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 Shashwat Tripathi of **D15A/D15B** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Project Title: **Roll No.**

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

Year/Sem/Class : D15A/D15B **A.Y.:** 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: **Roll No.**

Lab Objectives:

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

Lab Outcomes:

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

Project Title: **Roll No.**

Index

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

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	64
Name	Shashwat Tripathi
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

Experiment 1

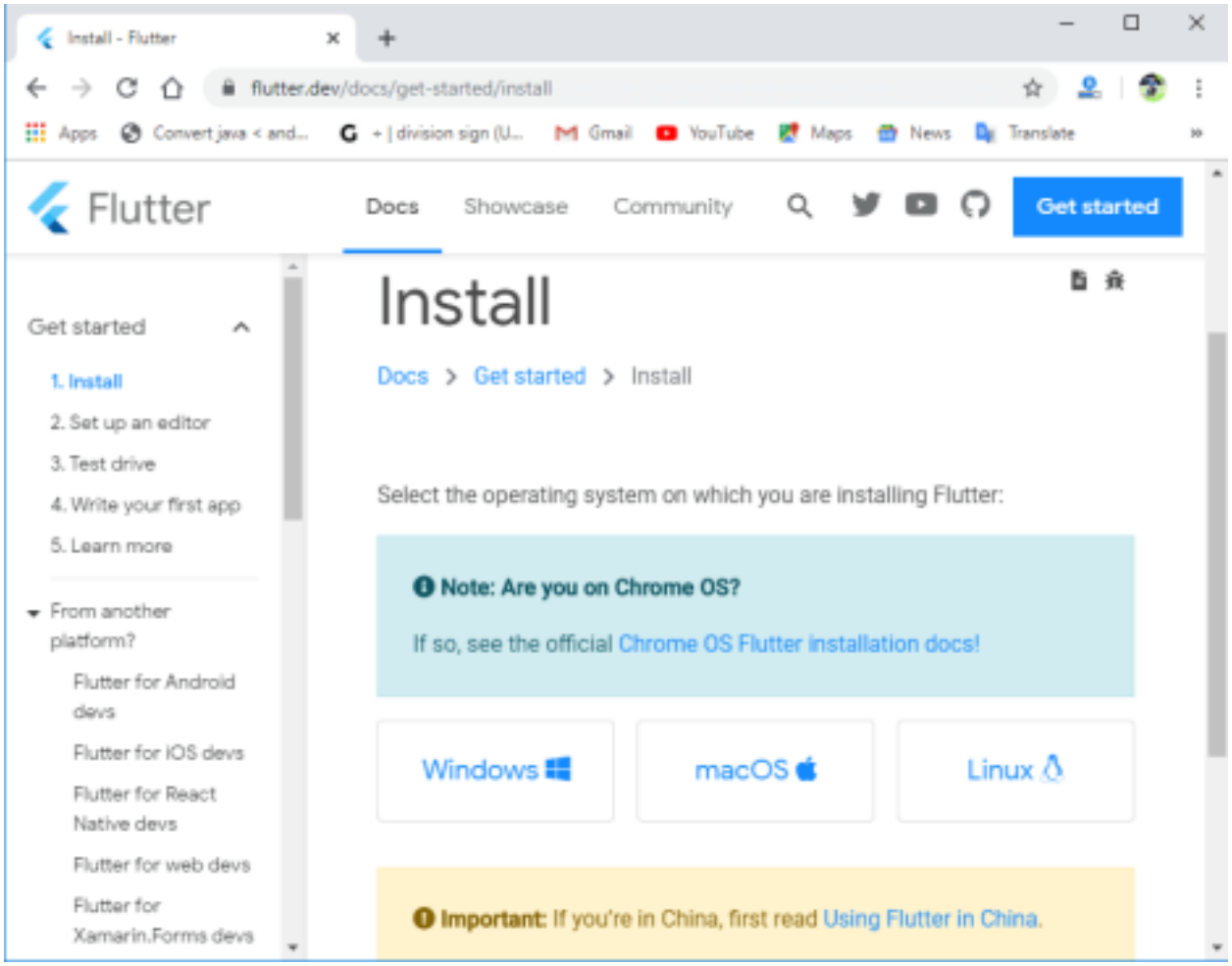
Shashwat Tripathi
D15A Batch C
Roll No: 64

AIM : To install flutter and Android Studio on our machine.

THEORY :

Install the Flutter SDK

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



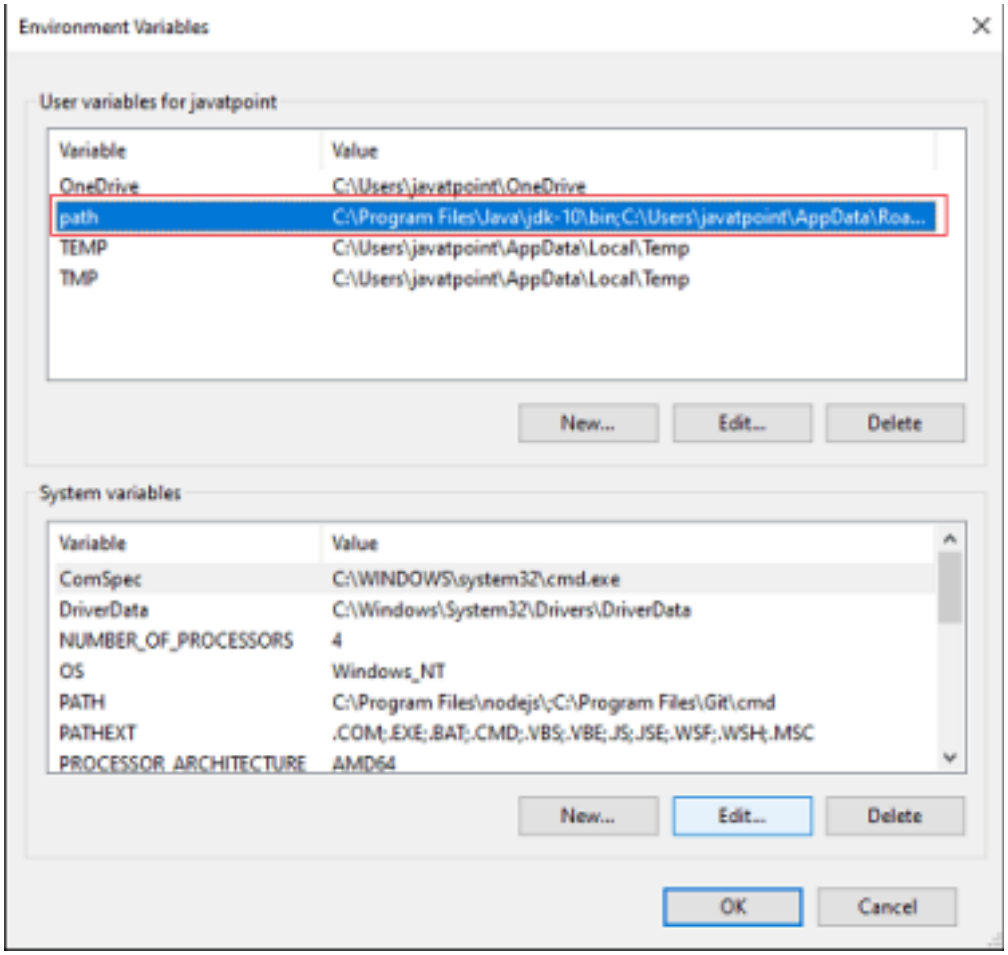
Step 2: Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will find the download link for SDK.

Step 3: When your download is complete, extract the **zip** file and place it in the desired

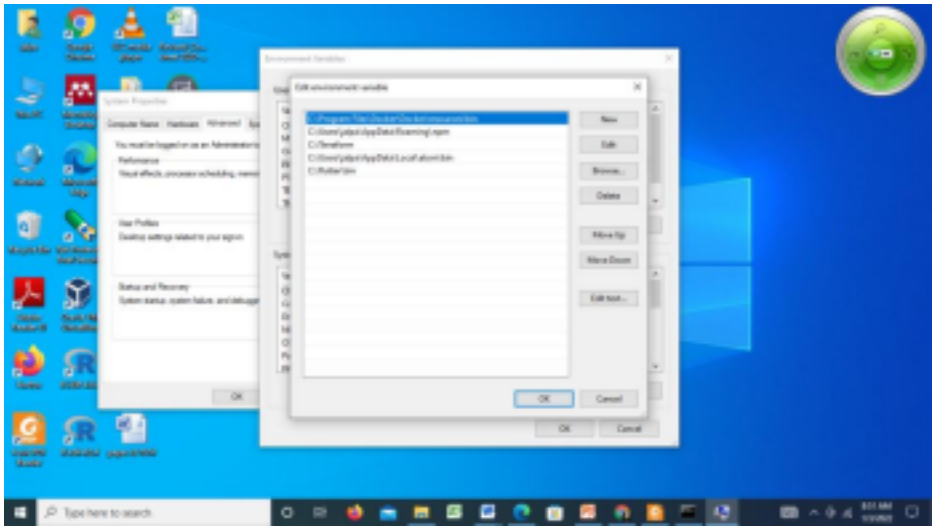
installation folder or location, for example, C: /Flutter.

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

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

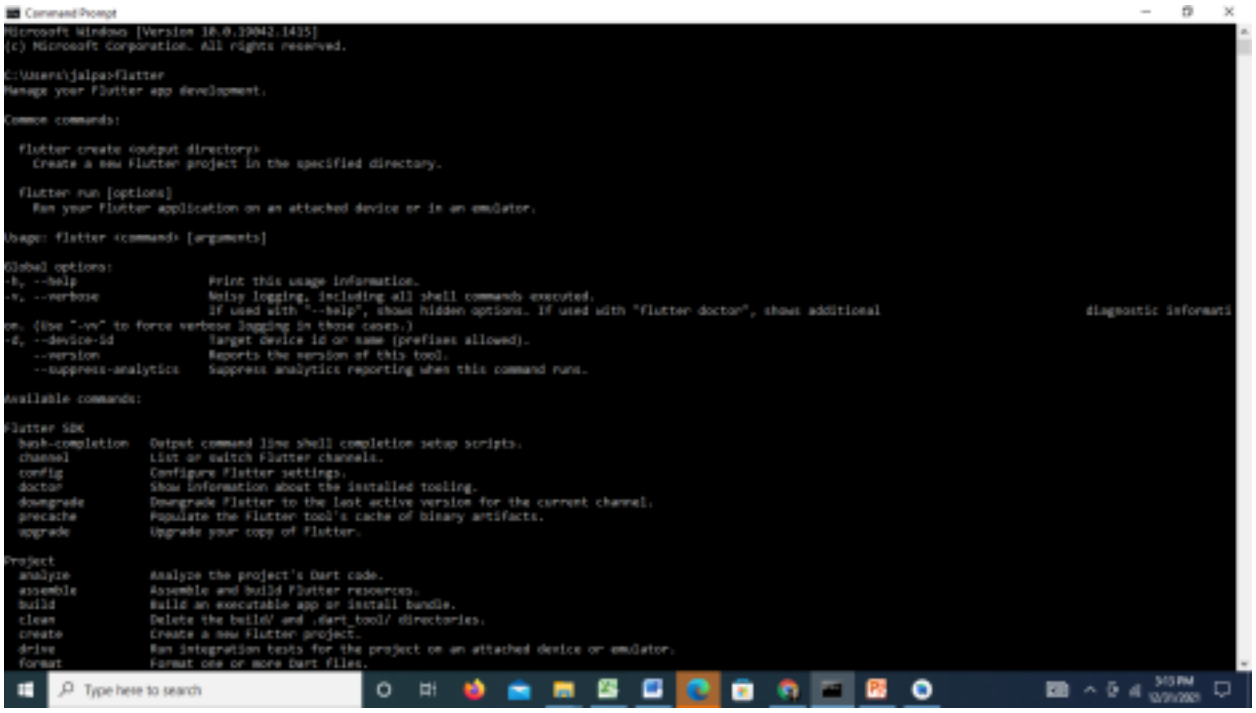


Step 4.2: Now, select path -> click on edit. The following screen appears

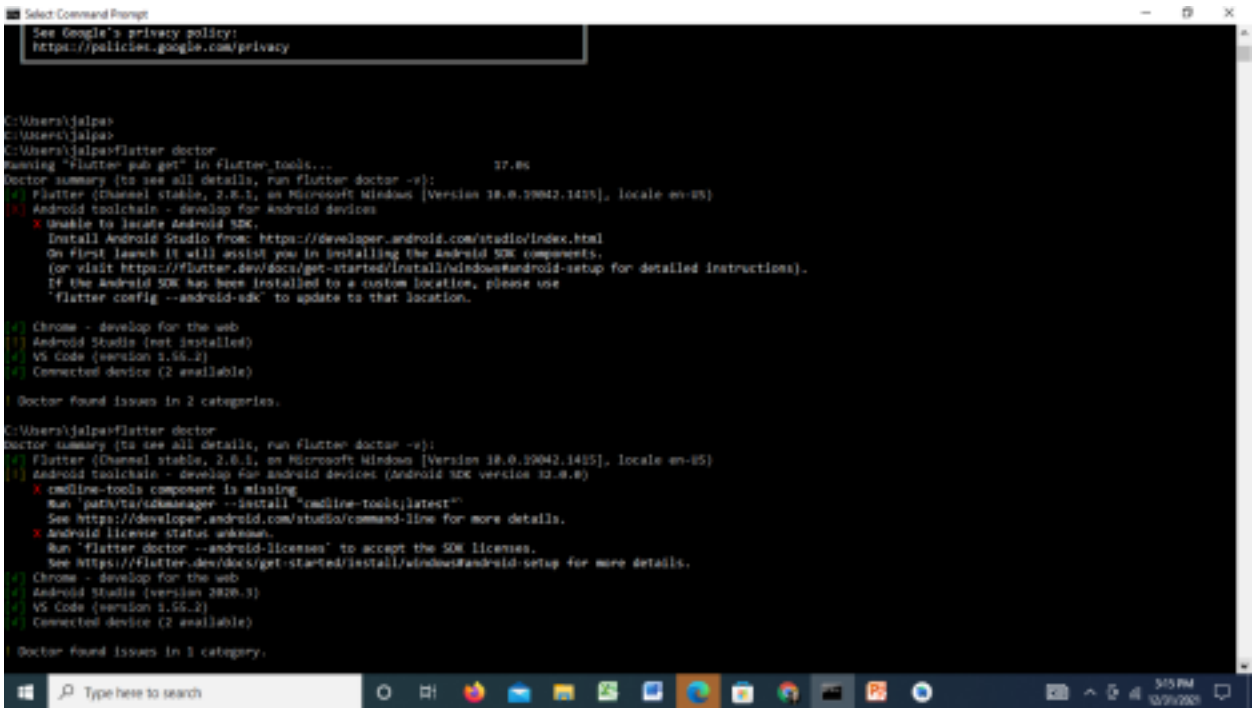


Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value - > ok -> ok -> ok.

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



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



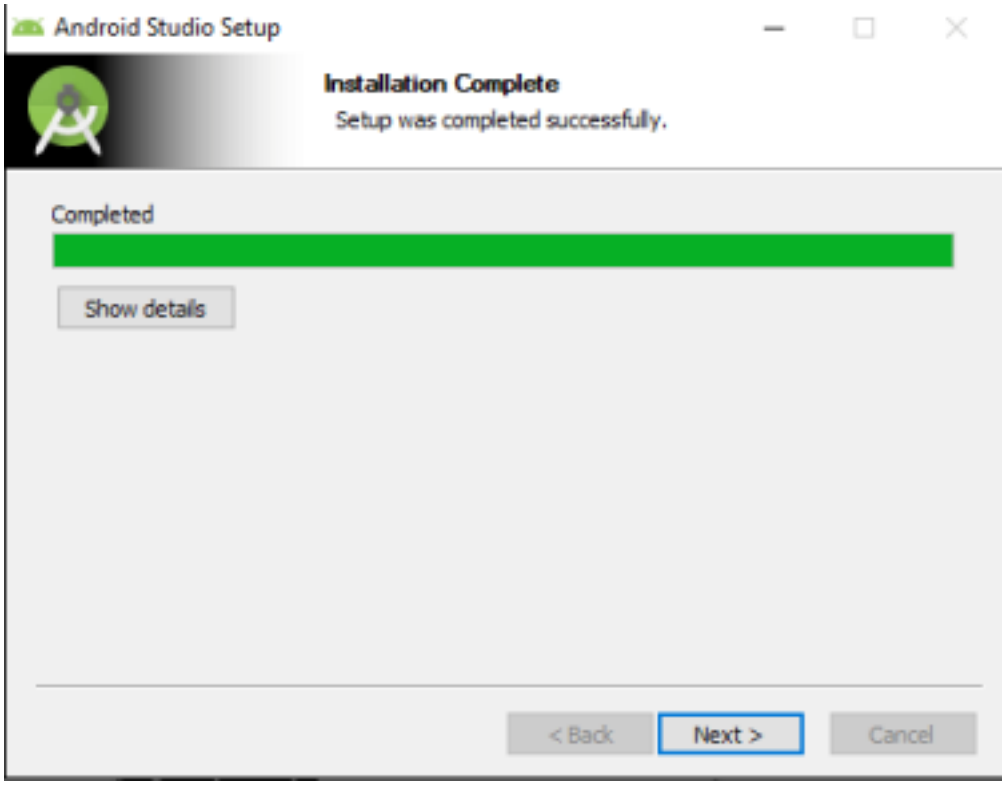
Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

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

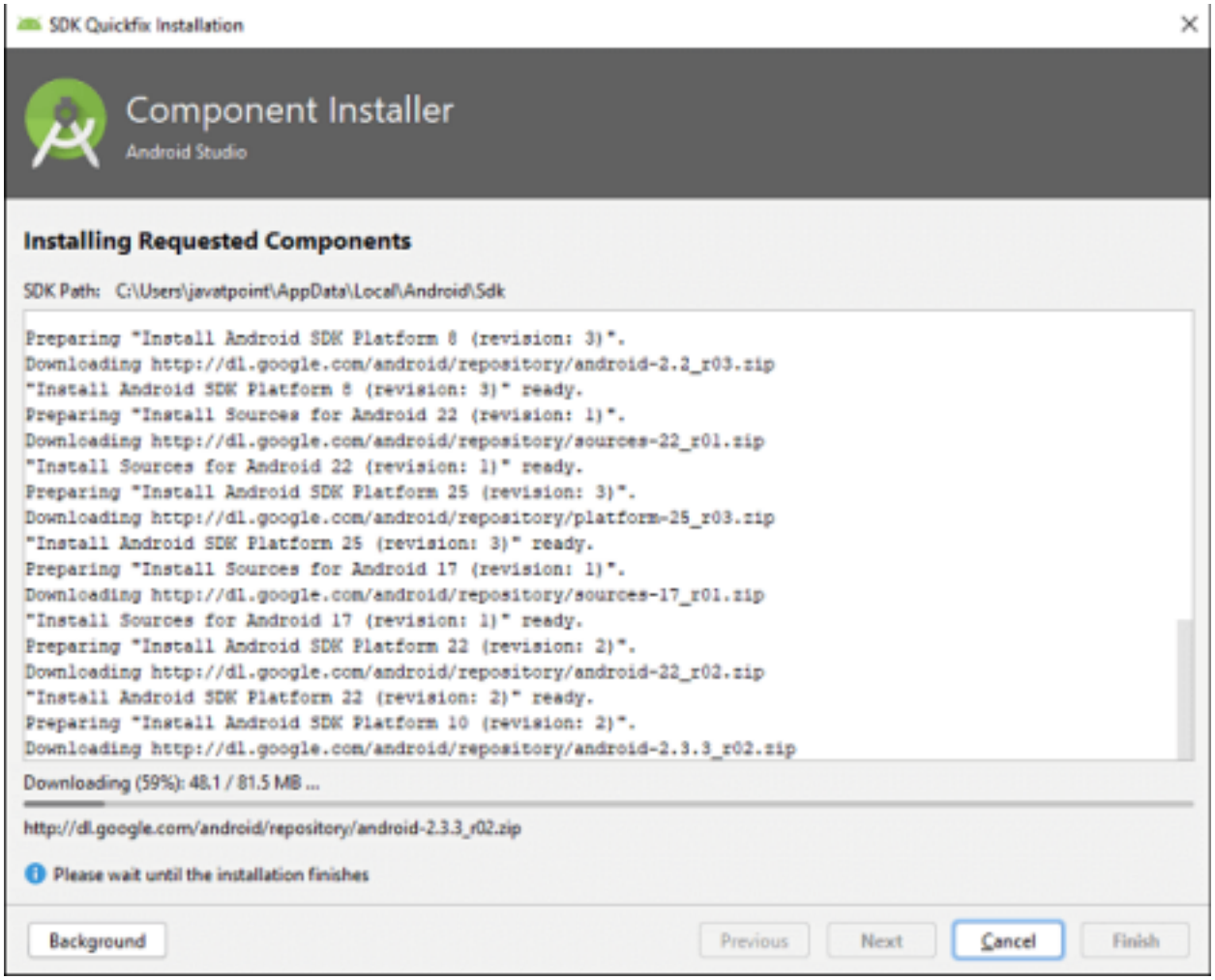
Step 7.1: Download the latest Android Studio executable or zip file from the official site.

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

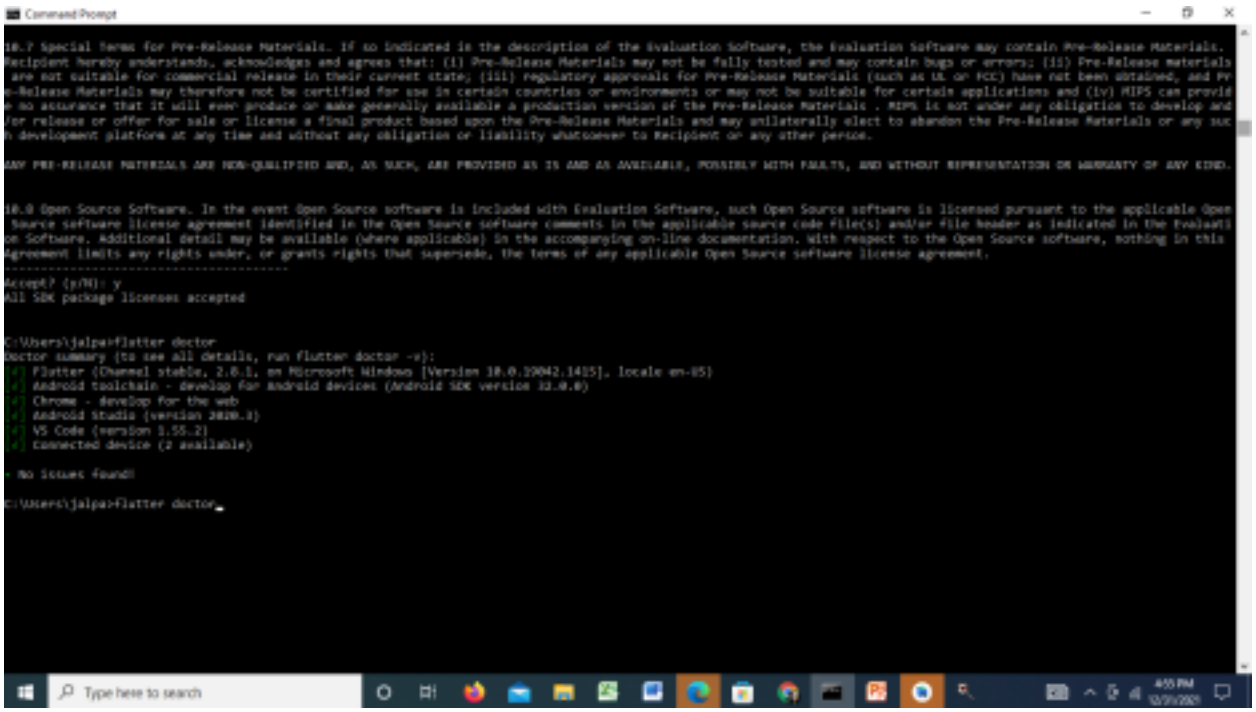
Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



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

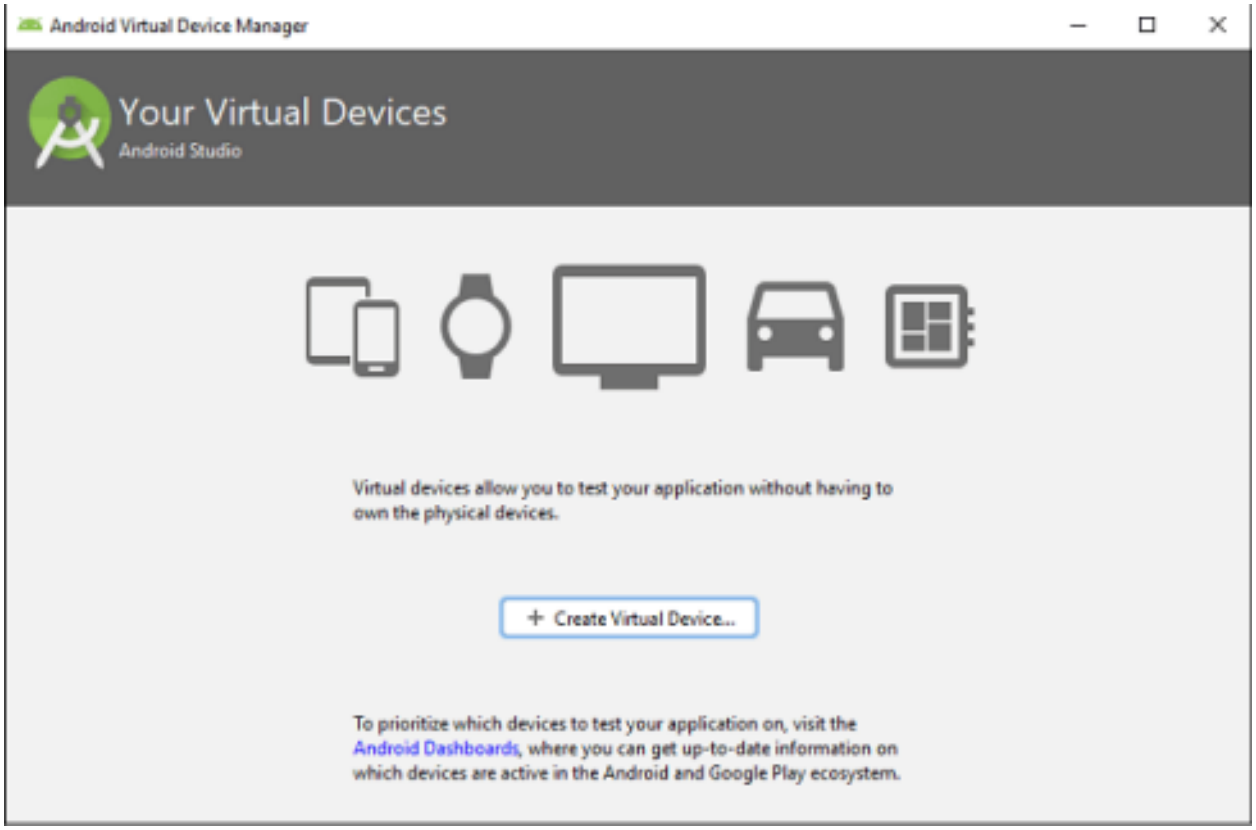


Step 7.5 run the $ **flutter doctor** command and Run flutter doctor --android-licenses command.



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

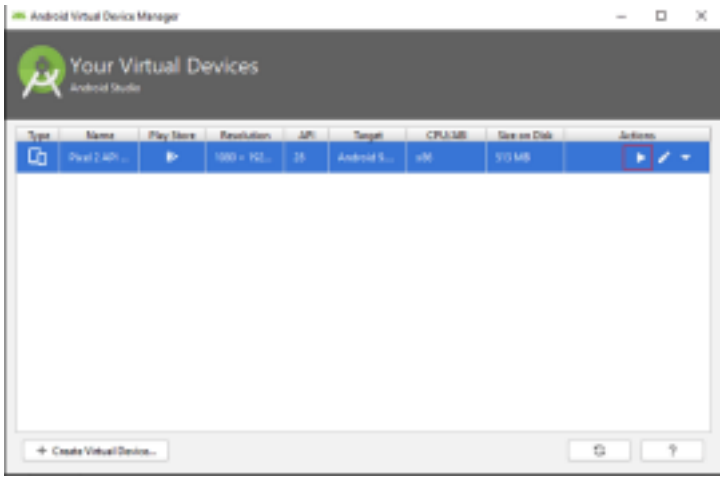
Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.



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

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

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

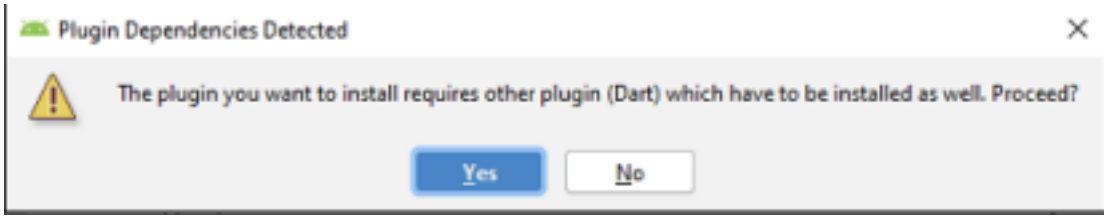


Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

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

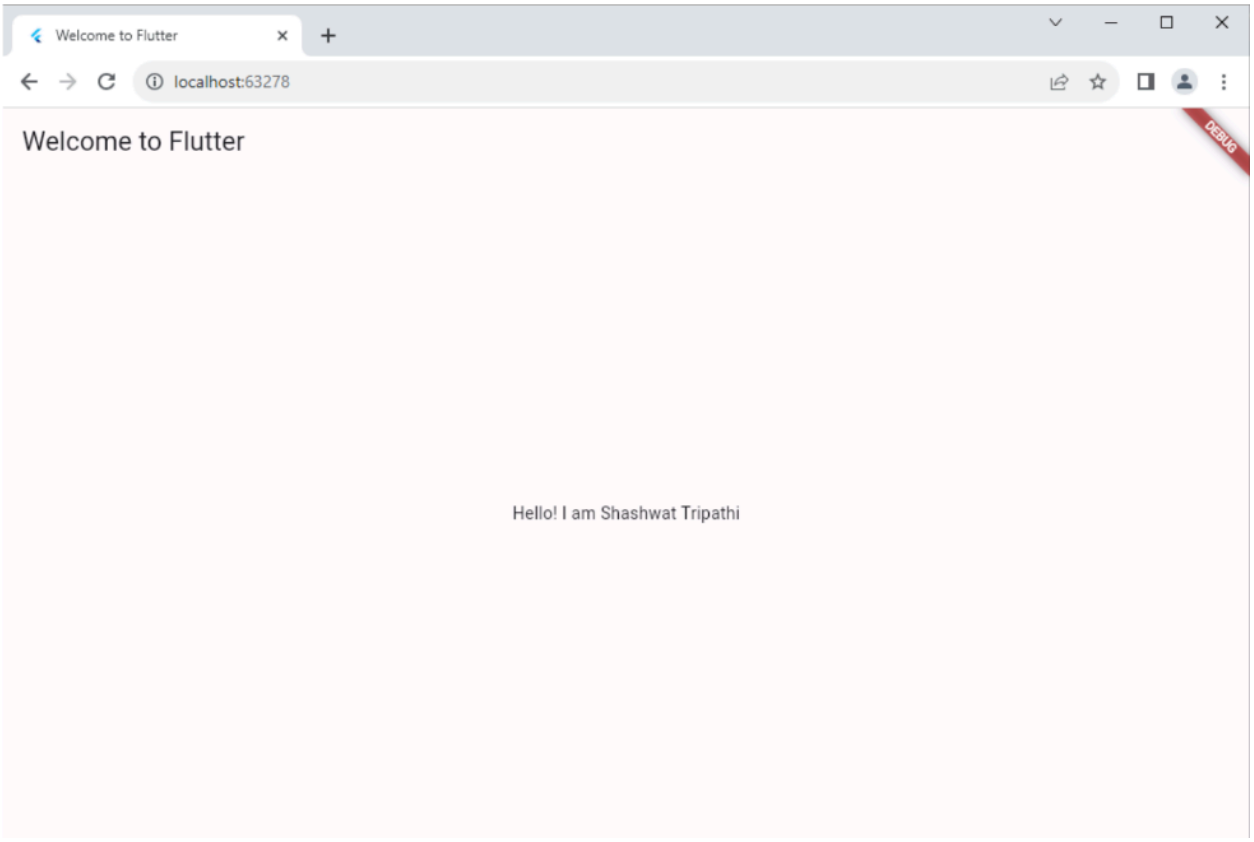
Step 9.1: Open the Android Studio and then go to File->Settings->Plugins.

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



Step 9.3: Restart the Android Studio.

OUTPUT :



CONCLUSION : Thus, we have successfully installed flutter and Android Studio and executed a basic project.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	64
Name	Shashwat Tripathi
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Experiment 2

Shashwat Tripathi
D15A Batch C
Roll No: 64

AIM: To design Flutter UI by including common widgets.

THEORY:

Flutter Widgets: An Overview

Flutter widgets are the basic building blocks that construct the user interface of a Flutter application. They are responsible for defining the structure, appearance, and behavior of the app. Here are some fundamental widgets:

StatelessWidget and StatefulWidget:

StatelessWidget:

Represents an immutable part of the user interface.
It does not change over time and does not depend on any mutable state.

StatefulWidget:

Represents a mutable part of the user interface.
Can change over time based on user interactions or other factors.

1. Flutter Scaffold:

The Scaffold widget is the basic structure for a Flutter app, providing a layout for the visual elements.
It includes an AppBar, BottomNavigationBar, and a body for the main content.

2. Flutter Container:

The Container widget is a versatile box model that can contain other widgets.
It's used for layout, padding, margin, decoration, and constraints.

3. Flutter Row & Column:

Row and Column widgets help in arranging child widgets horizontally (Row) or vertically (Column).
Useful for creating flexible and responsive layouts.

4. Flutter Text:

The Text widget is used to display text on the screen.
It supports various styling options like font size, color, and alignment.

5. Flutter TextField:

TextField is a widget for capturing user input, such as text, numbers, or passwords.
The onChanged property is commonly used for dynamic updates based on user input.

6. Flutter Buttons:
Button widgets, such as ElevatedButton or TextButton, trigger actions when pressed.
Provide a way for users to interact with the app.

7. Flutter Forms:
The Form widget helps in managing a group of TextFormField widgets.
Facilitates the validation and submission of user input.

8. Flutter Icons:
The Icon widget displays icons from various icon libraries, such as Material Icons or custom icons.
Enhances visual elements and conveys meaning through symbols.

Key Design Principles:
Consistency: Use of common widgets fosters a consistent design language throughout the app.

Responsive Layouts: Row and Column help create responsive and flexible layouts, adapting to different screen sizes.

User Input Handling: TextField and Form widgets facilitate user input handling, ensuring data integrity and validation.

Interactive Elements: Buttons and icons contribute to the interactivity and user engagement of the app.

Visual Styling: Container and styling properties of widgets allow for visual customization and theming.

Code:
main.dart : Main entry point of our flutter app

```
import 'package:flutter/material.dart';
import 'package:tiktok_shashwat/constants.dart';
import
'package:tiktok_shashwat/views/screens/auth/login_screen.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  const MyApp({super.key});
  // This widget is the root of your application.
  @override
```

```
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      // Application name
      title: 'TikTok Clone',
      theme: ThemeData.dark().copyWith(
        scaffoldBackgroundColor: backgroundColor,
      ),
      // A widget which will be started on application startup
      home: LoginScreen(),
    );
  }
}
```

constants.dart : Here the colors are defined to maintain consistency throughout the app.

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

// COLORS
const backgroundColor = Colors.black;
var buttonColor = Colors.red[400];
const borderColor = Colors.grey;
```

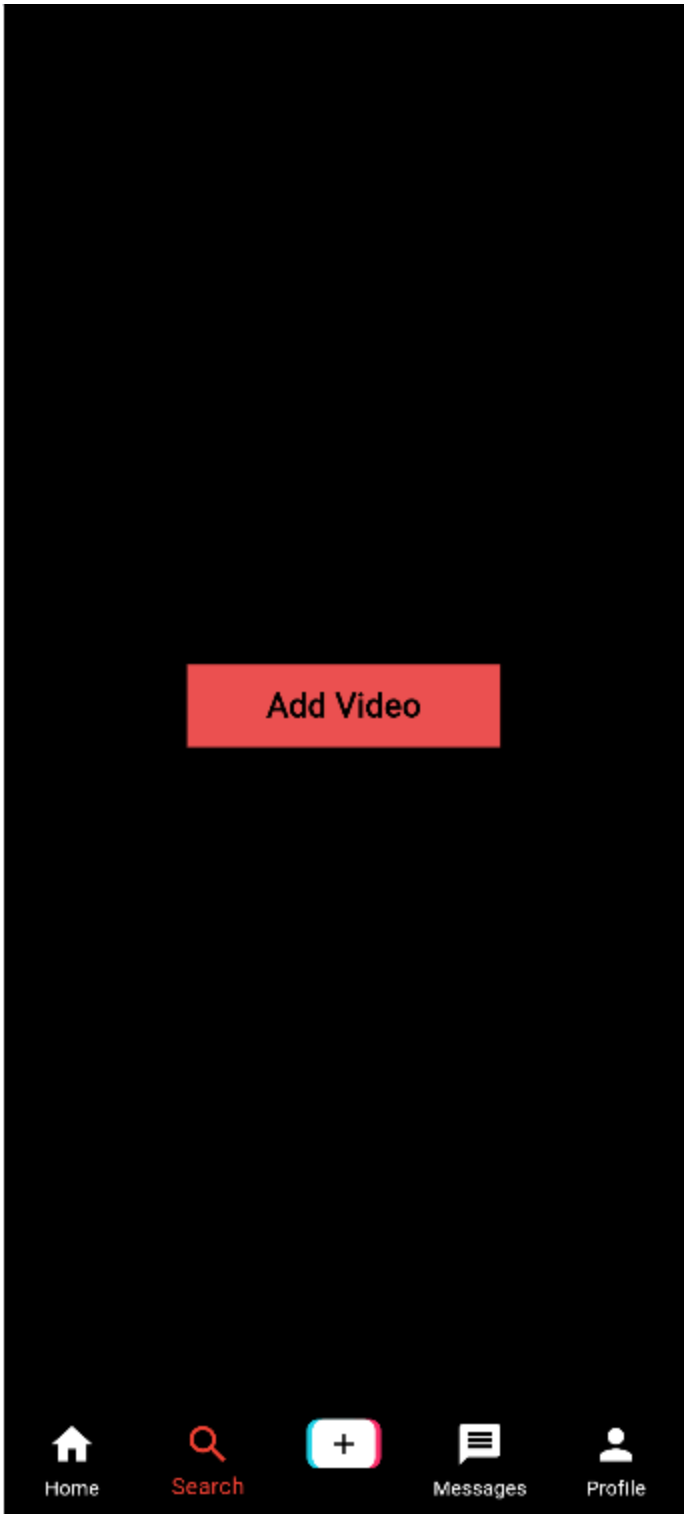
add_video_screen.dart :

```
import 'dart:io';

import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'package:tiktok_tutorial/constants.dart';
import 'package:tiktok_tutorial/views/screens/confirm_screen.dart';

class AddVideoScreen extends StatelessWidget {
  const AddVideoScreen({Key? key}) : super(key: key);

  pickVideo(ImageSource src, BuildContext context) async {
    final video = await ImagePicker().pickVideo(source: src);
    if (video != null) {
      Navigator.of(context).push(
        MaterialPageRoute(
          builder: (context) => ConfirmScreen(
            videoFile: File(video.path),
            videoPath: video.path,
          ),
        ),
      );
    }
```

```
}

showOptionsDialog(BuildContext context) {
  return showDialog(
    context: context,
    builder: (context) => SimpleDialog(
      children: [
        SimpleDialogOption(
          onPressed: () => pickVideo(ImageSource.gallery, context),
          child: Row(
            children: const [
              Icon(Icons.image),
              Padding(
                padding: EdgeInsets.all(7.0),
                child: Text(
                  'Gallery',
                  style: TextStyle(fontSize: 20),
                ),
              ),
            ],
          ),
        ),
        SimpleDialogOption(
          onPressed: () => pickVideo(ImageSource.camera, context),
          child: Row(
            children: const [
              Icon(Icons.camera_alt),
              Padding(
                padding: EdgeInsets.all(7.0),
                child: Text(
                  'Camera',
                  style: TextStyle(fontSize: 20),
                ),
              ),
            ],
          ),
        ),
        SimpleDialogOption(
          onPressed: () => Navigator.of(context).pop(),
          child: Row(
            children: const [
              Icon(Icons.cancel),
              Padding(
                padding: EdgeInsets.all(7.0),
                child: Text(
```

```
                    'Cancel',
                    style: TextStyle(fontSize: 20),
                  ),
                ),
              ],
            ),
          ),
        ],
      ),
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: InkWell(
          onTap: () => showOptionsDialog(context),
          child: Container(
            width: 190,
            height: 50,
            decoration: BoxDecoration(color: buttonColor),
            child: const Center(
              child: Text(
                'Add Video',
                style: TextStyle(
                  fontSize: 20,
                  color: Colors.black,
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```

OUTPUT :



CONCLUSION: Thus, we have used some common widgets like Scaffold, Icon, Container, Button, etc. to create our login page of the application.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	64
Name	Shashwat Tripathi
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Experiment 3

Shashwat Tripathi
D15A Batch C
Roll No: 64

AIM: To design Flutter UI by including common widgets (using Images).

THEORY:

Widgets as Building Blocks:

Flutter apps are built using widgets, which are reusable components that represent UI elements.
Common widgets like Text, Image, Container, Row, Column, ListView, etc., provide building blocks for your UI.
Widgets can be nested within each other to create complex layouts and hierarchies.

Image Handling:

Flutter provides various ways to display images in your UI:
Asset Images: Images stored within your app's assets folder. Use AssetImage widget.
Network Images: Images loaded from URLs. Use NetworkImage widget.
Memory Images: Images loaded from memory buffers. Use MemoryImage widget.
File Images: Images loaded from local files. Use FileImage widget.
Each widget offers customization options like scaling, fitting, and alignment.

Layout and Composition:

Flutter uses a flexible layout system based on widgets.
Widgets like Container can define padding, margins, and background colors.
Layout widgets like Row and Column arrange children horizontally or vertically.
Stack widget allows layering widgets with positioning control.
Use padding, margins, and alignment properties to fine-tune the visual hierarchy.

Best Practices:

Choose the appropriate image widget based on the source and loading strategy.
Optimize image sizes and compression to improve performance.
Use placeholder widgets while images are loading to enhance user experience.
Consider using image caches to avoid redundant downloads.
Leverage Flutter's built-in animation features for smooth transitions and effects.
Follow accessibility guidelines to ensure your UI is usable for everyone.

Additional Considerations:

Explore advanced image manipulation packages like image and cached_network_image.
Experiment with different layout widgets and techniques to achieve desired UI structures.
Test your UI on different devices and screen sizes for responsiveness.
Consider incorporating image gestures like tapping, zooming, and panning.

Code:
profile.dart

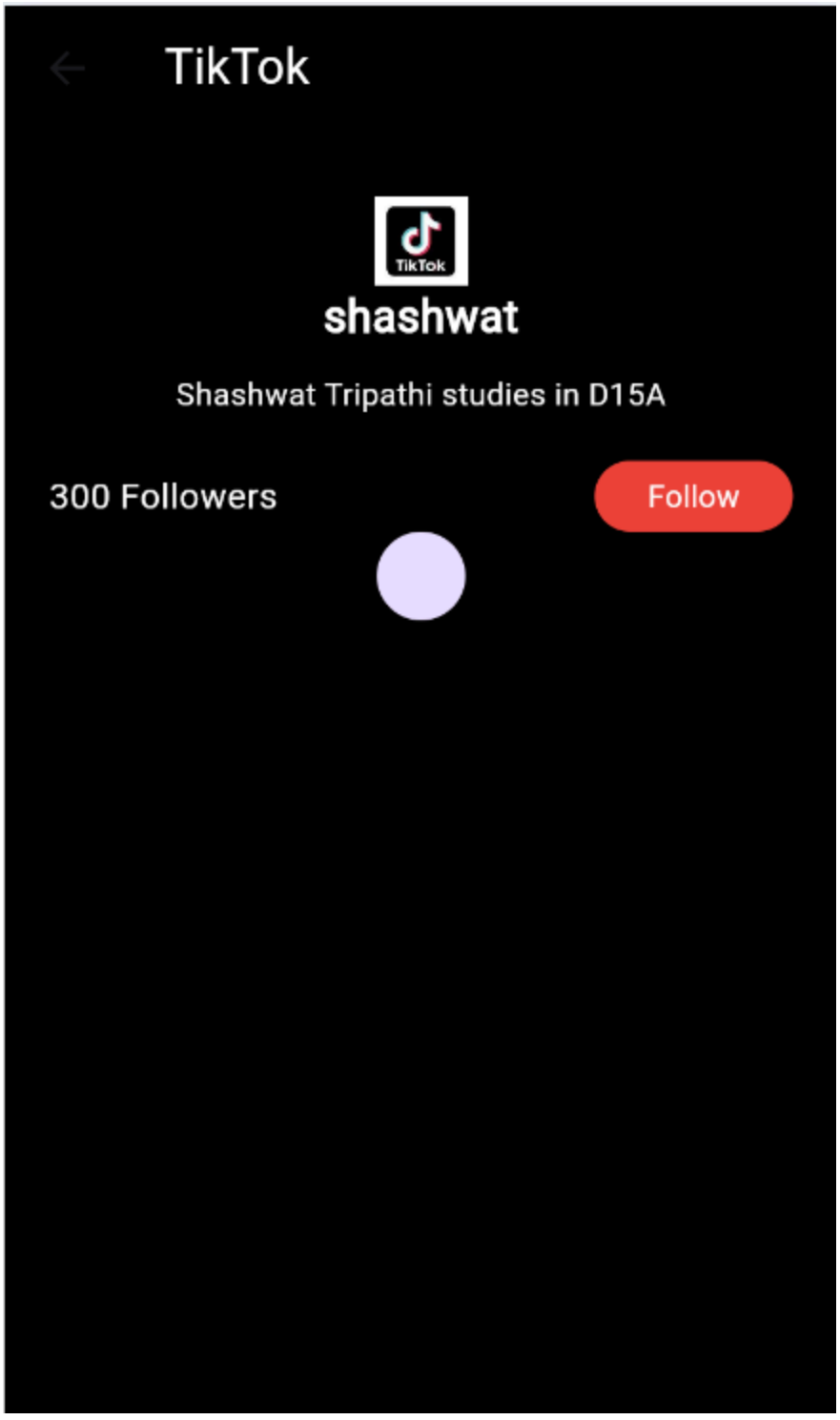
```
import 'package:flutter/material.dart';
import
'package:flutter_tiktok_shashwat/features/user_auth/presentation/page
s/upload.dart';

class ProfilePage extends StatefulWidget {
  // Pass user information as arguments
  final String userName = "shashwat";
  final String bio = "Shashwat Tripathi studies in D15A";
  final String profileImageUrl = "";
  final int followerCount = 300;
  final bool isFollowing = false;

  const ProfilePage({
    Key? key,
    // required this.userName,
    // required this.bio,
    // required this.profileImageUrl,
    // required this.followerCount,
    // required this.isFollowing,
  }) : super(key: key);

  @override
  State<ProfilePage> createState() => _ProfilePageState();
}

class _ProfilePageState extends State<ProfilePage> {
  bool _isFollowing = false;

  @override
  void initState() {
    super.initState();
    _isFollowing = widget.isFollowing;
  }

  void _toggleFollowing() {
    setState(() {
      _isFollowing = !_isFollowing;
    });
  }

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

```
    return Scaffold(
      backgroundColor:
          Colors.black, // Set background color to black for TikTok
theme
      appBar: AppBar(
        backgroundColor: Colors.black, // Apply black background to
AppBar
        title: Text(
          "TikTok",
          style:
              TextStyle(color: Colors.white), // Set title text color
to white
        ),
      ),
      body: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(20.0),
          child: Column(
            children: [
              // Profile Image
              // CircleAvatar(
              //   backgroundImage:
NetworkImage(widget.profileImageUrl),
              //   radius: 50.0,
              // ),
              // SizedBox(height: 10.0),
              SizedBox(height: 10), // Adjust the height as needed
              Image.asset(
                'assets/tiktok_logo.jpg', // Replace with the correct
path to your TikTok logo image
                height: 40, // Adjust height as needed
              ),

              // Username
              Text(
                widget.userName,
                style: TextStyle(
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,
                  color: Colors.white, // Set text color to white
                ),
              ),
              SizedBox(height: 10.0),

              // Bio
```

```
              Text(
                widget.bio,
                style:
                    TextStyle(color: Colors.white), // Set text color
to white
              ),
              SizedBox(height: 20.0),

              // Follower Count
              Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                  Text(
                    "${widget.followerCount.toString()} Followers",
                    style: TextStyle(
                        fontSize: 16.0,
                        color: Colors.white), // Set text color to
white
                  ),
                  // Follow Button
                  ElevatedButton(
                    onPressed: _toggleFollowing,
                    child: Text(
                      _isFollowing ? "Following" : "Follow",
                      style: TextStyle(
                          color: Colors.white), // Set text color to
white
                    ),
                    style: ElevatedButton.styleFrom(
                      primary: Colors.red, // Set button color to red
                    ),
                  ),
                ],
              ),

              // ... other profile content (e.g., posts, etc.)

              Align(
                alignment: Alignment.bottomCenter,
                child: Padding(
                  padding: const EdgeInsets.only(bottom: 20.0),
                  // child: FloatingActionButton(
                  //   onPressed: () => Navigator.pushNamed(context,
                  //       '/upload'), // Replace with your upload
page route name
```

```
                  //   backgroundColor: Colors.red, // Set button
color to red
                  //   child: Icon(Icons.add),
                  // ),
                  child: GestureDetector(
                    onTap: () {
                      Navigator.of(context).push(
                        MaterialPageRoute(
                            builder: (context) =>
VideoUploaderPage()),
                      );
                    },
                    child: CircleAvatar(
                      backgroundImage:
NetworkImage(widget.profileImageUrl),
                      radius: 20.0,
                    ),
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

OUTPUT :



CONCLUSION: Thus, we have used some common widgets like Images to create our login page of the application.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	64
Name	Shashwat Tripathi
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Experiment 4

Shashwat Tripathi
D15A Batch C
Roll No: 64

AIM: To create an interactive Form using form widget

THEORY:

User Interface Design: Interactive forms require a well-designed user interface (UI) to ensure ease of use and clarity for users. Considerations include layout, labeling of fields, grouping related fields, and providing appropriate input controls (text fields, checkboxes, radio buttons, dropdown menus, etc.).

Form Structure: Define the structure of the form, including the types of information you want to collect from users. This could include personal details, contact information, preferences, or any other relevant data.

Form Widgets: Form widgets are the interactive elements used to collect data from users. Common form widgets include text fields, text areas, checkboxes, radio buttons, dropdown menus, and buttons. Each widget serves a specific purpose and is used to capture different types of input.

Validation: Implement validation rules to ensure that the data entered by users is correct and complete. Validation can include checking for required fields, enforcing data formats (e.g., email addresses, phone numbers), and validating input ranges or constraints.

Error Handling: Provide feedback to users when errors occur during form submission or validation. This helps users understand what went wrong and how to correct it. Error messages should be clear, concise, and displayed near the relevant input fields.

Accessibility: Ensure that the form is accessible to all users, including those with disabilities. This involves using semantic HTML, providing appropriate labels and instructions, ensuring keyboard navigation, and testing with assistive technologies.

Submission Handling: Define how form submissions are processed. This may involve sending data to a server-side script for processing, storing data in a database, or triggering other actions based on user input.

User Experience (UX): Consider the overall user experience when designing the form. Aim for simplicity, clarity, and efficiency to minimize user frustration and increase completion rates.

Testing and Iteration: Test the form extensively to identify any usability issues, bugs, or errors. Gather feedback from users and iterate on the design based on their input to improve the overall user experience.

Code:

main.dart : Main entry point of our flutter app

```
import 'package:flutter/material.dart';
import 'package:tiktok_shashwat/constants.dart';
import
'package:tiktok_shashwat/views/screens/auth/login_screen.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  const MyApp({super.key});
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      // Application name
      title: 'TikTok Clone',
      theme: ThemeData.dark().copyWith(
        scaffoldBackgroundColor: backgroundColor,
      ),
      // A widget which will be started on application startup
      home: LoginScreen(),
    );
  }
}
```

login_screen.dart : This contains the login page UI and functionalities, at the moment it is not connected to the firebase.

```
import 'package:flutter/material.dart';
import 'package:tiktok_shashwat/constants.dart';
import
'package:tiktok_shashwat/views/screens/auth/signup_screen.dart';
import 'package:tiktok_shashwat/views/widgets/text_input_field.dart';

class LoginScreen extends StatelessWidget {
  LoginScreen({Key? key}) : super(key: key);

  final TextEditingController _emailController =
TextEditingController();
  final TextEditingController _passwordController =
TextEditingController();

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

```
return Scaffold(
  body: Container(
    alignment: Alignment.center,
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Text(
          'Tiktok Clone',
          style: TextStyle(
            fontSize: 35,
            color: buttonColor,
            fontWeight: FontWeight.w900,
          ),
        ),
        const Text(
          'Login',
          style: TextStyle(
            fontSize: 25,
            fontWeight: FontWeight.w700,
          ),
        ),
        const SizedBox(
          height: 25,
        ),
        Container(
          width: MediaQuery.of(context).size.width,
          margin: const EdgeInsets.symmetric(horizontal: 20),
          child: TextInputField(
            controller: _emailController,
            labelText: 'Email',
            icon: Icons.email,
          ),
        ),
        const SizedBox(
          height: 25,
        ),
        Container(
          width: MediaQuery.of(context).size.width,
          margin: const EdgeInsets.symmetric(horizontal: 20),
          child: TextInputField(
            controller: _passwordController,
            labelText: 'Password',
            icon: Icons.lock,
            isObscure: true,
          ),
```

```
),
const SizedBox(
  height: 30,
),
Container(
  width: MediaQuery.of(context).size.width - 40,
  height: 50,
  decoration: BoxDecoration(
    color: buttonColor,
    borderRadius: const BorderRadius.all(
      Radius.circular(5),
    ),
  ),
  child: InkWell(
    // onTap: () => authController.loginUser(
    //   _emailController.text,
    //   _passwordController.text,
    // ),
    child: const Center(
      child: Text(
        'Login',
        style: TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.w700,
        ),
      ),
    ),
  ),
),
const SizedBox(
  height: 15,
),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    const Text(
      'Don\'t have an account? ',
      style: TextStyle(
        fontSize: 20,
      ),
    ),
    InkWell(
      // onTap: () => Navigator.of(context).push(
      //   MaterialPageRoute(
      //       builder: (context) => SignupScreen(),
```

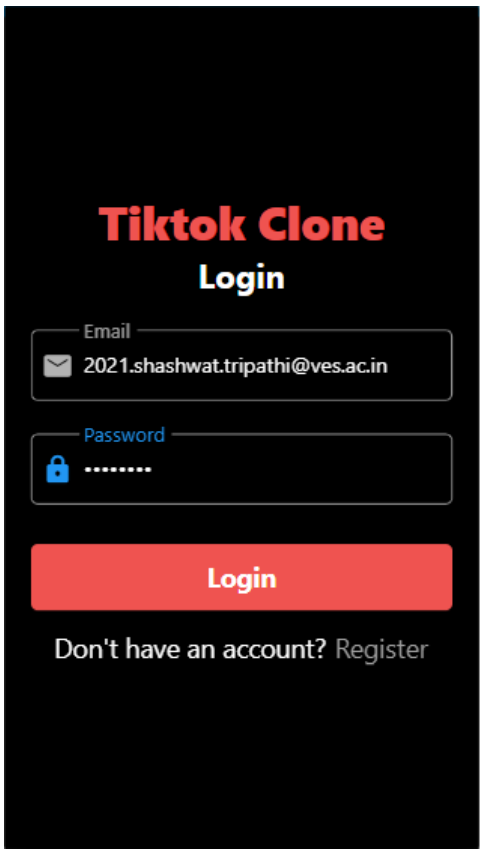
```
                  //       ),
                  // ),
                  child: Text(
                    'Register',
                    style: TextStyle(fontSize: 20, color:
borderColor),
                  ),
                ),
              ],
            ),
          ],
        ),
      ),
    );
  }
}
```

text_input_field.dart : This contains the text input field box code which is used in our login page and will be used further in our application, hence created a separate block of code for it.

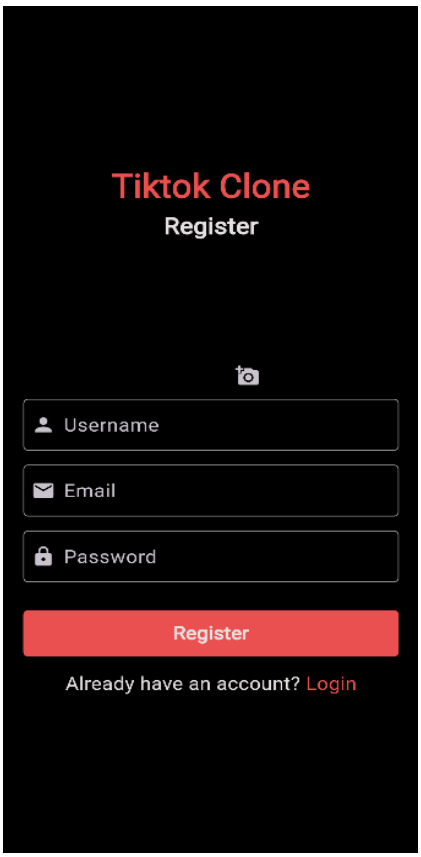
```
import 'package:flutter/material.dart';
import 'package:tiktok_shashwat/constants.dart';

class TextInputField extends StatelessWidget {
  final TextEditingController controller;
  final String labelText;
  final bool isObscure;
  final IconData icon;
  const TextInputField({
    Key? key,
    required this.controller,
    required this.labelText,
    this.isObscure = false,
    required this.icon,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return TextField(
      controller: controller,
      decoration: InputDecoration(
        labelText: labelText,
        prefixIcon: Icon(icon),
        labelStyle: const TextStyle(
          fontSize: 20,
```

```
        ),
        enabledBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(5),
            borderSide: const BorderSide(
              color: borderColor,
            )),
        focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(5),
            borderSide: const BorderSide(
              color: borderColor,
            )),
      ),
      obscureText: isObscure,
    );
  }
}
```

OUTPUT :



Login Screen



Signup Screen

CONCLUSION: Thus, we have used some common widgets like Scaffold, Textinputfield, Icon, Container, Button, etc. to create our login page of the application.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	64
Name	Shashwat Tripathi
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

Experiment 5

Shashwat Tripathi
D15A Batch C
Roll No: 64

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

THEORY:
Understanding the Flow:

Flutter apps often involve navigating between different screens or views.
Navigation and routing provide the mechanism to manage this flow of users through your app.
Gestures allow users to interact with the UI and trigger navigation actions.
Navigation Concepts:

Navigator: A widget responsible for managing the navigation stack and transitions.
Routing: The process of defining routes and their corresponding destinations.
Routes: Screens or views that users can navigate to within your app.
Navigation Stack: A stack of routes, where the top element represents the currently active screen.
Transitions: Animations used to create visual continuity between screens during navigation.
Routing Strategies:

Declarative Routing: Routes are defined explicitly using the Navigator widget and its associated classes.
Package-based Routing: Third-party packages like go_router or routemaster offer additional features and flexibility.
Dynamic Routing: Routes can be generated based on runtime conditions or user input.
Gesture Recognition:

Flutter uses a gesture recognizer system to detect various user interactions like taps, swipes, and drags.
Common gesture recognizers include TapGestureRecognizer, SwipeGestureRecognizer, and PanGestureRecognizer.
You can define custom gesture recognizers for specific needs.
Interactions and Navigation:

Gestures can be used to trigger navigation actions.
For example, a tap on a button might use Navigator.push to push a new route onto the stack.
Swiping left or right might use Navigator.pop to go back to the previous screen.
Gestures can also be used to manipulate UI elements within a screen.
Best Practices:

Use clear and consistent navigation patterns throughout your app.

Provide visual cues to indicate interactive elements.
Design gestures that feel natural and intuitive for users.
Consider accessibility guidelines when designing gestures.
Test your navigation flow on different devices and screen sizes.
Additional Considerations:

Explore advanced navigation features like nested navigators and deep linking.
Use animation and transitions to enhance the user experience.
Consider integrating gesture recognizers with other UI elements like images and text.

Code:
home_page.dart

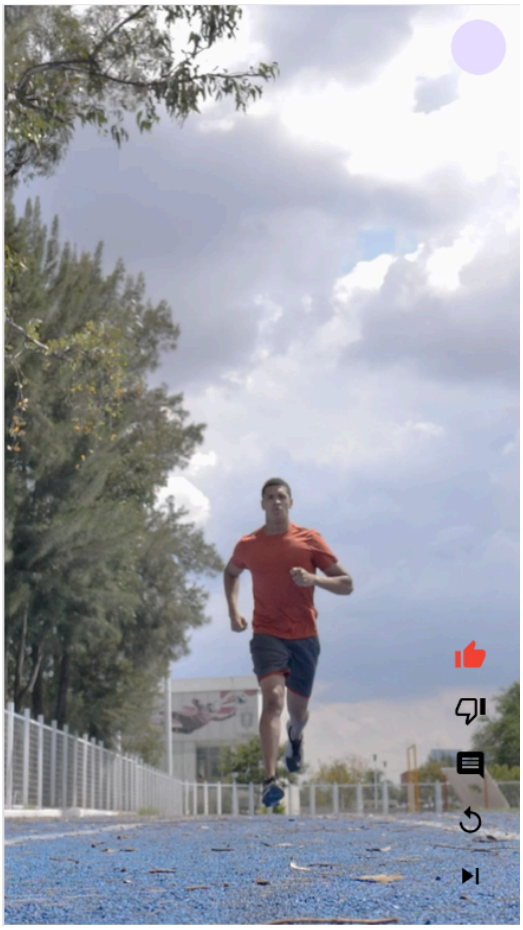
```
import 'package:flutter/material.dart';
import
'package:flutter_tiktok_shashwat/features/user_auth/presentation/page
s/profile.dart';
import
'package:flutter_tiktok_shashwat/features/user_auth/presentation/widg
ets/bottom_navbar.dart';
import 'package:video_player/video_player.dart';

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

  String get profileImageUrl => "null";

  // Navigator.pushNamed(context, '/homepage', arguments: {'user':
user, 'currentIndex': 0});

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  late VideoPlayerController controller;
  bool isLiked = false;
  bool isDisliked = false;
  bool isCommentOpen = false;
  List<String> videoUrls = [
    'assets/running.mp4',
    'assets/flowers.mp4',
    'assets/lights.mp4',
    'assets/rowing.mp4',
    'assets/traffic.mp4',
  ];
```

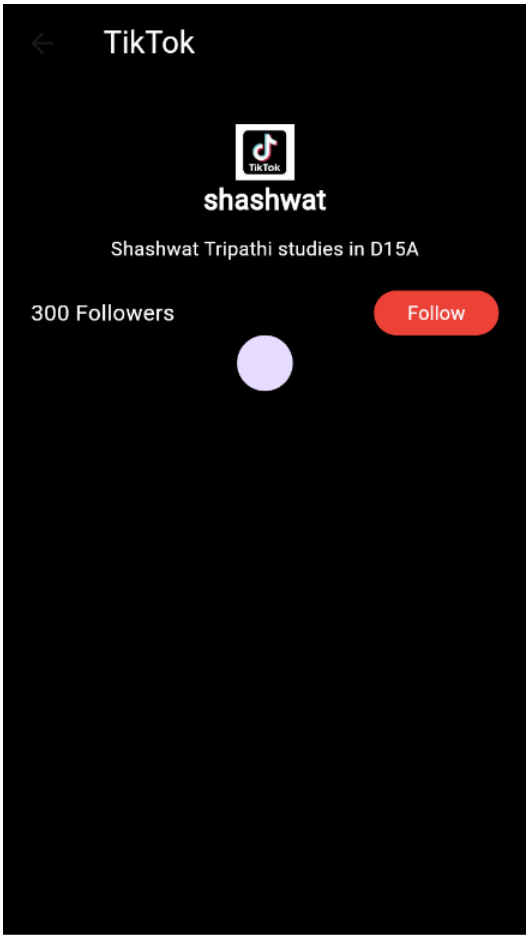
```
  int currentVideoIndex = 0;

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

  loadVideoPlayer() async {
    controller =
VideoPlayerController.asset(videoUrls[currentVideoIndex]);
    await controller.initialize();
    controller.addListener(() {
      setState(() {});
    });
    controller.play(); // Start playing the video automatically
    setState(() {});
  }

  @override
  void dispose() {
    controller.removeListener(() {});
    controller.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    const String profileImageUrl = "";
    return Scaffold(
      backgroundColor: Colors.black,
      body: Stack(
        children: [
          // Video Player
          Expanded(
            child: controller.value.isInitialized
                ? AspectRatio(
                    aspectRatio: controller.value.aspectRatio,
                    child: VideoPlayer(controller),
                  )
                : Center(child: CircularProgressIndicator()),
          ),
          Positioned(
            top: 10.0,
            right: 10.0,
```

```
            child: GestureDetector(
              onTap: () {
                Navigator.of(context).push(
                  MaterialPageRoute(builder: (context) =>
ProfilePage()),
                );
              },
              child: CircleAvatar(
                backgroundImage:
NetworkImage(widget.profileImageUrl),
                radius: 20.0,
              ),
            ),
          ),

          // Controls
          Positioned(
            bottom: 0,
            right: 0,
            child: Padding(
              padding: const EdgeInsets.all(16.0),
              child: Column(
                mainAxisAlignment: MainAxisAlignment.end,
                children: [
                  IconButton(
                    icon: Icon(
                      isLiked ? Icons.thumb_up :
Icons.thumb_up_outlined,
                      color: isLiked ? Colors.red : Colors.black,
                    ),
                    onPressed: () {
                      setState(() {
                        isLiked = !isLiked;
                        isDisliked = false;
                      });
                    },
                  ),
                  IconButton(
                    icon: Icon(
                      isDisliked ? Icons.thumb_down :
Icons.thumb_down_outlined,
                      color: isDisliked ? Colors.blue : Colors.black,
                    ),
                    onPressed: () {
                      setState(() {
```

```
                        isDisliked = !isDisliked;
                        isLiked = false;
                      });
                    },
                  ),
                  IconButton(
                    icon: Icon(Icons.comment, color: Colors.black),
                    onPressed: () {
                      // ... (Show comments as before)
                    },
                  ),
                  IconButton(
                    icon: Icon(Icons.replay, color: Colors.black),
                    onPressed: () {
                      controller.seekTo(Duration.zero);
                      setState(() {});
                    },
                  ),
                  IconButton(
                    icon: Icon(Icons.skip_next, color: Colors.black),
                    onPressed: () {
                      setState(() {
                        currentVideoIndex =
                            (currentVideoIndex + 1) %
videoUrls.length;
                        loadVideoPlayer();
                      });
                    },
                  ),
                ],
              ),
            ),
          ),
        ],
      ),
    );
  }
}
```

OUTPUT :





CONCLUSION: Thus, we have used navigation, routing and gestures in Flutter App.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	64
Name	Shashwat Tripathi
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Experiment 6

Shashwat Tripathi
D15A Batch C
Roll No: 64

AIM: To Connect Flutter UI with fireBase database

THEORY:
Understanding the Connection:

Firebase Realtime Database (RTDB) and Cloud Firestore are NoSQL databases offered by Firebase.
Firebase provides Flutter libraries (firebase_database and cloud_firestore) to connect your Flutter UI with these databases.
You can read, write, and update data from your UI, synchronizing it in real-time (RTDB) or with flexible query options (Firestore).
Key Concepts:

Database References: Represent a specific location within the database hierarchy.
Data Structure: RTDB uses JSON-like objects, while Firestore uses collections and documents with structured fields.
Authentication: Secure access to your database using Firebase Authentication.
Security Rules: Define read/write permissions for different database paths.
Connection Process:

Initialization: Initialize Firebase app and required libraries in your Flutter app.
Reference Creation: Create references to specific database locations using DatabaseReference (RTDB) or CollectionReference (Firestore).
Data Operations: Read, write, or update data using methods like get(), set(), and update().
Listening: Implement real-time data updates using onChildAdded, onChildChanged, etc. (RTDB) or listeners/snapshots (Firestore).
Best Practices:

Choose the appropriate database (RTDB for real-time needs, Firestore for complex queries and scalability).
Structure your data efficiently based on your app requirements.
Implement robust error handling and user feedback mechanisms.
Use security rules to restrict unauthorized access.
Optimize data access patterns for performance.
Advanced Topics:

Explore offline data persistence (Firestore) and cloud functions for backend logic.
Implement complex queries and data transformations.
Integrate user authentication with database access control.

Additional Considerations:

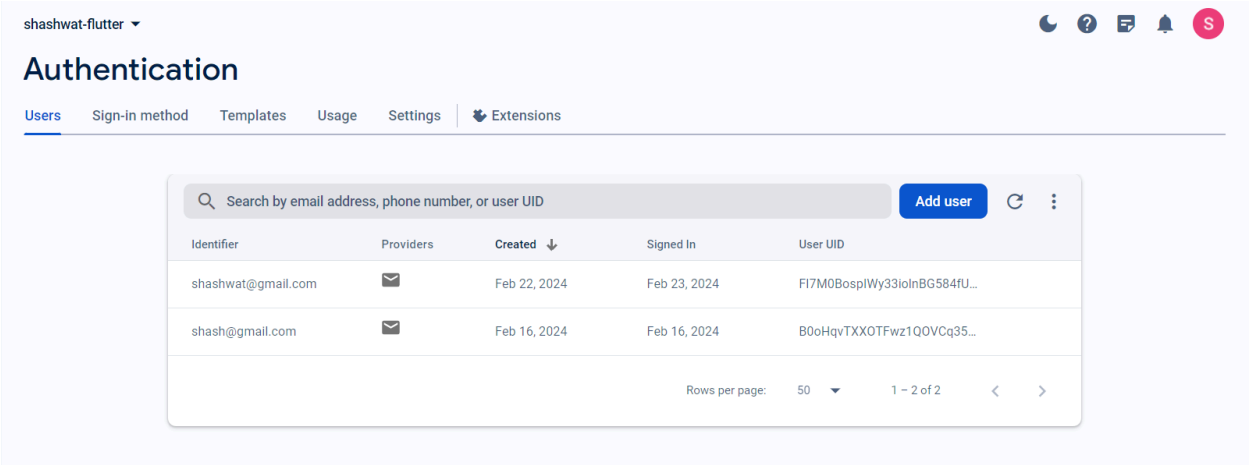
Code:

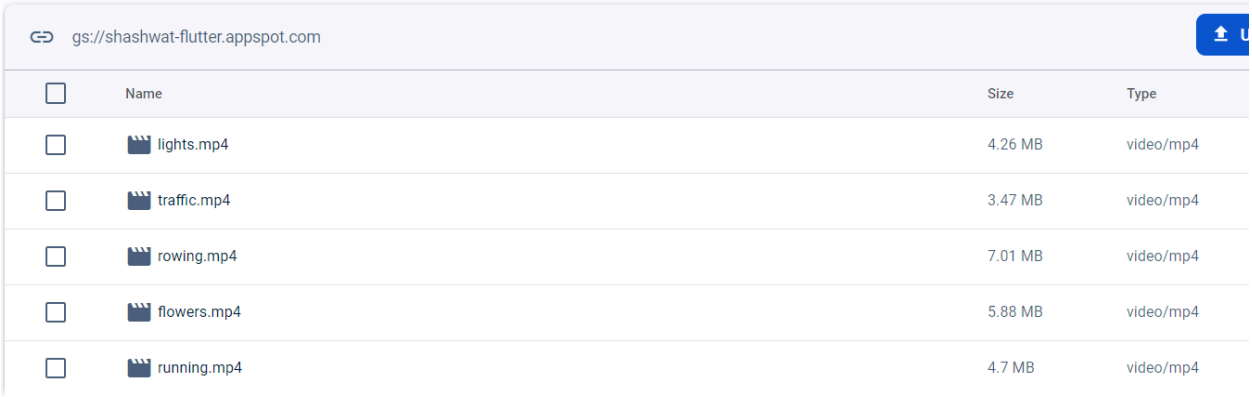
firebase_options.dart

```
// File generated by FlutterFire CLI.
// ignore_for_file: lines_longer_than_80_chars,
avoid_classes_with_only_static_members
import 'package:firebase_core/firebase_core.dart' show
FirebaseOptions;
import 'package:flutter/foundation.dart'
    show defaultTargetPlatform, kIsWeb, TargetPlatform;

/// Default [FirebaseOptions] for use with your Firebase apps.
///
/// Example:
/// ```dart
/// import 'firebase_options.dart';
/// // ...
/// await Firebase.initializeApp(
///   options: DefaultFirebaseOptions.currentPlatform,
/// );
/// ```
class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
      return web;
    }
    switch (defaultTargetPlatform) {
      case TargetPlatform.android:
        return android;
      case TargetPlatform.iOS:
        return ios;
      case TargetPlatform.macOS:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for macos
- '
          'you can reconfigure this by running the FlutterFire CLI
again.',
        );
      case TargetPlatform.windows:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for
windows - '
```

```
          'you can reconfigure this by running the FlutterFire CLI
again.',
        );
      case TargetPlatform.linux:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for linux
- '
          'you can reconfigure this by running the FlutterFire CLI
again.',
        );
      default:
        throw UnsupportedError(
          'DefaultFirebaseOptions are not supported for this
platform.',
        );
    }
  }

  static const FirebaseOptions web = FirebaseOptions(
    apiKey: 'AIzaSyAEhzs4liL0k0_Sawn3DM2JaDUGXYDR_Vs',
    appId: '1:581041793774:web:d95bb71974b88cceff21ff',
    messagingSenderId: '581041793774',
    projectId: 'shashwat-flutter',
    authDomain: 'shashwat-flutter.firebaseapp.com',
    storageBucket: 'shashwat-flutter.appspot.com',
    measurementId: 'G-FSPJTPZXQB',
  );

  static const FirebaseOptions android = FirebaseOptions(
    apiKey: 'AIzaSyApIXoEAlWJ1J3saVcmeNL4qSxuB3JaStA',
    appId: '1:581041793774:android:1f764f270c0c12e5ff21ff',
    messagingSenderId: '581041793774',
    projectId: 'shashwat-flutter',
    storageBucket: 'shashwat-flutter.appspot.com',
  );

  static const FirebaseOptions ios = FirebaseOptions(
    apiKey: 'AIzaSyBWIJtQOrbh96OkJpD1fp4L6zSBdw7cRW0',
    appId: '1:581041793774:ios:7a87673a5c4dbca6ff21ff',
    messagingSenderId: '581041793774',
    projectId: 'shashwat-flutter',
    storageBucket: 'shashwat-flutter.appspot.com',
    iosBundleId: 'com.example.flutterTiktokShashwat',
  );
}
```

OUTPUT :





CONCLUSION: Thus, we have connected firebase to our flutter UI.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	07
Experiment Title.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.
Roll No.	64
Name	Shashwat Tripathi
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

Experiment 7

Name: Shashwat Tripathi
Div: D15A
Roll no: 64

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

Theory:-

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

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

- Difference between PWAs vs. Regular Web Apps:

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

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

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

4. Engaging Approach

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

5. Updated Real-Time Data Access
Another plus point of PWAs is that these apps get updated on their own. They do not demand the end-users to go to the App Store or other such platforms to download the update and wait until installed.
In this app type, the web app developers can push the live update from the server, which reaches the apps residing on the user's devices automatically. Therefore, it is easier for the mobile app developer to provide the best of the updated functionalities and services to the end-users without forcing them to update their app.
6. Discoverable
PWAs reside in web browsers. This implies higher chances of optimizing them as per the Search Engine Optimization (SEO) criteria and improving the Google rankings like that in websites and other web apps.

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

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

- Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.
- Responsive — They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can later become available.
- App-like — They behave with the user as if they were native apps, in terms of interaction and navigation.
- Updated — Information is always up-to-date thanks to the data update process offered by service workers.
- Secure — Exposed over HTTPS protocol to prevent the connection from displaying information or altering the contents.
- Searchable — They are identified as “applications” and are indexed by search engines.
- Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
- Installable — They allow the user to “save” the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to face all the steps and problems related to the use of the app store.
- Linkable — Easily shared via URL without complex installations.
- Offline — Once more it is about putting the user before everything, avoiding the usual error message in case of weak or no connection. The PWA are based on two

particularities: first of all the ‘skeleton’ of the app, which recalls the page structure, even if its contents do not respond and its elements include the header, the page layout, as well as an illustration that signals that the page is loading.

Weaknesses refer to:

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

Code:
index.html of the react app

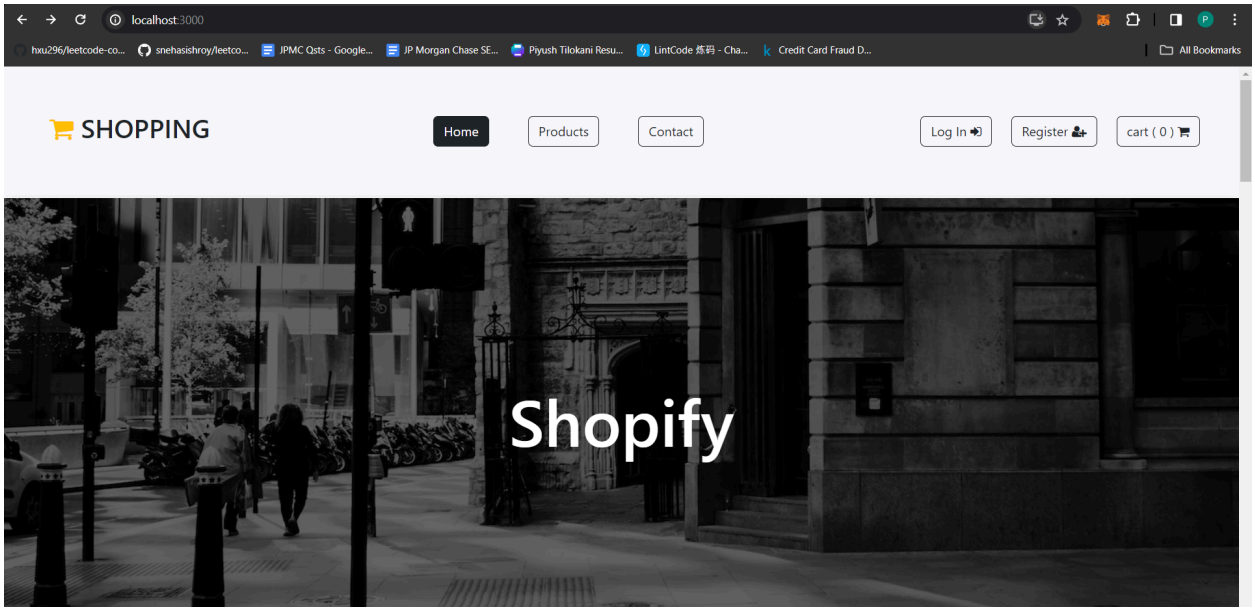
```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <meta name="theme-color" content="#000000" />
  <meta name="description" content="Web site created using create-react-app" />
  <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
  <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

  <meta charset="utf-8">
  <meta name="viewport"
      content="width=device-width,
            initial-scale=1">
  <meta http-equiv="X-UA-Compatible"
      content="ie=edge">

  <!-- Title -->
  <title>PWA Tutorial</title>
```

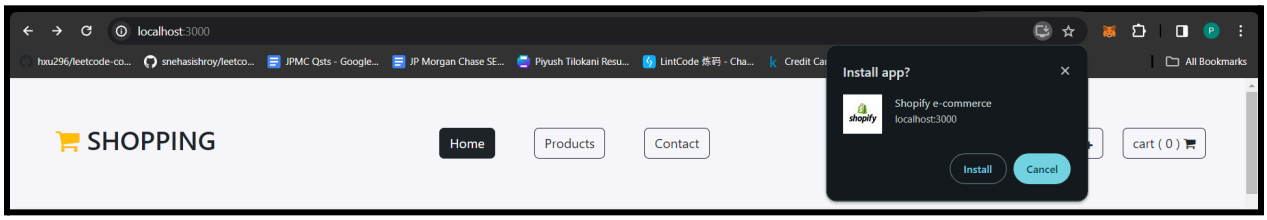
```
  <!-- Meta Tags required for
       Progressive Web App -->
  <meta name=
  "apple-mobile-web-app-status-bar"
        content="#aa7700">
  <meta name="theme-color"
        content="black">

  <!-- Manifest File link -->
  <link rel="manifest"
        href="manifest.json">

  <title>React App</title>
</head>

<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
</body>

</html>
```

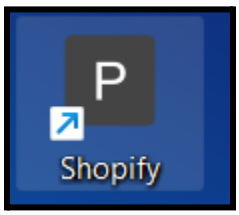
manifest.json

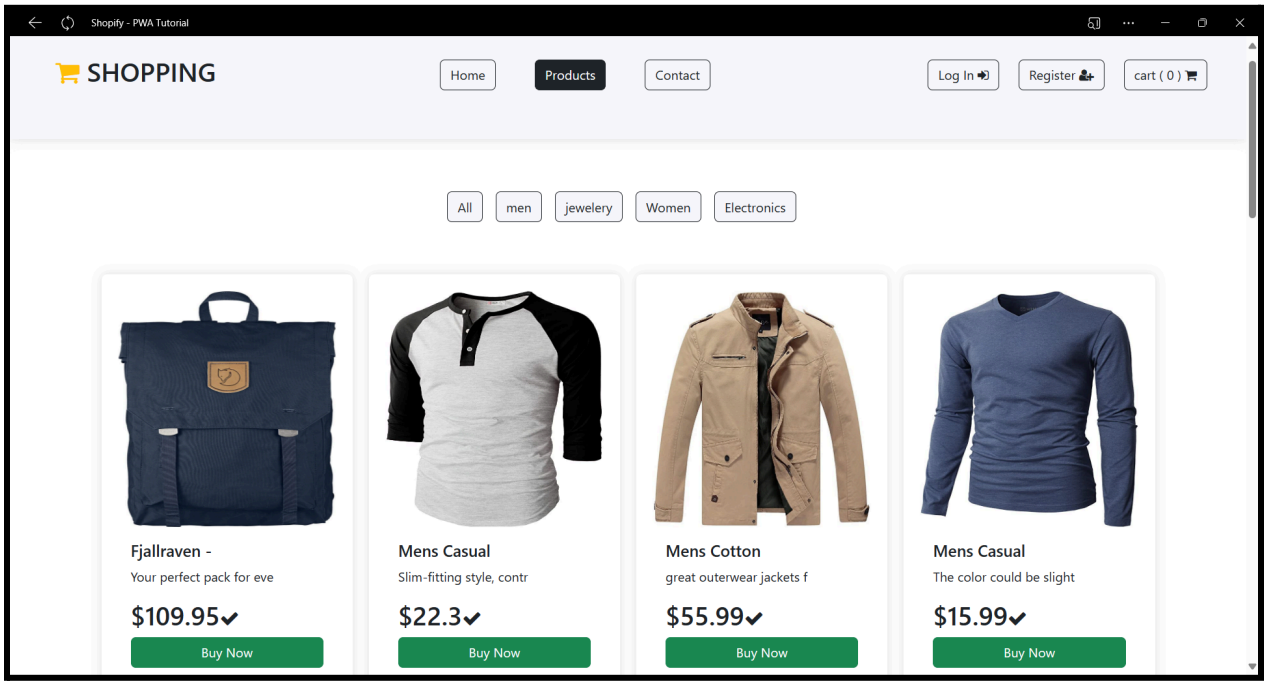
```
{
          "name":"Shopify e-commerce",
          "short_name":"Shopify e-commerce",
          "start_url":"index.html",
          "display":"standalone",
          "background_color":"#5900b3",
          "theme_color":"black",
          "scope": ".",
          "description":"Shopify e-commerce.",
          "icons":[
          {
          "src":"images/icon-192x192.png",
          "sizes":"192x192",
          "type":"image/png"
          },
          {
          "src":"images/icon-512x512.png",
          "sizes":"512x512",
          "type":"image/png"
          }
]
}
```

```
Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

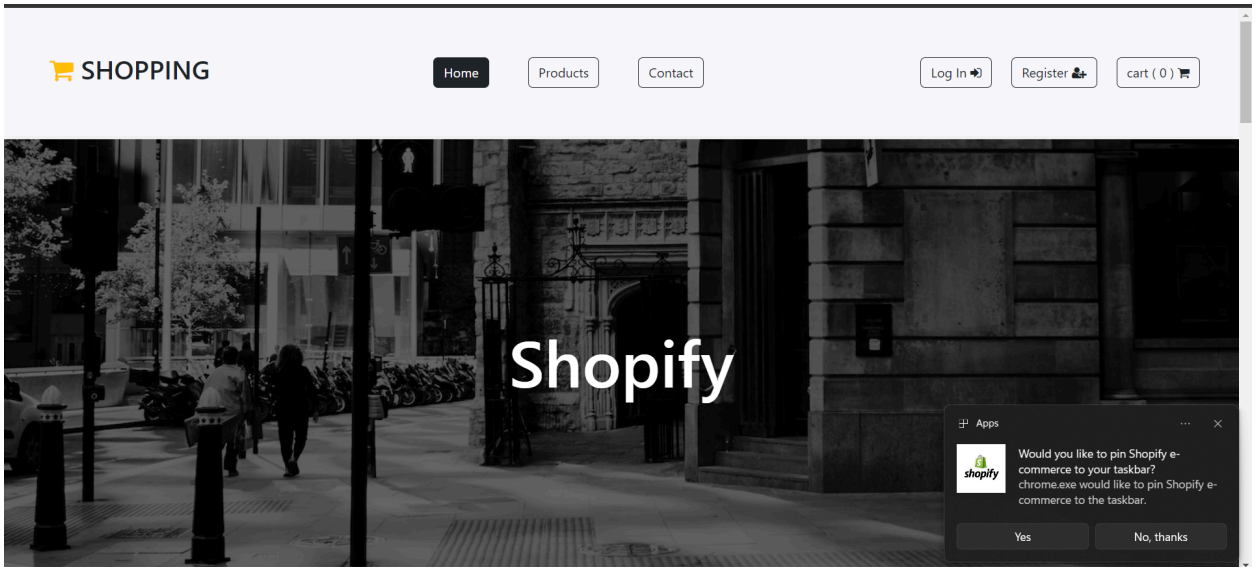








Conclusion: Therefore, I created my first Progressive Web App by adding metadata to the index.html of my e-commerce web app and installed the Application on my local machine.



Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	08
Experiment Title.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA
Roll No.	64
Name	Shashwat Tripathi
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment 8

Name: Shashwat Tripathi
Div: D15A
Roll no: 64

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

Theory:-

Service Worker

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

Things to note about Service Worker:

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

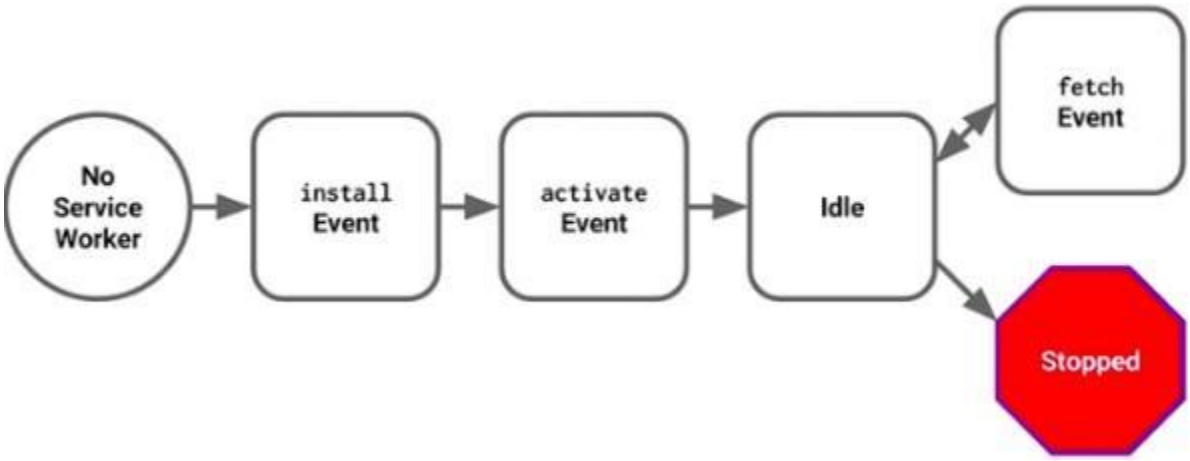
What can we do with Service Workers?

- You can dominate **Network Traffic**
  You can manage all network traffic of the page and do any manipulations. For example, when the page requests a CSS file, you can send plain text as a response or when the page requests an HTML file, you can send a png file as a response. You can also send a true response too.
- You can **Cache**
  You can cache any request/response pair with Service Worker and Cache API and you can access these offline content anytime.
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can’t we do with Service Workers?

- You can’t access the **Window**
  You can’t access the window, therefore, You can’t manipulate DOM elements. But, you can communicate to the window through post Message and manage processes that you want.
- You can’t work it on **80 Port**
  Service Worker just can work on HTTPS protocol. But you can work on localhost during development.

Service Worker Cycle



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

- Registration
- Installation
- Activation

Registration

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

main.js

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

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

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

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

main.js

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

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

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

main.js

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

Installation

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

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

Code:

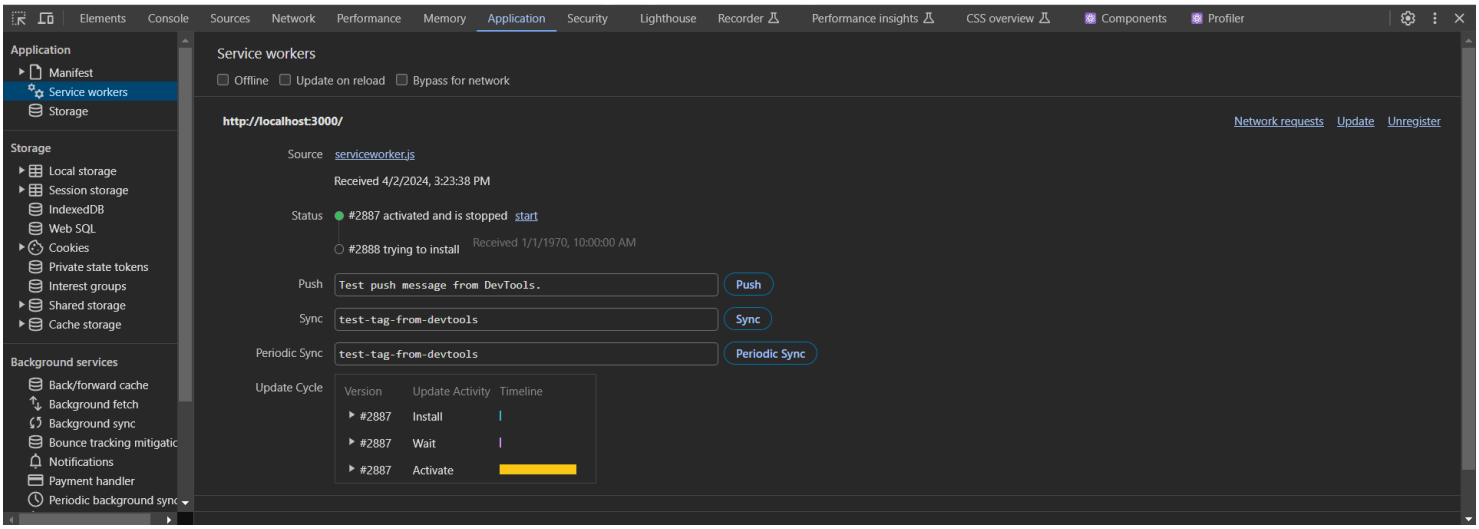
index.html

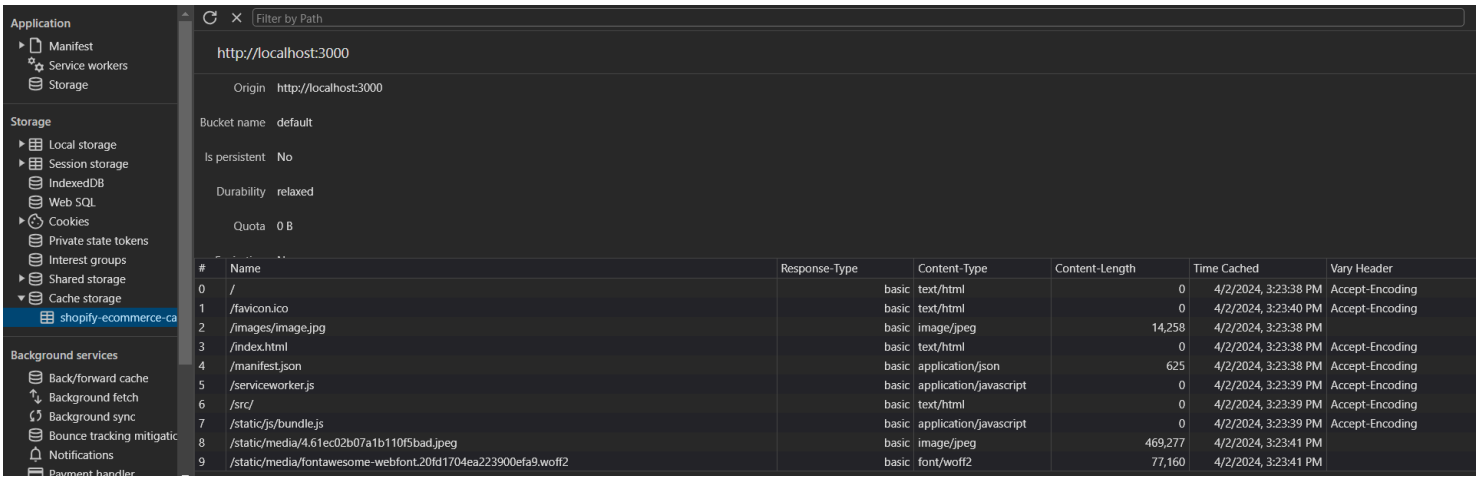
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <script src="serviceworker.js"></script>
    <link rel="manifest" href="manifest.json" />

    <meta charset="utf-8" />
    <meta
      name="viewport"
      content="width=device-width,
               initial-scale=1"
    />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <meta name="theme-color" content="#4285f4" />

    <!-- Title -->
    <title>PWA Tutorial</title>
    <link rel="apple-touch-icon" href="" />
```

```
    <!-- Meta Tags required for
      Progressive Web App -->
    <meta name="apple-mobile-web-app-status-bar" content="#aa7700" />
    <meta name="theme-color" content="black" />

    <!-- Manifest File link -->
    <link rel="manifest" href="manifest.json" />

    <title>React App</title>
  </head>

  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <script>
      window.addEventListener("load", () => {
        registerSW();
      });
      // Register the Service Worker
      async function registerSW() {
        if ("serviceWorker" in navigator) {
          try {
            const registration = await navigator.serviceWorker.register(
              "serviceworker.js"
            );
            console.log("Registered Service Worker:", registration);
          } catch (error) {
            console.log("SW Registration Failed:", error);
          }
        }
      }
    </script>
  </body>
</html>

serviceworker.js
var staticCacheName = "pwa";
self.addEventListener("install", function (e) {
e.waitUntil(
caches.open(staticCacheName).then(function (cache) {
return cache.addAll(["/"]);
})
);
});
self.addEventListener('activate', event => {
event.waitUntil(
caches.keys().then(cacheNames => {
return Promise.all(
```

```
cacheNames.filter(name => {
return name !== staticCacheName;
}).map(name => {
return caches.delete(name);
})
);
})
);
});
self.addEventListener("fetch", function (event) {
console.log(event.request.url);
event.respondWith(
caches.match(event.request).then(function (response) {
return response || fetch(event.request);
})
);
});
```

Output:





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

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	09
Experiment Title.	To implement Service worker events like fetch, sync and push for E-commerce PWA
Roll No.	64
Name	Shashwat Tripathi
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment 9

SHASHWAT TRIPATHI
D15A ROLLNO: 64
BATCH C

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

Theory:-

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

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

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

information messages to the page.

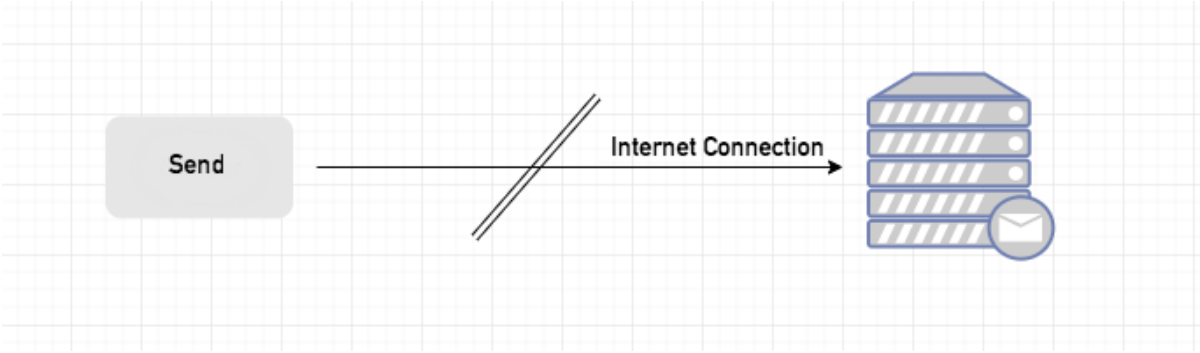
```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);

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

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

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

Sync Event

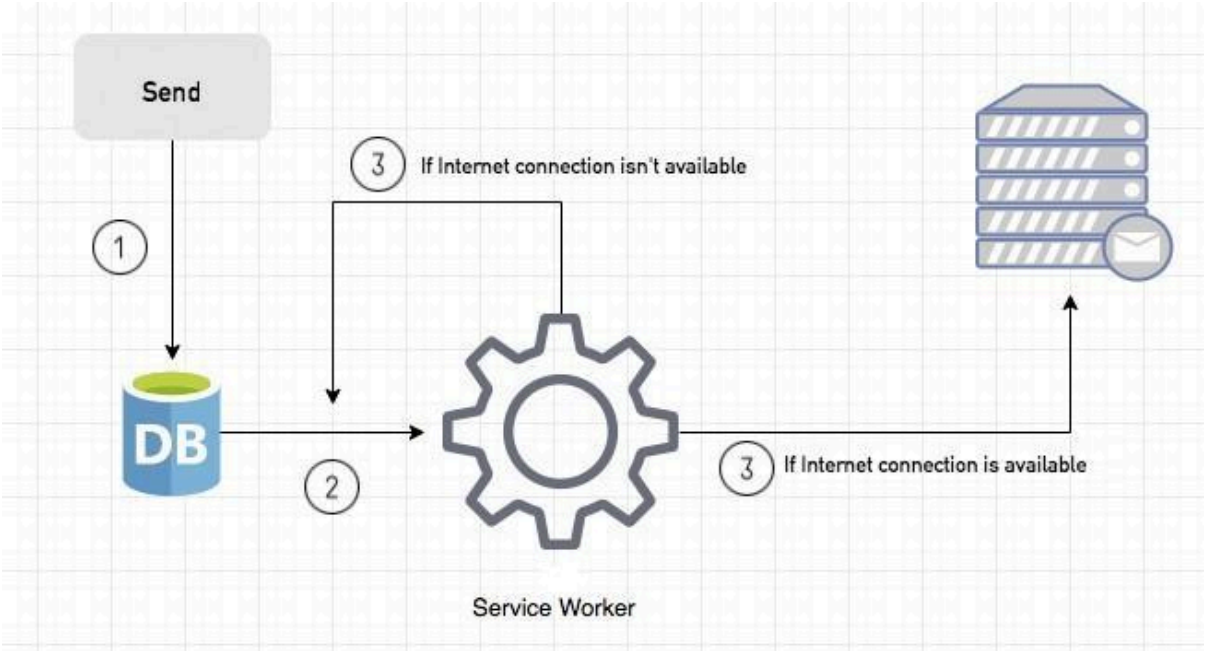
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are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

Here is a job for the Background Sync.

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



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



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

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

Event Listener for Background Sync Registration

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

Event Listener for sw.js

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

Push Event

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

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

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

Code:

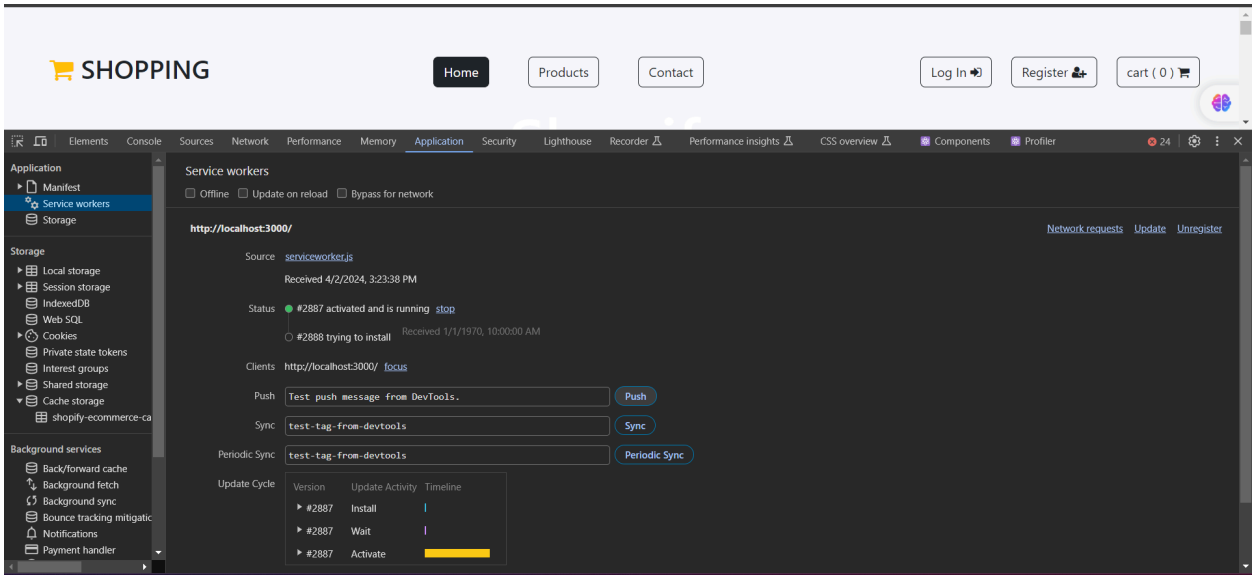
index.html

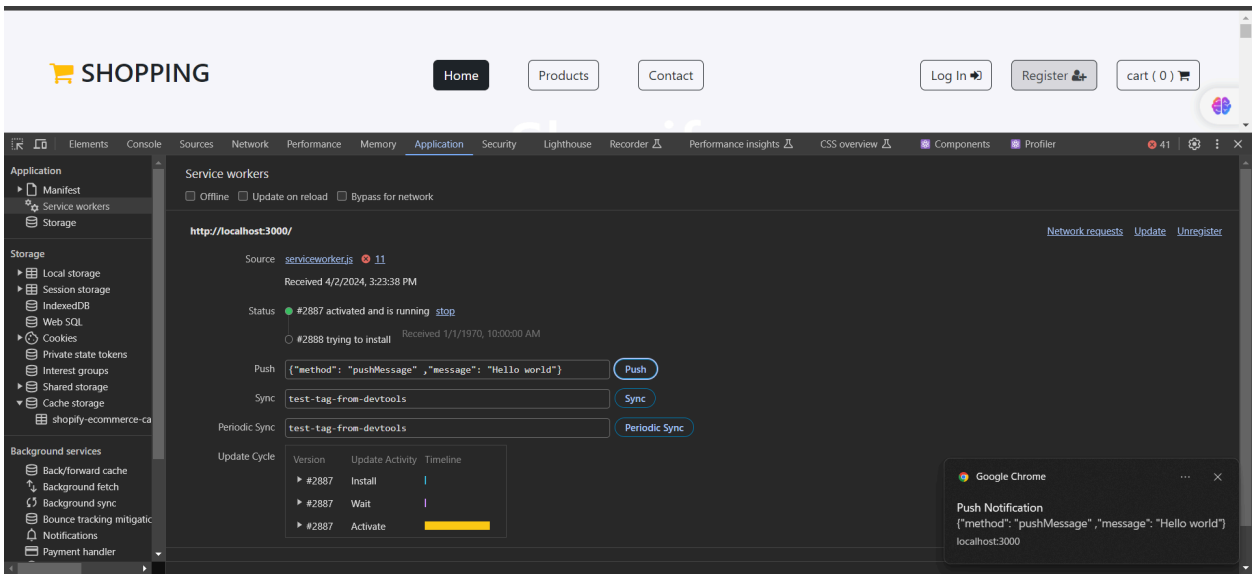
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <script src="serviceworker.js"></script>
    <link rel="manifest" href="manifest.json" />

    <meta charset="utf-8" />
    <meta
      name="viewport"
      content="width=device-width,
```

```
            initial-scale=1"
    />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <meta name="theme-color" content="#4285f4" />

    <!-- Title -->
    <title>PWA Tutorial</title>
    <link rel="apple-touch-icon" href="" />
    <!-- Meta Tags required for
      Progressive Web App -->
    <meta name="apple-mobile-web-app-status-bar" content="#aa7700" />
    <meta name="theme-color" content="black" />

    <!-- Manifest File link -->
    <link rel="manifest" href="manifest.json" />

    <title>React App</title>
  </head>

  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <script>
      window.addEventListener("load", () => {
        registerSW();
      });
      // Register the Service Worker
      async function registerSW() {
        if ("serviceWorker" in navigator) {
          try {
            const registration = await navigator.serviceWorker.register(
              "serviceworker.js"
            );
            console.log("Registered Service Worker:", registration);
          } catch (error) {
            console.log("SW Registration Failed:", error);
          }
        }
      }
    </script>
  </body>
</html>

serviceworker.js
var staticCacheName = "pwa";
self.addEventListener("install", function (e) {
```

```
e.waitUntil(
caches.open(staticCacheName).then(function (cache) {
return cache.addAll(["/"]);
})
);
});
self.addEventListener('activate', event => {
event.waitUntil(
caches.keys().then(cacheNames => {
return Promise.all(
cacheNames.filter(name => {
return name !== staticCacheName;
}).map(name => {
return caches.delete(name);
})
);
})
);
});
self.addEventListener("fetch", function (event) {
console.log(event.request.url);
event.respondWith(
caches.match(event.request).then(function (response) {
return response || fetch(event.request);
})
);
});
```

Output:





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

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	64
Name	Shashwat Tripathi
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment 10

Name: Shashwat Tripathi
Div: D15A
Roll no: 64

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

Deployed link: https://piyush-tilokani.github.io/PWA-Ecommerce/

Theory:

GitHub Pages

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

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

Pros

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

Cons

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

Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

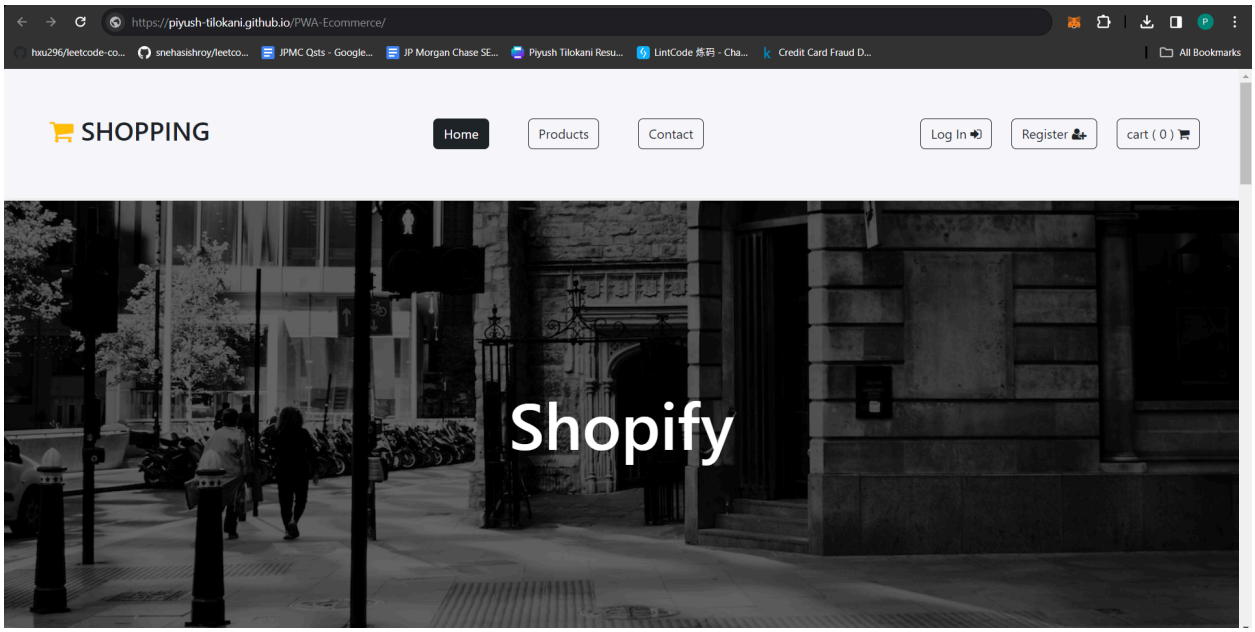
Instacart, 9GAG, and Twitch are some of the popular companies that use Firebase
Firebase has a broader approval, being mentioned in 1215 company stacks & 4651 developers stacks

Pros

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

Cons

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



Conclusion: Thus, we have deployed our app on github pages

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	64
Name	Shashwat Tripathi
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

Experiment 11

Name: Shashwat Tripathi
Div: D15A
Roll no: 64

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

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

Google Lighthouse :

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

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

Key Features and Audit Metrics

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

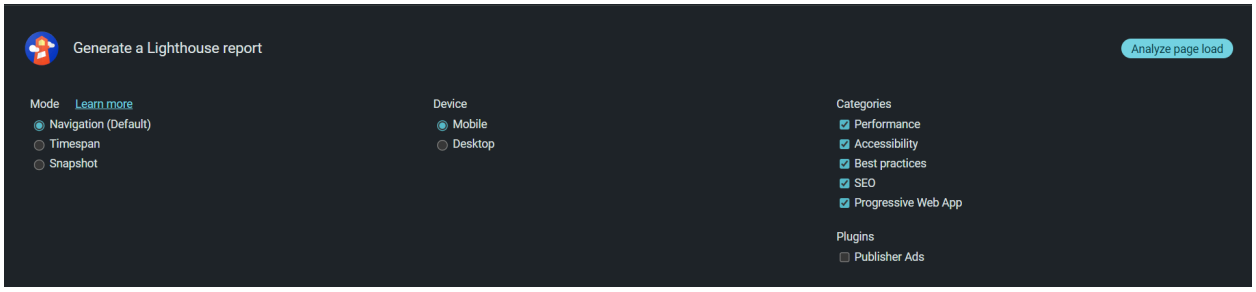
1. **Performance:** This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th percentile and so on.
2. **PWA Score (Mobile):** Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards the PWA paradigm, where the objective is to make the application behave as close as possible to native mobile applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled environments, etc.

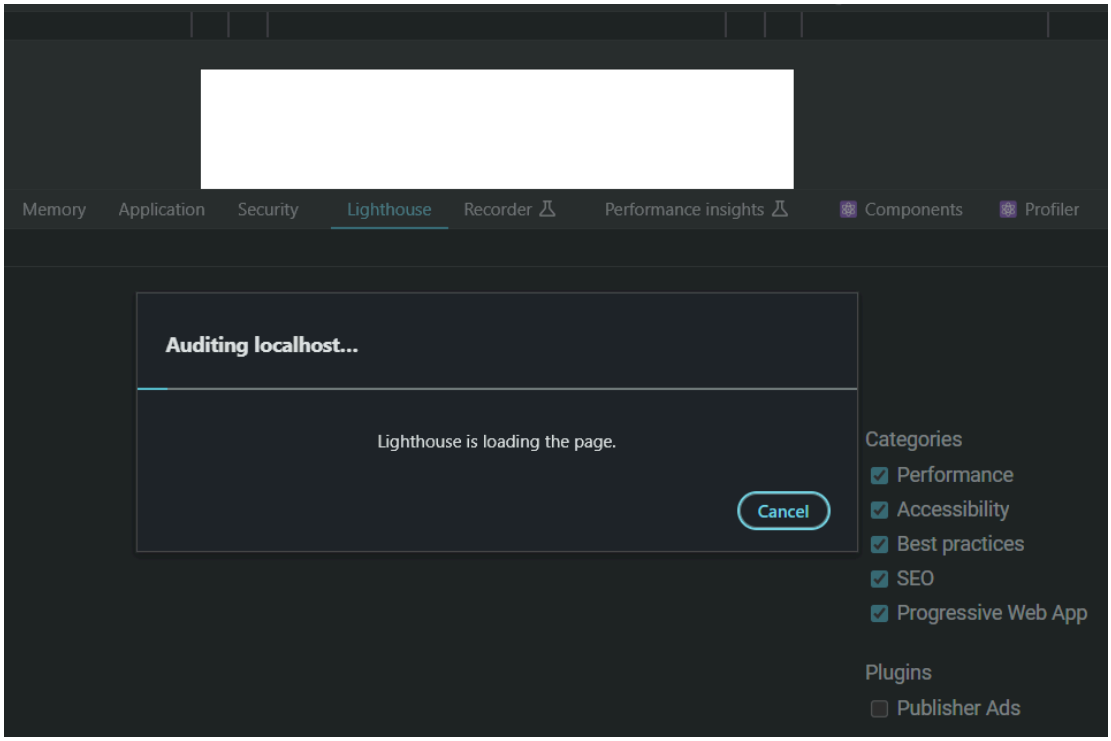
3. **Accessibility:** As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the ‘aria-’ attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics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

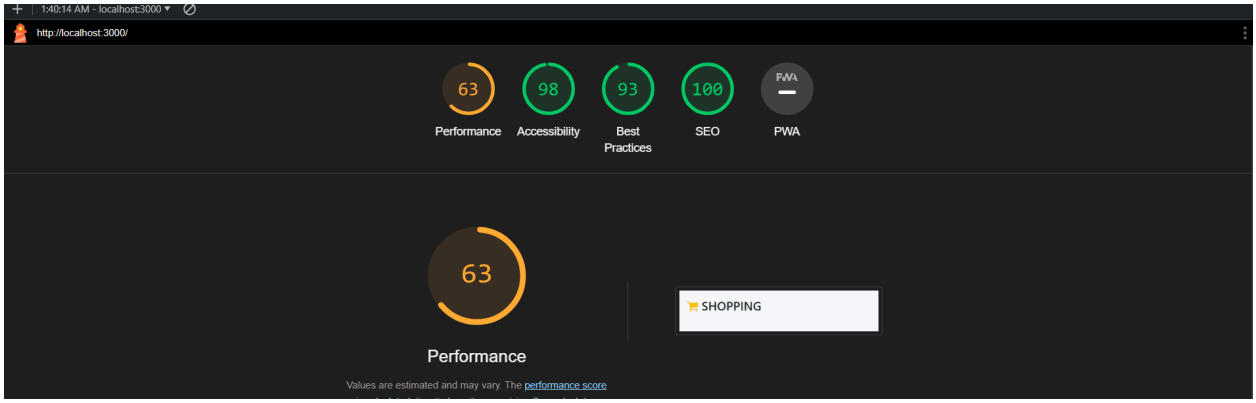
4. **Best Practices:** As any developer would know, there are a number of practices that have been deemed ‘best’ based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS

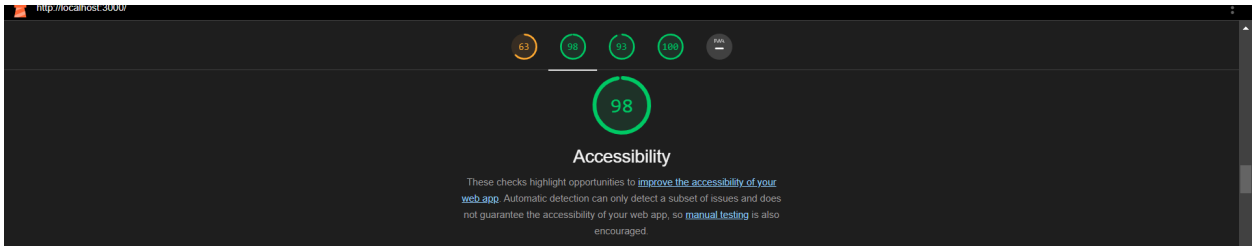
Avoiding the use of deprecated code elements like tags, directives, libraries, etc. Password input with paste-into disabled

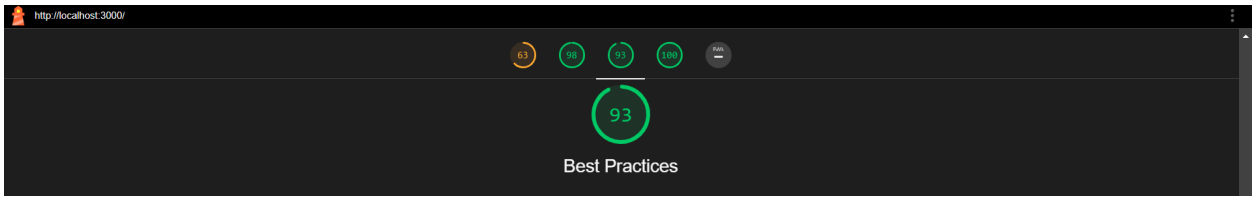
Geo-Location and cookie usage alerts on load, etc.

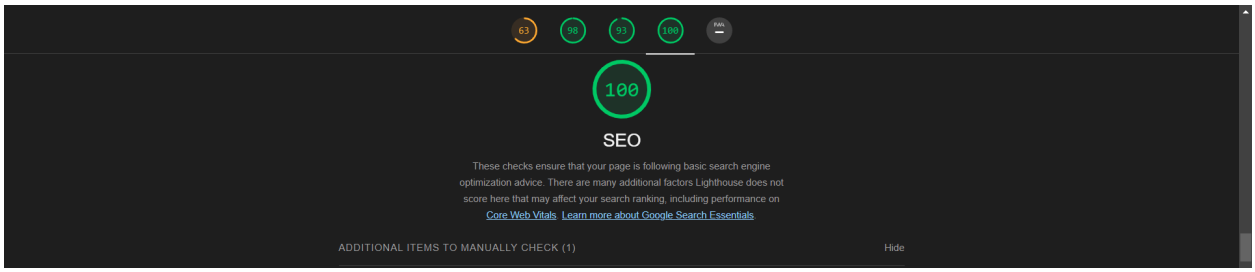


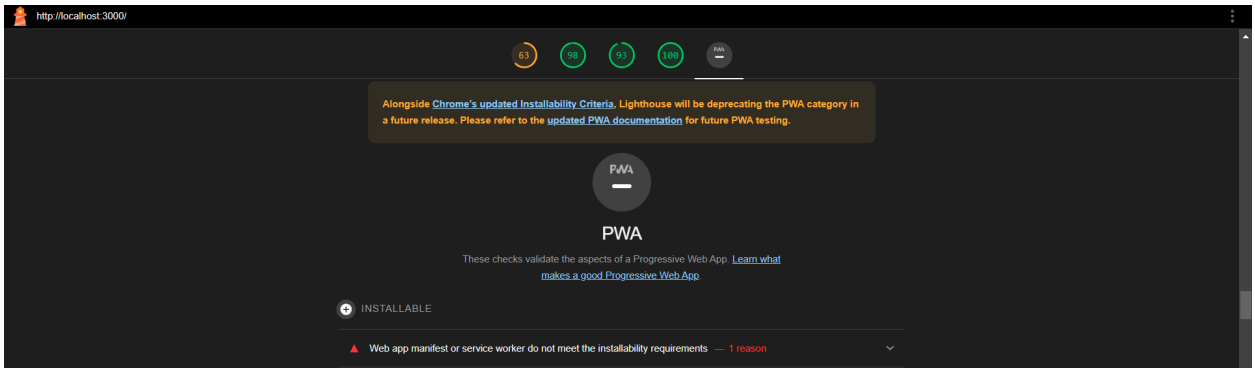

Memory
Application
Security
Lighthouse
Recorder
Performance insights
Components
Profiler
Auditing localhost...
Lighthouse is loading the page.
Cancel
Categories
Performance
Accessibility
Best practices
SEO
Progressive Web App
Plugins
Publisher Ads


1:40:14 AM - localhost:3000
http://localhost:3000/
63
Performance
98
Accessibility
93
Best Practices
100
SEO
PWA
PWA
63
Performance
Values are estimated and may vary. The performance score
SHOPPING


http://localhost:3000/
98
Accessibility
These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.


http://localhost:3000/
93
Best Practices

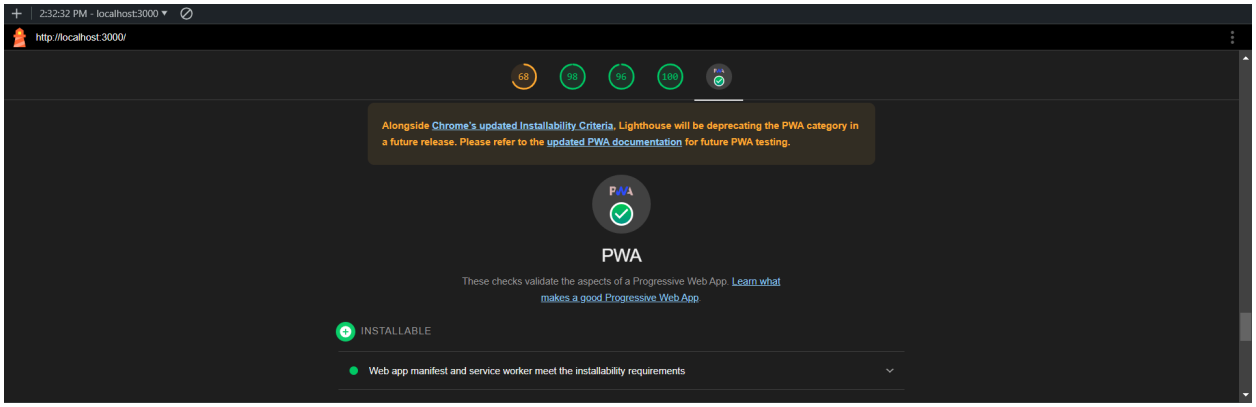




Changes made to manifest.json

```
{
        "name":"Shopify e-commerce",
        "short_name":"Shopify e-commerce",
        "start_url":"index.html",
        "display":"standalone",
        "background_color":"#5900b3",
        "theme_color":"black",
        "scope": "./",
        "description":"Shopify e-commerce.",
        "icons":[
        {
        "src":"images/image.jpg",
        "sizes":"192x192",
        "type":"image/png",
        "purpose": "maskable"
        },
        {
        "src":"images/image.jpg",
        "sizes":"512x512",
        "type":"image/png"
  ,"purposes": "any maskable"
        },
  {
     "src": "images/image.jpg",
     "type": "image/png",
     "sizes":"144x144",
     "purpose": "any maskable"
```

```
    }
]
}
```



Conclusion: Therefore, created my first Progressive Web App by adding metadata to the index.html of my e-commerce web app and installed the Application on my local machine.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	64
Name	Shashwat Tripathi
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Shashwat Tripathi
D15A Batch C
Roll No: 64

Flutter Assignment 1

Q1. 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.

Ans: 1. • The key features of flutter are:

1) Single codebase for multiple platforms: Flutter allows developers to write code and deploy it on both Android & IOs platforms.

2) Hot Reload: Developers can instantly see the results of code changes.

3) Expressive UI: Developers have the flexibility to create expressive & flexible UIs.

4) Integration with other tools: Flutter can easily integrate with other popular development tools and frameworks.

8 5

• Advantages of Flutter:

1) Faster Development:
Uses single codebase for multiple platforms.

2) Consistent UI across platforms:
Widgets provide a consistent look and feel across different platforms.

3) Cost Efficiency

Developing and maintaining single codebase for both Android and iOS reduces development cost & resources.

- Differ from Traditional Approach.

1) Traditional approach uses a hierarchical structure for UI components, whereas flutter uses a widget-based approach.
2) Flutter compiles to native ARM code, providing performance comparable to native applications.
3) Hot Reload allows to see changes made instantly.

Flutters popularity is driven by increased productivity, a growing community, flexibility in UI design, cross-platform development capabilities and adoption by major companies.

Q2. Widget Tree and Composition: Describe the concept of widget tree in flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets & their roles in creating a widget tree.

Ans:2. •) Widget Tree

- The widget tree is a hierarchical structure of widge that defines the user interface of an application.
- Every visual element, from simple components to complex layouts, is represented by a widget.

- Widget can be categorized into 2 types.

ⓐ Stateless Widget

It is immutable and cannot change over time.

Eg: images, text.

ⓑ Stateful Widget

Widget that can change its state over time

Eg: buttons, forms.

•) Widget Composition

- Widget composition in flutter involves combining multiple simple widgets to create more complex and compound widgets.
- This ~~como~~ composability is a powerful concept that allows developers to build sophisticated user interfaces by nesting widgets within each other.

•) Commonly Used Widgets.

a) Container

- A box model for padding, margin & decoration.

b) Column & Row

- Layout widgets for arranging children vertically or horizontally.

c) Stack

- Overlapping widgets, allowing them to be layered on top of each other.

d) List View
- A scrallable list of widgets.

e) Grid View
- A scrollable grid of widgets.

f) App Bar
- A material design app bar typically at the top of screen.

g) Textfield
- An input field for users to enter text.

h) Button Widgets
- Interactive buttons for user actions.

Q3. State Management in Flutter : Discuss the importance of state management in flutter apps. compare & contrast the different state management approaches available in ~~ft~~ flutter, such as setState, Provider and Riverpod. Provide ~~sen~~ scenarios where each approach is suitable.

Ans: State management is crucial in flutter apps as it governs how data changes and propagates throughout the UI. Efficient state management ensures responsive and maintainable apps.

1. setState
- Suitable for small to medium-sized projects with limited interactivity.
- Ideal for simple applications where the state is

local to widget.
- Can lead to code duplication and difficulty in managing state across multiple widgets.

2. Provider
- Suitable for medium to large-sized projects with complex state structures.
- Excellent for managing global application state.
- Reduces boilerplate code and offers a clean way to provide and consume data.
- Great for dependency injection, making it easier to test and maintain code.

3. Riverpod
- An evolution of Provider, offering improvements in terms of scope and lifecycle management.
- Ideal for projects where so scalability and testability are priorities.
- Provides a more robust and flexible solution compared to Provider.
- Allows for better organization of code with the ability to define providers in a separate file.

- SCENARIOS
- setState: Simple apps with few interactive elements like a basic calculator.

- Provider : Medium to large projects with complex state where a centralized state management solution is beneficial, such as e-commerce apps.
- Riverpod : Projects demanding scalability and testability, especially when dealing with large and dynamic data, like social media application.

Q4. Firebase Integration in Flutter : Explain the process of integrating Firebase with a Flutter applic^{n}. 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.

Ans 4. • Process of integrating Firebase

1) Setup a Firebase project
- Create a new project firebase console. Configure your project settings.

2) Add Firebase to Flutter.
- Add necessary firebase dependencies using "pubspec.yaml" file.

3) Initialize Firebase:
- Initialize by calling 'Firebase.initializeApp()' in the main.dart file.

4) Configure Firebase Services:
- Setup services like Authentication, Firestore, Realtime

Database, Cloud Storage, etc. based on needs of App.
5) Implement Firebase Services.
- Use Firebase SDKs in your Dart code to interact with the configured services.

- Benefits
1) Realtime Data Sync.
2) Authentication.
3) Scalability
4) Serverless
5) Analytics & performance monitoring.

- Commonly Used Firebase Service.
1) Firebase ~~Authentia~~ Authentication
2) Cloud Firestore
3) Realtime Database.
4) Cloud Functions
5) Cloud Storage

- Data Synchronization in Firebase.
- Firebase databases like Firestore provide real-time synchronization out of the box.

- When data changes on the server, clients are notified, ensuring a seamless update of the app's UI.

- This real-time sync simplifies implementing features like live chat, collaborative editing, and dynamic content updates in Flutter applications.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	Assignment-2
Assignment 2 Questions	1. Define Progressive Web App (PWA) and explain its significance in modern web development.Discuss the key characteristics that differentiate PWAs from traditional mobile apps 2. Define responsive web design and explain its importance in the context of Progressive Web Apps.Compare and contrast responsive, fluid, and adaptive web design approaches. 3. Describe the lifecycle of Service Workers, including registration, installation, and activation phases. 4. Explain the use of IndexedDB in the Service Worker for data storage.
Roll No.	64
Name	Shashwat Tripathi
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:	

Shashwat Tripathi
DISA
Roll No: 64

CLASSMATE
Date :
Page :

PWA Assignment 2

Q1) What is Progressive Web App (PWA) and explain its significance in modern web development. Discuss the key characteristics that differentiate PWAs from traditional mobile apps.

Ans1)
- A progressive web app (PWA) is a type of web application that uses modern web technologies to provide users with a native app-like experience directly through their web browser. PWAs are designed to be reliable, fast and engaging, offering features such as offline access, push notifications and the ability to install the app.
- Significance of PWAs in modern web development lies in their ability to bridge the gap between web and native mobile applications.

- Differentiating factors:
  1) Cross platform compatibility: PWAs are built using web technologies like HTML, CSS, JS which makes them compatible with platforms like desktops, smartphones, tablets.
  2) Responsiveness: PWAs are designed to adapt seemlessly to different screen sizes.
  3) Offline functionality: PWAs can cache resources and content, enabling users to access the app even when they're offline or have poor internet connection.

4) Progressive enhancement: They can deliver basic functionality to all users while providing enhanced features to modern devices.
5) Installability: PWAs can be installed on a user's device directly on the browser.
6) Discoverability: Are discoverable through search engines, social media & links.
7) Push notifications: Can send push notifications to users, increasing engagement and enabling real-time communication.

Q2) Define responsive web design and explain its importance in the context of PWA. Compare & contrast responsive, fluid and adaptive web design approaches.

Ans: 2)
- Responsive web design is an approach to wed dev aimed at creating websites that provide an optimal viewing and interaction experience across a wide range of devices & screen sizes.
- In context of PWA, responsive design is crucial for ensuring that the app functions and looks good on various devices.

- Responsive Web Design
  - Uses flexible layouts and fluid grids to adapt the ~~lay~~ layout and content of a website to different screen sizes.
  - Achieves responsiveness through css media queries that adjust styles based on screen

CLASSMATE
Date :
Page :

width, height and orientation.

- Fluid web design
  - Similar to responsive design, fluid design uses flexible layouts and fluid grids to adapt to different screen sizes.
  - Focuses more on fluidity and flexibility.

- Adaptive Web design
  - Involves creating multiple layouts or designs tailored to specific screen sizes or device categories.
  - User-server side techniques to detect the user's device and deliver the appropriate layout or design.

Q3) Describe the lifecycle of Service Workers, including registrations, installation and activation phases.

Ans3) 1. Registration
- Occurs in the main script of web application, the main js file.
- Script is fetched from the server & registered with the browser using navigator.serviceWorker.register().

2. Installation
- The browser fires the 'install' event, allowing the Service Worker to cache static assets

and other resources needed to run the web application offline.
- The service worker can use the ~~event.w~~ event.waitUntil() method to extend the installation process.

3. Activation
- Once the Service Worker is successfully installed, browser fires the 'activate' event.
- The activate event is also an opportunity to take control of client pages & intercept network request using event listeners such as 'fetch' and 'message'.

Qu) Explain the use of IndexedDB in the Service Worker for data storage.

Ans4) • IndexedDB is a low-level API provided by modern web browsers for client-side storage of large amounts of structured data, including Service Workers to store & retrieve data in a structured manner.

• Persistent Storage: IndexedDB provides persistent storage, meaning the data stored in the database persists even after the web page or the service worker is closed or refreshed.

2. Asynchronous ~~AA~~ API : IndexedDB operations are asynchronous which means they don't block the main thread of the web app.

3. Structured Data Storage : IndexedDB storage data in a structured manner, similar to a NoSQL database.

4. Large Data Handling : IndexedDB is capable of handling large amounts of data efficiently, making it suitable for applications that require storage of substantial datasets.

5. Transaction-based : IndexedDB operations are performed within transactions, ensuring data integrity and consistency.

- In context of service Worker, IndexedDB can be used for various purposes.

1) Caching : Storing resources such as HTML, CSS, JS files, images and other assets for offline access.

2) Data Persistence : Storing application data, such as user preferences, settings or application state to provide a seamless user experience across sessions.

3) Background Synchronization: Storing data received from application background synchronization tasks such as periodic updates or push notifications without interrupting the user's interaction with the web application.