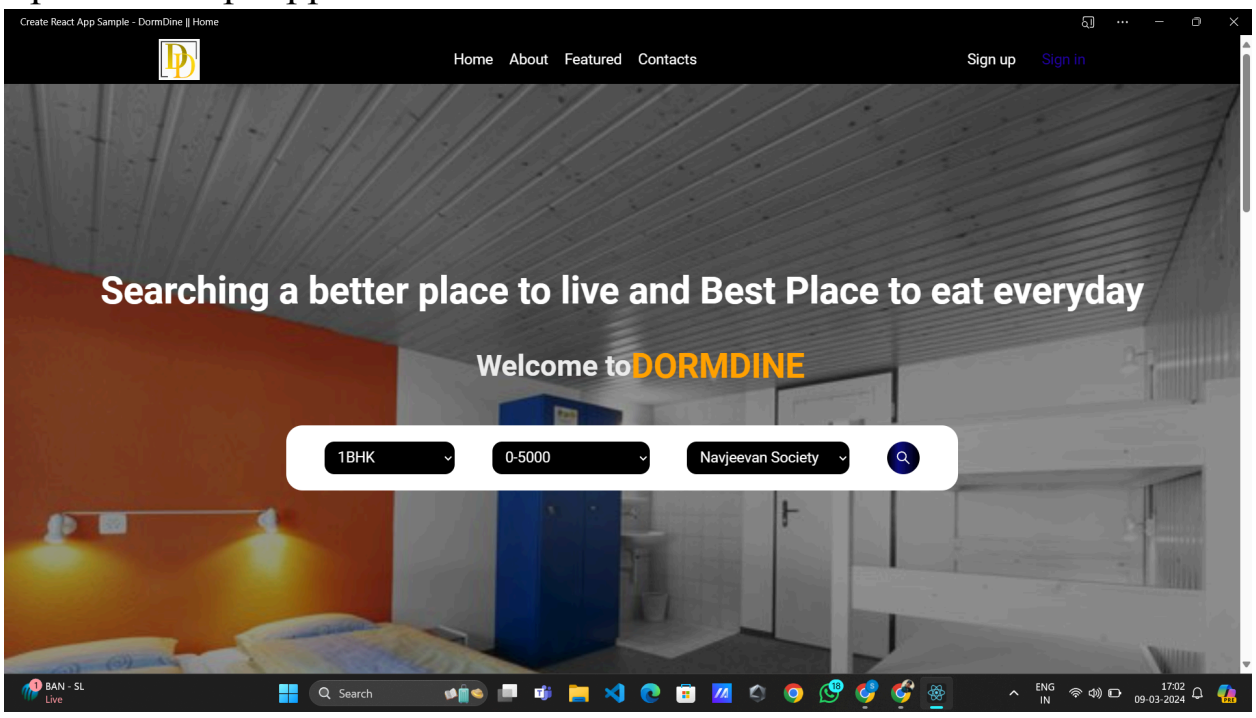
App icon will appear at the bottom

Output:

Desktop App Created Successfully.



Open Desktop App:



Conclusion:

In this experiment, we have successfully created a basic progressive web app of our web page and installed it in our desktop successfully.

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.	04
Name	Sarvadnya Rajendra Awaghad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15 M

Experiment - 8

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

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

The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.

What can we do with Service Workers?

You can dominate Network Traffic

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

You can Cache

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

You can manage Push Notifications

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

What can't we do with Service Workers?

You can't access the Window

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

You can't work it on 80 Port

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

Registration

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

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

```
console.log('Service worker registration failed, error:', error);
});
}
```

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

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

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

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

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

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

serving the service worker script.

main.js

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

Installation

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

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

service-worker.js

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

Activation

Once a service worker has successfully installed, it transitions into the activation stage. If there are any open pages controlled by the previous service worker, the

new service worker enters a waiting state. The new service worker only activates when there are no longer any pages loaded that are still using the old service worker. This ensures that only one version of the service worker is running at any given time.

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

Code:

index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs
&amp;amp; more</title>

  <link
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.mi
n.css" rel="stylesheet">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/
K68vbdEjh4u" crossorigin="anonymous">
  <link rel="stylesheet" href="css/style.css"/>
</head>

<body>
  <a id="banner" href="#"></a>
  <!-- Main body -->

  <header>
    <a id="nav-top"></a>
    <nav id="nav-main">
      <div class="nav-left">
        <div class="nav-shop">
          <a class="nav-a" href="#">
            Departments
            <i class="fa fa-caret-down" aria-hidden="true"></i>
          </a>
```

```
      </div>
     </div>
     <div class="nav-right">
      <a class="nav-a" href="#">
       <span>EN</span>
       <i class="fa fa-globe" aria-hidden="true"></i>
       <i class="fa fa-caret-down" aria-hidden="true"></i>
      </a>

      <a class="nav-a" href="#">
       <span>Hello. Sign in</span>
       Accounts &amp;amp; Lists
       <i class="fa fa-caret-down" aria-hidden="true"></i>
      </a>

      <a class="nav-a" href="#">
       Orders
      </a>

      <a class="nav-a" href="#">
       Try Prime
       <i class="fa fa-caret-down" aria-hidden="true"></i>
      </a>

      <a class="nav-a cart" href="#">
       <span>0</span>
       Cart
      </a>
     </div>
     <div class="nav-fill">
      <ul>
       <li><a href="#">Your Amazon.com</a></li>
       <li><a href="#">Today's Deals</a></li>
       <li><a href="#">Gift Cards &amp;amp; Registry</a></li>
       <li><a href="#">Sell</a></li>
       <li><a href="#">Help</a></li>
      </ul>
     </div>
    </nav>
  </header>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8
wGNIcPD7Txa" crossorigin="anonymous"></script>
  <script src="js/app.js"></script>
</body>
</html>
```

main.css

```
html, body {
   margin: 0;
   font-family: arial,sans-serif;
   min-width: 900px;
   line-height: 14px;
   font-size: 14px;
 }

 * {
  box-sizing: border-box;
 }

 a {
  color: #0066c0;
 }

 a:hover {
  color: #c45500;
 }

 #banner {
  background: #F6F6F6 url('../img/banner.jpg') no-repeat top left;
  height: 55px;
  background-size: 1920px;
  min-width: 1000px;
  display: block;
 }

 header {
  background-color: #232f3e;
  height: 99px;
 }
```

app.js

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

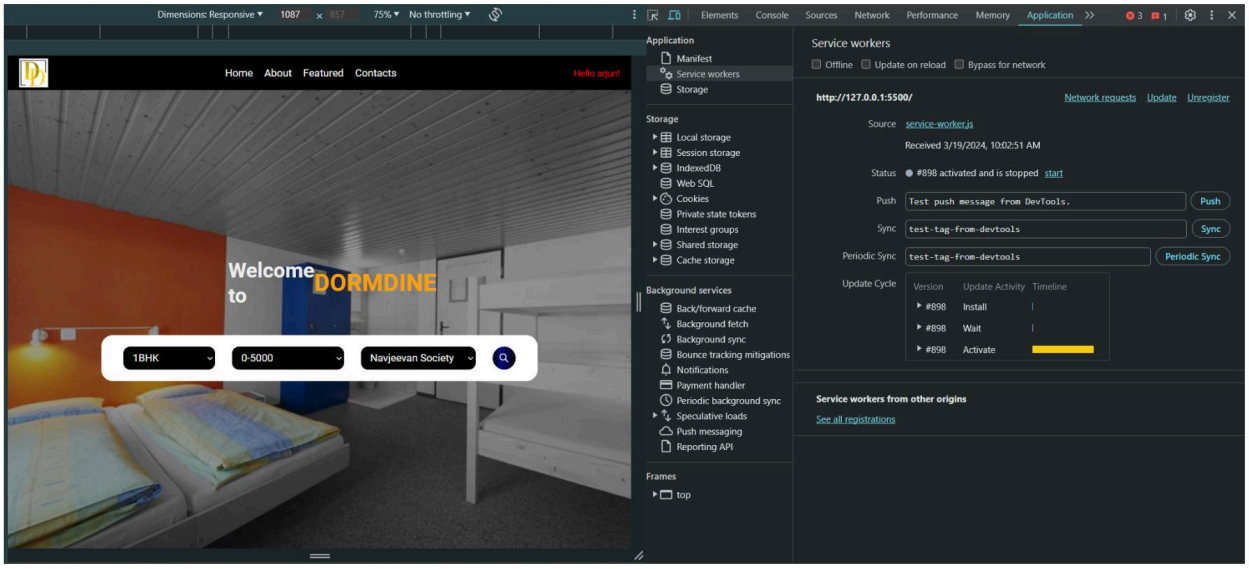
service-worker.js

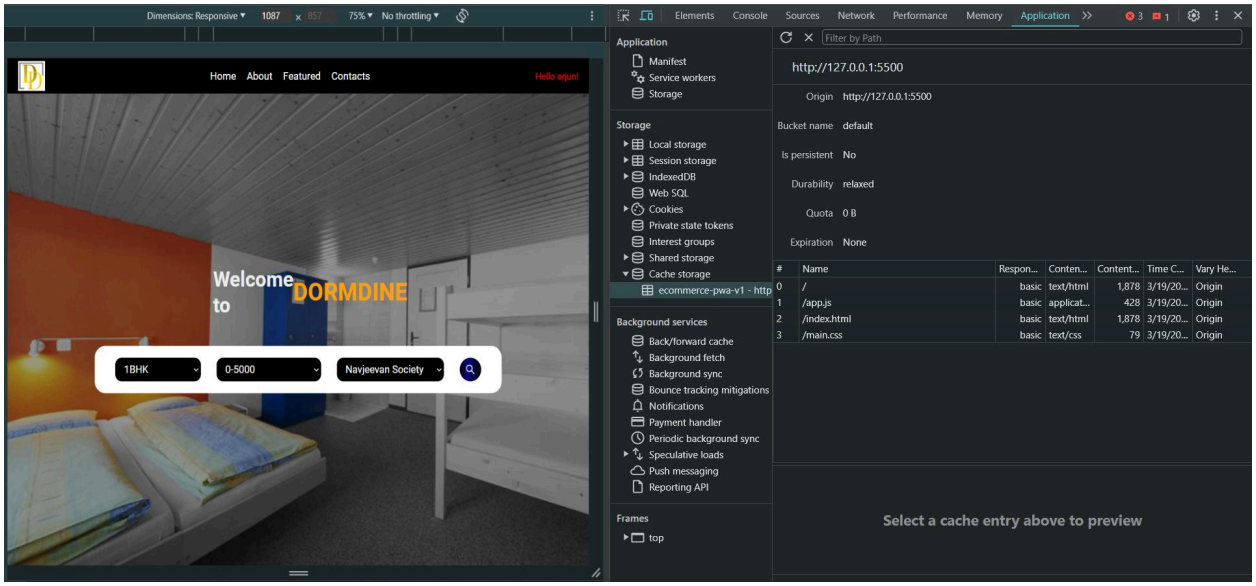
```
const cacheName = 'ecommerce-pwa-v1';
const assetsToCache = [
   '/',
   '/index.html',
   '/main.css',
   '/app.js'
]

self.addEventListener('install', event => {
   event.waitUntil(
      caches.open(cacheName)
         .then(cache => {
            return cache.addAll(assetsToCache);
         })
   );
});


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

Steps for Execution
Create a folder and put all 4 files main.css , service-worker.js, app.js, index.html
open visual studio
install extension Live server
open folder in visual studio open index.html
on bottom right corner click go Live
it will open html page in browser
go to developer tools

Project Title: Flutter : Tinder Clone / PWA : Dormdine **Roll No. 04**

Output:





Conclusion:

In this experiment, we have registered a service worker, and completed the install and activation process for a new service worker for the E-commerce PWA.

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.	04
Name	Sarvadnya Rajendra Awaghad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15 M

Experiment - 9

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

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

Things to note about Service Worker:

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

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

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

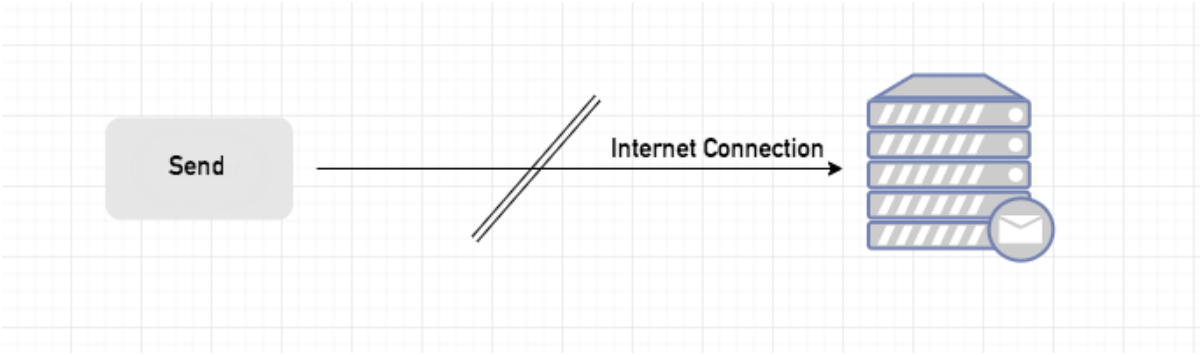
- **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
- **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed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

Sync Event

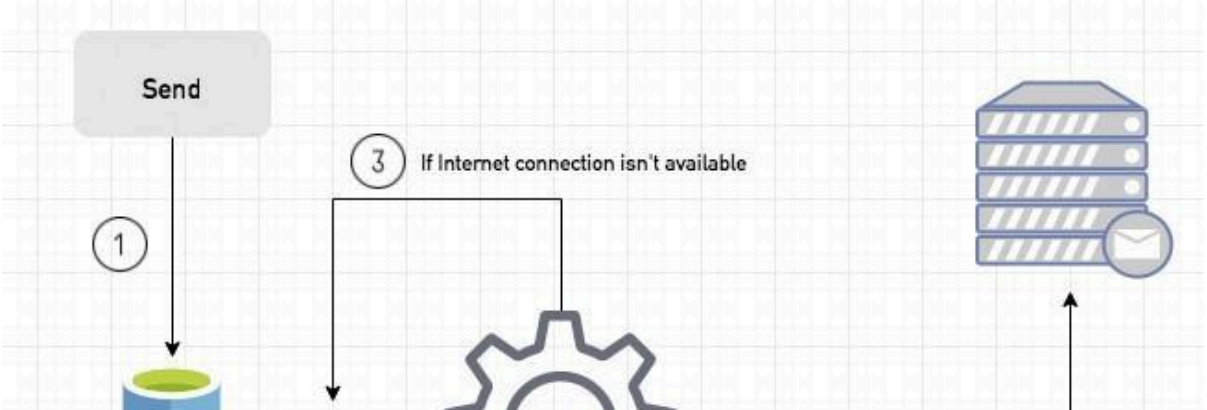
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are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

Here is a job for the Background Sync.

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



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



case.

1. When we click the “send” button, email content will be saved to IndexedDB.
2. Background Sync registration.
3. **If the Internet connection is available**, all email content will be read and sent to Mail Server.

   If the Internet connection is unavailable, the service worker waits until the connection is available even though the window is closed. When it is available, email content will be sent to Mail Server.

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

Push Event

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

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

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

Code:

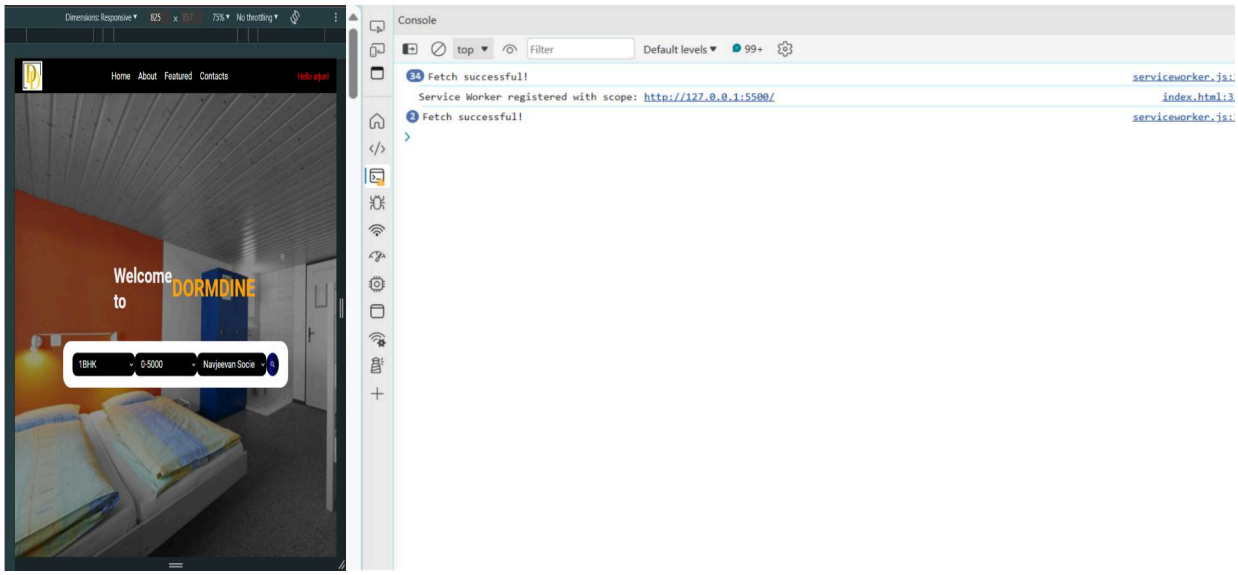
```
//serviceworker
self.addEventListener("install", function (event) {
  event.waitUntil(preLoad());
});
self.addEventListener("fetch", function (event) {
  event.respondWith(
    checkResponse(event.request).catch(function () {
      console.log("Fetch from cache successful!");
      return returnFromCache(event.request);
    })
  );
  console.log("Fetch successful!");
```

```
  event.waitUntil(addToCache(event.requesst));
});
self.addEventListener("sync", (event) => {
  if (event.tag === "syncMessage") {
    console.log("Sync successful!");
  }
});
self.addEventListener("push", function (event) {
  if (event && event.data) {
    try {
      var data = event.data.json();
      if (data && data.method === "pushMessage") {
        console.log("Push notification sent");
        self.registration.showNotification("Ecommerce website", {
          body: data.message,
        });
      }
    } catch (error) {
      console.error("Error parsing push data:", error);
    }
  }
});
var preLoad = function () {
  return caches.open("offline").then(function (cache) {
    // caching index and important routes
    return cache.addAll([
      "/",
      "/index.html",
      "/about.html",
      "/blog.html",
      "/contact.html",
      "/services.html",
      "/img/slider-img4.jpg",
      "/css/main.css",
    ]);
  });
};
var checkResponse = function (request) {
  return new Promise(function (fulfill, reject) {
    fetch(request)
      .then(function (response) {
        if (response.status !== 404) {
          fulfill(response);
        } else {
          reject(new Error("Response not found"));
        }
```

```
    })
    .catch(function (error) {
      reject(error);
    });
  });
};
var returnFromCache = function (request) {
  return caches.open("offline").then(function (cache) {
    return cache.match(request).then(function (matching) {
      if (!matching || matching.status == 404) {
        return cache.match("offline.html");
      } else {
        return matching;
      }
    });
  });
};
var addToCache = function (request) {
  return caches.open("offline").then(function (cache) {
    return fetch(request).then(function (response) {
      return cache.put(request, response.clone()).then(function () {
        return response;
      });
    });
  });
};
```

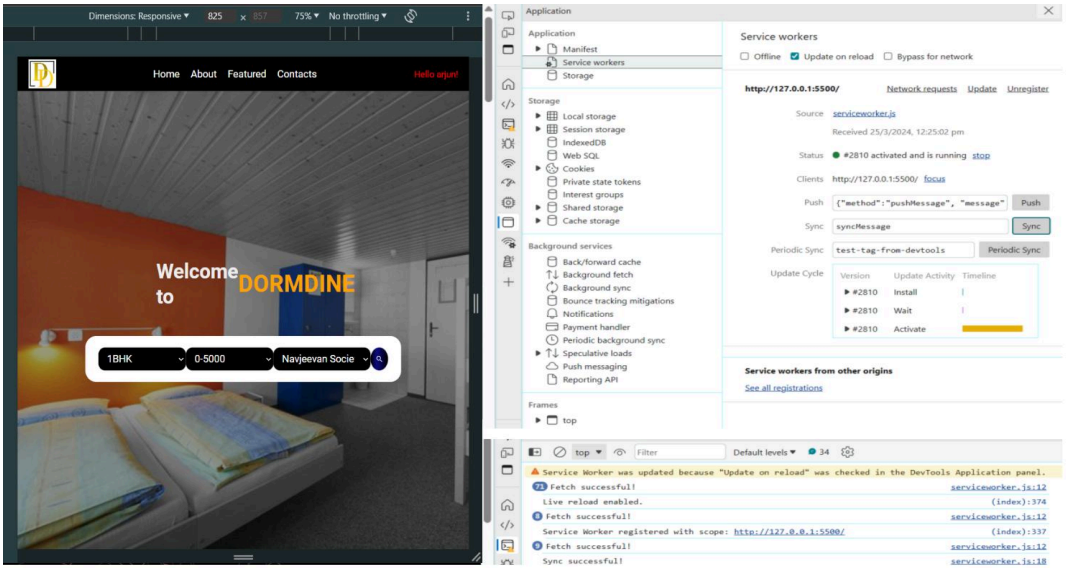
Output:

Fetch:



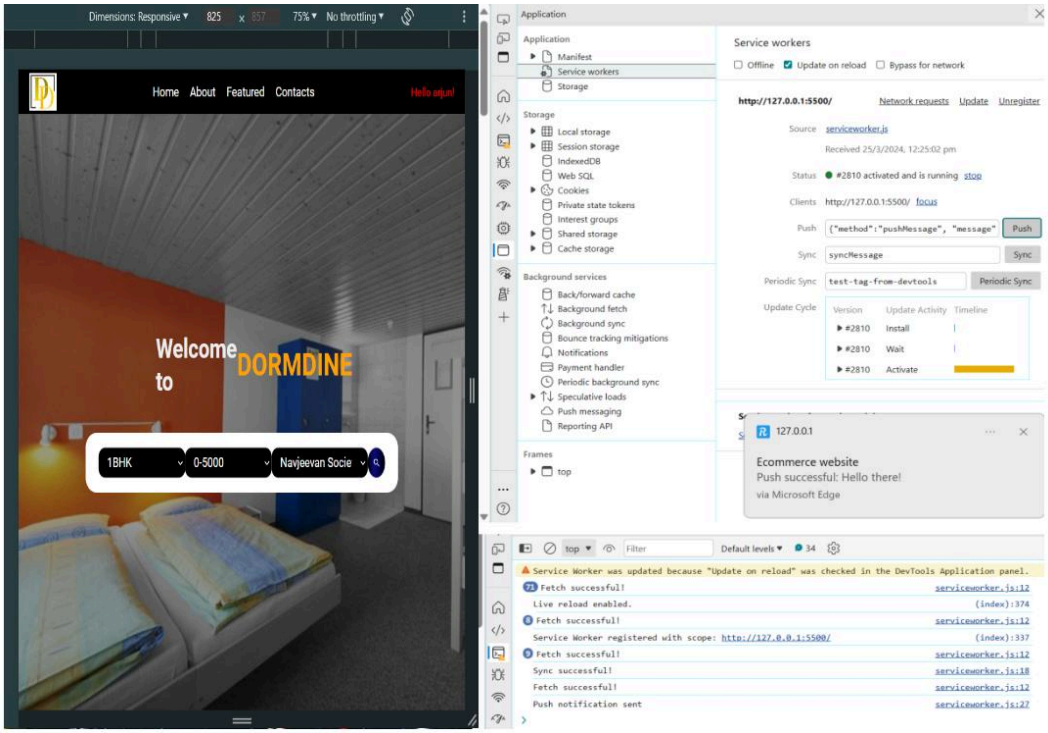
SYNC:

Project Title: Flutter : Tinder Clone / PWA : Dormdine **Roll No. 04**



Project Title: Flutter : Tinder Clone / PWA : Dormdine Roll No. 04

PUSH:



Conclusion :

In this experiment, we have successfully implemented service worker events like fetch, sync and push for E-commerce PWA and found out output for above implementation.

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	04
Name	Sarvadnya Rajendra Awaghad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15M

Experiment - 10

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

Theory:

GitHub Pages

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

GitHub Pages provides the following key features:

1. Blogging with Jekyll
2. Custom URL
3. Automatic Page Generator

Reasons for favoring this over Firebase:

1. Free to use
2. Right out of github
3. Quick to set up

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

Pros

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

Cons

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

Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

Pros

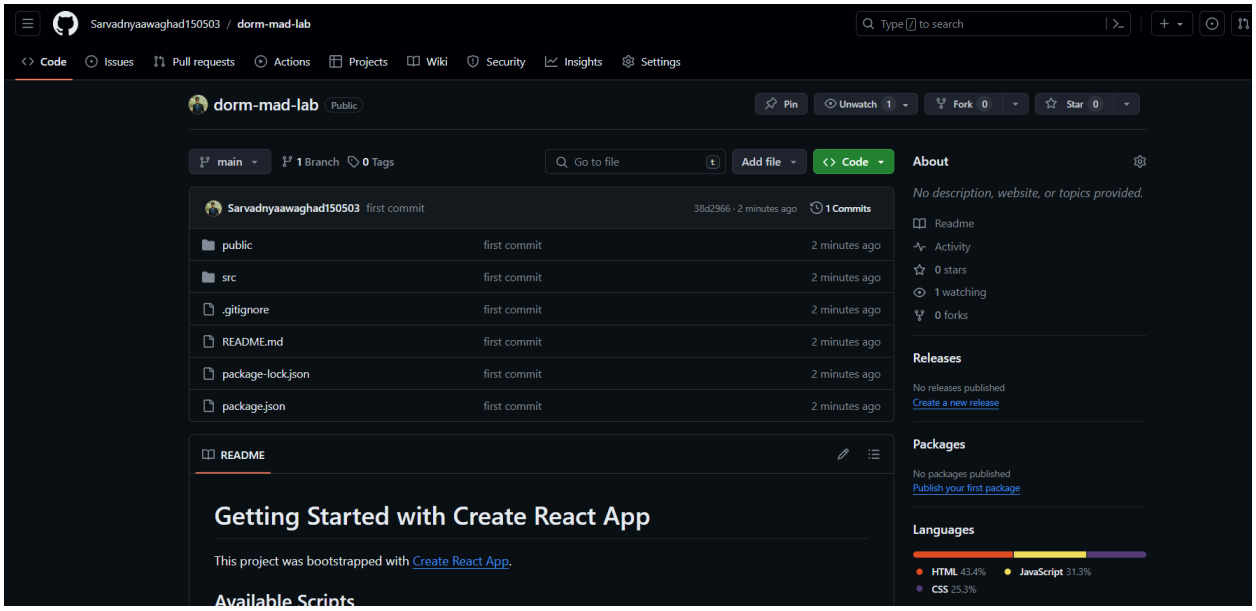
1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5.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

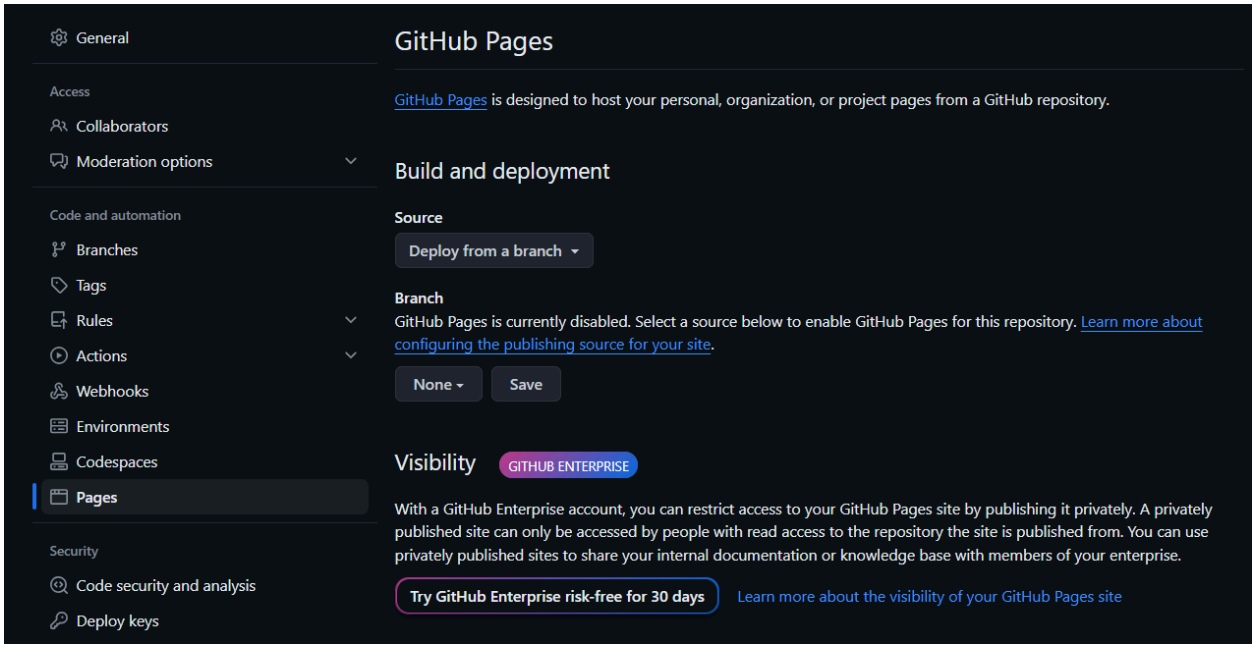
Cons

1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.

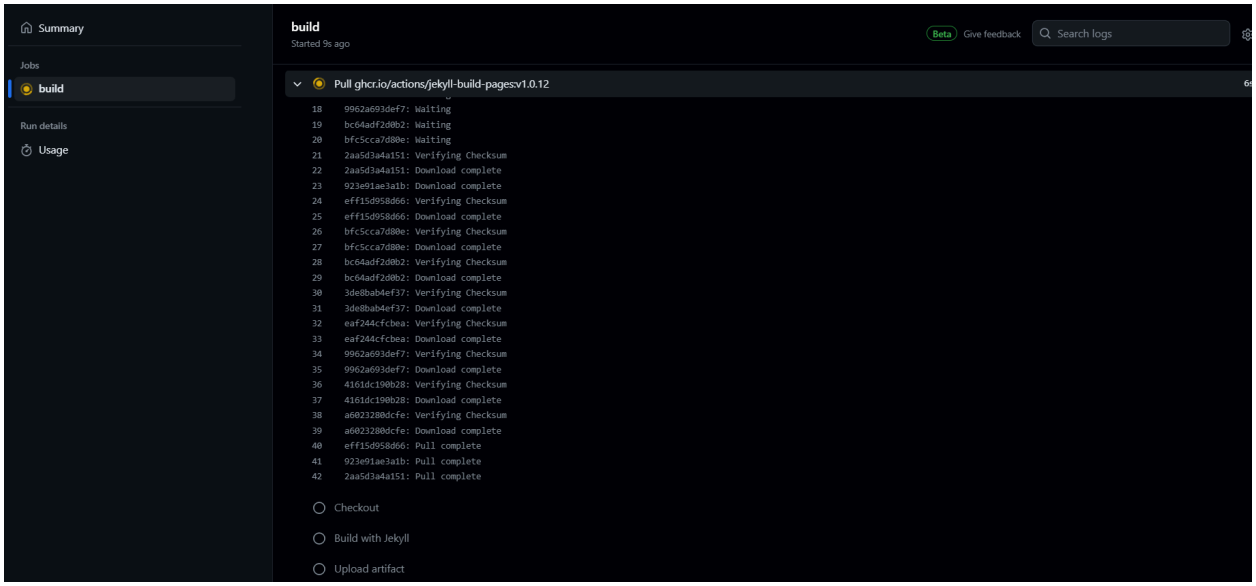
Project Title: Flutter : Tinder Clone / PWA : Dormdine **Roll No. 04**

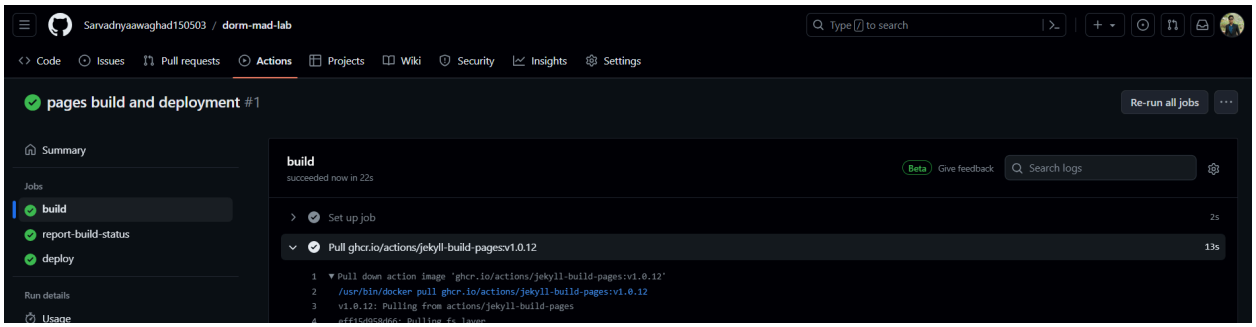
Implementation:





Project Title: Flutter : Tinder Clone / PWA : Dormdine **Roll No. 04**





Link to our GitHub repository:
https://github.com/Sarvadnyaawaghad150503/dormhost

Hosted Link: https://sarvadnyaawaghad150503.github.io/dormhost/

Github Screenshot:

Conclusion:

In this experiment, we have registered a service worker, and completed the install and activation process for a new service worker for the E-commerce PWA.

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	04
Name	Sarvadnya Rajendra Awaghad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15 M

Experiment - 11

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

Theory:

Google Lighthouse

Lighthouse is an open-source, automated tool for improving the quality of web pages. You can run it against any web page, public or requiring authentication. It has audits for performance, accessibility, progressive web apps, SEO and more.

You can run Lighthouse in Chrome DevTools, from the command line, or as a Node module. You give

Lighthouse a URL to audit, it runs a series of audits against the page, and then it generates a report on how well the page did. From there, use the failing audits as indicators on how to improve the page. Each audit has a reference doc explaining why the audit is important, as well as how to fix it.

Features of Lighthouse

Google Lighthouse gives a breakdown of your site into the accompanying metrics. Here is a brief explanationof each of the aforementioned metrics:

1. Performance

Performance is generally viewed as the most valuable metric given by the Google Lighthouse tool. Like the PageSpeed Insights, the Performance area of the Lighthouse report contains a few helpful metrics you can use to advance your site to climb Google's rankings. The Performance segment of the Lighthouse report joins the Opportunities, Field Data, Lab Data, and Diagnostics metrics of the PageSpeed Insights tool.

A great example is the opportunities metric as it flags three types of render-blocking URL’s namely stylesheets, scripts, and HTML imports. This merged perspective on performance metrics gives an exact and valuable analysis of your site's performance and any progressions you should make to expand your site's exhibition.

2. Accessibility

The first of the new regions of Google Lighthouse is the Accessibility metric. Basically what this metric does is feature potential chances to

improve the availability and client experience of your mobile app or website.
Following the accessibility improvement report will guarantee that your clients can without much of a stretch explore and utilize your site. Just as guaranteeing that you have the most obvious opportunity with regards to positioning better on web search engines.

3. Best Practices
Another segment new to Google's analysis tools is the Best Practices metric. This region of the Lighthouse report doesn't carefully give execution related data. However, it will give you recommendations which can improve both your exhibition and client experience, particularly for mobile sites.

4. SEO
The latest and most dynamic of the highlights in Google's Lighthouse instrument is the SEO metric.
PageSpeed Insights doesn't offer this tool. This is why most web designers and SEO specialists prefer to utilize Google Lighthouse to analyze a website. The SEO metric gives fundamental tools to examine your page's streamlining for search engine results rankings. While there are numerous more factors which Lighthouse doesn't consider or quantify, the most essential focuses are secured.

5. Progressive Web Applications
The Progressive Web App area is another of Google's most up to date execution measurements incorporated into its Lighthouse tool. While the meaning of a Progressive Web App (PWA) hasn't been especially clear, Google's documentation expresses that there are a few key variables which make a site a PWA. A great feature of this metric is registering service workers which allow you to enable push notifications on your web app

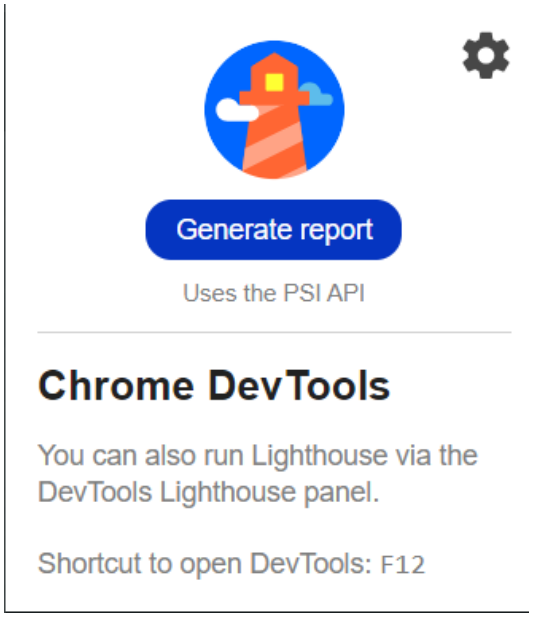
IMPLEMENTATION:

Manifest.json :

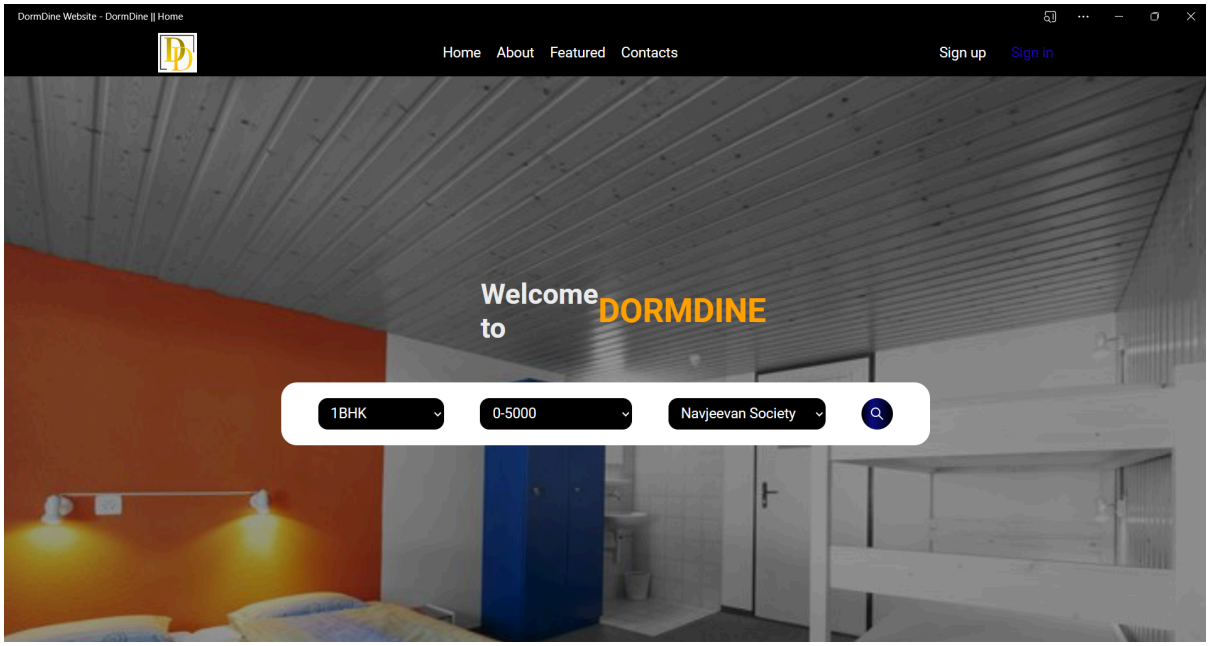
```
{
  "short_name": "DormDine",
  "name": "DormDine Website",
  "icons": [
    {
```

```
    "src": "logo192.png",
    "type": "image/png",
    "sizes": "192x192",
    "purpose": "maskable"
   },
   {
    "src": "logo512.png",
    "type": "image/png",
    "sizes": "512x512"
   }
 ],
 "start_url": ".",
 "display": "standalone",
 "theme_color": "#000000",
 "background_color": "#ffffff"
}
```

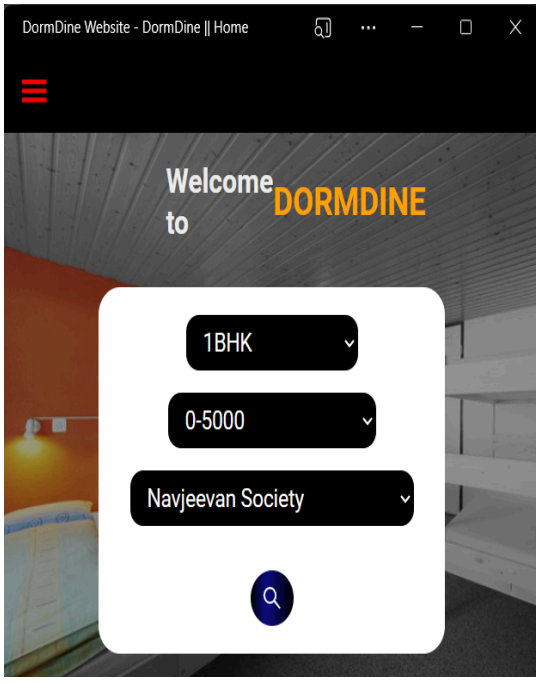
FOR DESKTOP DEVICE



Project Title: Flutter : Tinder Clone / PWA : Dormdine **Roll No. 04**

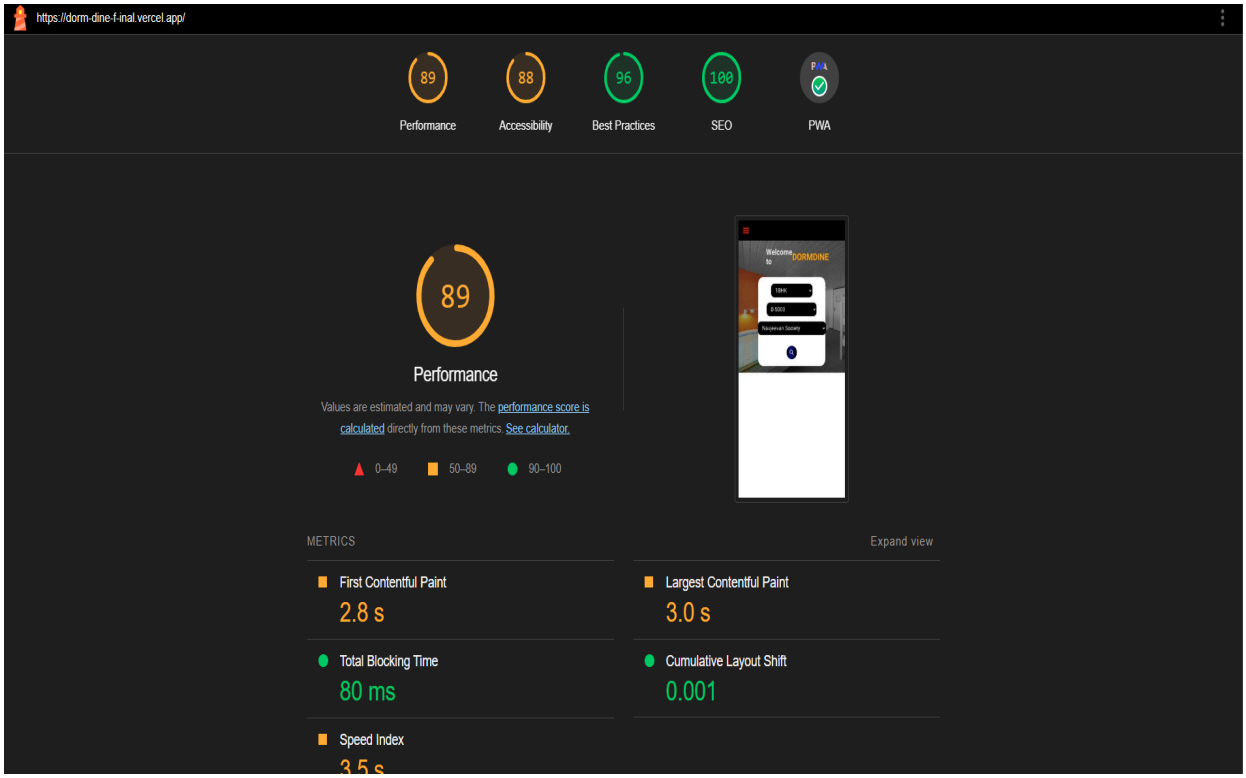


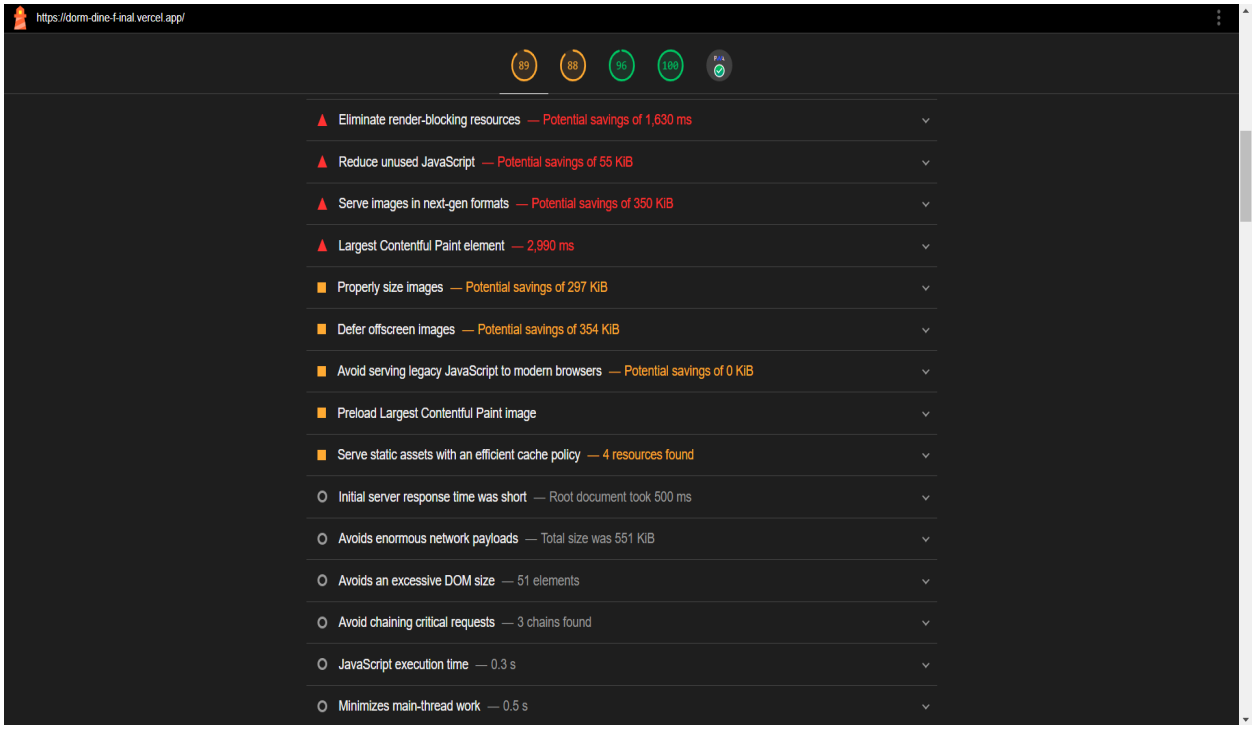
FOR MOBILE DEVICES



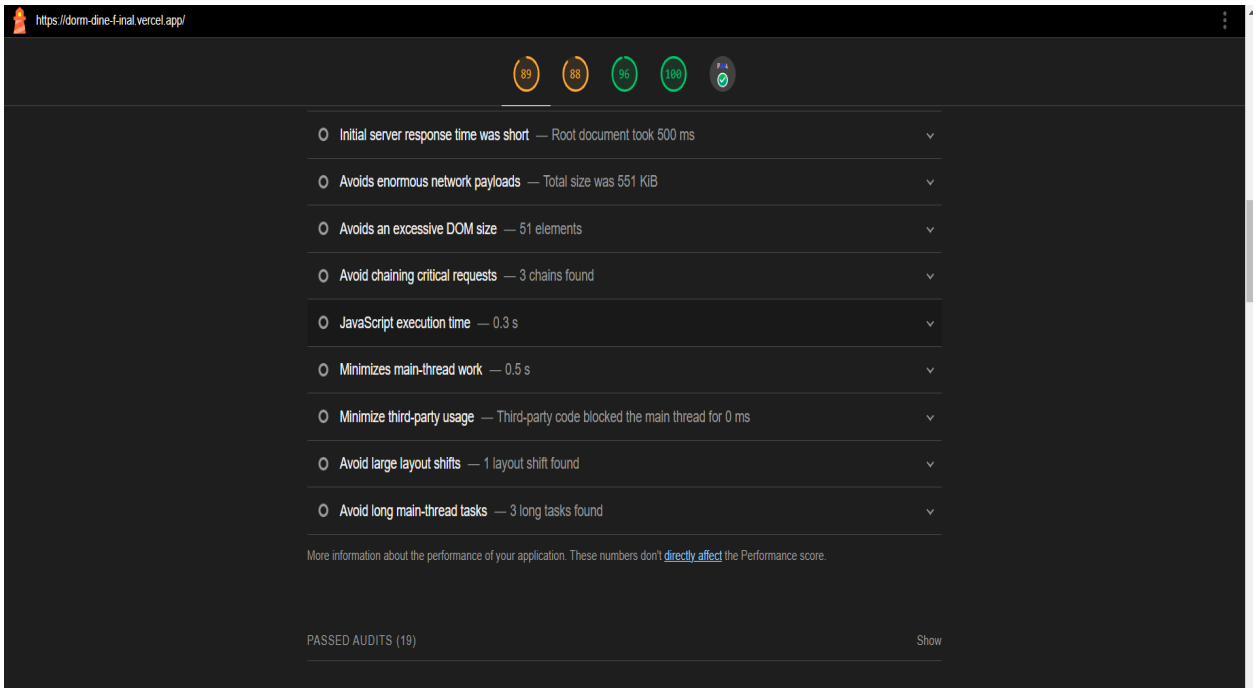
REPORT GENERATED:

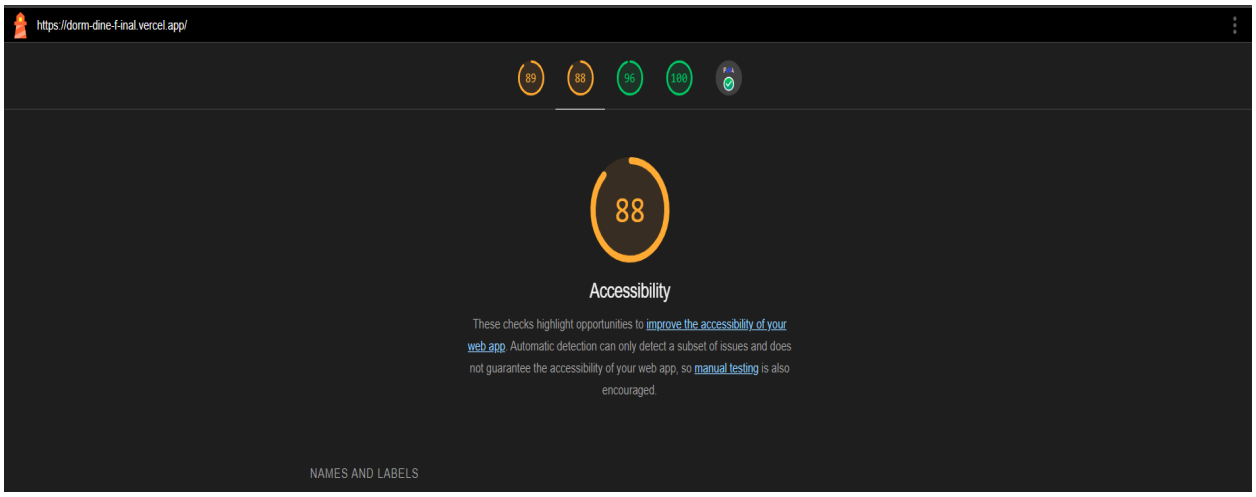
Project Title: Flutter : Tinder Clone / PWA : Dormdine **Roll No. 04**



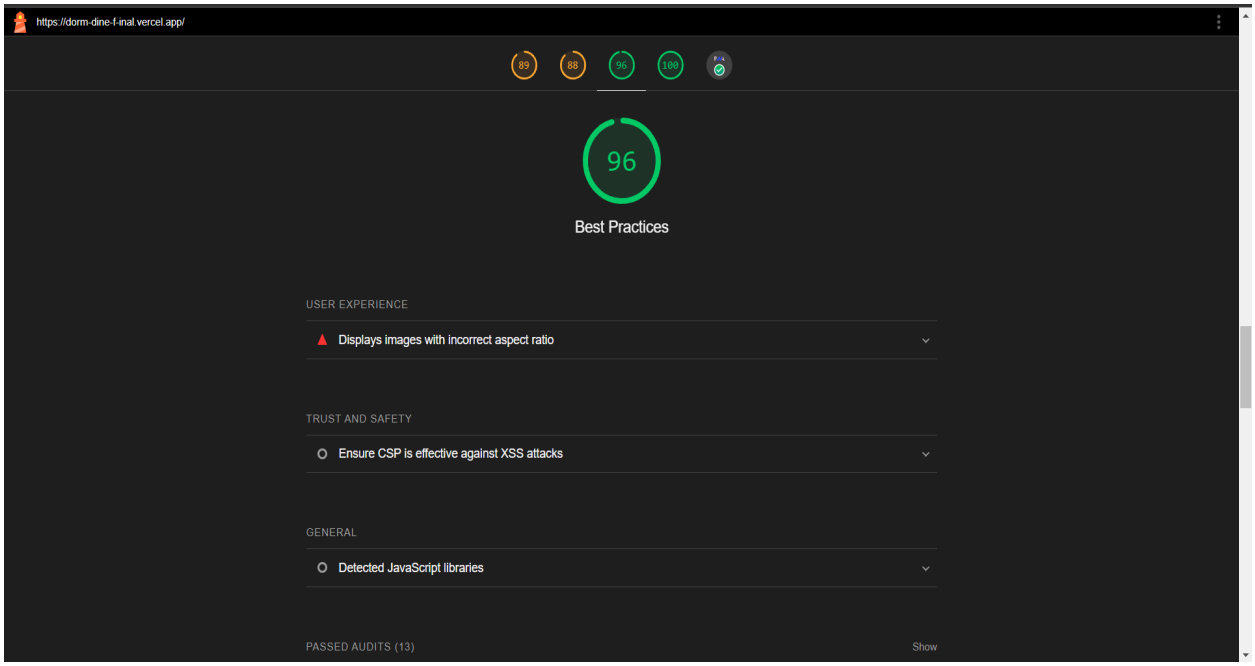


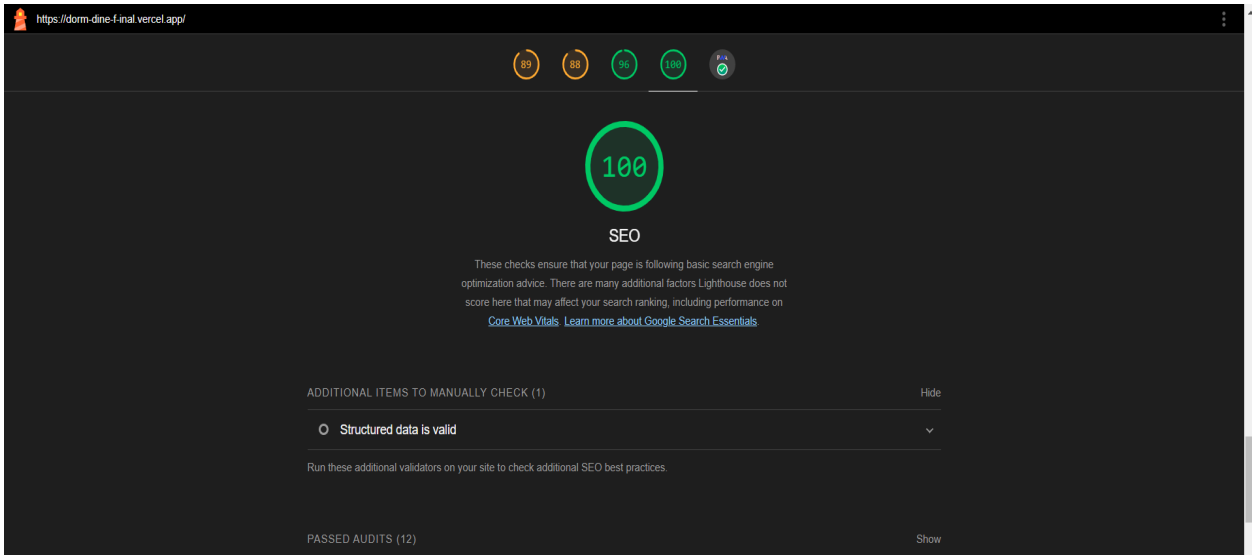
Project Title: Flutter : Tinder Clone / PWA : Dormdine **Roll No. 04**

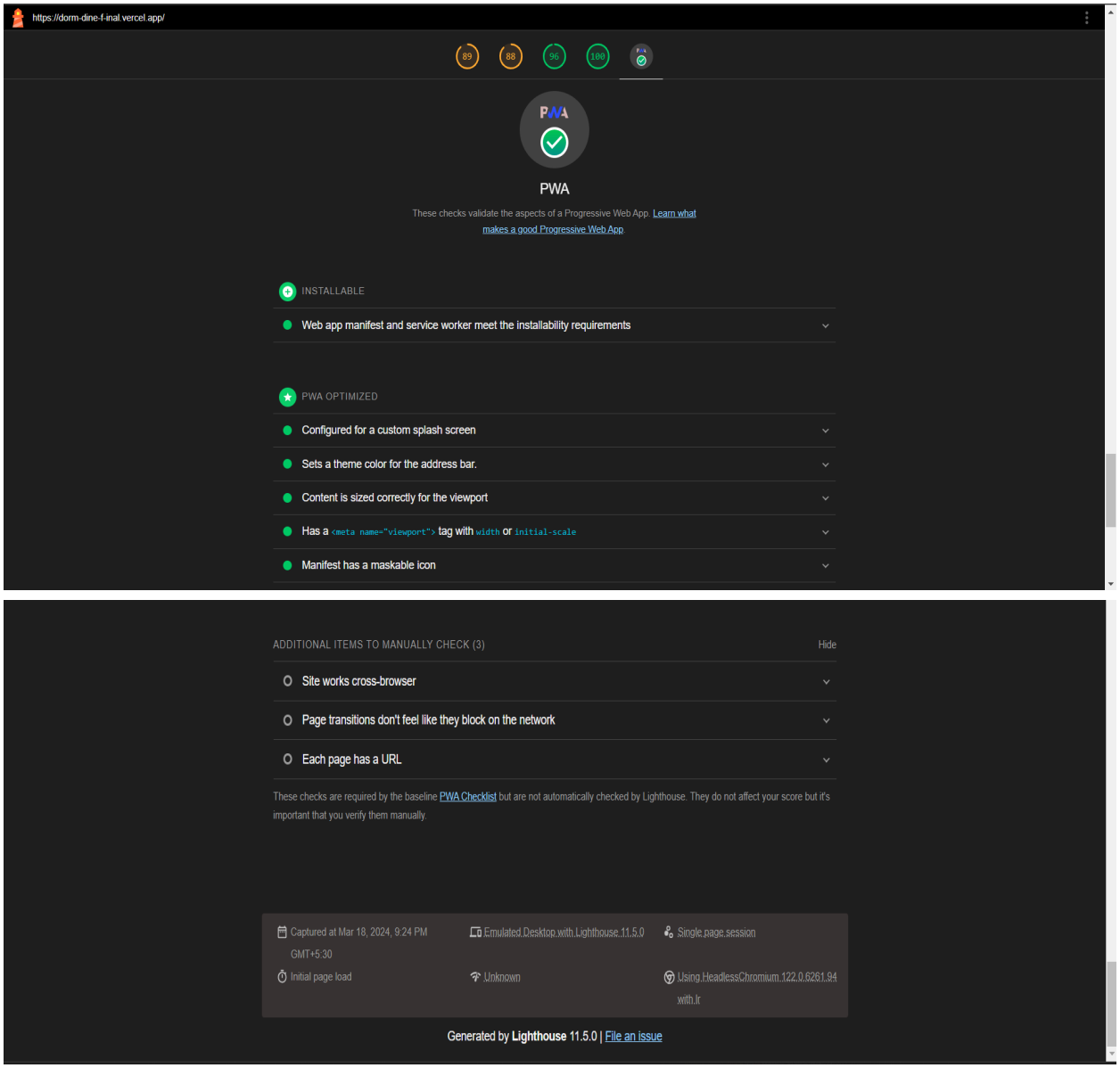




Project Title: Flutter : Tinder Clone / PWA : Dormdine **Roll No. 04**







Conclusion:

In this experiment, we have successfully used Google Lighthouse PWA Analysis Tool for testing the PWA functioning.

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.	04
Name	Sarvadnya Rajendra Awaghad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	5 M

Sarvodnya Awaghad
D15A 04
Page No.
Date :

Assignment 1 flutter

Q i) Flutter Overview:- Explain key features and advantages of using flutter for mobile app development. Discuss how flutter framework differ from traditional approach and why it gained popularity

→ Key features:-

ⓐ Single codebase for multiple platforms:-
Flutter allows developer to write code and deploy it on both ios and android.

ⓑ Hot Reload:-
This enables developers to instantly see results of code changes they make.

ⓒ Express UI:-
Developers have flexibility to create expressive and flexible UI.

ⓓ Integration with other tool.
Flutter easily integrate with other popular development tool and framework.

2. Advantages:-

ⓐ faster Development
Uses single codebase for multiple platform

ⓑ Consistent UI
Widget provide a consistent look

and feel across different platform

© Cost Efficiency.
Developing and maintaining single codebase for both ios and android reducer development cost.

3) Differ from Traditional approach
@ Traditional approach uses a hierarchical structure for UI component, where as Flutter use a widget based approach
ⓑ Hot Reload allows to see changes made instantly

Flutter popularity is driven by increased productivity, a growing community, cross platform and adoption by companies

Q 2) Widget Tree and Composition: Describe the concept of widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide eg of commonly used widget & their role in widget tree.

→

Widget Tree:-

The widget Tree is a hierarchical structure of widget that defines the user interface

Every visual element, from simple component to complex layout, is represented by widget.

Widget are categorized in 2 types

ⓐ Stateless Widget

It is immutable and cannot change over time

ⓑ Stateful Widget

Widget that can change state over time.

Widget Composition

- Widget composition in Flutter involves combining multiple simple widget to create more complex and compound widget.

Teacher's Sign.: _____

Commonly used widget :-

(a) Container :-
A box model for padding, margin and decoration

(b) Column and Row
· layout widget for arranging children vertically or horizontally

(c) Stack
Overlapping widget allowing them to be layered on top of each other.

(d) List View :-
A scrollable grid of widget.

(e) Grid View :-
A scrollable grid of widget.

(f) Text field :-
An input field for user

Q3] State Management in Flutter:
Discuss importance of state management in Flutter application. Compare and contrast different state management approaches available in Flutter, such as setState, Provider scenario wher each approach suitable.

→ State management is crucial in Flutter application because it involves managing the data that can chang over time

	setState	Provider	Riverpod.
①	Built in Flutter method	External package named ('provider')	External package ('riverpod')
	local state within widget	Global state within widget tree	Global state with additional feature
	limited scalability for large app	Suitable for medium size app	Designed for large and complex app
	may lead to code redudancy	Balances simplicity and readability	Emphorizes readiability and clean syntax.
	Testing is challenging	Good testability support	Enhanced testing experience.

Scknario where each is applicable.

① Set State
for small to moderately complex application when managing local state within a widget.
eg:- Simple form, UI Specific State.

② Provider
For medium to large sized application when a centralized state is needed accessible by multiple widget
eg:- managing user authentication, theme change or appwide configuration.

③ Riverpod :-
For large and complex application when testability and maintainability are top priorities
eg:- Complex application with multiple feature dynamic UIs

Q4) Firebase Integration in flutter. Explain the process of integrating Firebase with Flutter application. Discuss benefit of using Firebase as a backend solution. Highlight Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.

→

Integration :-

1. Go to Firebase console and create new project
2. Add Firebase SDK by including dependencies in pubspec.yaml

```
dependencies:
    firebase_core: ^ version
    firebase_auth: ^ version
    cloud_firestore: ^ version
```

3. Run flutter pub get
4. Initialize firebase by calling 'firebase.initialiseApp()

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

void main () async {
  Widget FlutterBinding.ensureInitialized();
  await Firebase.initializeApp()
  run App (myApp());
  }
```

Teacher's Sign.:

Benefits of using Firebase as Backend:

① Realtime Database
: firebase offer a realtime NoSQL database.

② Authentication
Provides a secure and easy to implement solution for authentication

③ Cloud firestore:
Firebase's Cloud firestore provide a secure and easy to implement scalable NoSQL db that allows you to store and sync data

Data Synchronization :-

① Realtime Database
when data changes on one client, it triggers event that automatically update data on other client.

② Cloud firestore
It notifies client when data changes, allowing for seamless realtime updates.

③ Authentication:
If user sign in or out on one device, the authentication state is automatically reflected on other devices

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.	04
Name	Sarvadnya Rajendra Awaghad
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:	4 M