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

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Project Title: **Roll No.** 23

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

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

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers : Mrs. Kajal Joseph.

Email : kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: **Roll No.** 23

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.** 23

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.** 23

MAD & PWA Lab
Journal

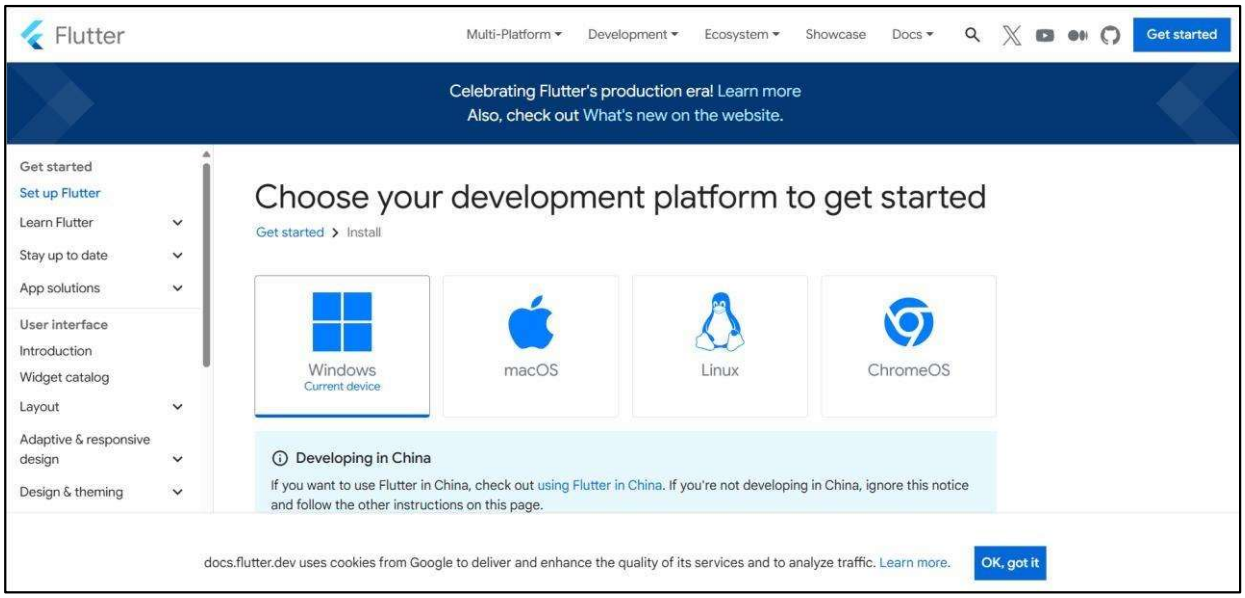
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	23
Name	Ronak Katariya
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

EXPERIMENT NO: - 01

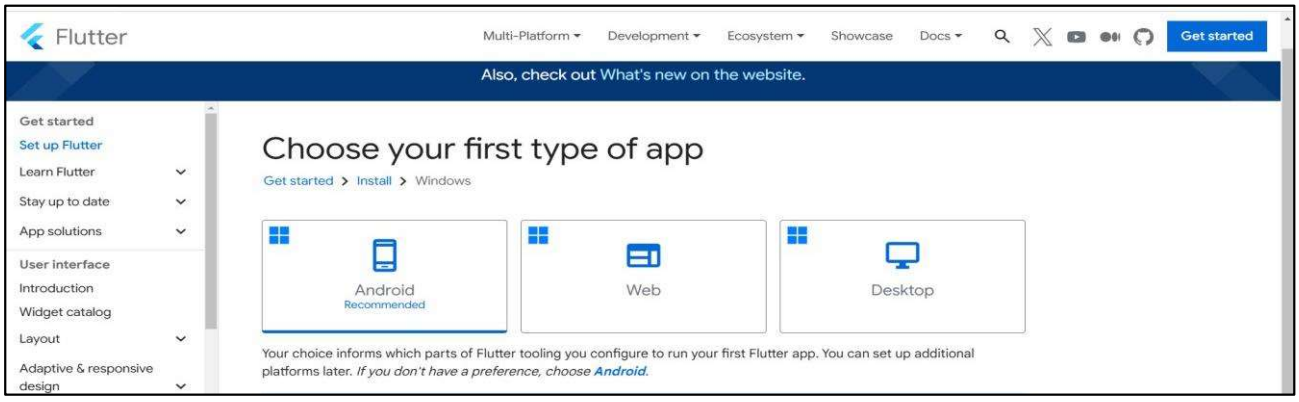
Name:- Ronak Katariya **Class:-** D15A **Roll:No: -** 23

AIM: - Installation and Configuration of Flutter Environment.

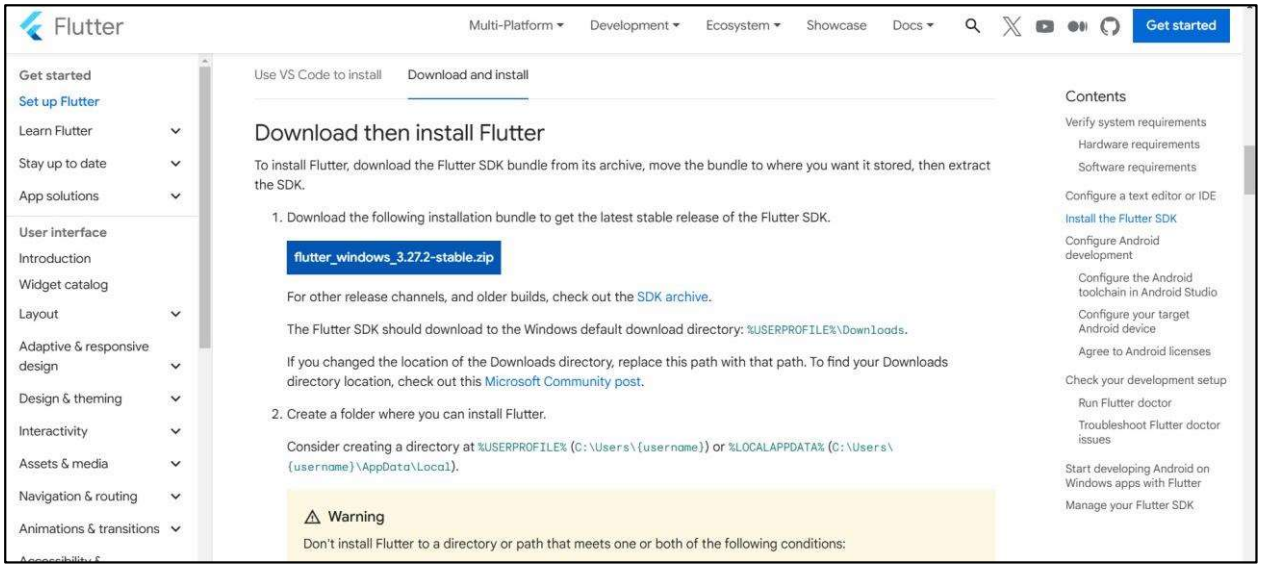
Step 1: Go to the official Flutter website: https://docs.flutter.dev/get-started/install



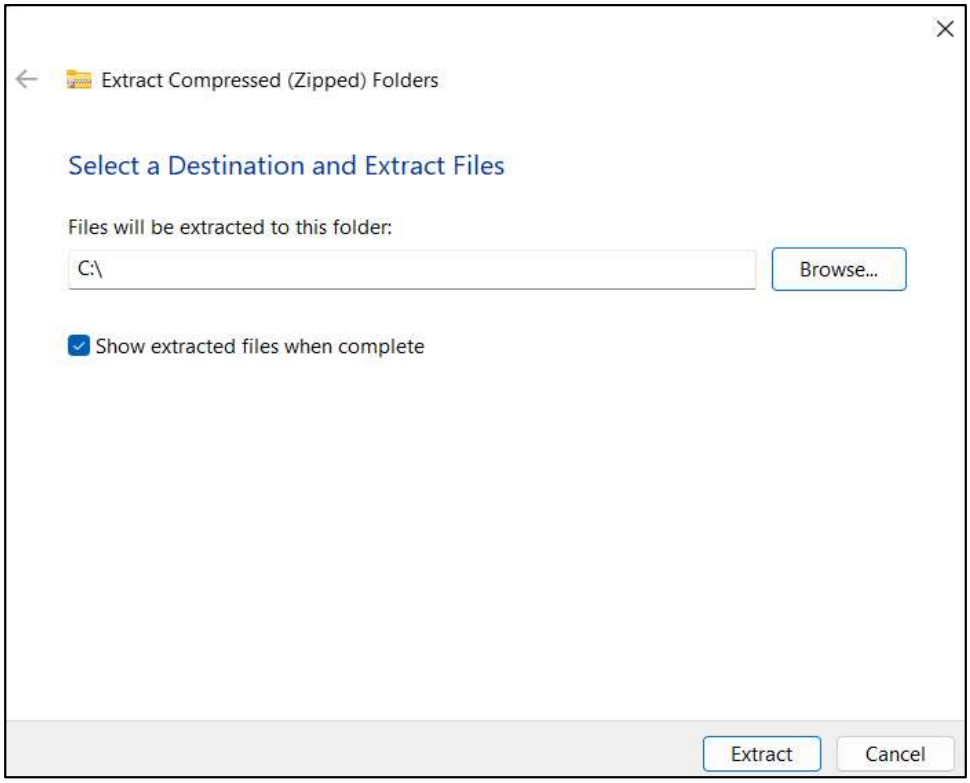
Step 2: To download the latest Flutter SDK, click on the Windows icon > Android



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



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

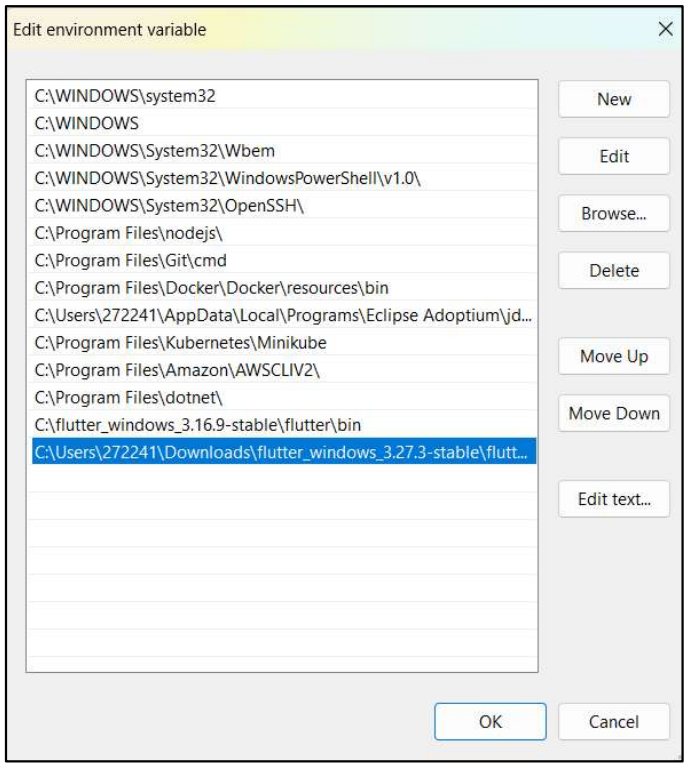


Step 5 :- Add Flutter to System PATH

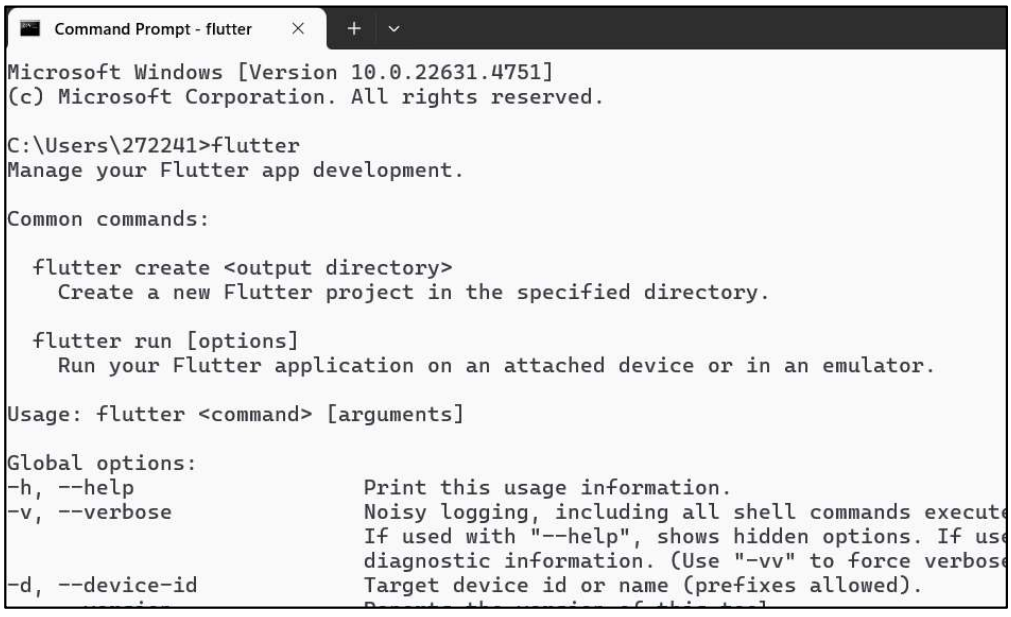
Right-click on the Start Menu > System > Advanced system settings > Environment Variables.

Under System Variables, find Path and click Edit.

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



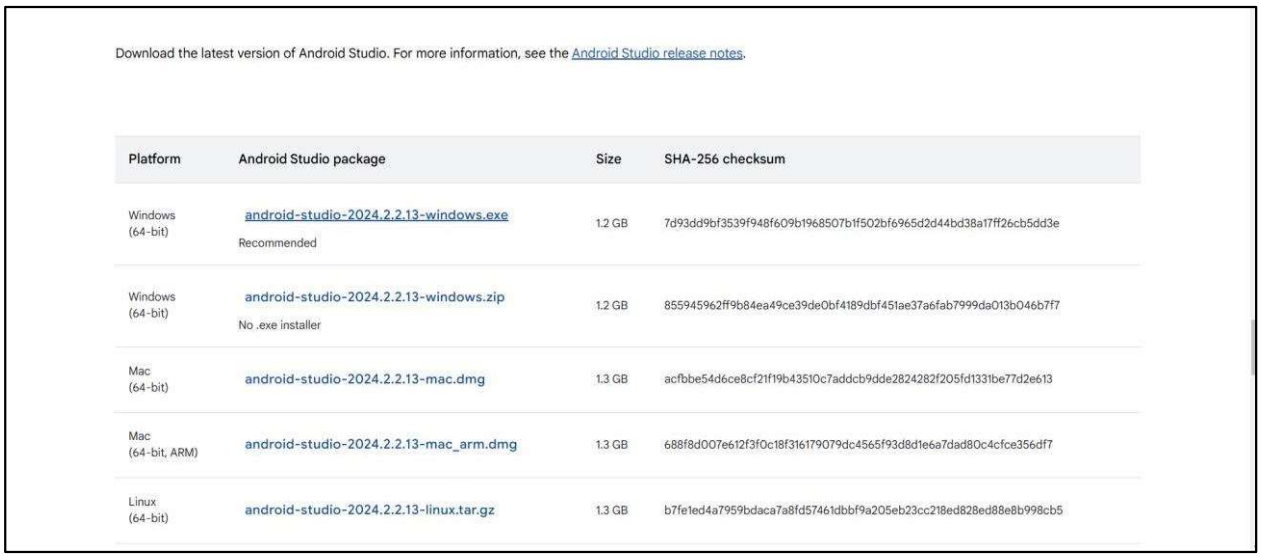
Step 6 : - Now, run the $ flutter command in command prompt.



Step 7:- Run the $ flutter doctor command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation

```
C:\Users\272241>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.27.3, on Microsoft Windows [Version 10.0.22631.4751], locale en
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[✓] Android toolchain - develop for Android devices (Android SDK version 34.0.0)
[✓] Chrome - develop for the web
[!] Visual Studio - develop Windows apps (Visual Studio Community 2022 17.12.4)
    X Visual Studio is missing necessary components. Please re-run the Visual Studio installer
      development with C++" workload, and include these components:
        MSVC v142 - VS 2019 C++ x64/x86 build tools
         - If there are multiple build tool versions available, install the latest
        C++ CMake tools for Windows
        Windows 10 SDK
[✓] Android Studio (version 2023.1)
[✓] VS Code (version 1.96.4)
[✓] Connected device (3 available)
[✓] Network resources

! Doctor found issues in 1 category.
```

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

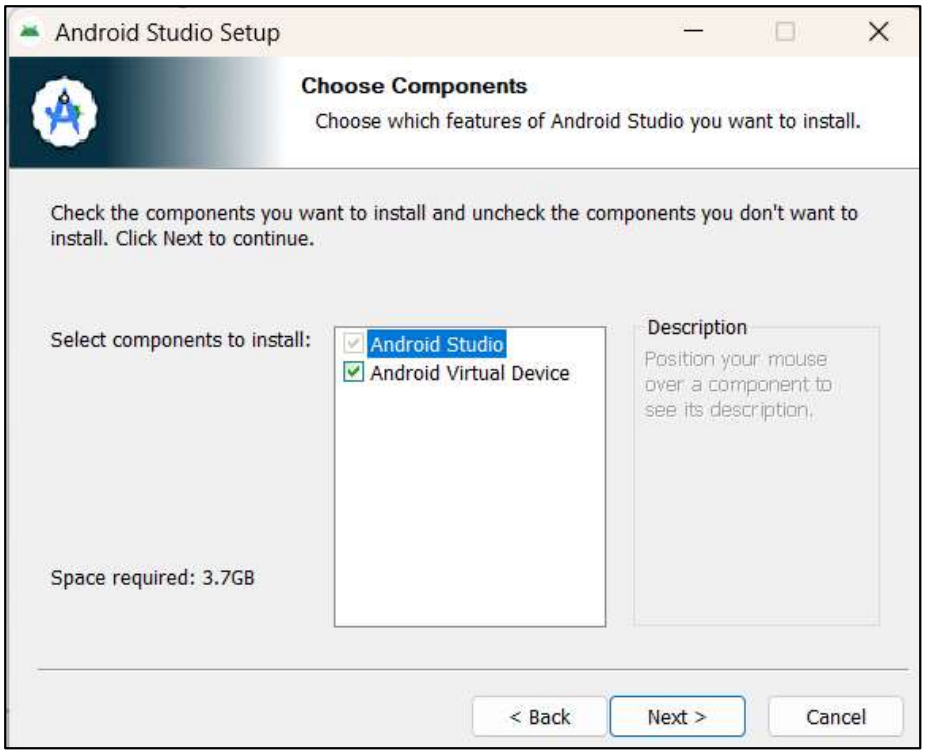


Download the latest version of Android Studio. For more information, see the Android Studio release notes.

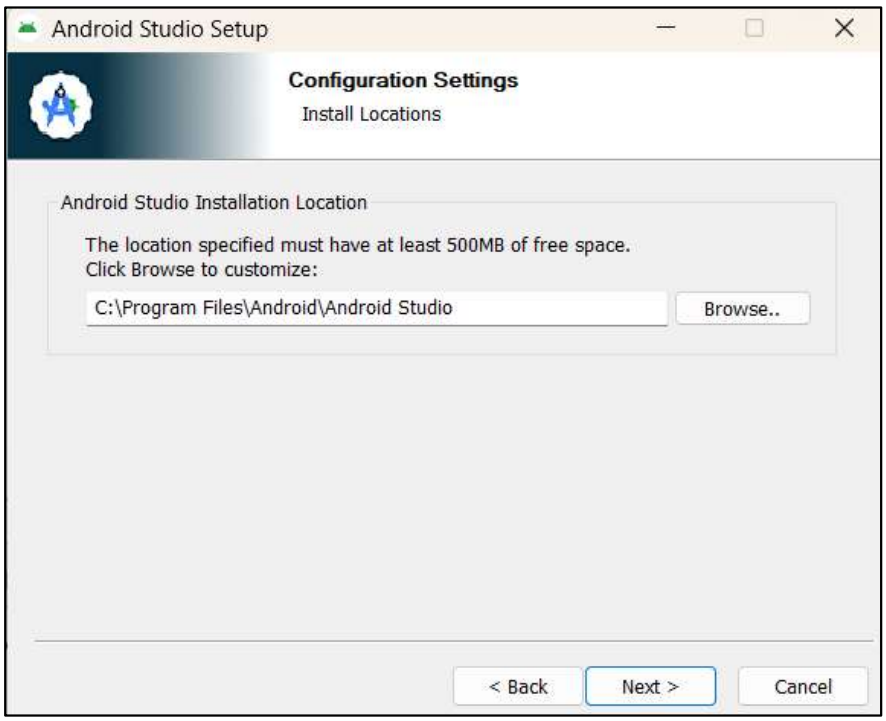
Platform	Android Studio package	Size	SHA-256 checksum
Windows (64-bit)	android-studio-2024.2.2.13-windows.exe Recommended	1.2 GB	7d93dd9bf3539f948f609b1968507b1f502bf6965d2d44bd38a17ff26cb5dd3e
Windows (64-bit)	android-studio-2024.2.2.13-windows.zip No .exe installer	1.2 GB	855945962ff9b84ea49ce39de0bf4189dbf451ae37a6fab7999da013b046b7f7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

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

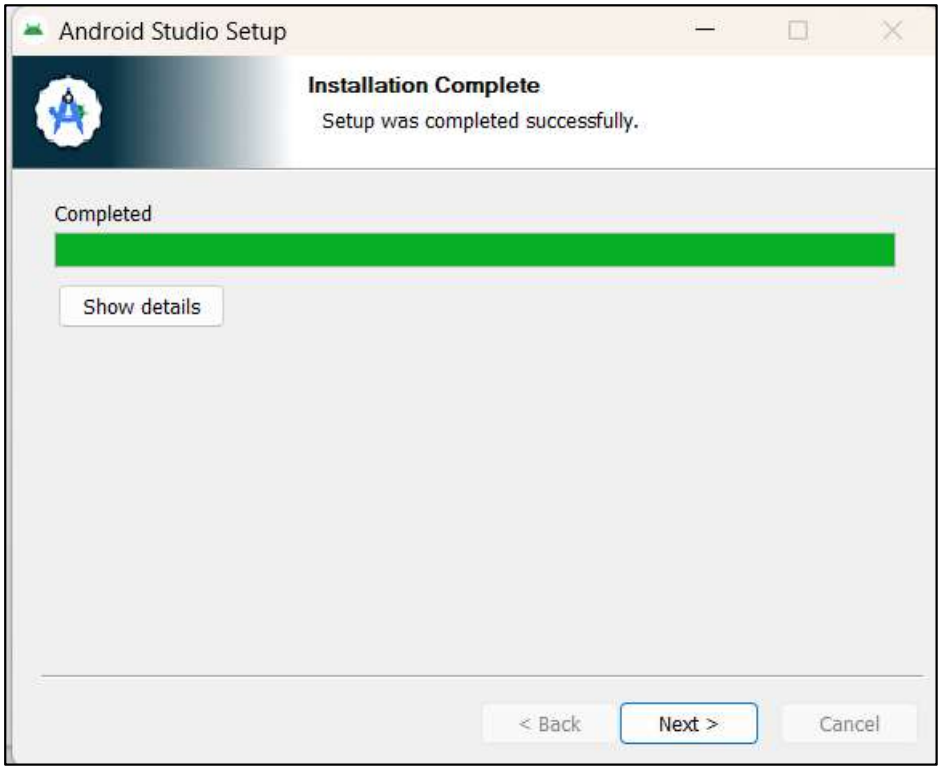
Step 8.2: - Select all the Checkboxes and Click on 'Next' Button.

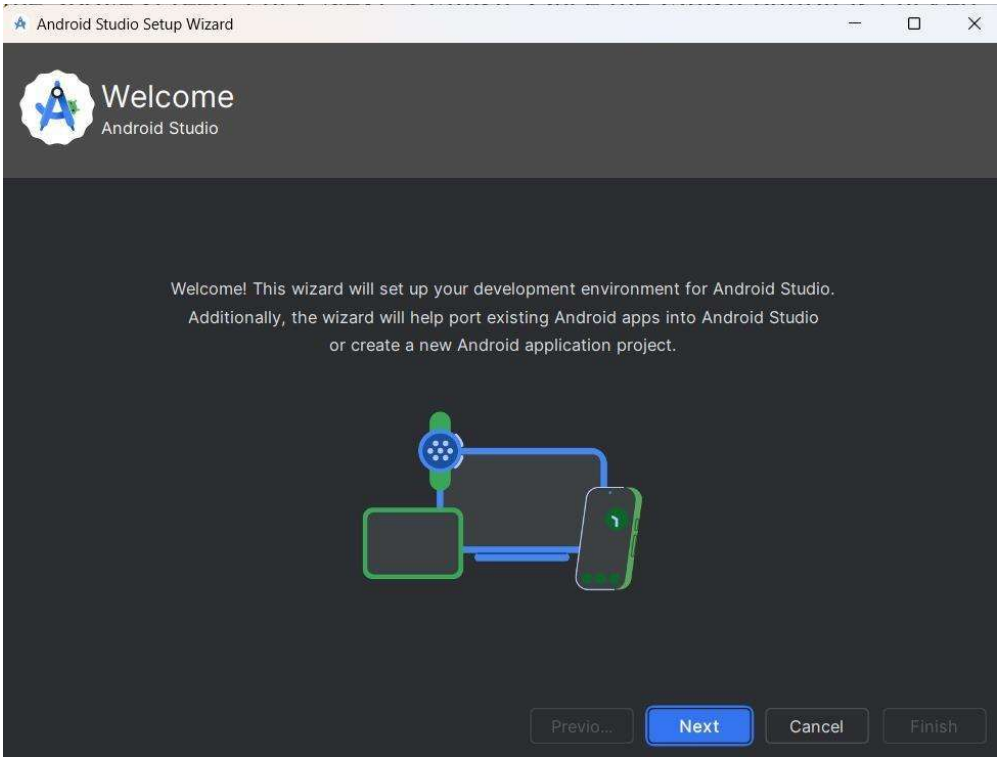


Step 8.3: - Change the destination as per your convenience and click on 'Next' Button.



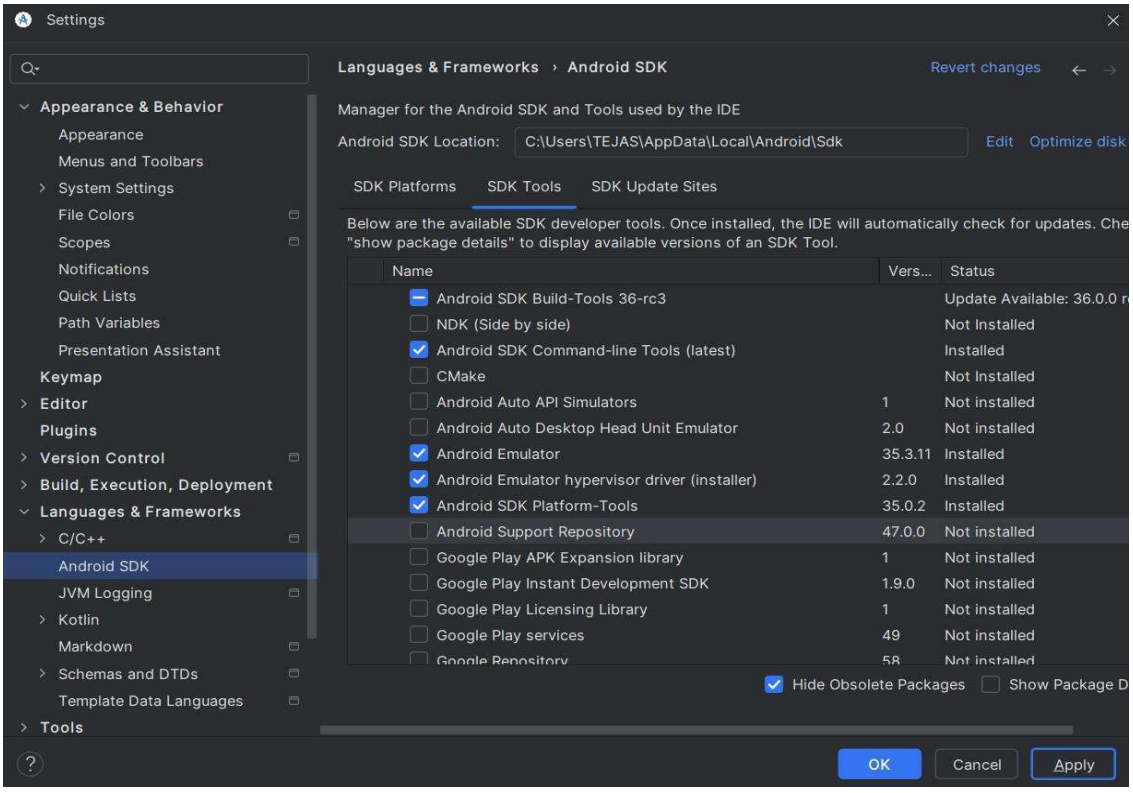
Step 8.4: - Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.





Step 8.5: - Go to Preferences > Appearance & Behavior > System Settings > Android SDK.

Select the SDK Tools tab and check Android SDK Command-line Tools and Install it.

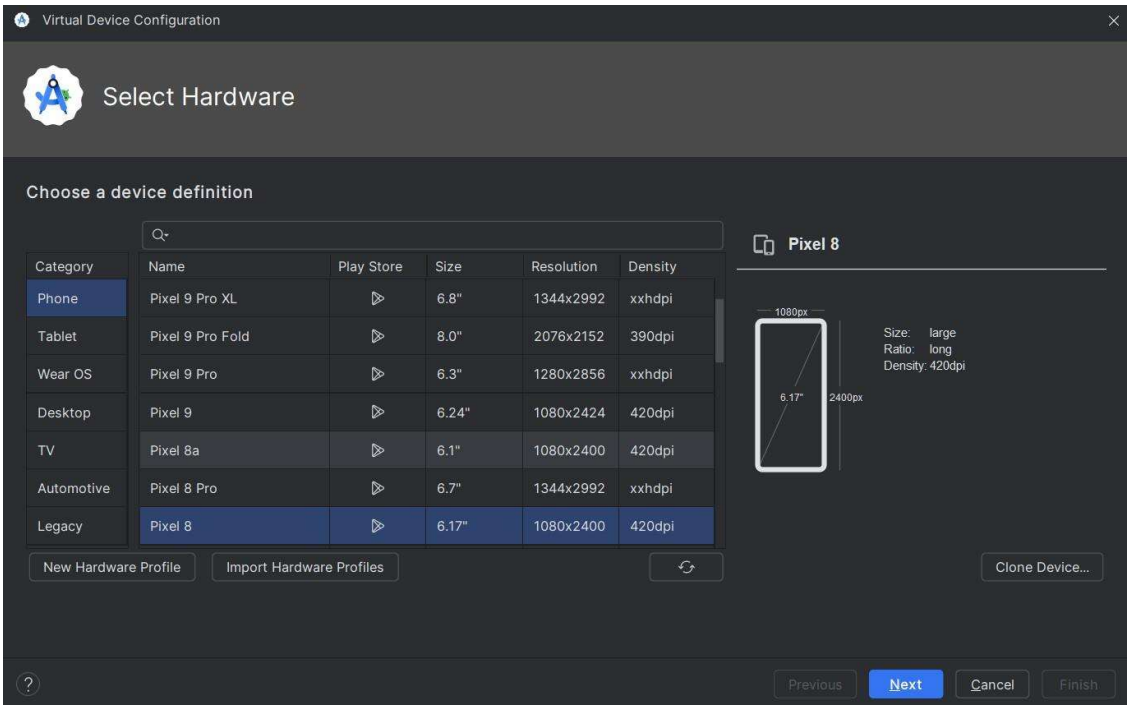


Step 9: - Open a terminal and run the following command

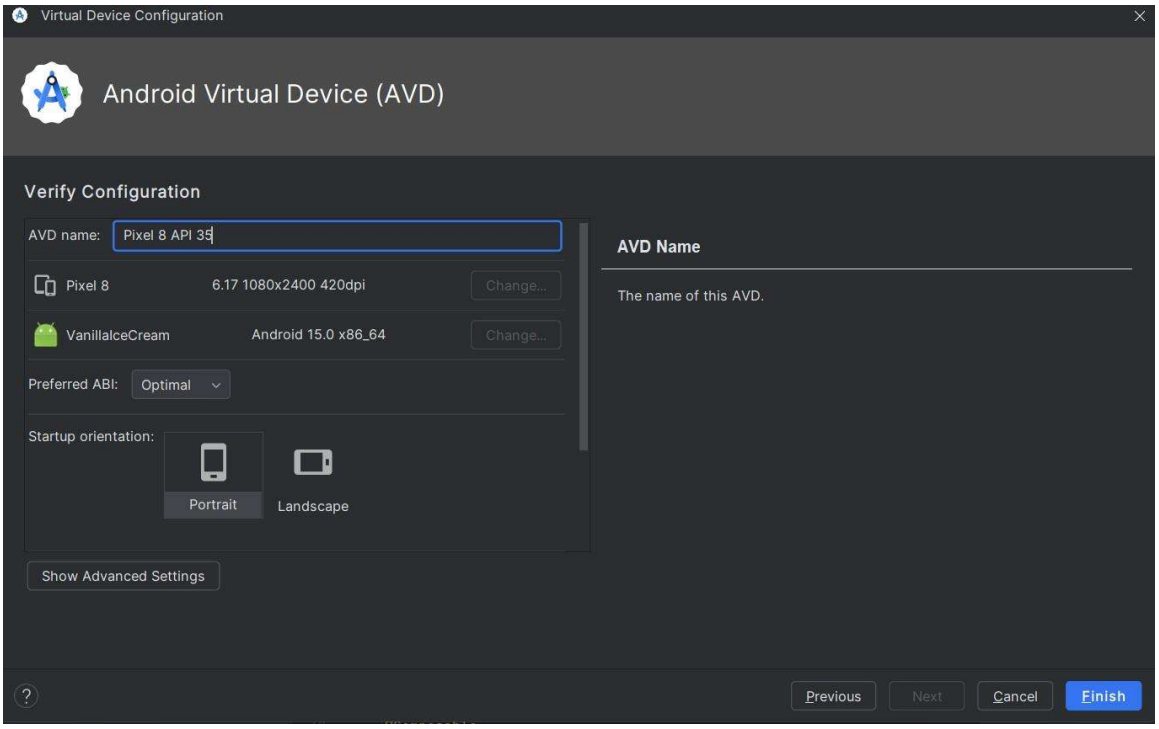
```
C:\Users\272241>flutter doctor --android-licenses
                                                                    Warning: Errors during XML parse:
Warning: Additionally, the fallback loader failed to parse the XML.
Warning: Errors during XML parse:       ] 64% Fetch remote repository...
Warning: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


C:\Users\272241>
```

```
C:\Users\272241>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.27.3, on Microsoft Windows [Version 10.0.22631.4751], locale en
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[✓] Android toolchain - develop for Android devices (Android SDK version 34.0.0)
[✓] Chrome - develop for the web
[!] Visual Studio - develop Windows apps (Visual Studio Community 2022 17.12.4)
    X Visual Studio is missing necessary components. Please re-run the Visual Studio installer
      development with C++" workload, and include these components:
        MSVC v142 - VS 2019 C++ x64/x86 build tools
          - If there are multiple build tool versions available, install the latest
        C++ CMake tools for Windows
        Windows 10 SDK
[✓] Android Studio (version 2023.1)
[✓] VS Code (version 1.96.4)
[✓] Connected device (3 available)
[✓] Network resources

! Doctor found issues in 1 category.
```

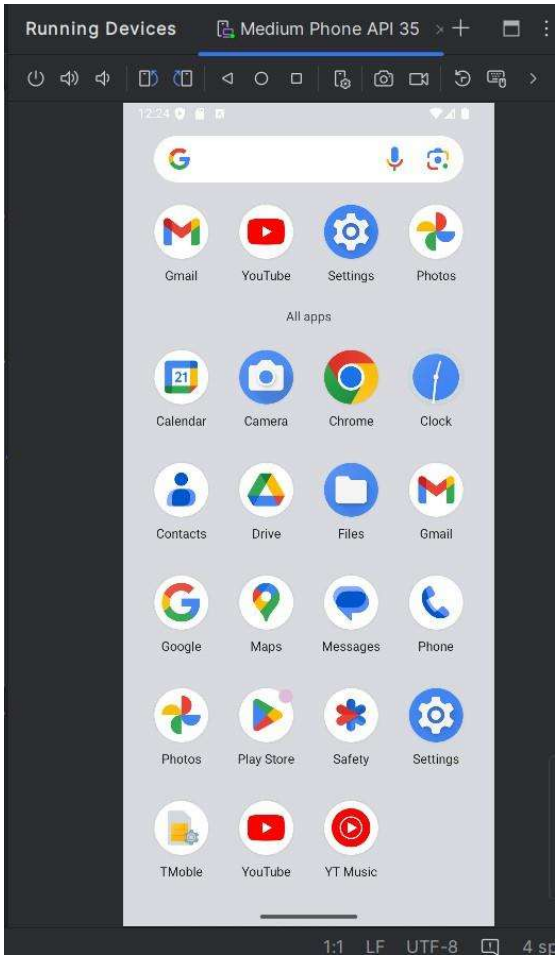
Step 10: - Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application



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

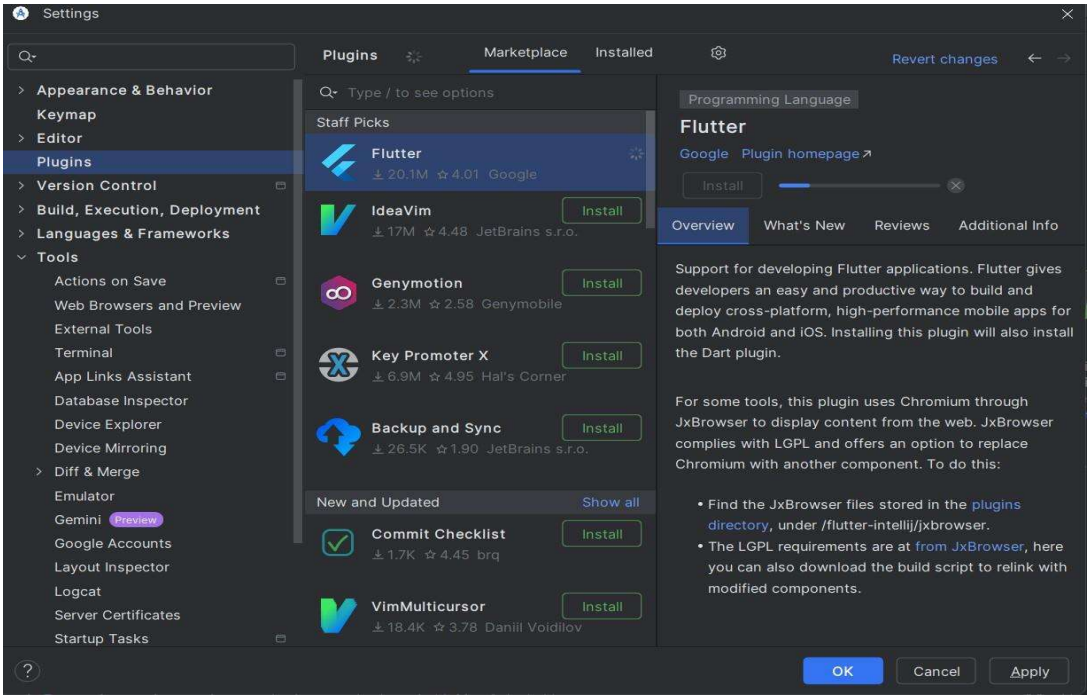


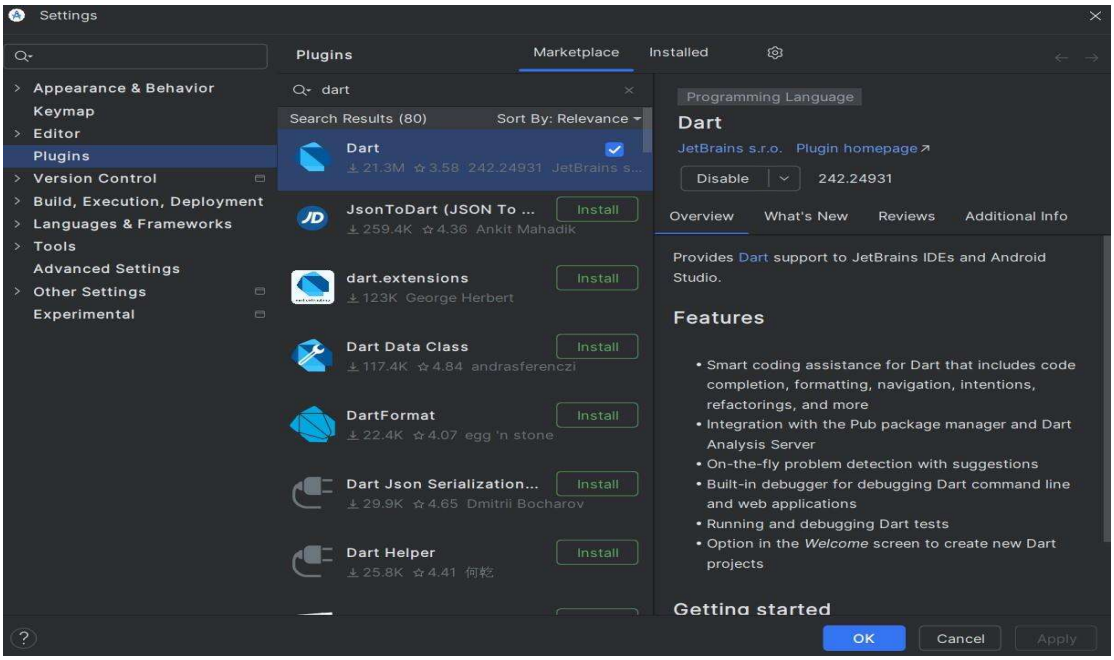
Step 10.2: - Click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen



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

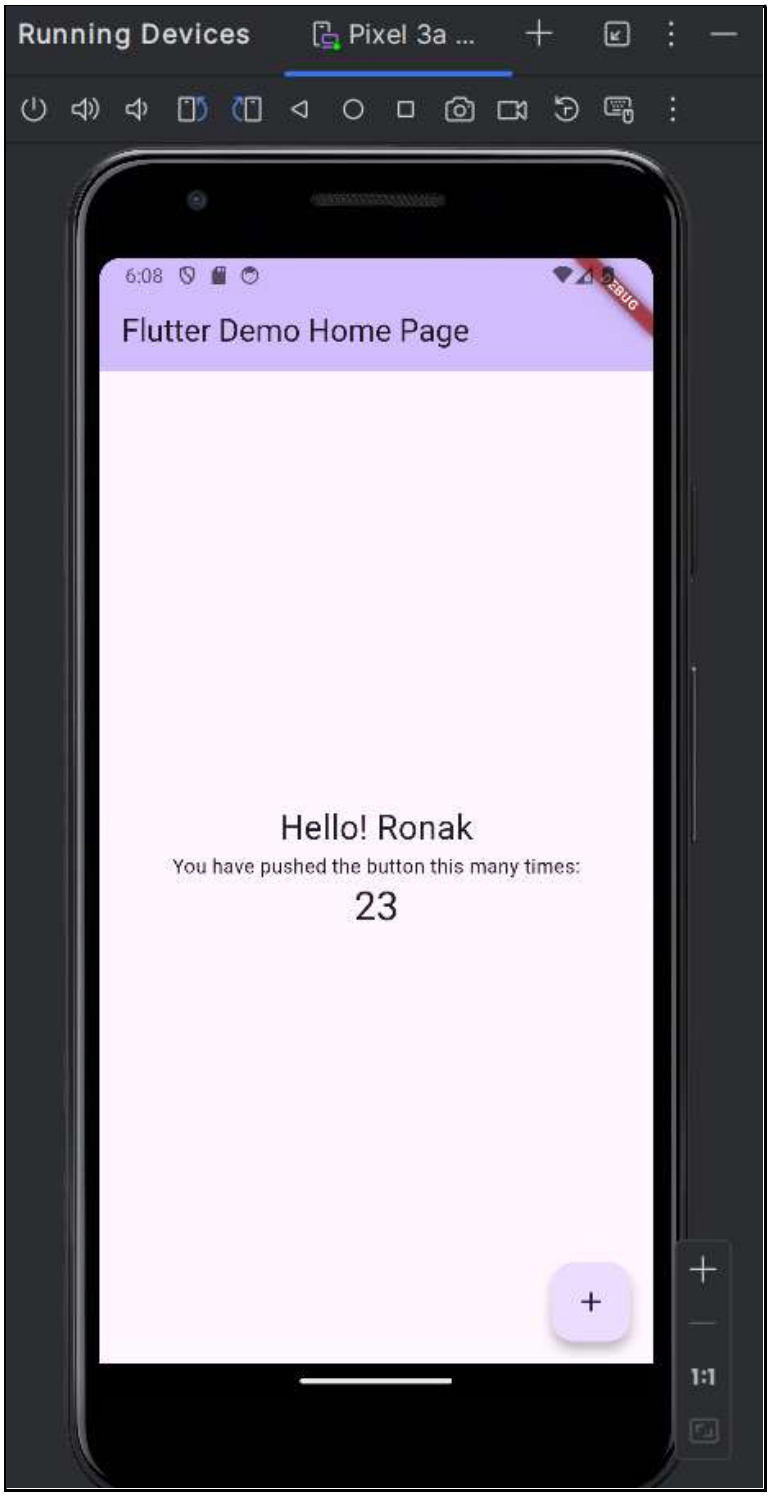
Step 11.1: - Open the Android Studio and then go to File->Settings->Plugins. Now, search the Flutter plugin. If found, select Flutter plugin and click install





Step 11.2: - Restart the Android Studio

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



MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	23
Name	Ronak Katariya
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

EXPERIMENT NO: - 02

Name:- Ronak Katariya **Class:-** D15A **Roll:No: -** 23

AIM: - To design Flutter UI by including common widgets.

Theory: -

Each element on the screen of the Flutter app is a widget. The view of the screen completely depends upon the choice and sequence of the widgets used to build the apps. And the structure of the code of apps is a tree of widgets.

When you made any alteration in the code, the widget rebuilds its description by calculating the difference of previous and current widget to determine the minimal changes for rendering in UI of the app.Widgets are nested with each other to build the app. It means the root of your app is itself a widget, and all the way down is a widget also. For example, a widget can display something, can define design, can handle interaction, etc.

The single child layout widget is a type of widget, which can have only **one child widget** inside the parent layout widget. These widgets can also contain special layout functionality. Flutter provides us many single child widgets to make the app UI attractive. If we use these widgets appropriately, it can save our time and makes the app code more readable.

The multiple child widgets are a type of widget, which contains **more than one child widget**, and the layout of these widgets are **unique**. For example, Row widget laying out of its child widget in a horizontal direction, and Column widget laying out of its child widget in a vertical direction. If we combine the Row and Column widget, then it can build any level of the complex widget.

Type of Widgetss

- **StatefulWidget**

  A StatefulWidget has state information. It contains mainly two classes: the state object and the widget. It is dynamic because it can change the inner data during the widget lifetime. This widget does not have a build() method. It has createState() method, which returns a class that extends the Flutters State Class. The examples of the StatefulWidget are Checkbox, Radio, Slider, InkWell, Form, and TextField.

- **StatelessWidget**

  The StatelessWidget does not have any state information. It remains static throughout its lifecycle. The examples of the StatelessWidget are Text, Row, Column, Container, etc.

Some of the commonly used widgets

Container – A box widget used for styling with padding, margins, colors, borders, and constraints. It helps in layout structuring and positioning.

Row & Column – Used to arrange widgets in horizontal (Row) or vertical (Column) orientation. They manage spacing, alignment, and distribution of child widgets.

Stack – Overlaps widgets on top of each other, useful for creating layered UIs like banners, tooltips, or floating elements.

Text – Displays text on the screen with customizable font size, color, alignment, and styling options

Image – Loads and displays images from assets, network, or memory with scaling, fit, properties.

Scaffold – Provides a basic layout structure with an app bar, body, floating action button, and bottom navigation.

ListView – A scrollable list widget that efficiently renders large amounts of dynamic content. Supports both vertical and horizontal scrolling.

GridView – Displays widgets in a grid format, useful for galleries, product listings, or dashboards. It supports dynamic column adjustments.

SizedBox – Used to create space between widgets or define fixed width and height for layout adjustments.

ElevatedButton – A button with elevation that provides a raised effect, customizable with color, shape, and click actions.

TextField – A user input field that supports text entry, keyboard configurations, validation.

AppBar – A top navigation bar that includes a title, actions, and menu icons, commonly used in Scaffold.

BottomNavigationBar – A bar at the bottom of the screen used for navigation between different app sections with icons and labels.

Drawer – A side navigation panel that slides out from the left, typically used for app menus and quick navigation.

Card – A material design component that displays content inside a box with elevation.

Code: - quiz_summary.dart

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

class QuizSummaryScreen extends
StatelessWidget {
  final int score;
  final int total;
  final List<Map<String, dynamic>>
userResponses;

  const QuizSummaryScreen({super.key,
required this.score, required this.total, required
this.userResponses});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("Quiz
Summary")),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment:
CrossAxisAlignment.start,
          children: [
            Text("Your Score: $score / $total", style:
const TextStyle(fontSize: 26, fontWeight:
FontWeight.bold)),
            const SizedBox(height: 20),
            Expanded(
              child: ListView(
                children:
userResponses.map((response) => Card(
                  child: ListTile(
                    title: Text(response['question']),
                    subtitle: Text("Your Answer:
${response['selected']} \nCorrect Answer:
${response['correct']}"),
                    trailing: response['isCorrect'] ?
const Icon(Icons.check, color: Colors.green) :
const Icon(Icons.close, color: Colors.red),
                  ),
                )).toList(),
              ),
            ),
            ElevatedButton(
              onPressed: () => Navigator.pop(context),
              child: const Text("Back to Categories"),
            ),
          ],
        ),
      ),
    );
  }
}
```

Code: landing_page.dart

```
import 'package:flutter/material.dart';
import 'package:quiz/widgets/bottom_navbar.dart';
import
'package:quiz/screens/category_selection_screen.dart'
;
import 'package:quiz/screens/leaderboard.dart';
import 'package:quiz/screens/profile.dart';

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

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

class _LandingPageState extends
State<LandingPage> {
  int _selectedIndex = 0;

  final List<Widget> _pages = [
    const HomeScreen(), // Landing Page UI
    const CategorySelectionScreen(),
    const LeaderboardPage(),
    const ProfilePage(),
  ];

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: _pages[_selectedIndex],
      bottomNavigationBar: BottomNavBar(
        currentIndex: _selectedIndex,
        onTap: _onItemTapped,
      ),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: HomeContent(),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: const BoxDecoration(
        gradient: LinearGradient(
          colors: [Color(0xFF6A11CB),
Color(0xFF2575FC)],
          begin: Alignment.topCenter,
          end: Alignment.bottomCenter,
        ),
      ),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          const Padding(
            padding: EdgeInsets.symmetric(horizontal: 20),
            child: Text(
              "Test Your Knowledge 🚀",
              style: TextStyle(
                fontSize: 26,
                fontWeight: FontWeight.bold,
                color: Colors.white,
              ),
              textAlign: TextAlign.center,
            ),
          ),
          const SizedBox(height: 20),
          _buildCategoryCards(context),
          const SizedBox(height: 20),
          ElevatedButton(
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => const
CategorySelectionScreen()),
              );
            },
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.white,
              foregroundColor: Colors.blueAccent,
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(30),
              ),
```

```
            padding: const
EdgeInsets.symmetric(horizontal: 40, vertical: 15),
          ),
          child: const Text("Start Quiz", style:
TextStyle(fontSize: 18)),
        ),
       ],
     ),
   );
 }

 /// ✅ Function to build category cards
 Widget _buildCategoryCards(BuildContext context)
{
   List<Map<String, String>> categories = [
     {"title": "CNS", "image": "assets/cns.jpg"},
     {"title": "OS", "image": "assets/os.jpg"},
     {"title": "DSA", "image": "assets/dsa.jpg"},
     {"title": "SQL", "image": "assets/sql.png"},
   ];

   return SizedBox(
     height: 200,
     child: ListView.builder(
       scrollDirection: Axis.horizontal,
       itemCount: categories.length,
       itemBuilder: (context, index) {
         return
_categoryCard(categories[index]["title"]!,
categories[index]["image"]!, context);
       },
     ),
   );
 }

 /// ✅ Category Card Widget
 Widget _categoryCard(String title, String
imagePath, BuildContext context) {
   return GestureDetector(
     onTap: () {
       Navigator.push(
         context,
         MaterialPageRoute(builder: (context) => const
CategorySelectionScreen()),
       );
     },
     child: Card(
       margin: const EdgeInsets.symmetric(horizontal: 10),
       shape: RoundedRectangleBorder(
         borderRadius: BorderRadius.circular(20),
       ),
       elevation: 5,
       child: Container(
         width: 150,
         decoration: BoxDecoration(
           borderRadius: BorderRadius.circular(20),
           image: DecorationImage(
             image: AssetImage(imagePath),
             fit: BoxFit.cover,
           ),
         ),
         child: Center(
           child: Text(
             title,
             style: const TextStyle(
               fontSize: 20,
               fontWeight: FontWeight.bold,
               color: Colors.white,
               backgroundColor: Colors.black54,
             ),
           ),
         ),
       ),
     ),
   );
 }
}
```

Code: -

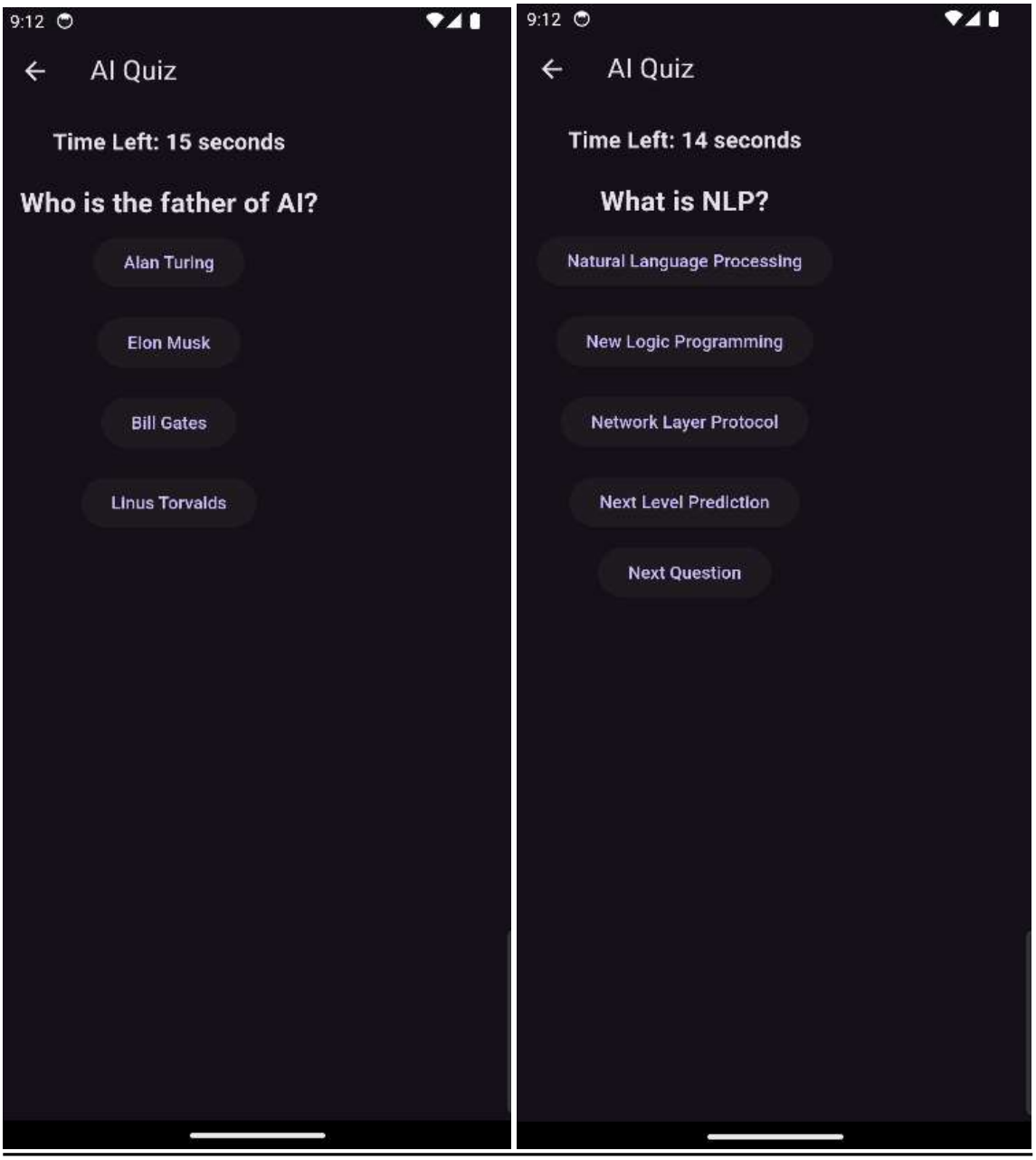
quiz_screen.dart

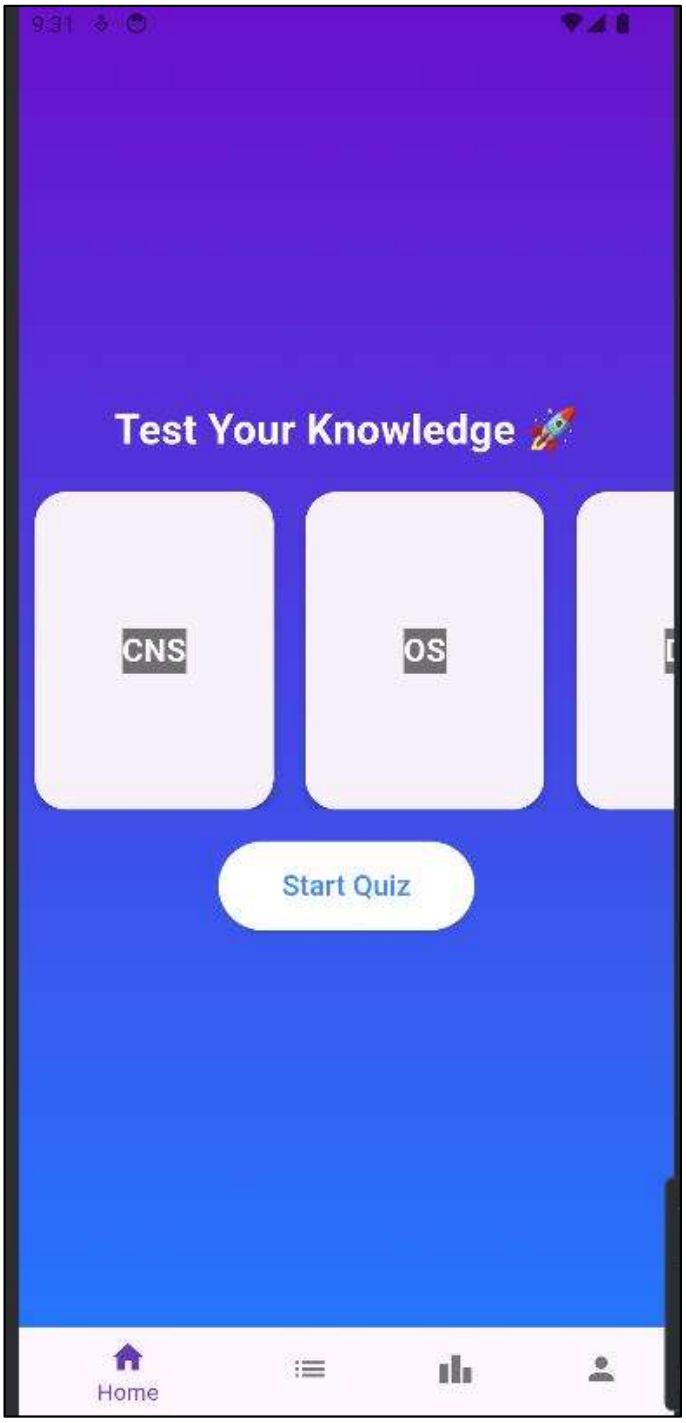
```
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:quiz/screens/quiz_summary.dart';

class QuizScreen extends StatefulWidget {
  final String category;
  const QuizScreen({super.key, required
this.category});

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

class _QuizScreenState extends State<QuizScreen>
{
  int currentQuestion = 0;
  int score = 0;
  late Timer timer;
  int timeLeft = 20;
  bool answered = false;
  List<Map<String, dynamic>> userResponses = [];

  final Map<String, List<Map<String, dynamic>>>
questionBank = {
    "CNS": [
      {
        "question": "What is a firewall?",
        "options": ["Security Device", "OS",
"Protocol", "Network"],
        "answer": "Security Device"
      },
      {
        "question": "What is encryption?",
        "options": ["Encoding data", "Deleting data",
"Sending messages", "Accessing files"],
        "answer": "Encoding data"
      },
      {
        "question": "What is a VPN?",
        "options": ["Virtual Private Network",
"Variable Public Network", "Visual Private
Network", "Virtual Protected Network"],
        "answer": "Virtual Private Network"
      },
      {
        "question": "What does IDS stand for?",
        "options": ["Intrusion Detection System", "Internal
Data Security", "Internet Data Service", "Integrated
Defense System"],
        "answer": "Intrusion Detection System"
      },
      {
        "question": "What is a DDoS attack?",
        "options": ["Distributed Denial of Service", "Direct
Denial of Service", "Distributed Data Service", "Direct
Data Service"],
        "answer": "Distributed Denial of Service"
      },
      {
        "question": "What is a proxy server?",
        "options": ["Intermediary for requests", "Data
storage device", "Network protocol", "Firewall type"],
        "answer": "Intermediary for requests"
      },
      {
        "question": "What does SSL stand for?",
        "options": ["Secure Sockets Layer", "Standard
Security Layer", "Simple Sockets Layer", "Secure
System Layer"],
        "answer": "Secure Sockets Layer"
      },
      {
        "question": "What is malware?",
        "options": ["Malicious software", "Machine
learning software", "Management software", "Multi-
layer software"],
        "answer": "Malicious software"
      },
      {
        "question": "What is phishing?",
        "options": ["Fraudulent attempt to obtain sensitive
info","Data encryption method","Network security
measure","Type of firewall"],
        "answer":"Fraudulent attempt to obtain sensitive
info"
      },
```

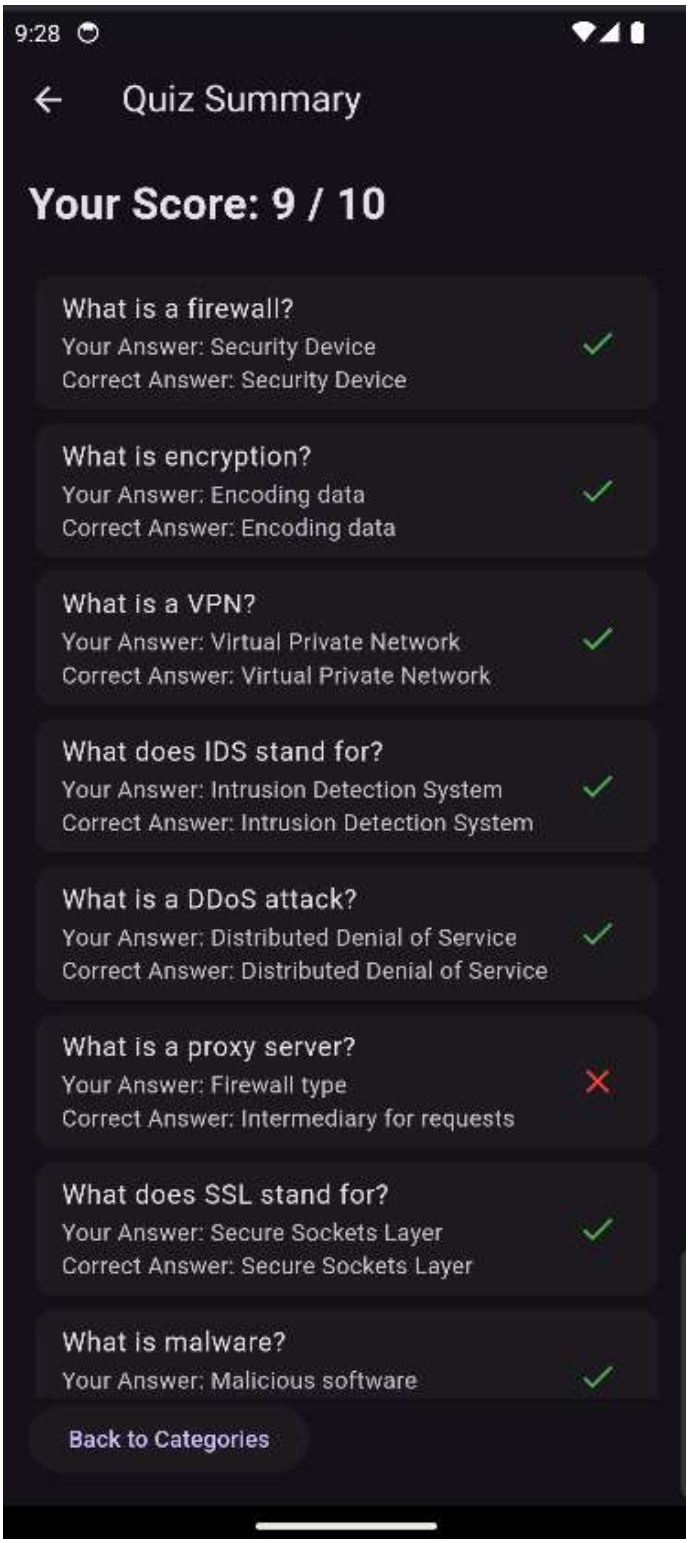
```
  {
      "question":"What is a botnet?",
      "options":["Network of infected
devices","Type of firewall","Data encryption
method","Software application"],
      "answer":"Network of infected devices"
    }
   ],
   "OS": [
    {"question": "What is CPU scheduling?",
"options": ["Round Robin", "Mutex", "Binary
Search", "Recursion"], "answer": "Round Robin"},
    {"question": "What does OS stand for?",
"options": ["Operating System", "Open Source",
"Output System", "Offline Storage"], "answer":
"Operating System"},
   ],
   "Cloud Computing": [
    {"question": "What is SaaS?", "options":
["Software as a Service", "Storage as a Service",
"Security as a Service", "System as a Service"],
"answer": "Software as a Service"},
    {"question": "AWS stands for?", "options":
["Amazon Web Services", "Azure Web Solutions",
"Advanced Web Server", "Automated Web
Security"], "answer": "Amazon Web Services"},
   ],
   "AI": [
    {"question": "Who is the father of AI?",
"options": ["Alan Turing", "Elon Musk", "Bill
Gates", "Linus Torvalds"], "answer": "Alan
Turing"},
    {"question": "What is NLP?", "options":
["Natural Language Processing", "New Logic
Programming", "Network Layer Protocol", "Next
Level Prediction"], "answer": "Natural Language
Processing"},
   ],
  };

  late List<Map<String, dynamic>> questions;

  @override
  void initState() {
   super.initState();
   questions = questionBank[widget.category] ?? [];
   timer = Timer.periodic(const Duration(seconds:
1), (timer) {
    if (timeLeft == 0) {
     nextQuestion();
    } else {
     setState(() {
      timeLeft--;
     });
    }
   });
  }

  void nextQuestion() {
   if (currentQuestion < questions.length - 1) {
    setState(() {
     currentQuestion++;
     timeLeft = 20;
     answered = false;
    });
   } else {
    Navigator.pushReplacement(
     context,
     MaterialPageRoute(
      builder: (_) => QuizSummaryScreen(score: score,
total: questions.length, userResponses: userResponses),
     ),
    );
   }
  }

  void selectAnswer(String selected) {
   if (!answered) {
    bool isCorrect = selected ==
questions[currentQuestion]['answer'];
    if (isCorrect) score += 1;

    userResponses.add({
     "question": questions[currentQuestion]['question'],
     "selected": selected,
     "correct": questions[currentQuestion]['answer'],
     "isCorrect": isCorrect,
    });

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

  @override
  void dispose() {
   timer.cancel();
   super.dispose();
  }
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("${widget.category} Quiz")),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            Text("Time Left: $timeLeft seconds", style: const TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
            const SizedBox(height: 20),
            Text(questions[currentQuestion]['question'], style: const TextStyle(fontSize: 22, fontWeight: FontWeight.bold)),

...questions[currentQuestion]['options'].map((option) => Padding(
              padding: const EdgeInsets.symmetric(vertical: 8.0),
              child: ElevatedButton(
                onPressed: () => selectAnswer(option),
                child: Text(option),
              ),
            )),
            if (answered)
              ElevatedButton(
                onPressed: nextQuestion,
                child: const Text("Next Question"),
              ),
          ],
        ),
      ),
    );
  }
}
```

OUTPUT: -


9:12
AI Quiz
Time Left: 15 seconds
Who is the father of AI?
Alan Turing
Elon Musk
Bill Gates
Linus Torvalds
9:12
AI Quiz
Time Left: 14 seconds
What is NLP?
Natural Language Processing
New Logic Programming
Network Layer Protocol
Next Level Prediction
Next Question


9:31
Test Your Knowledge 🚀
CNS
OS
Start Quiz
Home


9:28
Quiz Summary
Your Score: 9 / 10
What is a firewall?
Your Answer: Security Device
Correct Answer: Security Device
What is encryption?
Your Answer: Encoding data
Correct Answer: Encoding data
What is a VPN?
Your Answer: Virtual Private Network
Correct Answer: Virtual Private Network
What does IDS stand for?
Your Answer: Intrusion Detection System
Correct Answer: Intrusion Detection System
What is a DDoS attack?
Your Answer: Distributed Denial of Service
Correct Answer: Distributed Denial of Service
What is a proxy server?
Your Answer: Firewall type
Correct Answer: Intermediary for requests
What does SSL stand for?
Your Answer: Secure Sockets Layer
Correct Answer: Secure Sockets Layer
What is malware?
Your Answer: Malicious software
Back to Categories

Project Title: **Roll No.** 23

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	23
Name	Ronak Katariya
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

EXPERIMENT NO: - 03

Name:- Ronak Katariya **Class:-** D15A **Roll:No: -** 23

AIM: - To include icons, images, fonts in Flutter app.

Theory: -

Flutter is a versatile open-source UI framework , which allows developers to build natively compiled applications for mobile, web, and desktop platforms from a single codebase. One of the key strengths of Flutter is its flexibility in creating highly customizable UIs. This practical focuses on incorporating essential visual elements—icons, images, and custom fonts—into a Flutter application. These elements enhance the visual appeal and usability of the app, providing an engaging experience for users.

A flutter app when built has both assets (resources) and code. Assets are available and deployed during runtime. The asset is a file that can include static data, configuration files, icons, and images. The Flutter app supports many image formats, such as JPEG, WebP, PNG, GIF, animated WebP/GIF, BMP, and WBMP.

Visual elements play a significant role in app development.

- **Enhanced User Experience**: Images and icons make your app visually appealing and user-friendly.
- **Information Conveyance**: They convey information quickly and intuitively. A well-chosen icon can replace lengthy text.
- **Branding**: Custom icons and images reinforce your app's branding, making it memorable.

➢ **Adding Icons in Flutter**

Flutter provides built-in material design icons through the Icons class. Custom icons can also be added using third-party packages such as flutter_launcher_icons and font_awesome_flutter.

```
Icon(
  Icons.home,
  size: 40,
);
```

➢ Adding Images in Flutter

Flutter supports images from three sources:

1. **Assets** (Stored locally in the project)

   - Place the image inside the assets/images folder in the project.
   - Declare the image in pubspec.yaml

     ```
     flutter:
       assets:
         - assets/images/sample.png
     ```

   - Display the image in the app

     ```
     Image.asset('assets/images/sample.png');
     ```

2. **Network** (Fetched from the internet)

   Displaying images from the internet or network is very simple. Flutter provides a built-in method Image.network to work with images from a URL. The Image.network method also allows you to use some optional properties, such as height, width, color, fit, and many more.

   ```
   Image.network('https://example.com/sample.jpg');
   ```

3. **Memory or File** (Stored on the device)

➢ Adding Custom Fonts in Flutter

By default, Flutter uses the Roboto font, but custom fonts can be added for a unique UI.

- Download the font and place it in the assets/fonts/ folder.
- Declare the font in pubspec.yaml
- Use the font in the app

  ```
  Text(
    'Custom Font Example',
    style: TextStyle(fontFamily: 'CustomFont', fontSize: 24),
  );
  ```

Code: -

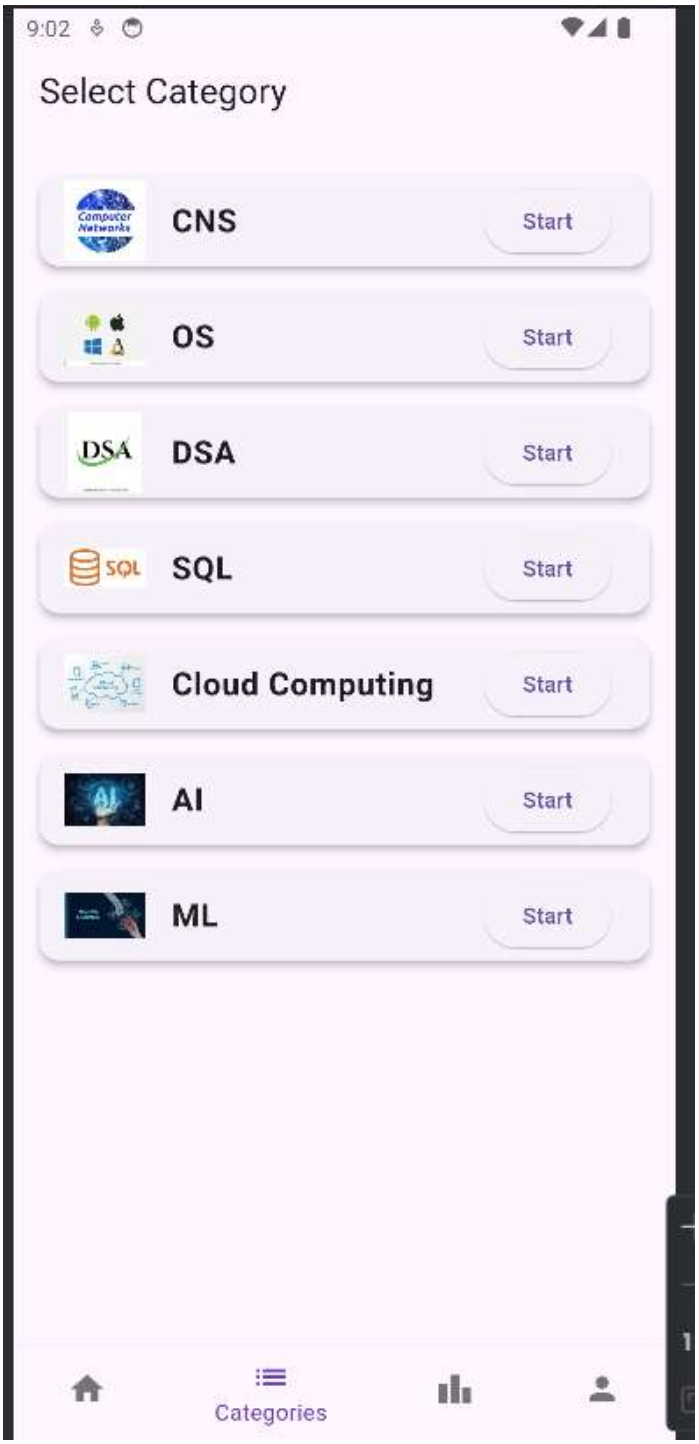
Category_selection_screen.dart

```
import 'package:flutter/material.dart';
import 'package:quiz/screens/quiz_screen.dart';

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

  @override
  Widget build(BuildContext context) {
    final categories = [
      {"title": "CNS", "image":
"assets/images/cns.jpg"},
      {"title": "OS", "image": "assets/images/os.jpg"},
      {"title": "DSA", "image":
"assets/images/dsa.jpg"},
      {"title": "SQL", "image":
"assets/images/sql.png"},
      {"title": "Cloud Computing", "image":
"assets/images/cloud.png"},
      {"title": "AI", "image": "assets/images/ai.jpg"},
      {"title": "ML", "image":
"assets/images/ml.jpg"},
    ];

    return Scaffold(
      appBar: AppBar(title: const Text("Select
Category")),
      body: ListView.builder(
        padding: const EdgeInsets.all(16),
        itemCount: categories.length,
        itemBuilder: (context, index) {
          return Card(
            elevation: 4,
            margin: const EdgeInsets.symmetric(vertical:
8),
            child: ListTile(
              leading: Image.asset(categories[index]['image']!,
width: 50, height: 50),
              title: Text(categories[index]['title']!, style: const
TextStyle(fontSize: 20, fontWeight: FontWeight.bold)),
              trailing: ElevatedButton(
                onPressed: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (_) => QuizScreen(category:
categories[index]['title']!),
                    ),
                  );
                },
                child: const Text("Start"),
              ),
            ),
          );
        },
      ),
    );
  }
}
```

pubspec.yaml

```
flutter:
assets:
    - assets/images/
    - assets/animations/
    - assets/fonts/

  fonts:
    - family: Poppins
      fonts:
        - asset: assets/fonts/Poppins-Regular.ttf
        - asset: assets/fonts/Poppins-Bold.ttf
          weight: 700
```

OUTPUT: -


9:02
Select Category
CNS
Start
OS
Start
DSA
Start
SQL
Start
Cloud Computing
Start
AI
Start
ML
Start
Categories

Project Title: **Roll No.** 23

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	23
Name	Ronak Katariya
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

EXPERIMENT NO: - 04

Name:- Ronak Katariya **Class:-** D15A **Roll:No: - 23**

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

Theory: -

A form in Flutter is a structured container that collects user input through various fields like text fields, dropdowns, checkboxes, and buttons. It plays a crucial role in applications that require user data entry, such as login pages, registration forms, and feedback submissions. Flutter provides the Form widget, which works alongside TextFormField and other input elements to manage validation, state handling, and error messages efficiently. By using form validation techniques, developers can ensure data accuracy and enhance user experience.

When you create a form, it is necessary to provide the GlobalKey. This key uniquely identifies the form and allows you to do any validation in the form fields. The form widget uses child widget TextFormField to provide the users to enter the text field. This widget renders a material design text field and also allows us to display validation errors when they occur.

Creation of a Form

- While creating a form in Flutter, the **Form widget** is essential as it acts as a container for grouping multiple form fields and managing validation.
- A **GlobalKey<FormState>** is required to uniquely identify the form and enable validation or data retrieval from the form fields.
- The **TextFormField widget** is used to provide input fields where users can enter data such as names, phone numbers, or email addresses.
- To enhance the appearance and usability of input fields, **InputDecoration** is used, allowing customization of labels, icons, borders, and hint text.
- Validation plays a crucial role in forms, and the **validator property** within **TextFormField** ensures user input meets specific criteria before submission.

- Different types of input require appropriate **keyboard types**, such as TextInputType.number for numeric fields or TextInputType.emailAddress for email fields.

- Proper **state management** is needed to store and retrieve user input, ensuring the form data is processed correctly.

- A **submit button** is necessary to trigger form validation and submit the collected data for further processing.

Some Properties of Form Widget

- **key:** A GlobalKey that uniquely identifies the Form. You can use this key to interact with the form, such as validating, resetting, or saving its state.
- **child:** The child widget that contains the form fields. Typically, this is a Column, ListView, or another widget that allows you to arrange the form fields vertically.
- **autovalidateMode:** An enum that specifies when the form should automatically validate its fields.

Some Methods of Form Widget

- **validate():** This method is used to trigger the validation of all the form fields within the Form. It returns true if all fields are valid, otherwise false. You can use it to check the overall validity of the form before submitting it.
- **save():** This method is used to save the current values of all form fields. It invokes the onSaved callback for each field. Typically, this method is called after validation succeeds.
- **reset():** Resets the form to its initial state, clearing any user-entered data.
- **currentState:** A getter that returns the current FormState associated with the Form.

Code: -

login_screen.dart

```
import 'package:flutter/material.dart';
import 'signup_screen.dart';
import 'landing_page.dart'; // Import the
LandingPage

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

  @override
  State<LoginScreen> createState() =>
_LoginScreenState();
}

class _LoginScreenState extends
State<LoginScreen> {
  final TextEditingController _emailController =
TextEditingController();
  final TextEditingController _passwordController =
TextEditingController();

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  void _login(BuildContext context) {
    String email = _emailController.text;
    String password = _passwordController.text;

    if (email == 'abc@gmail.com' && password ==
'abc') {
      // Simulate successful login
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Login
Successful!')),
      );
      // Navigate to the LandingPage
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => const
LandingPage()),
      );
    } else {
      // Show error message for invalid credentials
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Invalid email or
password.')),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        decoration: const BoxDecoration(
          gradient: LinearGradient(
            begin: Alignment.topCenter,
            end: Alignment.bottomCenter,
            colors: [Color(0xFF4A00E0),
Color(0xFF8E2DE2)],
          ),
        ),
        child: Center(
          child: Padding(
            padding: const EdgeInsets.all(20.0),
            child: Column(
              mainAxisAlignment:
MainAxisAlignment.center,
              children: [
                const Text(
                  "Welcome Back",
                  style: TextStyle(fontSize: 28, fontWeight:
FontWeight.bold, color: Colors.white),
                ),
                const SizedBox(height: 10),
                const Text(
                  "Enter your credentials to login",
                  style: TextStyle(fontSize: 16, color:
Colors.white70),
                ),
                const SizedBox(height: 30),
                _buildTextField(Icons.email, "Email",
controller: _emailController),
                const SizedBox(height: 15),
                _buildTextField(Icons.lock, "Password",
isPassword: true, controller: _passwordController),
                const SizedBox(height: 20),
                ElevatedButton(
                  onPressed: () {
                    _login(context);
                  },
```

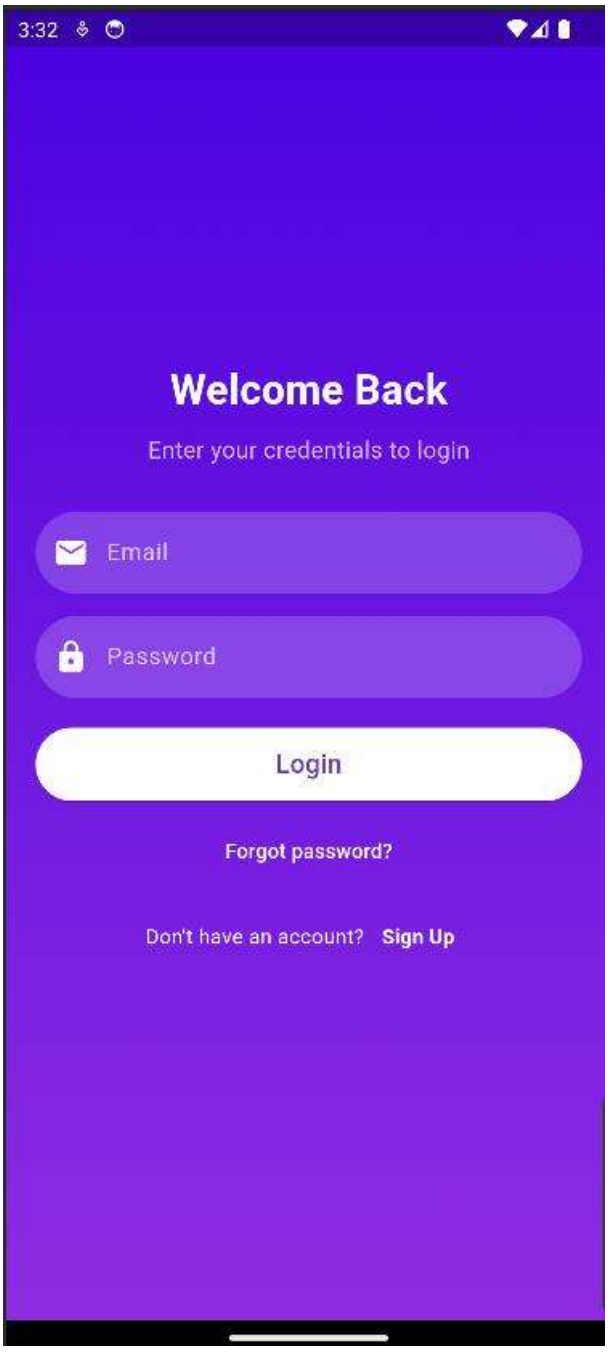
```
   style: ElevatedButton.styleFrom(
                backgroundColor: Colors.white,
                foregroundColor: Colors.deepPurple,
                minimumSize: const
Size(double.infinity, 50),
                shape:
RoundedRectangleBorder(borderRadius:
BorderRadius.circular(30)),
              ),
              child: const Text("Login", style:
TextStyle(fontSize: 18)),
            ),
            const SizedBox(height: 10),
            TextButton(
              onPressed: () {},
              child: const Text("Forgot password?",
style: TextStyle(color: Colors.white)),
            ),
            const SizedBox(height: 10),
            Row(
              mainAxisAlignment:
MainAxisAlignment.center,
              children: [
                const Text("Don't have an account?",
style: TextStyle(color: Colors.white)),
                TextButton(
                  onPressed: () {
                    Navigator.push(context,
MaterialPageRoute(builder: (_) => const
SignUpScreen()));
                  },
                  child: const Text("Sign Up", style:
TextStyle(fontWeight: FontWeight.bold, color:
Colors.white)),
                ),
              ],
            ),
          ],
        ),
      ),
    ),
  );
 }

  Widget _buildTextField(IconData icon, String
hint, {bool isPassword = false, required
TextEditingController controller}) {
   return TextField(
     controller: controller,
     obscureText: isPassword,
     style: const TextStyle(color: Colors.white),
     decoration: InputDecoration(
       prefixIcon: Icon(icon, color: Colors.white),
       hintText: hint,
       hintStyle: const TextStyle(color:
Colors.white70),
       filled: true,
       fillColor: Colors.white.withOpacity(0.2),
       border: OutlineInputBorder(borderRadius:
BorderRadius.circular(30), borderSide:
BorderSide.none),
     ),
   );
 }
}
```

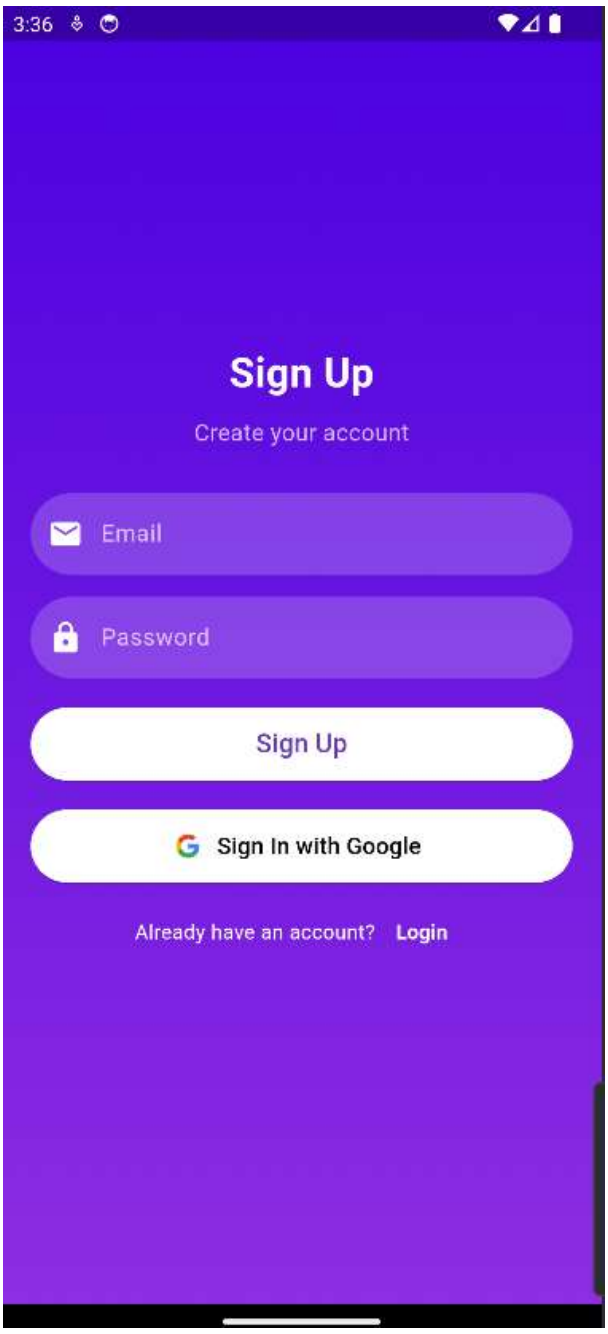
Code:Signup_screen.dart

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        decoration: const BoxDecoration(
          gradient: LinearGradient(
            begin: Alignment.topCenter,
            end: Alignment.bottomCenter,
            colors: [Color(0xFF4A00E0), Color(0xFF8E2DE2)],
          ),
        ),
        child: Center(
          child: Padding(
            padding: const EdgeInsets.all(20.0),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                const Text(
                  "Sign Up",
                  style: TextStyle(fontSize: 28, fontWeight: FontWeight.bold, color: Colors.white),
                ),
                const SizedBox(height: 10),
                const Text(
                  "Create your account",
                  style: TextStyle(fontSize: 16, color: Colors.white70),
                ),
                const SizedBox(height: 30),
                _buildTextField(Icons.email, "Email"),
                const SizedBox(height: 15),
                _buildTextField(Icons.lock, "Password", isPassword: true),
                const SizedBox(height: 20),
                ElevatedButton(
                  onPressed: () {}, // Add Firebase signup logic
                  style: ElevatedButton.styleFrom(
                    backgroundColor: Colors.white,
                    foregroundColor: Colors.deepPurple,
                    minimumSize: const Size(double.infinity, 50),
                    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(30)),
                  ),
                  child: const Text("Sign Up", style: TextStyle(fontSize: 18)),
                ),
                const SizedBox(height: 20),
                _googleSignInButton(),
                const SizedBox(height: 10),
                Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    const Text("Already have an account?", style: TextStyle(color: Colors.white)),
                    TextButton(
                      onPressed: () {
                        Navigator.push(context, MaterialPageRoute(builder: (_) => const LoginScreen()));
                      },
                      child: const Text("Login", style: TextStyle(fontWeight: FontWeight.bold, color: Colors.white)),
                    ),
                  ],
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }

  Widget _buildTextField(IconData icon, String hint, {bool isPassword = false}) {
    return TextField(
      obscureText: isPassword,
      style: const TextStyle(color: Colors.white),
      decoration: InputDecoration(
        prefixIcon: Icon(icon, color: Colors.white),
        hintText: hint,
        hintStyle: const TextStyle(color: Colors.white70),
        filled: true,
        fillColor: Colors.white.withOpacity(0.2),
        border: OutlineInputBorder(borderRadius: BorderRadius.circular(30), borderSide:
```

```
BorderSide.none),
      ),
    );
  }

  Widget _googleSignInButton() {
    return ElevatedButton.icon(
      onPressed: () {}, // Implement Google Sign-In
      style: ElevatedButton.styleFrom(
        backgroundColor: Colors.white,
        foregroundColor: Colors.black,
        minimumSize: const Size(double.infinity, 50),
        shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(30)),
      ),
      icon: Image.asset("assets/images/google.png",
height: 24), // Add Google logo in assets
      label: const Text("Sign In with Google", style:
TextStyle(fontSize: 16)),
    );
  }
}
```

OUTPUT: -


3:32
Welcome Back
Enter your credentials to login
Email
Password
Login
Forgot password?
Don't have an account? Sign Up


3:36
Sign Up
Create your account
Email
Password
Sign Up
Sign In with Google
Already have an account? Login

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	23
Name	Ronak Katariya
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

EXPERIMENT NO: - 05

Name:- Ronak Katariya **Class:-** D15A **Roll:No: -** 23

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

Theory: -

In Flutter, the screens and pages are known as routes, and these routes are just a widget. In Android, a route is similar to an Activity.

In any mobile app, navigating to different pages defines the workflow of the application, and the way to handle the navigation is known as routing. Flutter provides a basic routing class MaterialPageRoute and two methods Navigator.push() and Navigator.pop() that shows how to navigate between two routes. The following steps are required to start navigation in your application.

Gestures enable the app to respond to user interactions, making the application more dynamic and responsive.

- **Navigation and Routing in Flutter**

  Navigation is the process of moving between different screens or pages in an app. Flutter provides a simple and effective way to handle this through the use of the Navigator widget and routes.

  1. Using Navigator Widget

  The Navigator widget manages a stack of routes, allowing for pushing and popping routes on the stack.

  - **Pushing a Route**: To navigate to a new screen, use Navigator.push().
  - **Popping a Route**: To go back to the previous screen, use Navigator.pop().

```
ElevatedButton(
  onPressed: () {
  Navigator.push(
```

```
      context,
      MaterialPageRoute(builder: (context) => SecondScreen()),
    );},
);
```

2. **Named Routes**

   Flutter also allows the use of named routes to navigate, which can make the routing process cleaner, especially in larger applications.

```
MaterialApp(
  initialRoute: '/',
  routes: {
    '/': (context) => HomeScreen(),
    '/second': (context) => SecondScreen(),
  },
);
```

Navigate to the route using Navigator.pushNamed()

```
Navigator.pushNamed(context, '/second');
```

Handling Gestures in Flutter

Gestures refer to user interactions with the app, such as taps, swipes, pinches, and drags. Flutter provides several widgets and gesture detectors to handle these interactions.

Tap Gestures

The most common gesture is the tap, which can be handled using the GestureDetector widget or specific buttons like InkWell or ElevatedButton.

Long Press Gesture

For long press gestures, Flutter provides the onLongPress callback in GestureDetector or InkWell.

Swipe and Drag Gestures

Flutter also provides swipe and drag gesture handling. The onHorizontalDragUpdate and onVerticalDragUpdate callbacks are used for dragging gestures.

Code: -

main.dart

```
import 'package:flutter/material.dart';
import
'package:quiz/screens/login_screen.dart';

void main() {
  runApp(const MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner:
false,
      title: "Login & SignUp",
      home: const LoginScreen(),
    );
  }
}
initialRoute: '/login', // Set initial page
routes: {
  '/login': (context) => LoginPage(),
  '/register': (context) => RegistrationPage(),
  '/home': (context) => HomePage(),
},
);
}
}
```

Login_screen.dart:

```
import 'package:flutter/material.dart';
import 'package:quiz/widgets/bottom_navbar.dart';
import 'package:quiz/screens/category_selection_screen.dart';
import 'package:quiz/screens/leaderboard.dart';
import 'package:quiz/screens/profile.dart';

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

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

class _LandingPageState extends State<LandingPage> {
  int _selectedIndex = 0;

  final List<Widget> _pages = [
    const HomeScreen(), // Landing Page UI
    const CategorySelectionScreen(),
    const LeaderboardPage(),
    const ProfilePage(),
  ];

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
    });
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: _pages[_selectedIndex],
      bottomNavigationBar: BottomNavBar(
        currentIndex: _selectedIndex,
        onTap: _onItemTapped,
      ),
    );
  }
}
class HomeScreen extends StatelessWidget {
  const HomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: HomeContent(),
    );
  }
}
class HomeContent extends StatelessWidget {

  const HomeContent({super.key});
```

```
@override
Widget build(BuildContext context) {
  return Container(
    decoration: const BoxDecoration(
      gradient: LinearGradient(
        colors: [Color(0xFF6A11CB), Color(0xFF2575FC)],
        begin: Alignment.topCenter,
        end: Alignment.bottomCenter,
      ),
    ),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const Padding(
          padding: EdgeInsets.symmetric(horizontal: 20),
          child: Text(
            "Test Your Knowledge 🚀",
            style: TextStyle(
              fontSize: 26,
              fontWeight: FontWeight.bold,
              color: Colors.white,
            ),
            textAlign: TextAlign.center,
          ),
        ),
        const SizedBox(height: 20),
        _buildCategoryCards(context),
        const SizedBox(height: 20),
        ElevatedButton(
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => const CategorySelectionScreen()),
            );
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.white,
            foregroundColor: Colors.blueAccent,
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(30),
            ),
            padding: const EdgeInsets.symmetric(horizontal: 40, vertical: 15),
          ),
          child: const Text("Start Quiz", style: TextStyle(fontSize: 18)),
        ),
      ],
    ),
  );
}
/// ✅ Function to build category cards
Widget _buildCategoryCards(BuildContext context) {
  List<Map<String, String>> categories = [
    {"title": "CNS", "image": "assets/cns.jpg"},
    {"title": "OS", "image": "assets/os.jpg"},
    {"title": "DSA", "image": "assets/dsa.jpg"},
```

```
    {"title": "SQL", "image": "assets/sql.png"},
  ];
  return SizedBox(
    height: 200,
    child: ListView.builder(
      scrollDirection: Axis.horizontal,
      itemCount: categories.length,
      itemBuilder: (context, index) {
        return _categoryCard(categories[index]["title"]!, categories[index]["image"]!, context);
      },
    ),
  );
}
/// ✅ Category Card Widget
Widget _categoryCard(String title, String imagePath, BuildContext context) {
  return GestureDetector(
    onTap: () {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => const CategorySelectionScreen()),
      );
    },
    child: Card(
      margin: const EdgeInsets.symmetric(horizontal: 10),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(20),
      ),
      elevation: 5,
      child: Container(
        width: 150,
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(20),
          image: DecorationImage(
            image: AssetImage(imagePath),
            fit: BoxFit.cover,
          ),
        ),
        child: Center(
          child: Text(
            title,
            style: const TextStyle(
              fontSize: 20,
              fontWeight: FontWeight.bold,
              color: Colors.white,
              backgroundColor: Colors.black54,
            ),
          ),
        ),
      ),
    ),
  );
}
}
```

Landing_page.dart

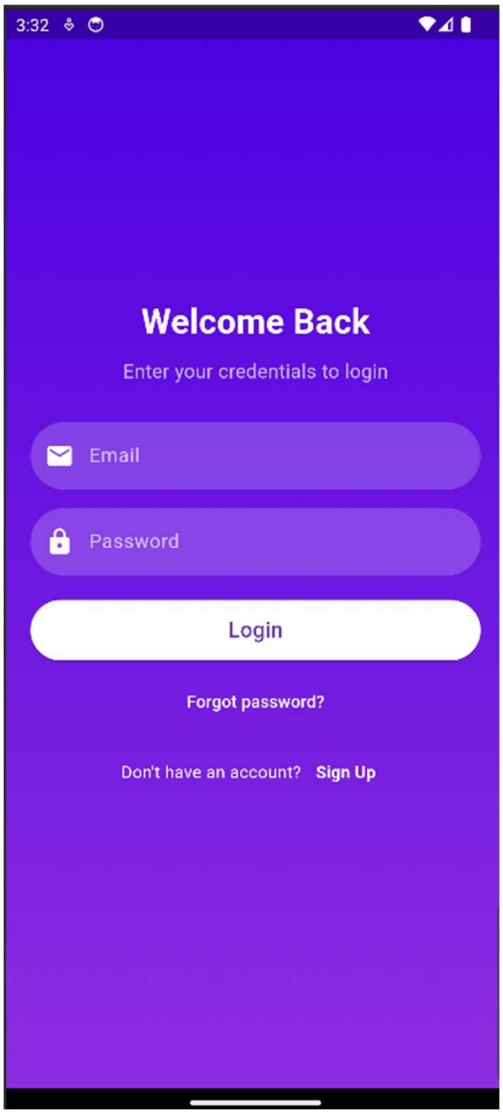
```
import 'package:flutter/material.dart';
import 'package:quiz/widgets/bottom_navbar.dart';
import 'package:quiz/screens/category_selection_screen.dart';
import 'package:quiz/screens/leaderboard.dart';
import 'package:quiz/screens/profile.dart';

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

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

class _LandingPageState extends State<LandingPage> {
  int _selectedIndex = 0;

  final List<Widget> _pages = [
    const HomeScreen(), // Landing Page UI
    const CategorySelectionScreen(),
    const LeaderboardPage(),
    const ProfilePage(),
  ];

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: _pages[_selectedIndex],
      bottomNavigationBar: BottomNavBar(
        currentIndex: _selectedIndex,
        onTap: _onItemTapped,
      ),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: HomeContent(),
    );
  }
}

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

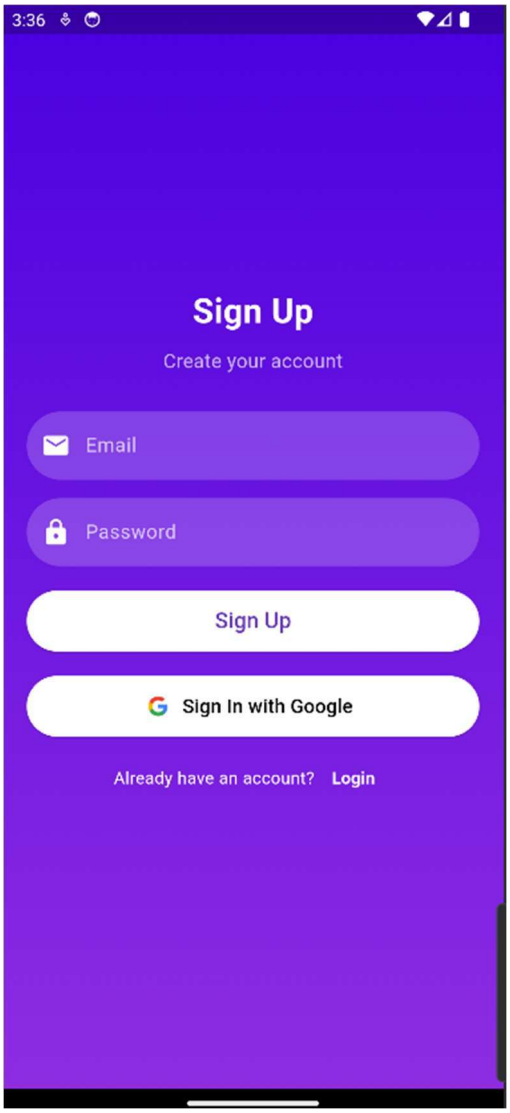
```
@override
Widget build(BuildContext context) {
  return Container(
    decoration: const BoxDecoration(
      gradient: LinearGradient(
        colors: [Color(0xFF6A11CB), Color(0xFF2575FC)],
        begin: Alignment.topCenter,
        end: Alignment.bottomCenter,
      ),
    ),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const Padding(
          padding: EdgeInsets.symmetric(horizontal: 20),
          child: Text(
            "Test Your Knowledge 🚀",
            style: TextStyle(
              fontSize: 26,
              fontWeight: FontWeight.bold,
              color: Colors.white,
            ),
            textAlign: TextAlign.center,
          ),
        ),
        const SizedBox(height: 20),
        _buildCategoryCards(context),
        const SizedBox(height: 20),
        ElevatedButton(
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => const CategorySelectionScreen()),
            );
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.white,
            foregroundColor: Colors.blueAccent,
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(30),
            ),
            padding: const EdgeInsets.symmetric(horizontal: 40, vertical: 15),
          ),
          child: const Text("Start Quiz", style: TextStyle(fontSize: 18)),
        ),
      ],
    ),
  );
}
/// ✅ Function to build category cards
Widget _buildCategoryCards(BuildContext context) {
  List<Map<String, String>> categories = [
    {"title": "CNS", "image": "assets/cns.jpg"},
    {"title": "OS", "image": "assets/os.jpg"},
    {"title": "DSA", "image": "assets/dsa.jpg"},
```

```
    {"title": "SQL", "image": "assets/sql.png"},
  ];

  return SizedBox(
    height: 200,
    child: ListView.builder(
      scrollDirection: Axis.horizontal,
      itemCount: categories.length,
      itemBuilder: (context, index) {
        return _categoryCard(categories[index]["title"]!, categories[index]["image"]!, context);
      },
    ),
  );
}
/// ✅ Category Card Widget
Widget _categoryCard(String title, String imagePath, BuildContext context) {
  return GestureDetector(
    onTap: () {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => const CategorySelectionScreen()),
      );
    },
    child: Card(
      margin: const EdgeInsets.symmetric(horizontal: 10),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(20),
      ),
      elevation: 5,
      child: Container(
        width: 150,
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(20),
          image: DecorationImage(
            image: AssetImage(imagePath),
            fit: BoxFit.cover,
          ),
        ),
        child: Center(
          child: Text(
            title,
            style: const TextStyle(
              fontSize: 20,
              fontWeight: FontWeight.bold,
              color: Colors.white,
              backgroundColor: Colors.black54,
            ),
          ),
        ),
      ),
    ),
  );
 }
}
```

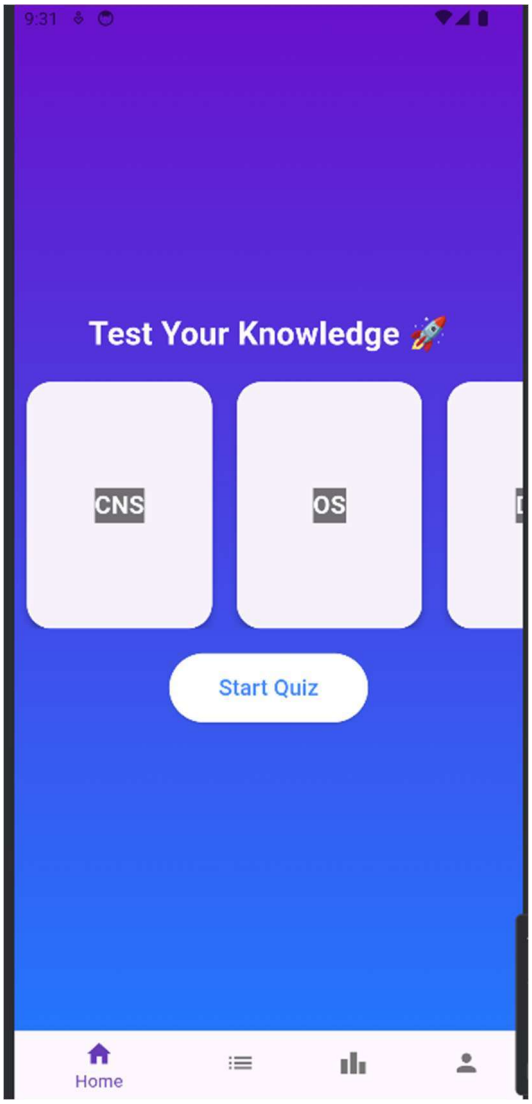
OUTPUT: -

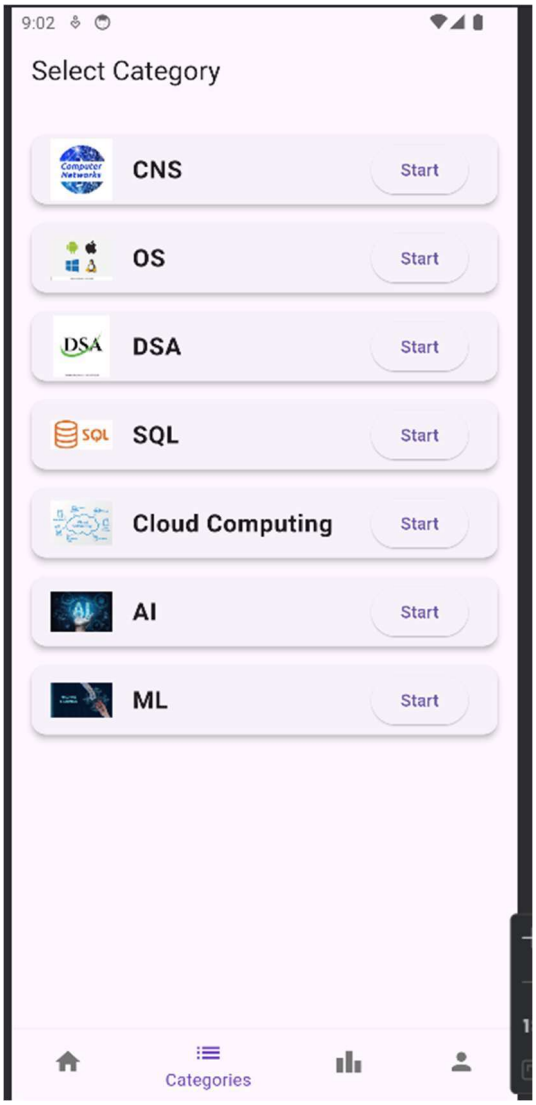
After clicking on Don’t have an account? it navigates to the registration page.




3:36
Sign Up
Create your account
Email
Password
Sign Up
Sign In with Google
Already have an account? Login

In home page, after clicking on Start Quiz it navigates to the Quiz categories.




9:02
Select Category
CNS
Start
OS
Start
DSA
Start
SQL
Start
Cloud Computing
Start
AI
Start
ML
Start
Categories

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	23
Name	Ronak Katariya
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 NO: - 06

Name:- Ronak Katariya **Class:-** D15A **Roll:No: -** 23

AIM: - To connect Flutter UI with Firebase database.

Theory: -

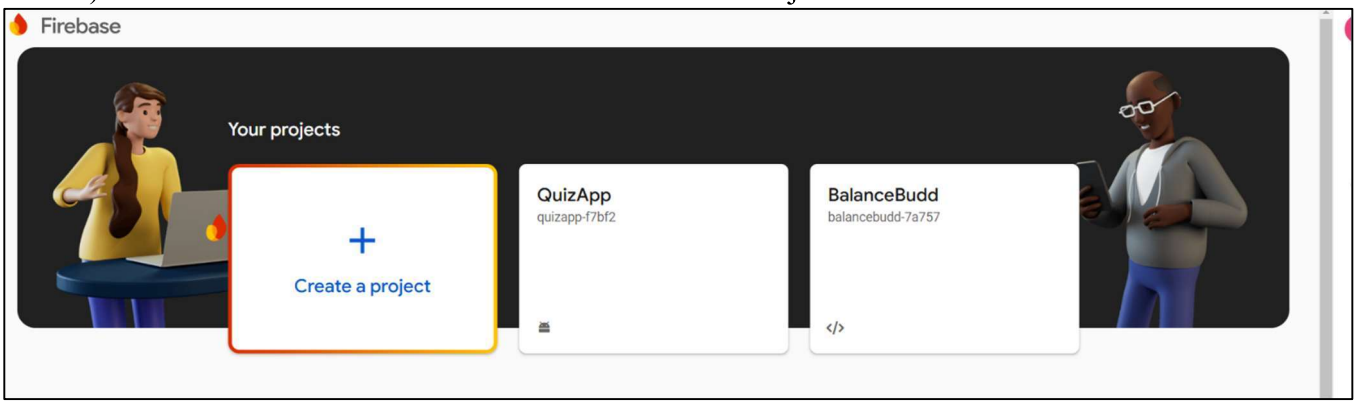
Flutter is an open-source UI toolkit developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. Firebase, a Backend-as-a-Service (BaaS) platform, provides real-time database, authentication, and cloud storage services, making it a powerful backend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

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

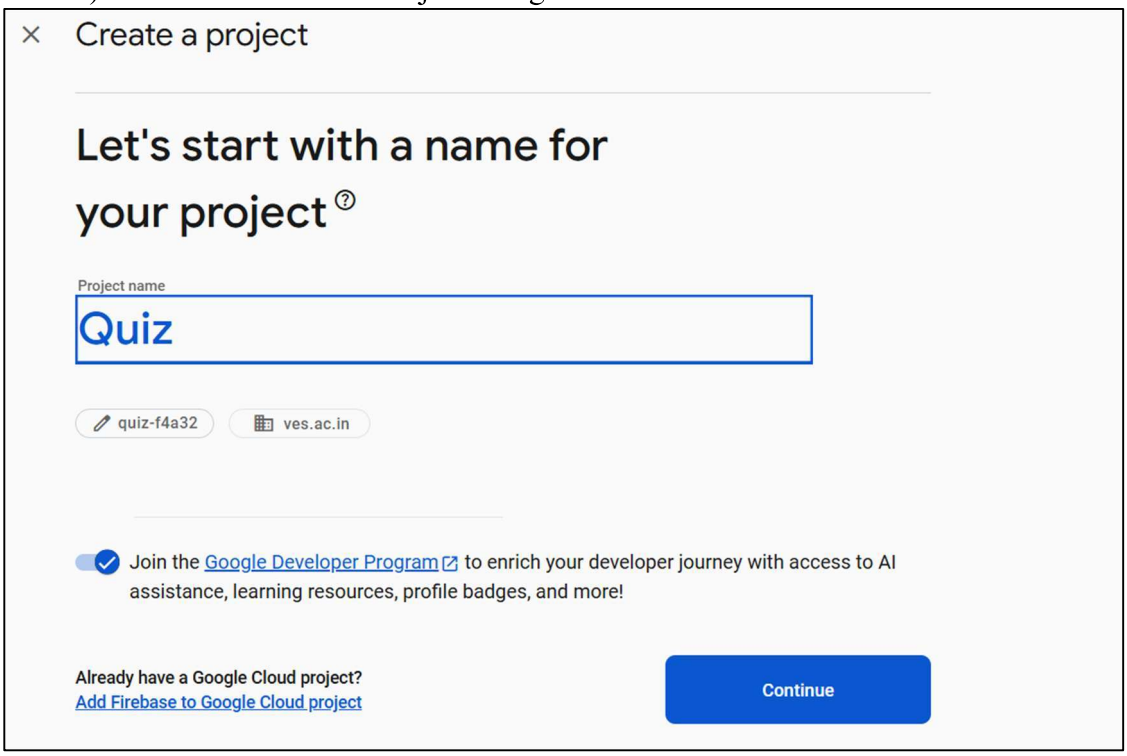
- **Steps to Connect Flutter UI with Firebase Database**

Step 1:

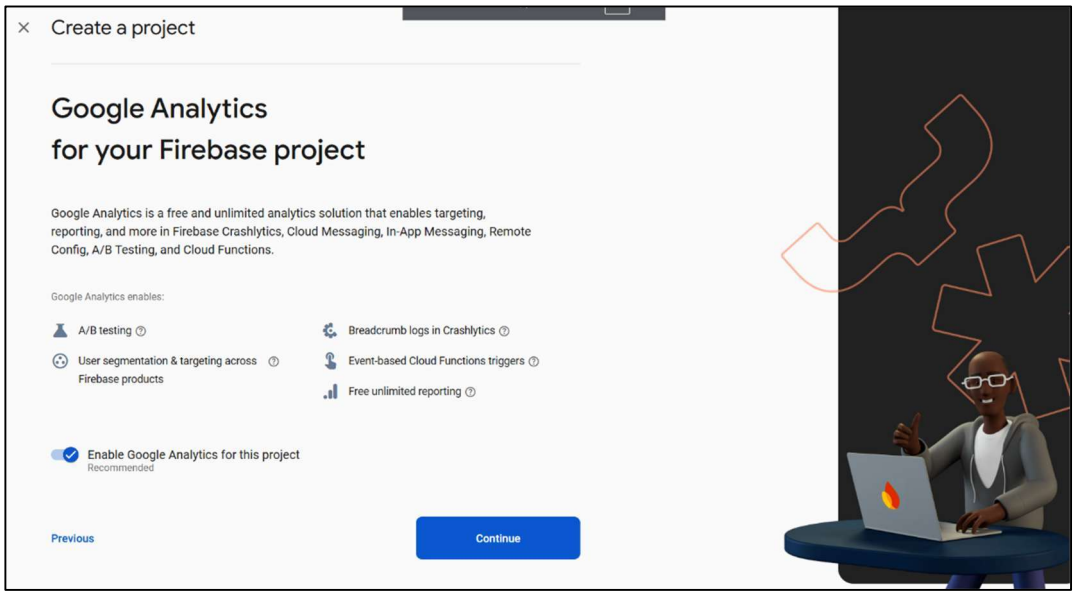
1.1) Go to Firebase Console and Create a Firebase Project



1.2) Click on Create a Project and give it a suitable name.

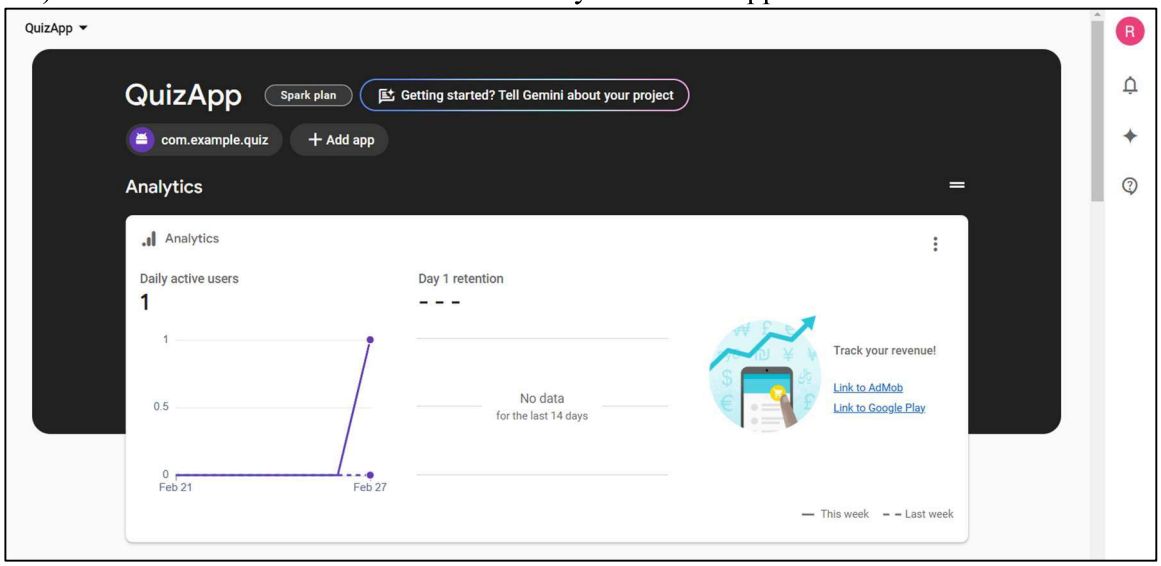


1.3) Enable Google Analytics (optional) & Click continue and complete the setup

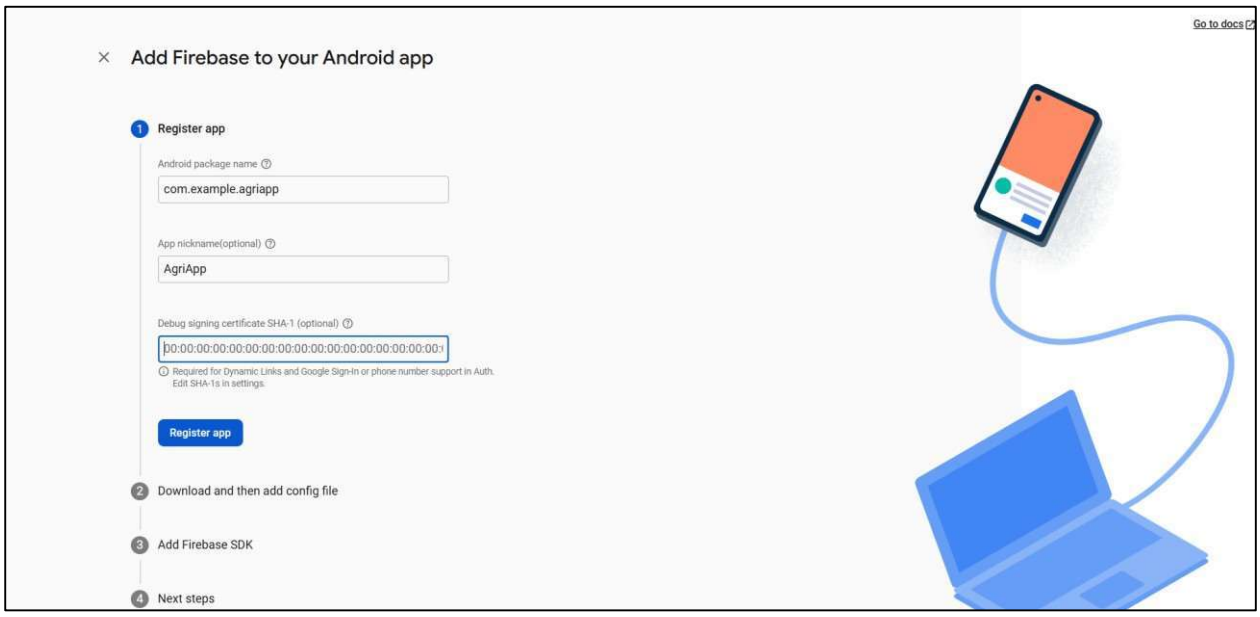


Step 2:- Add Firebase to Your Flutter App

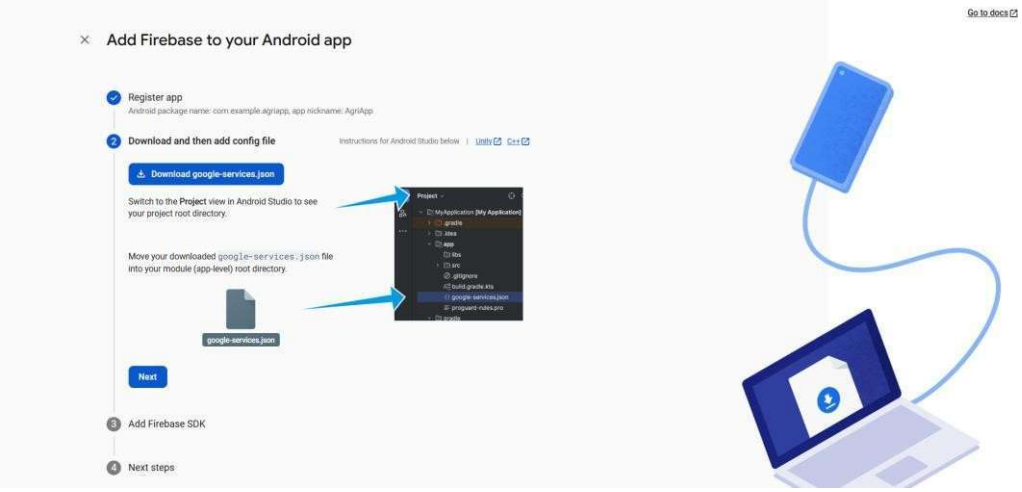
2.1) Click on Android/iOS/Web based on your Flutter application



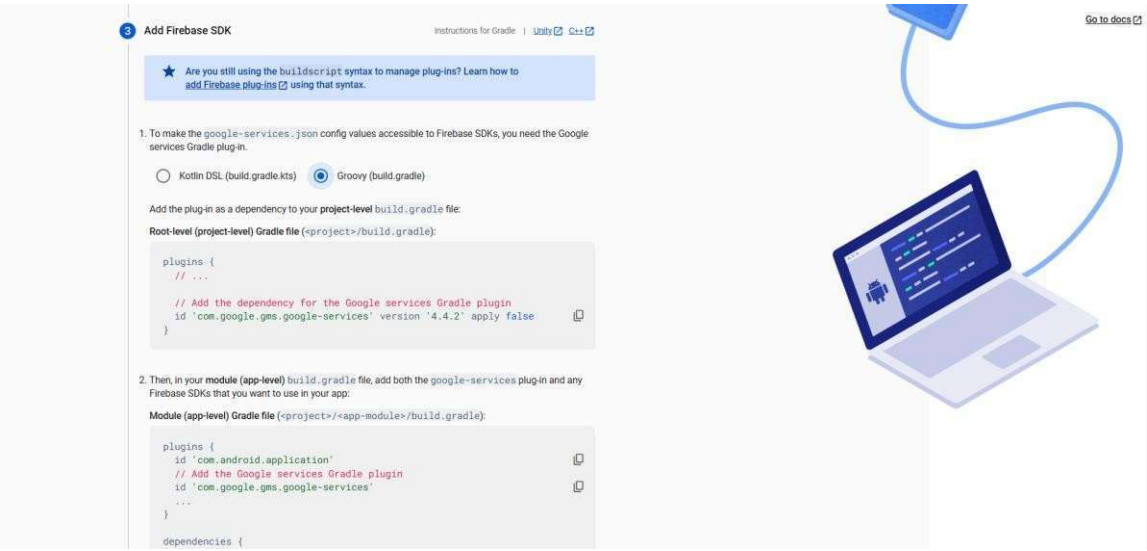
2.2) Register your app with a unique package name (found in android/app/build.gradle for Android).



2.3) Download the google-services.json (for Android) & place the JSON file inside android/app/ directory.



2.4) Add Firebase SDK dependencies to android/build.gradle

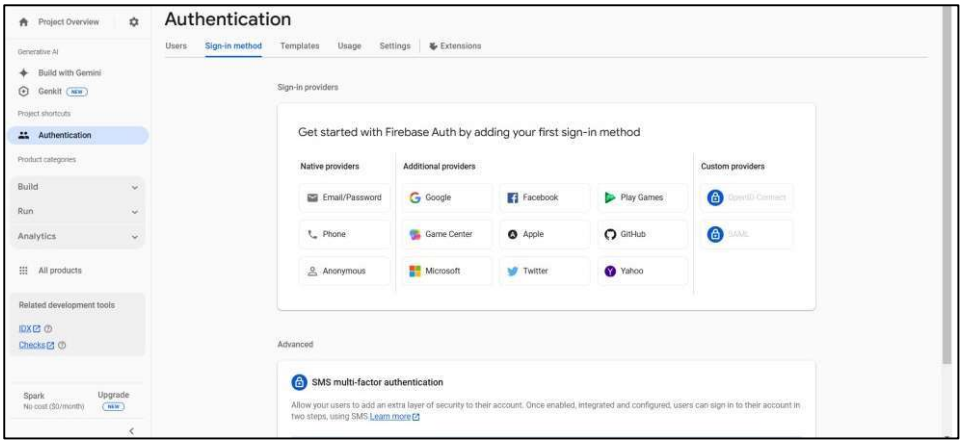


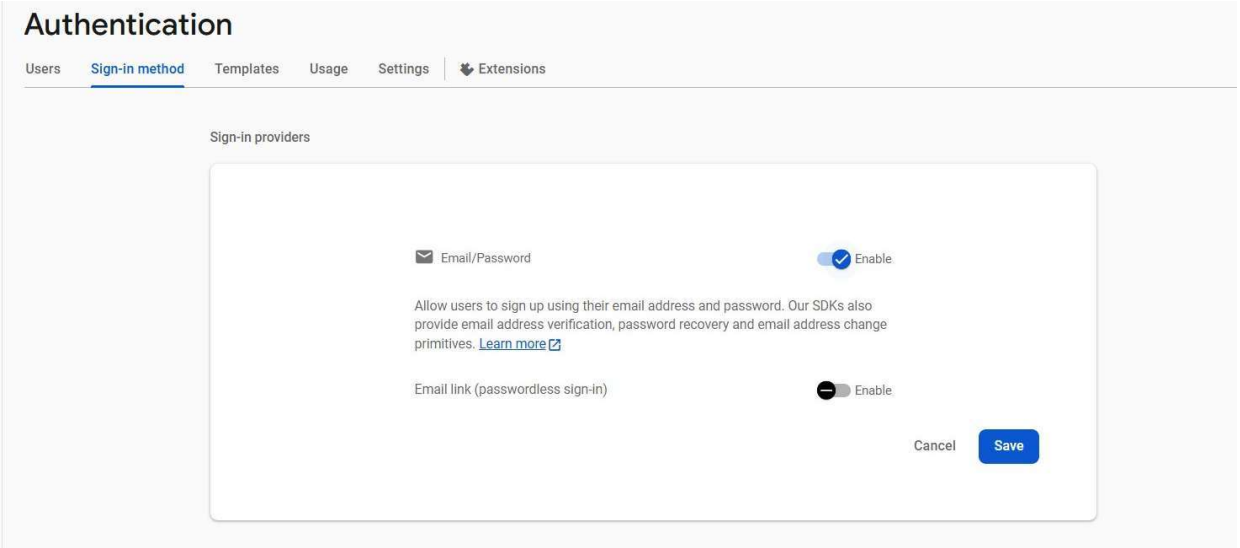
Step 3: - Add Firebase Authentication to Your App

3.1) Add Firebase Authentication Dependencies

```yaml
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^3.11.0
  firebase_auth: ^5.4.2  # For authentication
  cloud_firestore: ^5.6.3  # For Firestore, if you need it
  firebase_messaging: ^15.2.2
  http: ^0.13.3
  image_picker: ^1.0.4
  tflite_flutter: ^0.11.0
  image: ^3.2.0
  url_launcher: ^6.1.14
```

3.2) Enable Authentication in Firebase Console
Go to **Firebase Console → Authentication**.
Click on **Sign-in method** and enable **Email/Password** (or any other method like Google).
Click Save





3.3) Implement Authentication in Flutter
Modify main.dart

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

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

Step 4: -Configure Firebase Realtime Database

4.1) Go to Firebase Console → Realtime Database.
4.2) Click **Create Database** → Choose location → Set rules (for development, set read/write to true).
4.3) Click **Publish**.

Code:

Signup_screen.dart

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

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

  @override
  State<SignUpScreen> createState() =>
_SignUpScreenState();
}

class _SignUpScreenState extends
State<SignUpScreen> {
  final TextEditingController _emailController =
TextEditingController();
  final TextEditingController
_passwordController = TextEditingController();
  bool _isLoading = false;

  Future<void> _signUp(BuildContext context)
async {
    setState(() => _isLoading = true);

    try {
      await
FirebaseAuth.instance.createUserWithEmailAn
dPassword(
        email: _emailController.text.trim(),
        password: _passwordController.text.trim(),
      );
      Navigator.pushReplacement(context,
MaterialPageRoute(builder: (_) => const
LoginScreen()));
    } on FirebaseAuthException catch (e) {

ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(e.message ??
"Signup failed")),
      );
    }

    setState(() => _isLoading = false);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        decoration: const BoxDecoration(
          gradient: LinearGradient(colors:
[Color(0xFF4A00E0), Color(0xFF8E2DE2)],
begin: Alignment.topCenter, end:
Alignment.bottomCenter),
        ),
        child: Center(
          child: Padding(
            padding: const EdgeInsets.all(20.0),
            child: Column(
              mainAxisAlignment:
MainAxisAlignment.center,
              children: [
                _buildTextField(Icons.email, "Email",
controller: _emailController),
                const SizedBox(height: 15),
                _buildTextField(Icons.lock, "Password",
isPassword: true, controller: _passwordController),
                const SizedBox(height: 20),
                ElevatedButton(
                  onPressed: _isLoading ? null : () =>
_signUp(context),
                  style: _buttonStyle(),
                  child: _isLoading ? const
CircularProgressIndicator(color:
Colors.deepPurple) : const Text("Sign Up", style:
TextStyle(fontSize: 18)),
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }

  Widget _buildTextField(IconData icon, String
hint, {bool isPassword = false, required
TextEditingController controller}) {
```

```
 return TextField(controller: controller,
obscureText: isPassword, decoration:
InputDecoration(prefixIcon: Icon(icon)));
  }

  ButtonStyle _buttonStyle() =>
ElevatedButton.styleFrom(minimumSize: const
Size(double.infinity, 50));
}
```

Login_screen.dart:

```
import 'package:flutter/material.dart';
import
'package:firebase_auth/firebase_auth.dart';
import
'package:google_sign_in/google_sign_in.dart';
import 'signup_screen.dart';
import 'landing_page.dart';

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

  @override
  State<LoginScreen> createState() =>
_LoginScreenState();
}

class _LoginScreenState extends
State<LoginScreen> {
  final TextEditingController _emailController =
TextEditingController();
  final TextEditingController
_passwordController = TextEditingController();
  bool _isLoading = false;

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  Future<void> _login(BuildContext context)
async {
    setState(() => _isLoading = true);
  try {
      await
FirebaseAuth.instance.signInWithEmailAndPasswo
rd(
        email: _emailController.text.trim(),
        password: _passwordController.text.trim(),
      );
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => const
LandingPage()),
      );
    } on FirebaseAuthException catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(e.message ?? "Login
failed")),
      );
    }

    setState(() => _isLoading = false);
  }

  Future<void> _signInWithGoogle(BuildContext
context) async {
    try {
      final GoogleSignInAccount? googleUser =
await GoogleSignIn().signIn();
      if (googleUser == null) return;

      final GoogleSignInAuthentication googleAuth =
await googleUser.authentication;
      final credential =
GoogleAuthProvider.credential(
        accessToken: googleAuth.accessToken,
        idToken: googleAuth.idToken,
      );
```

```
  await
FirebaseAuth.instance.signInWithCredential(cre
dential);
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) =>
const LandingPage()),
      );
    } catch (e) {

ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Google Sign-In
failed")),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        decoration: const BoxDecoration(
          gradient: LinearGradient(
            begin: Alignment.topCenter,
            end: Alignment.bottomCenter,
            colors: [Color(0xFF4A00E0),
Color(0xFF8E2DE2)],
          ),
        ),
        child: Center(
          child: Padding(
            padding: const EdgeInsets.all(20.0),
            child: Column(
              mainAxisAlignment:
MainAxisAlignment.center,
              children: [
                const Text(
                  "Welcome Back",
                  style: TextStyle(fontSize: 28,
fontWeight: FontWeight.bold, color:
Colors.white),
                ),
                const SizedBox(height: 10),
                const Text(
                  "Enter your credentials to login",
                  style: TextStyle(fontSize: 16, color:
Colors.white70),
                ),
                const SizedBox(height: 30),
                _buildTextField(Icons.email, "Email",
controller: _emailController),
                const SizedBox(height: 15),
                _buildTextField(Icons.lock, "Password",
isPassword: true, controller: _passwordController),
                const SizedBox(height: 20),
                ElevatedButton(
                  onPressed: _isLoading ? null : () =>
_login(context),
                  style: _buttonStyle(),
                  child: _isLoading
                      ? const
CircularProgressIndicator(color:
Colors.deepPurple)
                      : const Text("Login", style:
TextStyle(fontSize: 18)),
                ),
                const SizedBox(height: 10),
                _googleSignInButton(),
                const SizedBox(height: 10),
                Row(
                  mainAxisAlignment:
MainAxisAlignment.center,
                  children: [
                    const Text("Don't have an account?",
style: TextStyle(color: Colors.white)),
                    TextButton(
                      onPressed: () =>
Navigator.push(context,
MaterialPageRoute(builder: (_) => const
SignUpScreen())),
                      child: const Text("Sign Up", style:
TextStyle(fontWeight: FontWeight.bold, color:
Colors.white)),
                    ),
                  ],
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }

  Widget _buildTextField(IconData icon, String
hint, {bool isPassword = false, required
TextEditingController controller}) {
    return TextField(
      controller: controller,
      obscureText: isPassword,
```

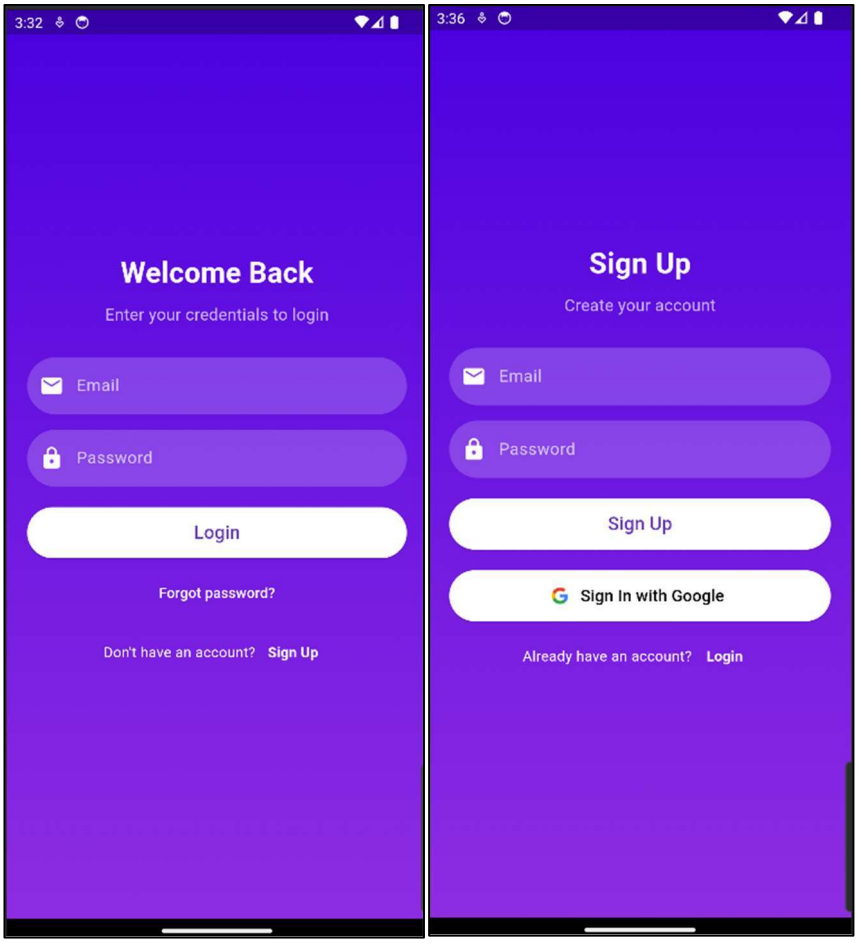
```
    style: const TextStyle(color: Colors.white),
      decoration: InputDecoration(
        prefixIcon: Icon(icon, color: Colors.white),
        hintText: hint,
        hintStyle: const TextStyle(color:
Colors.white70),
        filled: true,
        fillColor: Colors.white.withOpacity(0.2),
        border: OutlineInputBorder(borderRadius:
BorderRadius.circular(30), borderSide:
BorderSide.none),
      ),
    );
  }

  Widget _googleSignInButton() {
    return ElevatedButton.icon(
      onPressed: () =>
_signInWithGoogle(context),
      style: _buttonStyle(),
      icon:
Image.asset("assets/images/google.png", height:
24),
      label: const Text("Sign In with Google",
style: TextStyle(fontSize: 16)),
    );
  }

  ButtonStyle _buttonStyle() {
    return ElevatedButton.styleFrom(
      backgroundColor: Colors.white,
      foregroundColor: Colors.deepPurple,
      minimumSize: const Size(double.infinity,
50),
      shape:
RoundedRectangleBorder(borderRadius:
BorderRadius.circular(30)),
    );
  }
}
```

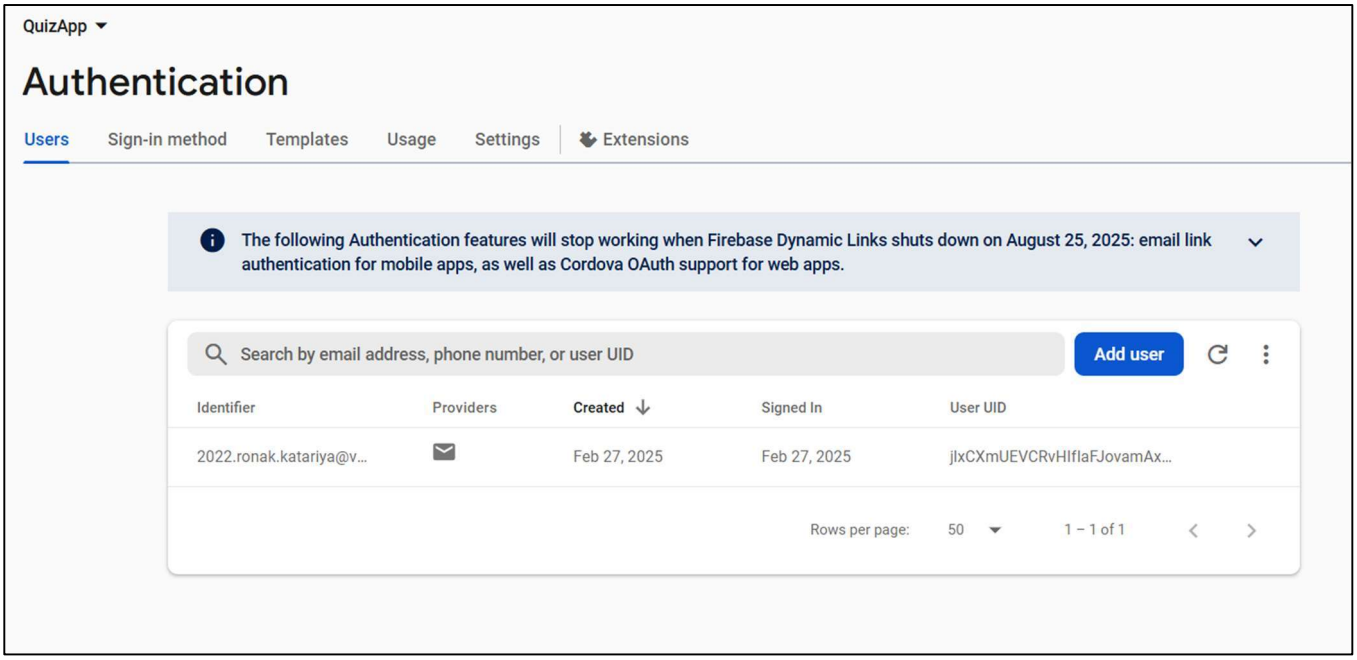
Auth_service.dart

```
import 'package:firebase_auth/firebase_auth.dart';
import
'package:cloud_firestore/cloud_firestore.dart';
import
'package:google_sign_in/google_sign_in.dart';

class AuthService {
  final FirebaseAuth _auth =
FirebaseAuth.instance;
  final FirebaseFirestore _firestore =
FirebaseFirestore.instance;

  // Sign Up
  Future<User?> signUp(String email, String
password) async {
    try {
      UserCredential userCredential = await
_auth.createUserWithEmailAndPassword(
        email: email,
        password: password,
      );
      User? user = userCredential.user;

      // Save user details in Firestore
      await
_firestore.collection('users').doc(user!.uid).set({
        'email': email,
        'createdAt': DateTime.now(),
      });

      return user;
    } catch (e) {
      return null;
    }
  }

  // Sign In
  Future<User?> signIn(String email, String
password) async {
    try {
      UserCredential userCredential = await
_auth.signInWithEmailAndPassword(
        email: email,
        password: password,
      );
      return userCredential.user;
    } catch (e) {
      return null;
    }
  }

  // Google Sign-In
  Future<User?> googleSignIn() async {
    final GoogleSignInAccount? googleUser = await
GoogleSignIn().signIn();
    final GoogleSignInAuthentication? googleAuth =
await googleUser?.authentication;

    if (googleAuth == null) return null;

    final credential = GoogleAuthProvider.credential(
      accessToken: googleAuth.accessToken,
      idToken: googleAuth.idToken,
    );

    UserCredential userCredential = await
_auth.signInWithCredential(credential);
    return userCredential.user;
  }

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

Main.dart

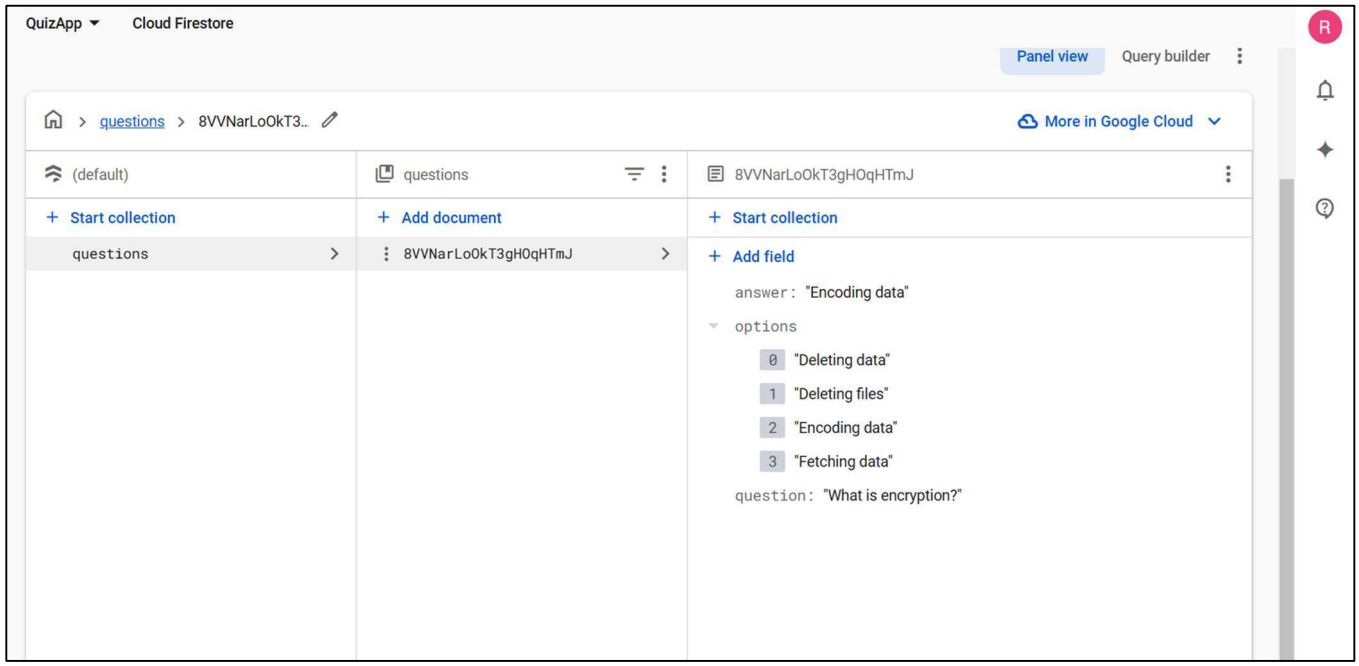
```
import
'package:flutter/material.dart';
import
'package:firebase_core/firebase_core.dart';
import
'package:quiz/screens/landing_page.dart';
import
'package:quiz/screens/category_selection_scree
n.dart';
import 'package:quiz/screens/leaderboard.dart';
import 'package:quiz/screens/profile.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Quiz App',
      theme: ThemeData.dark(),
      home: const LandingPage(),
    );
  }
}
```





After Registering, the user details is saved in the database.

User details get fetched from the database in the profile page.



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.	23
Name	Ronak Katariya
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

Experiment 7: Adding Metadata to Web App Manifest for PWA

Ronak Katariya 23

Prajjwal Pandey 32

Snehal Patil 38

Objective:

To write metadata for an eCommerce Progressive Web App (PWA) in a Web App Manifest file to enable the "Add to Home Screen" 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.).

Requirements:

- A code editor (e.g., VS Code, Sublime Text)
- A basic eCommerce web app
- A browser supporting PWA features (e.g., Chrome, Edge)

Procedure:

1. **Create the Manifest File:**
    - In the root directory of your eCommerce web app, create a file named `manifest.json`.
    - Add the following metadata:

```
{
  "name": "My eCommerce App",
  "short_name": "ShopApp",
  "description": "A fast and secure eCommerce PWA",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#2196F3",
  "icons": [
    {
      "src": "/icons/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

2. **Link the Manifest File to HTML:**

- In the `<head>` section of `index.html`, add:

<link rel="manifest" href="/manifest.json">

4.
5. **Serve the App and Test Installation:**
    - Host the application using a local server (e.g., `Live Server` extension in VS Code or Node.js `http-server`).
    - Open the app in a PWA-compatible browser.
    - Check for the "Add to Home Screen" prompt or manually install it from the browser menu.
6. **Verify in DevTools:**
    - Open Chrome DevTools (`F12` or `Ctrl+Shift+I`).
    - Go to the "Application" tab → "Manifest".
    - Ensure all metadata is correctly loaded.

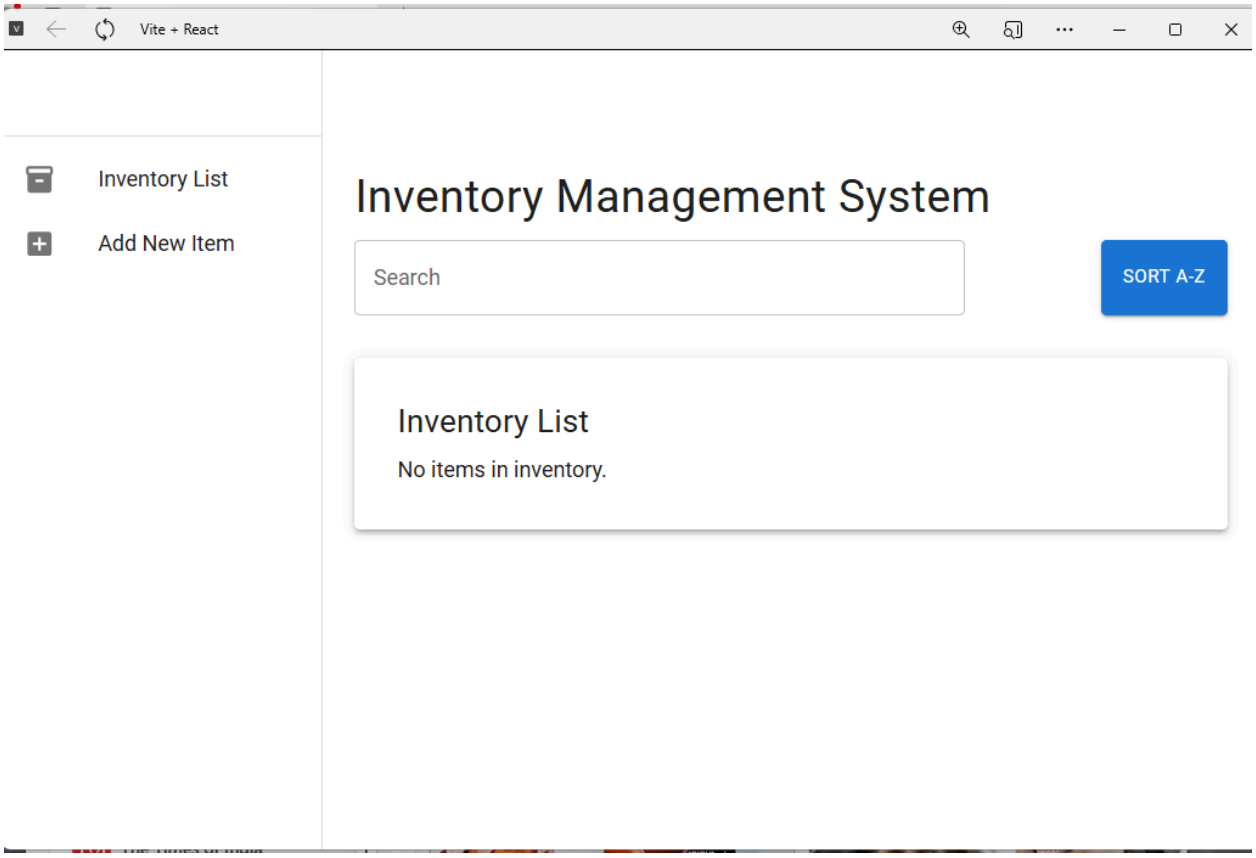
Observations:

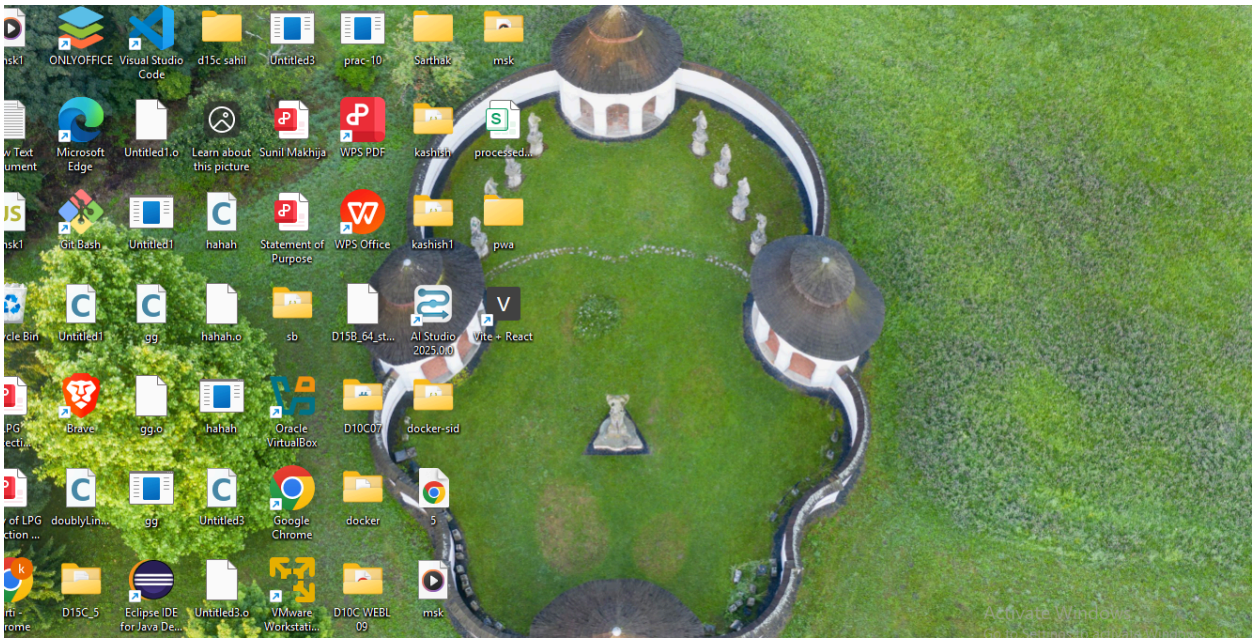
- The manifest file enables the PWA installation.
- The app runs in standalone mode without a browser UI.
- The defined icons and theme colors appear as expected.

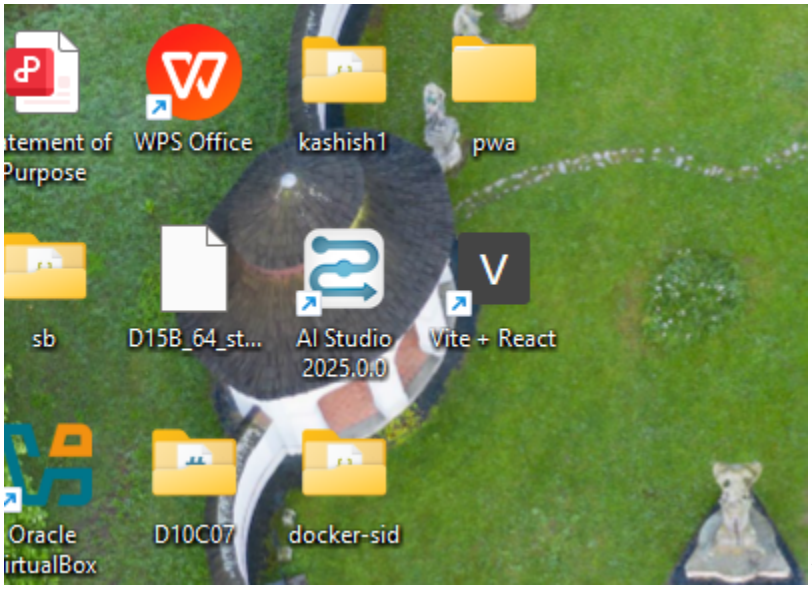
Conclusion:

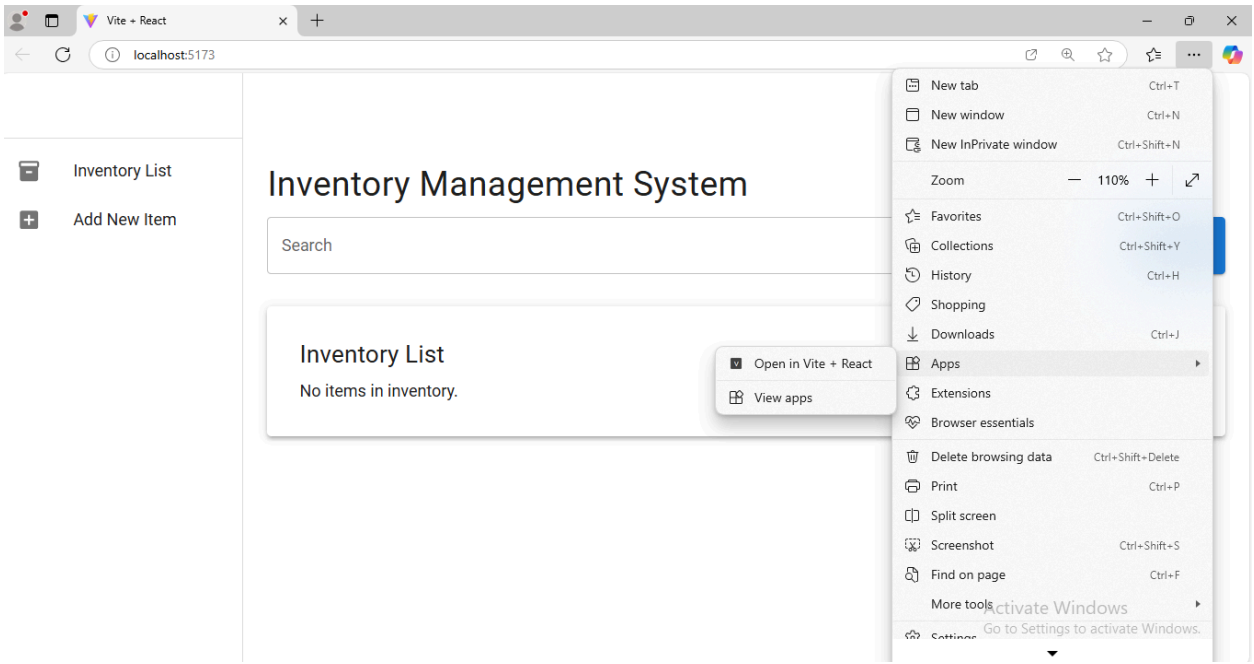
By adding a Web App Manifest file with proper metadata, the eCommerce PWA supports the "Add to Home Screen" feature, improving the user experience with an app-like interface.

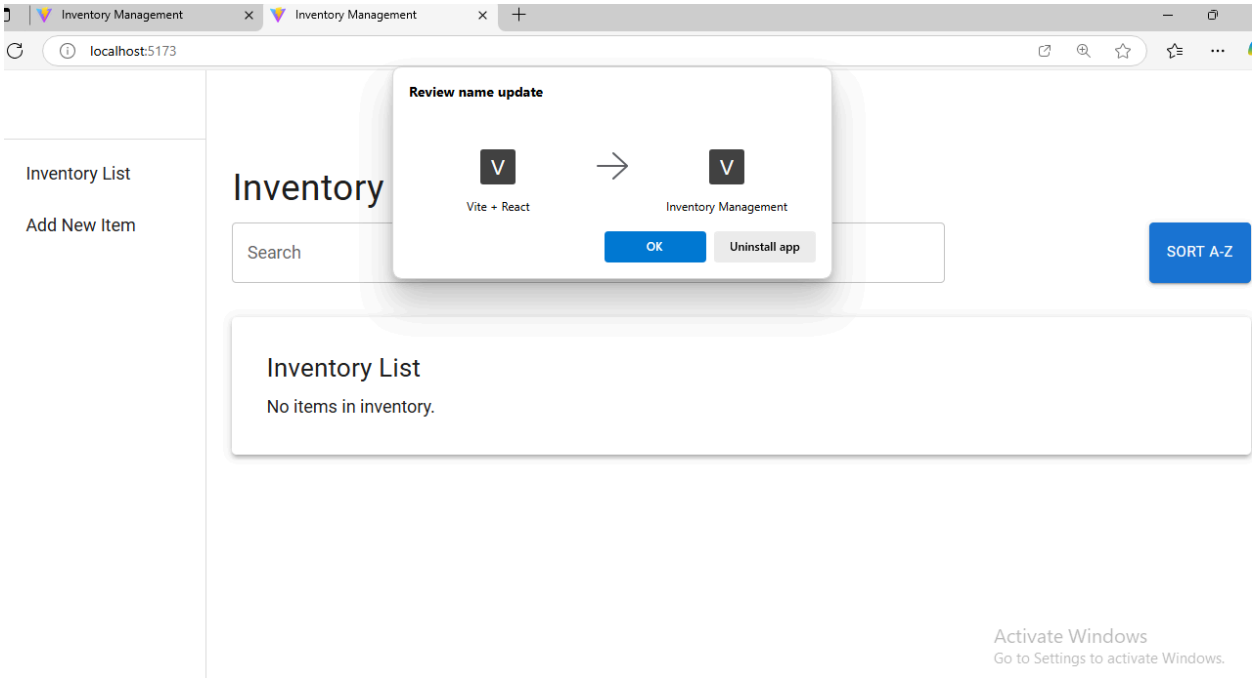
Screenshots:

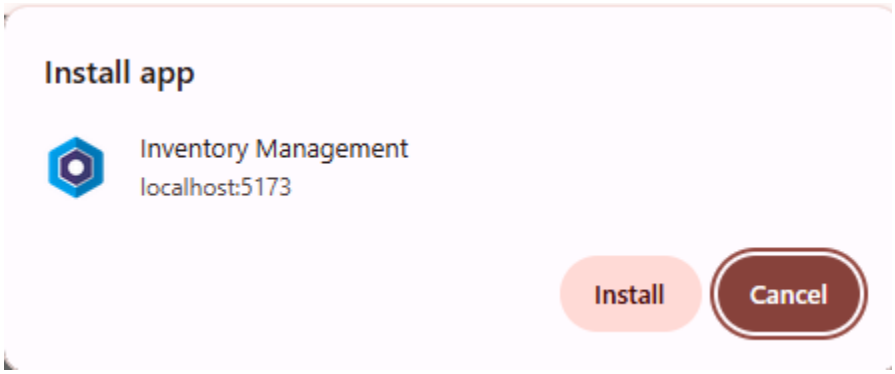


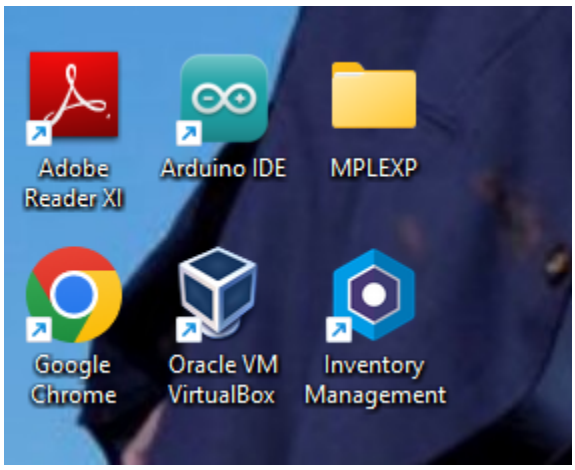



WPS Office
kashish1
pwa
sb
D15B_64_st...
AI Studio 2025.0.0
Vite + React
Oracle VirtualBox
D10C07
docker-sid
\


Vite + React
localhost:5173
Inventory List
Add New Item
Inventory Management System
Search
Inventory List
No items in inventory.
Open in Vite + React
View apps
New tab Ctrl+T
New window Ctrl+N
New InPrivate window Ctrl+Shift+N
Zoom 110%
Favorites Ctrl+Shift+O
Collections Ctrl+Shift+Y
History Ctrl+H
Shopping
Downloads Ctrl+J
Apps
Extensions
Browser essentials
Delete browsing data Ctrl+Shift+Delete
Print Ctrl+P
Split screen
Screenshot Ctrl+Shift+S
Find on page Ctrl+F
More tools
Activate Windows
Go to Settings to activate Windows.


Inventory Management
Inventory Management
localhost:5173
Review name update
Vite + React
Inventory Management
OK
Uninstall app
Inventory List
Add New Item
Inventory
Search
SORT A-Z
Inventory List
No items in inventory.
Activate Windows
Go to Settings to activate Windows.


Install app
Inventory Management
localhost:5173
Install
Cancel


Adobe Reader XI
Arduino IDE
MPLEXP
Google Chrome
Oracle VM VirtualBox
Inventory Management

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.	23
Name	Ronak Katariya
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Ronak Katariya 23
Prajjwal Pandey 32
Snehal Patil 38

Experiment - 8 : Registering and Activating a Service Worker for E-commerce PWA

Objective:
To code and register a service worker and complete the install and activation process for an eCommerce Progressive Web App (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.

A service worker follows three main lifecycle phases:

1. **Installation** - The service worker is downloaded and installed.
2. **Activation** - The service worker takes control of the pages and manages caching.
3. **Fetching and Events Handling** - The service worker intercepts network requests and serves cached responses when necessary.

Requirements:

- A code editor (e.g., VS Code, Sublime Text)
- A basic eCommerce web app
- A browser that supports service workers (e.g., Chrome, Edge, Firefox)

Procedure:

1. Creating the Service Worker File

- In the root directory of the eCommerce PWA, create a file named `service-worker.js`.
- Add the following code to set up basic caching:

```
const CACHE_NAME = 'ecommerce-pwa-v1';
const urlsToCache = [
  '/',
  '/index.html',
  '/styles.css',
  '/script.js',
  '/icons/icon-192x192.png',
  '/icons/icon-512x512.png'
];

// Install event - Caching files
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
      return cache.addAll(urlsToCache);
    })
  );
});

// Activate event - Cleanup old caches
self.addEventListener('activate', (event) => {
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.filter((cache) => cache !== CACHE_NAME).map((cache) =>
caches.delete(cache))
```

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

// Fetch event - Serve files from cache
self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request).then((response) => {
      return response || fetch(event.request);
    })
  );
});
```

---

2. Registering the Service Worker

- In the `index.html` or `app.js` file, register the service worker with the following code:

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

---

3. Testing the Service Worker

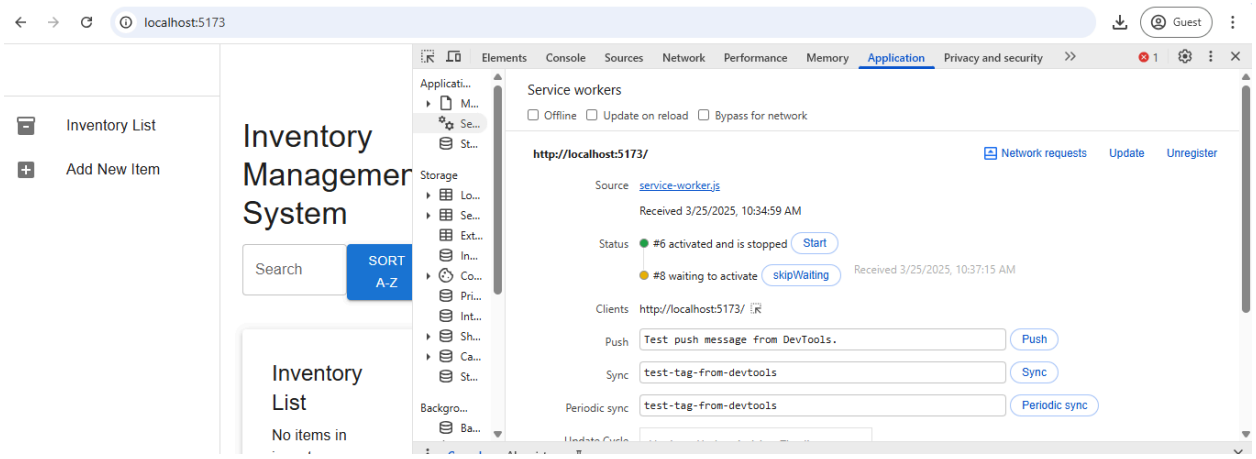
- Host the application on a local server (e.g., using `Live Server` in VS Code or `http-server` in Node.js).
- Open the web application in a browser.
- Open Chrome DevTools (`F12` or `Ctrl+Shift+I`).

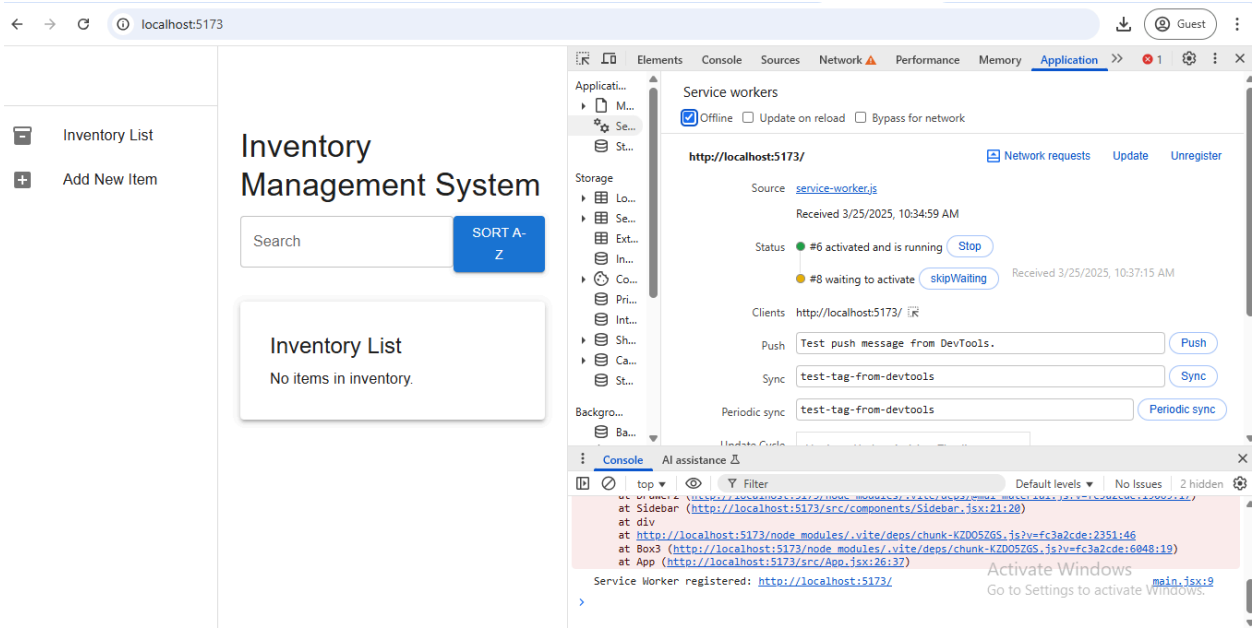
- Go to the "Application" tab → "Service Workers".
- Verify the service worker is installed and activated.

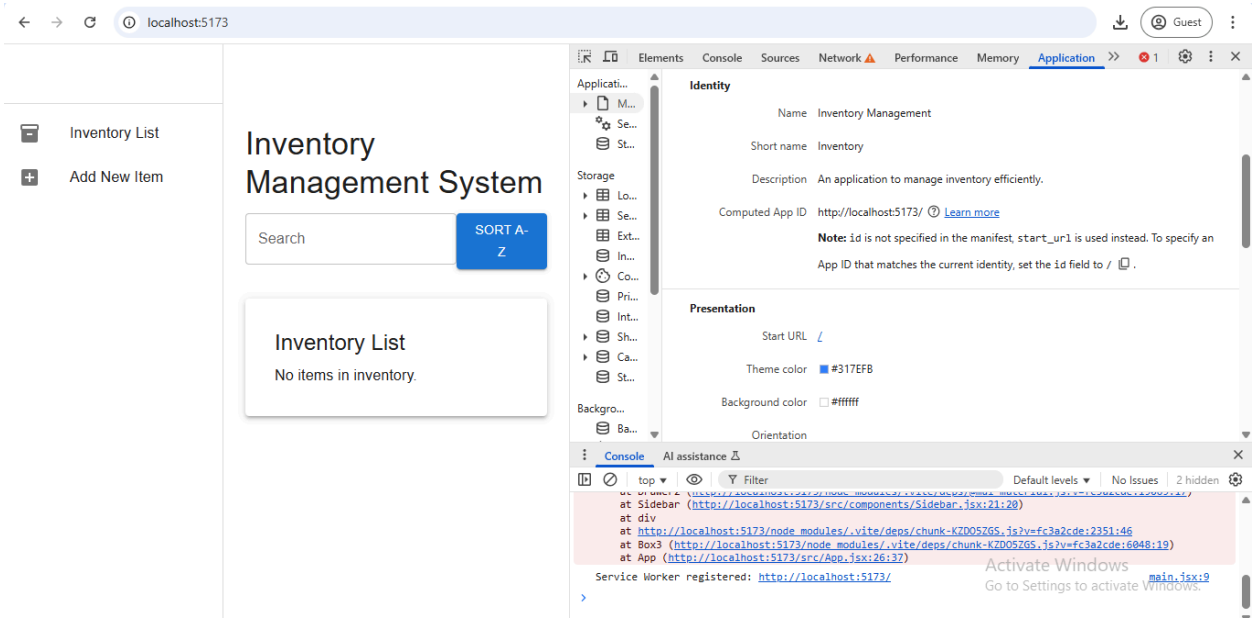
Observations:

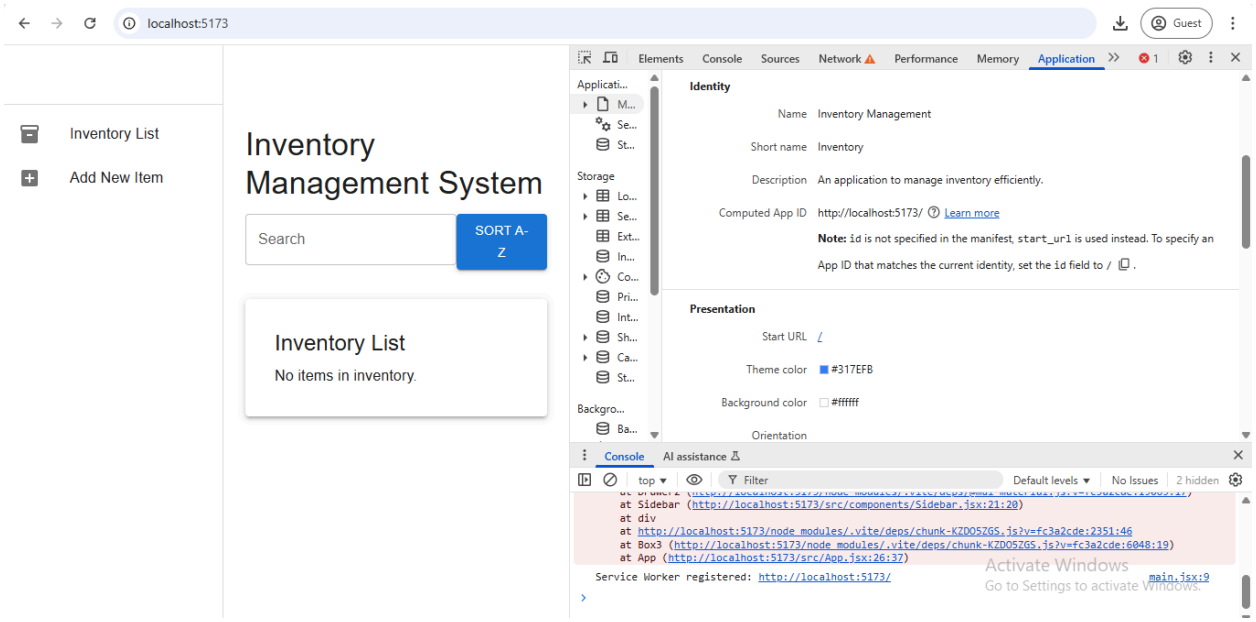
- The service worker successfully caches essential files.
- The app works offline using cached files.
- The activation process ensures that old caches are removed.

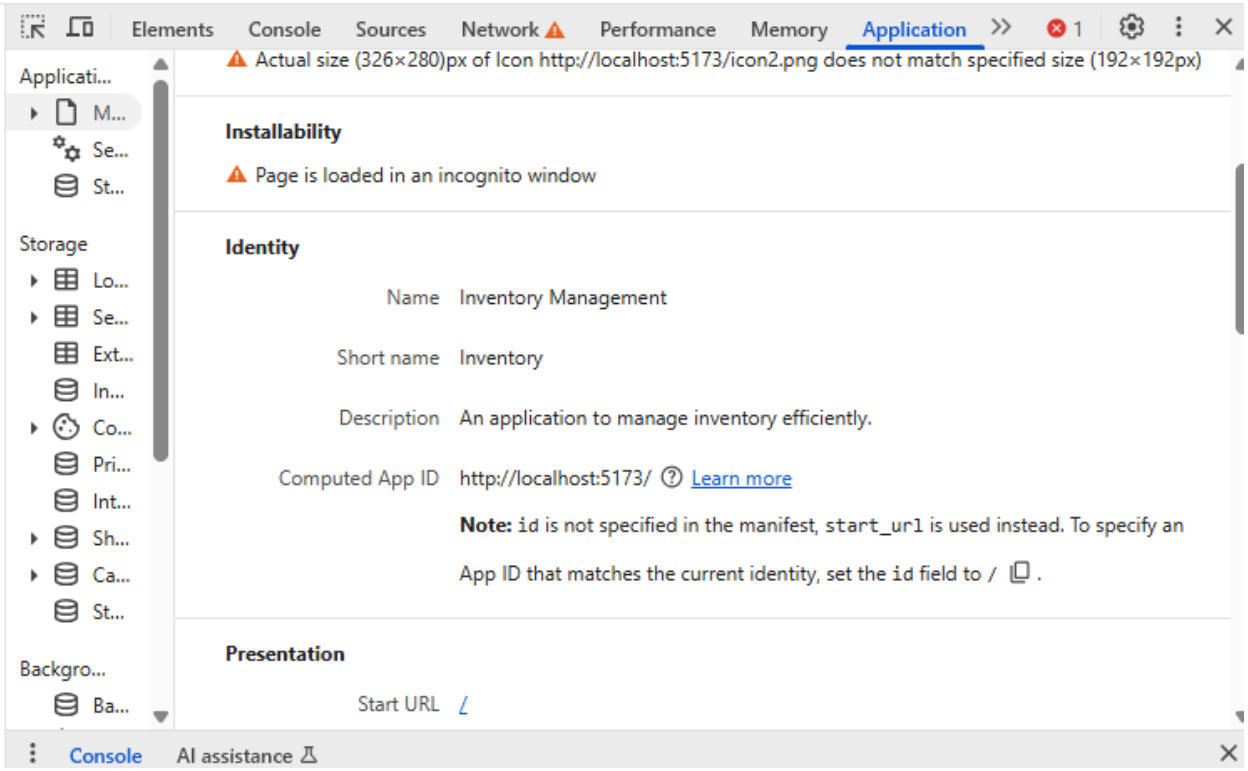
Conclusion:
By coding and registering a service worker, we successfully enabled caching and offline support for the eCommerce PWA, improving performance and reliability.

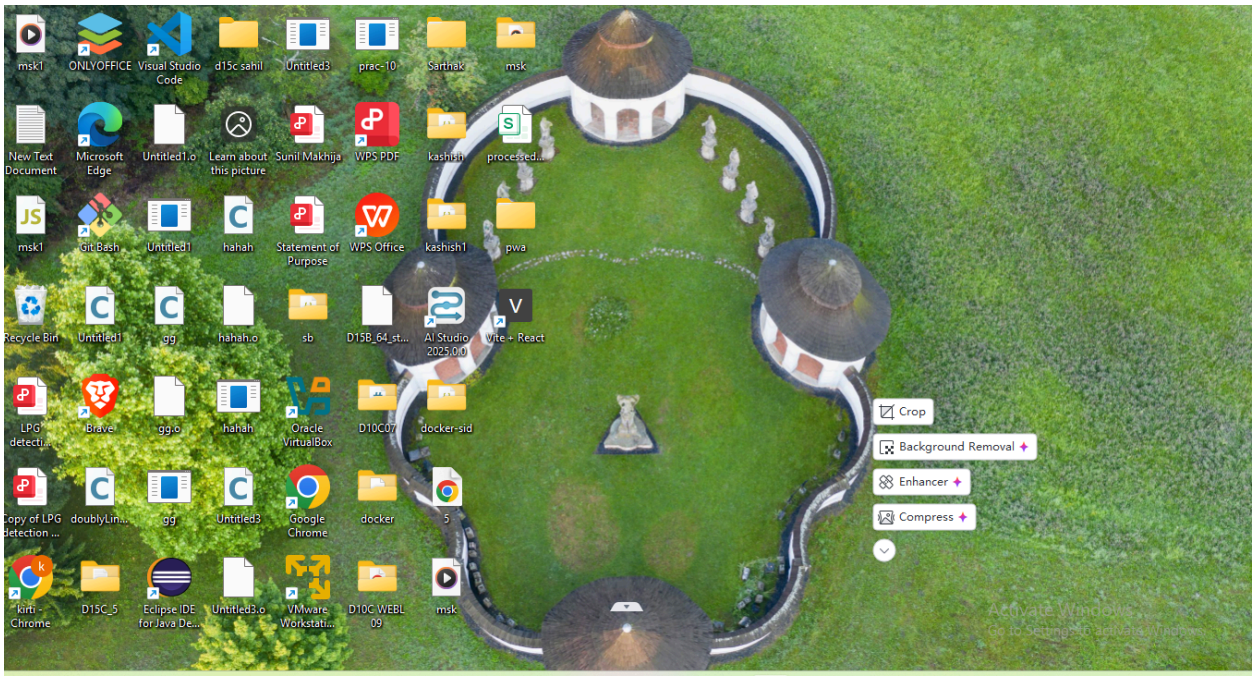



localhost:5173
Guest
Inventory List
Add New Item
Inventory Management System
Search
SORT A-Z
Inventory List
No items in inventory.
Elements
Console
Sources
Network
Performance
Memory
Application
Service workers
Offline
Update on reload
Bypass for network
http://localhost:5173/
Network requests
Update
Unregister
Source
service-worker.js
Received 3/25/2025, 10:34:59 AM
Status
#6 activated and is running
Stop
#8 waiting to activate
skipWaiting
Received 3/25/2025, 10:37:15 AM
Clients
http://localhost:5173/
Push
Test push message from DevTools.
Push
Sync
test-tag-from-devtools
Sync
Periodic sync
test-tag-from-devtools
Periodic sync
Console
AI assistance
Filter
Default levels
No Issues
2 hidden
at Sidebar (http://localhost:5173/src/components/Sidebar.jsx:21:20)
at div
at http://localhost:5173/node_modules/.vite/deps/chunk-KZDO5ZGS.js?v=fc3a2cde:2351:46
at Box3 (http://localhost:5173/node_modules/.vite/deps/chunk-KZDO5ZGS.js?v=fc3a2cde:6048:19)
at App (http://localhost:5173/src/App.jsx:26:37)
Service Worker registered: http://localhost:5173/
main.jsx:9
Activate Windows
Go to Settings to activate Windows.


localhost:5173
Guest
Inventory List
Add New Item
Inventory Management System
Search
SORT A-Z
Inventory List
No items in inventory.
Elements
Console
Sources
Network
Performance
Memory
Application
Identity
Name
Inventory Management
Short name
Inventory
Description
An application to manage inventory efficiently.
Computed App ID
http://localhost:5173/
Learn more
Note: id is not specified in the manifest, start_url is used instead. To specify an App ID that matches the current identity, set the id field to /.
Presentation
Start URL
/
Theme color
#317EFB
Background color
#ffffff
Orientation
Console
AI assistance
Filter
Default levels
No Issues
2 hidden
at Sidebar (http://localhost:5173/src/components/Sidebar.jsx:21:20)
at div
at http://localhost:5173/node_modules/.vite/deps/chunk-KZDO5ZGS.js?v=fc3a2cde:2351:46
at Box3 (http://localhost:5173/node_modules/.vite/deps/chunk-KZDO5ZGS.js?v=fc3a2cde:6048:19)
at App (http://localhost:5173/src/App.jsx:26:37)
Service Worker registered: http://localhost:5173/
main.jsx:9
Activate Windows
Go to Settings to activate Windows.


localhost:5173
Guest
Inventory List
Add New Item
Inventory Management System
Search
SORT A-Z
Inventory List
No items in inventory.
Elements
Console
Sources
Network
Performance
Memory
Application
Identity
Name Inventory Management
Short name Inventory
Description An application to manage inventory efficiently.
Computed App ID http://localhost:5173/ Learn more
Note: id is not specified in the manifest, start_url is used instead. To specify an App ID that matches the current identity, set the id field to /.
Presentation
Start URL /
Theme color #317EFB
Background color #ffffff
Orientation
Console
AI assistance
Service Worker registered: http://localhost:5173/
Activate Windows
Go to Settings to activate Windows.


Elements
Console
Sources
Network
Performance
Memory
Application
Actual size (326×280)px of Icon http://localhost:5173/icon2.png does not match specified size (192×192px)
Installability
Page is loaded in an incognito window
Identity
Name Inventory Management
Short name Inventory
Description An application to manage inventory efficiently.
Computed App ID http://localhost:5173/ Learn more
Note: id is not specified in the manifest, start_url is used instead. To specify an App ID that matches the current identity, set the id field to /.
Presentation
Start URL /
Storage
Backgro...
Console
AI assistance


msk1
ONLYOFFICE
Visual Studio Code
d15c sahil
Untitled3
prac-10
Sarthak
msk
New Text Document
Microsoft Edge
Untitled1.o
Learn about this picture
Sunil Makhija
WPS PDF
kashish
processed...
msk1
Git Bash
Untitled1
hahah
Statement of Purpose
WPS Office
kashish1
pwa
Recycle Bin
Untitled1
gg
hahah.o
sb
D15B_64_st...
AI Studio 2025.0.0
Vite + React
LPG detecti...
Brave
gg.o
hahah
Oracle VirtualBox
D10C07
docker-sid
Copy of LPG detection ...
doublyLin...
gg
Untitled3
Google Chrome
docker
5
kirti - Chrome
D15C_5
Eclipse IDE for Java De...
Untitled3.o
VMware Workstati...
D10C WEBL 09
msk
Crop
Background Removal
Enhancer
Compress
Activate Windows
Go to Settings to activate Windows.

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.	23
Name	Ronak Katariya
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

Ronak Katariya 23
Prajjwal Pandey 32
Snehal Patil 38

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

Theory:
Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

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

- **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
- **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed successfully, the new response will be cached and returned.

But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

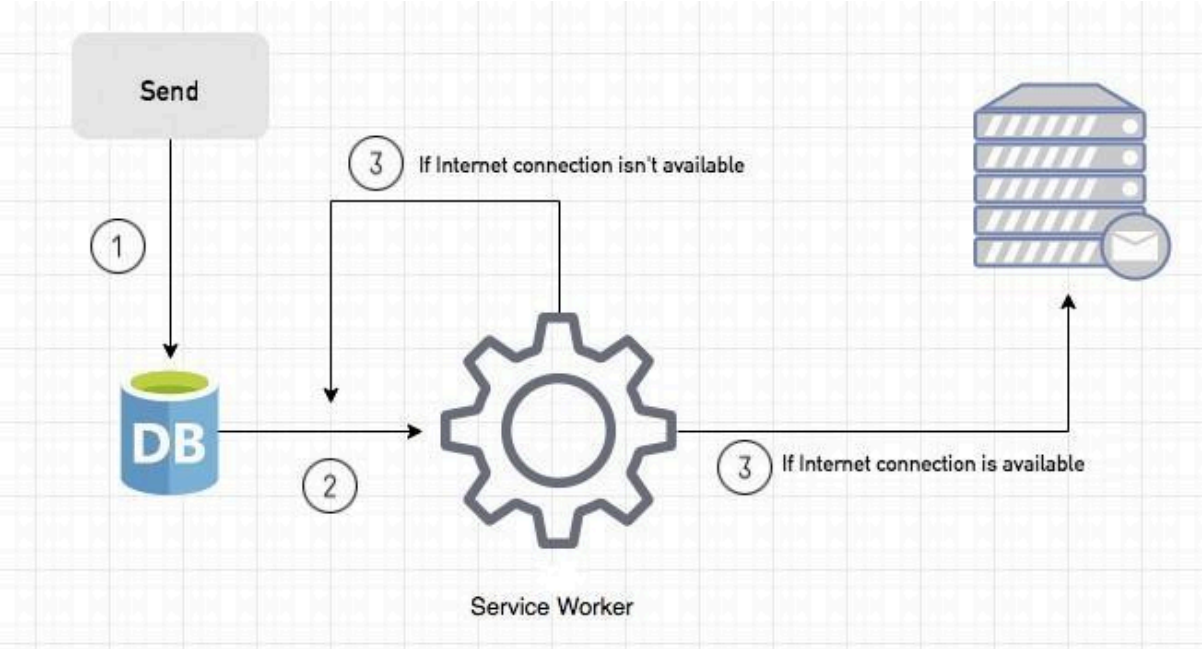
Sync Event

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

Here is a job for the Background Sync.

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

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



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

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

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.

```
self.addEventListener('push', event => {
    if (event && event.data) {
        var data = event.data.json();
        if (data.method === "pushMessage") {
            event.waitUntil(self.registration.showNotification("Test App", {
                body: data.message
            }));
        }
    }
});
```

CODE:

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Stopwatch</title>
    <link rel="stylesheet" href="style.css">
    <link rel="manifest" href="/manifest.json">
  </head>
  <body>
    <div class="container">
      <h1>Stopwatch</h1>
      <p class="time">
        <span id="minutes">00</span>:<span id="seconds">00</span>:<span id="tens">00</span>
      </p>
      <ul id="lapList"></ul>
      <button id="start">Start</button>
      <button id="stop">Stop</button>
      <button id="reset">Reset</button>
      <button id="lap">Lap</button>
    </div>
    <script src="stopwatch.js"></script>
    <script>
      if ('serviceWorker' in navigator) {
        window.addEventListener('load', () => {
          navigator.serviceWorker.register('/service-worker.js')
            .then(registration => {
              console.log('ServiceWorker registered with scope: ', registration.scope);
            })
            .catch(error => {
              console.log('ServiceWorker registration failed: ', error);
            });
        });
      }

      // Request Notification Permission and Show Popup
      if ('Notification' in window) {
        Notification.requestPermission().then(permission => {
          if (permission === 'granted') {
            console.log('Notification permission granted');
            showNotification();
```

```
        }
      });
    }

    function showNotification() {
      if (Notification.permission === 'granted') {
        const options = {
          body: 'Welcome Snehal, Ronak, Prajjwal !',
          icon: '/images/icon.png',
          badge: '/images/icon.png'
        };

        // Display the notification
        new Notification('Welcome Snehal, Ronak, Prajjwal ', options);
      }
    }
  </script>
 </body>
</html>
```

Manifest.json

```
{
  "name": "PWA",
  "short_name": "RPS",
  "description": "A simple and elegant stopwatch application.",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#000000",
  "icons": [
    {
      "src": "/images/icon2.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/images/icon.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
 }
```

serviceworker.js

```
const CACHE_NAME = 'prodigy-stopwatch-cache-v1';
const urlsToCache = [
  '/',
  '/index.html',
  '/style.css',
  '/stopwatch.js',
  '/images/icon2.png',
  '/images/icon.png'
];

// Install Service Worker and Cache Files
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        console.log('Opened cache');
        return cache.addAll(urlsToCache);
      })
  );
});

// Cache and Return Requests
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request).then(response => {
      return response || fetch(event.request)
        .then(networkResponse => {
          if (!networkResponse || networkResponse.status !== 200) {
            return networkResponse;
          }
          return caches.open(CACHE_NAME).then(cache => {
            cache.put(event.request, networkResponse.clone());
            return networkResponse;
          });
        }).catch(() => {
          if (event.request.mode === 'navigate') {
            return caches.match('/index.html');
          }
        });
    })
  );
});
```

```
// Activate & Remove Old Caches
self.addEventListener('activate', event => {
  const cacheWhitelist = [CACHE_NAME];
  event.waitUntil(
    caches.keys().then(cacheNames => {
      return Promise.all(
        cacheNames.map(cacheName => {
          if (!cacheWhitelist.includes(cacheName)) {
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});
```

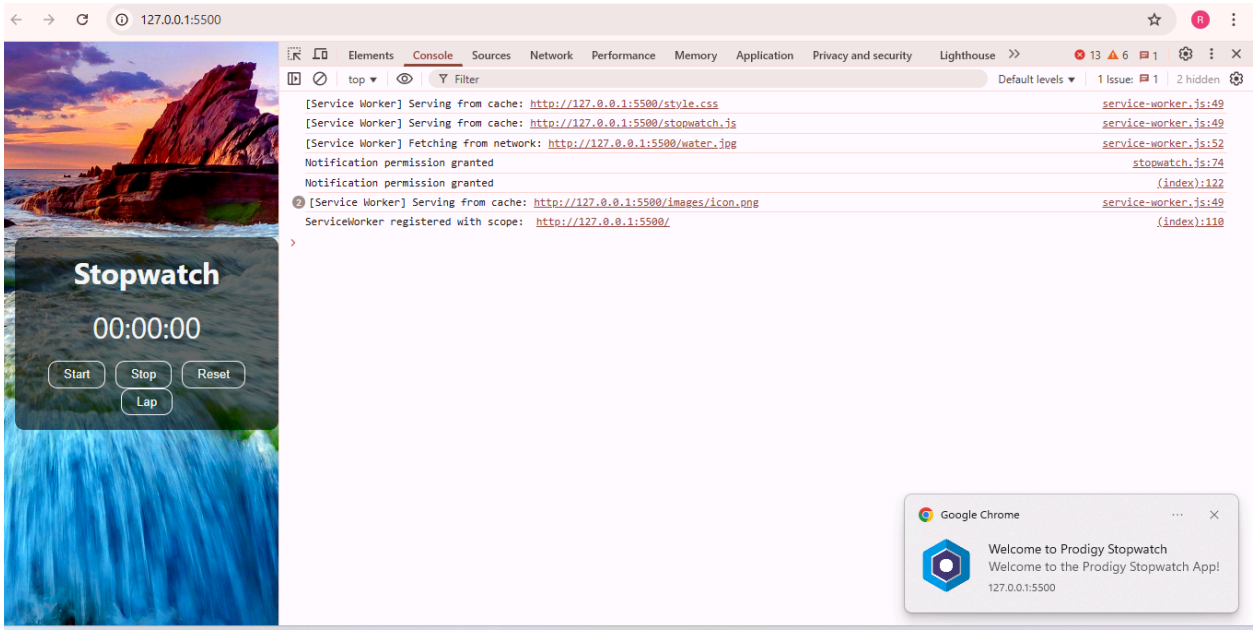
stopwatch.js

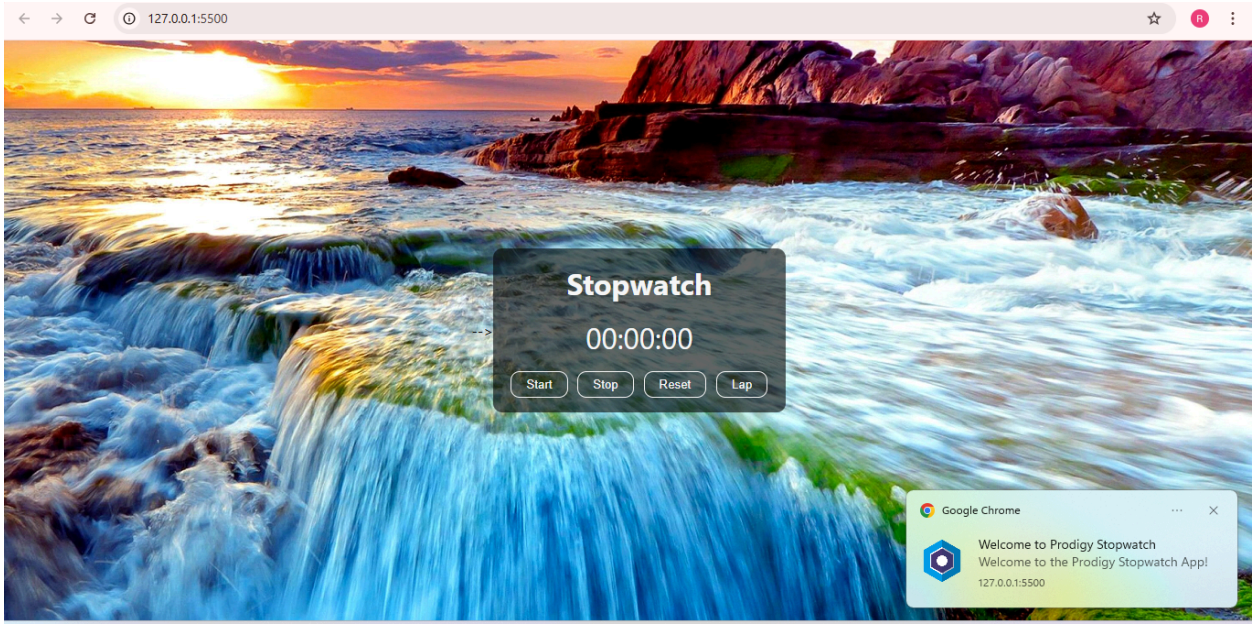
```
window.onload = function () {
    let minutes = 0;
    let seconds = 0;
    let tens = 0;
    let appendMinutes = document.querySelector('#minutes');
    let appendTens = document.querySelector('#tens');
    let appendSeconds = document.querySelector('#seconds');
    let startBtn = document.querySelector('#start');
    let stopBtn = document.querySelector('#stop');
    let resetBtn = document.querySelector('#reset');
    let lapBtn = document.querySelector('#lap');
    let lapList = document.querySelector('#lapList');
    let Interval;

    const startTimer = () => {
      tens++;
      if (tens <= 9) {
        appendTens.innerHTML = '0' + tens;
      }
      if (tens > 9) {
        appendTens.innerHTML = tens;
      }

      if (tens > 99) {
        seconds++;
        appendSeconds.innerHTML = '0' + seconds;
        tens = 0;
```

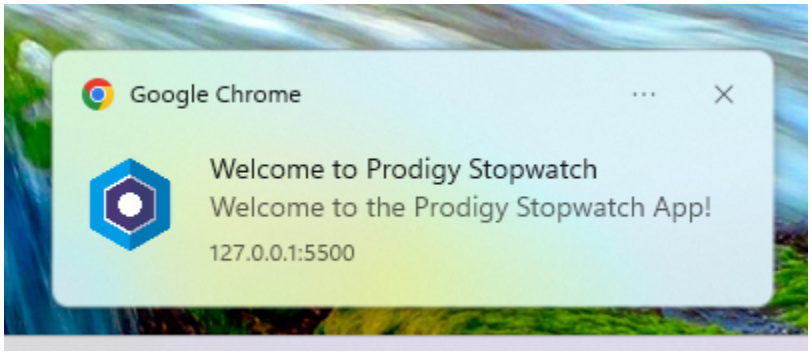
```
      appendTens.innerHTML = '0' + 0;
    }

    if (seconds > 9) {
      appendSeconds.innerHTML = seconds;
    }

    if (seconds > 59) {
      minutes++;
      appendMinutes.innerHTML = '0' + minutes;
      seconds = 0;
      appendSeconds.innerHTML = '0' + 0;
    }
  };

  startBtn.onclick = () => {
    clearInterval(Interval);
    Interval = setInterval(startTimer, 10);
  };

  stopBtn.onclick = () => {
    clearInterval(Interval);
  };

  resetBtn.onclick = () => {
    clearInterval(Interval);
    tens = '00';
    seconds = '00';
    minutes = '00';
    appendTens.innerHTML = tens;
    appendSeconds.innerHTML = seconds;
    appendMinutes.innerHTML = minutes;
    lapList.innerHTML = '';
  };

  lapBtn.onclick = () => {
    var li = document.createElement('li');
    li.innerHTML = appendMinutes.innerHTML + ':' + appendSeconds.innerHTML + '.' +
appendTens.innerHTML;
    lapList.appendChild(li);
  };
};

// Request Notification Permission when the app is loaded
```

```
if ('Notification' in window && 'serviceWorker' in navigator) {
  Notification.requestPermission().then(permission => {
    if (permission === 'granted') {
      console.log('Notification permission granted');
    }
  });
}
```

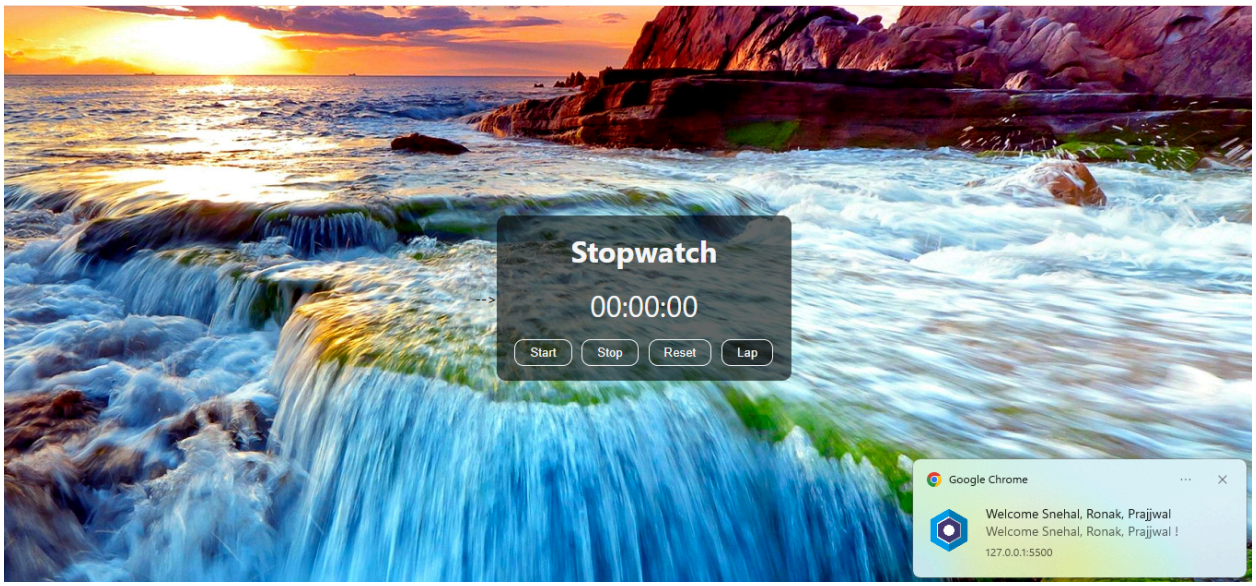
Screenshots:

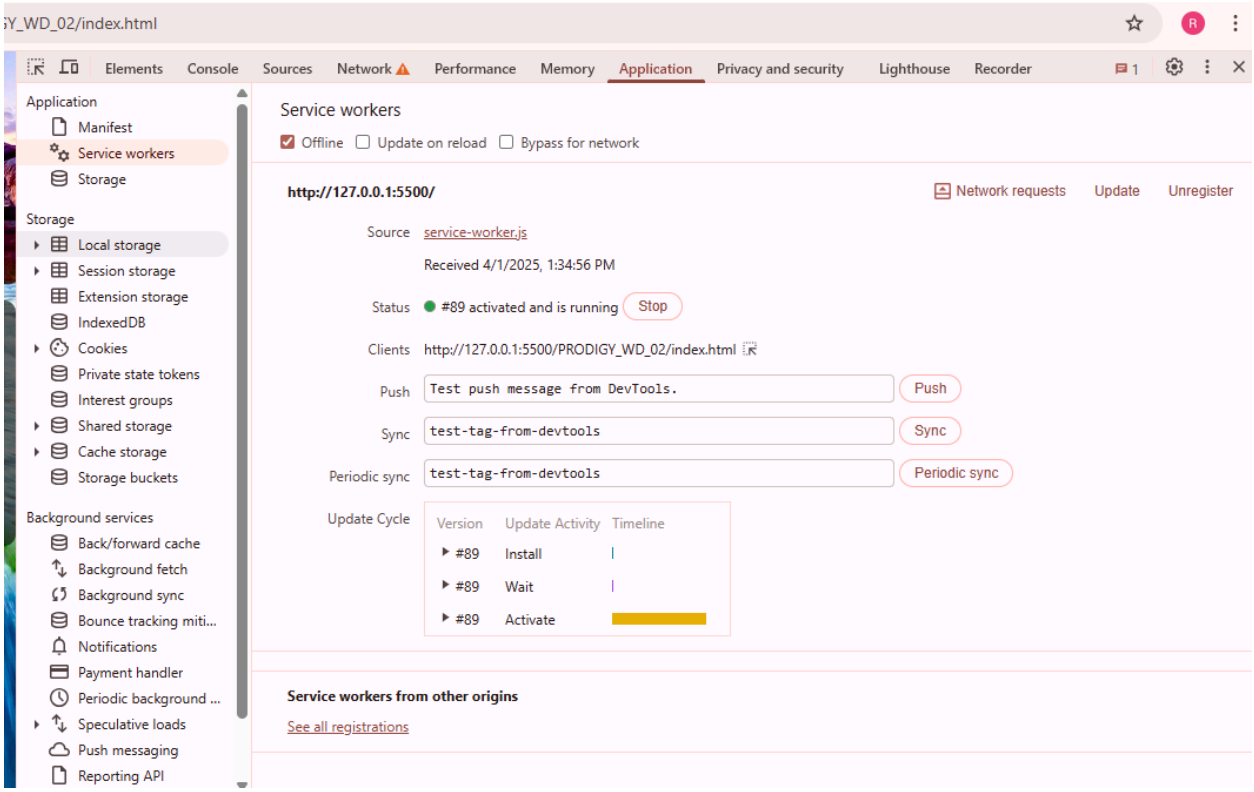


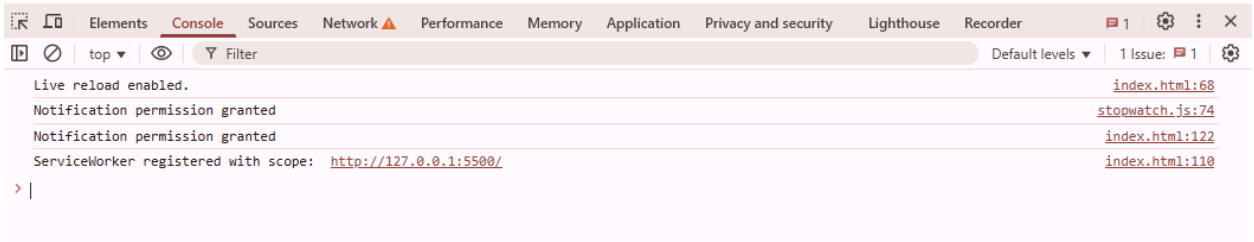





```
Notification permission granted                                  stopwatch.js:74
Notification permission granted                                  (index):122
ServiceWorker registered with scope:  http://127.0.0.1:5500/     (index):110
```




Y_WD_02/index.html
Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
Back/forward cache
Background fetch
Background sync
Bounce tracking miti...
Notifications
Payment handler
Periodic background ...
Speculative loads
Push messaging
Reporting API
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5500/
Network requests
Update
Unregister
Source
service-worker.js
Received 4/1/2025, 1:34:56 PM
Status
#89 activated and is running
Stop
Clients
http://127.0.0.1:5500/PRODIGY_WD_02/index.html
Push
Test push message from DevTools.
Push
Sync
test-tag-from-devtools
Sync
Periodic sync
test-tag-from-devtools
Periodic sync
Update Cycle
Version
Update Activity
Timeline
#89
Install
#89
Wait
#89
Activate
Service workers from other origins
See all registrations


Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
top
Filter
Default levels
1 Issue:
Live reload enabled.
index.html:68
Notification permission granted
stopwatch.js:74
Notification permission granted
index.html:122
ServiceWorker registered with scope:
http://127.0.0.1:5500/
index.html:110

Project Title: **Roll No.** 23

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	23
Name	Ronak Katariya
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment No.
10 MAD & PWA
Lab

Ronak Katariya 23
Prajjwal Pandey 32
Snehal Patil 38

Aim:

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

Theory:

GitHub Pages

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

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

Pros

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

Cons

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

Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

Pros

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

Cons

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

Link to our GitHub repository: https://github.com/ronak03rsk/PRODIGY_WD_02

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	23
Name	Ronak Katariya
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

Experiment: Using Google Lighthouse for PWA Analysis

Objective:
To use Google Lighthouse PWA Analysis Tool to test the Progressive Web App (PWA) functionality.

Theory:
Google Lighthouse is an open-source automated tool used to audit and analyze web applications for performance, accessibility, best practices, SEO, and Progressive Web App (PWA) capabilities. Lighthouse evaluates whether a web app meets PWA standards, such as:

- Service worker registration for offline support
- Web App Manifest inclusion
- HTTPS security compliance
- Fast and responsive performance
- Installability and app-like user experience

Lighthouse assigns a score based on these criteria, helping developers optimize their PWA.

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.

Requirements:

- Google Chrome browser
- A PWA-enabled web application
- Chrome DevTools or Lighthouse CLI

Procedure:

1. Open Lighthouse in Chrome DevTools

- Launch the PWA in Google Chrome.
- Open DevTools using `F12` or `Ctrl+Shift+I`.
- Navigate to the **Lighthouse** tab.

2. Configure Lighthouse Audit

- Select the **Progressive Web App** category.
- Ensure other relevant categories like Performance and Best Practices are checked.
- Choose the mode:

    - **Mobile** (default) for testing mobile performance.
    - **Desktop** for desktop evaluation.
- Click **Generate Report**.

3. Analyze the Results

- Lighthouse generates a PWA score based on:
    - **Fast and reliable:** Checks service worker caching.
    - **Installable:** Verifies manifest and service worker.
    - **PWA Optimizations:** Ensures proper web app experience.
- Click on individual sections to view recommendations for improvements.

4. Running Lighthouse via CLI (Optional)

Install Lighthouse CLI using Node.js:

```
npm install -g lighthouse
```

-

Run an audit on a PWA URL:

```
lighthouse https://your-pwa-url.com --view
```

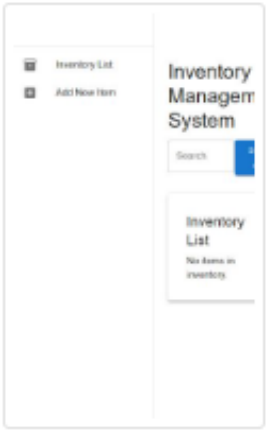
-

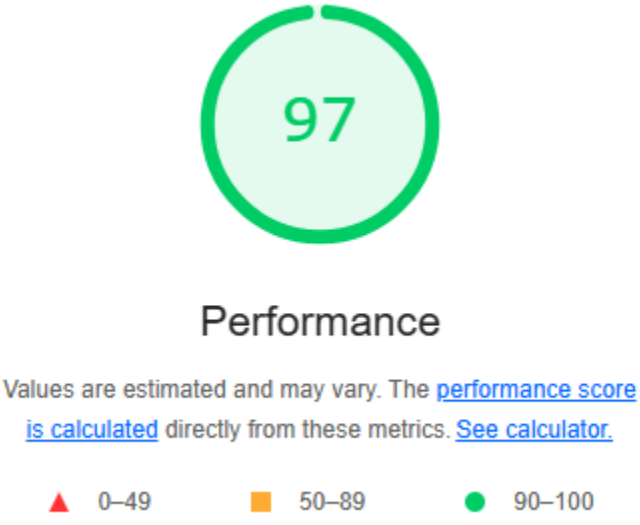
Observations:

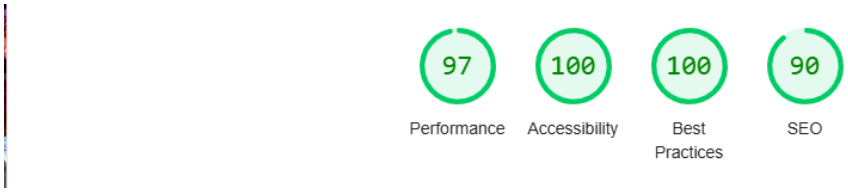
- The PWA is analyzed based on Lighthouse criteria.
- Identified areas needing improvement (e.g., caching strategies, manifest completeness, accessibility fixes).
- Scores indicate overall PWA compliance and optimization level.

Conclusion:
By using Google Lighthouse, we effectively evaluated the PWA's functionality, installability, and performance, enabling improvements for a better user experience.

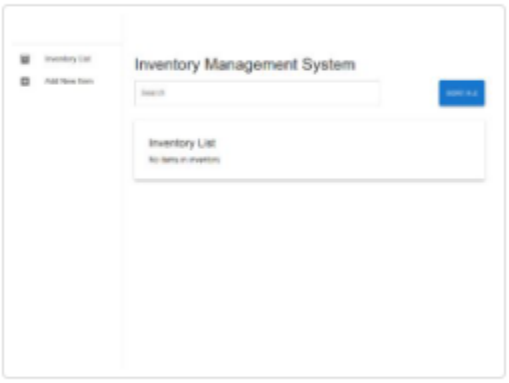
Mobile View:

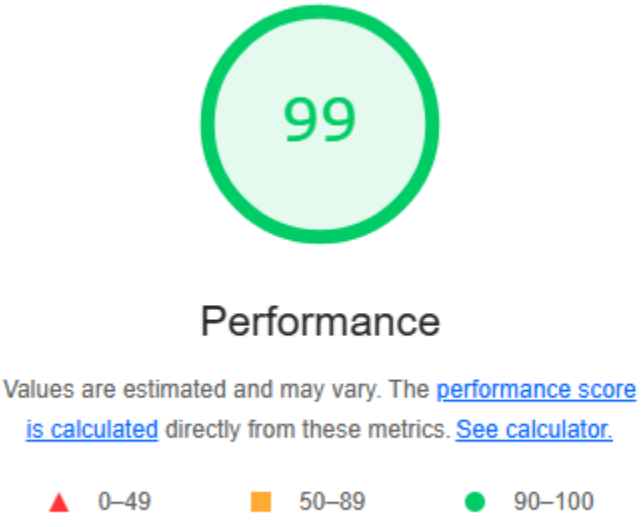


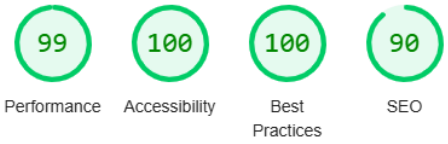




Desktop View:






99
Performance
100
Accessibility
100
Best Practices
90
SEO