Local Resource Sharing Network

Problem Statement:

Many people own tools, books, and equipment that they rarely use, while others need these items for a short time but do not want to buy them. There is no simple and secure way for people in local communities to share resources with each other. This leads to wasted resources and unnecessary spending.

Solution:

The Local Resource Sharing Network is a platform that allows people to lend and borrow items within their community in a safe and organized way. It will help:

- Connect people who have extra resources with those who need them.
- Ensure safe lending and borrowing through user verification and secure transactions.
- Provide a rating and review system to build trust among users.
- Reduce waste and promote sustainability by encouraging shared use of items.

Example:

Rahul owns a drill machine but rarely uses it. His neighbor, Anjali, needs a drill for a small home project but does not want to buy one. Through the app, Anjali finds Rahul's drill and requests to borrow it. Rahul checks Anjali's ratings and agrees to lend it. The app keeps track of the transaction and ensures a smooth return process. Anjali completes her project without spending money on a new drill, and Rahul earns a positive review for sharing his tool.

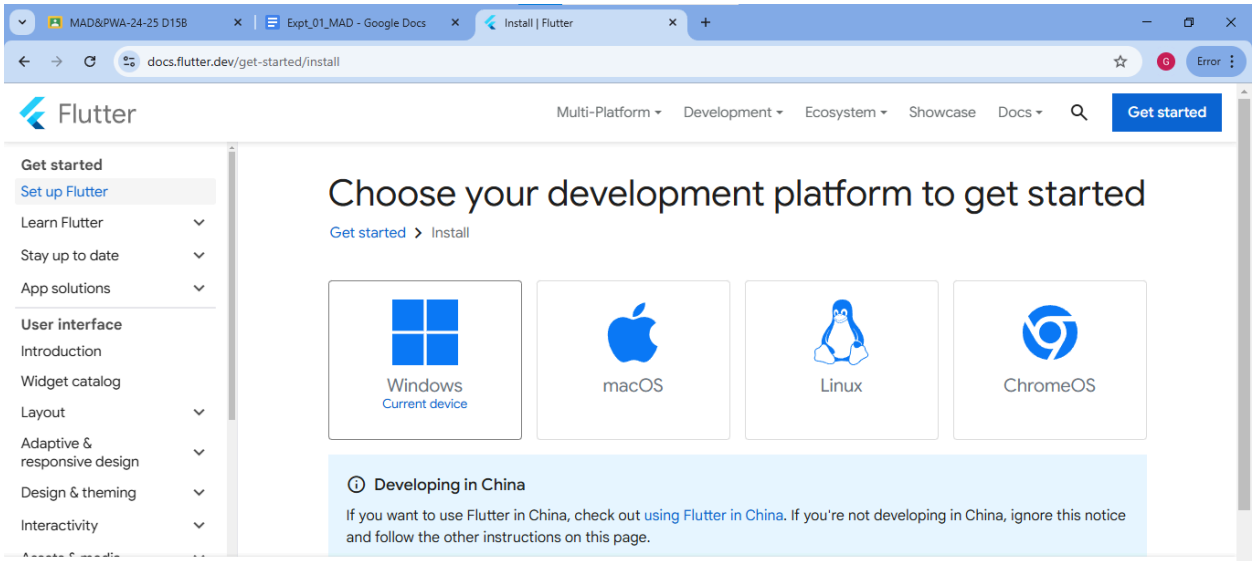
Impact:

This platform will make it easier for communities to share resources, saving money and reducing waste. It will encourage a culture of trust and cooperation, helping people make better use of the items they already have while promoting sustainability.

Aim: Installation and Configuration of Flutter Environment.

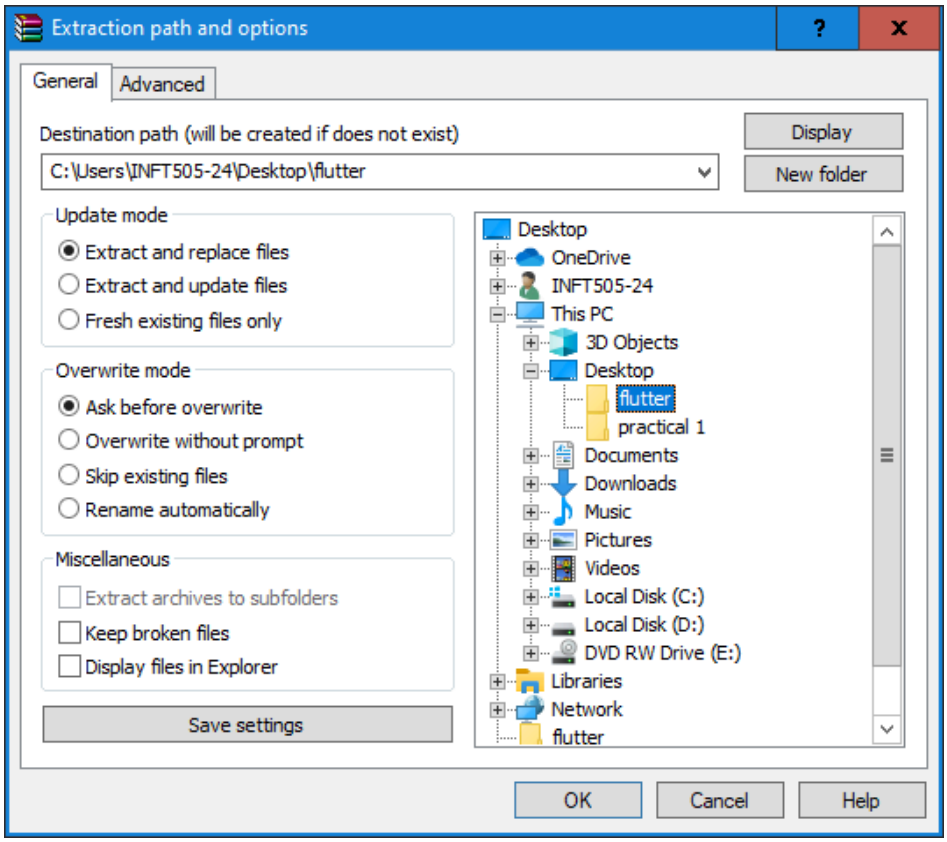
Install the Flutter SDK

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



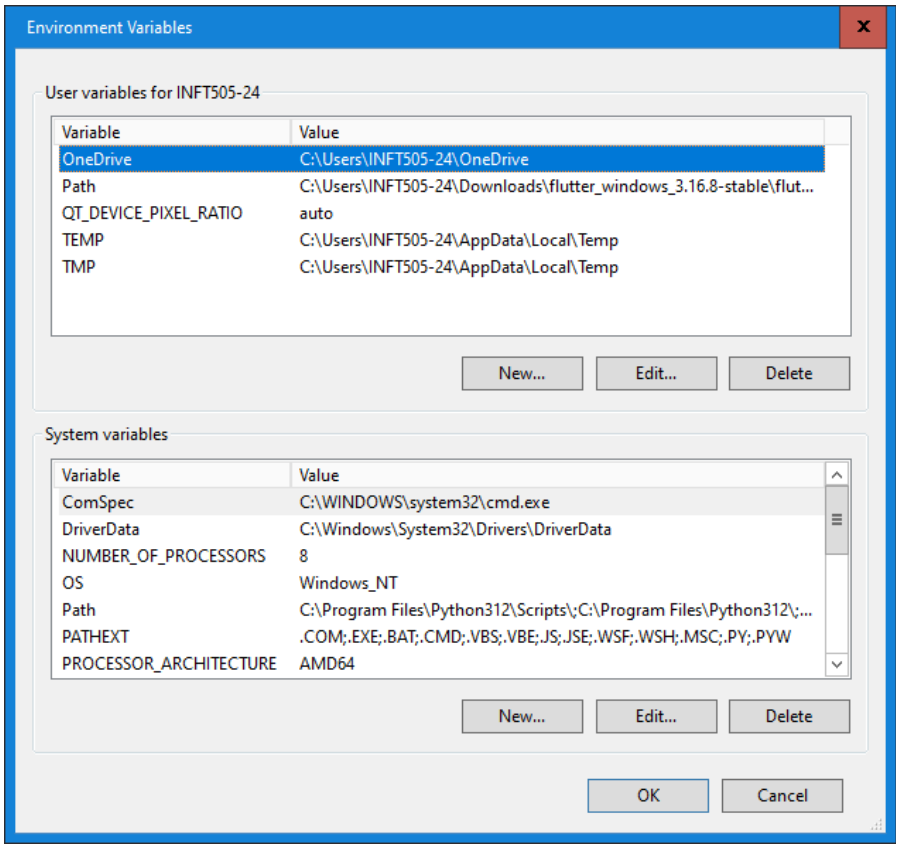
Step 2: Next, to download the latest Flutter SDK, click on the Windows icon. Here, you will find the download link for SDK.

Step 3: When your download is complete, extract the zip file and place it in the desired installation folder or location, for example, C: /Flutter.

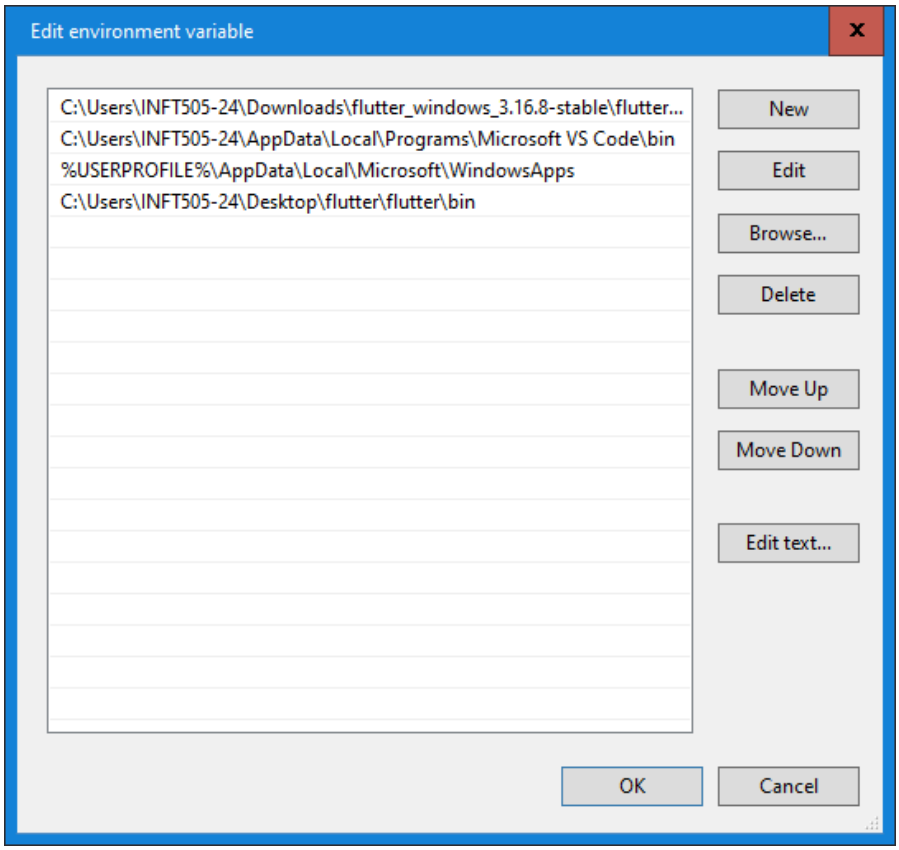


Step 4: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:

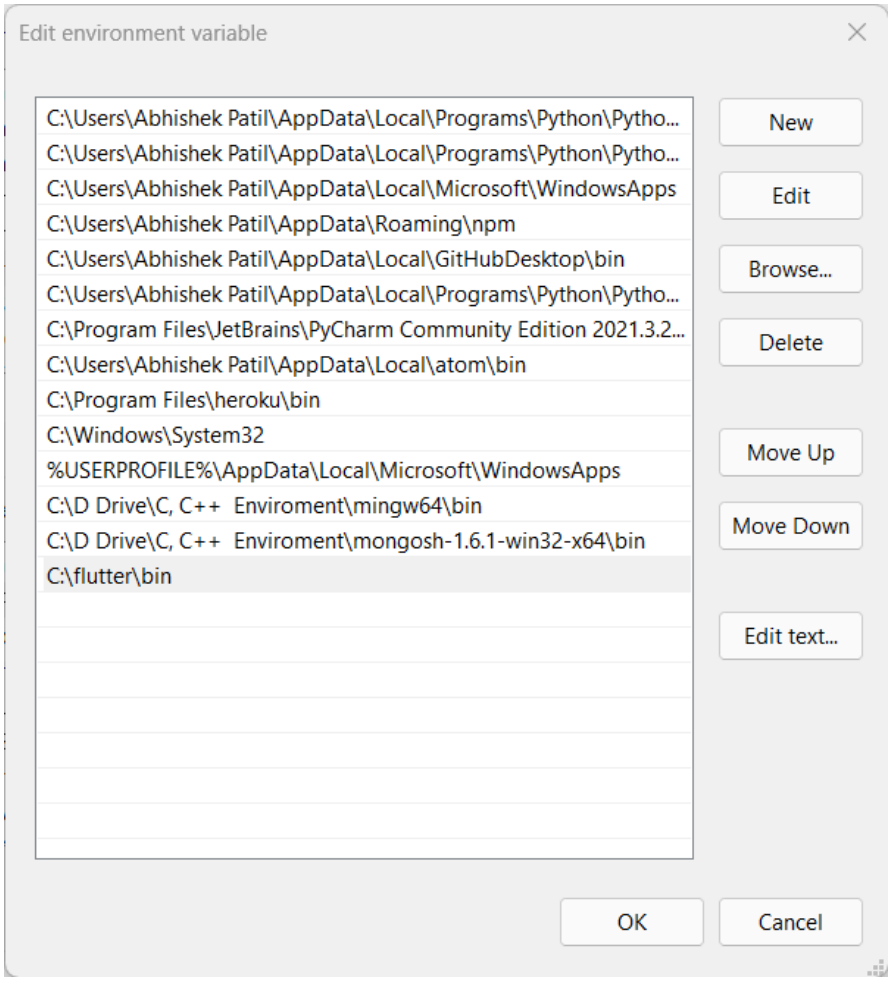
Step 4.1: Go to MyComputer properties -> advanced tab -> environment variables. You will get the following screen.



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

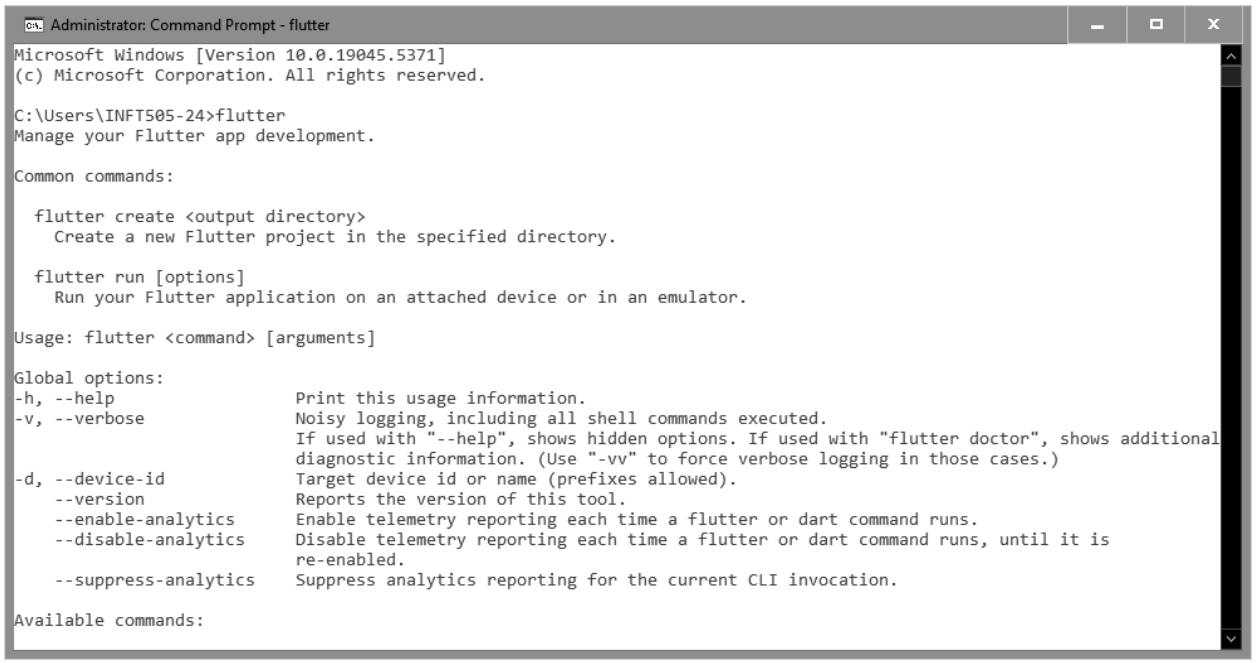


Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value -> ok -> ok -> ok.

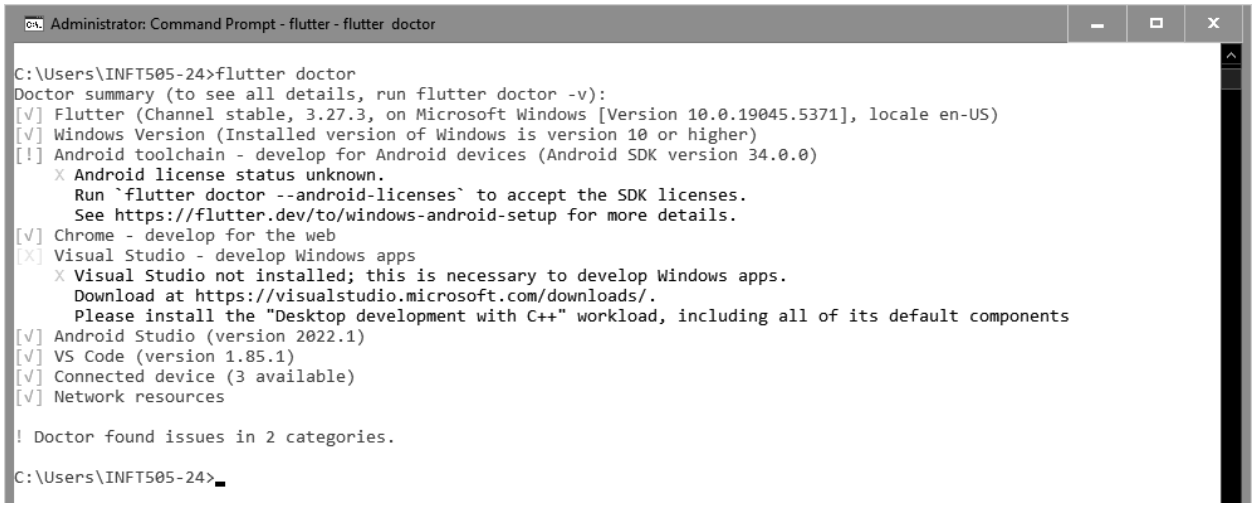


Step 5: Now, run the $ flutter command in the command prompt.

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

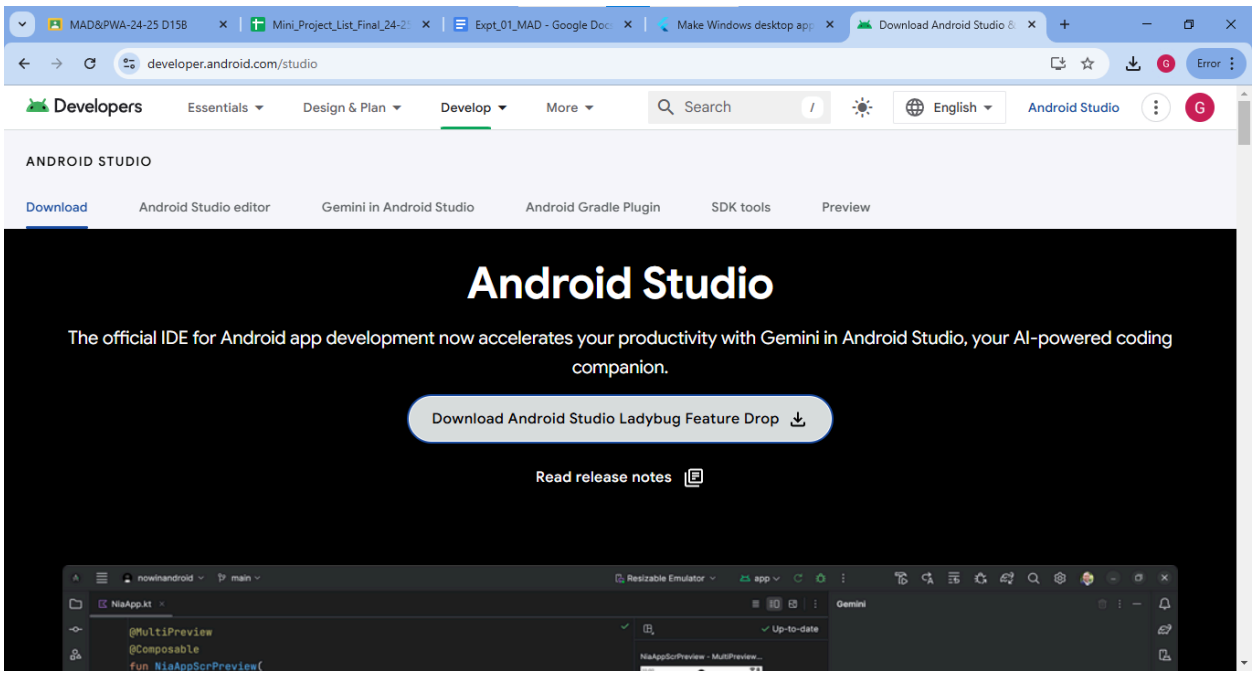


Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.



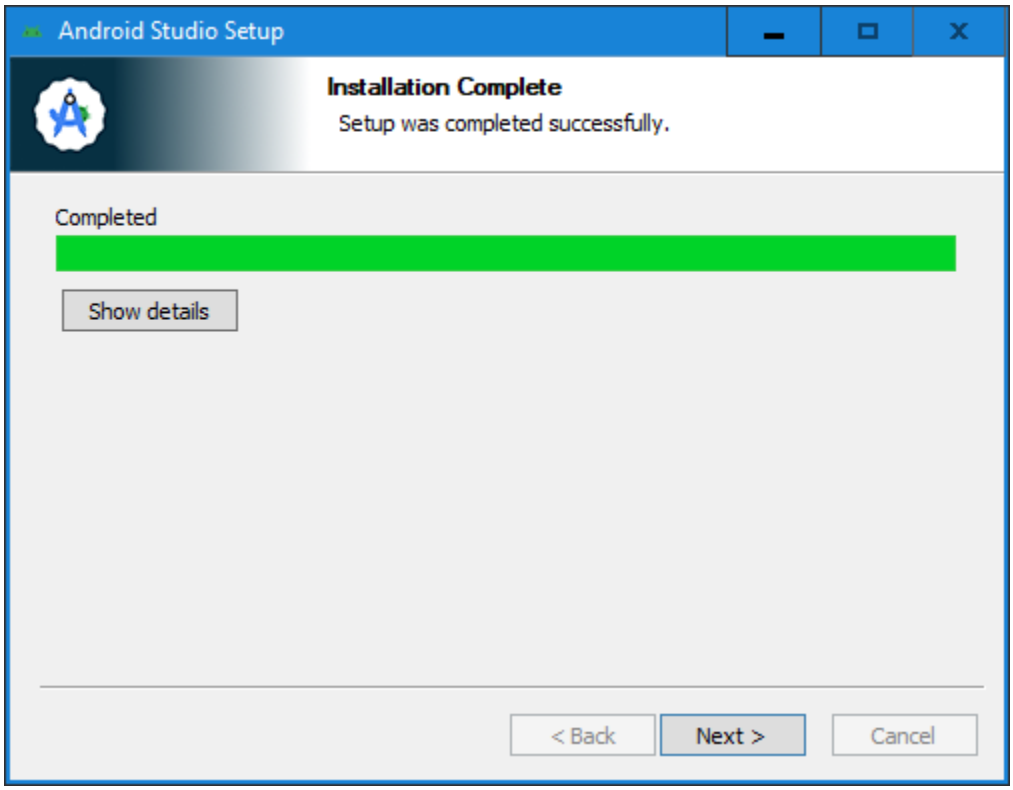
Step 7: Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.

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



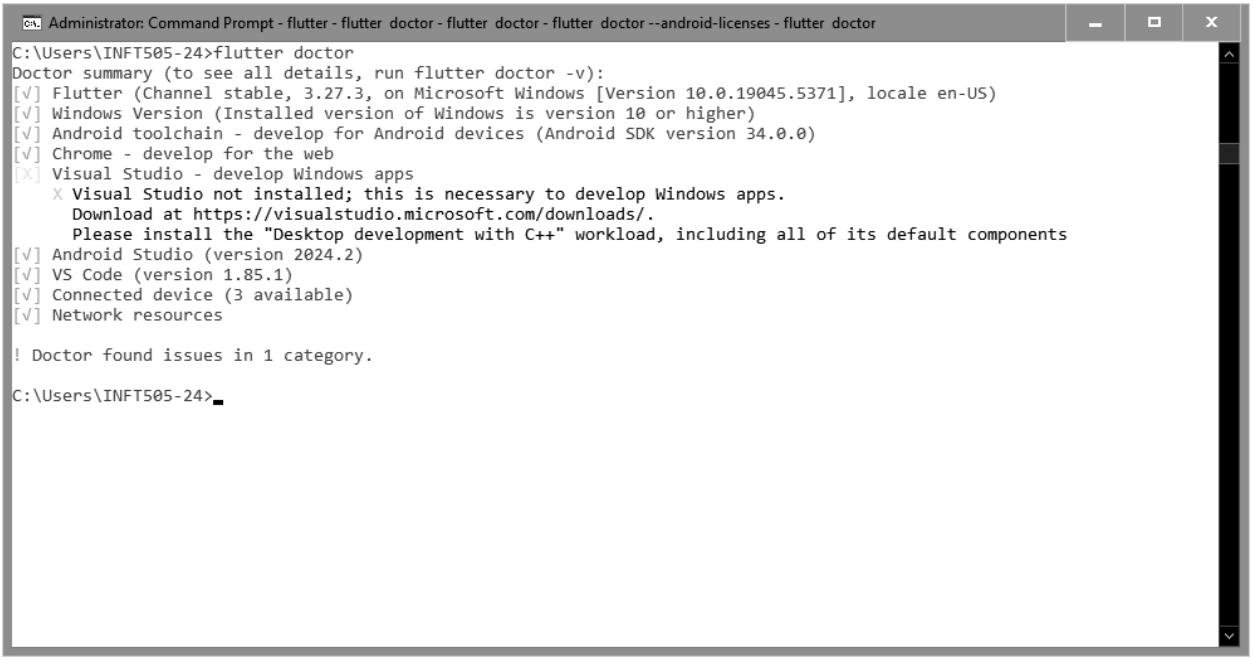
Step 7.2: When the download is complete, open the .exe file and run it. You will get the following dialog box.

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



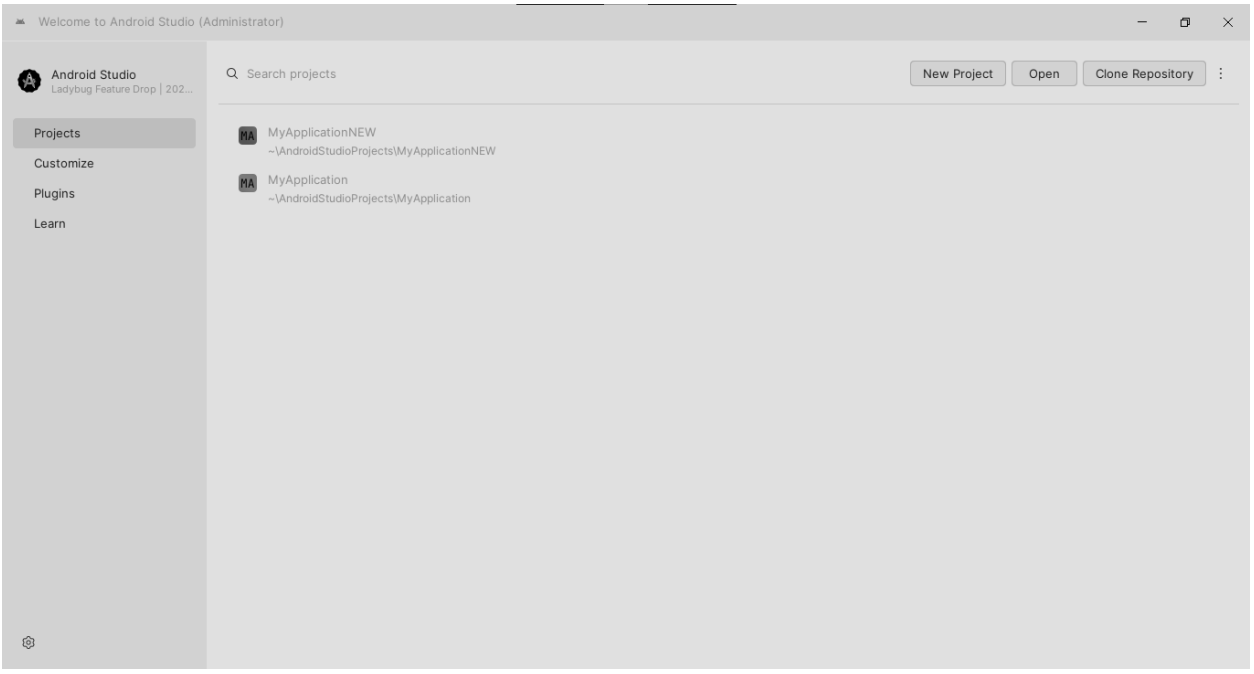
Step 7.4: In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to

step 7.5: run the $ flutter doctor command and Run flutter doctor --android-licenses command.

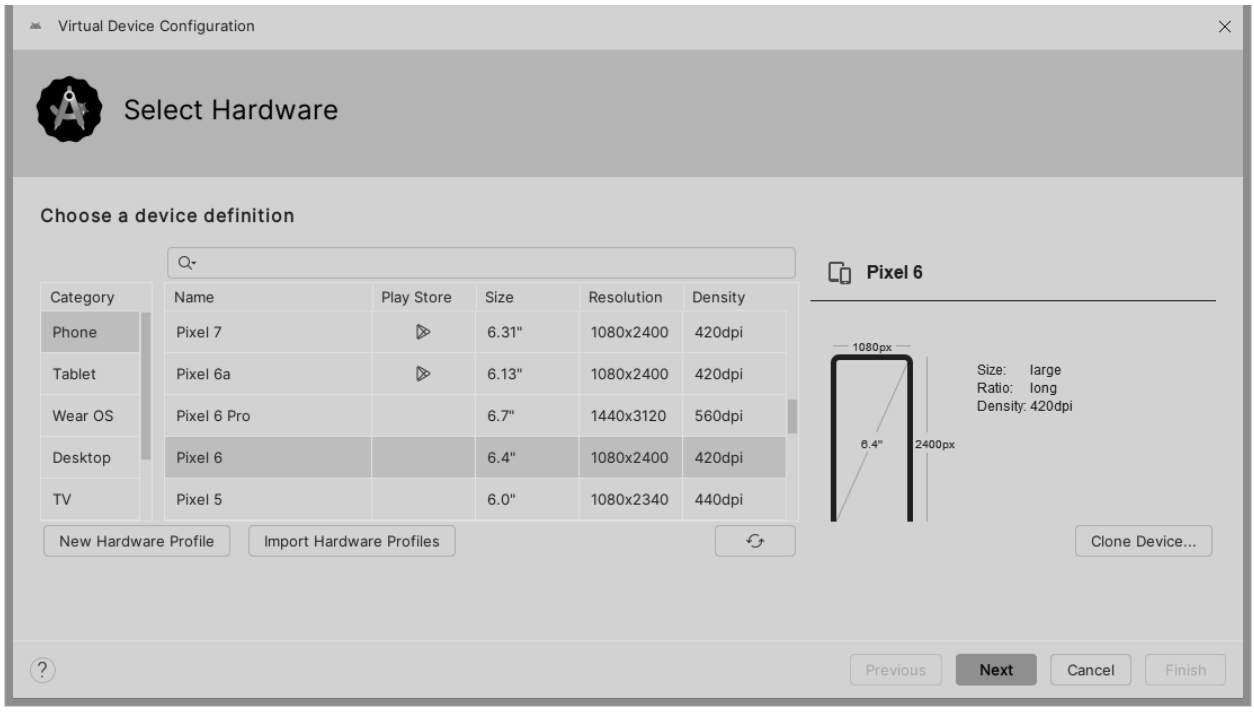


Step 8: Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.

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

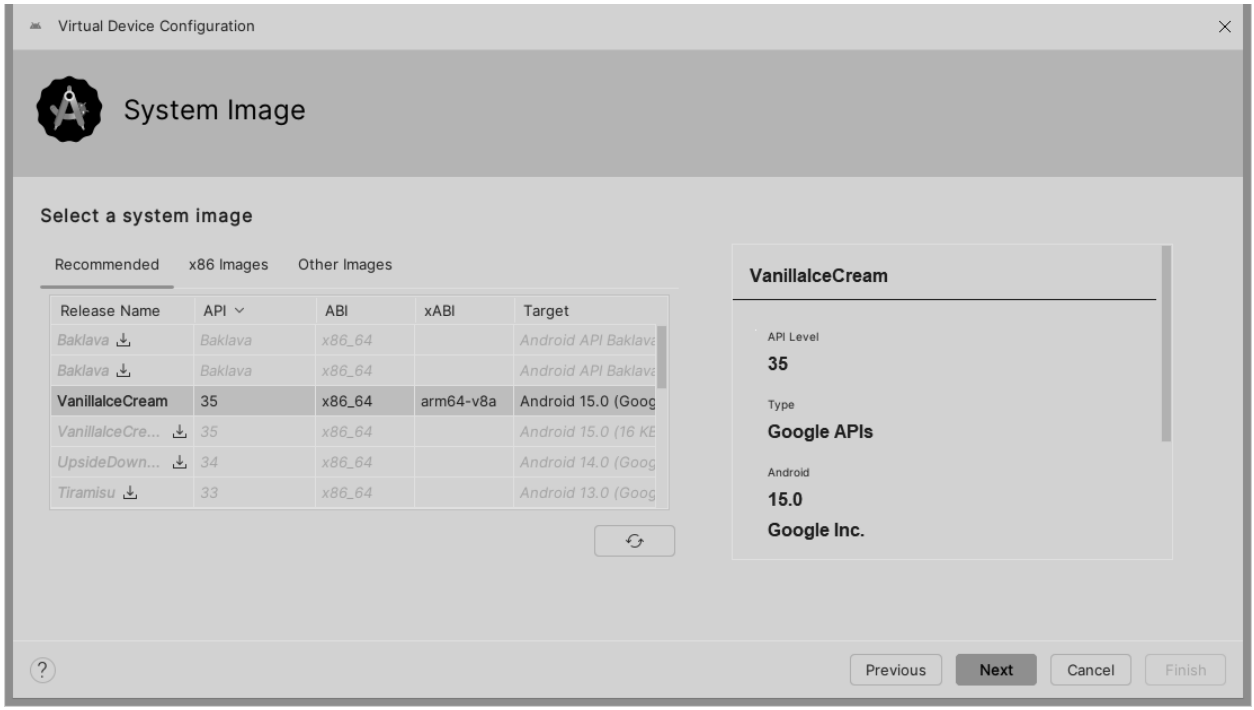


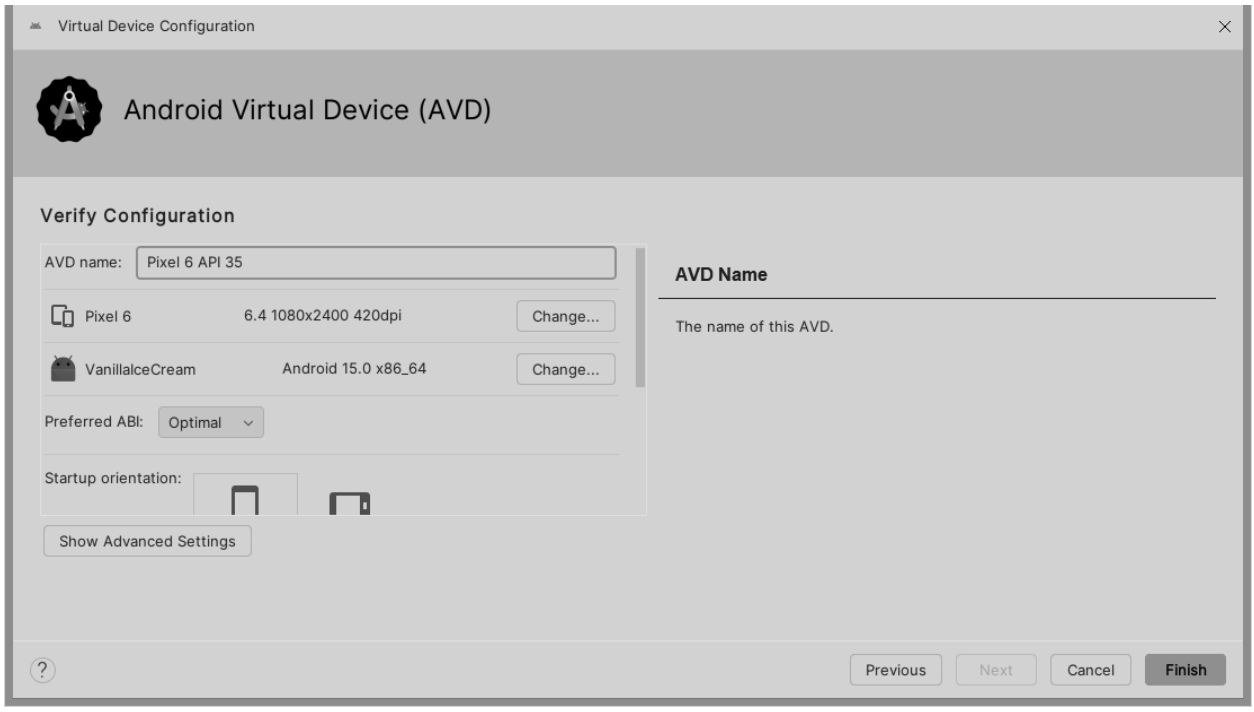
Step 8.2: Choose your device definition and click on Next.



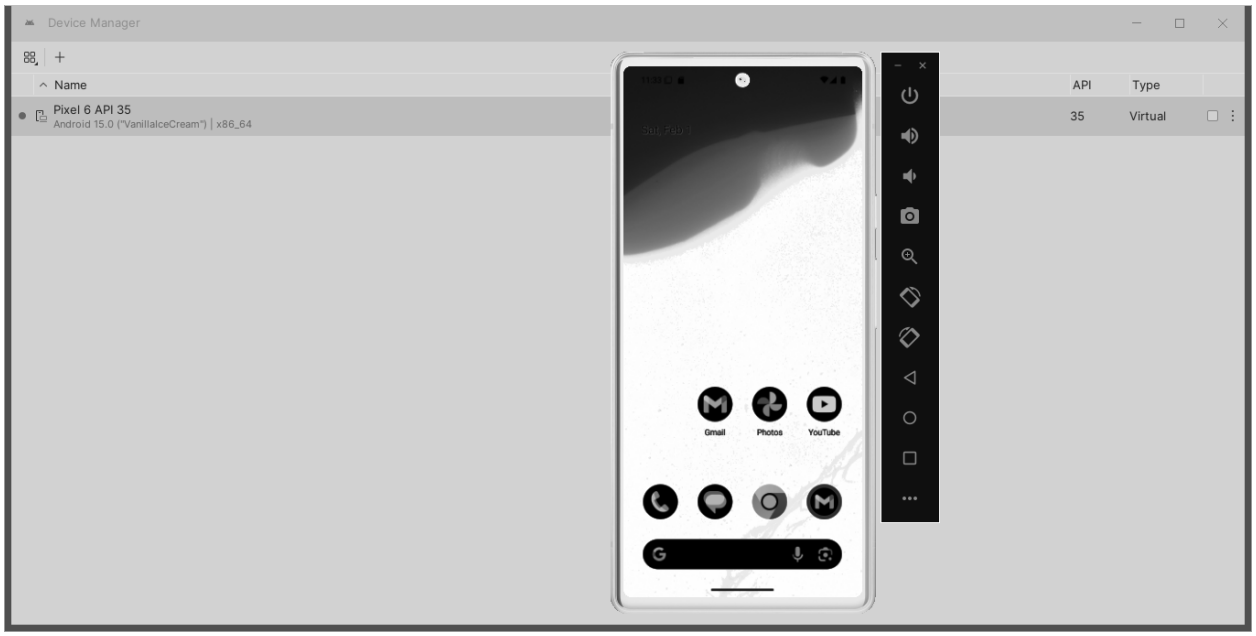
Step 8.3: Select the system image for the latest Android version and click on Next.

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



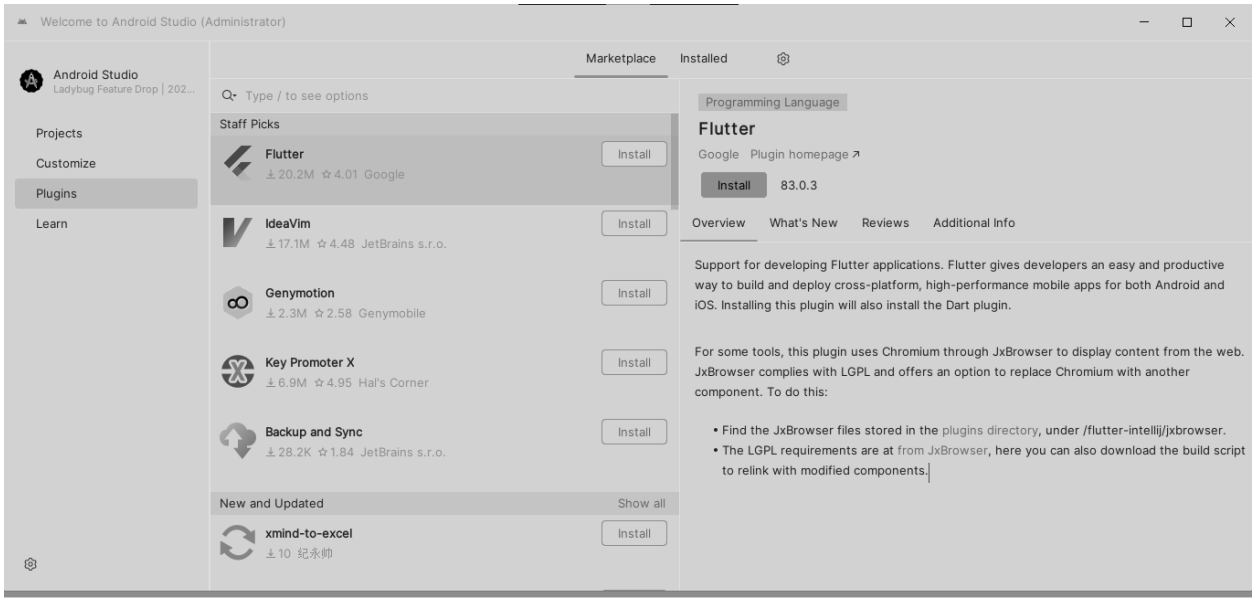


Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

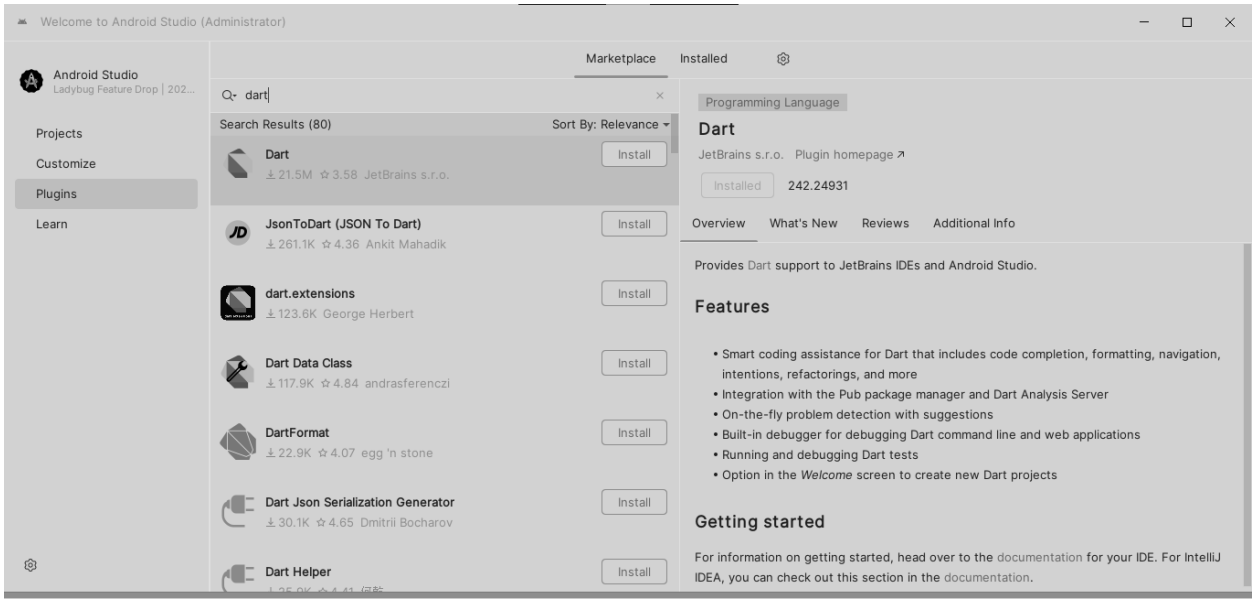


Step 9: Now, install Flutter and Dart plugin for building Flutter applications in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.

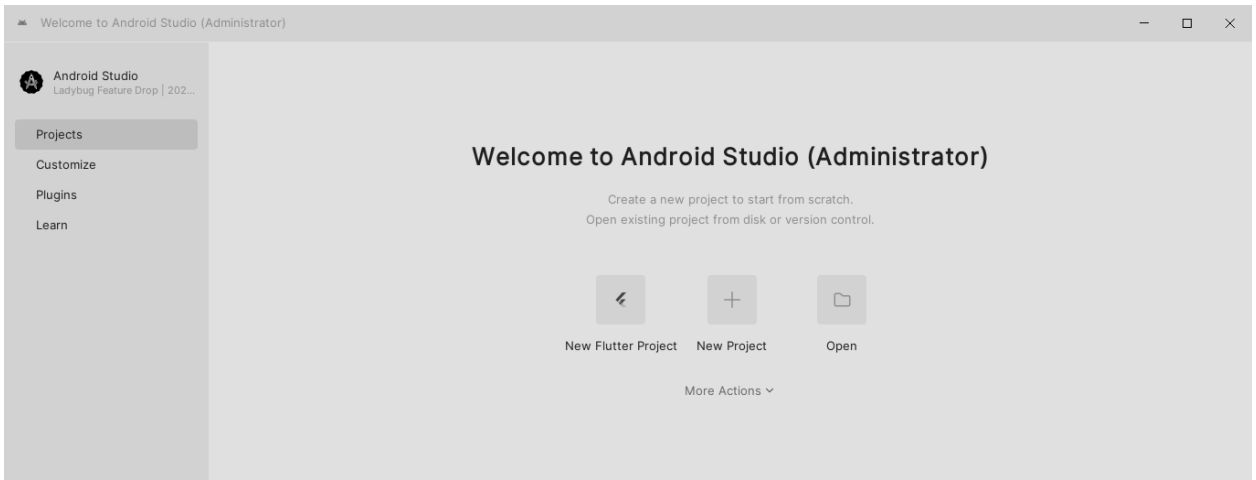
Step 9.1: Open the Android Studio and then go to File->Settings->Plugins.



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



Step 9.3: Restart the Android Studio.



Aim: To design a Flutter UI by including common widgets.

Introduction

Flutter is an open-source UI software development toolkit created by Google. It allows developers to build natively compiled applications for mobile, web, and desktop from a single codebase. One of Flutter's key strengths is its widget-based architecture, where everything in the UI is a widget, enabling a highly customizable and flexible design.

A Flutter UI is structured using **widgets**, which can be broadly classified into:

- **StatelessWidget**: A widget that does not change over time.
- **StatefulWidget**: A widget that can update dynamically based on user interaction.

Flutter applications typically use a **Scaffold widget**, which provides a structure that includes an AppBar, body, floating action buttons, bottom navigation bars, and other UI elements.

Implementation in Our Project

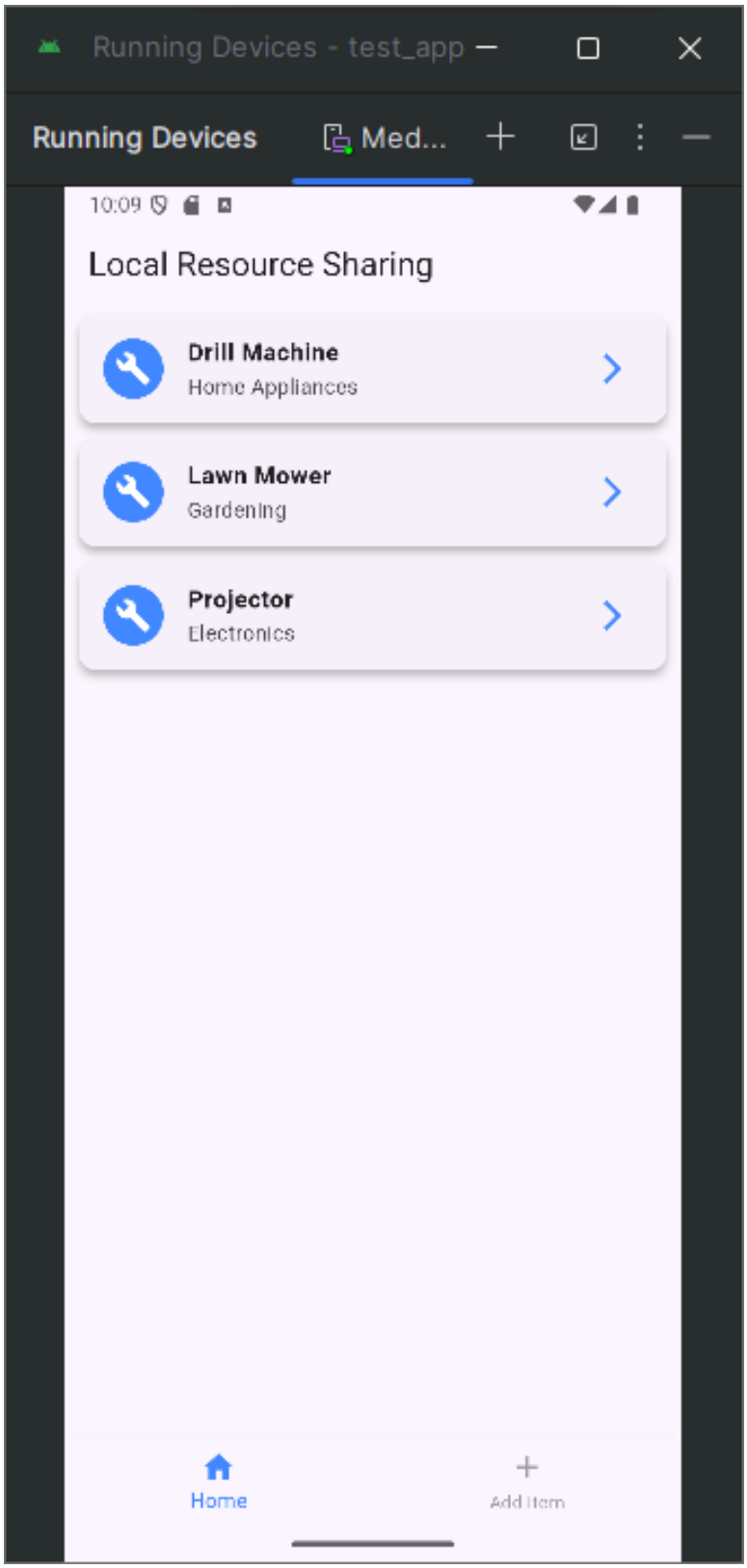
To achieve the aim of designing a Flutter UI with common widgets, we have implemented the following in our project:

1. **Scaffold and AppBar**
    - The Scaffold widget provides the basic structure for our app, including an AppBar to display the title.
2. **Bottom Navigation Bar**
    - We have integrated a BottomNavigationBar to allow users to navigate between different sections of the app smoothly.
3. **ListView and Cards**
    - The ListView widget is used to display a scrollable list of shared resources, with each item represented using a Card widget for better UI presentation.
4. **TextFields for Input**
    - The TextField widget allows users to enter item details such as name and category when adding a resource.
5. **Buttons and Snackbars**
    - We have implemented ElevatedButton to submit input, and SnackBar provides instant feedback on user actions.
6. **State Management**
    - We use the StatefulWidget to manage dynamic UI elements, such as switching between screens and updating lists.

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

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

class LocalResourceApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   debugShowCheckedModeBanner: false,
   title: 'Local Resource Sharing',
   theme: ThemeData(primarySwatch: Colors.blue),
   home: HomeScreen(),
  );
 }
}

class HomeScreen extends StatefulWidget {
 @override
 _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
 int _selectedIndex = 0;
 final List<Widget> _pages = [ResourceList(), AddItemScreen()];

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

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(title: Text('Local Resource Sharing')),
   body: _pages[_selectedIndex],
   bottomNavigationBar: BottomNavigationBar(
    items: [
     BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Home'),
     BottomNavigationBarItem(icon: Icon(Icons.add), label: 'Add Item'),
    ],
    currentIndex: _selectedIndex,
    selectedItemColor: Colors.blueAccent,
    unselectedItemColor: Colors.grey,
```

```
      onTap: _onItemTapped,
    ),
   );
  }
}

class ResourceList extends StatelessWidget {
  final List<Map<String, dynamic>> items = [
   {'name': 'Drill Machine', 'category': 'Home Appliances'},
   {'name': 'Lawn Mower', 'category': 'Gardening'},
   {'name': 'Projector', 'category': 'Electronics'},
  ];

  @override
  Widget build(BuildContext context) {
   return ListView.builder(
    itemCount: items.length,
    itemBuilder: (context, index) {
     return Card(
      elevation: 4,
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),
      margin: EdgeInsets.symmetric(horizontal: 10, vertical: 5),
      child: ListTile(
       leading: CircleAvatar(
        backgroundColor: Colors.blueAccent,
        child: Icon(Icons.build, color: Colors.white),
       ),
       title: Text(items[index]['name'], style: TextStyle(fontWeight: FontWeight.bold)),
       subtitle: Text(items[index]['category']),
       trailing: Icon(Icons.arrow_forward_ios, color: Colors.blueAccent),
       onTap: () {
        ScaffoldMessenger.of(context).showSnackBar(
         SnackBar(content: Text('${items[index]['name']} selected')),
        );
       },
      ),
     );
    },
   );
  }
}

class AddItemScreen extends StatelessWidget {
  final TextEditingController _nameController = TextEditingController();
  final TextEditingController _categoryController = TextEditingController();
```

```
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.all(16.0),
      child: Column(
        children: [
          TextField(
            controller: _nameController,
            decoration: InputDecoration(labelText: 'Item Name', border: OutlineInputBorder()),
          ),
          SizedBox(height: 10),
          TextField(
            controller: _categoryController,
            decoration: InputDecoration(labelText: 'Category', border: OutlineInputBorder()),
          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: () {
              if (_nameController.text.isNotEmpty && _categoryController.text.isNotEmpty) {
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(content: Text('Item Added: ${_nameController.text}')),
                );
                _nameController.clear();
                _categoryController.clear();
              } else {
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(content: Text('Please enter all details')),
                );
              }
            },
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.blueAccent,
              padding: EdgeInsets.symmetric(horizontal: 30, vertical: 15),
            ),
            child: Text('Add Item', style: TextStyle(color: Colors.white, fontSize: 16)),
          )
        ],
      ),
    );
  }
}
```



Conclusion

In this experiment, we implemented a Flutter UI using common widgets like Scaffold, AppBar, BottomNavigationBar, ListView, TextField, and ElevatedButton to create an interactive interface. During development, we faced issues such as improper widget alignment and navigation errors, which we resolved by adjusting layout constraints and properly managing state changes.

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

Theory:

In this experiment, we focused on enhancing the Flutter app's user interface by integrating icons, images, and fonts. These components help to create a more engaging, intuitive, and visually appealing app.

1. **Icons**: Icons serve as visual representations of actions, features, or items. In Flutter, we can use pre-defined icons from the Icons class or custom icons to represent various functionalities. Icons help improve navigation and user interaction.
2. **Images**: Images make an app more dynamic and provide visual context. Flutter allows us to include images either from the internet or local assets. Images can be used to represent items, categories, or other visual elements in the app.
3. **Fonts**: Fonts are crucial for displaying text in an aesthetically pleasing way. Flutter supports both default and custom fonts. By setting fonts in the app's theme, we can maintain consistency and style throughout the interface.

Implementation in the Code

In our code, we applied these concepts as follows:

- **Icons**: We used built-in Icons (e.g., Icons.home, Icons.add) for the app bar, navigation bar, and buttons. Custom icons were also used for representing items in the resource list, such as Icons.construction, Icons.grass, and Icons.tv.
- **Images**: We included images for each resource (e.g., Drill Machine, Lawn Mower, Projector) stored locally in the assets/images directory. These images are displayed alongside their respective details (name, category) in the app.
- **Fonts**: We used the default Roboto font throughout the app to keep the text readable and consistent.

Code:

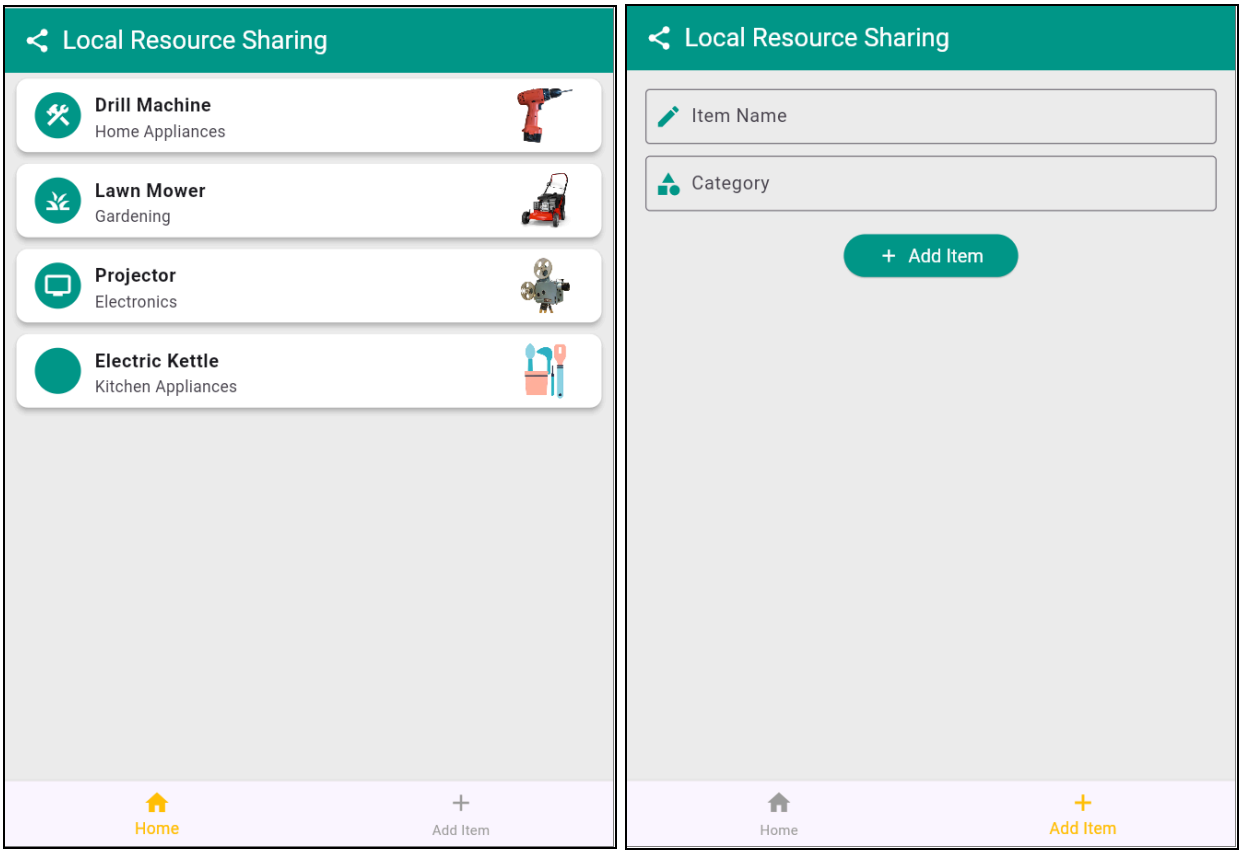
```
import 'package:flutter/material.dart';

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

class LocalResourceApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Local Resource Sharing',
      theme: ThemeData(
```

```
      primaryColor: Colors.teal,
      scaffoldBackgroundColor: Colors.grey[200],
      textTheme: TextTheme(
        bodyLarge: TextStyle(fontFamily: 'Roboto', color: Colors.black),
        bodyMedium: TextStyle(fontFamily: 'Roboto', color: Colors.black87),
      ),
    ),
    home: HomeScreen(),
   );
 }
}

class HomeScreen extends StatefulWidget {
 @override
 _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
 int _selectedIndex = 0;
 final List<Widget> _pages = [ResourceList(), AddItemScreen()];

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

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Row(
     children: [
      Icon(Icons.share, color: Colors.white),
      SizedBox(width: 10),
      Text('Local Resource Sharing', style: TextStyle(color: Colors.white)),
     ],
    ),
    backgroundColor: Colors.teal,
```

```
    ),
    body: _pages[_selectedIndex],
    bottomNavigationBar: BottomNavigationBar(
     items: [
      BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Home'),
      BottomNavigationBarItem(icon: Icon(Icons.add), label: 'Add Item'),
     ],
     currentIndex: _selectedIndex,
     selectedItemColor: Colors.amber,
     unselectedItemColor: Colors.grey,
     onTap: _onItemTapped,
    ),
   );
 }
}

class ResourceList extends StatelessWidget {
 final List<Map<String, dynamic>> items = [
     {'name': 'Drill Machine', 'category': 'Home Appliances', 'image': 'assets/images/drill_machine.png',
'icon': Icons.construction},
      {'name': 'Lawn Mower', 'category': 'Gardening', 'image': 'assets/images/lawn_mower.png', 'icon':
Icons.grass},
   {'name': 'Projector', 'category': 'Electronics', 'image': 'assets/images/projector.png', 'icon': Icons.tv},
    {'name': 'Electric Kettle', 'category': 'Kitchen Appliances', 'image': 'assets/images/default.png', 'icon':
null},
 ];

 @override
 Widget build(BuildContext context) {
  return ListView.builder(
   itemCount: items.length,
   itemBuilder: (context, index) {
    return Card(
     color: Colors.white,
     elevation: 4,
     shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),
     margin: EdgeInsets.symmetric(horizontal: 10, vertical: 5),
     child: ListTile(
      leading: CircleAvatar(
```

```
            backgroundColor: Colors.teal,
            child: Icon(items[index]['icon'], color: Colors.white),
          ),
          title: Text(items[index]['name'], style: TextStyle(fontWeight: FontWeight.bold)),
          subtitle: Text(items[index]['category']),
          trailing: Image.asset(items[index]['image'], width: 50, height: 50),
          onTap: () {
            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(content: Text('${items[index]['name']} selected')),
            );
          },
        ),
      );
    },
  );
 }
}

class AddItemScreen extends StatelessWidget {
 final TextEditingController _nameController = TextEditingController();
 final TextEditingController _categoryController = TextEditingController();

 @override
 Widget build(BuildContext context) {
  return Padding(
   padding: EdgeInsets.all(16.0),
   child: Column(
    children: [
     TextField(
      controller: _nameController,
      decoration: InputDecoration(
       labelText: 'Item Name',
       border: OutlineInputBorder(),
       prefixIcon: Icon(Icons.edit, color: Colors.teal),
      ),
     ),
     SizedBox(height: 10),
     TextField(
      controller: _categoryController,
```

```
          decoration: InputDecoration(
            labelText: 'Category',
            border: OutlineInputBorder(),
            prefixIcon: Icon(Icons.category, color: Colors.teal),
          ),
        ),
        SizedBox(height: 20),
        ElevatedButton.icon(
          onPressed: () {
            if (_nameController.text.isNotEmpty && _categoryController.text.isNotEmpty) {
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(content: Text('Item Added: ${_nameController.text}')),
              );
              _nameController.clear();
              _categoryController.clear();
            } else {
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(content: Text('Please enter all details')),
              );
            }
          },
          icon: Icon(Icons.add, color: Colors.white),
          label: Text('Add Item', style: TextStyle(color: Colors.white, fontSize: 16)),
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.teal,
            padding: EdgeInsets.symmetric(horizontal: 30, vertical: 15),
          ),
        )
      ],
    ),
  );
 }
}
```



Conclusion:

In this experiment, we successfully implemented icons, images, and fonts in the Flutter app, enhancing its user interface. We faced challenges with correctly linking asset images and ensuring proper font application, but resolved them by verifying asset paths and setting up the font in the pubspec.yaml file.

Aim: To create an interactive Form using a form widget.

Theory:

Forms are an essential part of mobile applications, allowing users to input and submit data. In Flutter, the Form widget is used to group multiple input fields together while managing user interactions and validation efficiently.

General Explanation

A form consists of input fields (TextField), validation logic, and submission actions. It helps in collecting structured data from users, ensuring that necessary fields are filled before processing. Forms also enhance user experience by providing feedback, such as error messages or success notifications.

Implementation in Our Code

- Used TextField widgets to collect item details (name and category).
- Implemented TextEditingController to manage input and clear fields after submission.
- Added validation checks to ensure required fields are not empty.
- Displayed feedback messages (SnackBar) to notify users of successful form submission or missing input.
- Used icons and UI styling to enhance form usability.

Code

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

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

class LocalResourceApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   debugShowCheckedModeBanner: false,
   title: 'Local Resource Sharing',
   theme: ThemeData(
    primaryColor: Colors.teal,
    scaffoldBackgroundColor: Colors.grey[200],
    textTheme: TextTheme(
     bodyLarge: TextStyle(fontFamily: 'Roboto', color: Colors.black),
     bodyMedium: TextStyle(fontFamily: 'Roboto', color: Colors.black87),
    ),
   ),
   home: AuthChecker(),
```

```
    );
  }
}

// Auth Checker to determine whether to show login or home screen
class AuthChecker extends StatefulWidget {
  @override
  _AuthCheckerState createState() => _AuthCheckerState();
}

class _AuthCheckerState extends State<AuthChecker> {
  bool isLoading = true;
  bool isLoggedIn = false;

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

  Future<void> checkLoginStatus() async {
    final prefs = await SharedPreferences.getInstance();
    final loggedIn = prefs.getBool('isLoggedIn') ?? false;

    setState(() {
      isLoggedIn = loggedIn;
      isLoading = false;
    });
  }

  @override
  Widget build(BuildContext context) {
    if (isLoading) {
      return Scaffold(
        body: Center(
          child: CircularProgressIndicator(
            color: Colors.teal,
          ),
        ),
      );
    }
```

```
    return isLoggedIn ? HomeScreen() : LoginScreen();
  }
}

// Login Screen
class LoginScreen extends StatefulWidget {
  @override
  _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final _formKey = GlobalKey<FormState>();
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  bool _isPasswordVisible = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey[200],
      body: SafeArea(
        child: Center(
          child: SingleChildScrollView(
            child: Padding(
              padding: const EdgeInsets.all(24.0),
              child: Form(
                key: _formKey,
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    // Logo and App Name
                    Icon(
                      Icons.share_rounded,
                      size: 70,
                      color: Colors.teal,
                    ),
                    SizedBox(height: 15),
                    Text(
                      'Local Resource Sharing',
                      style: TextStyle(
```

```
    fontSize: 24,
    fontWeight: FontWeight.bold,
    color: Colors.teal,
   ),
  ),
  SizedBox(height: 40),

  // Email Field
  TextFormField(
   controller: _emailController,
   keyboardType: TextInputType.emailAddress,
   decoration: InputDecoration(
    labelText: 'Email',
    hintText: 'Enter your email',
    prefixIcon: Icon(Icons.email, color: Colors.teal),
    border: OutlineInputBorder(
     borderRadius: BorderRadius.circular(10),
    ),
    focusedBorder: OutlineInputBorder(
     borderRadius: BorderRadius.circular(10),
     borderSide: BorderSide(color: Colors.teal, width: 2),
    ),
   ),
   validator: (value) {
    if (value == null || value.isEmpty) {
     return 'Please enter your email';
    }
    // Email format validation using regex
    final emailRegex = RegExp(r'^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$');
    if (!emailRegex.hasMatch(value)) {
     return 'Please enter a valid email';
    }
    return null;
   },
  ),
  SizedBox(height: 20),

  // Password Field
  TextFormField(
   controller: _passwordController,
   obscureText: !_isPasswordVisible,
```

```
    decoration: InputDecoration(
     labelText: 'Password',
     hintText: 'Enter your password',
     prefixIcon: Icon(Icons.lock, color: Colors.teal),
     suffixIcon: IconButton(
      icon: Icon(
       _isPasswordVisible ? Icons.visibility : Icons.visibility_off,
       color: Colors.grey,
      ),
      onPressed: () {
       setState(() {
        _isPasswordVisible = !_isPasswordVisible;
       });
      },
     ),
     border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
     ),
     focusedBorder: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
      borderSide: BorderSide(color: Colors.teal, width: 2),
     ),
    ),
    validator: (value) {
     if (value == null || value.isEmpty) {
      return 'Please enter your password';
     }
     if (value.length < 6) {
      return 'Password must be at least 6 characters';
     }
     return null;
    },
   ),
   SizedBox(height: 20),

   // Login Button
   SizedBox(
    width: double.infinity,
    height: 50,
    child: ElevatedButton(
     onPressed: () async {
```

```
          if (_formKey.currentState!.validate()) {
            // For demo purpose, just save login state
            final prefs = await SharedPreferences.getInstance();
            await prefs.setBool('isLoggedIn', true);
            await prefs.setString('userEmail', _emailController.text);

            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => HomeScreen()),
            );
          }
        },
        style: ElevatedButton.styleFrom(
          backgroundColor: Colors.teal,
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(10),
          ),
        ),
        child: Text(
          'Login',
          style: TextStyle(
            color: Colors.white,
            fontSize: 16,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
    ),
    SizedBox(height: 20),

    // Sign Up Link
    Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Text("Don't have an account? "),
        TextButton(
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => SignupScreen()),
            );
```

```
                    },
                    child: Text(
                     'Sign Up',
                     style: TextStyle(
                      color: Colors.teal,
                      fontWeight: FontWeight.bold,
                     ),
                    ),
                   ),
                  ],
                 ),
                ],
               ),
              ),
             ),
            ),
           ),
          ),
         ),
        );
       }
}

// Sign Up Screen
class SignupScreen extends StatefulWidget {
 @override
 _SignupScreenState createState() => _SignupScreenState();
}

class _SignupScreenState extends State<SignupScreen> {
 final _formKey = GlobalKey<FormState>();
 final TextEditingController _nameController = TextEditingController();
 final TextEditingController _emailController = TextEditingController();
 final TextEditingController _passwordController = TextEditingController();
 final TextEditingController _confirmPasswordController = TextEditingController();
 bool _isPasswordVisible = false;
 bool _isConfirmPasswordVisible = false;

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   backgroundColor: Colors.grey[200],
```

```
  appBar: AppBar(
   backgroundColor: Colors.teal,
   title: Text('Create Account'),
   foregroundColor: Colors.white,
  ),
  body: SafeArea(
   child: Center(
    child: SingleChildScrollView(
     child: Padding(
      padding: const EdgeInsets.all(24.0),
      child: Form(
       key: _formKey,
       child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
         // Name Field
         TextFormField(
          controller: _nameController,
          decoration: InputDecoration(
           labelText: 'Full Name',
           hintText: 'Enter your full name',
           prefixIcon: Icon(Icons.person, color: Colors.teal),
           border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(10),
           ),
           focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(10),
            borderSide: BorderSide(color: Colors.teal, width: 2),
           ),
          ),
          validator: (value) {
           if (value == null || value.isEmpty) {
            return 'Please enter your name';
           }
           return null;
          },
         ),
         SizedBox(height: 20),

         // Email Field
         TextFormField(
```

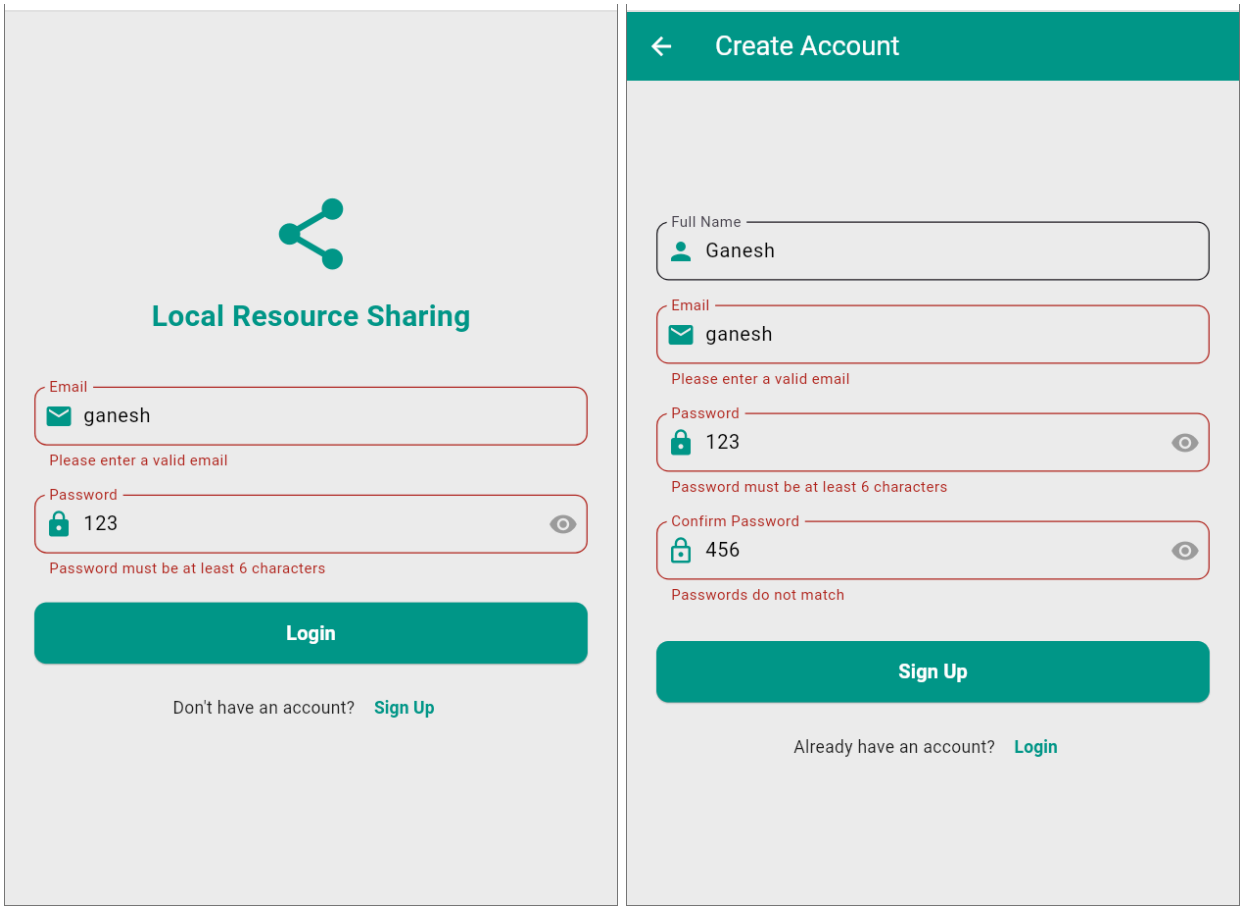
```
    controller: _emailController,
    keyboardType: TextInputType.emailAddress,
    decoration: InputDecoration(
     labelText: 'Email',
     hintText: 'Enter your email',
     prefixIcon: Icon(Icons.email, color: Colors.teal),
     border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
     ),
     focusedBorder: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
      borderSide: BorderSide(color: Colors.teal, width: 2),
     ),
    ),
    validator: (value) {
     if (value == null || value.isEmpty) {
      return 'Please enter your email';
     }
     // Email format validation using regex
     final emailRegex = RegExp(r'^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$');
     if (!emailRegex.hasMatch(value)) {
      return 'Please enter a valid email';
     }
     return null;
    },
   ),
   SizedBox(height: 20),

   // Password Field
   TextFormField(
    controller: _passwordController,
    obscureText: !_isPasswordVisible,
    decoration: InputDecoration(
     labelText: 'Password',
     hintText: 'Create a password',
     prefixIcon: Icon(Icons.lock, color: Colors.teal),
     suffixIcon: IconButton(
      icon: Icon(
       _isPasswordVisible ? Icons.visibility : Icons.visibility_off,
       color: Colors.grey,
      ),
```

```
          onPressed: () {
            setState(() {
              _isPasswordVisible = !_isPasswordVisible;
            });
          },
        ),
        border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(10),
        ),
        focusedBorder: OutlineInputBorder(
          borderRadius: BorderRadius.circular(10),
          borderSide: BorderSide(color: Colors.teal, width: 2),
        ),
      ),
      validator: (value) {
        if (value == null || value.isEmpty) {
          return 'Please enter a password';
        }
        if (value.length < 6) {
          return 'Password must be at least 6 characters';
        }
        // Password strength validation
        bool hasUppercase = value.contains(RegExp(r'[A-Z]'));
        bool hasDigits = value.contains(RegExp(r'[0-9]'));
        bool hasSpecialCharacters = value.contains(RegExp(r'[!@#$%^&*(),.?":{}|<>]'));

        if (!(hasUppercase && hasDigits && hasSpecialCharacters)) {
          return 'Password must contain uppercase, number, and special character';
        }
        return null;
      },
    ),
    SizedBox(height: 20),

    // Confirm Password Field
    TextFormField(
      controller: _confirmPasswordController,
      obscureText: !_isConfirmPasswordVisible,
      decoration: InputDecoration(
        labelText: 'Confirm Password',
        hintText: 'Confirm your password',
```

```
      prefixIcon: Icon(Icons.lock_outline, color: Colors.teal),
      suffixIcon: IconButton(
       icon: Icon(
        _isConfirmPasswordVisible ? Icons.visibility : Icons.visibility_off,
        color: Colors.grey,
       ),
       onPressed: () {
        setState(() {
         _isConfirmPasswordVisible = !_isConfirmPasswordVisible;
        });
       },
      ),
      border: OutlineInputBorder(
       borderRadius: BorderRadius.circular(10),
      ),
      focusedBorder: OutlineInputBorder(
       borderRadius: BorderRadius.circular(10),
       borderSide: BorderSide(color: Colors.teal, width: 2),
      ),
     ),
     validator: (value) {
      if (value == null || value.isEmpty) {
       return 'Please confirm your password';
      }
      if (value != _passwordController.text) {
       return 'Passwords do not match';
      }
      return null;
     },
    ),
    SizedBox(height: 30),

    // Sign Up Button
    SizedBox(
     width: double.infinity,
     height: 50,
     child: ElevatedButton(
      onPressed: () async {
       if (_formKey.currentState!.validate()) {
        // For demo purpose, save user info and login
        final prefs = await SharedPreferences.getInstance();
```

```
            await prefs.setBool('isLoggedIn', true);
            await prefs.setString('userName', _nameController.text);
            await prefs.setString('userEmail', _emailController.text);

            // Show success message
            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(
                content: Text('Account created successfully!'),
                backgroundColor: Colors.green,
              ),
            );

            // Navigate to Home Screen
            Navigator.pushAndRemoveUntil(
              context,
              MaterialPageRoute(builder: (context) => HomeScreen()),
              (route) => false,
            );
          }
        },
        style: ElevatedButton.styleFrom(
          backgroundColor: Colors.teal,
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(10),
          ),
        ),
        child: Text(
          'Sign Up',
          style: TextStyle(
            color: Colors.white,
            fontSize: 16,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
    ),
  ),
  SizedBox(height: 20),

  // Login Link
  Row(
    mainAxisAlignment: MainAxisAlignment.center,
```

```
                children: [
                  Text("Already have an account? "),
                  TextButton(
                    onPressed: () {
                      Navigator.pop(context);
                    },
                    child: Text(
                      'Login',
                      style: TextStyle(
                        color: Colors.teal,
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                  ),
                ],
              ),
            ],
          ),
        ),
      ),
    ),
  );
 }
}
```



Conclusion

In this experiment, we successfully implemented an interactive form using TextField, validation checks, and SnackBar feedback for user inputs. Initially, we faced issues with clearing input fields and displaying validation messages, which we resolved by using TextEditingController and condition checks before submission.

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

Theory:

Navigation, routing, and gestures are fundamental aspects of building interactive mobile applications. They define how users move between different screens (navigation), how specific URLs or paths map to particular screens or functionalities (routing), and how users interact with on-screen elements (gestures).

- Navigation: This involves moving between different screens or parts of an application. It typically involves using widgets like Navigator to push and pop routes (screens) onto a stack.
- Routing: This is the process of mapping a specific URL or path to a particular screen or functionality within an application. Named routes provide a way to navigate to different parts of the app using a string identifier.
- Gestures: These are user interactions like tapping, swiping, dragging, and pinching that can trigger specific actions in an application. Flutter provides GestureDetector and other widgets to recognize and respond to these gestures.

Implemented in our Code

In our Local Resource Sharing Flutter app, we used navigation, routing and gestures in following ways.

- Navigation: We implemented navigation to allow users to move between the login screen, sign-up screen, home screen, item detail screen, and add item screen using the Navigator widget. When a user successfully logs in or signs up, they are navigated to the home screen.
- Routing: We defined named routes in the MaterialApp widget to map route names to specific screens. This makes it easy to navigate to a specific screen by its name. The routes allowed us to move between Login, signup and the home screen.
- Gestures: We used the GestureDetector widget to make the resource cards on the home screen tappable. When a user taps on a resource card, they are navigated to the item detail screen, where they can see more information about the resource.

Code

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

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

class LocalResourceApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   debugShowCheckedModeBanner: false,
   title: 'Local Resource Sharing',
   theme: ThemeData(
    primaryColor: Colors.teal,
    scaffoldBackgroundColor: Colors.grey[200],
    textTheme: TextTheme(
```

```
      bodyLarge: TextStyle(fontFamily: 'Roboto', color: Colors.black),
      bodyMedium: TextStyle(fontFamily: 'Roboto', color: Colors.black87),
    ),

  // New Color Scheme based on the theme
    colorScheme: ColorScheme.fromSwatch().copyWith(
      primary: Colors.teal, // Primary color (teal)
      secondary: Colors.amber, // Secondary color (amber)
      background: Colors.grey[200], // Light gray background
      surface: Colors.white, // White cards
      onBackground: Colors.black87, // Dark gray text
      onSurface: Colors.black87, // Dark gray text
      onPrimary: Colors.white,
      onSecondary: Colors.white,
    ),
    appBarTheme: AppBarTheme(
      backgroundColor: Colors.teal,
      foregroundColor: Colors.white,
    ),
    elevatedButtonTheme: ElevatedButtonThemeData(
      style: ElevatedButton.styleFrom(
        backgroundColor: Colors.teal,
        foregroundColor: Colors.white,
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(10),
        ),
      ),
    ),
    inputDecorationTheme: InputDecorationTheme(
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(10),
      ),
      focusedBorder: OutlineInputBorder(
        borderSide: BorderSide(color: Colors.teal, width: 2),
        borderRadius: BorderRadius.circular(10),
      ),
      labelStyle: TextStyle(color: Colors.black87),
    ),
  ),
  home: AuthChecker(),
  routes: {
    '/home': (context) => HomeScreen(),
    '/login': (context) => LoginScreen(),
    '/signup': (context) => SignupScreen(),
    // '/item_detail': (context) => ItemDetailScreen(), // Route for Item Detail Screen
    '/add_item': (context) => AddItemScreen(), // Route for Add Item Screen
```

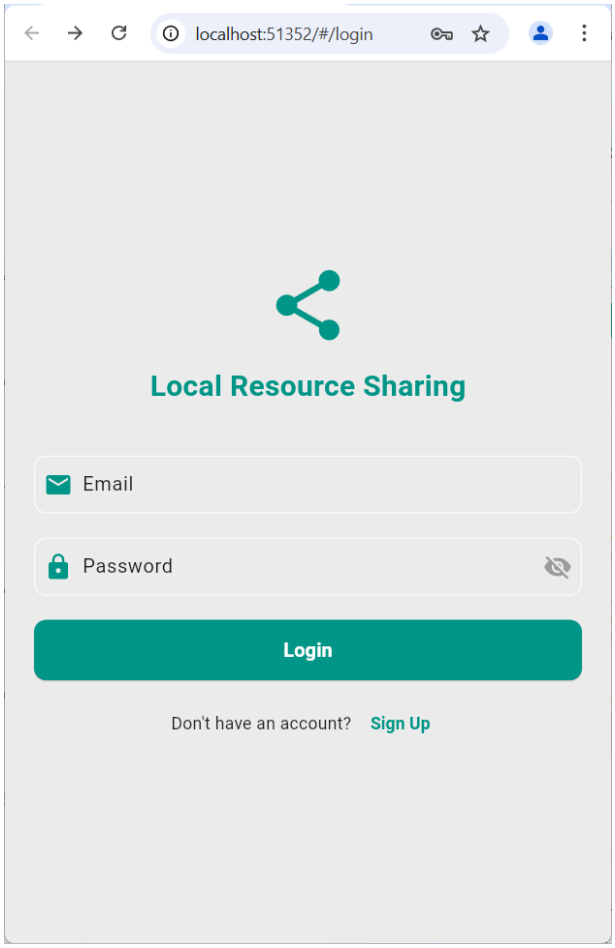
```
    },
   );
  }
}

// Home Screen
class HomeScreen extends StatefulWidget {
  @override
  _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  String userName = '';
  String userEmail = '';
  List<ResourceItem> items = [  // Changed to List<ResourceItem>
    ResourceItem(
      name: 'Drill Machine',
      category: 'Home Appliances',
      image: 'assets/images/drill_machine.png',
      icon: Icons.construction,
      owner: 'Rahul Sharma',
      description: 'Powerful drill for various tasks',
    ),
    ResourceItem(
      name: 'Lawn Mower',
      category: 'Gardening',
      image: 'assets/images/lawn_mower.png',
      icon: Icons.grass,
      owner: 'Priya Patel',
      description: 'Efficient lawn mower for a perfect lawn',
    ),
    ResourceItem(
      name: 'Projector',
      category: 'Electronics',
      image: 'assets/images/projector.png',
      icon: Icons.tv,
      owner: 'Amit Verma',
      description: 'High-resolution projector for home theater',
    ),
    ResourceItem(
      name: 'Electric Kettle',
      category: 'Kitchen Appliances',
      image: 'assets/images/default.png',
      icon: null,
      owner: 'Sneha Reddy',
      description: 'Fast boiling electric kettle for daily use',
```

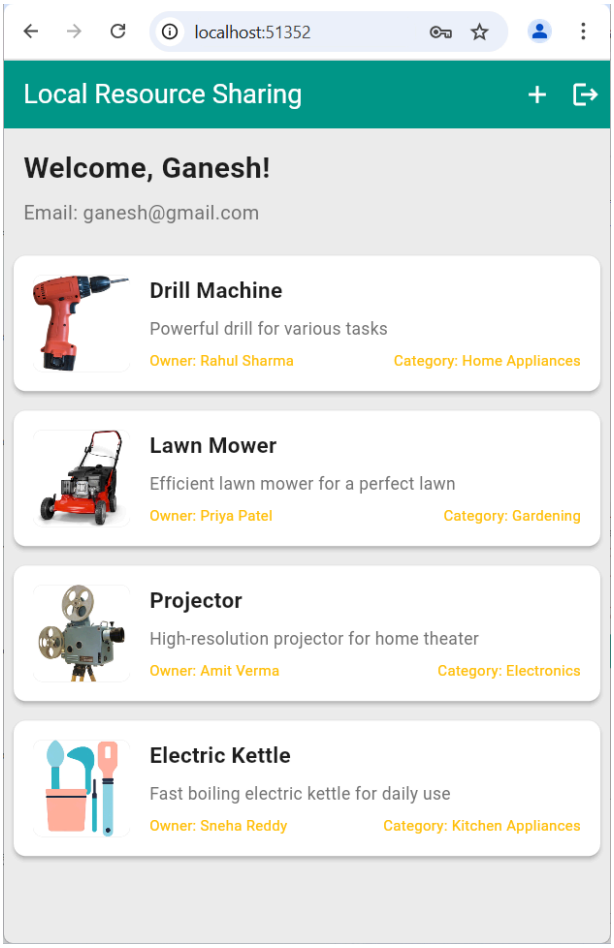
```
  ),
 ];


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

 Future<void> loadUserData() async {
  final prefs = await SharedPreferences.getInstance();
  setState(() {
   userName = prefs.getString('userName') ?? 'User';
   userEmail = prefs.getString('userEmail') ?? 'user@example.com';
  });
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('Local Resource Sharing'),
    actions: [
     IconButton(
      icon: Icon(Icons.add),
      onPressed: () {
       Navigator.pushNamed(context, '/add_item'); // Navigate to Add Item Screen
      },
     ),
     IconButton(
      icon: Icon(Icons.logout),
      onPressed: () async {
       final prefs = await SharedPreferences.getInstance();
       await prefs.setBool('isLoggedIn', false);
       Navigator.pushReplacementNamed(context, '/login');
      },
     ),
    ],
   ),
   body: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
     Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
```

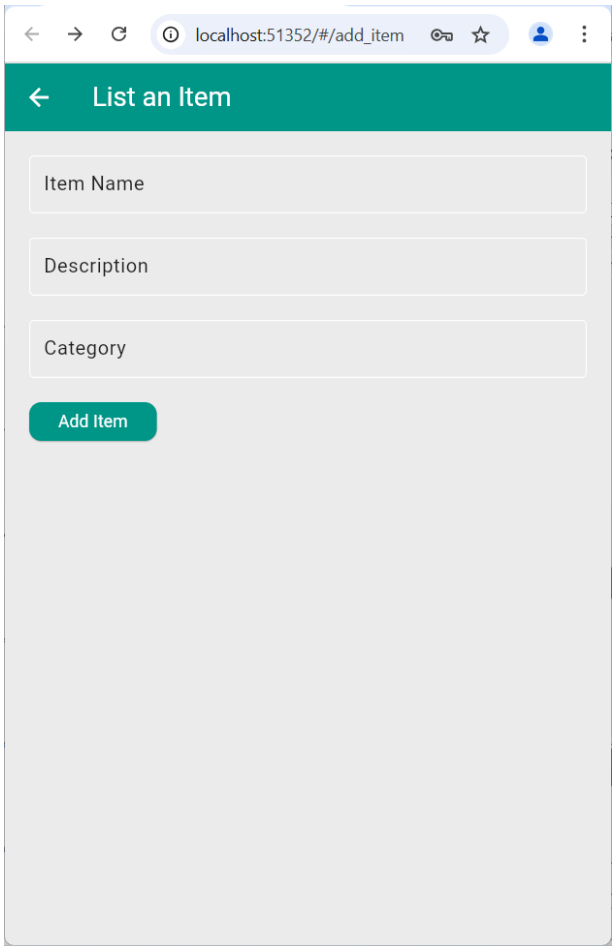
```
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text('Welcome, $userName!',
              style:
                TextStyle(fontSize: 24, fontWeight: FontWeight.bold)),
            SizedBox(height: 8),
            Text('Email: $userEmail',
              style: TextStyle(fontSize: 16, color: Colors.grey[600])),
          ],
        ),
      ),
      Expanded(
        child: ListView.builder(
          itemCount: items.length,
          itemBuilder: (context, index) {
            return ResourceCard(item: items[index]);
          },
        ),
      ),
    ],
  ),
 );
 }
}

// Add Item Screen
class AddItemScreen extends StatefulWidget {
 @override
 _AddItemScreenState createState() => _AddItemScreenState();
}

class _AddItemScreenState extends State<AddItemScreen> {
 final _formKey = GlobalKey<FormState>();
 final TextEditingController _nameController = TextEditingController();
 final TextEditingController _descriptionController = TextEditingController();
 final TextEditingController _categoryController = TextEditingController();

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('List an Item'),
   ),
   body: Padding(
    padding: const EdgeInsets.all(20.0),
    child: Form(
```

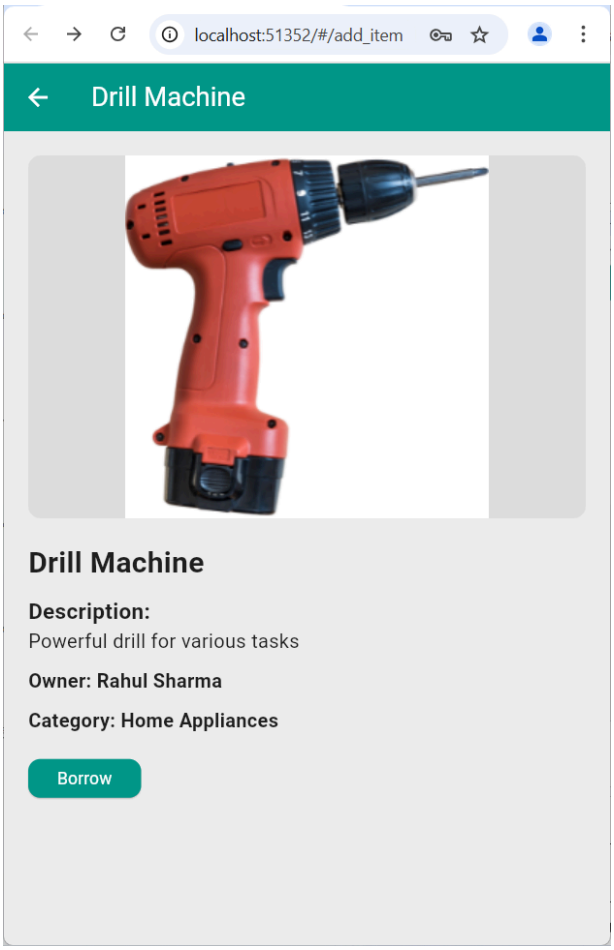
```
      key: _formKey,
      child: Column(
       crossAxisAlignment: CrossAxisAlignment.start,
       children: [
        TextFormField(
         controller: _nameController,
         decoration: InputDecoration(
          labelText: 'Item Name',
          border: OutlineInputBorder(),
         ),
         validator: (value) {
          if (value == null || value.isEmpty) {
           return 'Please enter item name';
          }
          return null;
         },
        ),
        SizedBox(height: 20),
        TextFormField(
         controller: _descriptionController,
         decoration: InputDecoration(
          labelText: 'Description',
          border: OutlineInputBorder(),
         ),
         validator: (value) {
          if (value == null || value.isEmpty) {
           return 'Please enter item description';
          }
          return null;
         },
        ),
        SizedBox(height: 20),
        TextFormField(
         controller: _categoryController,
         decoration: InputDecoration(
          labelText: 'Category',
          border: OutlineInputBorder(),
         ),
         validator: (value) {
          if (value == null || value.isEmpty) {
           return 'Please enter item category';
          }
          return null;
         },
        ),
        SizedBox(height: 20),
```

```
          ElevatedButton(
            onPressed: () {
              if (_formKey.currentState!.validate()) {
                // Implement adding item logic here
                String itemName = _nameController.text;
                String itemDescription = _descriptionController.text;
                String itemCategory = _categoryController.text;

                // For now, just show a snackbar
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(
                    content: Text(
                        'Adding $itemName to category $itemCategory'),
                  ),
                );

                // You can also navigate back to the HomeScreen or update the item list
                Navigator.pop(context);
              }
            },
            child: Text('Add Item'),
          ),
        ],
      ),
    ),
  ),
 );
 }
}
class ResourceItem {
 final String name;
 final String category;
 final String image;
 final String owner;  // Added owner
 final IconData? icon;
 final String description;

 ResourceItem({
  required this.name,
  required this.category,
  required this.image,
  required this.icon,
  required this.owner,
  required this.description,
 });
}
```


localhost:51352/#/login
Local Resource Sharing
Email
Password
Login
Don't have an account? Sign Up


localhost:51352
Local Resource Sharing
Welcome, Ganesh!
Email: ganesh@gmail.com
Drill Machine
Powerful drill for various tasks
Owner: Rahul Sharma
Category: Home Appliances
Lawn Mower
Efficient lawn mower for a perfect lawn
Owner: Priya Patel
Category: Gardening
Projector
High-resolution projector for home theater
Owner: Amit Verma
Category: Electronics
Electric Kettle
Fast boiling electric kettle for daily use
Owner: Sneha Reddy
Category: Kitchen Appliances


localhost:51352/#/add_item
List an Item
Item Name
Description
Category
Add Item


localhost:51352/#/add_item
Drill Machine
Drill Machine
Description:
Powerful drill for various tasks
Owner: Rahul Sharma
Category: Home Appliances
Borrow

Conclusion

In this experiment, we successfully implemented an interactive form using TextField, validation checks, and SnackBar feedback for user inputs. Initially, we faced issues with clearing input fields and displaying validation messages, which we resolved by using TextEditingController and condition checks before submission.

Aim: To Connect Flutter UI with fireBase database.

Theory:

Flutter is an open-source UI toolkit that allows developers to build cross-platform applications using a single codebase. Firebase, a cloud-based Backend-as-a-Service (BaaS) platform by Google, provides authentication, real-time databases, cloud storage, and other backend functionalities to support mobile and web applications. This experiment integrates Firebase Authentication with Flutter for user management.

Implemented in our Code

Firebase Setup: Configured Firebase project, enabled Authentication, and added Firebase dependencies.

Authentication (auth_service.dart):

- Signup: Registers users and sends email verification.
- Login: Allows access only after email verification.
- Password Reset: Sends reset email.
- Google Sign-In: Enables authentication via Google.
- Logout: Signs out users.

UI Screens:

- Signup & Login: User-friendly forms with validation.
- Reset Password: Allows password recovery.
- Auth Checker: Redirects users based on login status.

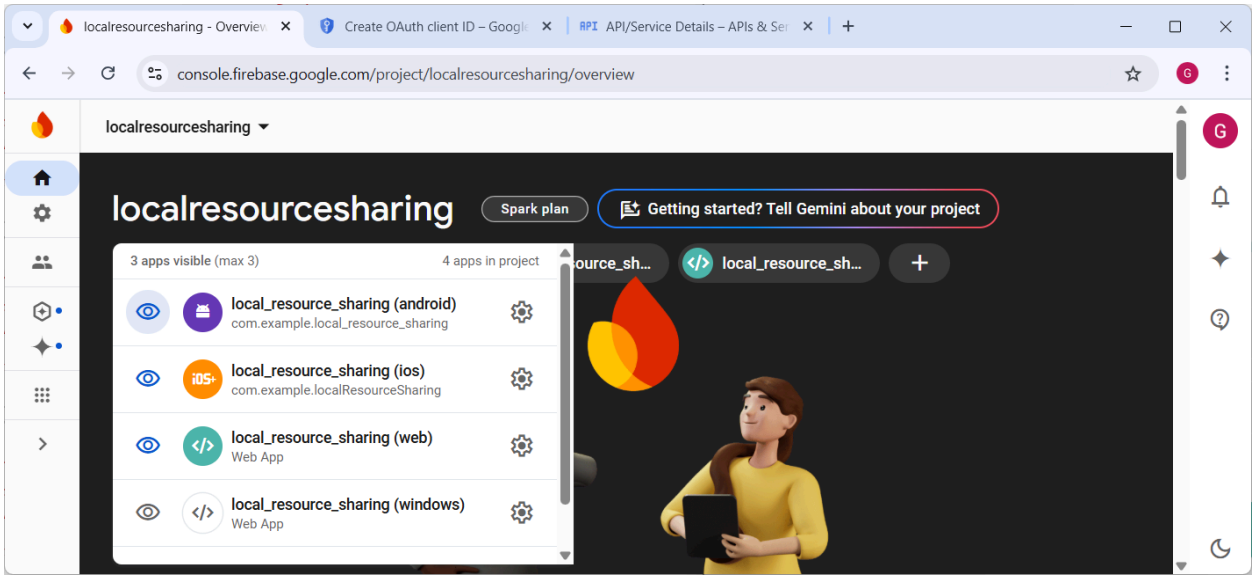
Code

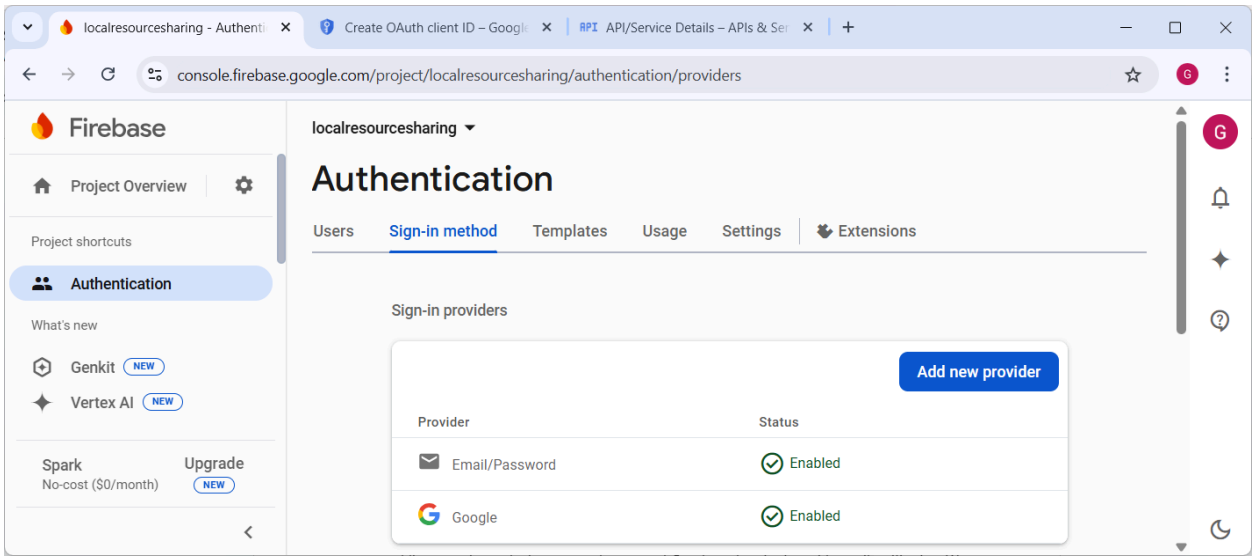
```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:google_sign_in/google_sign_in.dart';

/// **AuthService Class**
/// Handles authentication functionalities using Firebase Authentication.
class AuthService {
 final FirebaseAuth _auth = FirebaseAuth.instance;
 final GoogleSignIn _googleSignIn = GoogleSignIn(
  clientId: "45384462772-ia06jsdakisp6vhuf6pa3sf1n3kbv0cr.apps.googleusercontent.com", // Add your Web
Client ID here
 );

 /// **Checks if a user is logged in**
 bool isUserLoggedIn() {
  return _auth.currentUser != null && _auth.currentUser!.emailVerified;
 }

 /// **Gets the current logged-in user**
```

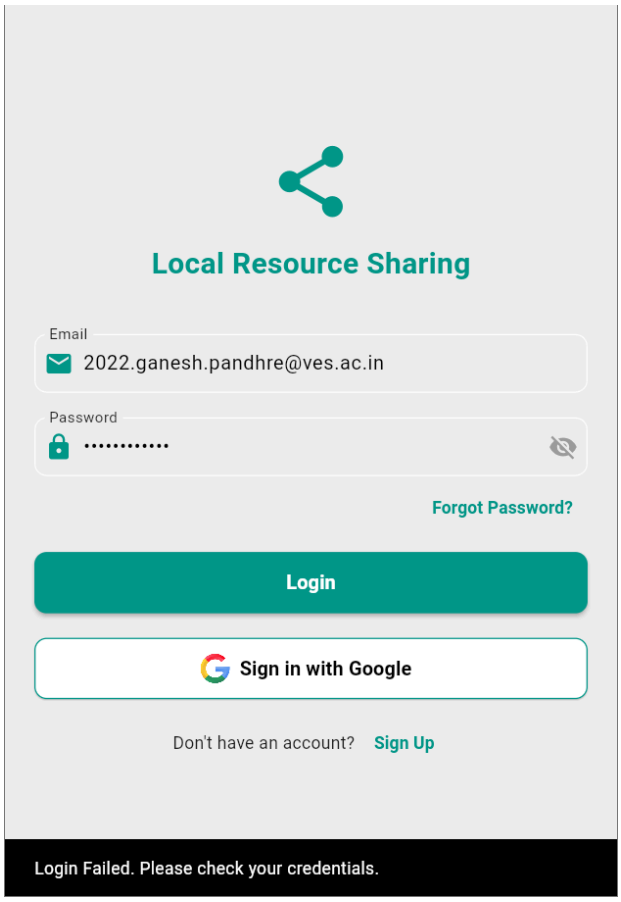
```
 User? getCurrentUser() {
  return _auth.currentUser;
 }

 /// **Signs up a user and sends an email verification**
 Future<bool> signUp(String email, String password) async {
  try {
   UserCredential userCredential = await _auth.createUserWithEmailAndPassword(
    email: email,
    password: password,
   );

   // Send email verification
   await userCredential.user!.sendEmailVerification();
   return true;
  } catch (e) {
   print("Sign Up Error: $e");
   return false;
  }
 }

 /// **Logs in a user only if email is verified**
 Future<User?> signIn(String email, String password) async {
  try {
   UserCredential userCredential = await _auth.signInWithEmailAndPassword(
    email: email,
    password: password,
   );

   if (userCredential.user!.emailVerified) {
    return userCredential.user;
   } else {
    await userCredential.user!.sendEmailVerification();
    print("Please verify your email first.");
    return null;
   }
  } catch (e) {
   print("Login Error: $e");
   return null;
  }
 }
```

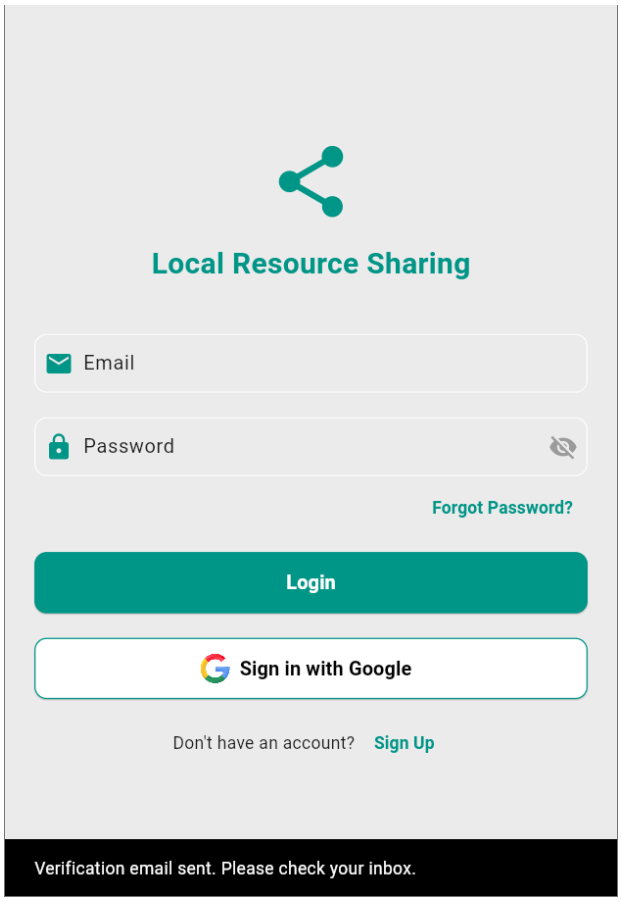
```
/// **Logs out the user (Google & Email)**
Future<void> signOut() async {
  await _auth.signOut();
  await _googleSignIn.signOut();
}

/// **Reset Password via Email**
Future<bool> resetPassword(String email) async {
  try {
    await _auth.sendPasswordResetEmail(email: email);
    return true; // Email sent successfully
  } catch (e) {
    print("Password Reset Error: $e");
    return false; // Failed to send email
  }
}

/// **Google Sign-In Authentication**
Future<User?> signInWithGoogle() async {
  try {
    final GoogleSignInAccount? googleUser = await _googleSignIn.signIn();
    if (googleUser == null) return null; // User canceled

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

    UserCredential userCredential = await _auth.signInWithCredential(credential);
    if (userCredential.user!.emailVerified) {
      return userCredential.user;
    } else {
      print("Please verify your email first.");
      return null;
    }
  } catch (e) {
    print("Google Sign-In Error: $e");
    return null;
  }
```

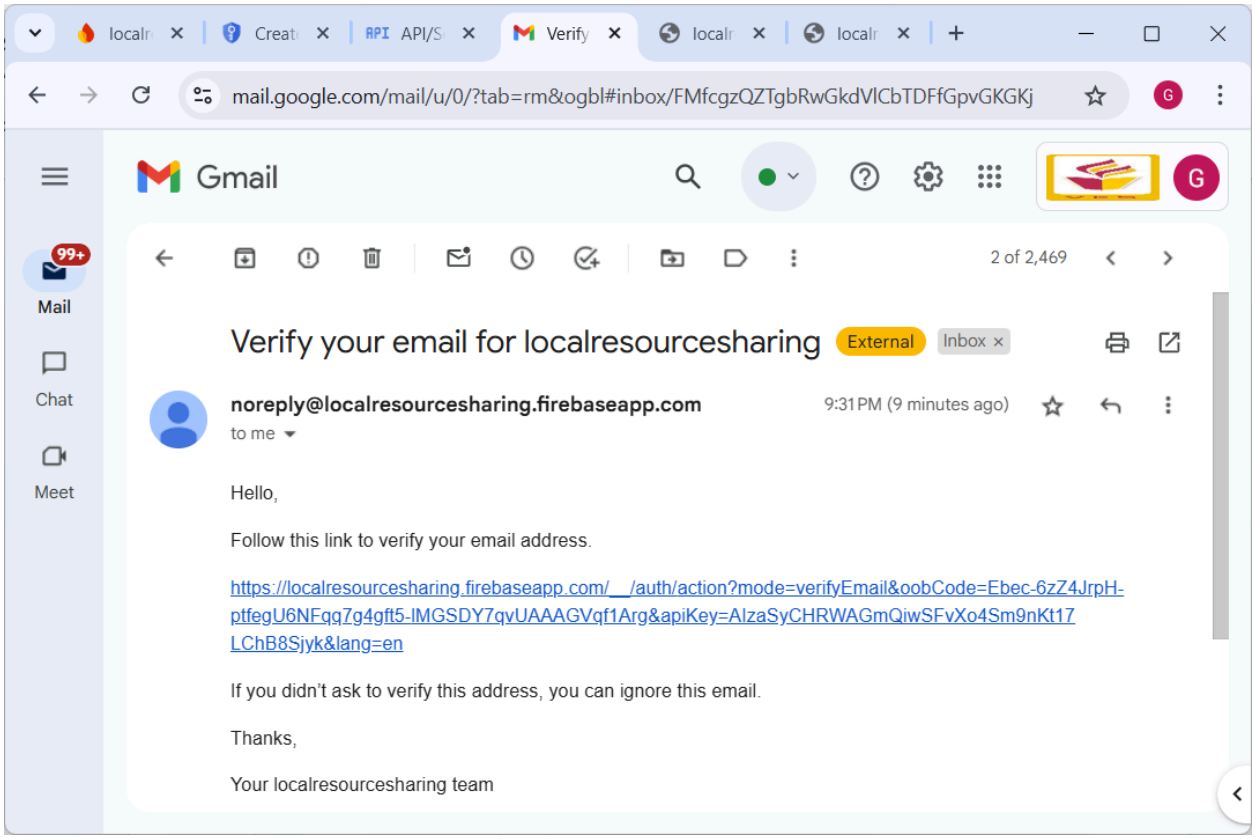
```
  }
}
```

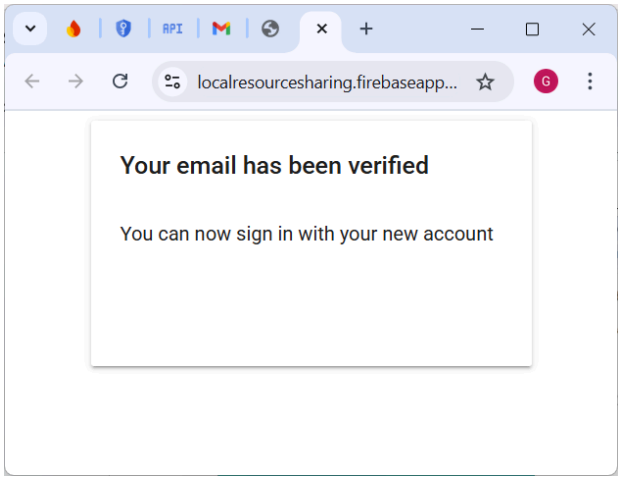

localresourcesharing - Overview
Create OAuth client ID – Google
API/Service Details – APIs & Ser
console.firebase.google.com/project/localresourcesharing/overview
localresourcesharing
localresourcesharing
Spark plan
Getting started? Tell Gemini about your project
3 apps visible (max 3)
4 apps in project
local_resource_sh...
local_resource_sharing (android)
com.example.local_resource_sharing
local_resource_sharing (ios)
com.example.localResourceSharing
local_resource_sharing (web)
Web App
local_resource_sharing (windows)
Web App

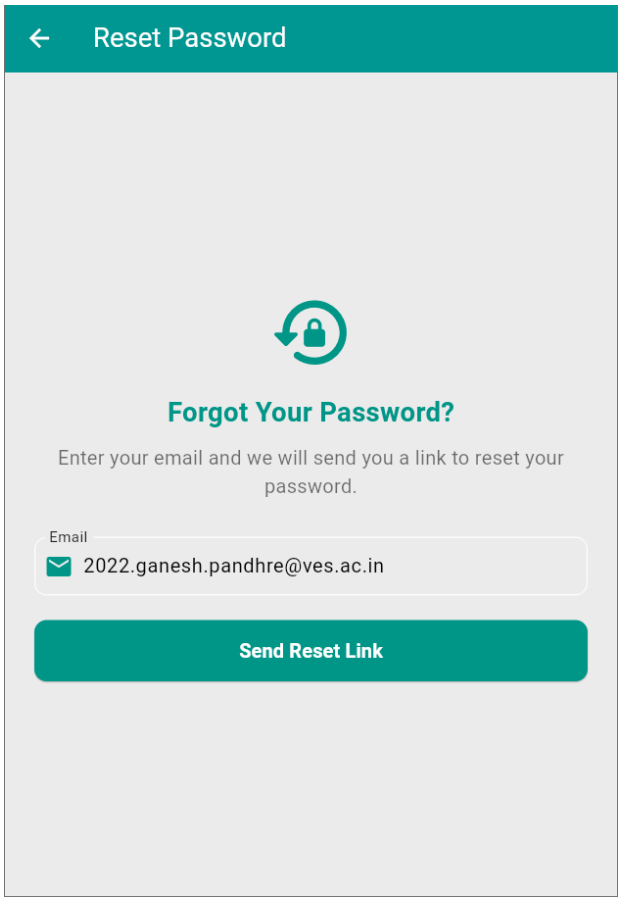

localresourcesharing - Authenti
Create OAuth client ID – Google
API/Service Details – APIs & Ser
console.firebase.google.com/project/localresourcesharing/authentication/providers
Firebase
Project Overview
Project shortcuts
Authentication
What's new
Genkit NEW
Vertex AI NEW
Spark
No-cost ($0/month)
Upgrade
NEW
localresourcesharing
Authentication
Users
Sign-in method
Templates
Usage
Settings
Extensions
Sign-in providers
Add new provider
Provider
Status
Email/Password
Enabled
Google
Enabled

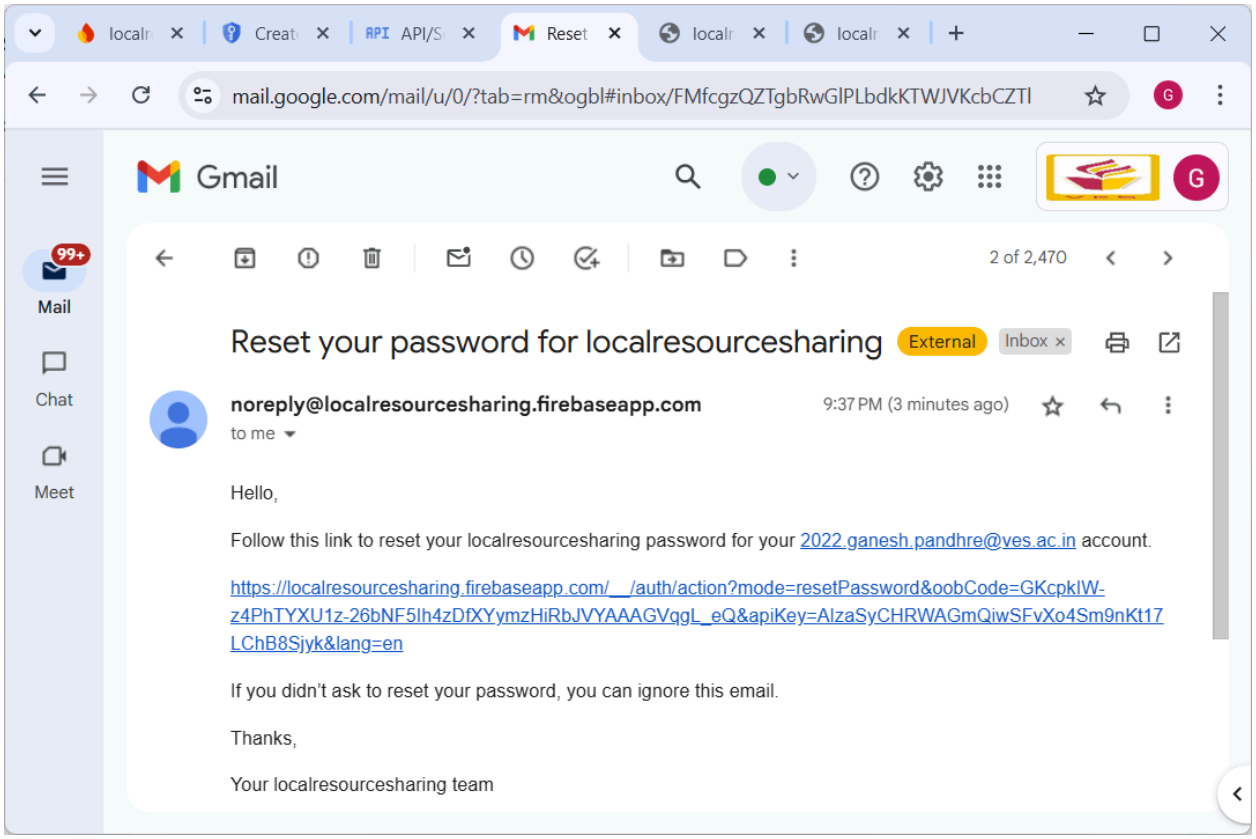

Local Resource Sharing
Email
2022.ganesh.pandhre@ves.ac.in
Password
Forgot Password?
Login
Sign in with Google
Don't have an account? Sign Up
Login Failed. Please check your credentials.

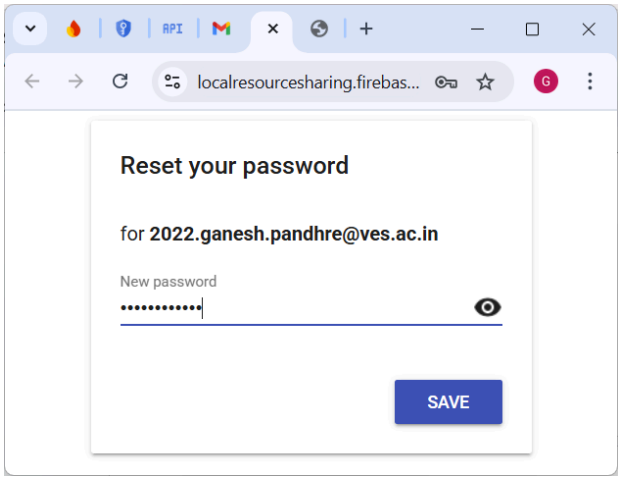

Local Resource Sharing
Email
Password
Forgot Password?
Login
Sign in with Google
Don't have an account? Sign Up
Verification email sent. Please check your inbox.

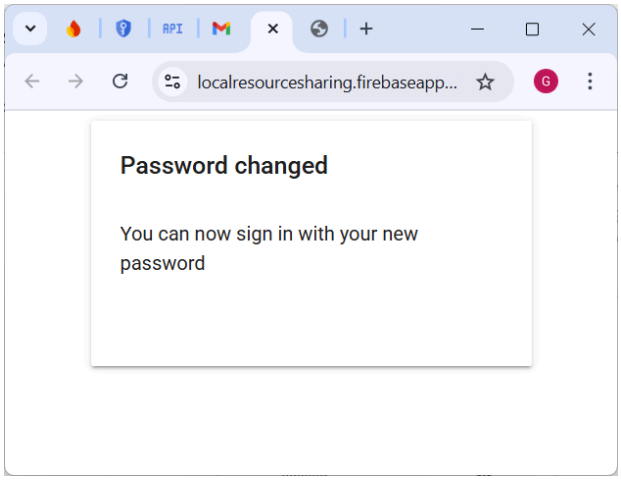

mail.google.com/mail/u/0/?tab=rm&ogbl#inbox/FMfcgzQZTgbRwGkdVlCbTDFfGpvGKGKj
Gmail
2 of 2,469
Verify your email for localresourcesharing
External
Inbox ×
noreply@localresourcesharing.firebaseapp.com
to me
9:31 PM (9 minutes ago)
Hello,
Follow this link to verify your email address.
https://localresourcesharing.firebaseapp.com/__/auth/action?mode=verifyEmail&oobCode=Ebec-6zZ4JrpH-ptfegU6NFqq7g4gft5-lMGSDY7qvUAAAGVqf1Arg&apiKey=AIzaSyCHRWAGmQiwSFvXo4Sm9nKt17LChB8Sjyk&lang=en
If you didn't ask to verify this address, you can ignore this email.
Thanks,
Your localresourcesharing team
Mail
Chat
Meet

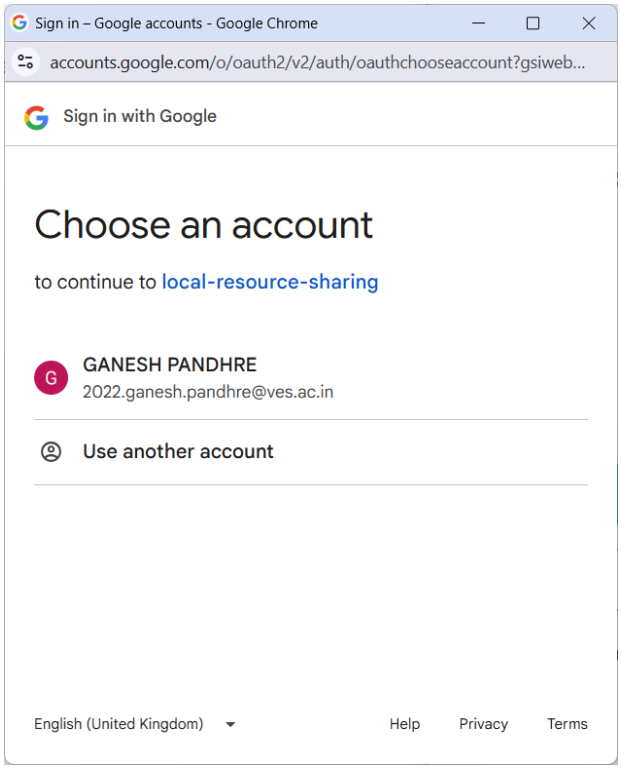

localresourcesharing.firebaseapp...
Your email has been verified
You can now sign in with your new account

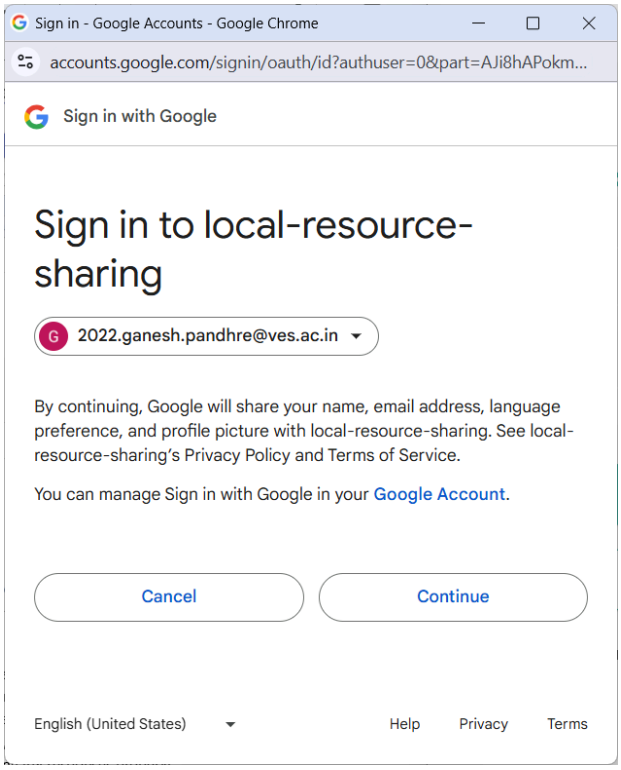

Reset Password
Forgot Your Password?
Enter your email and we will send you a link to reset your password.
Email
2022.ganesh.pandhre@ves.ac.in
Send Reset Link

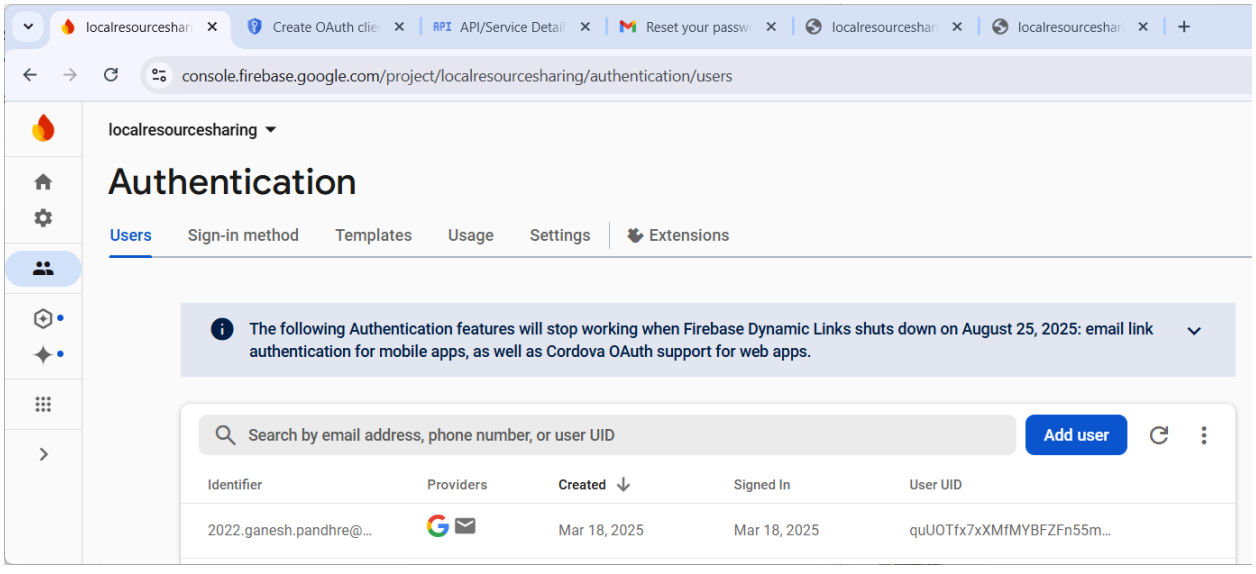

mail.google.com/mail/u/0/?tab=rm&ogbl#inbox/FMfcgzQZTgbRwGlPLbdkKTWJVKcbCZTl
Gmail
Mail
Chat
Meet
2 of 2,470
Reset your password for localresourcesharing
External
Inbox ×
noreply@localresourcesharing.firebaseapp.com
to me
9:37 PM (3 minutes ago)
Hello,
Follow this link to reset your localresourcesharing password for your 2022.ganesh.pandhre@ves.ac.in account.
https://localresourcesharing.firebaseapp.com/__/auth/action?mode=resetPassword&oobCode=GKcpkIW-z4PhTYXU1z-26bNF5Ih4zDfXYymzHiRbJVYAAAGVqgL_eQ&apiKey=AIzaSyCHRWAGmQiwSFvXo4Sm9nKt17LChB8Sjyk&lang=en
If you didn't ask to reset your password, you can ignore this email.
Thanks,
Your localresourcesharing team


localresourcesharing.firebas...
Reset your password
for 2022.ganesh.pandhre@ves.ac.in
New password
SAVE


localresourcesharing.firebaseapp...
Password changed
You can now sign in with your new password


Sign in – Google accounts - Google Chrome
accounts.google.com/o/oauth2/v2/auth/oauthchooseaccount?gsiweb...
Sign in with Google
Choose an account
to continue to local-resource-sharing
GANESH PANDHRE
2022.ganesh.pandhre@ves.ac.in
Use another account
English (United Kingdom)
Help
Privacy
Terms


Sign in - Google Accounts - Google Chrome
accounts.google.com/signin/oauth/id?authuser=0&part=AJi8hAPokm...
Sign in with Google
Sign in to local-resource-sharing
2022.ganesh.pandhre@ves.ac.in
By continuing, Google will share your name, email address, language preference, and profile picture with local-resource-sharing. See local-resource-sharing's Privacy Policy and Terms of Service.
You can manage Sign in with Google in your Google Account.
Cancel
Continue
English (United States)
Help
Privacy
Terms


console.firebase.google.com/project/localresourcesharing/authentication/users
localresourcesharing
Authentication
Users
Sign-in method
Templates
Usage
Settings
Extensions
The following Authentication features will stop working when Firebase Dynamic Links shuts down on August 25, 2025: email link authentication for mobile apps, as well as Cordova OAuth support for web apps.
Search by email address, phone number, or user UID
Add user
Identifier
Providers
Created
Signed In
User UID
2022.ganesh.pandhre@...
Mar 18, 2025
Mar 18, 2025
quUOTfx7xXMfMYBFZFn55m...

Conclusion

In this experiment, we successfully implemented Firebase Authentication in the Local Resource Sharing app, enabling user signup, login, password reset, and Google authentication with email verification. We encountered issues like Firebase email verification delays, Google Sign-In Client ID errors, and incorrect return types in authentication functions, which we resolved by correctly configuring Firebase settings, updating authentication logic, and handling API responses properly.

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

Theory

In Progressive Web Apps (PWA), the web app manifest is a JSON file that provides important metadata about the web application. This metadata tells the browser how the app should behave when installed on a user’s device, such as its name, icons, color themes, and launch behavior.
The main goal of this experiment is to enable the “Add to Home Screen” feature, which allows users to install the PWA directly on their device like a native app.

What We Implemented:
We created a manifest.json file and linked it in the <head> section of our index.html using:

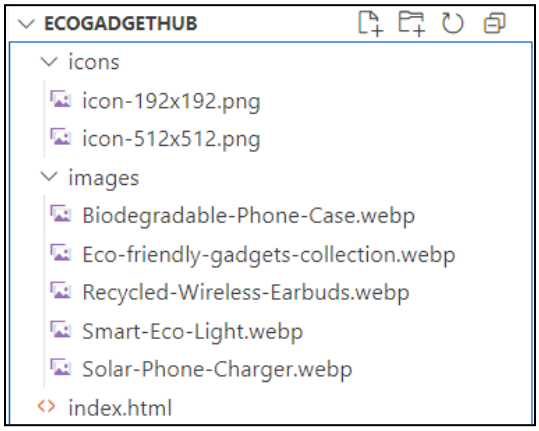
```
<link rel="manifest" href="manifest.json">
```

In the manifest file:

- "name" and "short_name" define how the app is named on the home screen and splash screen.
- "start_url" specifies the entry point when the app is opened.
- "display": "standalone" ensures the app opens without browser UI, giving it an app-like feel.
- "theme_color" and "background_color" customize the app's appearance on launch.
- "orientation": "portrait" locks the screen orientation.
- "scope" limits what URLs the PWA has control over.
- "icons" define images used for the home screen and splash screen when the app is installed.

By including this manifest and serving the site over HTTPS, the browser detects that it’s a PWA and automatically enables the Add to Home Screen option.

Folder Structure



Code

index.html

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

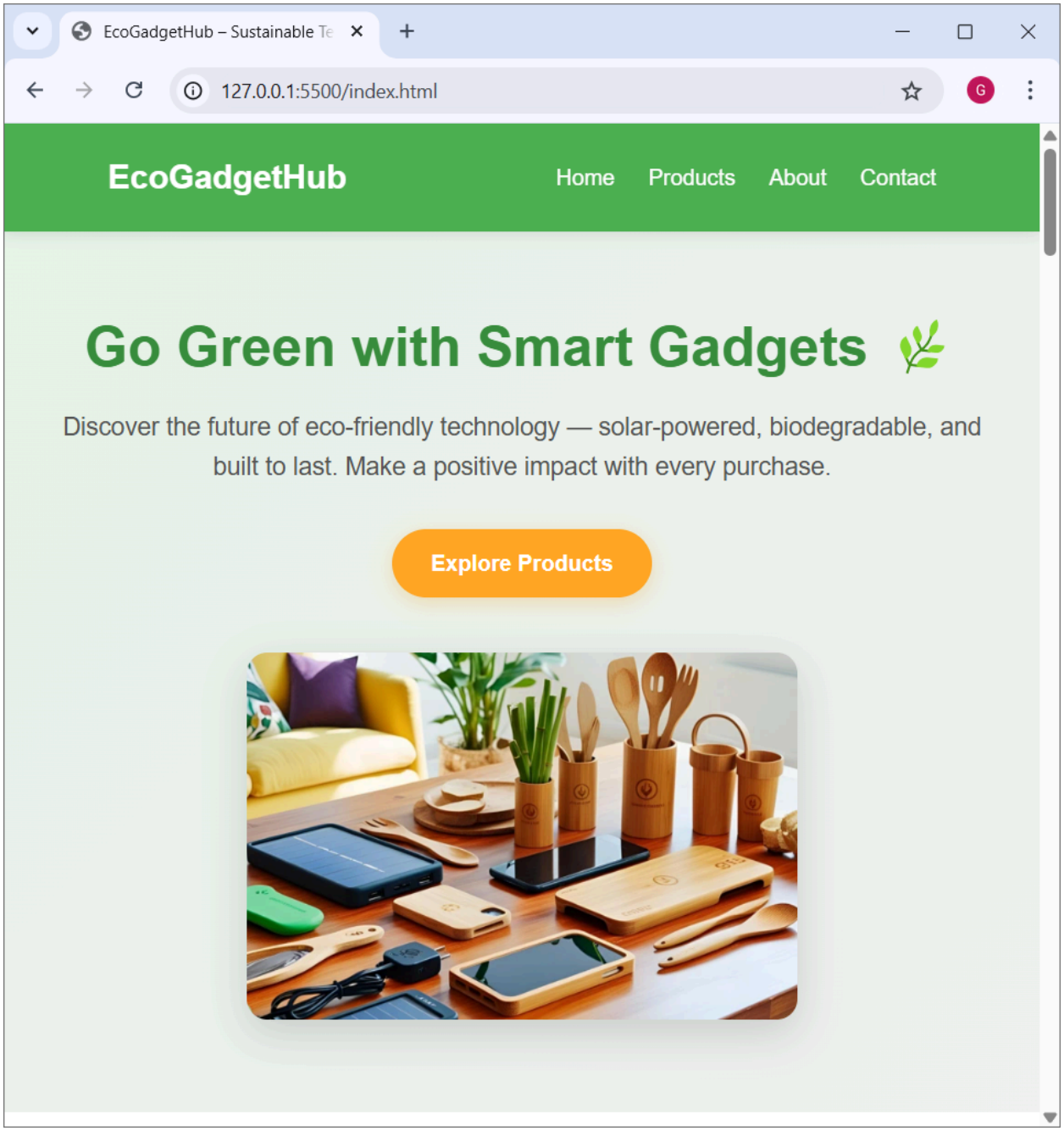
```
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>EcoGadgetHub – Sustainable Tech Gadgets</title>
  <link rel="manifest" href="manifest.json">
  </style>
 </head>
 <body>
  <script>
   if ("serviceWorker" in navigator) {
    navigator.serviceWorker
     .register("serviceworker.js")
     .then(() => console.log("Service Worker Registered"))
     .catch((error) =>
      console.log("Service Worker Registration Failed", error)
     );
   }
  </script>
 </body>
</html>
```

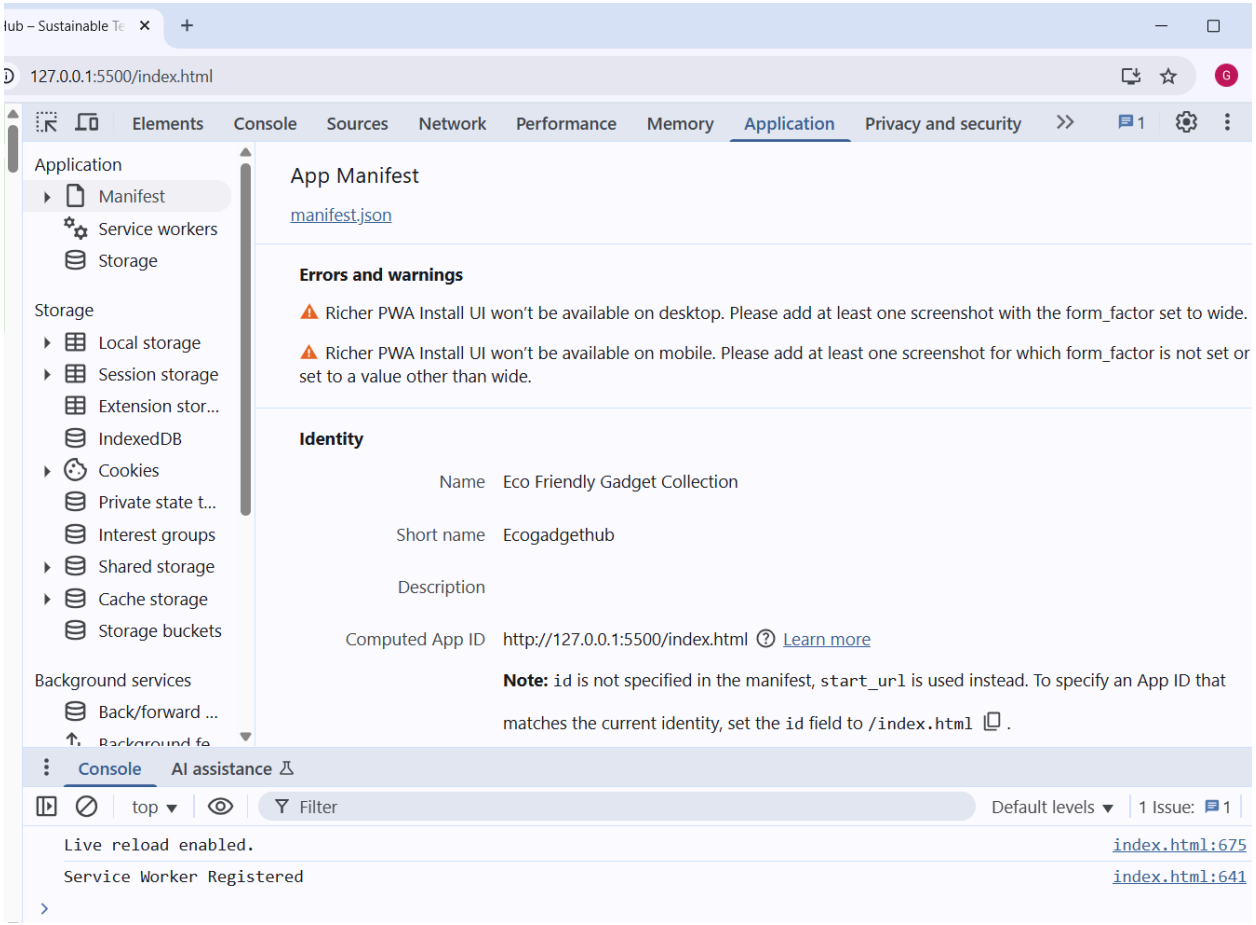
manifest.json

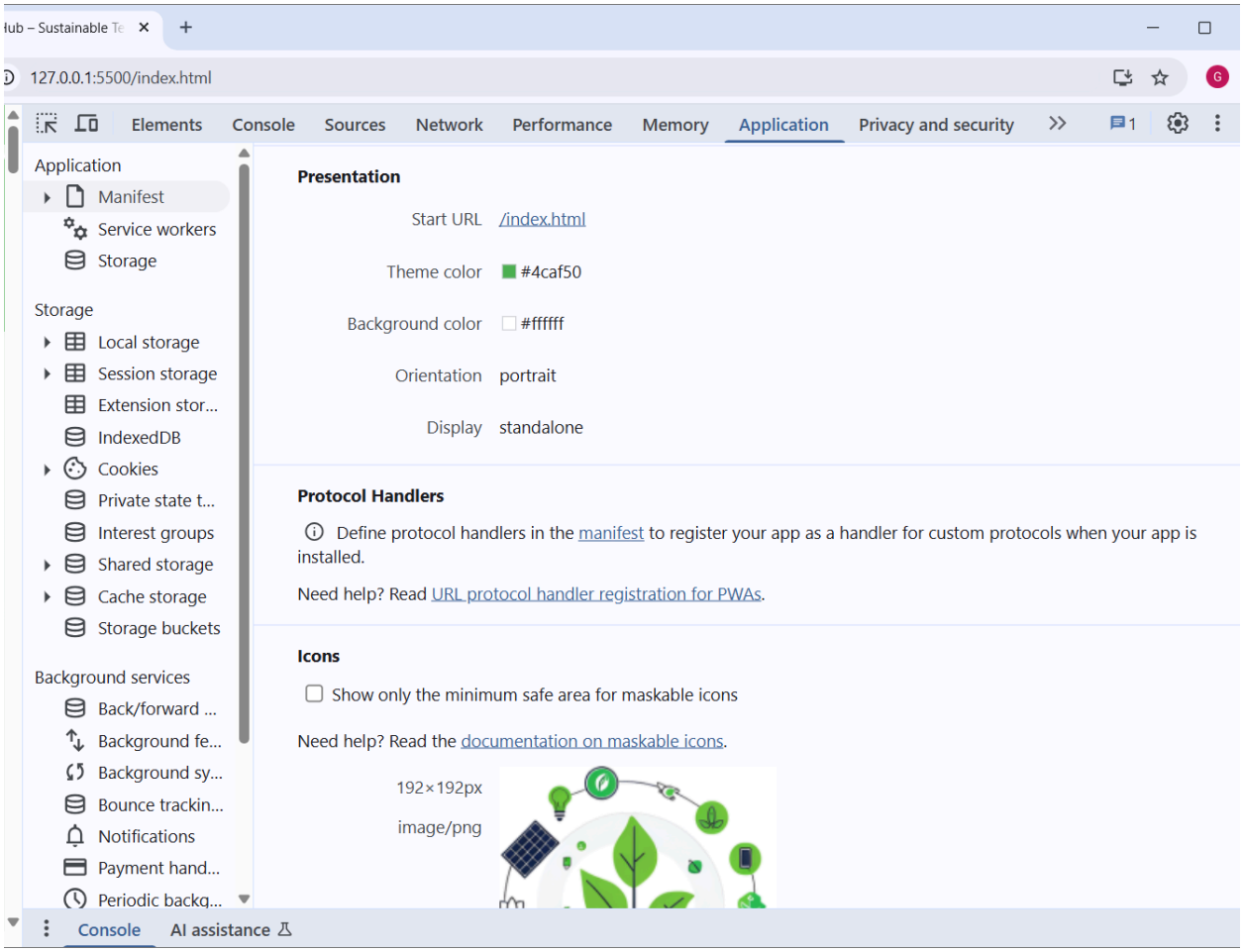
```
{
  "name": "Eco Friendly Gadget Collection",
  "short_name": "Ecogadgethub",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#4caf50",
  "orientation": "portrait",
  "scope": "/",
  "icons": [
   {
    "src": "icons/icon-192x192.png",
    "sizes": "192x192",
    "type": "image/png"
   },
   {
    "src": "icons/icon-512x512.png",
    "sizes": "512x512",
```

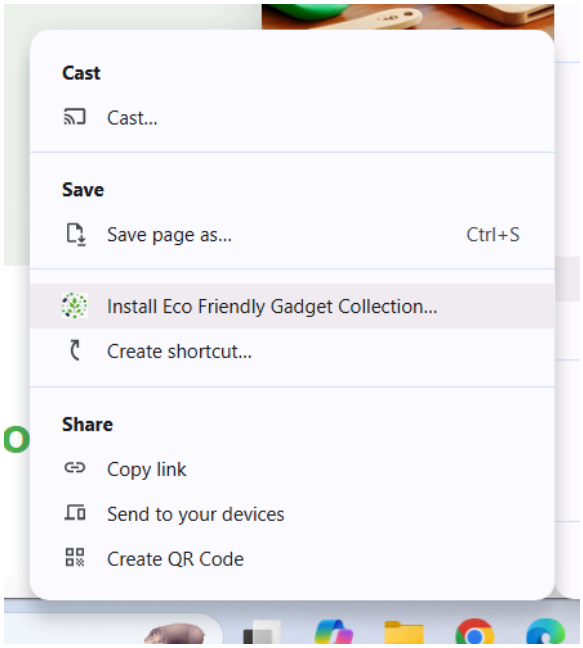
```
    "type": "image/png"
   }
  ]
}
```

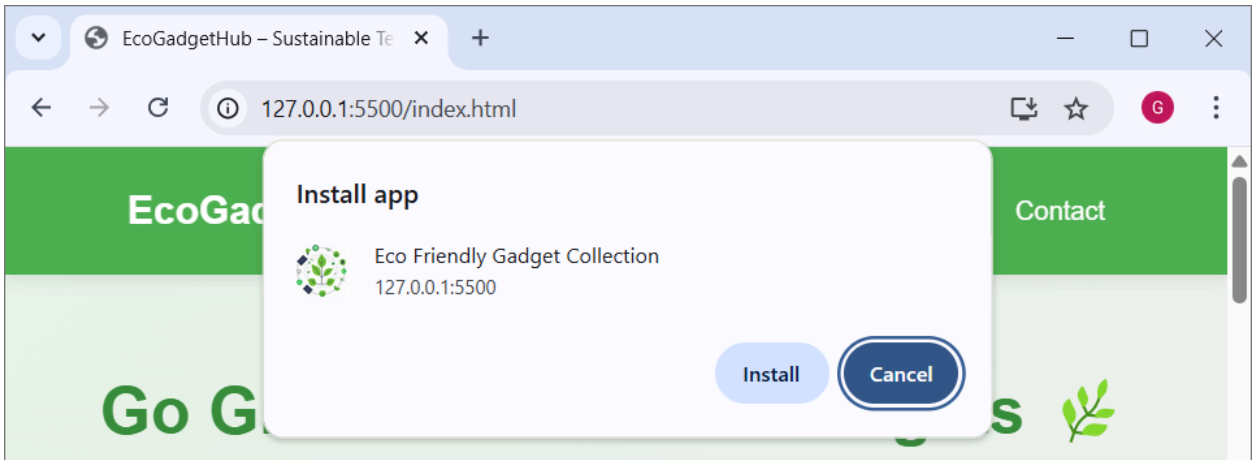
Screenshot

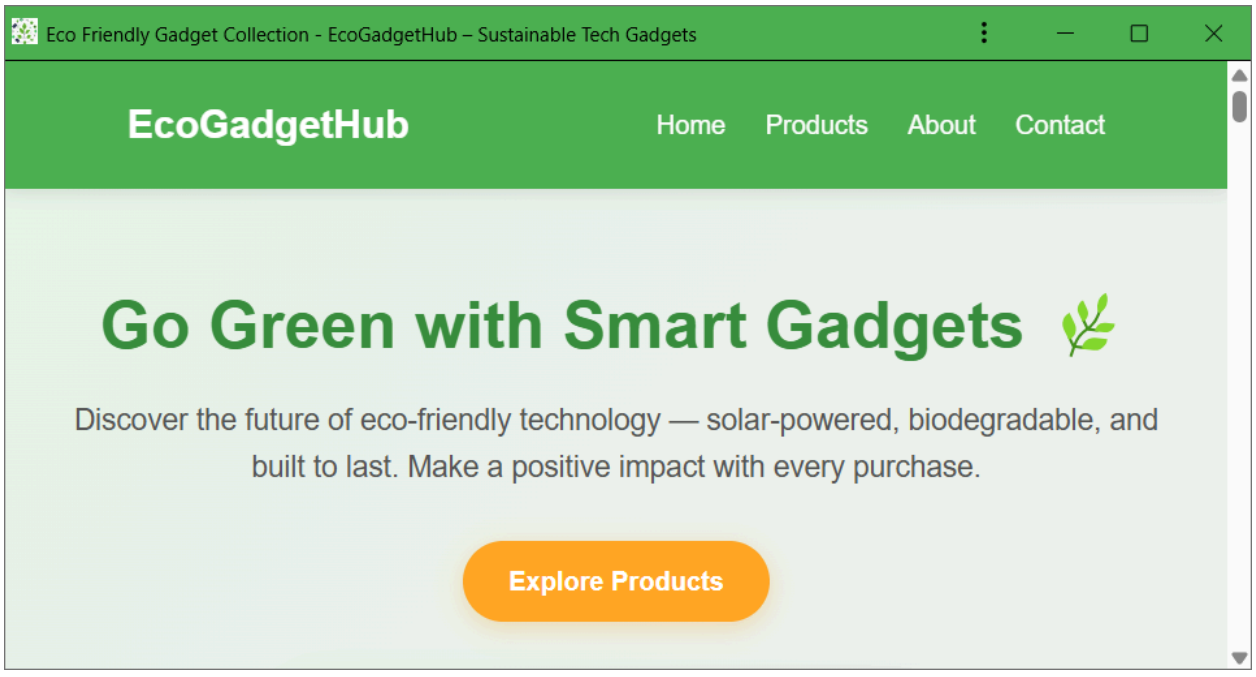


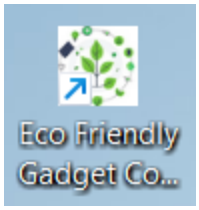

127.0.0.1:5500/index.html
Elements Console Sources Network Performance Memory Application Privacy and security
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension stor...
IndexedDB
Cookies
Private state t...
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
Back/forward ...
App Manifest
manifest.json
Errors and warnings
Richer PWA Install UI won't be available on desktop. Please add at least one screenshot with the form_factor set to wide.
Richer PWA Install UI won't be available on mobile. Please add at least one screenshot for which form_factor is not set or set to a value other than wide.
Identity
Name Eco Friendly Gadget Collection
Short name Ecogadgethub
Description
Computed App ID http://127.0.0.1:5500/index.html Learn more
Note: id is not specified in the manifest, start_url is used instead. To specify an App ID that matches the current identity, set the id field to /index.html .
Console AI assistance
top Filter Default levels 1 Issue: 1
Live reload enabled. index.html:675
Service Worker Registered index.html:641


127.0.0.1:5500/index.html
Elements Console Sources Network Performance Memory Application Privacy and security
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension stor...
IndexedDB
Cookies
Private state t...
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
Back/forward ...
Background fe...
Background sy...
Bounce trackin...
Notifications
Payment hand...
Periodic backg...
Presentation
Start URL /index.html
Theme color #4caf50
Background color #ffffff
Orientation portrait
Display standalone
Protocol Handlers
Define protocol handlers in the manifest to register your app as a handler for custom protocols when your app is installed.
Need help? Read URL protocol handler registration for PWAs.
Icons
Show only the minimum safe area for maskable icons
Need help? Read the documentation on maskable icons.
192×192px
image/png
Console AI assistance


Cast
Cast...
Save
Save page as... Ctrl+S
Install Eco Friendly Gadget Collection...
Create shortcut...
Share
Copy link
Send to your devices
Create QR Code


EcoGadgetHub – Sustainable Te
127.0.0.1:5500/index.html
Install app
Eco Friendly Gadget Collection
127.0.0.1:5500
Install
Cancel
Contact


Eco Friendly Gadget Collection - EcoGadgetHub – Sustainable Tech Gadgets
EcoGadgetHub
Home
Products
About
Contact
Go Green with Smart Gadgets
Discover the future of eco-friendly technology — solar-powered, biodegradable, and built to last. Make a positive impact with every purchase.
Explore Products



Conclusion

In this experiment, we successfully implemented the Web App Manifest for EcoGadgetHub, enabling the "Add to Home Screen" feature with proper icons and theme settings. Initially, we faced an issue where the manifest was not loading due to an incorrect file path, which we resolved by placing it in the root directory and correctly linking it in the HTML.

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

A Service Worker is a script that runs in the background of a web application and helps improve performance and offline functionality. It allows features like caching files, background syncing, and push notifications. For PWAs (Progressive Web Apps), service workers are essential for creating a reliable and fast experience, even without internet.

In this experiment, we focused on coding and registering a new service worker for our E-commerce PWA "EcoGadgetHub". Here's what we did:

1. Registered the Service Worker in index.html:
    - We used JavaScript to check if the browser supports service workers.
    - If supported, the service worker (serviceworker.js) is registered when the page loads.

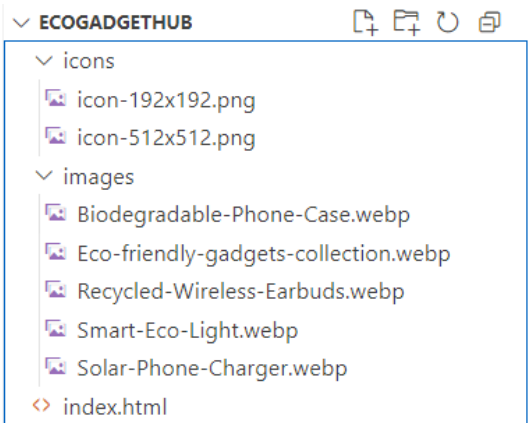
2. Installation Phase:
    - In the install event, we created a cache named 'eco-gadget-hub-v1'.
    - We preloaded and stored important files like HTML, images, and icons in this cache so they can be accessed offline.

3. Activation Phase:
    - In the activate event, we checked for older cache versions.
    - If any old cache is found, it is deleted to keep the storage clean and updated.

4. Fetch Handling:
    - For every request, the service worker checks if the file is in the cache.
    - If it is, the cached file is returned (faster and works offline).
    - If not, it fetches the file from the internet.

By doing this, our E-commerce PWA now works smoothly even without internet access, and loads faster due to cached resources.

Folder Structure



index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>EcoGadgetHub – Sustainable Tech Gadgets</title>
  <link rel="manifest" href="manifest.json">
  </style>
 </head>
 <body>
  <script>
   if ("serviceWorker" in navigator) {
    window.addEventListener("load", () => {
     navigator.serviceWorker
      .register("/serviceworker.js")
      .then((registration) => {
       console.log(
        "✅ Service Worker registered! Scope:",
        registration.scope
       );
      })
      .catch((error) => {
       console.log("❌ Service Worker registration failed:", error);
      });
    });
   }
  </script>
 </body>
</html>
```

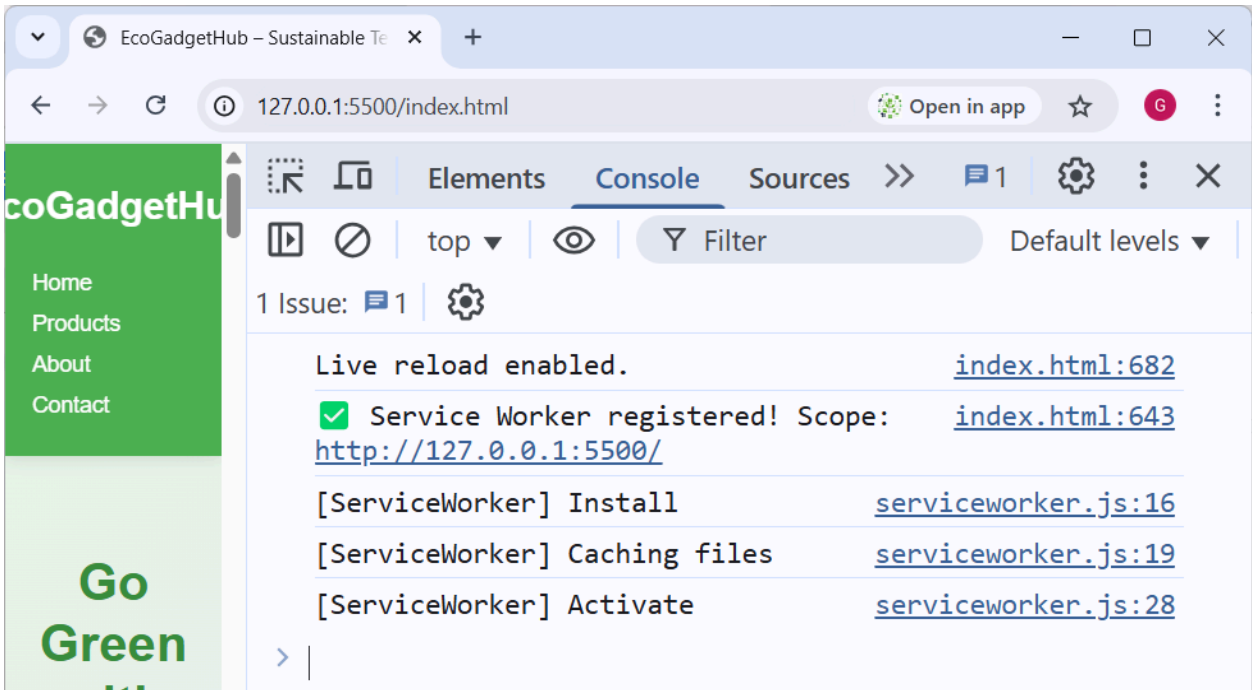
serviceworker.js

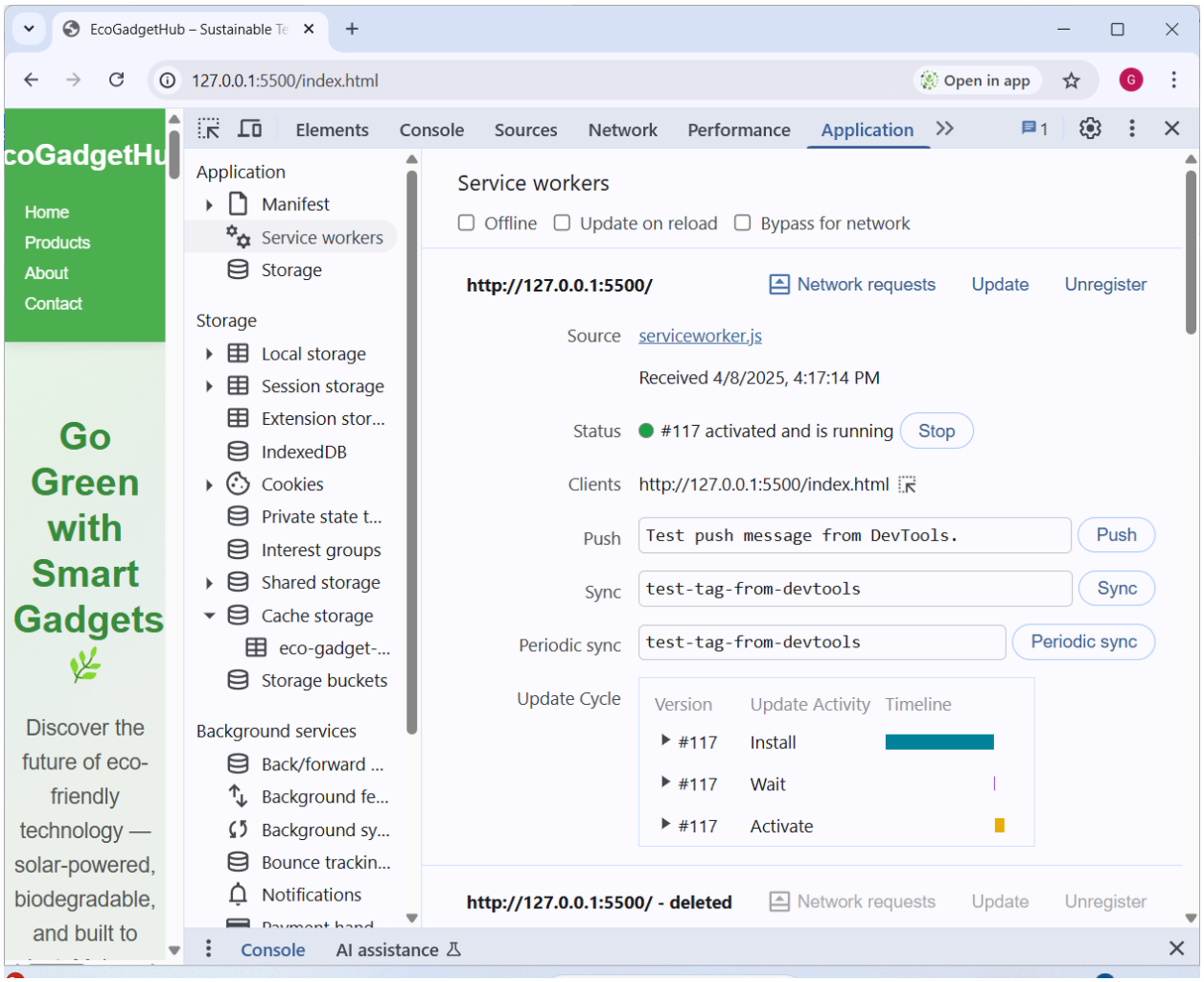
```
const CACHE_NAME = 'eco-gadget-hub-v1';
const urlsToCache = [
 '/',
 '/index.html',
 'icons/icon-192x192.png',
 'icons/icon-512x512.png',
 'images/Biodegradable-Phone-Case.webp',
 'images/Eco-friendly-gadgets-collection.webp',
```

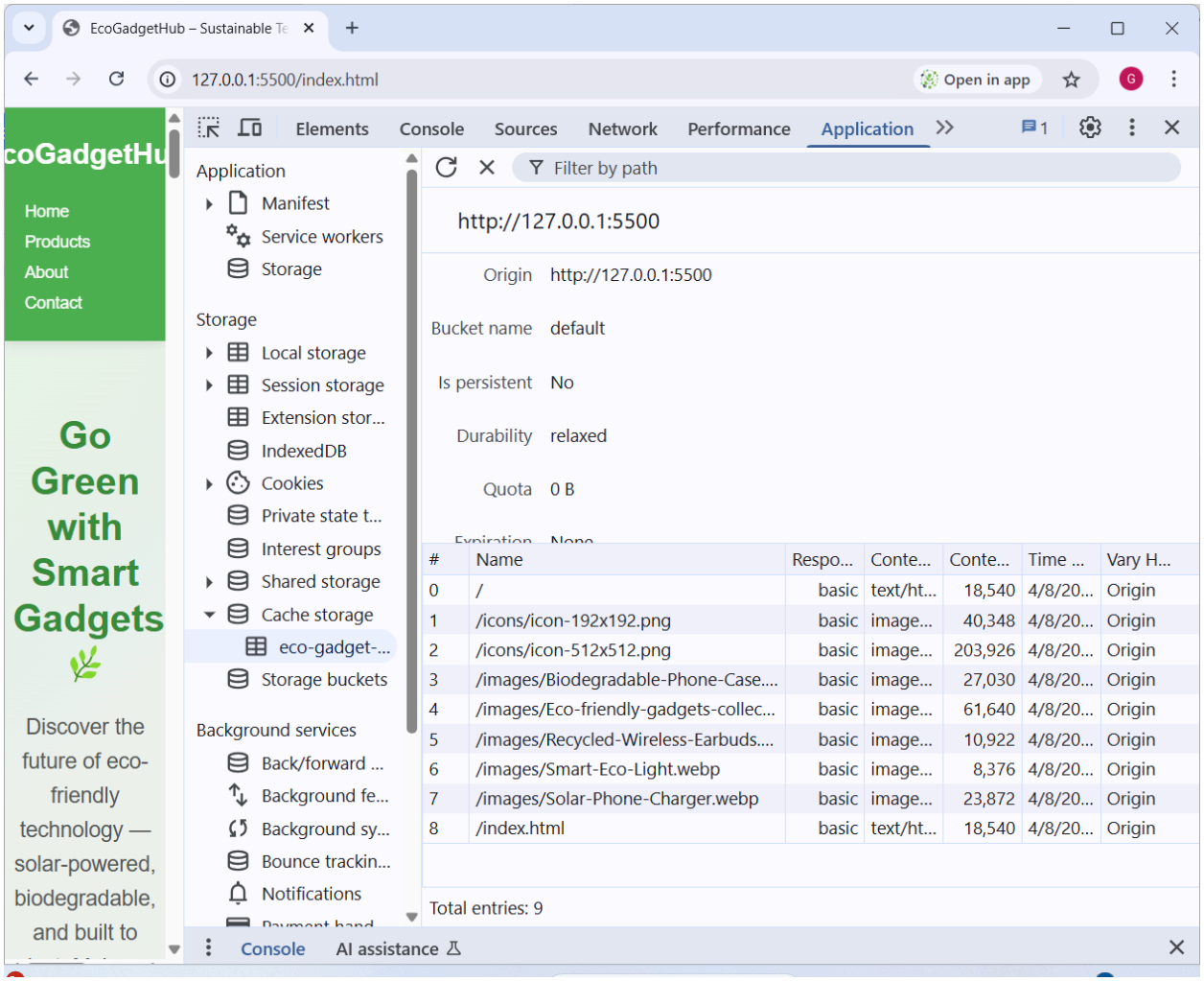
```
  'images/Recycled-Wireless-Earbuds.webp',
  'images/Smart-Eco-Light.webp',
  'images/Solar-Phone-Charger.webp',
];

// Install Event
self.addEventListener("install", (event) => {
  console.log("[ServiceWorker] Install");
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
      console.log("[ServiceWorker] Caching files");
      return cache.addAll(urlsToCache);
    })
  );
});

// Activate Event
self.addEventListener("activate", (event) => {
  console.log("[ServiceWorker] Activate");
  event.waitUntil(
    caches.keys().then((keyList) =>
      Promise.all(
        keyList.map((key) => {
          if (key !== CACHE_NAME) {
            console.log("[ServiceWorker] Removing old cache", key);
            return caches.delete(key);
          }
        })
      )
    )
  );
  return self.clients.claim();
});

// Fetch event
self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request)
      .then((response) => response || fetch(event.request))
```

```
  );
});
```

Screenshot


EcoGadgetHub – Sustainable Te
127.0.0.1:5500/index.html
Open in app
Elements
Console
Sources
top
Filter
Default levels
1 Issue: 1
Live reload enabled. index.html:682
Service Worker registered! Scope: http://127.0.0.1:5500/ index.html:643
[ServiceWorker] Install serviceworker.js:16
[ServiceWorker] Caching files serviceworker.js:19
[ServiceWorker] Activate serviceworker.js:28
Home
Products
About
Contact
Go Green


EcoGadgetHub – Sustainable Te
127.0.0.1:5500/index.html
Open in app
Elements
Console
Sources
Network
Performance
Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension stor...
IndexedDB
Cookies
Private state t...
Interest groups
Shared storage
Cache storage
eco-gadget-...
Storage buckets
Background services
Back/forward ...
Background fe...
Background sy...
Bounce trackin...
Notifications
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5500/
Network requests
Update
Unregister
Source serviceworker.js
Received 4/8/2025, 4:17:14 PM
Status #117 activated and is running
Stop
Clients http://127.0.0.1:5500/index.html
Push
Test push message from DevTools.
Push
Sync
test-tag-from-devtools
Sync
Periodic sync
test-tag-from-devtools
Periodic sync
Update Cycle
Version
Update Activity
Timeline
#117 Install
#117 Wait
#117 Activate
http://127.0.0.1:5500/ - deleted
Network requests
Update
Unregister
Console
AI assistance
Home
Products
About
Contact
Go Green with Smart Gadgets
Discover the future of eco-friendly technology — solar-powered, biodegradable, and built to



Conclusion

In this experiment, we successfully implemented and registered a service worker to cache essential files and enable offline access for EcoGadgetHub. Initially, we faced an issue where the service worker wasn't activating due to incorrect file paths, but we resolved it by verifying the file locations and correcting the URLs in the cache list.

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

Theory

Progressive Web Apps (PWAs) use service workers to enhance user experience by enabling features like offline support, background syncing, and push notifications. In this experiment, we focused on implementing three major service worker events:

1. Fetch Event (Offline Support)
The fetch event allows the service worker to intercept network requests made by the app. In our implementation:

- We first try to load files from the cache.
- If the file is not in cache, we fetch it from the network.
- If the network is unavailable, we show a custom offline.html page. This improves the user experience during poor or no internet connection.

2. Sync Event (Background Sync)
The sync event is used to handle background tasks when the device is back online. In our code:

- We register a background sync with the tag "sync-data".
- When triggered, it can be used to sync data like user cart, orders, etc., to the server. This helps in saving user actions even when they're offline and syncing them later.

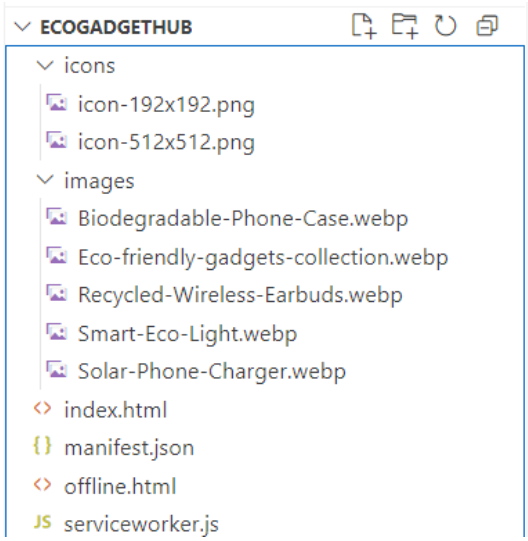
3. Push Event (Push Notifications)
The push event allows the PWA to receive messages even when the app is not open. In our project:

- We ask the user for notification permission.
- When a push message arrives, a notification is displayed to the user with custom content. This keeps users engaged with updates like offers, product arrivals, etc.

What We Did in Our E-commerce PWA:

- Registered a service worker in index.html.
- Handled caching and fallback in fetch for offline use.
- Registered sync-data to simulate background sync.
- Managed push events to show product-related notifications.
- Created a user-friendly offline.html page for no-network cases.

Folder Structure



index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>EcoGadgetHub – Sustainable Tech Gadgets</title>
  <link rel="manifest" href="manifest.json">
  </style>
 </head>
 <body>
  <script>
   if ("serviceWorker" in navigator) {
    window.addEventListener("load", () => {
     navigator.serviceWorker
      .register("/serviceworker.js")
      .then((registration) => {
       console.log(
        "✅ Service Worker registered! Scope:",
        registration.scope
       );

       // 🔔 Request Push Notification Permission
       if ("PushManager" in window) {
        Notification.requestPermission().then((permission) => {
         if (permission === "granted") {
          console.log("🔔 Push notifications granted.");
         } else {
```

```
            console.log("🔕 Push notifications denied.");
          }
        });
      }

      // 🔄 Register Background Sync
      if ("SyncManager" in window) {
        navigator.serviceWorker.ready.then((swReg) => {
          swReg.sync
            .register("sync-data")
            .then(() => {
              console.log("🔄 Sync registered");
            })
            .catch((err) => {
              console.log("❌ Sync registration failed:", err);
            });
        });
      }
    })
    .catch((error) => {
      console.log("❌ Service Worker registration failed:", error);
    });
  });
}
</script>
</body>
</html>
```

offline.html

```
<!-- offline.html -->
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <title>Offline</title>
 <style>
  body {
   font-family: Arial;
   text-align: center;
```

```
    padding: 50px;
    background: #f5f5f5;
   }
  </style>
</head>
<body>
  <h2>You are offline</h2>
  <p>Please check your internet connection and try again.</p>
</body>
</html>
```

serviceworker.js

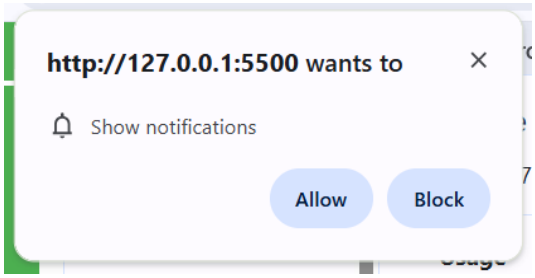
```
const CACHE_NAME = 'eco-gadget-hub-v1';
const urlsToCache = [
  '/',
  '/index.html',
  'icons/icon-192x192.png',
  'icons/icon-512x512.png',
  'images/Biodegradable-Phone-Case.webp',
  'images/Eco-friendly-gadgets-collection.webp',
  'images/Recycled-Wireless-Earbuds.webp',
  'images/Smart-Eco-Light.webp',
  'images/Solar-Phone-Charger.webp',
  'offline.html'
];

// Install Event
self.addEventListener("install", (event) => {
  console.log("[ServiceWorker] Install");
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
      console.log("[ServiceWorker] Caching files");
      return cache.addAll(urlsToCache);
    })
  );
});

// Activate Event
self.addEventListener("activate", (event) => {
  console.log("[ServiceWorker] Activate");
  event.waitUntil(
```

```
  caches.keys().then((keyList) =>
   Promise.all(
    keyList.map((key) => {
     if (key !== CACHE_NAME) {
      console.log("[ServiceWorker] Removing old cache", key);
      return caches.delete(key);
     }
    })
   )
  )
 );
 return self.clients.claim();
});


// Enhanced Fetch Event
self.addEventListener("fetch", (event) => {
 console.log("[ServiceWorker] Fetch", event.request.url);
 const requestURL = new URL(event.request.url);


 // If request is same-origin, use Cache First
 if (requestURL.origin === location.origin) {
  event.respondWith(
   caches.match(event.request).then((cachedResponse) => {
    return ( cachedResponse ||
     fetch(event.request).catch(() => caches.match("offline.html"))
    );
   })
  );
 } else {
  // Else, use Network First
  event.respondWith(
   fetch(event.request)
    .then((response) => {
     return response;
    })
    .catch(() =>
     caches.match(event.request).then((res) => {
      return res || caches.match("offline.html");
     })
```

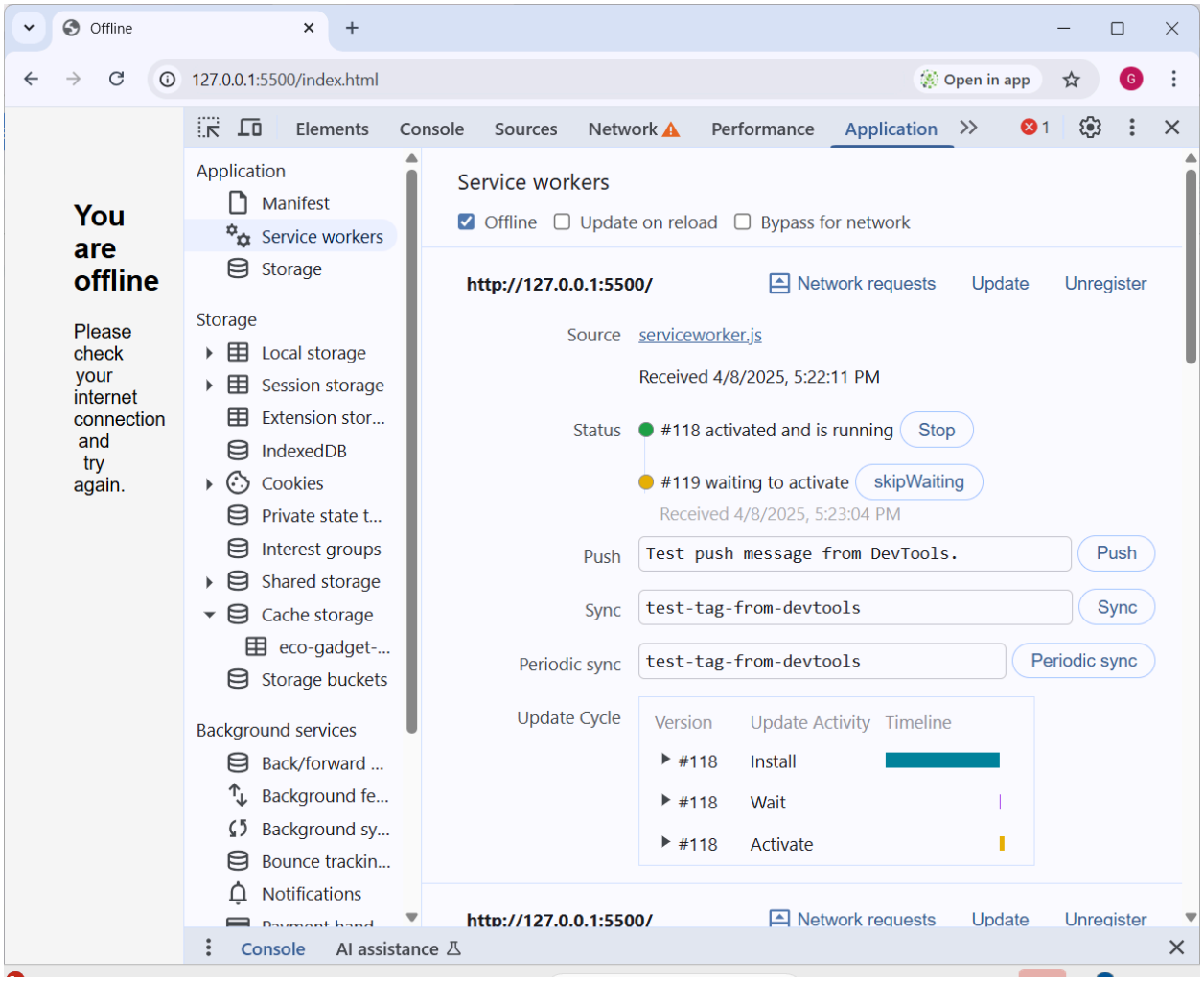
```
      )
    );
  }
});


// Sync Event (simulation)
self.addEventListener("sync", (event) => {
  if (event.tag === "sync-data") {
    event.waitUntil(
      (async () => {
        console.log("Sync event triggered: 'sync-data'");
        // Here you can sync data with server when online
      })()
    );
  }
});


// Push Event
self.addEventListener("push", function (event) {
  if (event && event.data) {
    let data = {};
    try {
      data = event.data.json();
    } catch (e) {
      data = {
        method: "pushMessage",
        message: event.data.text(),
      };
    }


    if (data.method === "pushMessage") {
      console.log("Push notification sent");
      event.waitUntil(
        self.registration.showNotification("Eco Friendly Gadgets Collection", {
          body: data.message,
        })
      );
    }
```

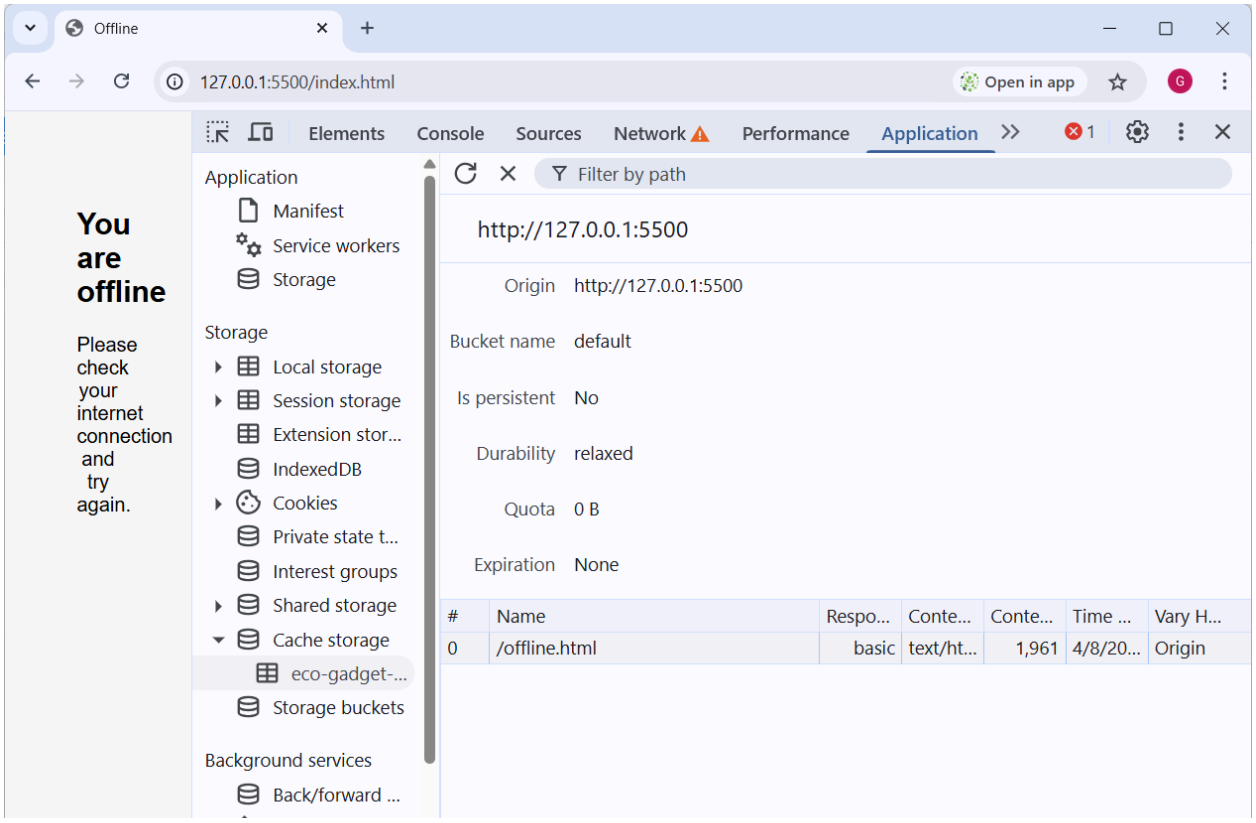
```
  }
});
```

Screenshot

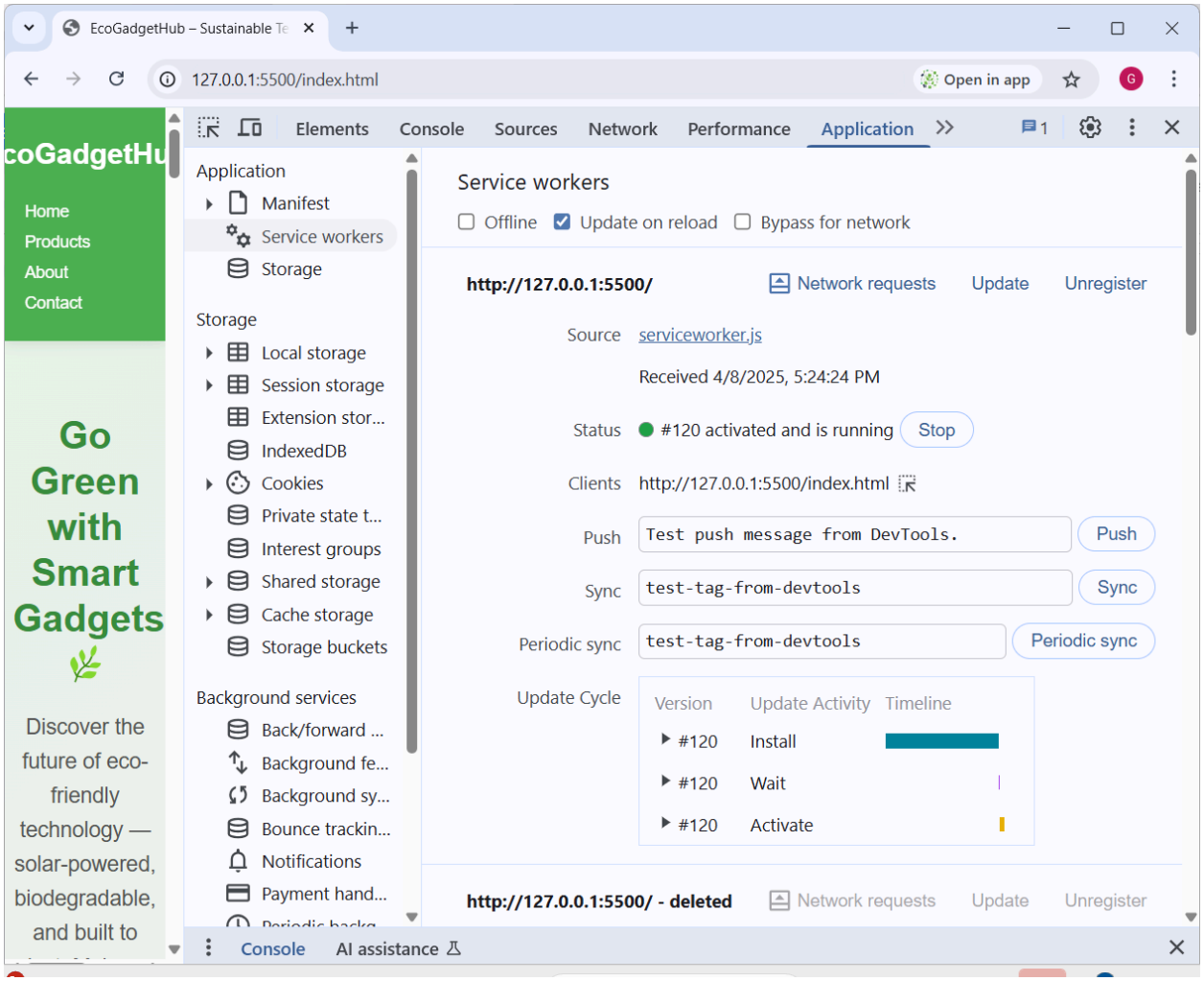


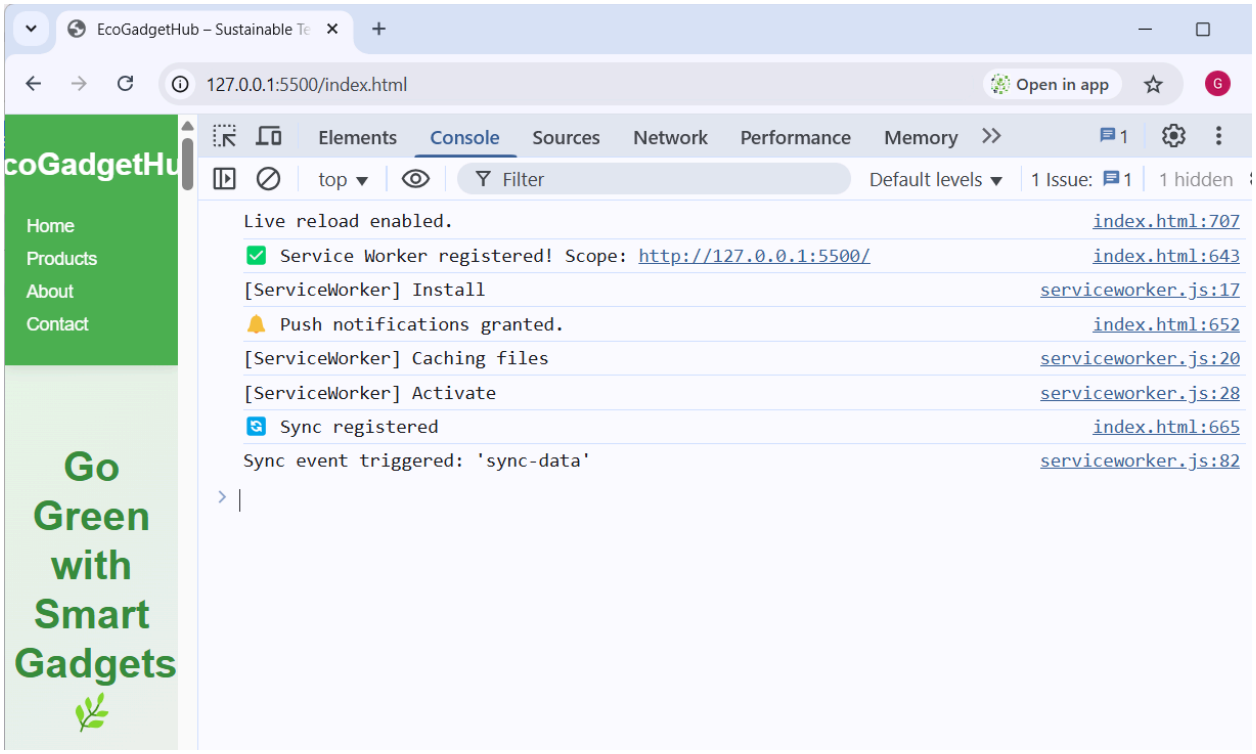
1. Test: Fetch Event (Offline Support)



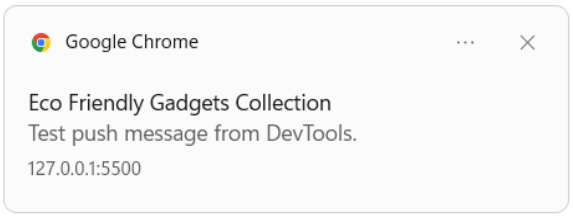


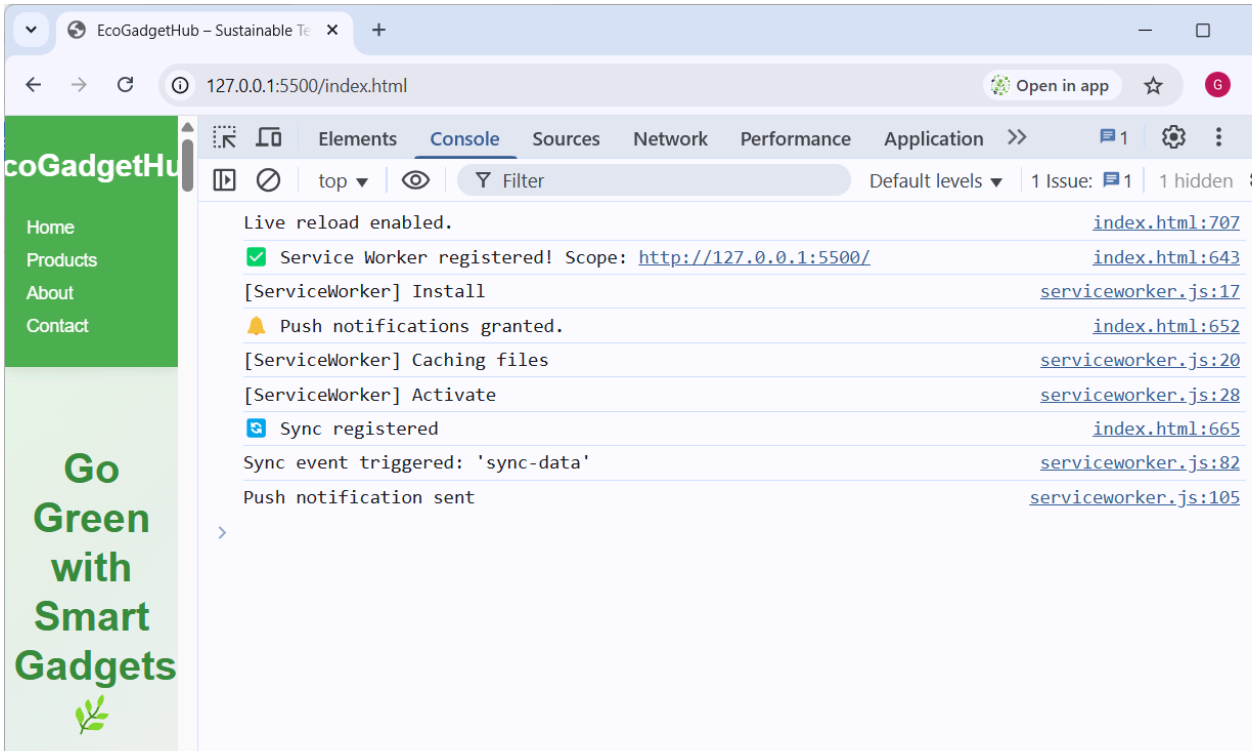
2. Test: Background Sync Event





3. Test: Push Notification Event





Conclusion

In this experiment, we successfully implemented fetch, sync, and push events using service workers to enhance offline functionality, background syncing, and notifications for EcoGadget Hub. Initially, we faced a cache loading issue due to incorrect file paths, which we resolved by carefully updating the paths and ensuring all required files were listed in the cache.

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

Github Link: https://ganesh23105.github.io/EcoGadgetHubPWA/

Theory

GitHub Pages is a free hosting service provided by GitHub to publish static websites directly from a GitHub repository. It supports HTML, CSS, JavaScript, and other front-end files, making it ideal for hosting PWAs.

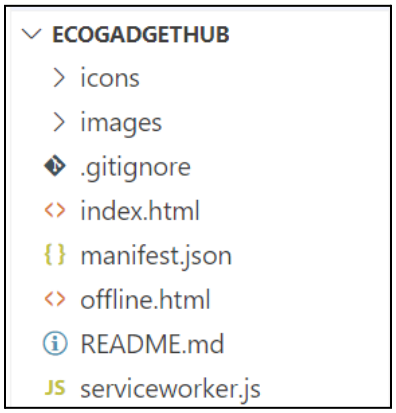
What we implemented in our code

In this experiment, we deployed our E-commerce PWA project, "Eco Friendly Gadgets Collection", to GitHub Pages using these steps:

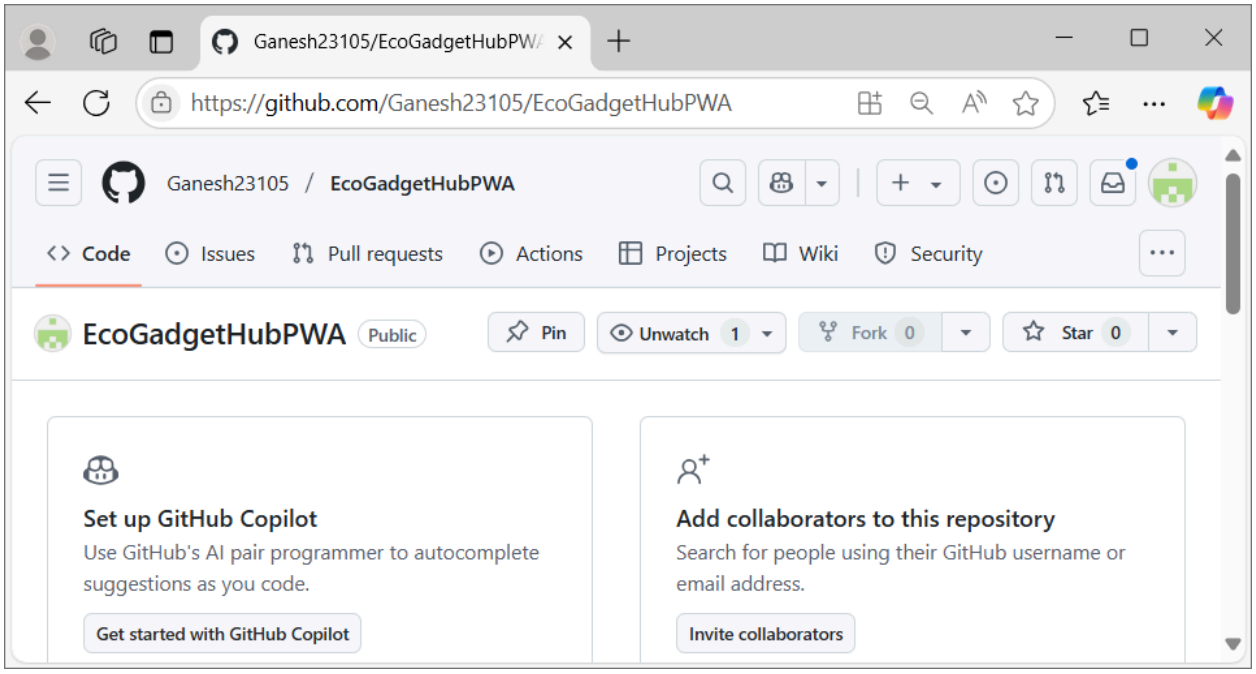
- We ensured all necessary PWA files were in place, such as index.html, manifest.json, serviceworker.js, and offline assets.
- The index.html file used relative paths correctly so resources load properly after deployment.
- We registered the service worker to enable offline access and caching of essential files.
- The manifest.json provided metadata to support "Add to Home Screen" functionality.
- We created a GitHub repository and pushed all project files to it.
- Then, from GitHub Settings → Pages, we selected the main branch and the / (root) folder as the source.
- After saving, GitHub generated a live link through which we accessed the deployed PWA.

This deployment made our PWA fully functional and accessible online with service worker support, offline capabilities, and installability features.

Folder Structure



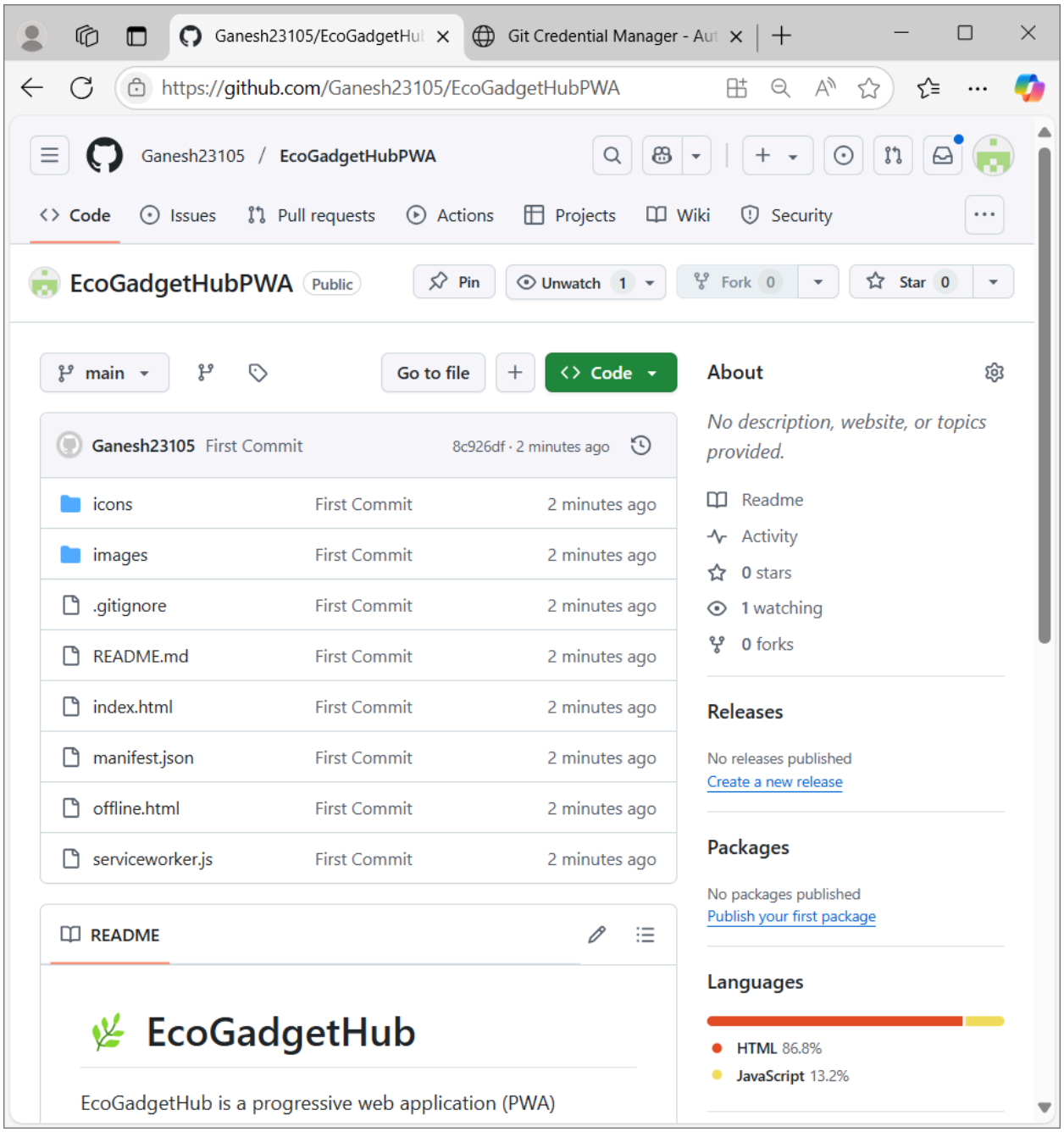
Create a GitHub Repository

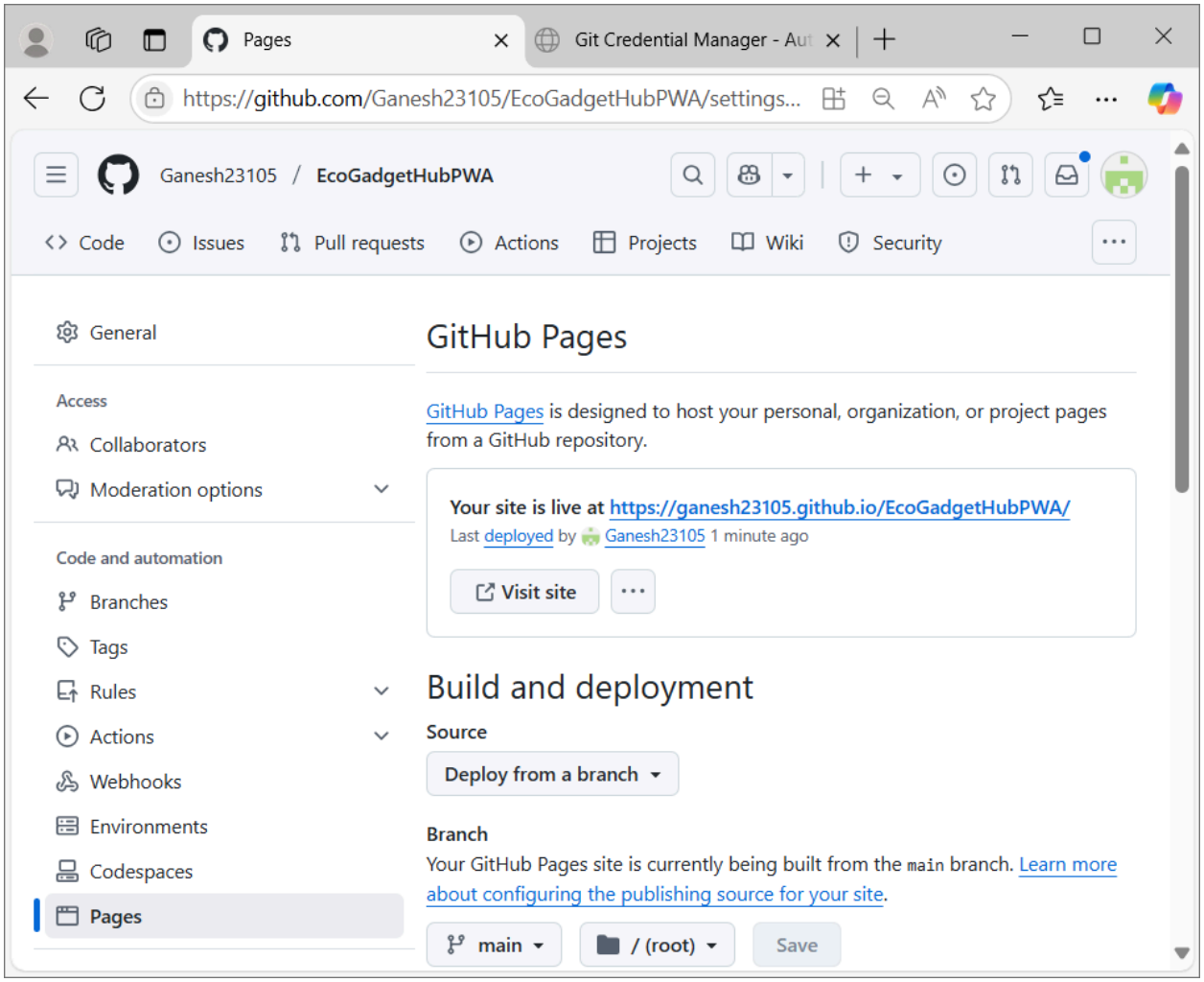


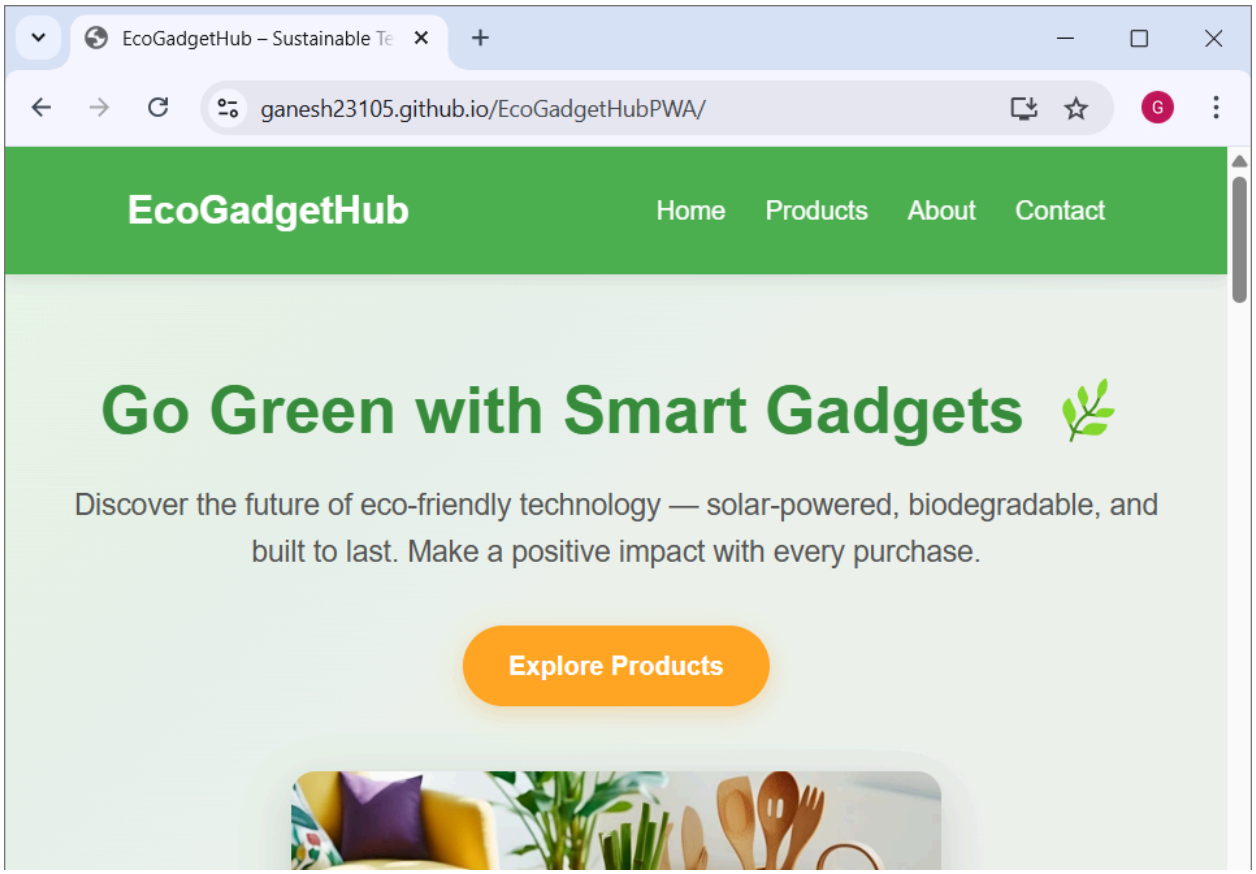
Link Your Local Project to GitHub and Push Changes.

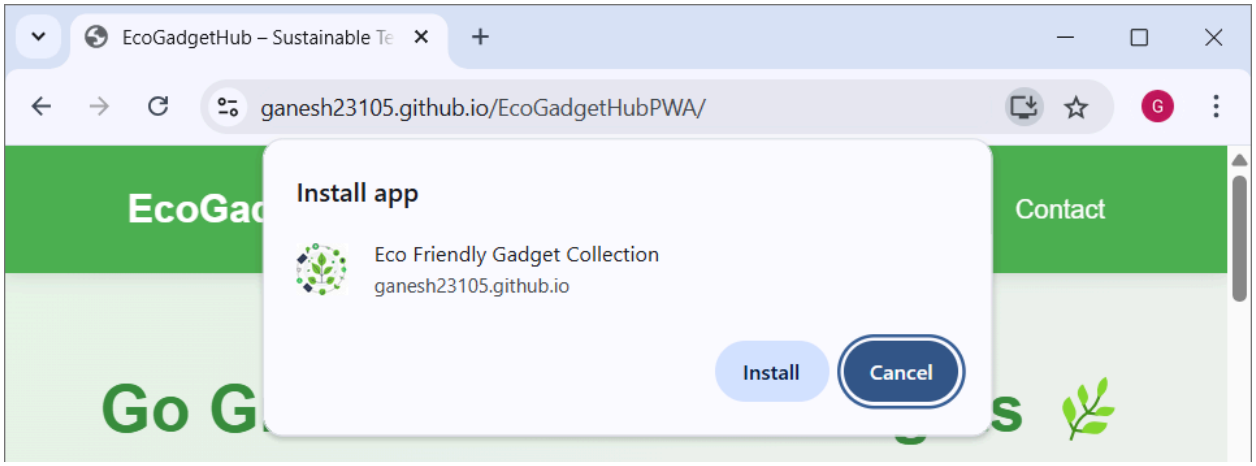
```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS D:\Users\Ganesh\Documents\EcoGadgetHub> git init
Initialized empty Git repository in D:/Users/Ganesh/Documents/EcoGadgetHub/.git/
PS D:\Users\Ganesh\Documents\EcoGadgetHub> git add .
PS D:\Users\Ganesh\Documents\EcoGadgetHub> git commit -m "First Commit"
[main (root-commit) 8c926df] First Commit
 13 files changed, 868 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README.md
 create mode 100644 icons/icon-192x192.png
 create mode 100644 icons/icon-512x512.png
 create mode 100644 images/Biodegradable-Phone-Case.webp
 create mode 100644 images/Eco-friendly-gadgets-collection.webp
 create mode 100644 images/Recycled-Wireless-Earbuds.webp
 create mode 100644 images/Smart-Eco-Light.webp
 create mode 100644 images/Solar-Phone-Charger.webp
 create mode 100644 index.html
 create mode 100644 manifest.json
 create mode 100644 offline.html
 create mode 100644 serviceworker.js
PS D:\Users\Ganesh\Documents\EcoGadgetHub>
```

```
PS D:\Users\Ganesh\Documents\EcoGadgetHub> git remote add origin https://github.com/Gar
PS D:\Users\Ganesh\Documents\EcoGadgetHub> git push -u origin main
info: please complete authentication in your browser...
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 4 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (17/17), 373.18 KiB | 7.18 MiB/s, done.
Total 17 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Ganesh23105/EcoGadgetHubPWA.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS D:\Users\Ganesh\Documents\EcoGadgetHub>
```




Pages
Git Credential Manager - Aut
https://github.com/Ganesh23105/EcoGadgetHubPWA/settings...
Ganesh23105 / EcoGadgetHubPWA
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
GitHub Pages
GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.
Your site is live at https://ganesh23105.github.io/EcoGadgetHubPWA/
Last deployed by Ganesh23105 1 minute ago
Visit site
Build and deployment
Source
Deploy from a branch
Branch
Your GitHub Pages site is currently being built from the main branch. Learn more about configuring the publishing source for your site.
main
/ (root)
Save


EcoGadgetHub – Sustainable Te
ganesh23105.github.io/EcoGadgetHubPWA/
EcoGadgetHub
Home
Products
About
Contact
Go Green with Smart Gadgets
Discover the future of eco-friendly technology — solar-powered, biodegradable, and built to last. Make a positive impact with every purchase.
Explore Products



Conclusion

In this experiment, we successfully deployed the Eco Friendly Gadgets Collection PWA to GitHub Pages by ensuring proper setup of service workers and manifest files. Initially, we faced 404 errors for a CSS-based image and offline caching issues, which we resolved by correcting the image path and ensuring only valid resources were cached.

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

Theory:

Lighthouse is a tool provided by Google that helps us check how well our web app performs as a Progressive Web App (PWA). It analyzes important factors like performance, accessibility, SEO, best practices, and PWA features.

In this experiment, we used Lighthouse in Chrome DevTools to test our deployed PWA. Lighthouse runs a series of audits and gives a score out of 100 for each category. These scores show how well our app performs and where we can improve.

What we implemented:

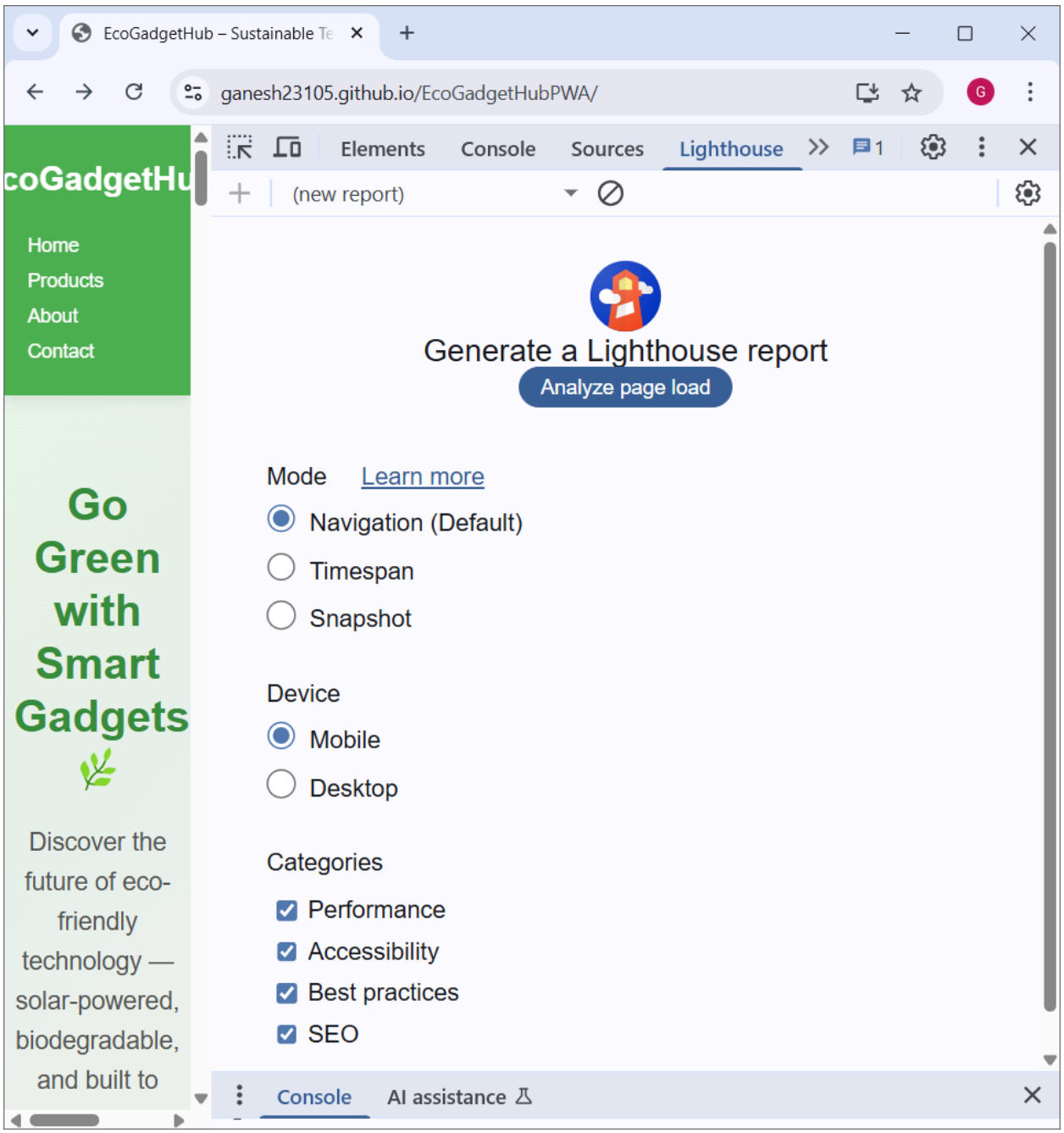
- We deployed our PWA (EcoGadgetHubPWA) on GitHub Pages.
- Then, we ran Lighthouse from Chrome DevTools on the live site.
- Lighthouse gave us excellent scores:
    - Performance: 100
    - Accessibility: 89
    - Best Practices: 93
    - SEO: 91

These high scores show that our app is fast, user-friendly, and optimized for search engines.
Some minor suggestions were also given, like improving color contrast and adding a meta description, which can help make the app even better.

manifest.json

```
{
  "name": "Eco Friendly Gadget Collection",
  "short_name": "Ecogadgethub",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#4caf50",
  "orientation": "portrait",
  "scope": "/",
  "icons": [
   {
    "src": "icons/icon-192x192.png",
    "sizes": "192x192",
    "type": "image/png",
    "purpose": "any maskable"
   },
   {
    "src": "icons/icon-512x512.png",
```

```
    "sizes": "512x512",
    "type": "image/png",
    "purpose": "any maskable"
   }
  ]
}
```



Conclusion

In this experiment, we tested the EcoGadgetHub PWA using Google Lighthouse and achieved high scores in performance, accessibility, best practices, and SEO. Initially, we faced a layout shift warning due to missing image dimensions, which we resolved by adding explicit width and height to all image tags.

https://ganesh23105.github.io/EcoGadgetHubPWA/

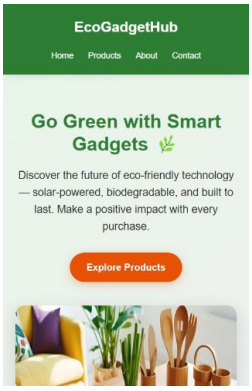
Performance	Accessibility	Best Practices	SEO
99	93	93	100

99

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

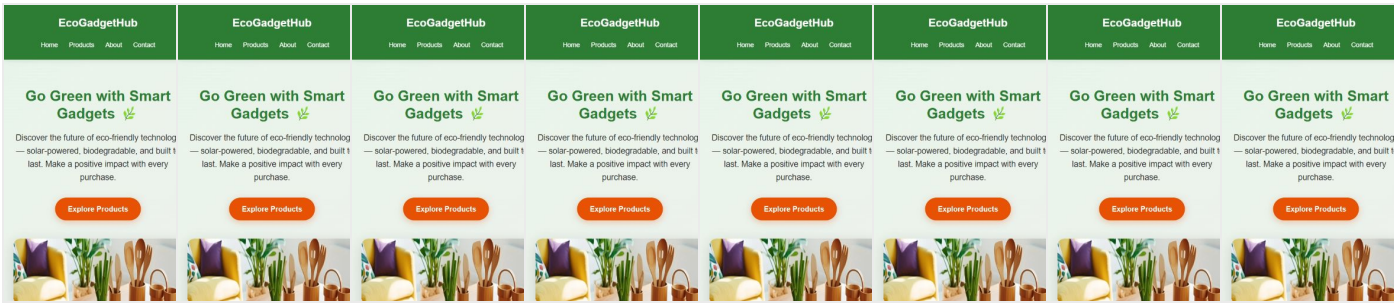


METRICS Expand view

- First Contentful Paint: 1.2 s
- Largest Contentful Paint: 1.7 s

- Total Blocking Time
  80 ms
- Cumulative Layout Shift
  0
- Speed Index
  1.2 s

View Treemap



Show audits relevant to: All LCP TBT

DIAGNOSTICS

- Serve static assets with an efficient cache policy — 5 resources found
- Avoid long main-thread tasks — 2 long tasks found
- Avoid chaining critical requests — 1 chain found
- Largest Contentful Paint element — 1,710 ms

More information about the performance of your application. These numbers don't directly affect the Performance score.

PASSED AUDITS (34)

Show

93

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.

CONTRAST

- ▲ Background and foreground colors do not have a sufficient contrast ratio.

These are opportunities to improve the legibility of your content.

NAVIGATION

- ▲ Heading elements are not in a sequentially-descending order

These are opportunities to improve keyboard navigation in your application.

ADDITIONAL ITEMS TO MANUALLY CHECK (10)

Show

These items address areas which an automated testing tool cannot cover. Learn more in our guide on conducting an accessibility review.

PASSED AUDITS (16)

Show

NOT APPLICABLE (39)

Show

Best Practices

USER EXPERIENCE

- ▲ Displays images with incorrect aspect ratio
- ▲ Serves images with low resolution

TRUST AND SAFETY

- ○ Ensure CSP is effective against XSS attacks
- ○ Use a strong HSTS policy

- Ensure proper origin isolation with COOP
- Mitigate clickjacking with XFO or CSP

PASSED AUDITS (13) Show

NOT APPLICABLE (2) Show

SEO

These checks ensure that your page is following basic search engine optimization advice. There are many additional factors Lighthouse does not score here that may affect your search ranking, including performance on Core Web Vitals. Learn more about Google Search Essentials.

ADDITIONAL ITEMS TO MANUALLY CHECK (1) Show

Run these additional validators on your site to check additional SEO best practices.

PASSED AUDITS (8) Show

NOT APPLICABLE (2) Show

Captured at Apr 9, 2025, 12:19 AM GMT+5:30
Initial page load
Emulated Moto G Power with Lighthouse 12.4.0
Slow 4G throttling
Single page session
Using Chromium 135.0.0.0 with devtools

Generated by **Lighthouse** 12.4.0 | File an issue

GANESH SANJAY PANDHRE / D15B / 41

SDS | Page No | Date

MPL Assignment 01

Q.1 a] Explain the key features and advantages of using Flutter for mobile app development.

=> key Features of Flutter :-

1. Single Codebase - write one code for both Android and iOS.
2. Fast Performance - Uses Dart language and a high-performance rendering engine.
3. Hot Reload - See changes instantly without restarting the app.
4. Rich UI components - comes with customizable widgets for smooth UI Design.
5. Native-like Experience - Provides high-quality animations and fast execution.
6. Cross-Platform support - can be used for mobile, web, and desktop apps.
7. Open-Source - Free to use and has a strong developer community.

Advantages of Using Flutter :-

1. Saves Times & Effort - single codebase for multiple platforms.
2. High Speed Development - Hot Reload feature speeds up coding.
3. cost-Effective - Reduces development cost and time.
4. Attractive UI - Provides beautiful and customizable widgets.
5. Good Performance - Uses Dart and Skia for fast and smooth rendering.

b] Discuss how the flutter framework differs from traditional approaches and why it has gained popularity in the developer community.

=>

How Flutter Differs from Traditional Approaches :-

1. Single Codebase - Traditional methods need separate code for Android (Java / Kotlin) and iOS (Swift / Objective-C), but Flutter uses one code for both.
2. Hot Reload - Traditional apps requires full restart after changes, but Flutter updates instantly.
3. UI Rendering - Traditional apps use native components, while Flutter has its own rendering engine (Skia) for faster Performance.
4. Performance - Flutter compiles directly to native machine code, making it faster than frameworks that use a bridge.
5. Customization - Traditional UI Design depends on platform-specific components, but Flutter provides fully customizable widgets.

Why Flutter is popular Among Developers :-

1. Fast Development - Hot Reload and single codebase save time.
2. Cross-Platform support - works on mobile, web & desktop.
3. Beautiful UI - Rich, customizable widgets for modern designs.

4. High Performance - Runs smoothly without a bridge like React Native.
5. Active community & Google support - Regular updates and strong community help developers.

Q. 2 a] Describe the concept of the widget tree in flutter. Explain how widget composition is used to build complex user interfaces.

⇒ Concept of widget Tree in Flutter :-
In flutter, everything is a widget. Widgets are arranged in a tree structure, called the widget tree. The tree representing the UI of the app, where parent widgets contain child widgets.
For example, a Scaffold widget can have a column widget, which contains Text and Button widgets. Changes in widgets update the tree dynamically.

Widget composition for complex UI :-
Flutter uses small, reusable widgets to build complex UI block, developers combine multiple small widgets like Rows, columns, containers, and Buttons.
For example :
1. A ListView can contain multiple card widgets.
2. A column can hold Text, Images, and Buttons.

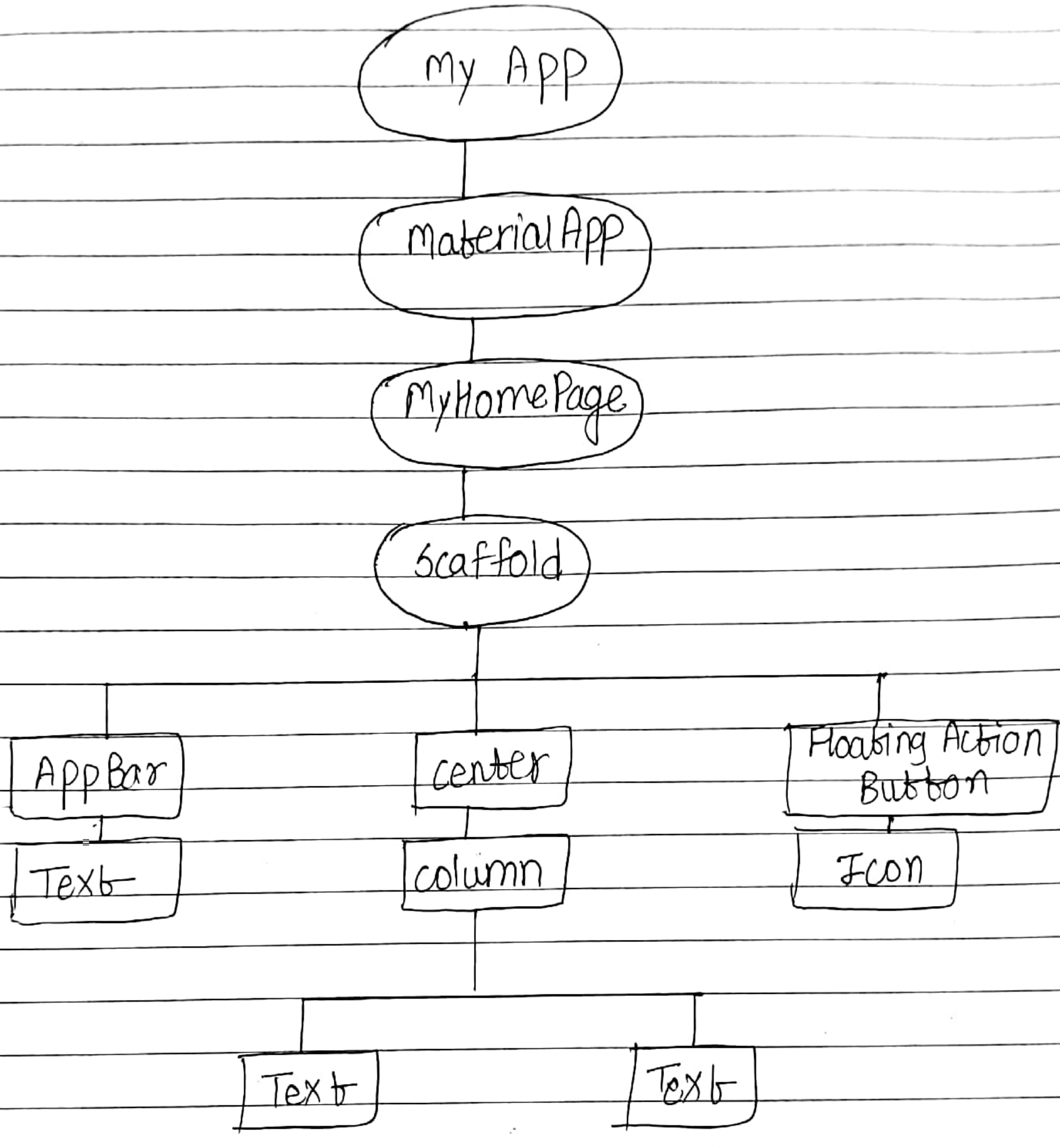
This modular approach makes the UI flexible, readable, and easy to manage.

Q. 2 b] Provide examples of commonly used widgets and their roles in creating a widget tree.

⇒ Commonly Used widgets and Their Roles in a Widget Tree :

1. Scaffold :- Provides the basic layout structure.
2. AppBar :- Displays the top navigation bar with a title.
3. Text :- Displays simple text on the screen.
4. Image :- shows images from assets or URLs.
5. Container :- Used for styling.
6. Row :- Arranges child widgets horizontally.
7. Column :- Arranges child widgets vertically.
8. ListView :- Displays scrollable lists.
9. ElevatedButton :- A clickable button with elevation.
10. TextField :- Used for input.
11. Card :- Creates a styled box for displaying content
12. Stack :- Overlays widgets on top of each other.

Example Widget Tree :-



~~Explain the importance of state management in flutter applications.~~

Importance of State Management in Flutter Applications

State management is important because it controls how the app stores, updates, and displays data when the user interacts with it.

Why State Management is Needed?

1. Keeps UI Updated - Ensures that the app reflects changes.
2. Improves Performance - Updates only necessary parts of the UI instead of reloading everything.
3. Manages Complex Data - Helps handle user inputs, API Data, and navigation efficiently.
4. Ensures Smooth User Experience - Keeps the app responsive and interactive.

Types of State in Flutter :-

1. Local State - Managed within a single widget using StatefulWidget.
2. Global State - Shared across multiple screens using Provider, Riverpod, BLOC, or Redux.

Without proper state management, the app may behave unpredictably or show outdated data.

Q.3 a] Discuss the importance of state management in Flutter applications.

=> Importance of State Management in Flutter Applications :-

State management is important because it controls how the app stores, updates, and displays data when the user interacts with it.

Why State Management is Needed?

1. Keeps UI Updated - Ensures that the app reflects changes.
2. Improves Performance - Updates only necessary parts of the UI instead of reloading everything.
3. Manages Complex Data - Helps handle user inputs, API Data, and navigation effeciently.
4. Ensures Smooth User Experience - keeps the app responsive and interactive.

Types of State in Flutter :-

1. Local state - Managed within a single widget using StatefulWidget.
2. Global state - shared across multiple screens using Provider, Riverpod, Bloc, or Redux.

Without proper state management, the app may behave unpredictably or show outdated data.

b] compare and contrast the different state Maragement approaches available in Flutter, such as setstate, Provider and Riverpod. Provide scenarios where each approach is suitable.

=>

Approach	How it Works	When to Use.
i) setstate	Updates UI by calling setstate() in a stateful Widget.	Best for small apps or managing state within a single widget. Example: Toggling a button color.
ii) Provider	Uses Inherited Widget to share across widgets efficiently.	Suitable for medium-sized apps where data needs to be shared between multiple widgets. Examples: Managing user authentication.
iii) Riverpod	An improved version of Provider with better Performance and simple syntax.	Best for large apps that need complex state management with dependency injection. Example: Handling API data and app-wide themes.

Choosing the Right Approach :-
- Use setState for simple UI updates.
- Use provider for moderate state sharing across widgets.
- Use Riverpod for scalable, well-structured applications.

Q.4 a] Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution.

=> Process of Integrating Firebase with a Flutter Application :-

1. Create a Firebase Project - Go to [Firebase console (https://console.firebase.google.com)], create a new project.
2. Add Firebase to Flutter App - Register the app and download the google-services.json or GoogleService-Info.plist.
3. Install Firebase Packages - Add dependencies like 'firebase_core' and 'firebase_auth' in 'pubspec.yaml'.
4. Initialize Firebase - Import Firebase in 'main.dart' and call Firebase.initializeApp().
5. Use Firebase services - Implement authentication, database, or cloud functions as needed.

Benefits of using Firebase as a backend solution:
1. Real-time Database - Syncs data instantly across devices.
2. Authentication - Provides ready-to-use sign-in options.
3. Cloud Firestore - Stores structured data efficiently.
4. Hosting & Sharage - Hosts web apps and stores files securely.
5. Scalability - Handles large user bases without manging servers.
6. Push Notifications - Sends alerts and updates to users.

Q.4 b] Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is acheived.

=> common Firebase services used in Flutter Development:
1. Firebase Authentication - Provides user sigh-in methods.
2. Cloud Firestore - A NoSQL database that stores and syncs data in real time.
3. Firebase Realtime Database - stores and updates data instantly across all connected devicees.
4. Firebase Cloud Storage - Used for storing and retrieving files like images and videos.
5. Firebase Cloud Messaging (FCM) - sends push notifications to users.
6. Firebase Hosting - Deploys web apps with fast and secure hosting.

How Data Synchronization is Achieved:

1. Real-time Updates - Firestore and Realtime Database sync data across devices instantly.
2. Listeners & Streams - Widgets ~~likely~~ listen for changes and update the UI automatically.
3. Offline Support - Firebase ~~catches~~ caches data, allowing apps to work offline and sync when online.

This ensures fast, smooth, and automatic data updates in Flutter apps.

05/05

SDS Page No. Date

GANESH SANJAY PANDHRE/DISB/41

*MPL Assignment 02 *

Q. 1 Define Progressive web App (PWA) and explain its significance in modern web development. Discuss the key characteris that differentiate PWAs from traditional mobile apps.

⇒ A progressive web App (PWA) is a type of web application that works like a mobile app but runs in a browser. It can be installed on a device, works offline, and provides a fast and smooth user experience.

Significance of PWA in modern web Development:

1. Cross-Platform Compatibility - Works on both mobile and desktop with a single codebase.
2. Offline Support - Can function without the internet using cached data.
3. Fast Performance - Loads quickly, even on slow networks.
4. No App store Required - Users can install it directly from the browser.
5. Lower Development cost - One PWA can replace separate Android and ios apps.

Key Differences Between PWA and Traditional Mobile Apps:

Feature	PWA	Traditional Mobile App
i) Installation	Direct from browser	Download from App Store.
ii) Internet Required	Works offline with caching	Usually requires internet.
iii) Performance	Fast with service workers.	Faster but needs installation.
iv) Updates	Automatic, no app store approval	Manual updates needed.
v) Development cost	Lower	Higher

PWAs combine the best of web and mobile apps, making them efficient and user-friendly.

Q. 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.

=> Responsive Web design (RWD) is a technique that makes web pages adjust automatically to different screen sizes and devices. It ensures a good user experience on mobiles, tablets, and desktops without needing separate versions of a website.

SDS | Page No.
Date / /

Importance of Responsive Design in PWAs:

1. Better User Experience - PWAs work smoothly on any Device.
2. Faster Load time - Optimized design improves speed.
3. SEO Benefits - Google ranks responsive sites higher.
4. Cost-Effective - No need to build multiple versions for different screens.

Comparison of Web Design approaches:

Approach	How It Works	Pros	Cons
Responsive	Uses flexible grids and CSS media queries to adjust layout.	Works on all devices, improves SEO.	Can be complex to design.
Fluid	Uses percent-based widths instead of fixed pixels, so elements resize smoothly	Works well on different screen sizes, easy to implement.	Less control over layout on large screens.
Adaptive	Uses fixed layouts that changes at specific breakpoints.	Optimized for known screen sizes.	More effort required to design for each screen size.

Q. 3 Describe the lifecycle of services workers, including registration, installation, and activation phases.

=>

Lifecycle of service workers :-

A service worker is a script that runs in the background and helps a web app work offline, load faster, and send push notifications. Its lifecycle has three main phases:

1. Registration Phase :-
- The browers registers the service worker using Javascript.

Code Example :-

```
if ('serviceWorker' in navigator) {
    navigator.service.worker.register ('/sw.js')
    .then (() => console.log ('service worker Registered'))
    .catch (error => console.log ('Registration Failed:', error));
}
```

- This tells the browser to install and activate the service worker.

2. Installation Phase
- The service worker downloads necessary files and stores them in cache.
- If successful, it moves to the activation phase.

SDS | Page No.
Date / /

code Example:

```
self.addEventListener('install', event => {
    event.waitUntil(
        caches.open('app-cache').then(cache => {
            return cache.addAll(['/', '/index.html',
                                 '/styles.css']);
        });
    });
});
```

- This ensures the app loads even without the internet.

3. Activation Phase
- The old service worker is replaced with the new one.
- Unused cache files from the previous version are deleted.

code Example:

```
self.addEventListener('activate', event => {
    event.waitUntil(
        caches.keys().then(keys => {
            return Promise.all(keys.map(key => {
                if(key !== 'app-cache'){
                    return caches.delete(key);
                }
            }));
        })
    );
});
```

Final step: Fetch & Sync
Once activated, the service worker intercepts network requests, serves cached files, and syncs data when the internet is available.

The lifecycle makes PWAs faster, more reliable, and capable of working offline.

Q. 4 Explain the use of IndexedDB in the service worker for data storage.

=> - Use of IndexedDB in service workers for Data storage :- IndexedDB is a browser database that stores large amounts of structured data like JSON objects. It helps PWAs work offline by saving and retrieving data efficiently.

- Why use IndexedDB in service Workers?
1. Offline support - stores data when offline and syns it later.
2. Efficient storage - saves structured data like user settings, cart items, or forms inputs.
3. Faster Access - Retrieves data quickly without needing a network request.
4. Persistent Data - Data remains saved even after the brower is closed.

SDS Page No.
Date / /

Opening the Database

```
let db;
let request = indexedDB.open('MyDatabase', 1);

request.onsuccess = function(event){
    db = event.target.result;
};
```

Creating a store & Adding Data.

```
request.onupgradeneeded = function(event){
    let db = event.target.result;
    let store = db.createObjectStore('users', {keyPath: "id"
                                               });
    store.add({id: 1, name: 'John Doe', age: 25});
};
```

Fetching Data in Service Worker

```
let transaction = db.transaction('users', 'readonly');
let store = transaction.objectStore('users');
let getUser = store.get(1);

getUser.onsuccess = function(){
    console.log(getUser.result);
};
```