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.

Since 1962

Department of Information Technology

CERTIFICATE

This is to certify that **Atharva Santosh Mulam** of **D15A** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Name of the Course : MAD & PWA Lab

Course Code : ITL604

Project Title: Flutter : Instagram clone/ PWA: Ecommerce website **Roll No. 37**

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

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers **:** Mrs. Kajal Jewani.

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

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: Flutter : Instagram clone/ PWA: Ecommerce website **Roll No. 37**

Lab Objectives:

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

Lab Outcomes:

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

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1	17/1/24		
2.	To design Flutter UI by including common widgets.	LO2	24/1/24		
3.	To include icons, images, fonts in Flutter app	LO2	31/1/24		
4.	To create an interactive Form using form widget	LO2	7/2/24		
5.	To apply navigation, routing and gestures in Flutter App	LO2	14/2/24		
6.	To Connect Flutter UI with fireBase database	LO3	21/2/24		
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4	6/3/24		
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	13/3/24		
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	20/3/24		
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	27/3/23		
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6			
12.	Assignment-1	LO1,LO2 ,LO3			
13.	Assignment-2	LO4,LO5 ,LO6			

MAD & PWA Lab Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	37
Name	Atharva Santosh Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	15

Atharva Mulam Batch B
D15A Roll No: 37

Exp No: 1

Aim: To install and configure flutter environment.

Prerequisites:

docs.flutter.dev/get-started/install/windows/mobile?tab=download
Flutter
Multi-Platform
Development
Ecosystem
Showcase
Docs
Get started
Install the Flutter SDK
To install the Flutter SDK, you can use the VS Code Flutter extension or download and install the Flutter bundle yourself.
Use VS Code to install
Download and install
Download then install Flutter
1. Download the following installation bundle to get the latest stable release of the Flutter SDK.
flutter_windows_3.16.5-stable.zip
For other release channels, and older builds, check out the SDK archive.
This guide presumes that you downloaded your Flutter SDK to the default download directory for Windows: %CSIDL_DEFAULT_DOWNLOADS%\.
2. Create a folder where you can install Flutter.
Consider %USERPROFILE% or C:\dev.
Get started
Install Flutter
Test drive
Write your first app
Learn more
From another platform?
Dart language overview
Stay up to date
Samples & codelabs
App solutions
User interface
Introduction
Widget catalog
Layout
Contents
System requirements
Hardware requirements
Software requirements
Configure a text editor or IDE
Install the Flutter SDK
Configure Android development
Configure the Android toolchain in Android Studio
Configure your target Android device
Agree to Android licenses
Check your

Administrator: Command Prompt - flutter
Microsoft Windows [Version 10.0.22000.2652]
(c) Microsoft Corporation. All rights reserved.
C:\Users\INFT505-11>flutter
Manage your Flutter app development.
Common commands:
flutter create <output directory>
Create a new Flutter project in the specified directory.
flutter run [options]
Run your Flutter application on an attached device or in an emulator.
Usage: flutter <command> [arguments]
Global options:
-h, --help Print this usage information.
-v, --verbose Noisy logging, including all shell commands executed.
If used with "--help", shows hidden options. If used with "flutter doctor", shows additional diagnostic informati
on. (Use "-vv" to force verbose logging in those cases.)
-d, --device-id Target device id or name (prefixes allowed).
--version Reports the version of this tool.
--enable-analytics Enable telemetry reporting each time a flutter or dart command runs.
--disable-analytics Disable telemetry reporting each time a flutter or dart command runs, until it is
re-enabled.
--suppress-analytics Suppress analytics reporting for the current CLI invocation.
Available commands:
Flutter SDK
bash-completion Output command line shell completion setup scripts.
channel List or switch Flutter channels.
config Configure Flutter settings.
doctor Show information about the installed tooling.
downgrade Downgrade Flutter to the last active version for the current channel.
precache Populate the Flutter tool's cache of binary artifacts.
upgrade Upgrade your copy of Flutter.
Project
analyze Analyze the project's Dart code.
assemble Assemble and build Flutter resources.
build Build an executable app or install bundle.

```
C:\Users\INFT505-11>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.16.5, on Microsoft Windows [Version 10.0.22000.2652], locale en-US)
[√] Windows Version (Installed version of Windows is version 10 or higher)
[!] Android toolchain - develop for Android devices (Android SDK version 33.0.1)
    ! Some Android licenses not accepted. To resolve this, run: flutter doctor --android-licenses
[√] Chrome - develop for the web
[X] Visual Studio - develop Windows apps
    X Visual Studio not installed; this is necessary to develop Windows apps.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++" workload, including all of its default components
[√] Android Studio (version 2021.3)
[√] VS Code (version 1.76.0)
[√] Connected device (3 available)
[√] Network resources

! Doctor found issues in 2 categories.

C:\Users\INFT505-11>
```

Code:

```
import 'package:flutter/material.dart';
void main() {
 runApp(const MyApp());
}
class MyApp extends StatelessWidget {
 const MyApp({Key? key}) : super(key: key);
 @override
 Widget build(BuildContext context) {
 return MaterialApp(
 title: 'Welcome to Flutter',
 home: Scaffold(
 appBar: AppBar(
 title: const Text('Welcome to Flutter'),
 ),
 body: const Center(
 child: Text('Hello Atharva Mulam!'),
 ),
 ),
 );
 }
}
```

OUTPUT:

MAD & PWA Lab Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	37
Name	Atharva Santosh Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Atharva Mulam Batch B
D15A Roll No: 37

MAD&PWA LAB2

Screenshots	Features
	1) Interactive form to create a new account 2) Navigation between pages using buttons 3) The components use are TextField for input, Icons & Containers, TextButton for functionality

	1) Interactive form to log in to existing account 2) Navigation between pages using buttons 3) A network image has been used to show the logo, along with TextFields in rows and TextButtons for functionality and are validated.
	1) Multiple widgets like home page, search, add post,profile,etc. 2) Posts in home feed which are created using widgets. 3) The Top App Bar is created using Icons, and the bottom bar is used for navigation using the Navigator.

	1) View the uploaded post of a user 2) Navigation between different. pages using buttons on navbar. 3) See the likes and do comments using the rows,text fields,etc. 4) Photos get stored in Firebase on posting and can be dynamically fetched as well.
	1) User profile page where posts,followers,etc. can be viewed. 2) All the details eg-Photos,Followers,etc are fetched from Firebase according to a particular user and are the displayed in the profile page.

	1) Uploading new posts where the widgets and TextButton with source as gallery are used to choose photos and videos from gallery. 2) Photos are stored in firebase and are dynamically displayed in the home feed using the post widget which has its layout defined.

MAD & PWA Lab Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	37
Name	Atharva Santosh Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Atharva Mulam Batch B
D15A Roll No: 37

Exp No: 3

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

Theory:

1) **NetworkImage:**
   Purpose: The NetworkImage widget is specifically designed to load and display images from URLs on the internet.
   Usage: You provide the URL of the image you want to display as a string parameter to the NetworkImage constructor.
   Loading: It handles the process of fetching the image from the network asynchronously, which means it won't block the UI thread while waiting for the image to download.
   Caching: Flutter's image caching mechanism helps improve performance by caching images, reducing unnecessary network requests.

2) **Icon:**
   Purpose: The Icon widget is used to display vector icons in Flutter apps. It's commonly used to represent actions, buttons, or other UI elements.
   IconData: Icons are represented by IconData objects, each of which uniquely identifies an icon. Flutter provides a set of built-in icons through the Icons class.
   Customization: Icons can be customized using the color, size, and semanticLabel properties. Additionally, you can customize the appearance of icons using the IconTheme widget.

3) **Image**
   Purpose: The Image widget is a versatile tool for displaying images in Flutter apps, supporting various image sources such as assets, files, memory, and network URLs.

Source Types: Depending on the source of the image, you can use different constructors such as AssetImage, FileImage, MemoryImage, and NetworkImage.

Performance: Flutter optimizes image loading and rendering for better performance. It also provides features like image caching and image format decoding to improve efficiency.

Code:

```
import 'package:flutter/material.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';
import 'package:ig/core/constants/app_colors.dart';
import 'package:ig/core/constants/constants.dart';
import 'package:ig/core/widgets/round_icon_button.dart';

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

  static const routeName = '/home';
  @override
  State<HomeScreen> createState() => _HomeScreenState();
}
class _HomeScreenState extends State<HomeScreen> with
TickerProviderStateMixin {
  late final TabController _tabController;

  @override
  void initState() {
    _tabController = TabController(length: 5, vsync: this);
    super.initState();
  }
  @override
  void dispose() {
    _tabController.dispose();
    super.dispose();
  }
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: AppColors.greyColor,
    appBar: AppBar(
      backgroundColor: AppColors.whiteColor,
      elevation: 0,
      title: _buildFacebookText(),
      actions: [
        _buildSearchWidget(),
        _buildMessengerWidget(),
      ],
    ),
    body: TabBarView(
      controller: _tabController,
      children: Constants.screens,
    ),
    bottomNavigationBar: Material(
      color: AppColors.whiteColor,
      child: TabBar(
        tabs: Constants.getHomeScreenTabs(_tabController.index),
        controller: _tabController,
        onTap: (index) {
          setState(() {});
        },
      ),
    ),
  );
}
Widget _buildFacebookText() => const Text(
      'Instagram',
      style: TextStyle(
        color: AppColors.blackColor,
        fontSize: 30,
        fontWeight: FontWeight.bold,
```

```
      ),
    );
  Widget _buildSearchWidget() => const RoundIconButton(
      icon: FontAwesomeIcons.heart,
    );
  Widget _buildMessengerWidget() => InkWell(
      onTap: () {},
      child: const RoundIconButton(
        icon: FontAwesomeIcons.facebookMessenger,
      ),
    );
}
```

Output:

MAD & PWA Lab Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	37
Name	Atharva Santosh Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Atharva Mulam Batch B
D15A Roll No: 37

Exp No: 4

Aim: To create an interactive Form using form widget.

Theory:
In Flutter, forms are used to collect user input and interact with users. Flutter provides a set of widgets and classes to create forms efficiently.

1) **Form Widget:**

The Form widget is a container for form fields and manages the form's state.
It's typically used as the root widget of a form and helps with validation and submission.
The Form widget doesn't display anything by itself but provides utilities to interact with its child form fields.

2) **FormField Widget:**

The FormField widget represents a single form field within a Form.
It's a generic class that's extended by various field-specific widgets like TextFormField, DropdownButtonFormField, CheckboxFormField, etc.
FormField widgets handle user input, validation, and error messages associated with the field.

3) **TextFormField:**

TextFormField is a commonly used form field widget for collecting text input from users.
It provides features like keyboard input, text editing, validation, error handling, and input formatting.
You can customize its appearance, input type, validator, controller, and more.

4) **Form Validation:**

Flutter provides built-in support for form validation, which ensures that user input meets specific criteria before submission.
You can define validation logic using validators like required, minLength, maxLength, email, numeric, etc., or create custom validators.

Form validation is typically performed within the validator parameter of form field widgets or by implementing the FormFieldValidator function.

5) **Form Submission:**

After validating user input, you can handle form submission using callbacks like onSaved or onFieldSubmitted provided by form field widgets.
Alternatively, you can use the FormState object to access and process the form data when the form is submitted.
Form submission involves processing the input data, performing additional actions, and updating the UI accordingly.

6) **GlobalKey<FormState>:**

To interact with a Form and its state, you typically use a GlobalKey<FormState> object.
The FormState object contains methods to validate, reset, and save form fields.
It's essential for accessing and manipulating the state of the form, especially when performing actions like validation and submission.

Code:

```
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:ig/core/constants/constants.dart';
import 'package:ig/core/widgets/round_button.dart';
import 'package:ig/core/widgets/round_text_field.dart';
import 'package:ig/features/auth/presentation/screens/create_account_screen.dart';
import 'package:ig/features/auth/providers/auth_provider.dart';
import 'package:ig/features/auth/utils/utils.dart';

final _formKey = GlobalKey<FormState>();

class LoginScreen extends ConsumerStatefulWidget {
  const LoginScreen({super.key});

  @override
  ConsumerState<LoginScreen> createState() => _LoginScreenState();
  static const String _routeName = '/login';
```

```
  static String get route => _routeName;
}
class _LoginScreenState extends ConsumerState<LoginScreen> {
  late final TextEditingController _emailController;
  late final TextEditingController _passwordController;
  bool isLoading = false;

  Future<void> login() async{
    if(_formKey.currentState!.validate()){
      setState(() => isLoading = true);
      ref.read(authProvider).signIn(email: _emailController.text,password:
_passwordController.text,);
      setState(() => isLoading = false);
    }
  }
  @override
  void initState(){
    _emailController = TextEditingController();
    _passwordController = TextEditingController();
  super.initState();
  }
  @override
  void dispose(){
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: Padding(
        padding: Constants.defaultPadding,
        child: Column(
          mainAxisSize: MainAxisSize.max,
```

```
mainAxisAlignment: MainAxisAlignment.spaceAround,
children: [
  Image.asset('assets/icons/igword.png',
  width: 220,),
  Form(
    key: _formKey,
    child: Column(
      children: [
        RoundTextField(
          controller: _emailController,
          hintText: 'Email',
          keyboardType: TextInputType.emailAddress,
          textInputAction: TextInputAction.next,
          validator: validateEmail,
          ),
          const SizedBox(height: 15,),
          RoundTextField(
          controller: _passwordController,
          hintText: 'Password',
          textInputAction: TextInputAction.next,
          keyboardType: TextInputType.visiblePassword,
          validator: validatePassword,
          isPassword: true,
          ),
          const SizedBox(height: 15,),
          RoundButton(onPressed: login, label: 'Log In'),
          const SizedBox(height: 15,),
          const Text('Forgot password',style: TextStyle(fontSize: 20),),
      ]),
  ),
  Column(
    children: [
        RoundButton(
          onPressed: (){
```

```
Navigator.of(context).pushNamed(CreateAccountScreen.routeName);
                },
                label: 'Create new account',
                color: Colors.transparent,),
                Image.asset('assets/icons/meta-logo.png',height: 80,),
            ],
          )
        ],

      ) ),
    );
  }
}
```

Output:

Login

Sign Up

MAD & PWA Lab Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	37
Name	Atharva Santosh Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Atharva Mulam Batch B
D15A Roll No: 37

Exp No: 5

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

Theory:
In Flutter, navigation, routing, and gestures play crucial roles in creating engaging and intuitive user interfaces. Here's a brief overview of each concept:

Navigation:
Navigation in Flutter refers to the process of moving between different screens or pages within an app. Flutter provides a Navigator class that manages a stack of Route objects, allowing you to push and pop routes onto and off of the navigation stack. There are several ways to navigate between screens in Flutter:
Pushing a new route: You can push a new route onto the navigation stack using the Navigator.push() method.
Popping a route: You can pop the current route off the stack using the Navigator.pop() method.
Named routes: You can define named routes for your app's screens and use them to navigate using the Navigator.pushNamed() method.
Modal routes: You can show a modal route that covers the screen using the showDialog() or showModalBottomSheet() methods.

Routing:
Routing in Flutter refers to the process of defining how the app's screens or pages are structured and organized. Flutter uses a hierarchical routing system, where each route corresponds to a widget subtree that can be pushed and popped onto and off of the navigation stack. Here are some key concepts related to routing in Flutter:
MaterialPageRoute: This is the most commonly used route in Flutter apps, which represents a full-screen page that transitions in and out using a material design-style animation.

PageRouteBuilder: This class allows you to create custom page transition animations by specifying a builder function that returns the widget subtree for the route.
Nested navigation: You can nest Navigator widgets within your app's widget tree to create nested navigation hierarchies, allowing for more complex navigation flows.
Gestures:
Gestures in Flutter refer to user interactions such as tapping, dragging, swiping, pinching, etc. Flutter provides a rich set of gesture recognizer classes that make it easy to handle these interactions. Here are some commonly used gesture recognizers in Flutter:
GestureDetector: This widget allows you to detect various gestures such as taps, drags, and long-presses on its child widget and respond to them with custom callback functions.
InkWell: This widget provides a material design-style ink splash effect in response to taps, and it's commonly used for creating clickable elements in Flutter apps.
Draggable: This widget allows you to make its child widget draggable, enabling users to drag it around the screen using touch gestures.

Code:

```
import 'package:flutter/material.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';
import 'package:ig/core/constants/app_colors.dart';
import 'package:ig/core/constants/constants.dart';
import 'package:ig/core/widgets/round_icon_button.dart';

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

  static const routeName = '/home';

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}
```

```
class _HomeScreenState extends State<HomeScreen> with
TickerProviderStateMixin {
  late final TabController _tabController;

  @override
  void initState() {
    _tabController = TabController(length: 5, vsync: this);
    super.initState();
  }

  @override
  void dispose() {
    _tabController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: AppColors.greyColor,
      appBar: AppBar(
        backgroundColor: AppColors.whiteColor,
        elevation: 0,
        title: _buildFacebookText(),
        actions: [
          _buildSearchWidget(),
          _buildMessengerWidget(),
        ],
      ),
      body: TabBarView(
        controller: _tabController,
        children: Constants.screens,
      ),
      bottomNavigationBar: Material(
        color: AppColors.whiteColor,
```

```
      child: TabBar(
        tabs: Constants.getHomeScreenTabs(_tabController.index),
        controller: _tabController,
        onTap: (index) {
          setState(() {});
        },
      ),
    ),
  );
}

Widget _buildFacebookText() => const Text(
      'Instagram',
      style: TextStyle(
        color: AppColors.blackColor,
        fontSize: 30,
        fontWeight: FontWeight.bold,
      ),
    );

Widget _buildSearchWidget() => const RoundIconButton(
      icon: FontAwesomeIcons.heart,
    );

Widget _buildMessengerWidget() => InkWell(
      onTap: () {},
      child: const RoundIconButton(
        icon: FontAwesomeIcons.facebookMessenger,
      ),
    );
}
```

Routing:

```
import 'package:ig/core/screens/error_screen.dart';
import 'package:flutter/cupertino.dart';
import 'package:ig/core/screens/home_screen.dart';
```

```
import 'package:ig/core/screens/profile_screen.dart';
import 'package:ig/features/auth/presentation/screens/create_account_screen.dart';
import 'package:ig/features/posts/presentation/screens/comments_screen.dart';
import 'package:ig/features/posts/presentation/screens/create_post_screen.dart';

class Routes {
  static Route onGenerateRoute(RouteSettings settings) {
    switch (settings.name) {
      case CreateAccountScreen.routeName:
      return _cupertinoRoute(const CreateAccountScreen(),);
      case HomeScreen.routeName:
      return _cupertinoRoute(const HomeScreen(),);
      case CreatePostScreen.routeName:
      return _cupertinoRoute(const CreatePostScreen(),);
      case CommentsScreen.routeName:
        final postId = settings.arguments as String;
        return _cupertinoRoute(
          CommentsScreen(postId: postId),
        );
       case ProfileScreen.routeName:
        final userId = settings.arguments as String;
        return _cupertinoRoute(
          ProfileScreen(
            userId: userId,
          ),
        );
      default:
        return _cupertinoRoute(
          ErrorScreen(
            error: 'Wrong Route provided ${settings.name}',
          ),
        );
    }
  }
```

```
  static Route _cupertinoRoute(Widget view) => CupertinoPageRoute(
      builder: (_) => view,
    );

  Routes._();
}
```

Output:

MAD & PWA Lab Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	37
Name	Atharva Santosh Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	15

Atharva Mulam Batch B
D15A Roll No: 37

Exp No: 6

Aim: To Connect Flutter UI with FireBase database.

Theory:
Prerequisites
To complete this tutorial, you will need:

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

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

2)Go to the Firebase Console and create a new project.
Add your Flutter app to the Firebase project:
Register your app in the Firebase project, and follow the instructions to download the configuration files (google-services.json for Android, GoogleService-Info.plist

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

3) Add Firebase to your Flutter project and add the necessary dependencies:

Code:

```
import 'dart:io';
import 'package:ig/core/constants/app_colors.dart';
import 'package:ig/core/constants/constants.dart';
import 'package:ig/core/utils/utils.dart';
import 'package:ig/core/widgets/pick_image_widget.dart';
import 'package:ig/core/widgets/round_button.dart';
import 'package:ig/core/widgets/round_text_field.dart';
import 'package:ig/features/auth/presentation/widgets/birthday_picker.dart';
import 'package:ig/features/auth/presentation/widgets/gender_picker.dart';
import 'package:ig/features/auth/providers/auth_provider.dart';
import 'package:ig/features/auth/utils/utils.dart';
import 'package:flutter/material.dart';
```

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

final _formKey = GlobalKey<FormState>();

class CreateAccountScreen extends ConsumerStatefulWidget {
  const CreateAccountScreen({super.key});

  static const routeName = '/create-account';

  @override
  ConsumerState<CreateAccountScreen> createState() =>
      _CreateAccountScreenState();
}

class _CreateAccountScreenState extends ConsumerState<CreateAccountScreen>
{
  File? image;
  DateTime? birthday;
  String gender = 'male';
  bool isLoading = false;

  // controllers
  late final TextEditingController _fNameController;
  late final TextEditingController _lNameController;
  late final TextEditingController _emailController;
  late final TextEditingController _passwordController;
  @override
  void initState() {
    _fNameController = TextEditingController();
    _lNameController = TextEditingController();
    _emailController = TextEditingController();
    _passwordController = TextEditingController();
    super.initState();
  }
  @override
```

```
void dispose() {
  _fNameController.dispose();
  _lNameController.dispose();
  _emailController.dispose();
  _passwordController.dispose();
  super.dispose();
}
Future<void> createAccount() async {
  if (_formKey.currentState!.validate()) {
    _formKey.currentState!.save();
    setState(() => isLoading = true);
    await ref
        .read(authProvider)
        .createAccount(
          fullName: '${_fNameController.text} ${_lNameController.text}',
          birthday: birthday ?? DateTime.now(),
          gender: gender,
          email: _emailController.text,
          password: _passwordController.text,
          image: image,
        )
        .then((credential) {
      if (!credential!.user!.emailVerified) {
        Navigator.pop(context);
      }
    }).catchError((_) {
      setState(() => isLoading = false);
    });
    setState(() => isLoading = false);
  }
}

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

```
backgroundColor: AppColors.realWhiteColor,
appBar: AppBar(),
body: SingleChildScrollView(
  child: Padding(
    padding: Constants.defaultPadding,
    child: Form(
      key: _formKey,
      child: Column(
        children: [
          GestureDetector(
            onTap: () async {
              image = await pickImage();
              setState(() {});
            },
            child: PickImageWidget(image: image),
          ),
          const SizedBox(height: 20),
          Row(
            children: [
              // First Name Text Field
              Expanded(
                child: RoundTextField(
                  controller: _fNameController,
                  hintText: 'First name',
                  textInputAction: TextInputAction.next,
                  validator: validateName,
                ),
              ),
              const SizedBox(width: 10),
              // Last Name Text Field
              Expanded(
                child: RoundTextField(
                  controller: _lNameController,
                  hintText: 'Last name',
                  textInputAction: TextInputAction.next,
```

```
        validator: validateName,
      ),
    ),
  ],
),
const SizedBox(height: 20),
BirthdayPicker(
  dateTime: birthday ?? DateTime.now(),
  onPressed: () async {
    birthday = await pickSimpleDate(
      context: context,
      date: birthday,
    );
    setState(() {});
  },
),
const SizedBox(height: 20),
GenderPicker(
  gender: gender,
  onChanged: (value) {
    gender = value ?? 'male';
    setState(() {});
  },
),
const SizedBox(height: 20),
// Phone number / email text field
RoundTextField(
  controller: _emailController,
  hintText: 'Email',
  textInputAction: TextInputAction.next,
  keyboardType: TextInputType.emailAddress,
  validator: validateEmail,
),
const SizedBox(height: 20),
// Password Text Field
```

```
            RoundTextField(
              controller: _passwordController,
              hintText: 'Password',
              textInputAction: TextInputAction.done,
              keyboardType: TextInputType.visiblePassword,
              validator: validatePassword,
              isPassword: true,
            ),
            const SizedBox(height: 20),
            isLoading
                ? const Center(child: CircularProgressIndicator())
                : RoundButton(
                    onPressed: createAccount,
                    label: 'Create Account',
                  ),
          ],
        ),
      ),
    ),
  ),
);
}
}
```

Output:

Conclusion: In this experiment, we have successfully connected firebase database and have authenticated login/signup and all the info related to a particular user are successfully stored in our database.

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.	37
Name	Atharva Santosh Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

Atharva Mulam Batch B
D15A Roll No: 37

Exp No: 7

Aim: To write meta data of your Ecommerce PWA

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:

1. Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.
2. Responsive — They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can later become available.
3. App-like — They behave with the user as if they were native apps, in terms of interaction and navigation.
4. Updated — Information is always up-to-date thanks to the data update process offered by service workers.
5. Secure — Exposed over HTTPS protocol to prevent the connection from displaying information or altering the contents.
6. Searchable — They are identified as “applications” and are indexed by search engines.
7. Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
8. 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.
9. Linkable — Easily shared via URL without complex installations.
10. 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:

1. IOS support from version 11.3 onwards

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

Output:

```
<!DOCTYPE html>
<html>
<head>

<meta charset="utf-8">
<meta name="viewport"
        content="width=device-width,
                initial-scale=1">
<meta http-equiv="X-UA-Compatible"
        content="ie=edge">
<title>PWA Tutorial</title>
<meta name=
"apple-mobile-web-app-status-bar"
        content="#aa7700">
<meta name="theme-color"
        content="black">
<link rel="manifest"
        href="manifest.json">
</head>
<body>
<h1 style="color: red;">
```

```
        PWA</h1>
<p>Hello Atharva!</p>

<script>
        window.addEventListener('load', () => {
                registerSW();
        });
        async function registerSW() {
                if ('serviceWorker' in navigator) {
                        try {
                                await navigator.serviceworker.register('serviceworker.js');
                        }
                        catch (e) {
                                console.log('SW registration failed');
                        }
                }
        }
</script>
</body>
</html>
```

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

Open Developer tools options

Go to Applications

Click on 3 dots on top right corner of browser from app option install this site as an app

PWA App icon and running application

Conclusion: Thus, the “add to homescreen” feature was successfully enabled by adding a manifest file to my web app.

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.	37
Name	Atharva Santosh Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Atharva Mulam Batch B
D15A Roll No: 37

Exp No: 8

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

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

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

What can we do with Service Workers?

You can dominate Network Traffic

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

You can Cache

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

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

What can't we do with Service Workers?
You can't access the Window
You can't access the window, therefore, You can't manipulate DOM elements. But, you can communicate to the window through post Message and manage processes that you want.
You can't work it on 80 Port
Service Worker just can work on HTTPS protocol. But you can work on localhost during development.

Service Worker Cycle
A service worker goes through three steps in its life cycle:
Registration
Installation
Activation

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

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

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

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

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

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

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

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

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

main.js

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

Installation

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

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

service-worker.js

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

Activation

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

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

service-worker.js

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

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

Code:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>E-commerce Store</title>
   <link rel="stylesheet" href="styles.css">
   <meta name="apple-mobile-web-app-status-bar" content="#aa7700">
   <meta name="theme-color" content="black">
   <meta name="theme-color" content="#4285f4">
   <link rel="apple-touch-icon" href="">
   <link rel="manifest" href="manifest.json">
   <script src="app.js" defer></script> <!-- Linked app.js file -->
   <style>
      body {
         margin: 0;
         padding: 0;
      }
```

```
    header, footer {
        background-color: grey;
        color: white;
        text-align: center;
        padding: 10px;
    }
    main {
        display: flex;
        flex-direction: column;
        align-items: center;
        justify-content: center;
        padding: 20px;
    }
    #welcome {
        text-align: center;
        background-size: cover;
        background-position: center;
        color: black;
        padding: 50px;
        background-size: 100% auto;
    }
    #products {
        display: flex;
        justify-content: center;
        margin: 0 -10px;
        flex-wrap: wrap;
}
    .product {
        margin: 10px;
        text-align: center;
        flex: 0 0 auto;
        width: 300px;
        height: 300px;
        object-fit: cover;
        margin-left: 50px;
```

```
        }

        .product img {
            max-width: 300px;
             width: auto;
            height: 200px;
            object-fit: cover;
}
    </style>
</head>
<body>
    <header>
        <h1>E-commerce Store</h1>
    </header>
    <main>
        <section id="welcome">
            <h2>Welcome to our vibrant marketplace, where shopping meets delight! Explore our curated collection of must-have items, handpicked just for you. Start your journey with us today and discover endless possibilities in every click!</h2>
        </section>
        <p>Explore our collection below:</p>
        <section id="products">
            <article class="product">
                <img src="./images/product1.jpg" alt="Product 1">
                <h2>Mobile Phones</h2>
                <p>Cutting-edge mobile phones: Explore the latest in mobile technology with our diverse range of smartphones.</p>
                <button>Add to Cart</button>
            </article>
            <article class="product">
                <img src="./images/product2.jpg" alt="Product 2">
                <h2>Cameras</h2>
                <p>Capture every moment: Discover high-quality cameras tailored to your photography needs.</p>
                <button>Add to Cart</button>
```

```
        </article>
        <article class="product">
            <img src="./images/product3.jpg" alt="Product 3">
            <h2>Laptops</h2>
            <p>Powerful computing solutions: Find laptops designed for
productivity, creativity, and beyond.</p>
            <button>Add to Cart</button>
        </article>
    </section>
  </main>

  <script>
    window.addEventListener('load', () => {
        registerSW();
    });

    async function registerSW() {
        if ('serviceWorker' in navigator) {
            try {
                await navigator.serviceWorker.register('service-worker.js');
            } catch (e) {
                console.log('SW registration failed');
            }
        }
    }
  </script>
</body>
</html>
```

app.js

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

```
        })
        .catch(error => {
            console.error('Service Worker registration failed:', error);
        });
    });
}
```

service-worker.js

```
const staticCacheName = "ecommerce-v1";

self.addEventListener("install", function (e) {
    e.waitUntil(
        caches.open(staticCacheName).then(function (cache) {
            return cache.addAll([
                "/",
                "/index.html",
                "/styles.css",
                "/product1.jpg", // Add more product images as needed
                "/product2.jpg",
                "/product3.jpg",
                "manifest.json", // Ensure manifest is cached
                "serviceworker.js" // Ensure serviceworker is cached
            ]);
        })
    );
});
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:

127.0.0.1:5501/index.html
E-commerce Store
Welcome to our vibrant marketplace, where shopping meets delight! Explore our curated collection of must-have items, handpicked just for you. Start your journey with us today and discover endless possibilities in every click!
Explore our collection below:
Welcome
Elements
Console
Sources
Network
Performance
Application
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5501/
Network requests
Update
Unregister
Source
service-worker.js
Received 3/27/2024, 12:36:34 AM
Status
#587 activated and is running
stop
Push
Test push message from DevTools.
Sync
test-tag-from-devtools
Periodic Sync
Update Cycle
Version
Update Activity
Timeline
#587 Install
#587 Wait
#587 Activate
Service workers from other origins
See all registrations
Manifest
Service workers
Storage
Local storage
Session storage
IndexedDB
Web SQL
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Background services
Back/forward cache
Background fetch
Background sync
Notifications
Payment handler
Periodic background sync
Speculative loads
Push messaging
Reporting API
Frames
top
Console
Issues

127.0.0.1:5501/index.html
E-commerce Store
Welcome to our vibrant marketplace, where shopping meets delight! Explore our curated collection of must-have items, handpicked just for you. Start your journey with us today and discover endless possibilities in every click!
Explore our collection below:
Welcome
Elements
Console
Sources
Network
Performance
Application
Filter by Path
http://127.0.0.1:5501
Origin
http://127.0.0.1:5501
Bucket name
default
Is persistent
No
Durability
strict
Quota
0 B
Expiration
None
Name
Response-Type
Content-Type
Content-Leng...
Time Cached
Vary Header
Select a cache entry above to preview
Total entries: 0
Manifest
Service workers
Storage
Local storage
Session storage
IndexedDB
Web SQL
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
ecommerce-v1 - http
ecommerce-store-cac
Background services
Back/forward cache
Background fetch
Background sync
Notifications
Payment handler
Periodic background syn
Speculative loads
Push messaging
Reporting API
Frames
top
Console
Issues

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.

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.	37
Name	Atharva Santosh Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Atharva Mulam Batch B
D15A Roll No: 37

Exp No: 9

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

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

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

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

Of course, you can use many different methods but you can find in the following example a “cache first” and “network first” approach. In this example, if the

request's and current location's origin are the same (Static content is requested.), this is called "cacheFirst" but if you request a targeted external URL, this is called "networkFirst".

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

Sync Event
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are writing e-mail content and we didn't realize it. When completing the writing, we click the send button. Here is a job for the Background Sync.
The following view shows the classical process of sending email to us. If the Internet Connection is broken, we can't send any content to Mail Server.

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

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:

```
const CACHE_NAME = 'my-ecommerce-app-cache-v1';
const urlsToCache = [
  '/',
  '/index.html',
  '/styles.css',
  '/app.js',
  '/manifest.json',
  '/images/product1.jpg',
```

```
  '/images/product2.jpg',
  '/images/product3.jpg'
];
var staticCacheName = "pwa";

self.addEventListener("install", function (e) {
e.waitUntil(
        caches.open(staticCacheName).then(function (cache) {
        return cache.addAll(["/"]);
        })
);
});
self.addEventListener("fetch", function (event) {
console.log(event.request.url);

event.respondWith(
        caches.match(event.request).then(function (response) {
        return response || fetch(event.request);
        })
);
});

self.addEventListener('install', function(event) {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(function(cache) {
        console.log('Opened cache');
        return cache.addAll(urlsToCache)
          .catch(function(error) {
            console.error('Cache.addAll error:', error);
          });
      })
  );
});
```

```
self.addEventListener('activate', function(event) {
  // Perform activation steps
  event.waitUntil(
    caches.keys().then(function(cacheNames) {
      return Promise.all(
        cacheNames.map(function(cacheName) {
          if (cacheName !== CACHE_NAME) {
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});
// Fetch event listener
self.addEventListener("fetch", function (event) {
  event.respondWith(checkResponse(event.request).catch(function () {
    console.log("Fetch from cache successful!");
    return returnFromCache(event.request);
  }));
  console.log("Fetch successful!");
  event.waitUntil(addToCache(event.request));
});
// Sync event listener
self.addEventListener('sync', function(event) {
  if (event.tag === 'syncMessage') {
    console.log("Sync successful!");
  }
});


// Push event listener
self.addEventListener("push", function (event) {
  if (event && event.data) {
```

```
    try {
      var data = event.data.json();
      if (data && data.method === "pushMessage") {
        console.log("Push notification sent");
        self.registration.showNotification("Ecommerce website", { body:
data.message });
      }
    } catch (error) {
      console.error("Error parsing push data:", error);
    }
  }
});


self.addEventListener('activate', async () => {
  if (Notification.permission !== 'granted') {
    try {
      const permission = await Notification.requestPermission();
      if (permission === 'granted') {
        console.log('Notification permission granted.');
      } else {
        console.warn('Notification permission denied.');
      }
    } catch (error) {
      console.error('Failed to request notification permission:', error);
    }
  }
});
var checkResponse = function (request) {
  return new Promise(function (fulfill, reject) {
    fetch(request)
      .then(function (response) {
        if (response.status !== 404) {
          fulfill(response);
        } else {
```

```
        reject(new Error("Response not found"));
      }
    })
    .catch(function (error) {
      reject(error);
    });
  });
};
var returnFromCache = function (request) {
  return caches.open("offline").then(function (cache) {
    return cache.match(request).then(function (matching) {
      if (!matching || matching.status == 404) {
        return cache.match("offline.html");
      } else {
        return matching;
      }
    });
  });
};
var addToCache = function (request) {
  return caches.open("offline").then(function (cache) {
    return fetch(request).then(function (response) {
      return cache.put(request, response.clone()).then(function () {
        return response;
      });
    });
  });
};
```

app.js

```
if ('Notification' in window) {
    Notification.requestPermission().then(function (permission) {
        if (permission === 'granted') {
            console.log('Notification permission granted.');
        } else {
            console.warn('Notification permission denied.');
```

```
        }
    });
}
```

Output:

E-commerce Store
http://127.0.0.1:5501/index.html
http://127.0.0.1:5501 wants to
Show notifications
Allow
Block
E-commerce Store
Welcome to our vibrant marketplace, where shopping meets delight! Explore our curated collection of must-have items, handpicked just for you. Start your journey with us today and discover endless possibilities in every click!
Explore our collection below:
Mobile Phones
Cutting-edge mobile phones: Explore the latest in mobile technology with our diverse range of smartphones.
Cameras
Capture every moment: Discover high-quality cameras tailored to your photography needs.
Laptops
Powerful computing solutions: Find laptops designed for productivity, creativity, and beyond.

Conclusion :

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

MAD & PWA Lab Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	37
Name	Atharva Santosh Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Atharva Mulam Batch B
D15A Roll No: 37

Exp No: 10

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

Theory:

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

GitHub Pages provides the following key features:
Blogging with Jekyll
Custom URL
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.

Output:

Conclusion:

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

MAD & PWA Lab Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	37
Name	Atharva Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

Atharva Mulam Batch B
D15A Roll No: 37

Exp No: 11

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

Theory :

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

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

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

Output:
For theme color add a meta tag in index.html-
<meta name="theme-color" content="#4285f4">

For a maskable icon add "purpose": "any maskable" to the icons in manifest.json file

Project Title: Flutter : Instagram clone/ PWA: Ecommerce website **Roll No. 37**

Changes in manifest.json

```
{
   "name": "E-commerce Store",
```

```
  "short_name": "Store",
  "start_url": "index.html",
  "display": "standalone",
  "background_color": "#5900b3",
  "theme_color": "black",
  "scope": ".",

  "description": "This is an e-commerce store.",
  "icons": [
    {
                "src": "images/icon-144x144.jpg",
                "sizes": "144x144",
                "type": "image/png",
                "purpose": "any maskable"
            },
            {
      "src": "images/icon-192x192.jpg",
      "sizes": "192x192",
      "type": "image/png",
                "purpose": "any maskable"
    },
    {
      "src": "images/icon-512x512.jpg",
      "sizes": "512x512",
      "type": "image/png",
                "purpose": "any maskable"
    }
  ]
}
```

Project Title: Flutter : Instagram clone/ PWA: Ecommerce website **Roll No. 37**

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

MAD & PWA Lab Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	37
Name	Atharva Santosh Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	5

Atharva Mulam
D15A Roll no. 37

Date
Page

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 gained popularity in the developer community.

Ans. Flutter is a cross-platform development framework developed by Google, allowing developers to create, mobile, desktop and web apps using a single codebase.

Key features and advantages →

1) Cross-platform development - Flutter enables developers to write code once and run it on multiple platforms including iOS, Android and web saving time and resources.

2) Hot reload - The Hot reload feature in flutter allows developers to see changes in real-time as they modify the code, significantly speeding up the development process.

3) Rich set of widgets - Flutter provides a variety of rich widgets, which are building blocks of flutter apps.

4) Publicly accessible - Being an open-source, flutter is publicly accessible allowing developers to easily access the original codebase for their projects.

5) Support for multiple languages - Flutter supports multiple programming languages such as Dart, Java, C/C++ and more providing flexibility in developing frontend and backend applications.

Teacher's Sign.: _____

The flutter framework differs from traditional approaches in several ways. Traditional android apps use Java code and skia graphics engine for rendering, while cross-platform frameworks use an abstraction layer that can result in significant overhead. However, flutter bypasses system UI libraries, and uses its own widgets set with native code compiled from dart code resulting in high performance.

Q27 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 and their roles in creating a widget tree.

Ans. In flutter, widget tree is a hierarchical structure where every widget is a node. The root of the tree is typically a MaterialApp or CupertinoApp widget. Each widget can have zero or more child widgets forming a tree structure.

Widget composition is a technique where you combine simple widgets to build more complex ones. You can nest widgets inside other widgets to create a widget tree. For example, you might have a column widget that contains multiple text widgets. The column widget serves as a parent for these widgets.

Examples of commonly used widgets -

1) Stateless widgets - These are simplest kind of widget. They describe part of the user interface which can depend on configuration information in the parent widget but cannot change dynamically.

2) Container - This is a convenience widget that combines painting, positioning, and sizing widgets.

3) Row and column – These are flex widgets that allow you to arrange their children horizontally (row) and vertically (column).
4) Scaffold – Provides a basic structure for visual elements of an app such as appbar, body and bottom navigation.

Q3) 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.

Ans. State management is a critical aspect of flutter development, playing a pivotal role in ensuring a responsive and efficient user interface. Effective state management strategies is key to building robust and maintainable applications.

Importance of state management –

1) Performance optimization – Effective state management minimizes unnecessary widget rebuilds, improving performance by avoiding redundant UI updates.
2) Scalability – As an application grows in complexity, proper state management becomes crucial for maintaining a manageable and organised databases.
3) Code maintainability – Well managed state promotes clean code architecture making it easier to understand, maintain and scale app.
4) Testing – Proper state management facilitates unit testing and makes it easier to write test cases for different application states.

Teacher's Sign.: _____

3) Real-time updates - when the data changes in firestore database the changes are instantly pushed into all the connected clients that have set up listeners for that particular data.

4) Offline support - Firestore provides offline support allowing users to interact with the app even when offline. Data modifications made while offline are stored locally and synced with server once device is online.

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.	37
Name	Atharva Santosh Mulam
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	4

Atharva Mulam
DISA Roll no: 37
Batch B

Assignment 2
PWA

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

Ans. (i) The progressive web apps (PWA) are a type of web application that uses modern web technologies to provide users with a native app-like experience directly through their web browser.

(ii) PWAs are designed to be reliable, fast and engaging offering features such as offline access, push notifications and the ability to install the app on user's device.

The significance of PWAs in modern web development lies in their ability to bridge the gap between web and native mobile application

Key characteristics of PWAs are as follows :

(1) cross platform compatibility - PWAs are built using web technologies like HTML, CSS, JS making them compatible with various platforms and devices.

(2) Responsiveness - PWAs are designed to adapt seamlessly to different screen sizes and resolutions.

(3) Offline functionality - PWAs can cache resources and content enabli users to access the app when they're offline or have poor connec

(4) Push notifications - PWAs can send push notifications to user increasing engagement and enabling realtime communication with users even when the app is not open.

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

Ans. (i) Responsive web design is an approach to web development aimed at creating website that provide an optimal viewing and interaction experience across a range of devices & screen sizes.

(ii) The key principle of responsive web design is the use of flexible layouts, fluid grids and media queries to adapt the layout and content of a website based on the device's screen size and orientation.

(iii) Since, PWAs are accessed through web browsers, they need to be responsive to provide a consistent and user-friendly experience regardless of the device being used.

	Responsive	Fluid	Adaptive
i)	Designs that adapt to any screen size using CSS media queries to adjust layout	A subset of responsive design where the layout is defined using percentages allowing elements to fit.	Uses multiple fixed layouts for different screen sizes serving the most appropriate on user's device.
ii)	Easier to implement requires a single design that fits different screens.	Also easy to implement, focusing on fluid resizing of elements.	More complex due to need to create and maintain layouts
iii)	Generally more search engine friendly.	Similar to responsive design	SEO benefits are not directly affected by the design approach
iv)	Requires ongoing maintainance.	Requires ongoing maintainance for fluid sizing	More complex to maintain due to multiple layouts.

FOR EDUCATIONAL USE

Sundaram®

FOR EDUCATIONAL USE

Q3) Describe the lifecycle of service workers including registration, installation and activation phases.

Ans. The lifecycle of service workers involves several phases: registration, installation and activation.

(1) Registration

→ This occurs in the main script of a web application.

→ During registration, the service worker script is fetched from the server and registered with the browser using 'navigator.service worker.register()' method.

→ The registration process typically occurs in the background and does not block the main thread of the web application

(2) Installation

→ After successful registration, the browser downloads the service worker script and begins the installation script.

→ During installation, the browser fires the 'install' event allowing service workers to cache static asset and other resources needed to run web application offline.

→ The service worker can use the 'event.waitUntil()' method to extend the installation process until all resources are cached.

(3) Activation

→ Once the service worker is successfully installed, the browser fires the 'activate' event.

→ During activation, the service worker can clean up any old caches to perform other tasks necessary to ~~secure~~ ensure it is ready to control client pages.

→ The 'activate' event is also an opportunity for service workers to take control of client pages and intercept n/w requests using event listeners such as 'fetch' and 'message'.

Q4) Explain the use of IndexedDB in the service worker for data storage.

Ans. (i) IndexedDB is a low level API for client side storage that allows web applications to store different types of data persistently in the user's browser.

(ii) It is a full-blown, persistent NoSQL storage system that supports various data types.

(iii) IndexedDB is particularly useful in scenarios such as caching and PWAs and gaming and it supports transactions to ensure data integrity.

(iv) Service workers can intercept network requests and serve cached responses but for applications that require data persistence and offline access to data stored in IndexedDB

(v) This combination allows web application to function offline by serving cached assets and data stored in IndexedDB ensuring a seamless UX even without an internet connection.