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

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Project Title: **Roll No.**

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

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

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers : Mrs. Kajal Jewani.

Email : kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: **Roll No.**

Lab Objectives:

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

Lab Outcomes:

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

Project Title: **Roll No.**

Index

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

Project Title: **Roll No.**

MAD & PWA Lab
Journal

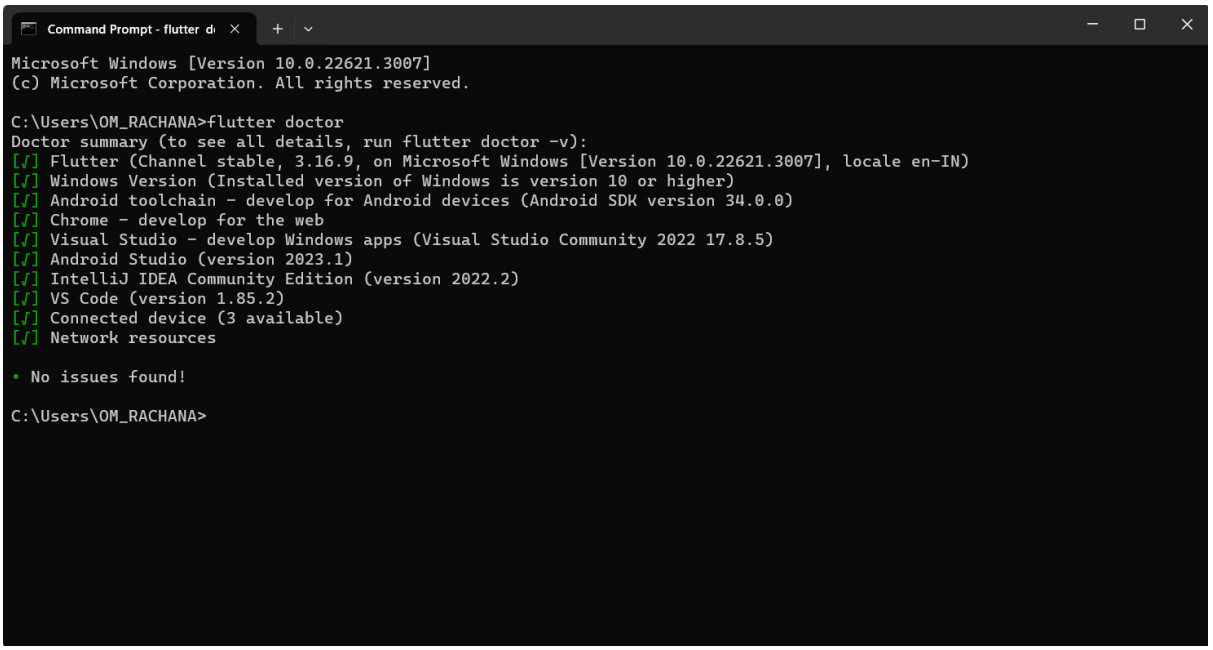
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

Exp1

Rachana Rane Roll No:48
Batch:C D15A

Aim:Installation and Configuration of Flutter Environment.

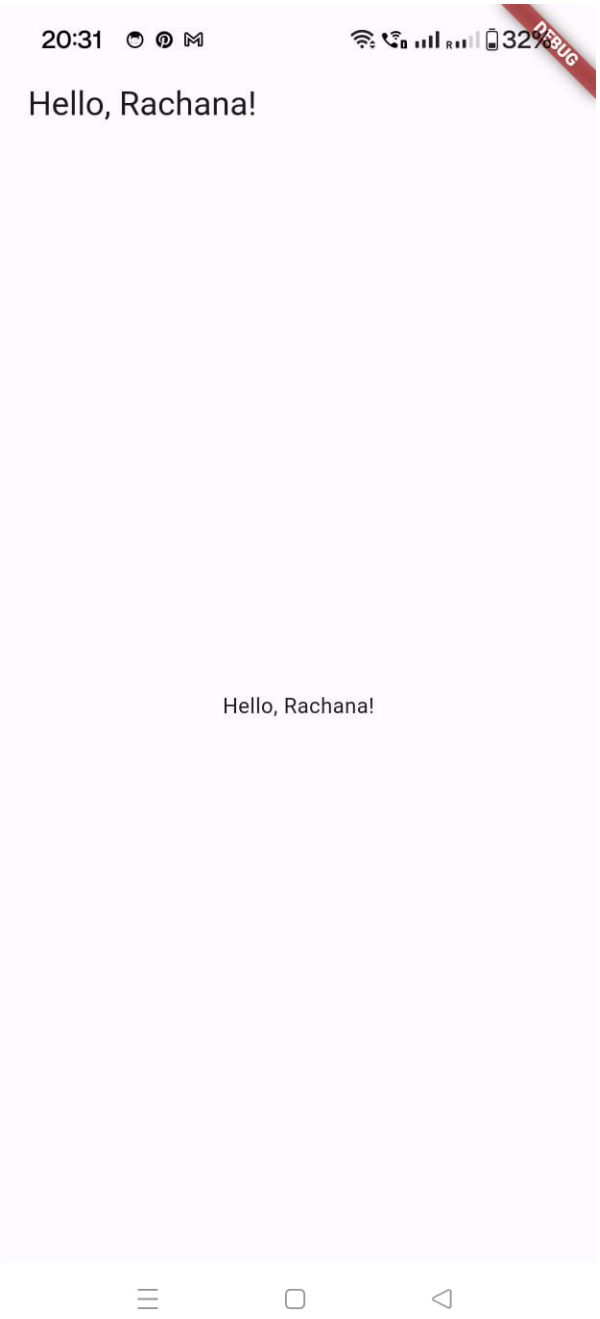
Install the Flutter SDK



Code:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Hello, Rachana!'),
        ),
        body: Center(
          child: Text('Hello, Rachana!'),
        ),
      ),
    );
  }
}
```

Output:



Conclusion:Thus,Flutter environment was created successfully.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Exp 2

Rachana Rane Roll No:48
Batch:C D15A

Aim: To design Flutter UI by including common widgets.

Theory: Flutter is a powerful open-source UI framework developed by Google for creating natively compiled applications for mobile, web, and desktop from a single codebase. It uses the Dart programming language.

Common Widgets in Flutter:

Container:
- A box model for styling and positioning other widgets.

Text:
- Displays text with customizable styles.

Image:
- Displays images from various sources.

Column and Row:
- Arranges children widgets in a vertical (Column) or horizontal (Row) sequence.

ListView:
- Displays a scrolling list of widgets.

AppBar:
- A material design app bar with a title and optional actions.

Scaffold:
- Implements the basic material design visual layout structure.

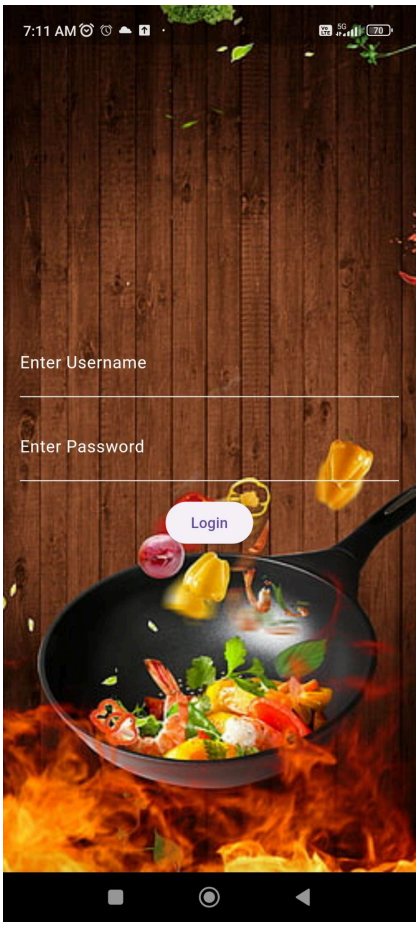
Code:

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

void main() {
  runApp(login());
}

class login extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Login Page',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: LoginPage(),
    );
```

```
  }
}

class LoginPage extends StatefulWidget {
  @override
  _LoginPageState createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  TextEditingController _usernameController = TextEditingController();
  TextEditingController _passwordController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        decoration: BoxDecoration(
          image: DecorationImage(
            image: NetworkImage(

'https://e0.pxfuel.com/wallpapers/281/550/desktop-wallpaper-advertising-background-stirfried-synthetic-creative-catering-stirfrystir-fried-creative-synthe-food-poster-food-poster-design-food-background-thumbnail.jpg',
            ),
            fit: BoxFit.cover,
          ),
        ),
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              TextField(
                controller: _usernameController,
                style: TextStyle(
                    color: Colors.white), // Change the color of entered text
                decoration: InputDecoration(
                  labelText: 'Enter Username',
                  labelStyle: TextStyle(color: Colors.white),
                  enabledBorder: UnderlineInputBorder(
                    borderSide: BorderSide(color: Colors.white),
                  ),
                ),
              ),
              SizedBox(height: 16.0),
              TextField(
                controller: _passwordController,
                obscureText: true,
```

```
            style: TextStyle(
                color: Colors.white), // Change the color of entered text
            decoration: InputDecoration(
              labelText: 'Enter Password',
              labelStyle: TextStyle(color: Colors.white),
              enabledBorder: UnderlineInputBorder(
                borderSide: BorderSide(color: Colors.white),
              ),
            ),
          ),
          SizedBox(height: 16.0),
          ElevatedButton(
            onPressed: () {
              // Implement your login logic here
              String username = _usernameController.text;
              String password = _passwordController.text;

              // Dummy validation for illustration purposes
              if (username == 'user' && password == 'password') {
                // Navigate to the home page on successful login
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => HomePage()),
                );
              } else {
                // Display an error message or handle unsuccessful login
                print('Login Failed');
              }
            },
            child: Text('Login'),
          ),
        ],
      ),
    ),
  );
 }
}
```

Output:



Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Exp 3

Rachana Rane Roll No:48
Batch:C D15A

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

Theory:

Icons:

- Use the built-in `Icon` class to include icons from Material Design or Cupertino Icons libraries.
- For custom icons, consider using packages like `flutter_svg` for handling SVG icons.

Images:

- Display images using the `Image` widget.
- For local images, declare them in the 'assets' section of the `pubspec.yaml` file and use `Image.asset`.
- For network images, use `Image.network`.

Custom Fonts:

- Include custom fonts by specifying font files in the 'fonts' section of the `pubspec.yaml` file.
- Apply custom fonts to text using the `TextStyle` class with the specified font family.

Code:

```
import 'package:my_first_app/home_page.dart';
import 'package:flutter/material.dart';
import 'profile_page.dart';

class Recipe {
  final String name;
  final String imageUrl;
  final String blogContent;

  Recipe({
    required this.name,
    required this.imageUrl,
    required this.blogContent,
  });
}

class MyApp extends StatelessWidget {
  final List<Recipe> recipes = [
    Recipe(
      name: 'Chocolate Cake',
```

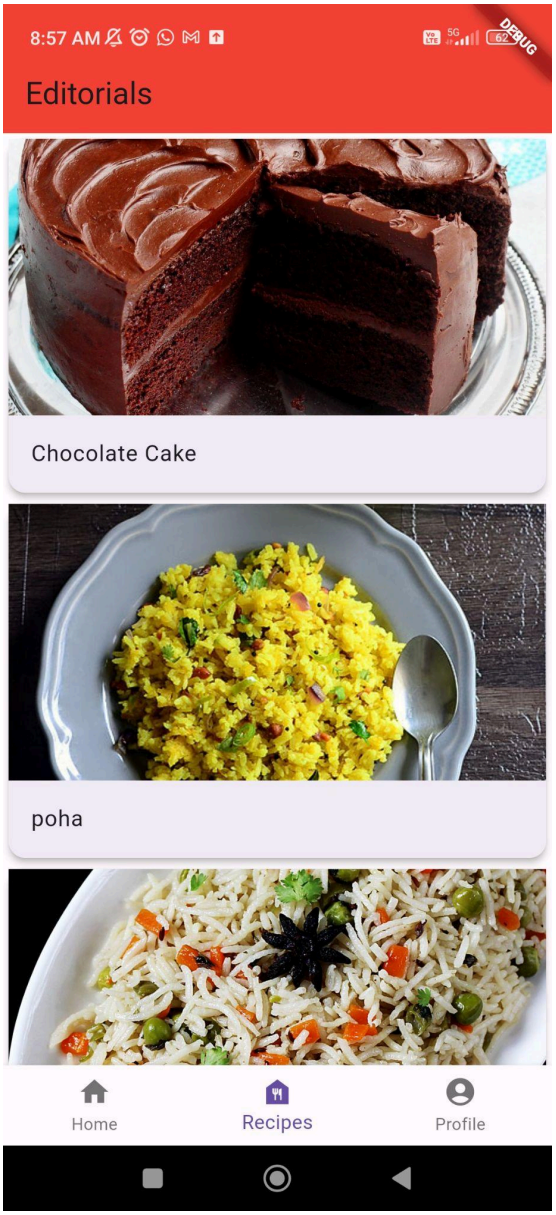
```
      imageUrl:
          'https://scientificallysweet.com/wp-content/uploads/2020/09/IMG_4087-feature-2.jpg',
      blogContent: 'This is a delicious chocolate cake recipe...',
    ),
    Recipe(
      name: 'poha',
      imageUrl:
          'https://c2.staticflickr.com/2/1703/25173602171_2a9465cd12_z.jpg',
      blogContent: 'Enjoy this flavorful poha recipe...',
    ),
    Recipe(
      name: 'pulaw',
      imageUrl:

'https://www.indianhealthyrecipes.com/wp-content/uploads/2022/04/veg-pulao-recipe.jpg',
      blogContent: 'Enjoy this flavorful pulaw recipe...',
    ),
    // Add more recipes as needed
  ];

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Recipe App',
      home: RecipeListPage(recipes: recipes),
    );
  }
}

class RecipeListPage extends StatelessWidget {
  final List<Recipe> recipes;

  RecipeListPage({required this.recipes});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Editorials'),
        backgroundColor: Colors.red,
      ),
      body: ListView.builder(
        itemCount: recipes.length,
        itemBuilder: (context, index) {
          return Card(
            elevation: 4.0,
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.stretch,
```

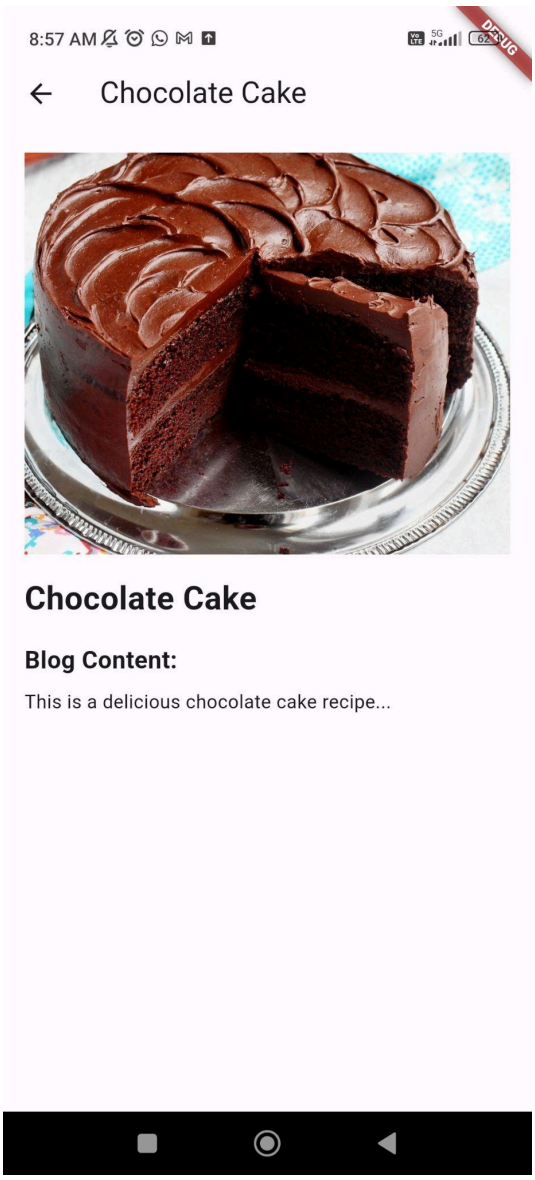
```
        children: [
          Image.network(
            recipes[index].imageUrl,
            fit: BoxFit.cover,
            height: 200.0, // Adjust the height as needed
          ),
          ListTile(
            title: Text(recipes[index].name),
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) =>
                      RecipeDetailPage(recipe: recipes[index]),
                ),
              );
            },
          ),
        ],
      ),
    );
  },
),
bottomNavigationBar: BottomNavigationBar(
  items: [
    BottomNavigationBarItem(
      icon: Icon(Icons.home),
      label: 'Home',
    ),
    BottomNavigationBarItem(
      icon: Icon(Icons.food_bank),
      label: 'Recipes',
    ),
    BottomNavigationBarItem(
      icon: Icon(Icons.account_circle),
      label: 'Profile',
    ),
  ],
  currentIndex: 1,
  onTap: (index) {
    if (index == 0) {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => HomePage(),
        ),
      );
    }
```

```
          if (index == 2) {
           Navigator.push(
             context,
             MaterialPageRoute(
               builder: (context) => ProfilePage(),
             ),
           );
          }
        },
      ),
    );
  }
}

class RecipeDetailPage extends StatelessWidget {
  final Recipe recipe;

  RecipeDetailPage({required this.recipe});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(recipe.name),
      ),
      body: Padding(
        padding: EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            Image.network(
              recipe.imageUrl,
              fit: BoxFit.cover,
              height: 300.0, // Adjust the height as needed
            ),
            SizedBox(height: 16.0),
            Text(
              recipe.name,
              style: TextStyle(
                fontSize: 24.0,
                fontWeight: FontWeight.bold,
              ),
            ),
            SizedBox(height: 16.0),
            Text(
              'Blog Content:',
              style: TextStyle(
                fontWeight: FontWeight.bold,
```

```
                fontSize: 18.0,
              ),
            ),
            SizedBox(height: 8.0),
            Text(recipe.blogContent),
          ],
        ),
      ),
    );
  }
}
```

Output:





Conclusion: Thus , included icons, images, and fonts in Flutter app.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Exp 4 To create an interactive Form using form widget

Steps

1. Create a Form with a GlobalKey
2. Add a TextFormField with validation logic
3. Create a button to validate and submit the form

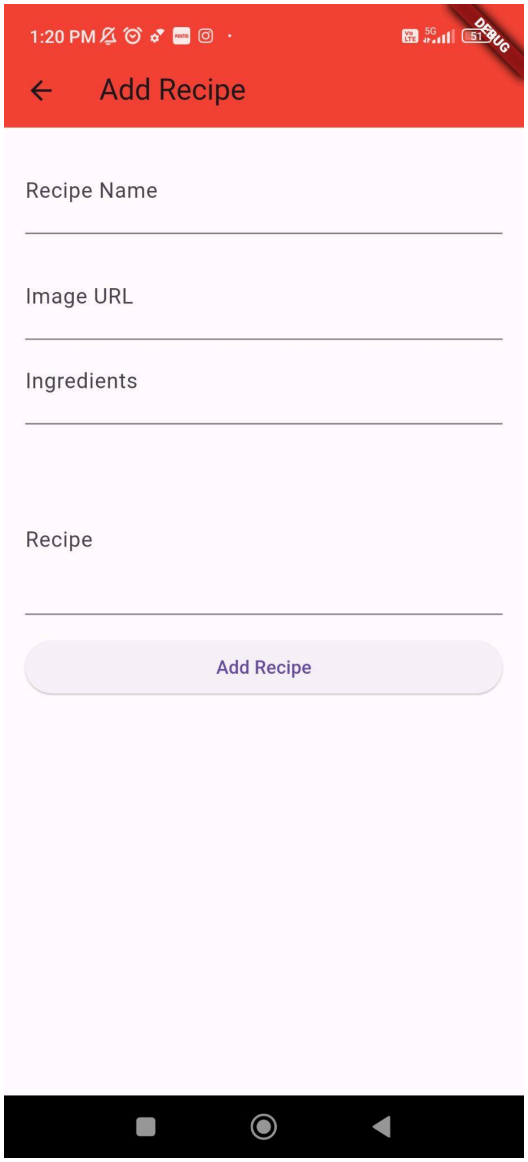
Sample Code

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

class AddRecipeForm extends StatefulWidget {
  @override
  _AddRecipeFormState createState() => _AddRecipeFormState();
}

class _AddRecipeFormState extends State<AddRecipeForm> {
  final _formKey = GlobalKey<FormState>();
  late TextEditingController _nameController;
  late TextEditingController _imageUrlController;
  late TextEditingController _blogContentController;
  late TextEditingController _ingredientsController;

  @override
  void initState() {
    super.initState();
    _nameController = TextEditingController();
    _imageUrlController = TextEditingController();
    _blogContentController = TextEditingController();
    _ingredientsController=TextEditingController();
  }

  @override
  void dispose() {
    _nameController.dispose();
```

```
  _imageUrlController.dispose();
  _blogContentController.dispose();
  _ingredientsController.dispose();
  super.dispose();
}

Future<void> _submitForm() async {
  if (_formKey.currentState!.validate()) {
    try {
      await FirebaseFirestore.instance.collection('recipes').add({
        'ingredients':_ingredientsController.text,
        'title': _nameController.text,
        'url': _imageUrlController.text,
        'recipe': _blogContentController.text,
      });
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Recipe added successfully')),
      );
      _formKey.currentState!.reset();
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Failed to add recipe: $e')),
      );
    }
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Add Recipe'),
      backgroundColor: Colors.red,
    ),
    body: Padding(
```

```
padding: EdgeInsets.all(16.0),
child: Form(
  key: _formKey,
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: [
      TextFormField(
        controller: _nameController,
        decoration: InputDecoration(labelText: 'Recipe Name'),
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'Please enter the recipe name';
          }
          return null;
        },
      ),
      SizedBox(height: 16.0),
      TextFormField(
        controller: _imageUrlController,
        decoration: InputDecoration(labelText: 'Image URL'),
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'Please enter the image URL';
          }
          return null;
        },
      ),
      TextFormField(
        controller: _ingredientsController,
        decoration: InputDecoration(labelText: 'Ingredients'),
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'Please enter the Ingredients';
          }
          return null;
```

```
            },
          ),
          SizedBox(height: 16.0),
          SizedBox(height: 16.0),
          TextFormField(
            controller: _blogContentController,
            decoration: InputDecoration(labelText: 'Recipe'),
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Please enter the blog content';
              }
              return null;
            },
            maxLines: 3,
          ),
          SizedBox(height: 16.0),
          ElevatedButton(
            onPressed: _submitForm,
            child: Text('Add Recipe'),
          ),
        ],
      ),
    ),
  ),
 );
}
```


1:20 PM
DEBUG
Add Recipe
Recipe Name
Image URL
Ingredients
Recipe
Add Recipe

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Exp 5

Rachana Rane Roll No:48
Batch:C D15A

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

//Navigation

Steps:

1. Create two routes. (In Flutter, *screens* and *pages* are called *routes*.)
2. Navigate to the second route using Navigator.push().
3. Return to the first route using Navigator.pop().

Code:

```
import 'package:my_first_app/home_page.dart';
import 'package:flutter/material.dart';
import 'package:my_first_app/recripe_library.dart';
import 'profile_page.dart';

class Recipe {
  final String name;
  final String imageUrl;
  final String blogContent;

  Recipe({
    required this.name,
    required this.imageUrl,
    required this.blogContent,
  });
}

class MyApp extends StatelessWidget {
  final List<Recipe> recipes = [
    Recipe(
      name: 'Chocolate Cake',
      imageUrl:
          'https://scientificallysweet.com/wp-content/uploads/2020/09/IMG_4087-feature-2.jpg',
      blogContent: 'This is a delicious chocolate cake recipe...',
    ),
    Recipe(
      name: 'poha',
      imageUrl:
          'https://c2.staticflickr.com/2/1703/25173602171_2a9465cd12_z.jpg',
      blogContent: 'Enjoy this flavorful poha recipe...',
    ),
```

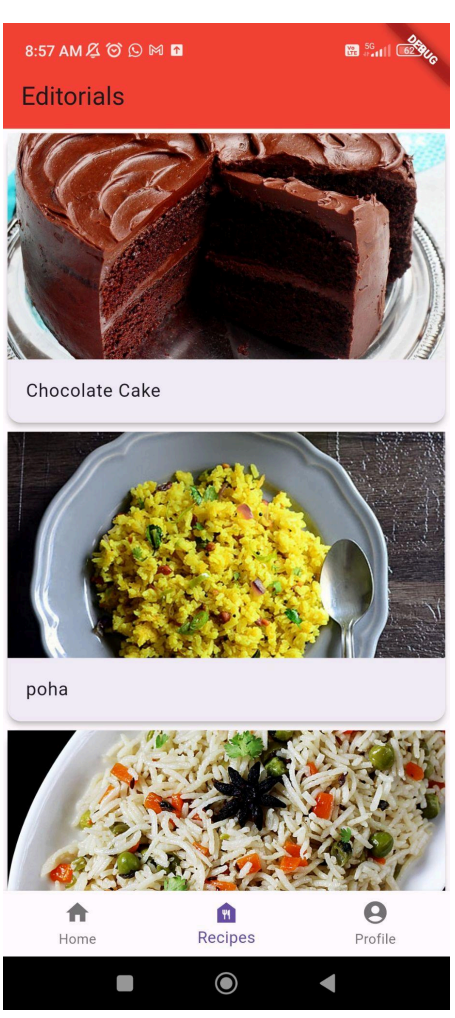
```
    Recipe(
      name: 'pulaw',
      imageUrl:

'https://www.indianhealthyrecipes.com/wp-content/uploads/2022/04/veg-pulao-recipe.jpg',
      blogContent: 'Enjoy this flavorful pulaw recipe...',
    ),
    // Add more recipes as needed
  ];

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Recipe App',
      home: RecipeListPage(recipes: recipes),
    );
  }
}

class RecipeListPage extends StatelessWidget {
  final List<Recipe> recipes;

  RecipeListPage({required this.recipes});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Editorials'),
        backgroundColor: Colors.red,
      ),
      body: ListView.builder(
        itemCount: recipes.length,
        itemBuilder: (context, index) {
          return Card(
            elevation: 4.0,
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.stretch,
              children: [
                Image.network(
                  recipes[index].imageUrl,
                  fit: BoxFit.cover,
                  height: 200.0, // Adjust the height as needed
                ),
                ListTile(
                  title: Text(recipes[index].name),
                  onTap: () {
                    Navigator.push(
```

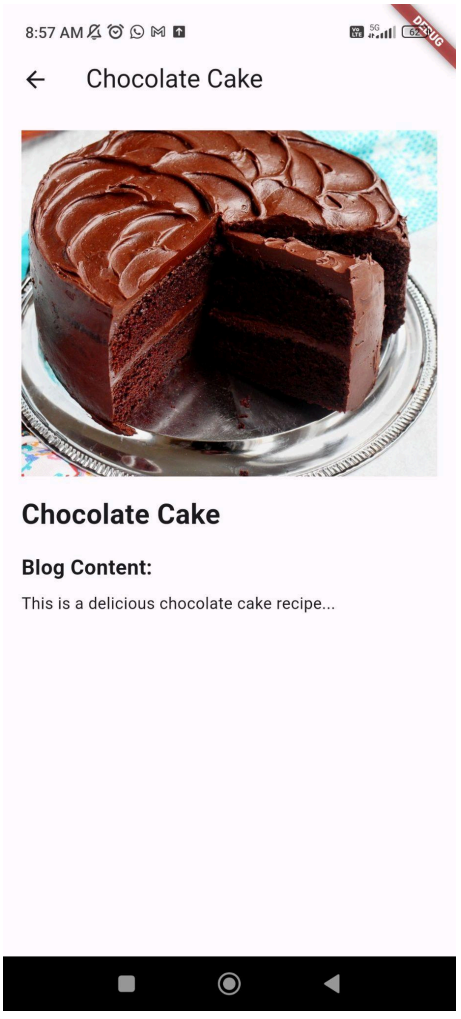
```
                context,
                MaterialPageRoute(
                  builder: (context) =>
                      RecipeDetailPage(recipe: recipes[index]),
                ),
              );
            },
          ),
        ],
      ),
    );
  },
),
bottomNavigationBar: BottomNavigationBar(
  items: [
    BottomNavigationBarItem(
      icon: Icon(Icons.home),
      label: 'Home',
    ),
    BottomNavigationBarItem(
      icon: Icon(Icons.food_bank),
      label: 'Recipes',
    ),
    BottomNavigationBarItem(
      icon: Icon(Icons.account_circle),
      label: 'Profile',
    ),
     BottomNavigationBarItem(
      icon: Icon(Icons.add),
      label: 'My Recipes',
    ),
  ],
  currentIndex: 1,
  onTap: (index) {
    if (index == 0) {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => HomePage(),
        ),
      );
    }
     if (index == 2) {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => ProfilePage(),
        ),
```

```
            );
          }
          if (index == 3) {
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => Recipelib(),
              ),
            );
          }
        },
      ),
    );
  }
}

class RecipeDetailPage extends StatelessWidget {
  final Recipe recipe;

  RecipeDetailPage({required this.recipe});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(recipe.name),
      ),
      body: Padding(
        padding: EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            Image.network(
              recipe.imageUrl,
              fit: BoxFit.cover,
              height: 300.0, // Adjust the height as needed
            ),
            SizedBox(height: 16.0),
            Text(
              recipe.name,
              style: TextStyle(
                fontSize: 24.0,
                fontWeight: FontWeight.bold,
              ),
            ),
            SizedBox(height: 16.0),
            Text(
              'Blog Content:',
```

```
            style: TextStyle(
              fontWeight: FontWeight.bold,
              fontSize: 18.0,
            ),
          ),
          SizedBox(height: 8.0),
          Text(recipe.blogContent),
        ],
      ),
    ),
  );
}
```

Output:





Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	
Name	
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:	

Name:Rachana Rane
Roll No:48
Class:D15A

Firebase is a great backend solution for anyone that wants to use authentication, databases, cloud functions, ads, and countless other features within an app.

In this article, you will create a Firebase project for iOS and Android platforms using Flutter.

Prerequisites

To complete this tutorial, you will need:

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

This tutorial was verified with Flutter v2.0.6, Android SDK v31.0.2, and Android Studio v4.1.

Creating a New Flutter Project

This tutorial will require the creation of an example Flutter app.

Once you have your environment set up for Flutter, you can run the following to create a new application:

```
flutter create flutterfirebaseexample
```

1.

Copy

Navigate to the new project directory:

```
cd flutterfirebaseexample
```

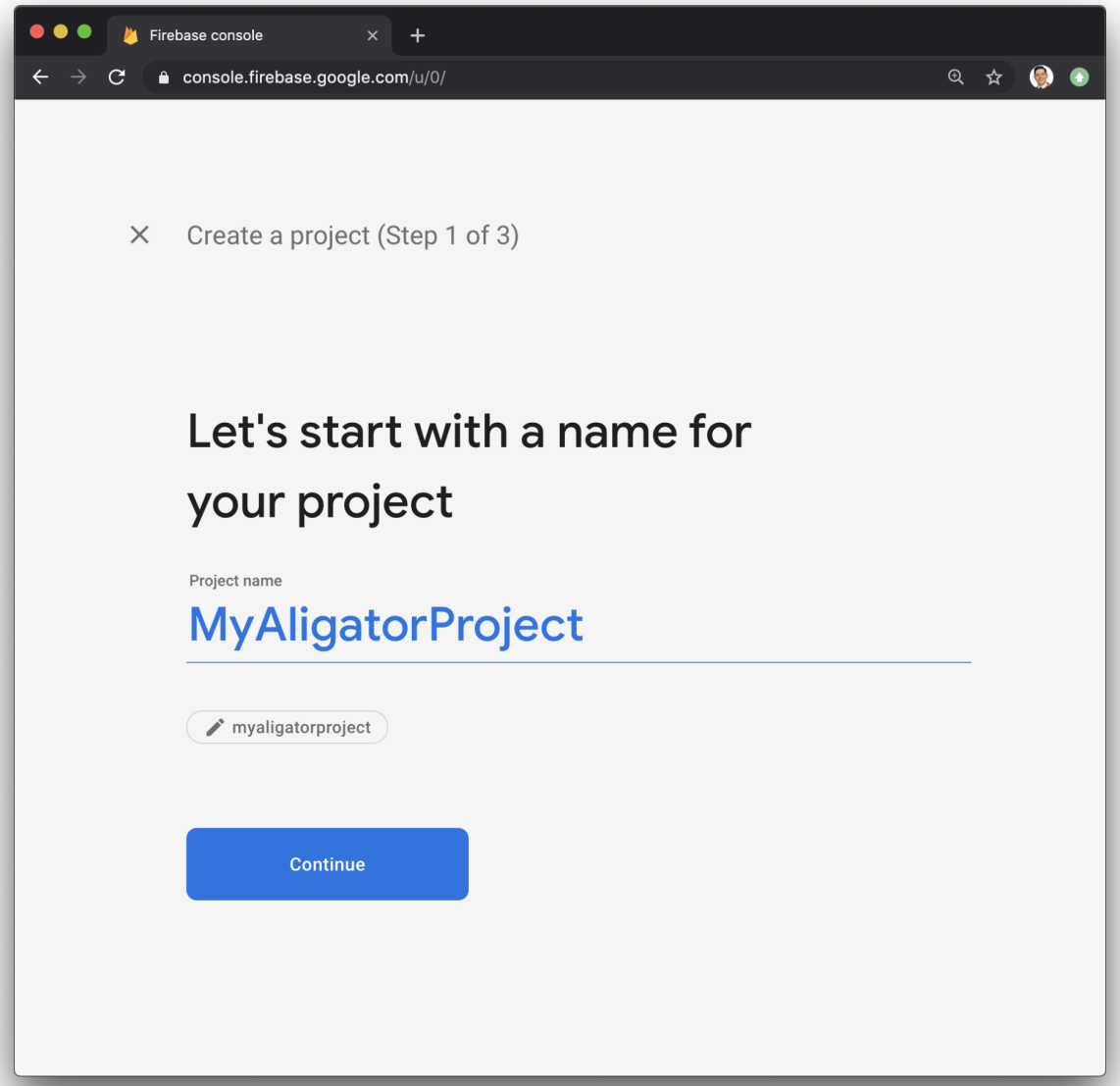
Copy

Using `flutter create` will produce a demo application that will display the number of times a button is clicked.

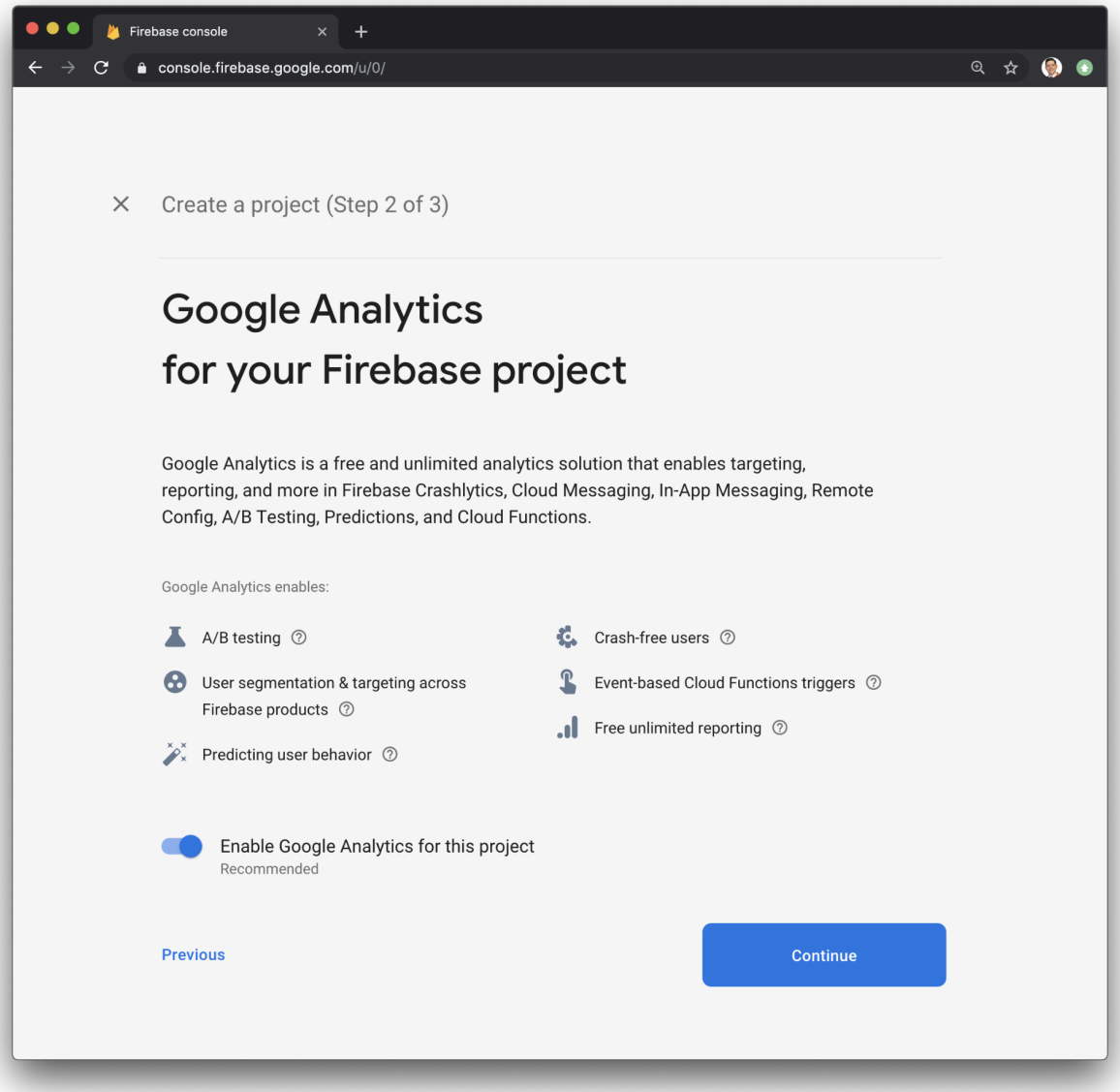
Now that we've got a Flutter project up and running, we can add Firebase.

Creating a New Firebase Project

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



Next, we're given the option to enable Google Analytics. This tutorial will not require Google Analytics, but you can also choose to add it to your project.



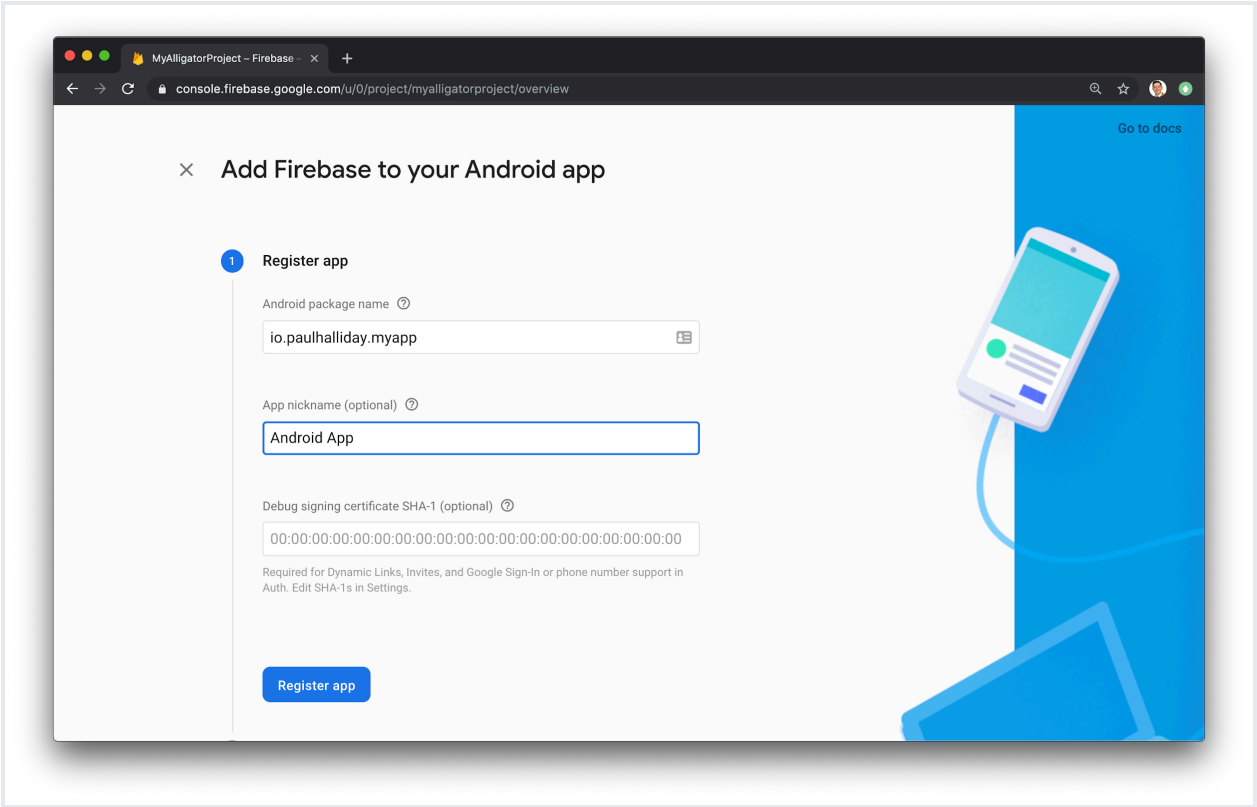
f you choose to use Google Analytics, you will need to review and accept the terms and conditions prior to project creation.

After pressing Continue, your project will be created and resources will be provisioned. You will then be directed to the dashboard for the new project.

Adding Android support

Registering the App

In order to add Android support to our Flutter application, select the Android logo from the dashboard. This brings us to the following screen:



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

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

`com.example.flutterfirebaseexample`

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

android/app/build.gradle

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

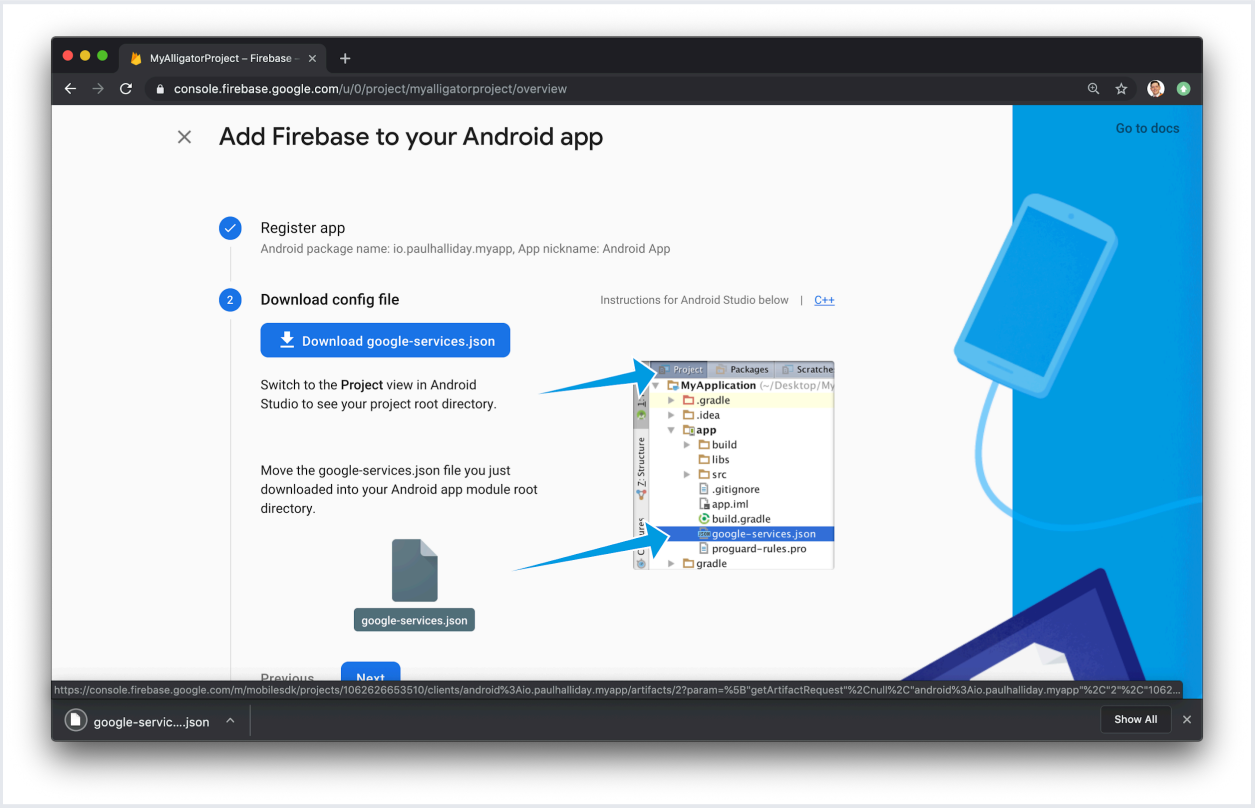
```
}
...
```

You can skip the app nickname and debug signing keys at this stage. Select Register app to continue.

Downloading the Config File

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

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



Next, move the `google-services.json` file to the `android/app` directory within the Flutter project.

Adding the Firebase SDK

We'll now need to update our Gradle configuration to include the Google Services plugin.

Open `android/build.gradle` in your code editor and modify it to include the following:

android/buiild.gradle

```
buildscript {
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
  }
  dependencies {
    ...
    // Add this line
    classpath 'com.google.gms:google-services:4.3.6'
  }
}

allprojects {
  ...
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
    ...
  }
}
```

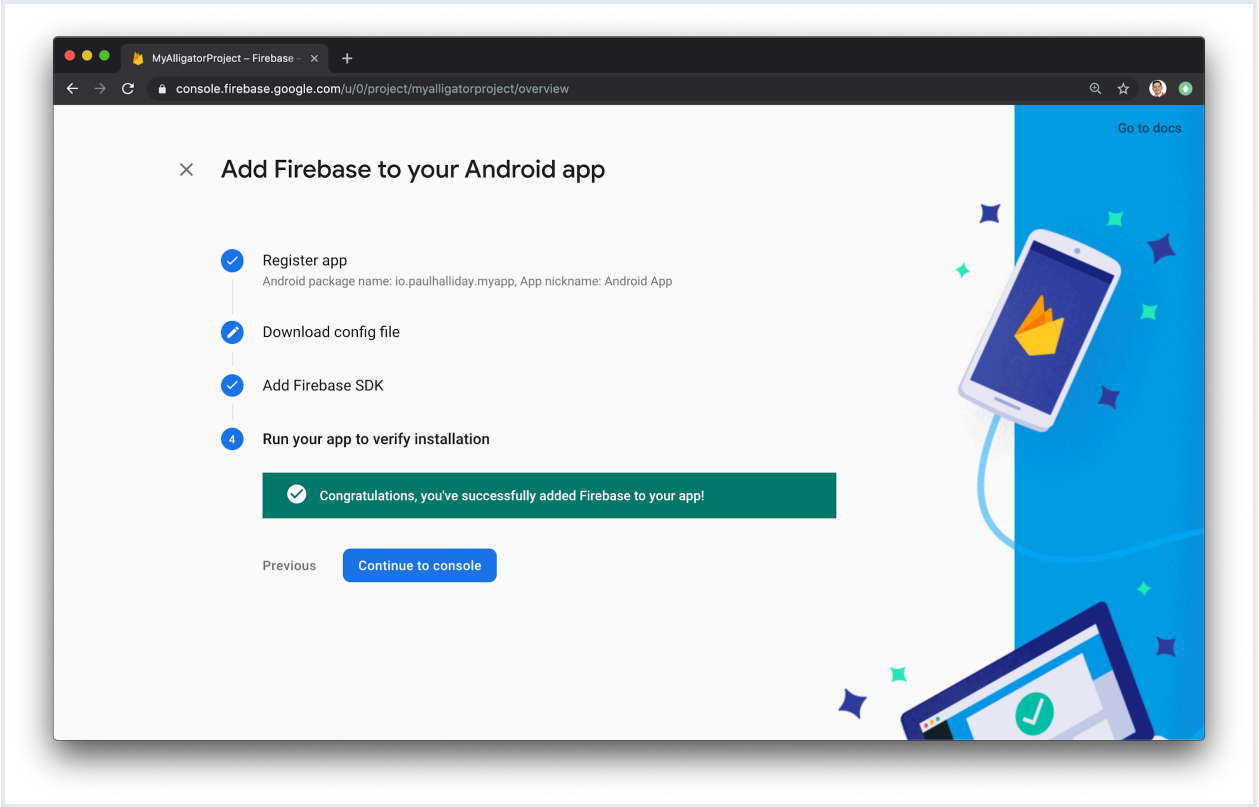
Finally, update the app level file at `android/app/build.gradle` to include the following:

android/app/build.gradle

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
  // Import the Firebase BoM
  implementation platform('com.google.firebase:firebase-bom:28.0.0')

  // Add the dependencies for any other desired Firebase products
  // https://firebase.google.com/docs/android/setup#available-libraries
}
```

With this update, we're essentially applying the Google Services plugin as well as looking at how other Flutter Firebase plugins can be activated such as Analytics.

From here, run your application on an Android device or simulator. If everything has worked correctly, you should get the following message in the dashboard:



Next up, let's add iOS support!

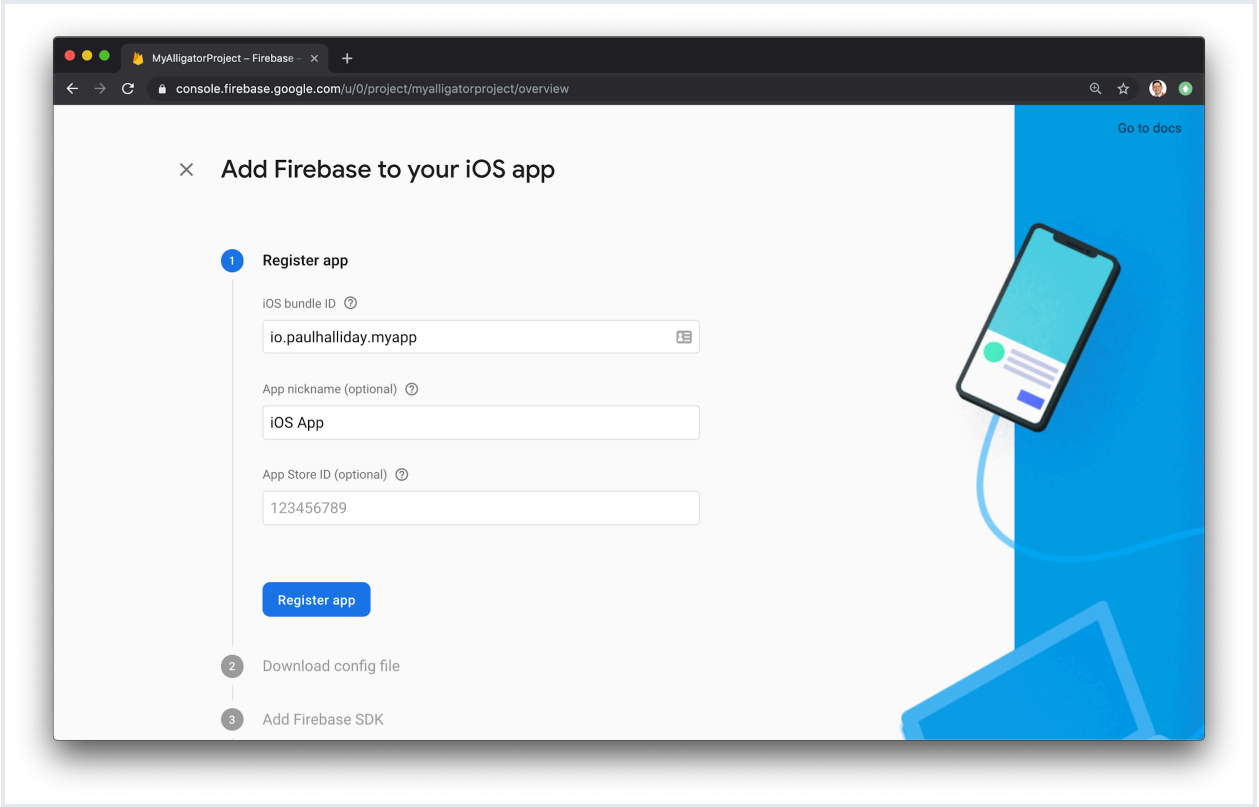
Adding iOS Support

In order to add Firebase support for iOS, we have to follow a similar set of instructions.

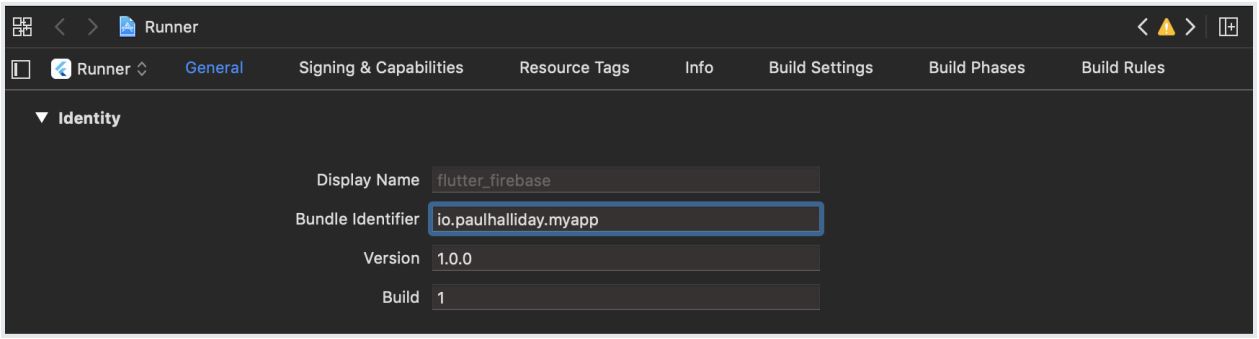
Head back over to the dashboard and select Add app and then iOS icon to be navigated to the setup process.

Registering an App

Once again, we'll need to add an "iOS Bundle ID". It is possible to use the "Android package name" for consistency:



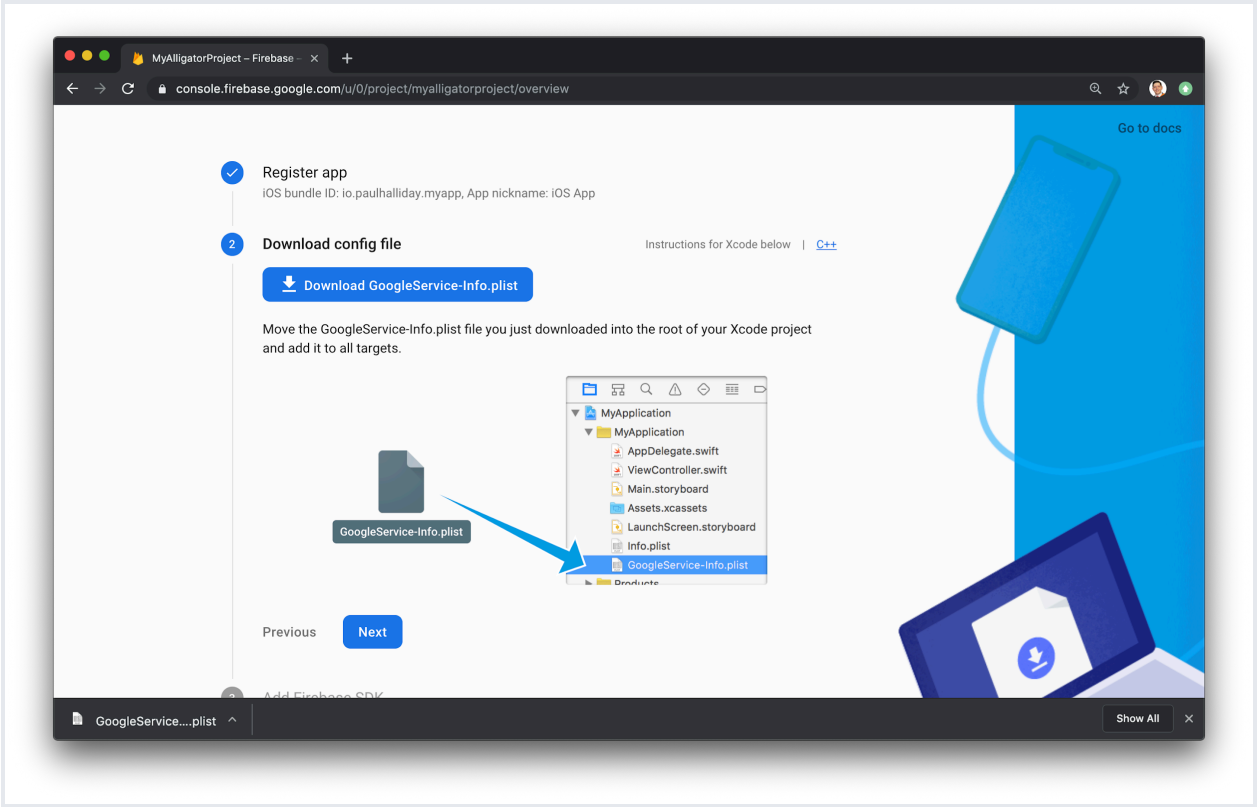
You'll then need to make sure this matches up by opening the iOS project up in Xcode at `ios/Runner/Runner.xcodeproj` and changing the Bundle identifier under General:



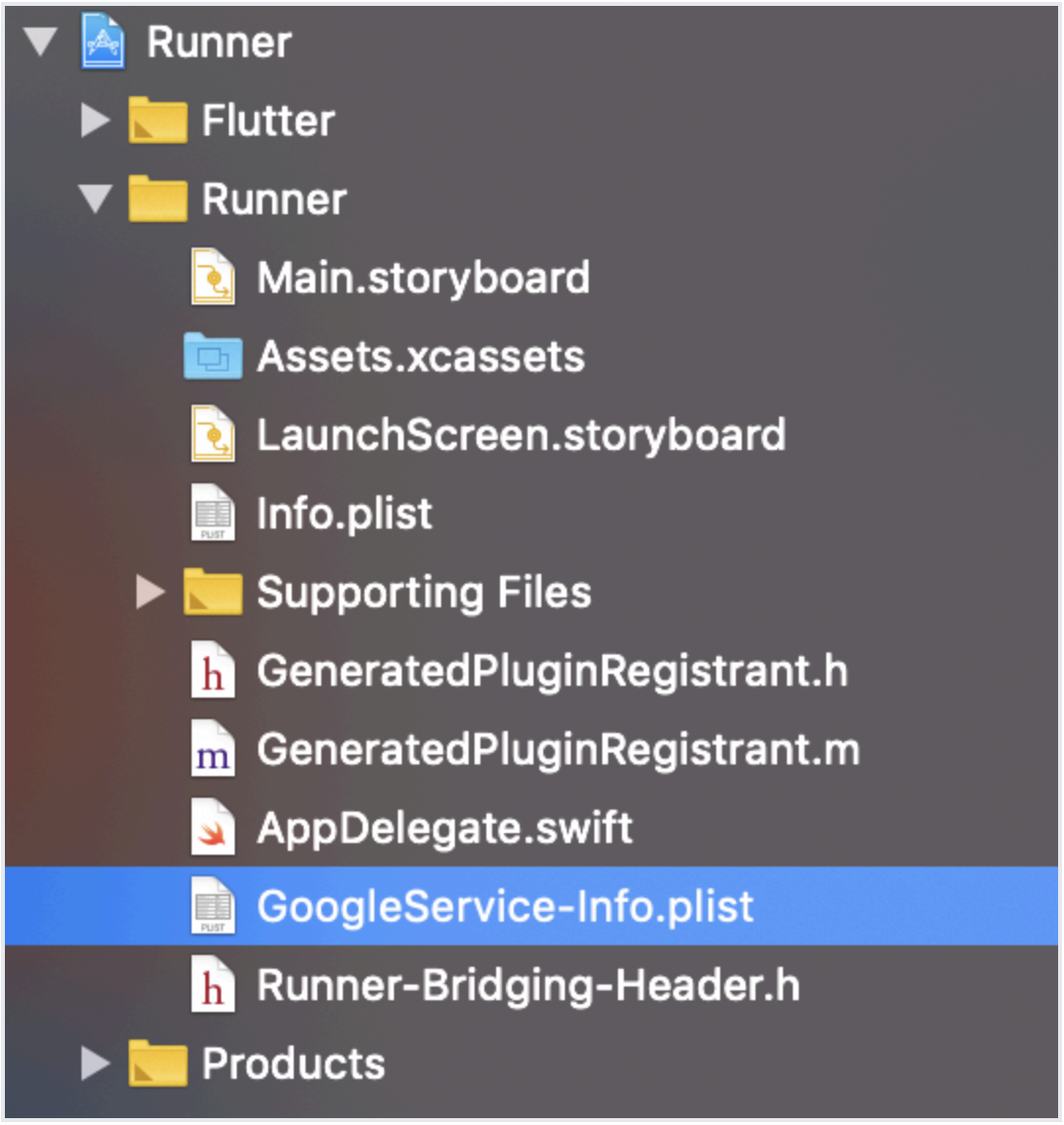
Click Register app to move to the next screen.

Downloading the Config File

In this step, we'll need to download the configuration file and add this to our Xcode project.



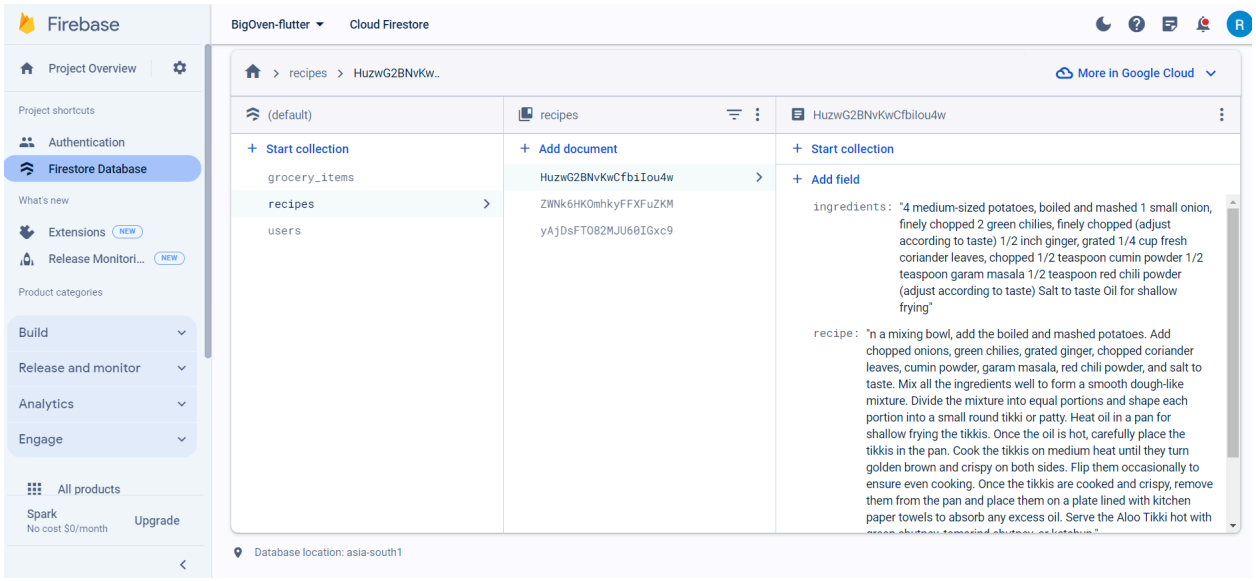
Download `GoogleService-Info.plist` and move this into the root of your Xcode project within `Runner`:



Be sure to move this file within Xcode to create the proper file references.

There are additional steps for installing the Firebase SDK and adding initialization code, but they are not necessary for this tutorial.

That's it!



Conclusion

In this article, you learned how to set up and ready our Flutter applications to be used with Firebase.

Flutter has official support for Firebase with the FlutterFire set of libraries.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Experiment 7

Name: Rachana Rane
Div: D15A
Roll no: 48

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

Theory:-

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

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

- Difference between PWAs vs. Regular Web Apps:

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

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

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

4. Engaging Approach

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

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

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

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

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

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

Weaknesses refer to:

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

Code:
index.html of the react app

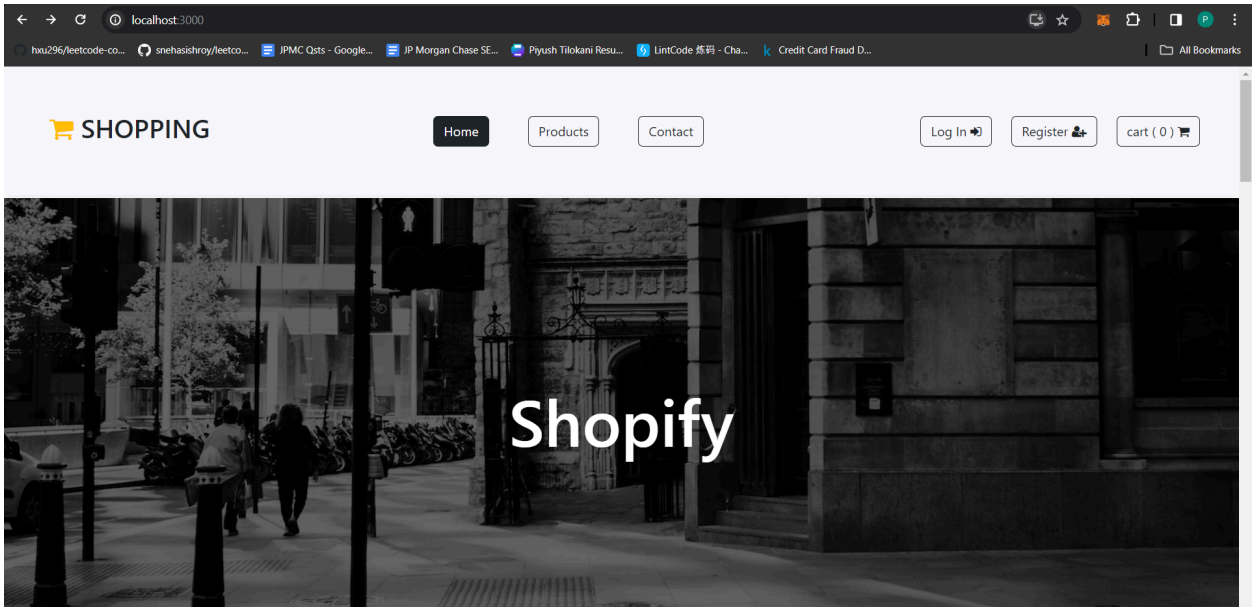
```
<!DOCTYPE html>
<html lang="en">

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

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

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

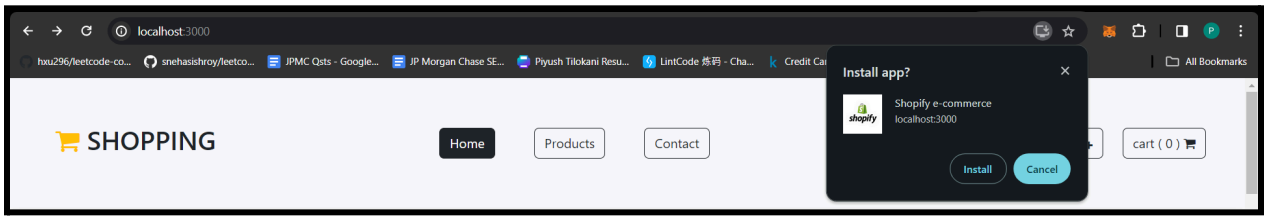
```
  <!-- Meta Tags required for
       Progressive Web App -->
  <meta name=
  "apple-mobile-web-app-status-bar"
        content="#aa7700">
  <meta name="theme-color"
        content="black">

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

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

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

</html>
```

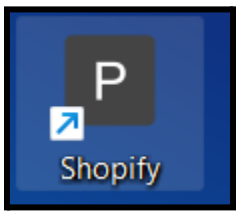
manifest.json

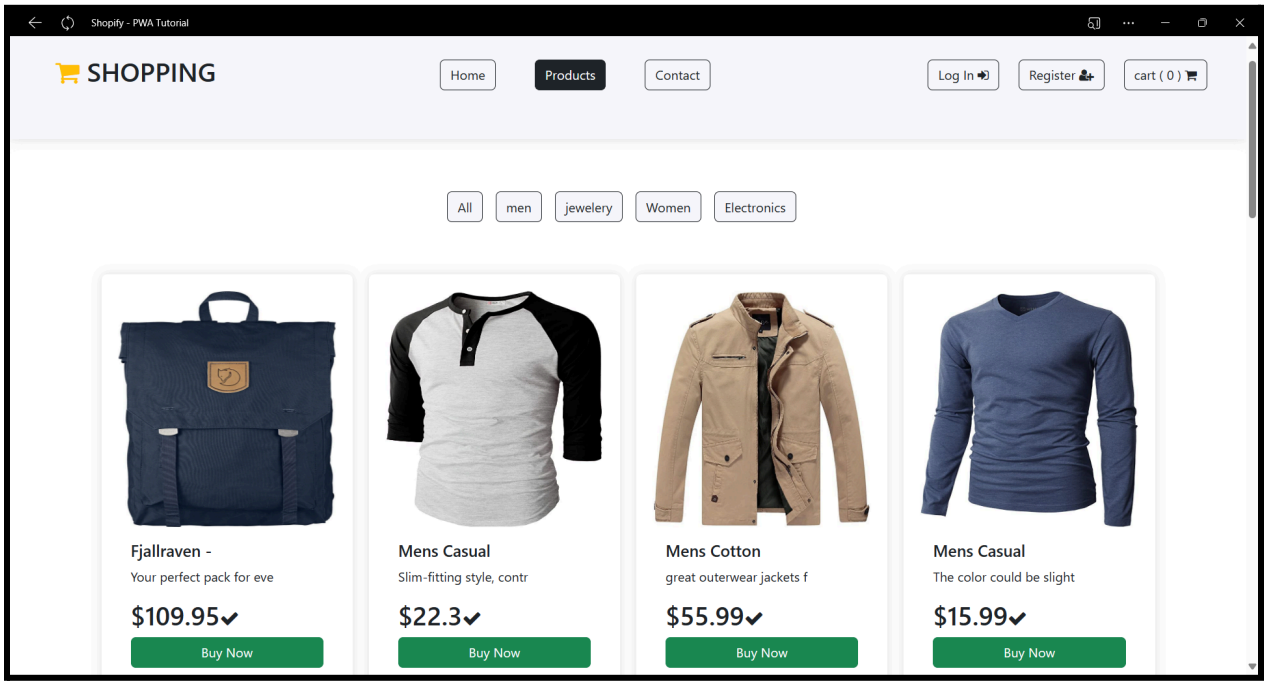
```
{
          "name":"Shopify e-commerce",
          "short_name":"Shopify e-commerce",
          "start_url":"index.html",
          "display":"standalone",
          "background_color":"#5900b3",
          "theme_color":"black",
          "scope": ".",
          "description":"Shopify e-commerce.",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

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

webpack compiled successfully
```

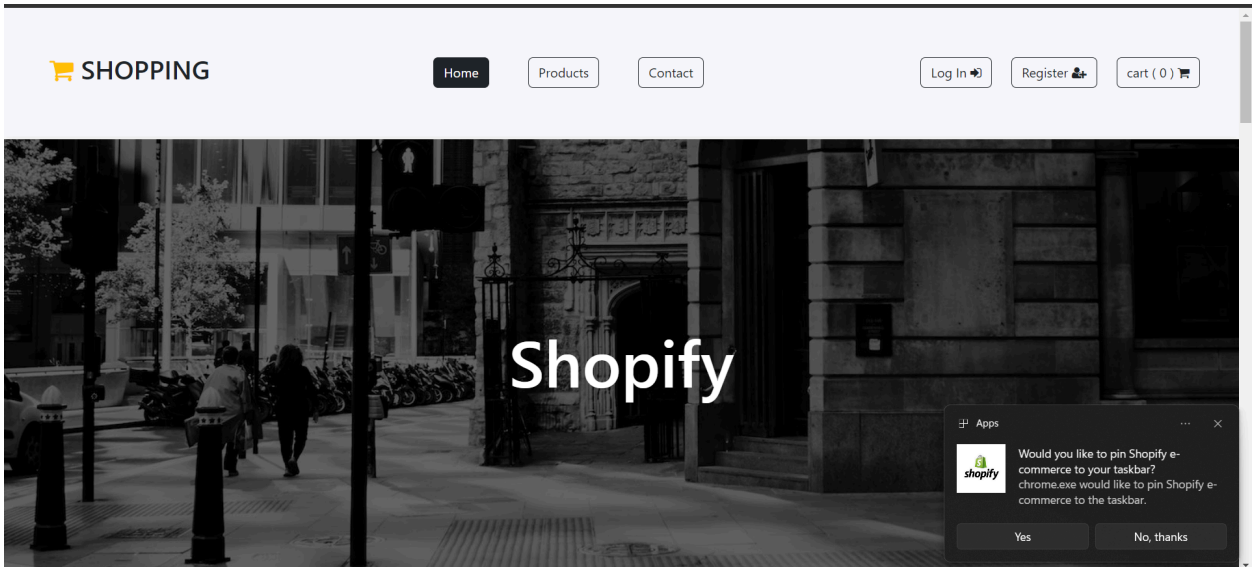








Conclusion: Therefore, I created my first Progressive Web App by adding metadata to the index.html of my e-commerce web app and installed the Application on my local machine.



Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Experiment 8

Name: Rachana Rane
Div: D15A
Roll no: 48

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

Theory:-

Service Worker

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

Things to note about Service Worker:

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

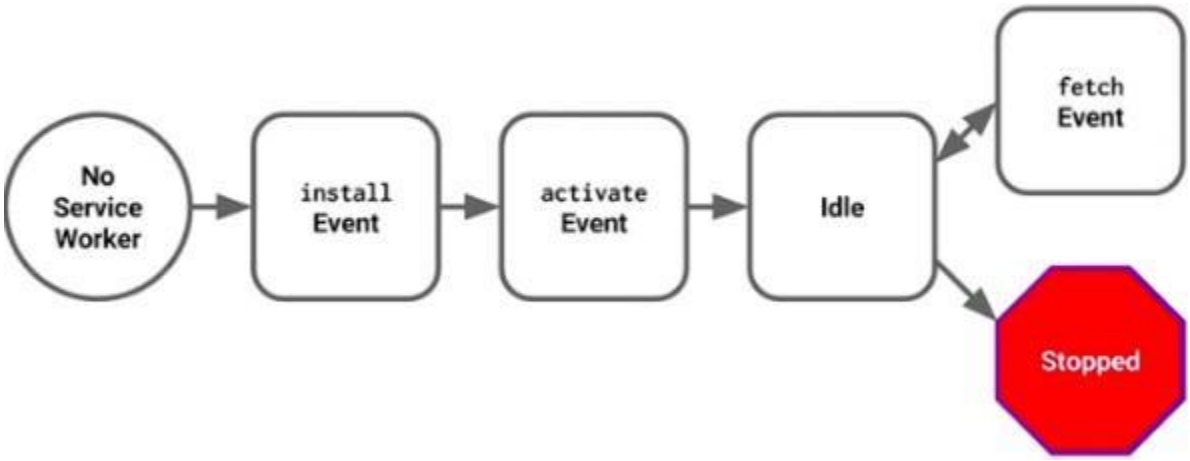
What can we do with Service Workers?

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

What can’t we do with Service Workers?

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

Service Worker Cycle



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

- Registration
- Installation
- Activation

Registration

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

main.js

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

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

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

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

main.js

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

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

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

main.js

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

Installation

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

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

Code:

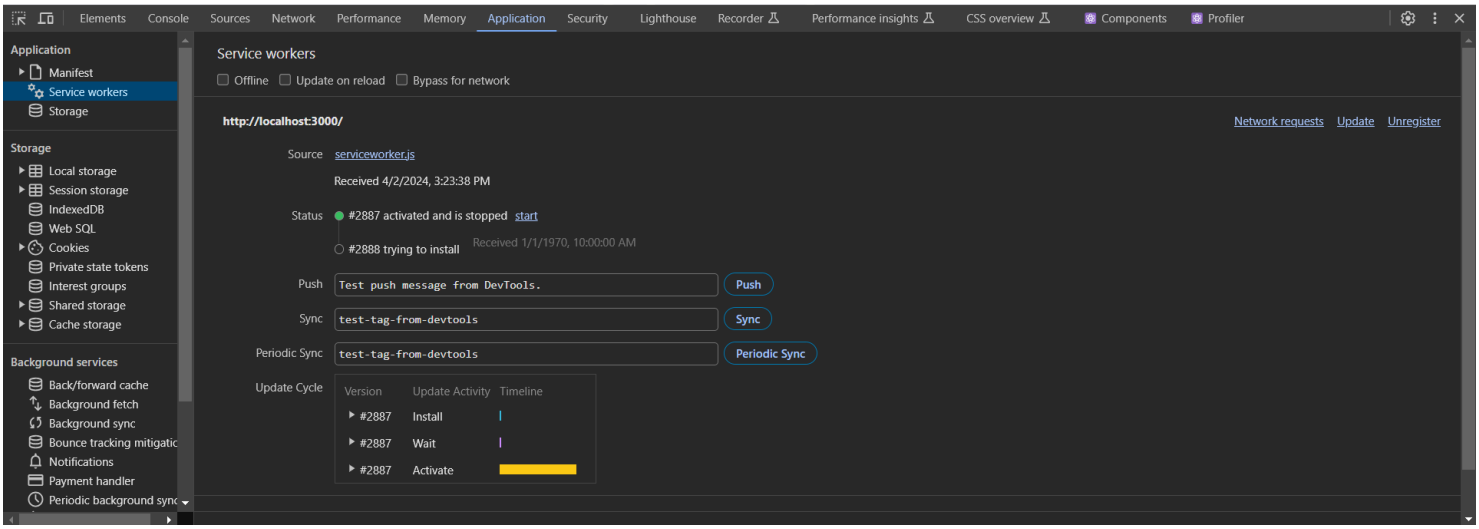
index.html

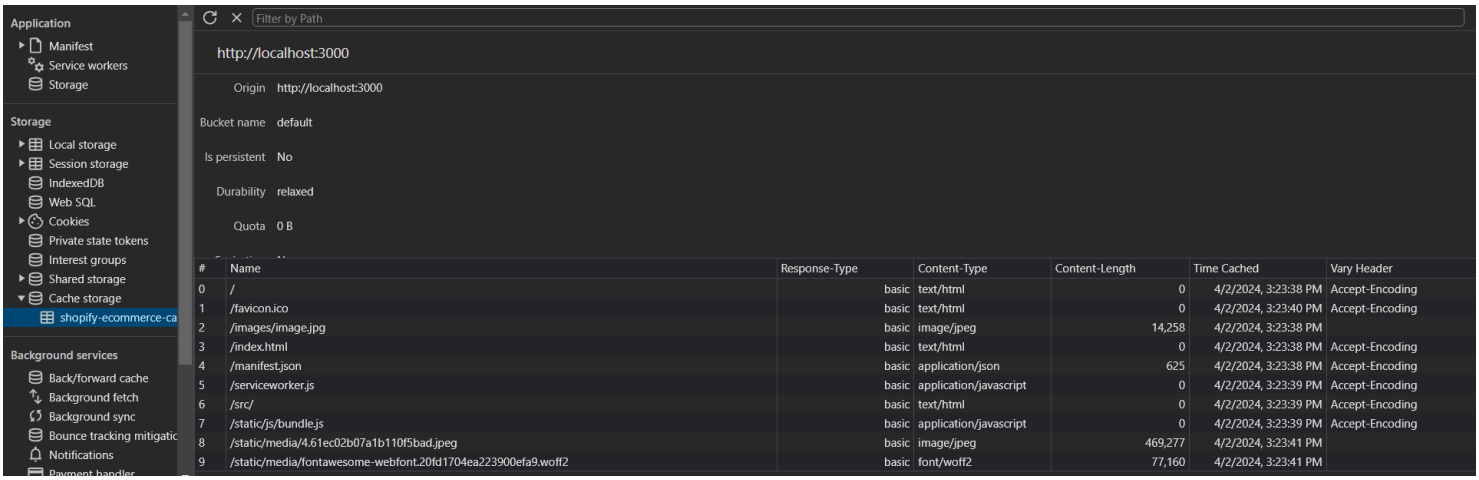
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web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

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

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

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

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

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

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

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

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

Output:





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

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	09
Experiment Title.	To implement Service worker events like fetch, sync and push for E-commerce PWA
Roll No.	
Name	
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

Name: Rachana Rane
Div: D15A
Roll no: 48

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

Theory:-

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

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

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

information messages to the page.

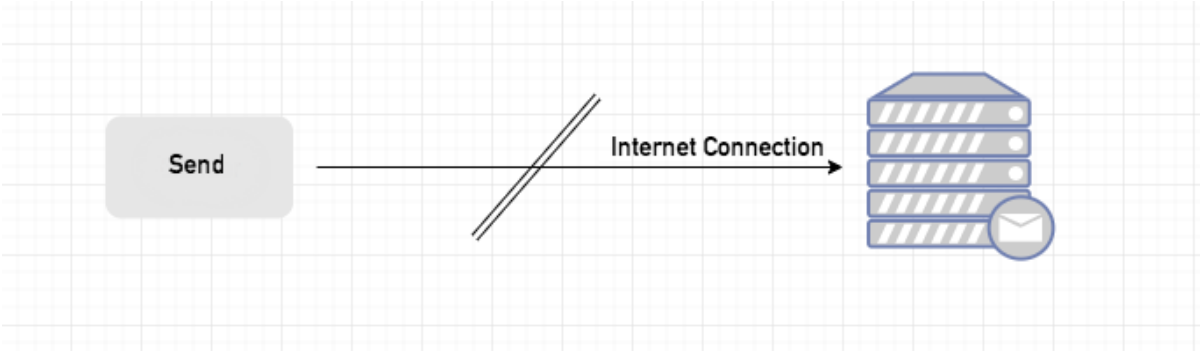
```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);

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

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

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

Sync Event

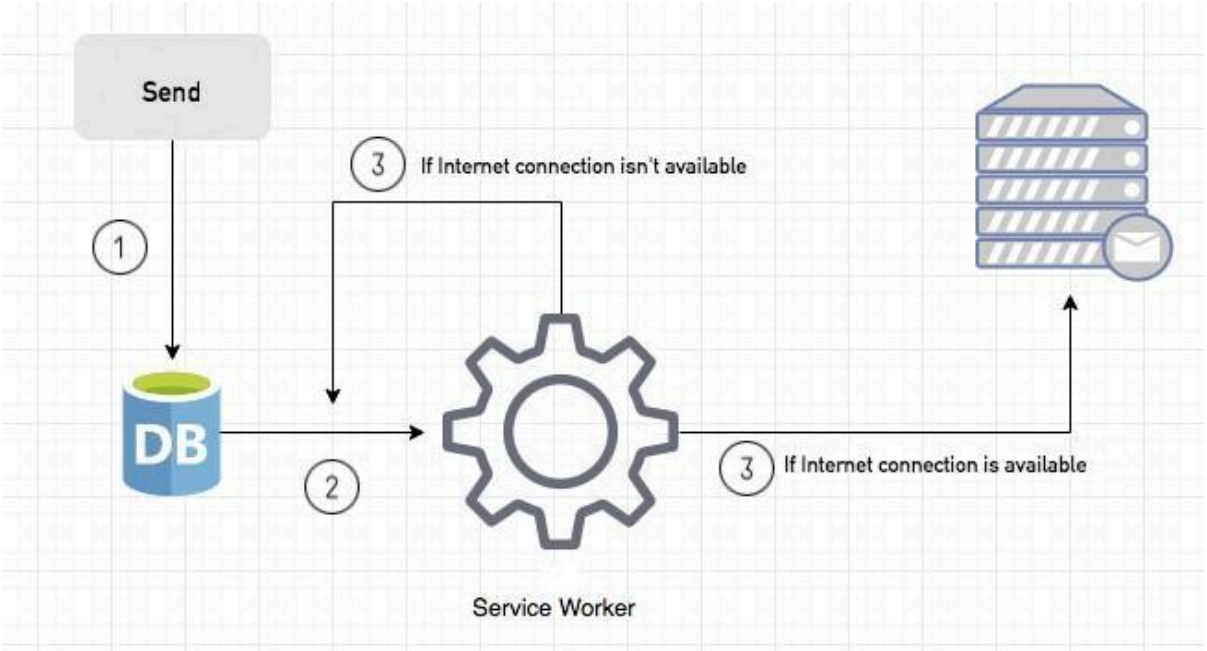
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are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

Here is a job for the Background Sync.

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



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



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

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

Event Listener for Background Sync Registration

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

Event Listener for sw.js

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

Push Event

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

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

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

Code:

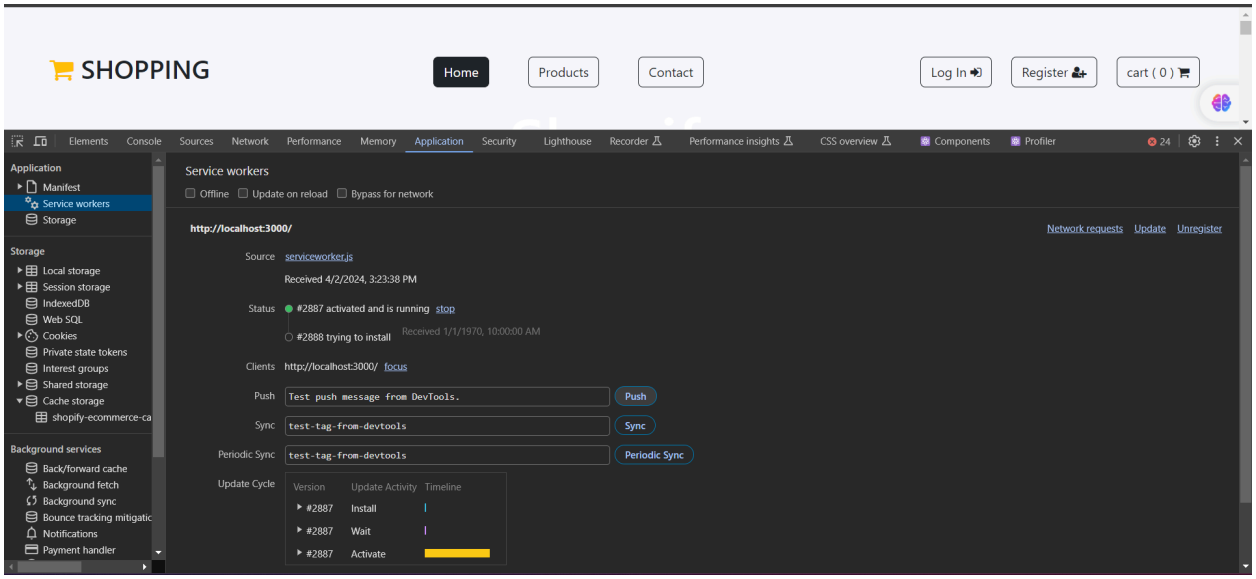
index.html

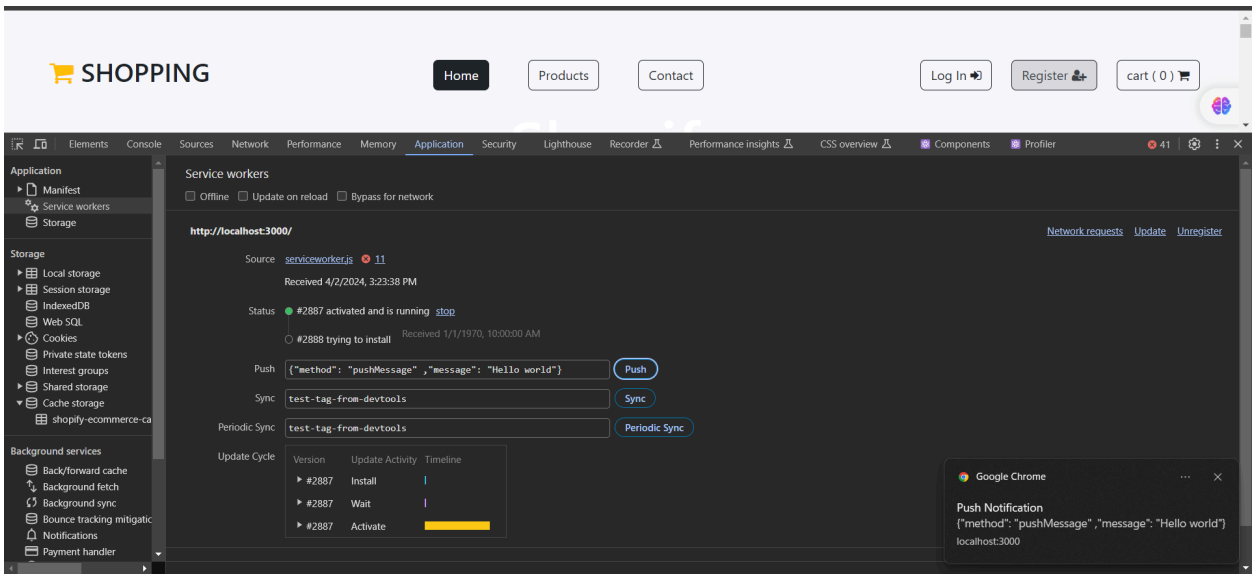
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web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

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

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

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

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

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

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

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

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

Output:





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

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Experiment 10

Name: Rachana Rane
Div: D15A
Roll no: 48

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

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

Theory:

GitHub Pages

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

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

Pros

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

Cons

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

Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

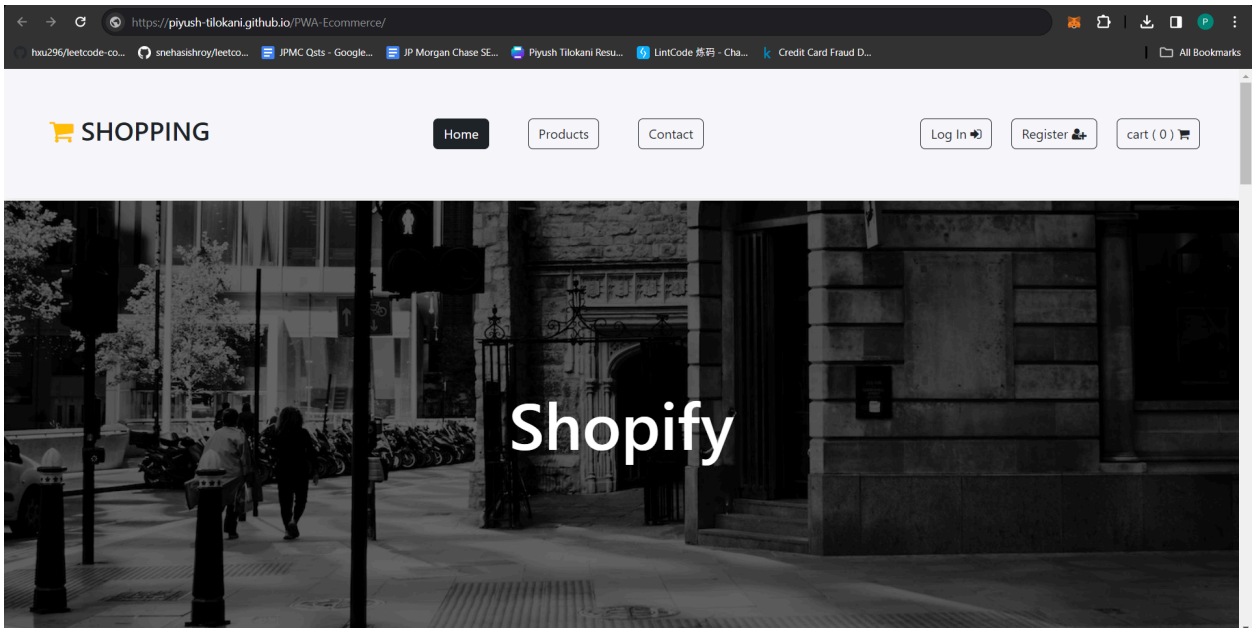
Instacart, 9GAG, and Twitch are some of the popular companies that use Firebase
Firebase has a broader approval, being mentioned in 1215 company stacks & 4651 developers stacks

Pros

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

Cons

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



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

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Experiment 11

Name: Rachana Rane
Div: D15A
Roll no: 48

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

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

Google Lighthouse :

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

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

Key Features and Audit Metrics

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

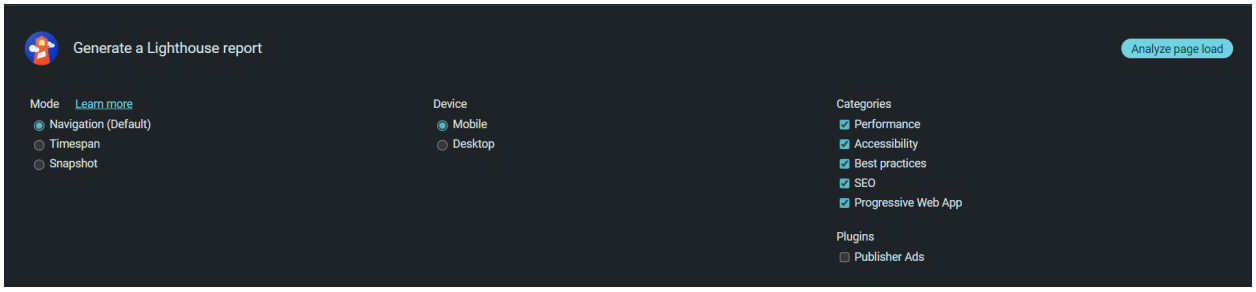
1. **Performance:** This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score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

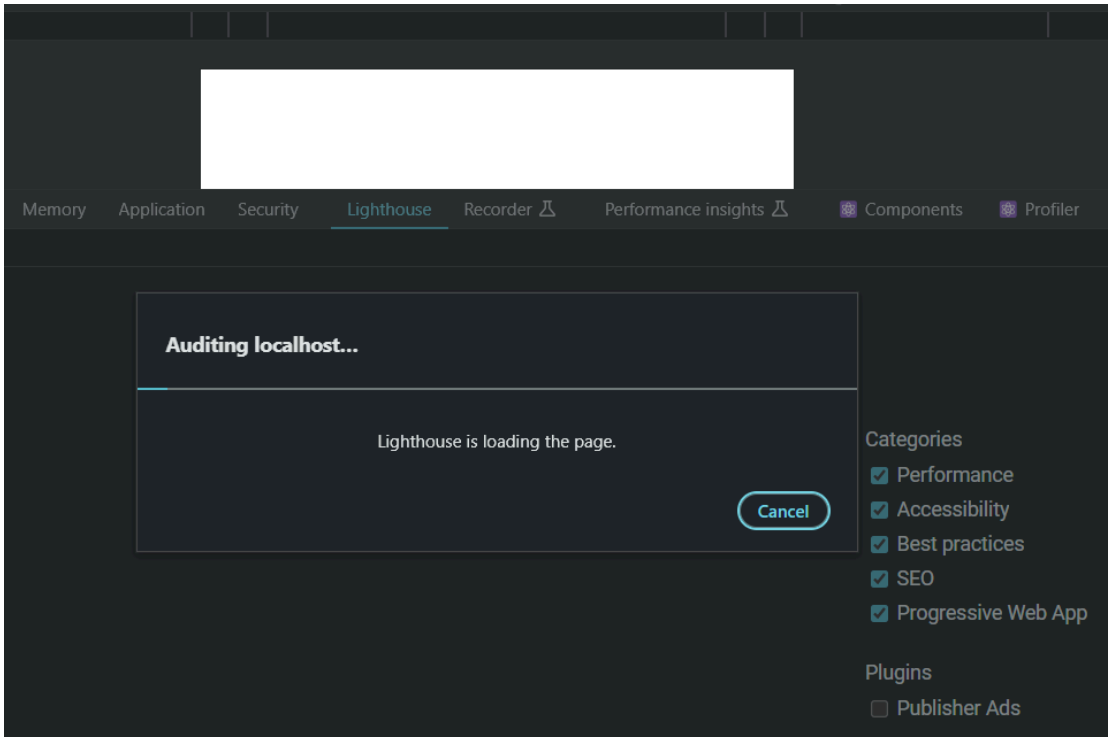
3. **Accessibility:** As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the ‘aria-’ attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics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

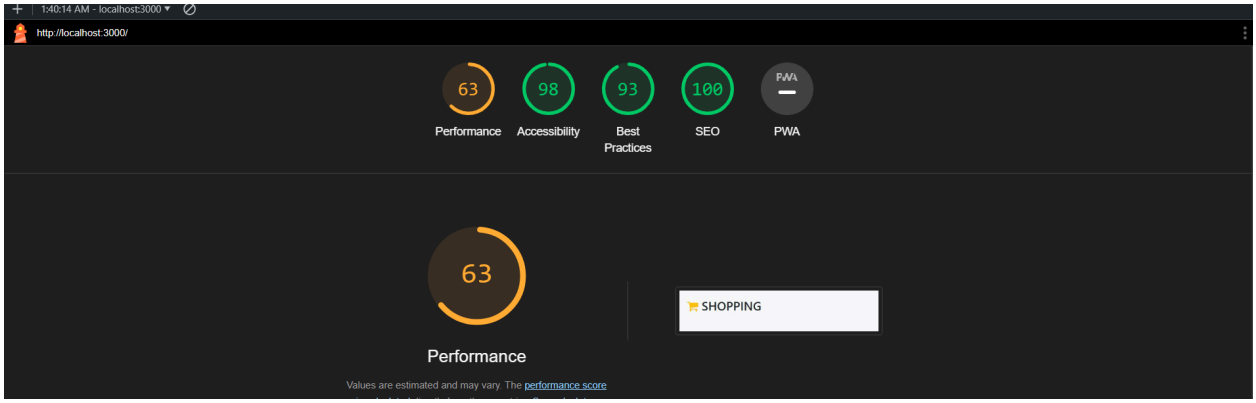
4. **Best Practices:** As any developer would know, there are a number of practices that have been deemed ‘best’ based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS

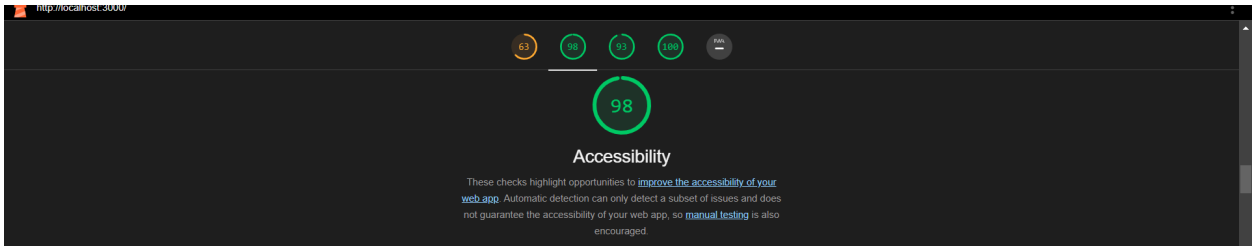
Avoiding the use of deprecated code elements like tags, directives, libraries, etc. Password input with paste-into disabled

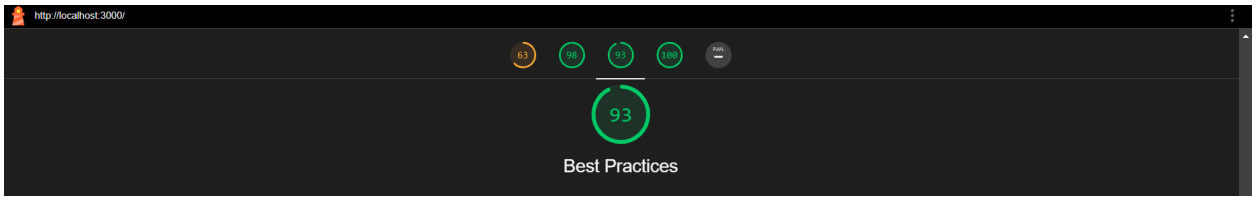
Geo-Location and cookie usage alerts on load, etc.

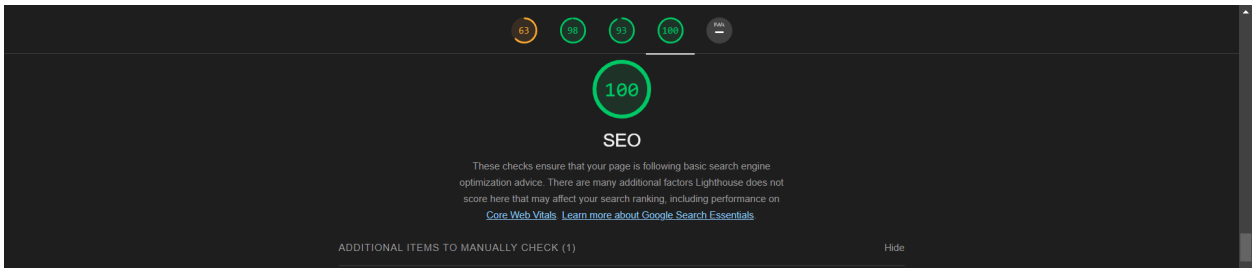


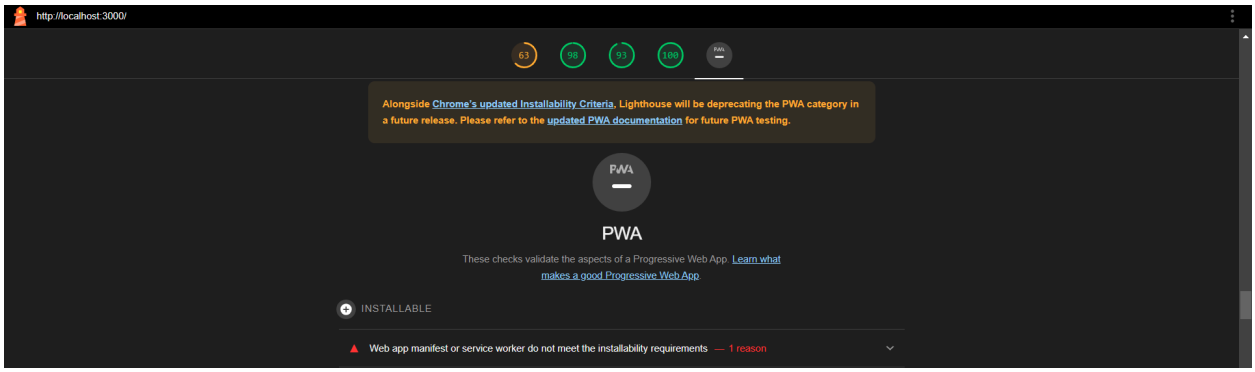

Memory
Application
Security
Lighthouse
Recorder
Performance insights
Components
Profiler
Auditing localhost...
Lighthouse is loading the page.
Cancel
Categories
Performance
Accessibility
Best practices
SEO
Progressive Web App
Plugins
Publisher Ads


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


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


http://localhost:3000/
93
Best Practices

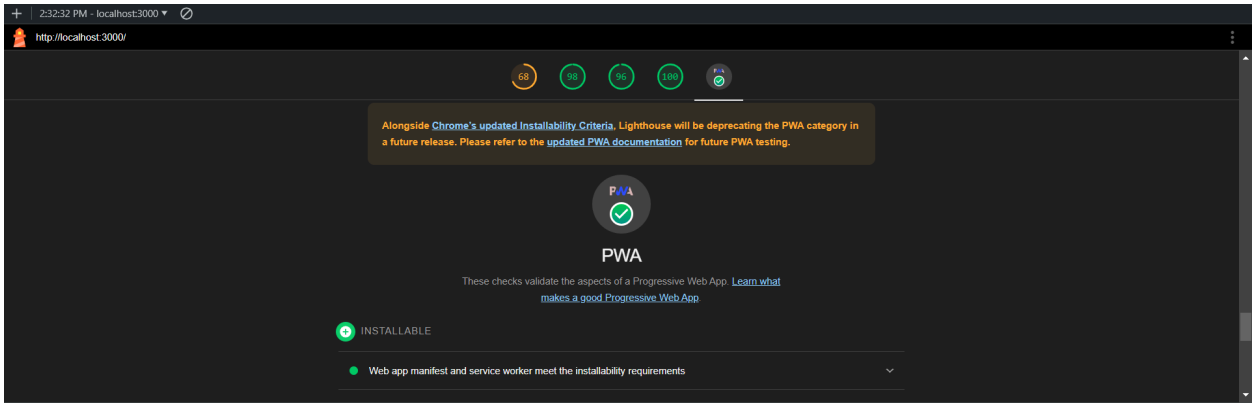




Changes made to manifest.json

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

```
    }
]
}
```



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

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Name- Rachana Rane
D15A
Roll no -48

DATE

Assignment - 1

1 Flutter Overview:-

Flutter is an open-source UI software development tool-kit created by google that allows developers to build natively compiled applications for mobile, web, and desktop from a single codebase. Here are some key features and advantages of using flutter for mobile app development.

Key Features:-

① Single Codebase for multiple platforms.
Flutter enables the development of cross-platform applications with a single codebase. Developers can write code once and deploy it on both iOS and Android platforms.

② Hot reload:-
One of the standout feature of flutter is its Hot Reload capability. It allows developers to see the changes they make in code immediately reflected in the app without restarting the entire application.

③ Rich Set of Widgets-
Flutter provides a comprehensive set of customizable widgets that facilitate the creation of complex and beautiful user interface.

④ Expressive UI:-
Flutter allows developers to create highly expressive & flexible UI designs. It provides a rich set of pre-designed widgets that can be customized to meet specific design require-ments.

FOR EDUCATIONAL USE

Scanned with OKEN Scanner

⑤ Dart Programming Language:-
Flutter uses dart programming language which is known for its simplicity and efficiency.

⑥ Access to native features:
Flutter enables developers to access native features and APIs, allowing them to integrate device-specific functionalities seamlessly.

⑦ Community Support:
Flutter has a growing and active community. This ensures ongoing support, a wealth of third party packages, & a collaborative environment for developers.

⑧ Performance:
Flutter apps are compiled to native ARM code, providing high performance & efficiency.

Advantages:-

① Faster development:
With a single codebase and hot reload, developers can iterate quickly, reducing developing time & efforts.

② Consistent UI Across platform:
Flutter's widgets provide a consistent look & feel across different platforms, eliminating the need for platform-specific UI development.

③ Cost-Effective -
Building cross-platform apps with flutter can be more cost-effective than developing seperate codebases for iOS & Android.

FOR EDUCATIONAL USE

Scanned with OKEN Scanner

④ Easy maintenance:-
Maintaining a single codebase is more straightforward than managing mutiple codebases for different platforms.

⑤ Wide range of plugins-
Flutter has growing ecosystem of plugins that allow developers to easily integrate various functionalities into their applications.

Differences from Traditional Approaches-

① No Bridge Needed-
Unlike some hybrid frameworks, Flutter dosen't rely on a bridge to communicate with native components. This contributes to better performance

② Widget-Based Architecture:-
Flutter bas uses a widget ~~bades~~ based architecture where entire UI is a composition of nested widgets.

③ Compiled to Native Code:-
Flutter apps are compiled to native ARM code, offering native performance without sacrificing the benefits of cross-development platform.

Popularity:-
① Developer Community
② Growing Community
③ Backing of Google.
④ Success Stories.

FOR EDUCATIONAL USE

Scanned with OKEN Scanner

Q.2] Widget Tree and Composition. Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets & their role in creating a widget tree.

→ In Flutter, the widget tree is a fundamental concept that represents the hierarchy & structure of user interface components within an application.
Widgets are the building blocks of the user interface, & they are organized in tree-like structure known as widget tree.

Flutter follows a compositional architecture where complex UIs are built by combining simpler widgets. This composition of widgets allows developers to create highly customizable & reusable UI components.

Widget Composition -

① Containe widgets
- It is a versatile widget that can contain other widgets. It's commonly used to define size, padding, margin & decoration of UI elements

② Column & row widgets
- 'Column' & 'Row' widgets allow developers to arrange multiple widgets vertically or

FOR EDUCATIONAL USE

Scanned with OKEN Scanner

or horizontally, respectively.

③ ListView Widget:-
It is used for scrolling lists. It can contain a large number of children.

④ Stack widget-
It allows widgets to be overlaid on top of each other.

⑤ AppBar widget –
It represents the top bar.

Widget Tree Example –

```
Column(
  children: [
    Container(
      child: Text('Welcome to Flutter'),
      padding: EdgeInsets.all(16.0),
      decoration: BoxDecoration(
        colour: Colours.blue,
        border radius: BorderRadius.circular(8.0)
      ),
    ),
    Image.asset('assets/flutter_logo.png'),
    ElevatedButton(
      onPressed: (){
        // Handle button click
      },
      child: Text('Click me'),
    ),
  ],
)
```

FOR EDUCATIONAL USE

REEGAL

Scanned with OKEN Scanner

Q.3.

→ Importance of State Manager in Flutter:

State Management is crucial in Flutter applications as it involves handling and updating the internal data of an application. Efficient state management ensures that the user interface reflects the latest changes in the application's data and maintains a consistant user experience.

Comparison of State Management Approaches:

① SetState:

Overview:- It is the most straight forward & built-in state management approch in Flutter. It is used within Stateful widgets & allows developers to rebuild the UI when the internal state changes.

Suitability:-

- Suitable for medium-sized applications.
- Ideal for scenarios where a simple, efficient & scalable state management solution is needed.
- Effective for dependency injection & providing global state.
- Ideal for Scenarios where state changes are localized to a specific widget or a small subtree of the widget tree.

② Provider:

Overview: The Provider package is a popular

FOR EDUCATIONAL USE

& lightweight state management Solution for flutter. It usese the 'Inherited Widget' to efficiently propagate & listen to changes in the application State.

- Suitability:
  - Suitable for medium to large-sized applications.
  - Ideal for scenarios where is a simple, efficient & scalable State management Solution is needed.
  - Effective for dependency injection & providing global state.

③ Riverpod:

- Overview: Riverpod is an extention of Provider, introducing more advanced features like global state management, lazy loading, & improved dependency injection. It aims to provide a simplified & declarative approach to state management.
- Suitability:
  - Suitable for medium to large-sized applications.
  - Ideals for scenarios where a more powerful & flexible state management solution is required.

FOR EDUCATIONAL USE

Scanned with OKEN Scanner

Q.8)

→ Firebase integration in flutter:

Integrating firebase with a Flutter application involves a series of steps to set up firebase in the project, configure authentication, and use firebase services for various functionalities. Here's a general process:

1. Create a firebase Project
2. Configure firebase in flutter
   Add the firebase configuration files to your flutter project.
   Add the necessary dependencies in your flutter pubspec.yaml file.
3. add dependencies
4. Run flutter pub get to fetch the dependencies
5. Initialize firebase in the App.

Benefits of using firebase as Backend solution:

1. Serverless architecture
2. Real-time database
3. Authentication
4. Cloud firestore
5. Cloud functions
6. Cloud storage
7. Analytics & crash
8. Data Synchronization in firebase.

FOR EDUCATIONAL USE

Scanned with OKEN Scanner

Project Title: **Roll No.**

MAD & PWA Lab
Journal

Experiment No.	Assignment-2
Assignment 2 Questions	1. Define Progressive Web App (PWA) and explain its significance in modern web development.Discuss the key characteristics that differentiate PWAs from traditional mobile apps 2. Define responsive web design and explain its importance in the context of Progressive Web Apps.Compare and contrast responsive, fluid, and adaptive web design approaches. 3. Describe the lifecycle of Service Workers, including registration, installation, and activation phases. 4. Explain the use of IndexedDB in the Service Worker for data storage.
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	

Rachana Rane
D15A
Roll no-48

Q.1] Define progressive web app (PWA) and explain its significance in modern web development. Discuss the key ~~that~~ characteristics that differentiate PWAs from traditional mobile apps.

→ A progressive web App (PWA) is a type of web Application that uses modern web technologies to provide users with native app-like experience directly through their web browser.

The Significance of PWAs in modern web development lies in their ability to bridge the gap between web & native mobile applications.

Key characteristics:-

① Cross platform Compatibility: PWAs are built using web technologies like HTML, CSS and JavaScript, making them compatible with various platforms & devices, including desktops, Smartphone & tablets.

② Responsiveness:-

PWAs are designed to adapt seamlessly to different screen sizes and resolutions, providing a consistent user experience across devices.

8 4

③ Offline functionality:-

PWA can cache resources and content, enabling users to access the app even when they're offline or have poor internet connection.

④ Progressive enhancement:-

PWAs employ a progressive enhancement strategy, meaning they can deliver basic functionality to all users while providing enhanced features to users with modern browsers or devices.

Sundaram
FOR EDUCATIONAL USE
Scanned with OKEN Scanner

Rachana Rane
D15A
Roll no-48

Q.1] Define progressive web app (PWA) and explain its significance in modern web development. Discuss the Key ~~that~~ characteristics that differentiate PWAs from traditional mobile apps.

→ A progressive web App (PWA) is a type of web Application that uses modern web technologies to provide users with native app-like experience directly through their web browser.

The Significance of PWAs in modern web development lies in their ability to bridge the gap between web & native mobile applications.

Key characteristics:-

① Cross platform Compatibility: PWAs are built using web technologies like HTML, CSS and JavaScript, making them compatible with various platforms & devices, including desktops, Smartphone & tablets.

② Responsiveness:-
PWAs are designed to adapt seamlessly to different screen sizes and resolutions, providing a consistent user experience across devices.

③ Offline functionality:-
PWA can cache resources and content, enabling users to access the app even when they're offline or have poor internet connection.

④ Progressive enhancement:-
PWAs employ a progressive enhancement strategy, meaning they can deliver basic functionality to all users while providing enhanced features to users with modern browsers or devices.

FOR EDUCATIONAL USE

Scanned with OKEN Scanner

⑤ Installability: PWAs can be installed on a user's device directly from the browser without the need of appstore.

⑥ Discoverability:-
PWAs are discoverable through search engines, social media, & links, making them more accessible to users compared to tradition mobile apps that often require installation from as app store.

⑦ Push notifications:-
PWAs can send push notifications to users, increasing engagement & enabling real-time communication with users even when the app is not open.

② Define responsive web design & explain its importance in the context of progressive web Apps. Compare & contrast responsive, fluid, & adaptive web design approaches.

→ Responsive web design is an approach to web development aimed at creating websites that provide an optimal viewing and interaction experience across a wide range of devices & screen sizes.
In the context of PWAs, responsive web design is crucial for ensuring that the app functions & looks good on various devices, including desktops, smartphones, & tablets.

Sundaram
FOR EDUCATIONAL USE
Scanned with OKEN Scanner

① Responsive web design:
- Uses flexible layouts & fluid grids to adapt the layout and content of a website to different screen sizes.
- Achieves responsiveness through CSS media queries that adjust styles based on screen width, height, & orientation.
- Provides a seamless user experience across devices by optimizing content layout & the functionality.

② Fluid web design:-
- Similar to responsive design, fluid design uses flexible layouts & fluid grids to adapt to different screen sizes.
- Focuses more on fluidity & flexibility, with elements scaling smoothly to fill available space.
- Does not rely heavily on fixed breakpoints like responsive design but instead allows elements to resize continuously based on available space.

③ Adaptive web design:
- Involves creating multiple layouts or designs tailored to specific screen sizes or device categories.
- Uses server-side techniques to detect the user's device & deliver the appropriate layout or design.
- Provides a more targeted and optimized experience for each device category but requires more effort to develop & maintain multiple versions of websites.

Sundaram
FOR EDUCATIONAL USE
Scanned with OKEN Scanner

Q.3 Describe the lifecycle of Service Workers, including registration, installation, and activation.

→ (1) Registration:

- The first step in the lifecycle of a Service Worker is registration. This occurs in the main script of a web application, typically the main JavaScript file.
- During registration, the Service Worker script is fetched from the server & registered with the browser using the 'navigator.serviceWorker.register()' method.
- The ServiceWorker can use the 'event.waitUntil()' method to extend the installation process until all necessary resources are cached.

(2) Installation:-

- After successful registration, the browser downloads the Service Worker script & begins the installation process.
- During installation, the browser fires the 'install' event, following the Services worker to cache static assets & other resources needed to run the web application offline.

(3) Activation:

- Once the Service worker is successfully installed, the browser fires the 'activate' event.
- During activation, the Service worker can clean up any old caches or perform other

Sundaram
FOR EDUCATIONAL USE
Scanned with OKEN Scanner

necessary to ensure it is ready to control client pages.

Q4 Explain the use of IndexedDB in the Service Worker for data storage.

→ ① Persistent Storage -
IndexedDB provides persistent storage, meaning the data stored in the database persists even after the web page or the Service worker is closed or refreshed. This makes it suitable for storing data that needs to be accessed across session or even when the user is offline.

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

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

④ Large Data Handling: Indexed DB is capable of handling large amounts of data efficiently, making it suitable for applications that require storage of substantial datasets, such as offline caching of resources or user-generated content.

⑤ Transation-based: IndexedDB operation are performed within transactions, ensuring data integrity & consistency. Transactions can be used to group multiple database operations

Sundaram
FOR EDUCATIONAL USE
Scanned with OKEN Scanner

together & ensure that either all operations succeed or none of them are applied.

In context of a Service Worker, IndexedDB can be used for various purposes, including-

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

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

③ Background Synchronization :-
Storing data received from background Synchronization tasks, such as periodic updates or push notifications, and processing them asynchronously without interrupting the user's interaction with the web application.

Sundaram
FOR EDUCATIONAL USE
Scanned with OKEN Scanner