Name : Vivek Gupta
Div : D15B
Roll No : 19

MPL Idea Summary

Team Finder: A Local Sports Team Formation App

Problem Statement:
Many sports enthusiasts struggle to find teammates in their locality for casual games or practice sessions. Whether it's football, basketball, cricket, or any other sport, the lack of an easy way to connect with nearby players often results in missed opportunities for playing. Existing platforms focus on professional-level team management, leaving local players without a dedicated solution.

Solution:
Team Finder is an application that helps users form or join local teams for instant play. It enables users to:

- Create teams by specifying a name and selecting a sport.
- Find and join existing teams based on their preferences.
- Communicate with team members through an in-app chat feature.
- Navigate easily with an intuitive UI and a bottom navigation bar for seamless interaction.

Example:
Rahul loves playing football but struggles to find enough players in his area. Using Team Finder, he creates a team, and within minutes, other local football players join. Meanwhile, Priya, a basketball player, searches for a team and finds "Thunder Hoops," which she joins instantly. The in-app chat feature allows members to coordinate game timings and locations.

Impact:
Team Finder simplifies the process of forming and managing local sports teams. It promotes social engagement, encourages physical activity, and helps users find like-minded sports enthusiasts effortlessly. The app creates a stronger community of players by bridging the gap between individual sports lovers and local teams.

Name : Vivek Gupta
Div : D15B
Roll No : 19

MPL Practical 01

Aim: Installation and Configuration of Flutter Environment.

Theory:

Flutter is an open-source UI toolkit by Google for building cross-platform applications. To start working with Flutter, we need to install the Flutter SDK and set up necessary tools.

Installation Steps:

Step 1: Download and Extract Flutter SDK
- Visit https://docs.flutter.dev/get-started/install and download Flutter SDK for Windows.
- Extract the ZIP file and place it in C:/Flutter.

Step 2: Set Up System Path
- Open System Properties → Environment Variables.
- Edit Path under System Variables.
- Add C:/Flutter/bin and save changes.

Step 3: Verify Installation
- Open Command Prompt and run: flutter doctor
- This checks if all components are installed.

Installing Android SDK and Android Studio

Step 4: Install Android Studio
- Download and install Android Studio.
- Follow setup instructions and complete the installation.

Step 5: Accept Android Licenses
- Run the command: flutter doctor --android-licenses
- Accept all licenses by typing y.

Setting Up an Android Emulator

Step 6: Create an Emulator
- Open Android Studio → AVD Manager → Create Virtual Device.
- Select a device and system image, then finish setup.
- Start the emulator.

Installing Flutter and Dart Plugins

Step 7: Install Plugins in Android Studio
- Go to File → Settings → Plugins.

- Search and install Flutter and Dart plugins.
- Restart Android Studio.

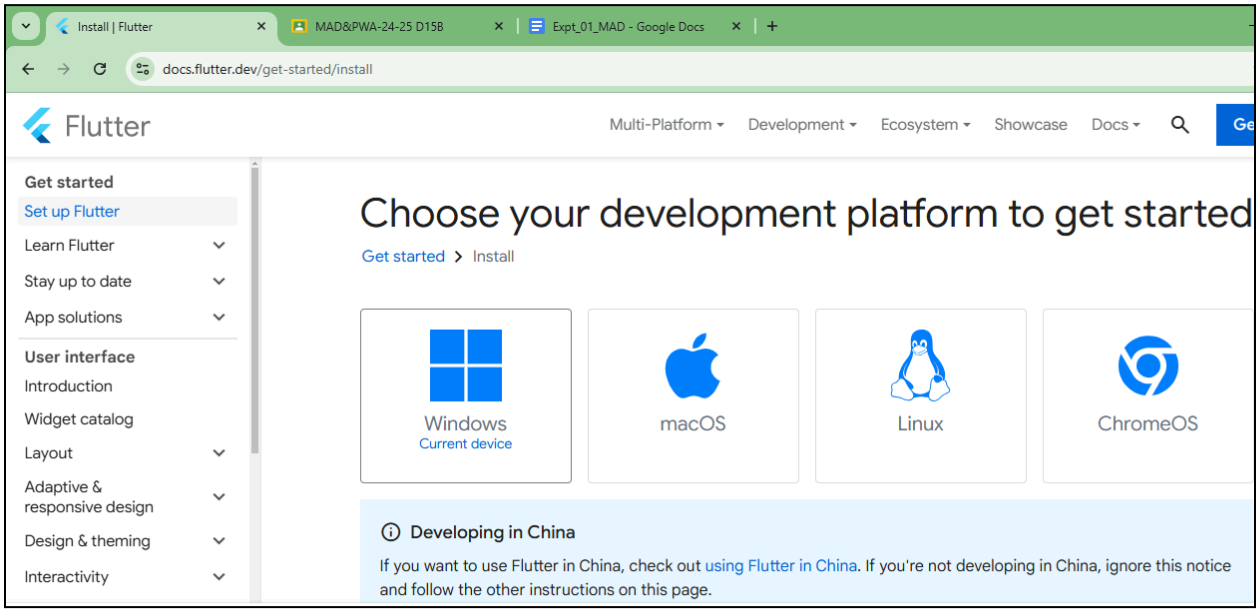
Final Verification

Step 8: Run Flutter Doctor

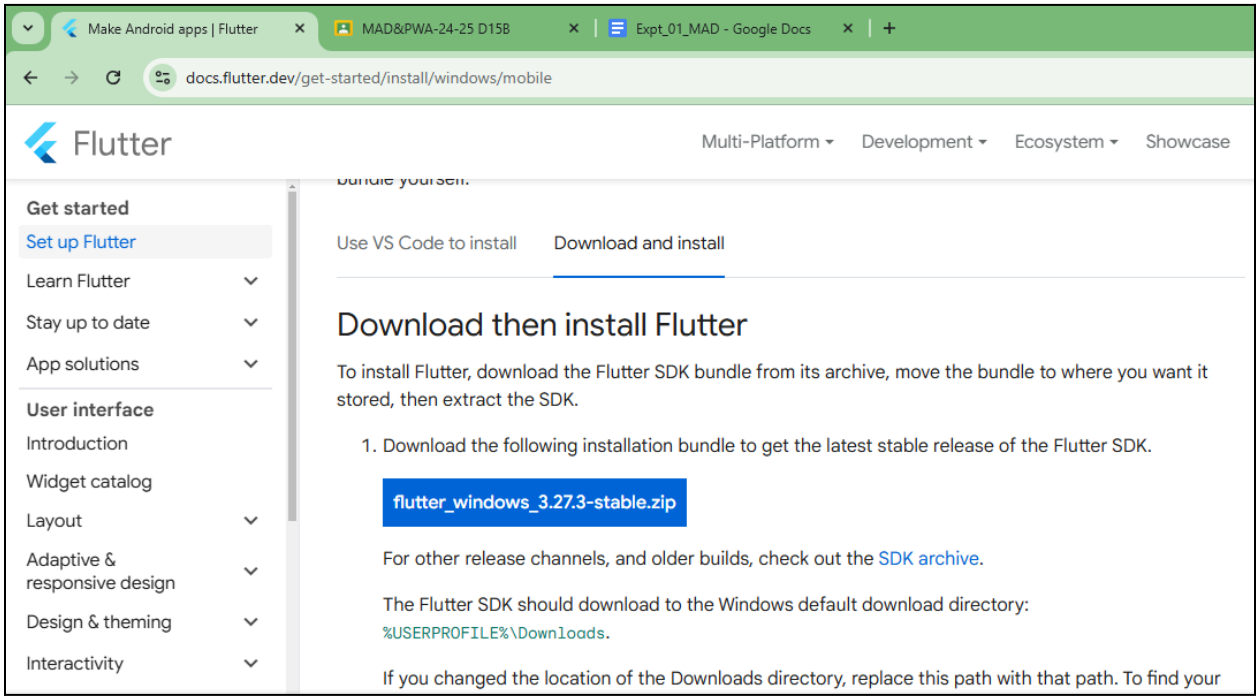
- Open Command Prompt and run: flutter doctor
- Ensure all tools are installed.

Install the Flutter SDK

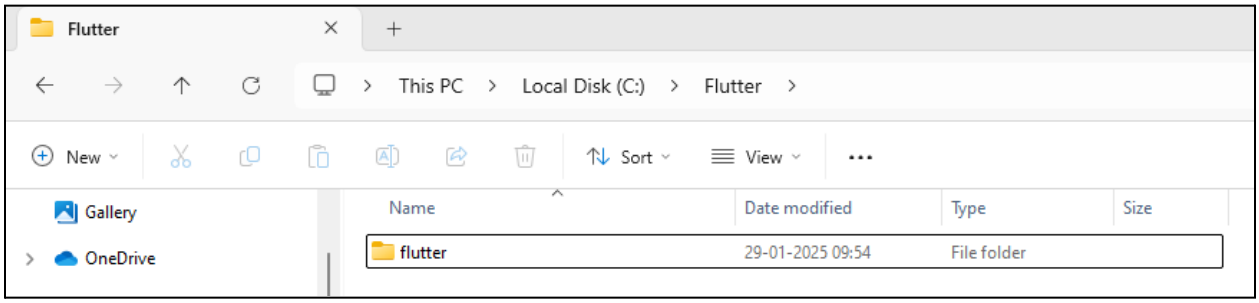
Step 1: Download the installation bundle of the Flutter Software Development Kit for windows. To download Flutter SDK, Go to its official website https://docs.flutter.dev/get-started/install , you will get the following screen.



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

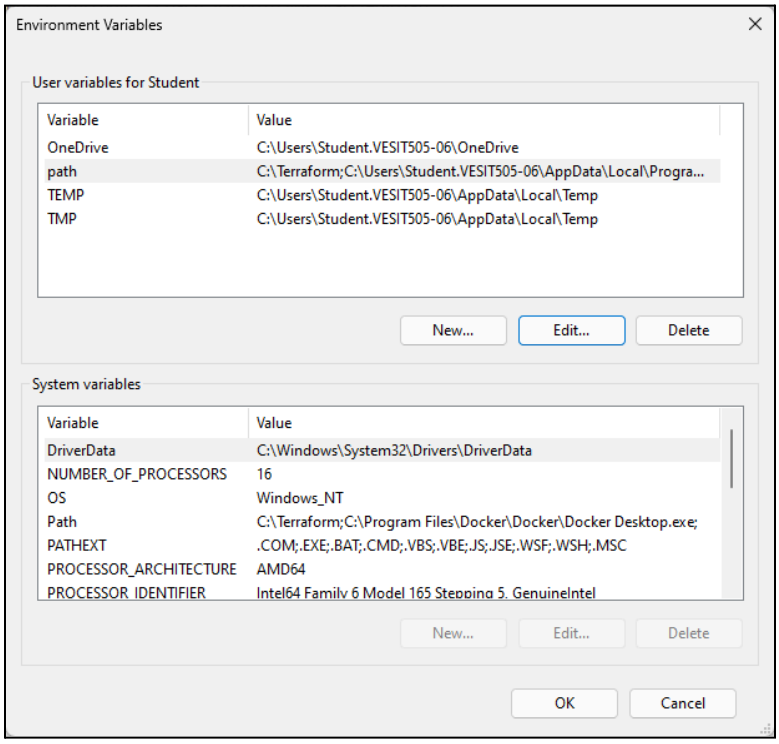


Step 3: When your download is complete, extract the zip file and place it in the desired installation folder or location, for example, C: /Flutter.



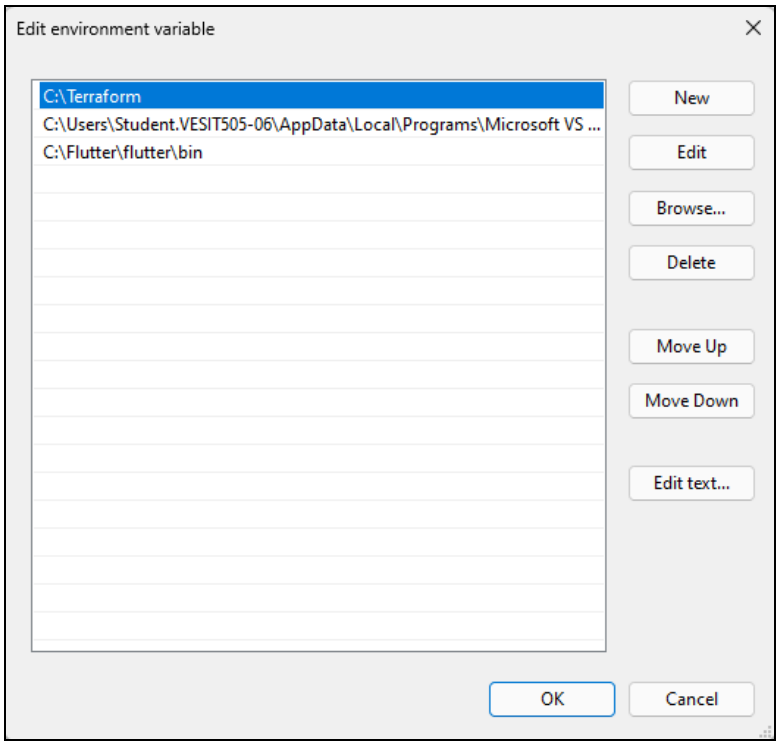
Step 4: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:

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

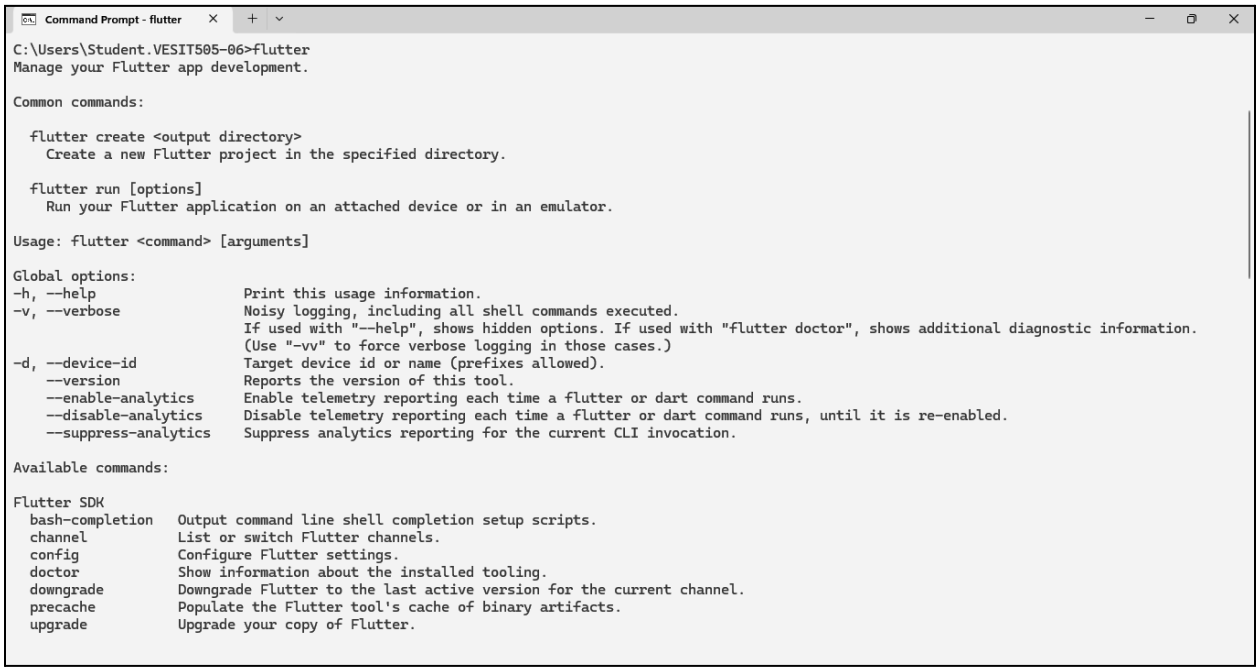


Step 4.2: Now, select path -> click on edit. The following screen appears

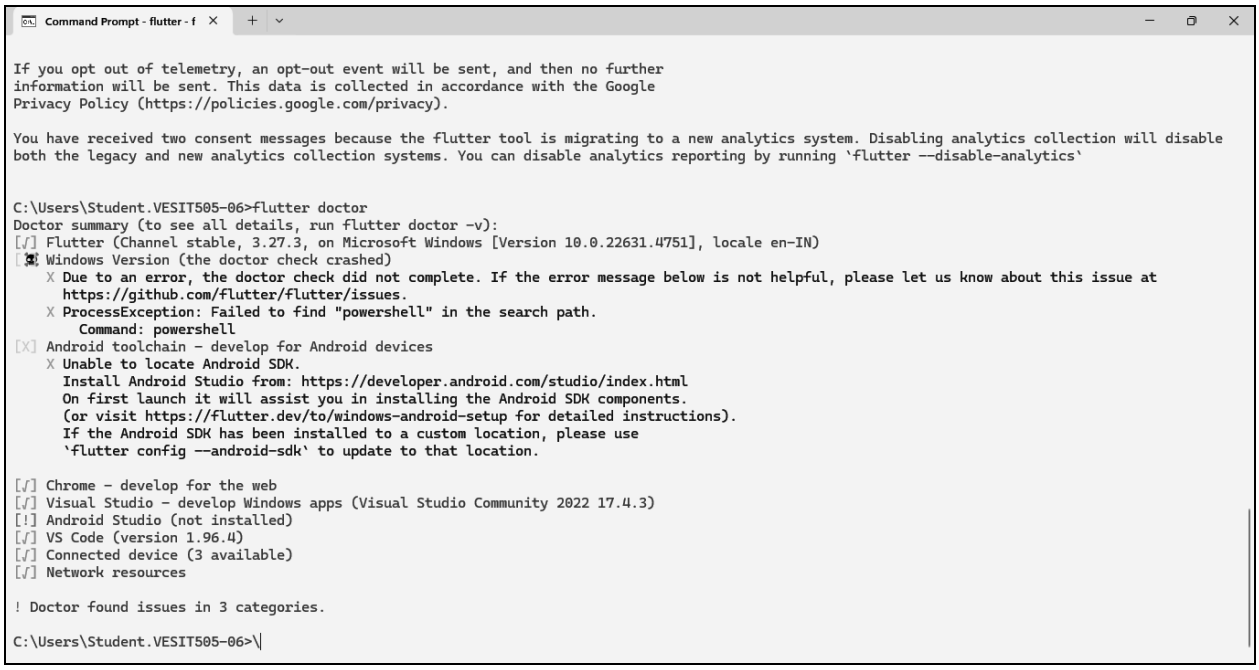
Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value -> ok -> ok -> ok.



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.

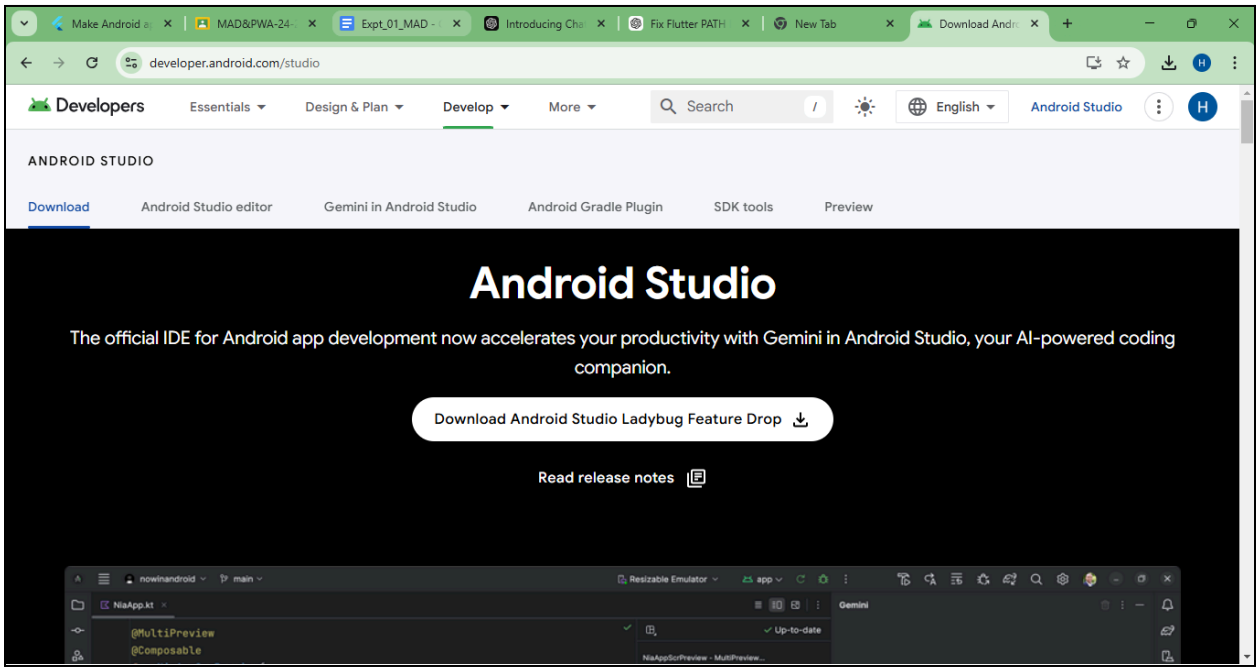


Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

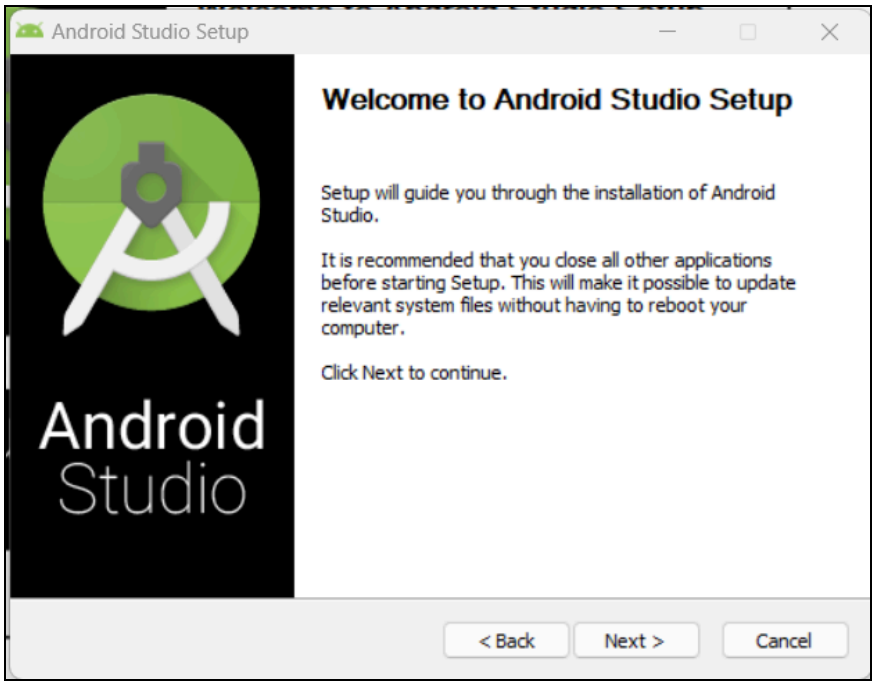


Step 7: Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.

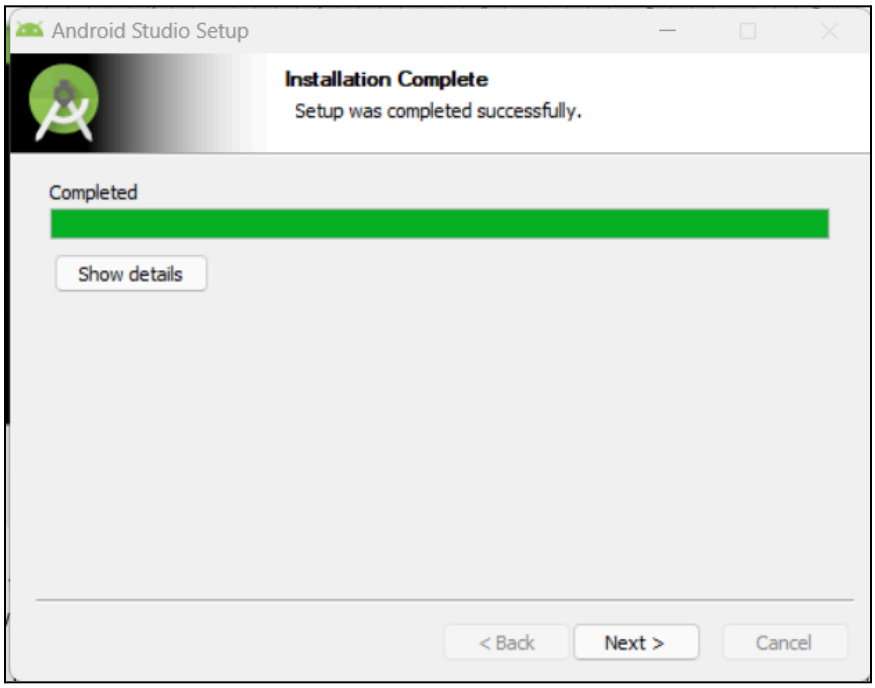
Step 7.1: Download the latest Android Studio executable or zip file from the official site.



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.



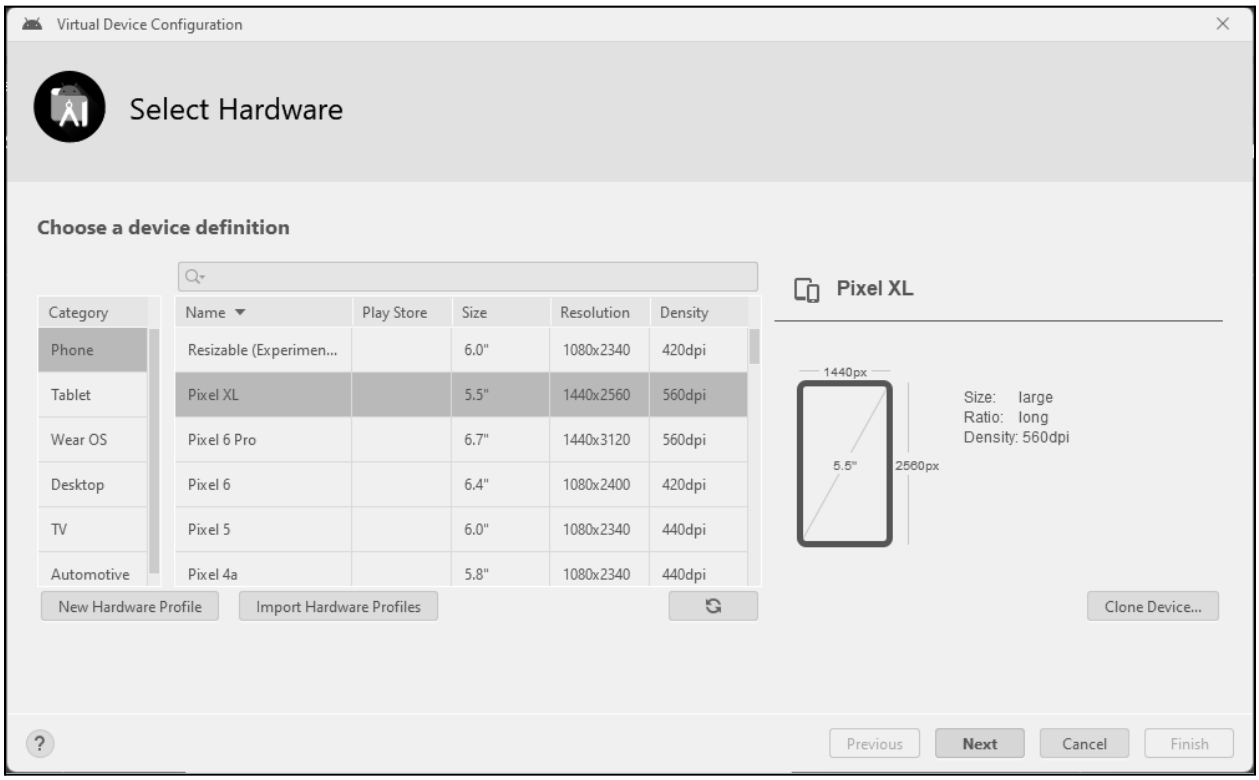
Step 7.4: In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to choose the 'Don't import Settings option' and click OK. It will start the Android Studio.

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

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

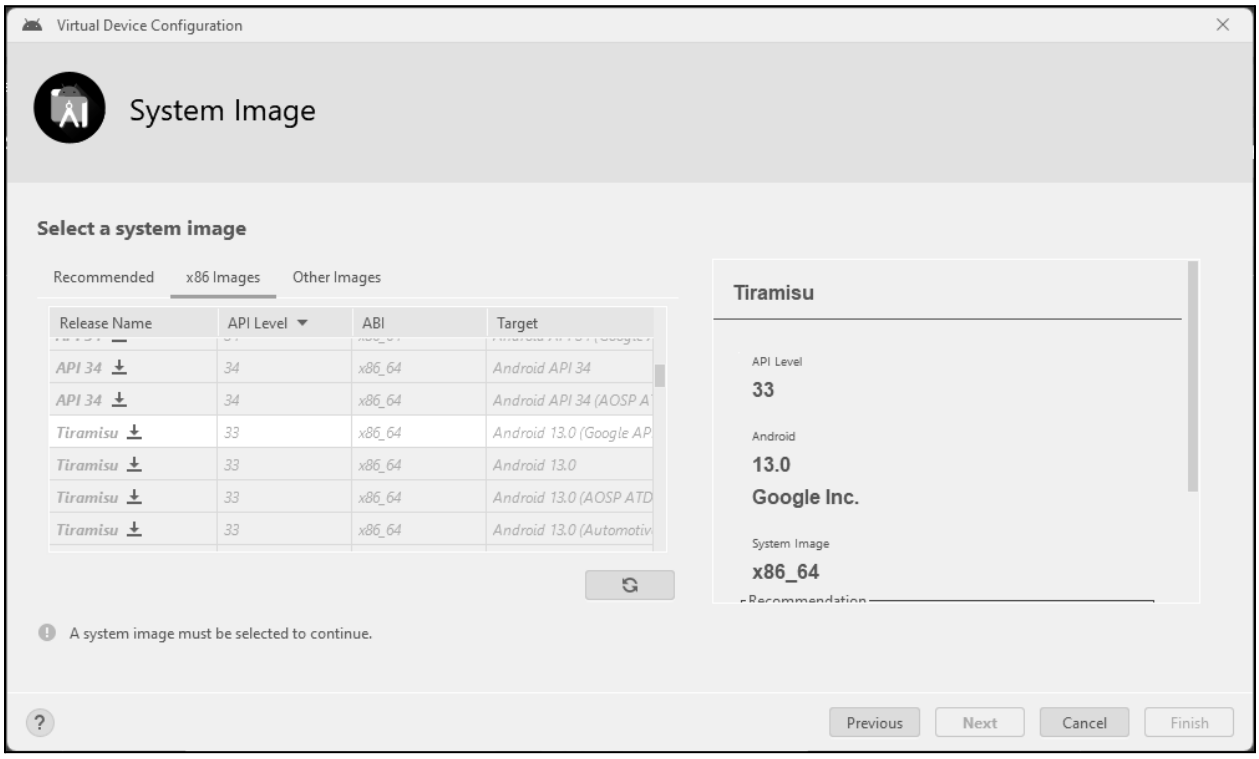
Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.

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

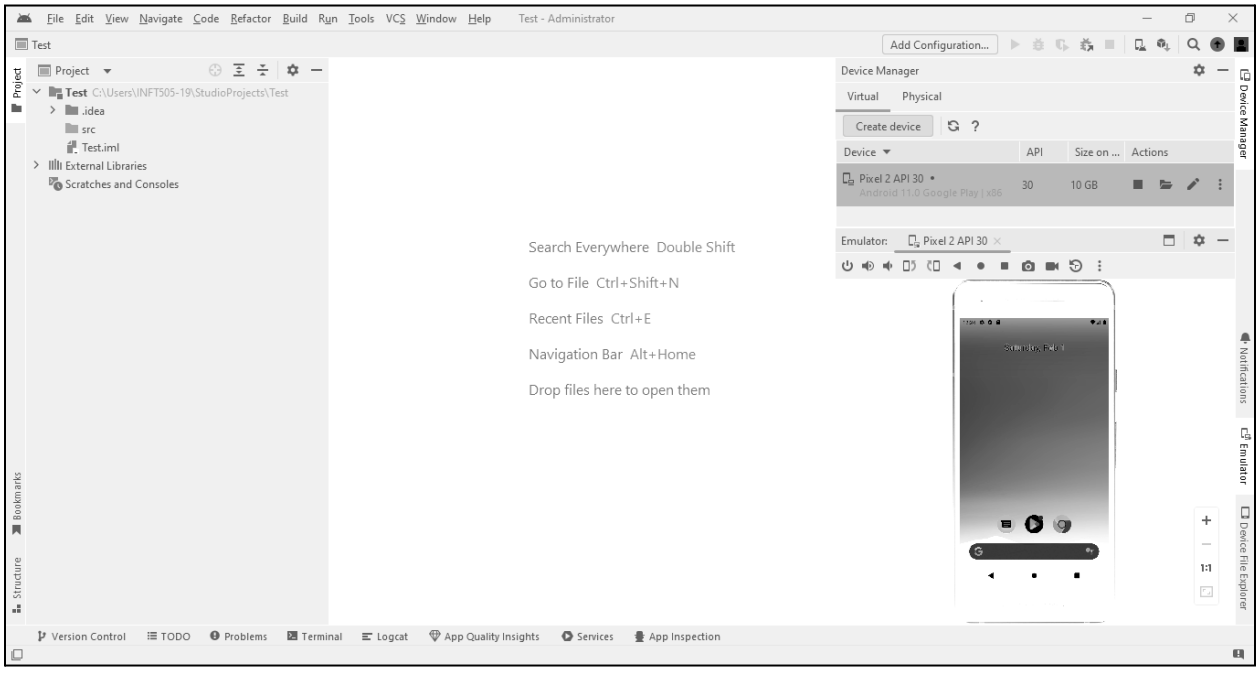


Step 8.3: Select the system image for the latest Android version and click on Next.

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

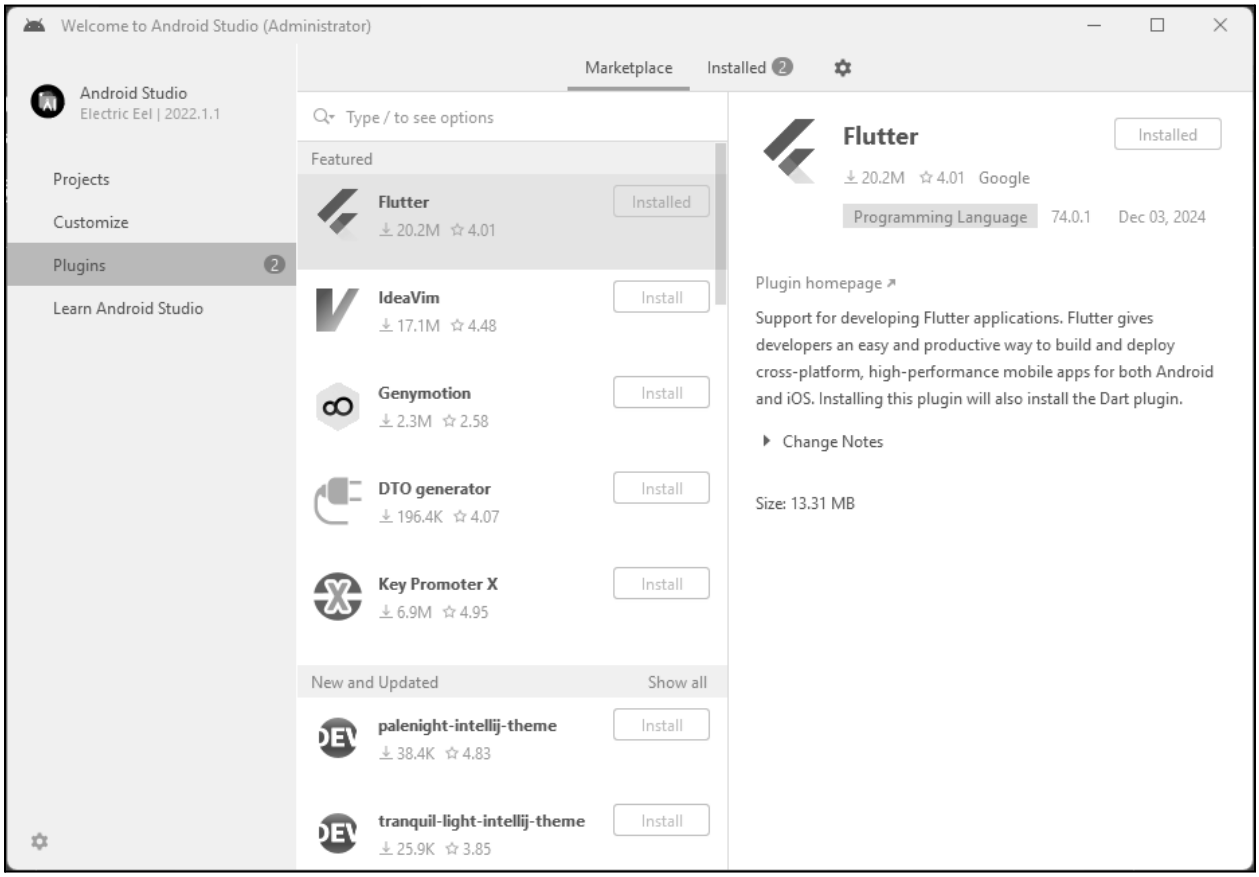


Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

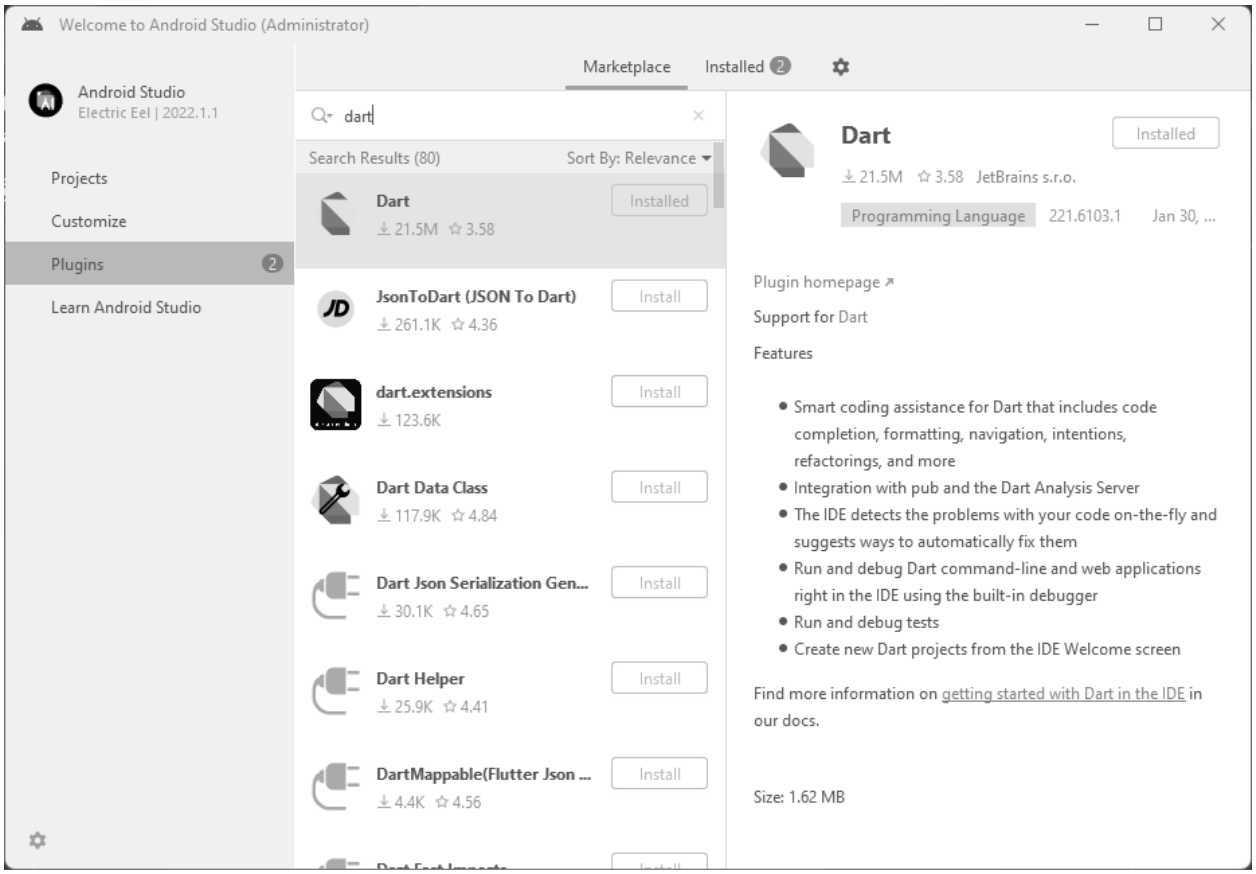


Step 9: Now, install the Flutter and Dart plugin for building Flutter applications in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.

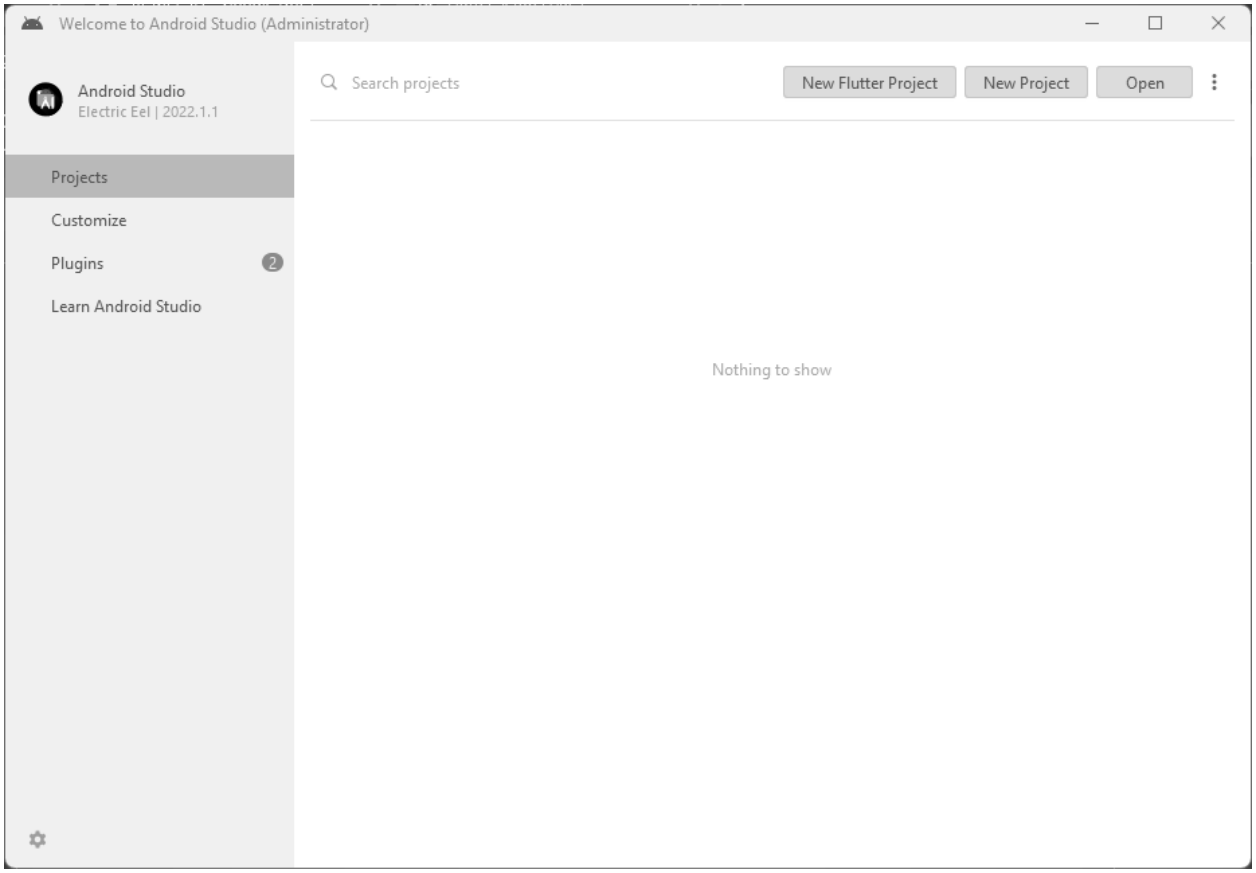
Step 9.1: Open the Android Studio and then go to File->Settings->Plugins.



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.



Conclusion:

This experiment was both interesting and frustrating at times. While installing Flutter was straightforward, setting up the Android Emulator gave me unexpected issues. The emulator kept crashing due to insufficient system memory. I had to tweak the AVD settings and allocate more RAM to fix it. This taught me that system requirements play a huge role in development.

Name : Vivek Gupta
Div : D15B
Roll No : 19

MPL Practical 02

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

Theory:

Flutter UI Design using Common Widgets

Flutter is a framework used for building mobile applications for Android and iOS. It allows developers to create beautiful and responsive user interfaces using widgets. Widgets are the basic building blocks of a Flutter app, and everything in Flutter is a widget, including buttons, text fields, and layout structures.

Common Widgets in Flutter

Flutter provides many built-in widgets that help in designing the UI of an application. Some of the commonly used widgets are:

1. Scaffold – Provides a basic structure with an app bar, body, and floating action button.
2. AppBar – Displays the title of the application and actions like buttons.
3. Text – Used to display text content in the app.
4. ListView – Helps in displaying a list of items in a scrollable manner.
5. Card – Used to display information inside a container with a shadow effect.
6. ElevatedButton – A button that performs an action when pressed.
7. TextField – Allows users to input text.
8. AlertDialog – Displays a pop-up dialog with options.

Implementation in Our Code

In our Sports Community Builder app, we have designed a simple user interface using common Flutter widgets. The main objective of this app is to allow users to form teams in their locality for instant play.

Features Implemented:

1. Displaying a List of Teams – We use ListView.builder() to show a list of available teams.
2. Adding a New Team – A TextField inside an AlertDialog allows users to enter a new team name.
3. Interactive Buttons – An ElevatedButton lets users create new teams.
4. Cards for Team Display – Each team name is displayed inside a Card for better presentation.
5. State Management – The list of teams is stored in a List<String> and updated dynamically using setState().

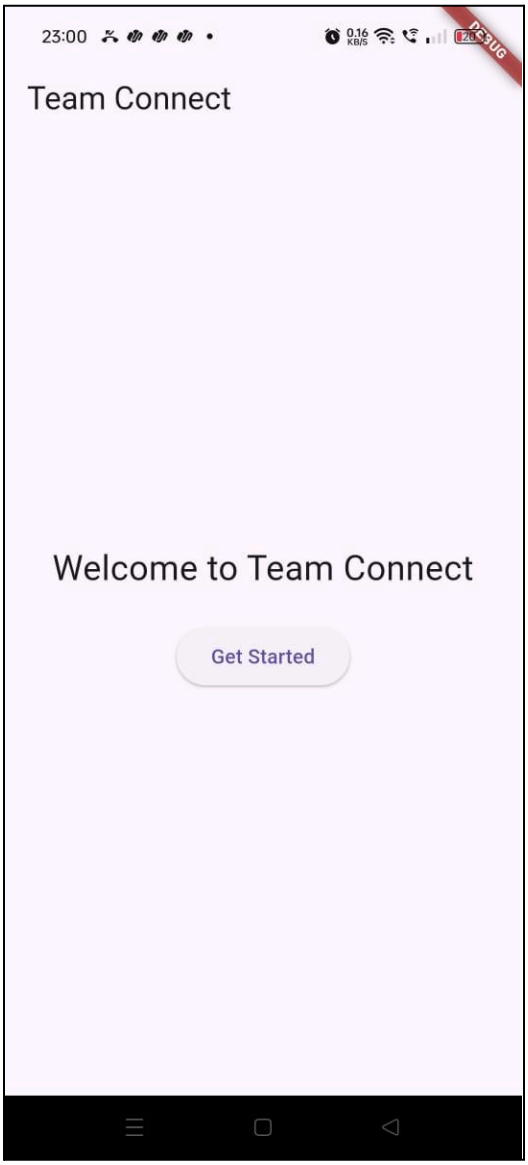
CODE:

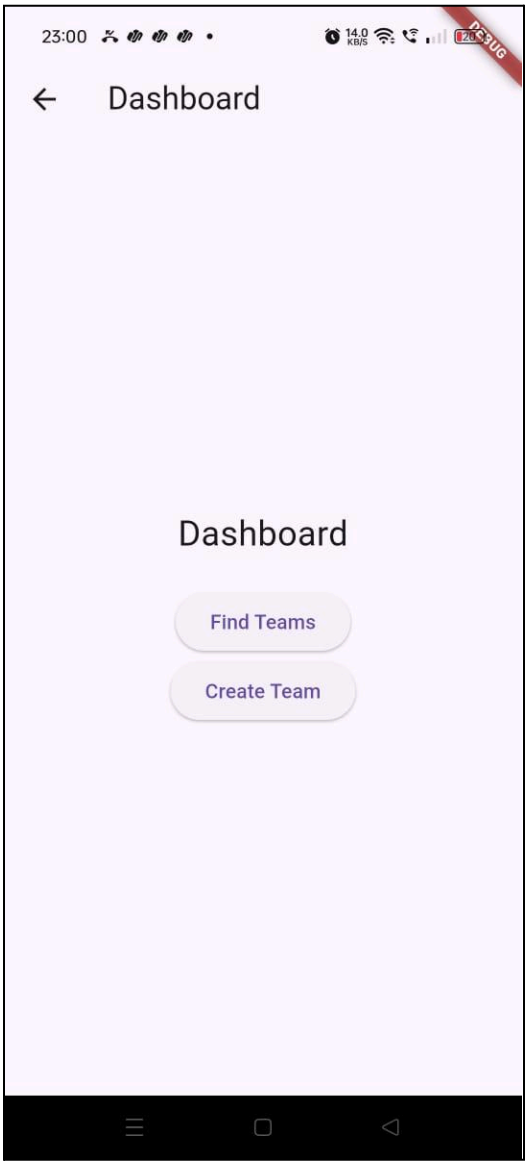
```
import 'package:flutter/material.dart';

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

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

 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   debugShowCheckedModeBanner: false,
   home: HomeScreen(),
  );
 }
}

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

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

class _HomeScreenState extends State<HomeScreen> {
 List<String> teams = ["Lions", "Tigers", "Eagles"];
 TextEditingController teamController = TextEditingController();

 void addTeam() {
  String newTeam = teamController.text.trim();
  if (newTeam.isNotEmpty) {
   setState(() {
    teams.add(newTeam);
   });
   teamController.clear();
   Navigator.pop(context);
  }
 }

 void openAddTeamDialog() {
  showDialog(
   context: context,
   builder: (context) {
    return AlertDialog(
     title: Text("Create a New Team"),
     content: TextField(
```

```
        controller: teamController,
        decoration: InputDecoration(hintText: "Enter team name"),
       ),
       actions: [
        TextButton(
         onPressed: () => Navigator.pop(context),
         child: Text("Cancel"),
        ),
        ElevatedButton(
         onPressed: addTeam,
         child: Text("Create"),
        ),
       ],
      );
    },
   );
}

@override
Widget build(BuildContext context) {
 return Scaffold(
  appBar: AppBar(title: Text("Sports Community Builder")),
  body: Padding(
   padding: EdgeInsets.all(10),
   child: Column(
    children: [
     Expanded(
      child: ListView.builder(
       itemCount: teams.length,
       itemBuilder: (context, index) {
        return Card(
         margin: EdgeInsets.symmetric(vertical: 5),
         child: ListTile(
          title: Text(teams[index]),
          trailing: Icon(Icons.sports_soccer),
         ),
        );
       },
      ),
     ),
     SizedBox(height: 10),
     ElevatedButton(
      onPressed: openAddTeamDialog,
      child: Text("Create a New Team"),
     ),
    ],
   ),
  ),
```

```
    );
  }
}
```

OUTPUT:





Conclusion:

In this experiment, we successfully designed the UI for our Sports Community Builder app using common Flutter widgets like ListView, Card, TextField, and ElevatedButton. Initially, we faced errors related to updating the list dynamically and handling the pop-up dialog, but we resolved them by using setState() for real-time UI updates and ensuring proper text input handling.

Name : Vivek Gupta
Div : D15B
Roll No : 19

MPL Practical 03

Aim: To learn how to add icons, images, and custom fonts in a Flutter application.

Theory:

In Flutter, icons, images, and fonts help in enhancing the UI and user experience.

1. Icons:
    - Flutter provides built-in icons through the Icons class.
    - Custom icons can be added using the Icon widget.
2. Images:
    - Images can be displayed using the Image.asset() method for local assets or Image.network() for online images.
    - Local images must be stored in the assets/ folder and declared in pubspec.yaml.
3. Fonts:
    - Custom fonts can be added by placing font files in the assets/fonts/ directory.
    - The font must be registered in pubspec.yaml and set using fontFamily in TextStyle.

Implementation in Our Code:

- Icons: Used Icons.sports_soccer in the AppBar and Icons.add_circle, Icons.search in buttons.
- Image: Displayed team_finder_logo.png using Image.asset().
- Custom Font: Added "Poppins" font in pubspec.yaml and applied it globally.

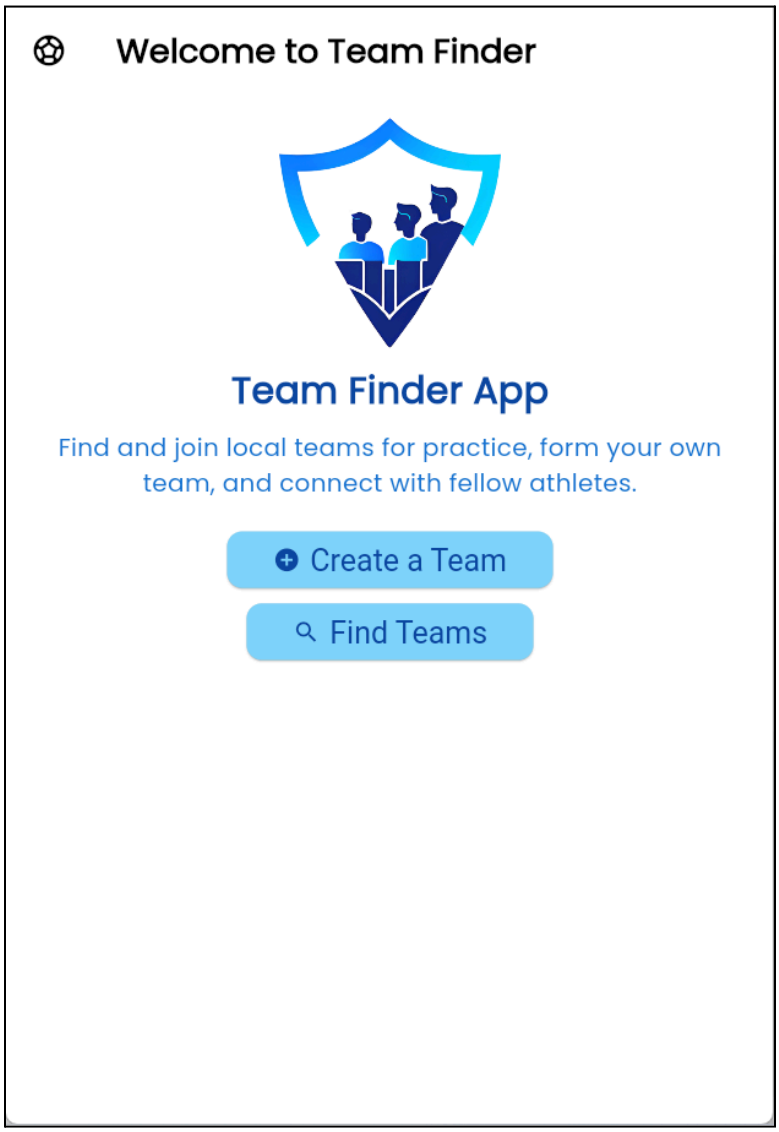
CODE:

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

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

class TeamFinderApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   debugShowCheckedModeBanner: false,
   title: 'Team Finder',
   theme: ThemeData(
    primarySwatch: Colors.blue,
    fontFamily: 'Poppins', // Custom font
    colorScheme: ColorScheme.fromSwatch().copyWith(
     primary: Colors.blue[900],
     secondary: Colors.blue[700],
```

```
      ),
      visualDensity: VisualDensity.adaptivePlatformDensity,
    ),
    home: HomePage(),
   );
  }
}

class HomePage extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('Welcome to Team Finder', style: TextStyle(fontWeight: FontWeight.bold)),
    leading: Icon(Icons.sports_soccer), // Icon in AppBar
   ),
   body: HomeContent(),
  );
 }
}

class HomeContent extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Padding(
   padding: EdgeInsets.all(16.0),
   child: Column(
    crossAxisAlignment: CrossAxisAlignment.center,
    children: [
     Center(
      child: Image.asset('assets/team_finder_logo.png', height: 150), // Display image
     ),
     SizedBox(height: 10),
     Text(
      'Team Finder App',
      style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold, color: Colors.blue[900]),
     ),
     SizedBox(height: 10),
     Text(
      'Find and join local teams for practice, form your own team, and connect with fellow athletes.',
      textAlign: TextAlign.center,
```

```
          style: TextStyle(fontSize: 16, color: Colors.blue[700]),
        ),
        SizedBox(height: 20),
        ElevatedButton.icon(
          onPressed: () {},
          icon: Icon(Icons.add_circle), // Button icon
          label: Text('Create a Team'),
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.lightBlue[200],
            padding: EdgeInsets.symmetric(horizontal: 30, vertical: 15),
            textStyle: TextStyle(fontSize: 20),
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(10),
            ),
          ),
        ),
        SizedBox(height: 10),
        ElevatedButton.icon(
          onPressed: () {},
          icon: Icon(Icons.search), // Button icon
          label: Text('Find Teams'),
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.lightBlue[200],
            padding: EdgeInsets.symmetric(horizontal: 30, vertical: 15),
            textStyle: TextStyle(fontSize: 20),
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(10),
            ),
          ),
        ),
      ],
    ),
   );
  }
}
```

OUTPUT:



Conclusion:

In this experiment, we successfully implemented icons, images, and custom fonts to enhance the UI of our Flutter app. Initially, we faced an issue where the custom font was not applying because it was not properly declared in pubspec.yaml, but we resolved it by correctly specifying the font path and restarting the app.

Name : Vivek Gupta
Div : D15B
Roll No : 19

MPL Practical 04

Aim: To create an interactive Form using form widget

Theory:

Forms are an essential part of mobile applications, allowing users to input and submit data. In Flutter, the Form widget provides an organized way to collect and validate user input. It works with TextFormField and DropdownButtonFormField to ensure structured data entry. The GlobalKey<FormState> is used to manage form validation and submission.

Implementation in Code
In our implementation, we created an interactive form to allow users to create a sports team. The key components include:

- Form Widget: Wraps input fields for validation.
- TextFormField: Captures the team name with validation.
- DropdownButtonFormField: Allows users to select a sport.
- Validation: Ensures fields are not left empty.
- Snackbar Feedback: Displays a success message when the form is submitted.

CODE:

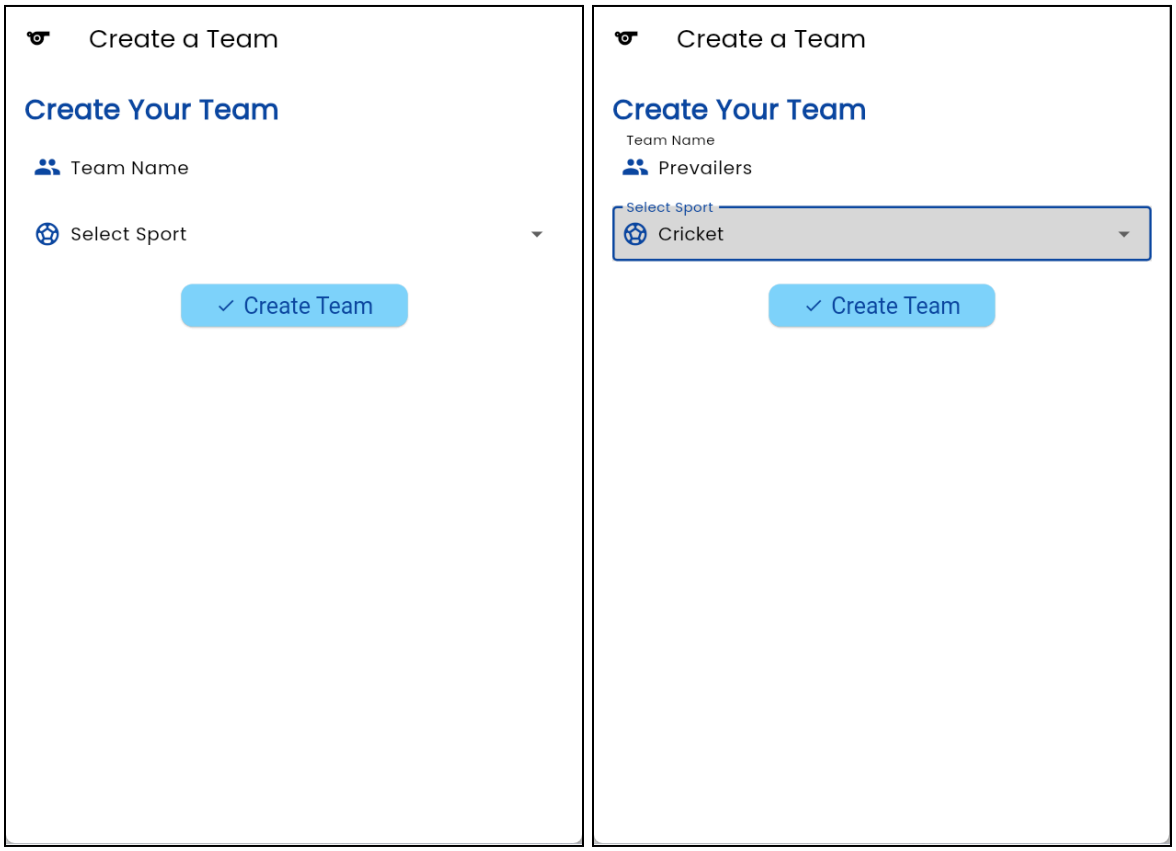
```
import 'package:flutter/material.dart';

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

class TeamFinderApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   debugShowCheckedModeBanner: false,
   title: 'Team Finder',
   theme: ThemeData(
    primarySwatch: Colors.blue,
    fontFamily: 'Poppins', // Custom font
    colorScheme: ColorScheme.fromSwatch().copyWith(
     primary: Colors.blue[900],
     secondary: Colors.blue[700],
    ),
    visualDensity: VisualDensity.adaptivePlatformDensity,
   ),
```

```
    home: CreateTeamScreen(), // Directly opening the form screen
   );
  }
}

class CreateTeamScreen extends StatefulWidget {
 @override
 _CreateTeamScreenState createState() => _CreateTeamScreenState();
}

class _CreateTeamScreenState extends State<CreateTeamScreen> {
 final _formKey = GlobalKey<FormState>(); // Form key for validation
 String? _teamName;
 String? _selectedSport;
 List<String> sports = ['Football', 'Basketball', 'Cricket', 'Tennis'];

 void _submitForm() {
  if (_formKey.currentState!.validate()) {
   _formKey.currentState!.save();
   ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text('Team $_teamName for $_selectedSport created!')),
   );
  }
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('Create a Team'),
    leading: Icon(Icons.sports), // Icon in AppBar
   ),
   body: Padding(
    padding: EdgeInsets.all(16.0),
    child: Form(
     key: _formKey,
     child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
       Text(
        'Create Your Team',
```

```
  style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold, color: Colors.blue[900]),
),
SizedBox(height: 10),
TextFormField(
  decoration: InputDecoration(
    labelText: 'Team Name',
    prefixIcon: Icon(Icons.group, color: Colors.blue[900]),
    border: OutlineInputBorder(),
  ),
  validator: (value) {
    if (value == null || value.isEmpty) return 'Please enter a team name';
    return null;
  },
  onSaved: (value) => _teamName = value,
),
SizedBox(height: 10),
DropdownButtonFormField<String>(
  decoration: InputDecoration(
    labelText: 'Select Sport',
    prefixIcon: Icon(Icons.sports_soccer, color: Colors.blue[900]),
    border: OutlineInputBorder(),
  ),
  items: sports.map((sport) {
    return DropdownMenuItem(
      value: sport,
      child: Text(sport),
    );
  }).toList(),
  validator: (value) => value == null ? 'Please select a sport' : null,
  onChanged: (value) {
    setState(() {
      _selectedSport = value;
    });
  },
),
SizedBox(height: 20),
Center(
  child: ElevatedButton.icon(
    onPressed: _submitForm,
    icon: Icon(Icons.check),
    label: Text('Create Team'),
```

```
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.lightBlue[200],
              padding: EdgeInsets.symmetric(horizontal: 30, vertical: 15),
              textStyle: TextStyle(fontSize: 20),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(10),
              ),
            ),
          ),
        ),
      ],
    ),
  ),
 ),
);
 }
}
```

OUTPUT:



Conclusion:

In this experiment, we successfully implemented an interactive form using the Form widget, allowing users to create a sports team with validation and user feedback. Initially, we faced issues with form validation not triggering properly, but we resolved it by ensuring that GlobalKey<FormState> was correctly linked and validation checks were properly implemented.

Name : Vivek Gupta
Div : D15B
Roll No : 19

MPL Practical 05

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

Theory:

Introduction
Navigation in Flutter allows users to switch between different screens (pages). It is implemented using routes and the Navigator widget. Routing can be defined using MaterialPageRoute, named routes, or the onGenerateRoute method. Gestures, such as taps and swipes, enhance user interaction using Flutter's GestureDetector.

Implementation in Our Code

- Named Routes: We defined named routes (/, /createTeam, /findTeams) in MaterialApp for structured navigation.
- Navigation: Navigator.pushNamed(context, routeName) is used to move between screens, and BottomNavigationBar provides seamless switching.
- Gestures: GestureDetector is used to handle taps on buttons (Create a Team, Find Teams) and list items in the TeamScreen.
- User Interaction: The navigation system ensures smooth movement across the Home Page, Create Team Page, and Find Teams Page with a bottom navigation bar for ease of access.

CODE:

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

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

class TeamFinderApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   debugShowCheckedModeBanner: false,
   title: 'Team Finder',
   theme: ThemeData(
    primarySwatch: Colors.blue,
    fontFamily: 'Poppins',
    colorScheme: ColorScheme.fromSwatch().copyWith(
     primary: Colors.blue[900],
     secondary: Colors.blue[700],
    ),
```

```
      visualDensity: VisualDensity.adaptivePlatformDensity,
    ),
    initialRoute: '/',
    routes: {
     '/': (context) => HomePage(),
     '/createTeam': (context) => CreateTeamScreen(),
     '/findTeams': (context) => TeamScreen(),
     '/chat': (context) => ChatScreen(),
    },
   );
  }
}

class HomePage extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(title: Text('Welcome to Team Finder'),
   leading: Icon(Icons.sports_soccer),
   ),
   body: HomeContent(),
   bottomNavigationBar: NavBar(currentIndex: 0),
  );
 }
}

class HomeContent extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Padding(
   padding: EdgeInsets.all(16.0),
   child: Column(
    crossAxisAlignment: CrossAxisAlignment.center,
    children: [
     Center(
      child: Image.asset('assets/team_finder_logo.png', height: 150), // Display image
     ),
     Text(
      'Team Finder App',
      style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold, color: Colors.blue[900]),
     ),
```

```
SizedBox(height: 10),
Text(
 'Find and join local teams for practice, form your own team, and connect with fellow athletes.',
 textAlign: TextAlign.center,
 style: TextStyle(fontSize: 16, color: Colors.blue[700]),
),
SizedBox(height: 20),
GestureDetector(
 onTap: () {
  Navigator.pushNamed(context, '/createTeam');
 },
 child: Container(
  padding: EdgeInsets.all(15),
  decoration: BoxDecoration(
   color: Colors.lightBlue[200],
   borderRadius: BorderRadius.circular(10),
  ),
  child: Row(
   mainAxisAlignment: MainAxisAlignment.center,
   children: [
    Icon(Icons.add, color: Colors.white),
    SizedBox(width: 10),
    Text(
     'Create a Team',
     style: TextStyle(fontSize: 20, color: Colors.white),
    ),
   ],
  ),
 ),
),
SizedBox(height: 10),
GestureDetector(
 onTap: () {
  Navigator.pushNamed(context, '/findTeams');
 },
 child: Container(
  padding: EdgeInsets.all(15),
  decoration: BoxDecoration(
   color: Colors.lightBlue[200],
   borderRadius: BorderRadius.circular(10),
  ),
```

```
            child: Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Icon(Icons.search, color: Colors.white),
                SizedBox(width: 10),
                Text(
                  'Find Teams',
                  style: TextStyle(fontSize: 20, color: Colors.white),
                ),
              ],
            ),
          ),
        ),
      ],
    ),
  );
 }
}

class CreateTeamScreen extends StatelessWidget {
 final _formKey = GlobalKey<FormState>();
 String? _teamName;
 String? _selectedSport;
 List<String> sports = ['Football', 'Basketball', 'Cricket', 'Tennis'];

 void _submitForm(BuildContext context) {
  if (_formKey.currentState!.validate()) {
   _formKey.currentState!.save();
   ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text('Team $_teamName for $_selectedSport created!')),
   );
  }
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(title: Text('Create a Team')),
   body: Padding(
    padding: EdgeInsets.all(16.0),
    child: Form(
```

```
key: _formKey,
child: Column(
 crossAxisAlignment: CrossAxisAlignment.start,
 children: [
  Text(
   'Create Your Team',
   style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold, color: Colors.blue[900]),
  ),
  SizedBox(height: 10),
  TextFormField(
   decoration: InputDecoration(
    labelText: 'Team Name',
    prefixIcon: Icon(Icons.group, color: Colors.blue[900]),
    border: OutlineInputBorder(),
   ),
   validator: (value) => value!.isEmpty ? 'Please enter a team name' : null,
   onSaved: (value) => _teamName = value,
  ),
  SizedBox(height: 10),
  DropdownButtonFormField<String>(
   decoration: InputDecoration(
    labelText: 'Select Sport',
    prefixIcon: Icon(Icons.sports_soccer, color: Colors.blue[900]),
    border: OutlineInputBorder(),
   ),
   items: sports.map((sport) {
    return DropdownMenuItem(
     value: sport,
     child: Text(sport),
    );
   }).toList(),
   validator: (value) => value == null ? 'Please select a sport' : null,
   onChanged: (value) => _selectedSport = value,
  ),
  SizedBox(height: 20),
  Center(
   child: ElevatedButton(
    onPressed: () => _submitForm(context),
    child: Text('Create Team'),
    style: ElevatedButton.styleFrom(
     backgroundColor: Colors.lightBlue[200],
```

```
            ),
          ),
        ),
      ],
    ),
  ),
 ),
 bottomNavigationBar: NavBar(currentIndex: 1),
 );
 }
}

class TeamScreen extends StatefulWidget {
 @override
 _TeamScreenState createState() => _TeamScreenState();
}

class _TeamScreenState extends State<TeamScreen> {
 final List<Map<String, String>> teams = [
  {'name': 'Warriors FC', 'sport': 'Football'},
  {'name': 'Thunder Hoops', 'sport': 'Basketball'},
  {'name': 'Strikers Club', 'sport': 'Cricket'}
 ];

 final Set<String> joinedTeams = {};

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(title: Text('Find Teams')),
   body: ListView.builder(
    itemCount: teams.length,
    itemBuilder: (context, index) {
     String teamName = teams[index]['name']!;
     String sport = teams[index]['sport']!;
     bool isJoined = joinedTeams.contains(teamName);

     return Card(
      margin: EdgeInsets.all(10),
      child: ListTile(
```

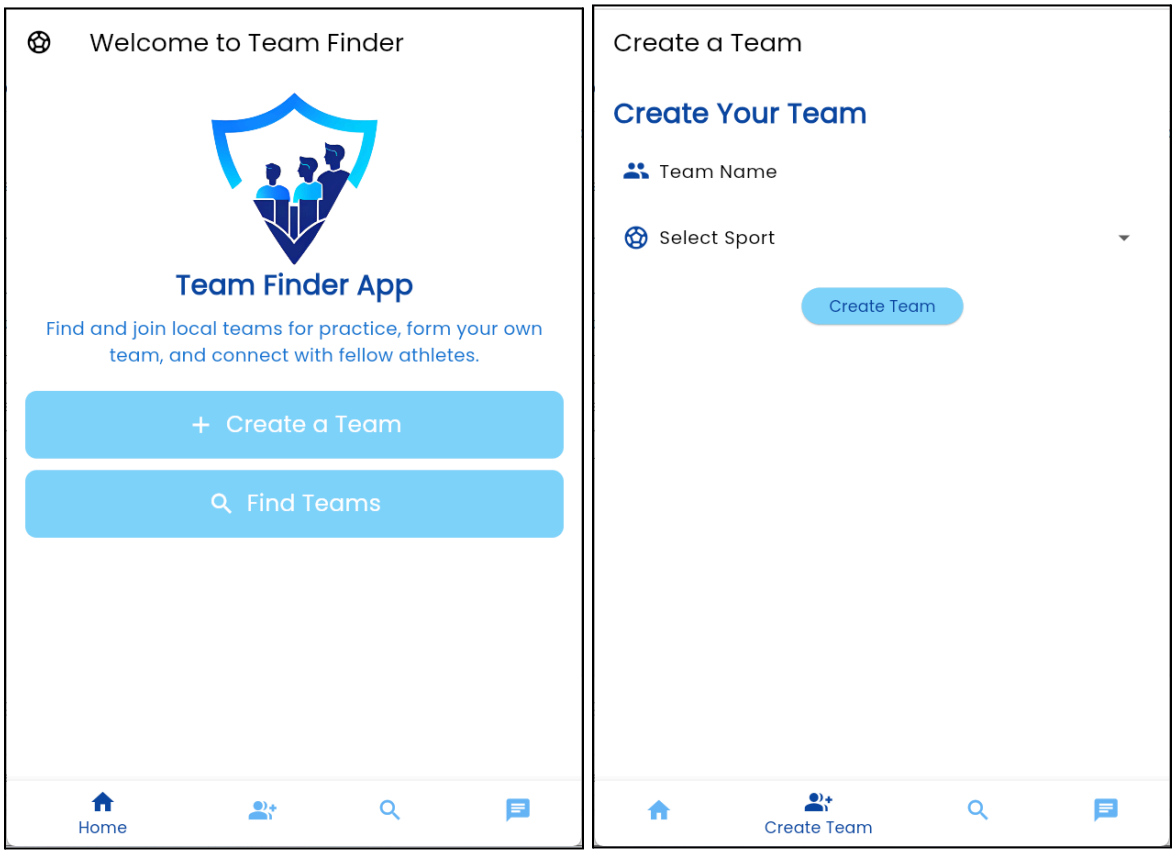
```
                    title: Text(teamName, style: TextStyle(color: Colors.blue[900], fontWeight:
FontWeight.bold)),
          subtitle: Text(sport, style: TextStyle(color: Colors.blue[700])),
          trailing: ElevatedButton(
           onPressed: () {
            setState(() {
             if (isJoined) {
              joinedTeams.remove(teamName);
             } else {
              joinedTeams.add(teamName);
             }
            });
           },
           style: ElevatedButton.styleFrom(
            backgroundColor: isJoined ? Colors.blue[900] : Colors.blue[300],
            shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(20)),
           ),
           child: Text(
            isJoined ? 'Leave' : 'Join',
            style: TextStyle(color: Colors.white),
           ),
          ),
         ),
        );
       },
      ),
      bottomNavigationBar: NavBar(currentIndex: 2),
    );
  }
}

// Chat Screen
class ChatScreen extends StatefulWidget {
 @override
 _ChatScreenState createState() => _ChatScreenState();
}

class _ChatScreenState extends State<ChatScreen> {
 final List<String> messages = [];
 final TextEditingController _controller = TextEditingController();
```

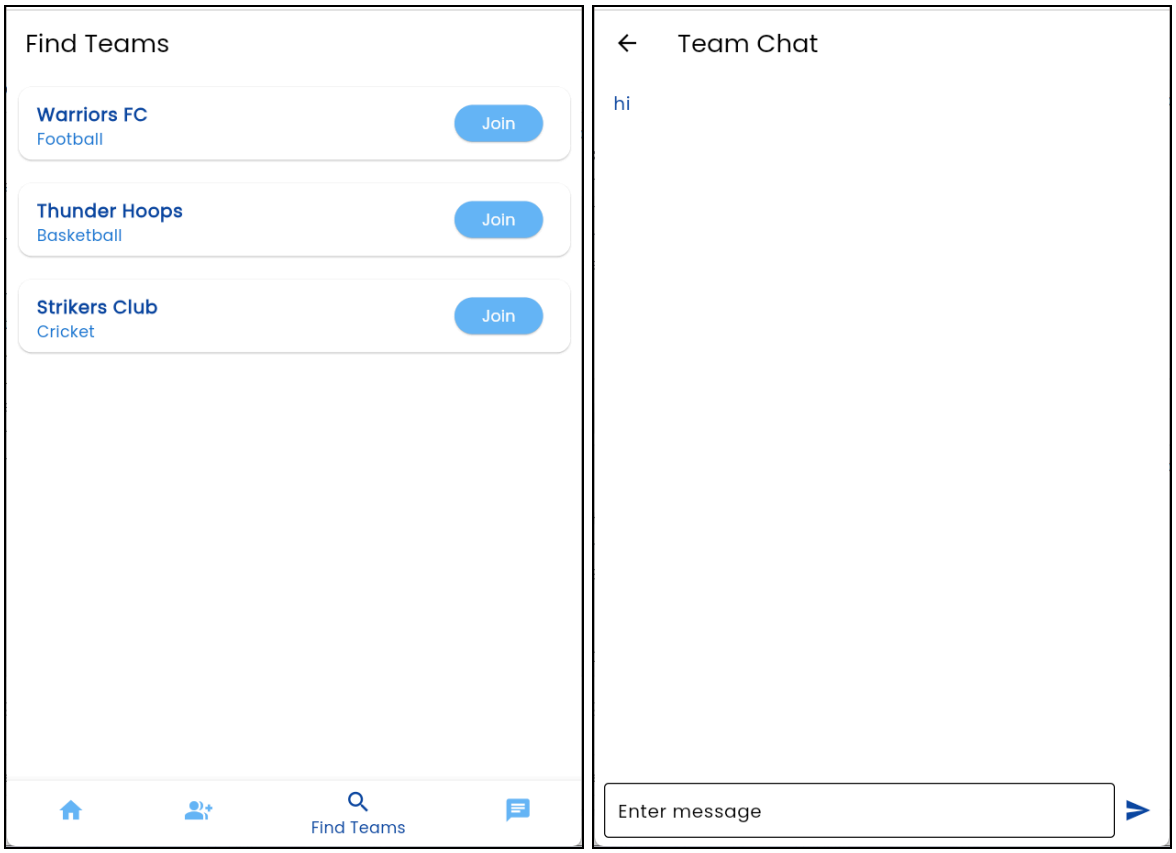
```
void _sendMessage() {
 if (_controller.text.isNotEmpty) {
  setState(() {
   messages.add(_controller.text);
  });
  _controller.clear();
 }
}

@override
Widget build(BuildContext context) {
 return Scaffold(
  appBar: AppBar(
   title: Text("Team Chat"),
   leading: IconButton(
    icon: Icon(Icons.arrow_back, color: Colors.black), // Back button
    onPressed: () {
     Navigator.pop(context); // Navigate back
    },
   ),
  ),
  body: Column(
   children: [
    Expanded(
     child: ListView.builder(
      itemCount: messages.length,
      itemBuilder: (context, index) => ListTile(
       title: Text(messages[index], style: TextStyle(color: Colors.blue[900])),
      ),
     ),
    ),
    Padding(
     padding: const EdgeInsets.all(8.0),
     child: Row(
      children: [
       Expanded(
        child: TextField(
         controller: _controller,
         decoration: InputDecoration(
          hintText: "Enter message",
          border: OutlineInputBorder(),
```

```
              ),
            ),
          ),
          IconButton(
            icon: Icon(Icons.send, color: Colors.blue[900]),
            onPressed: _sendMessage,
          ),
        ],
      ),
    ),
  ],
 ),
);
}
}

class NavBar extends StatelessWidget {
 final int currentIndex;
 NavBar({required this.currentIndex});

 @override
 Widget build(BuildContext context) {
  return BottomNavigationBar(
   currentIndex: currentIndex,
   selectedItemColor: Colors.blue[900],
   unselectedItemColor: Colors.blue[300],
   onTap: (index) {
    if (index == 0) {
     Navigator.pushReplacementNamed(context, '/');
    } else if (index == 1) {
     Navigator.pushReplacementNamed(context, '/createTeam');
    } else if (index == 2) {
     Navigator.pushReplacementNamed(context, '/findTeams');
    } else if (index == 3) { // Chat navigation added
     Navigator.pushReplacementNamed(context, '/chat');
    }
   },
   items: [
    BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Home'),
    BottomNavigationBarItem(icon: Icon(Icons.group_add), label: 'Create Team'),
    BottomNavigationBarItem(icon: Icon(Icons.search), label: 'Find Teams'),
```

```
      BottomNavigationBarItem(icon: Icon(Icons.chat), label: "Chat"), // Added Chat button
    ],
   );
  }
}
```

OUTPUT:





Conclusion:

In this experiment, we successfully implemented navigation using named routes and integrated gestures for user interaction in the **Team Finder** app. Initially, we faced issues with incorrect route navigation and unresponsive gestures, which we resolved by debugging route names and ensuring GestureDetector was properly wrapped around interactive widgets.

Name : Vivek Gupta
Div : D15B
Roll No : 19

MPL Practical 06

Aim: To integrate Firebase Authentication in a Flutter app.

Theory:

Introduction
Firebase provides a backend-as-a-service (BaaS), making it easy to handle user authentication and real-time database operations. Firebase Authentication allows secure user login and signup using email/password. With Firebase, we can:

- Register users and store their credentials.
- Authenticate users securely using FirebaseAuth.
- Verify emails to ensure valid registrations.

Flutter interacts with Firebase using the firebase_auth package, which provides methods for signup, login, logout, and authentication state management.

Implementation in Our Code

1. Signup Page (signup_screen.dart)
    - Allows users to create an account using email and password.
    - Sends an email verification after signup.
    - Displays messages to inform users about signup success or errors.

2. Login Page (login_screen.dart)
    - Allows users to log in if their credentials are correct.
    - Checks if the email is verified before granting access.
    - Displays proper error messages in case of invalid credentials.

3. Authentication Service (auth_service.dart)
    - Manages signup, login, logout, and email verification.
    - Uses Firebase's authentication API for secure user management.

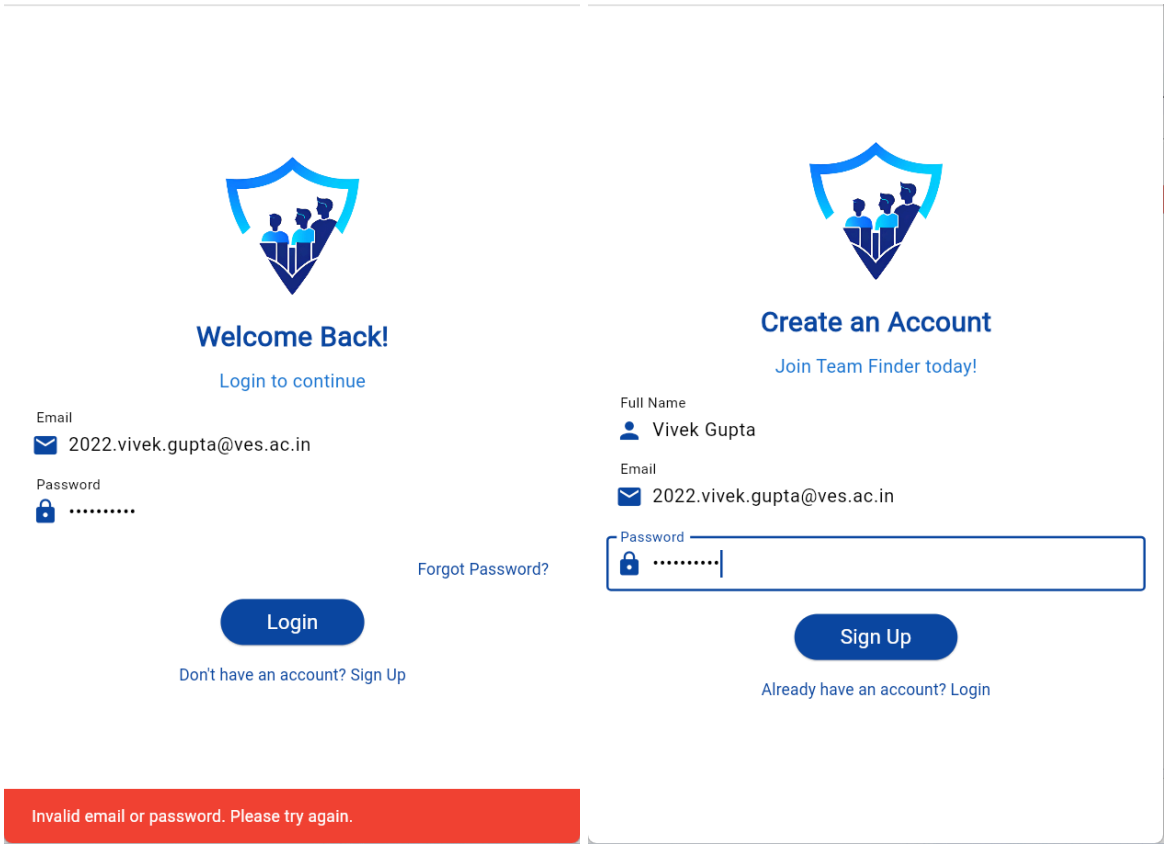
auth_service.dart

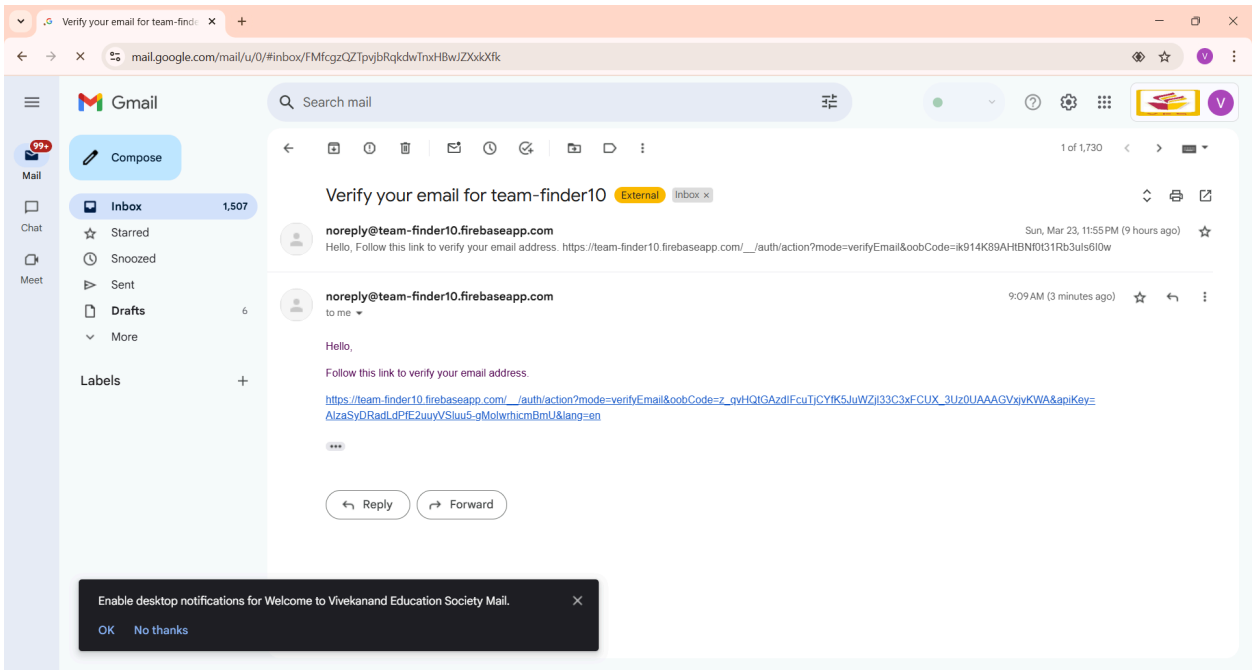
```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:google_sign_in/google_sign_in.dart';

class AuthService {
 final FirebaseAuth _auth = FirebaseAuth.instance;
 final GoogleSignIn _googleSignIn = GoogleSignIn(
 );

 // Sign Up with Email & Password + Store User in Firestore
```

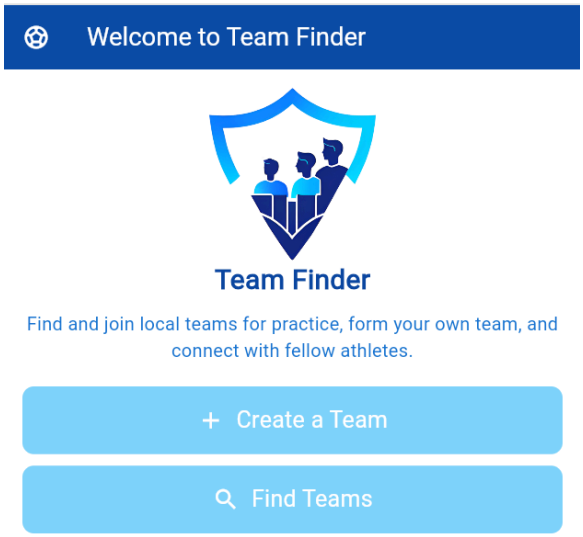
```
Future<User?> signUp(String email, String password, String fullName) async {
 try {
  UserCredential userCredential = await _auth.createUserWithEmailAndPassword(
   email: email,
   password: password,
  );

  User? user = userCredential.user;
  if (user != null) {
   await FirebaseFirestore.instance.collection('users').doc(user.uid).set({
    'uid': user.uid,
    'fullName': fullName,
    'email': email,
    'joinedTeams': [],
   });

   await user.sendEmailVerification();
   print("Verification email sent to: ${user.email}");
  }
  return user;
 } catch (e) {
  print("Sign Up Error: $e");
  return null;
 }
}

// Login with Email & Password (Only If Verified)
Future<User?> signIn(String email, String password) async {
 try {
  UserCredential userCredential = await _auth.signInWithEmailAndPassword(
   email: email,
   password: password,
  );

  if (!userCredential.user!.emailVerified) {
   print("Email not verified. Please check your inbox.");
   return null;
  }

  print("Login Successful!");
  return userCredential.user;
```

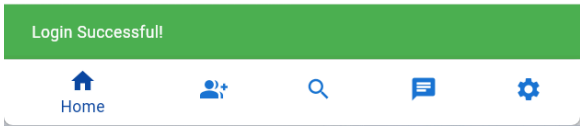
```
  } catch (e) {
    print("Login Error: $e");
    return null;
  }
 }

 // Get current user
 User? getCurrentUser() {
  return _auth.currentUser;
 }

 // Reset Password
 Future<bool> resetPassword(String email) async {
  try {
    await _auth.sendPasswordResetEmail(email: email);
    return true;
  } catch (e) {
    print("Password Reset Error: $e");
    return false;
  }
 }

 // Google Sign-In + Store User in Firestore if New
 Future<User?> signInWithGoogle() async {
  try {
    final GoogleSignInAccount? googleUser = await _googleSignIn.signIn();
    if (googleUser == null) return null;

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

    UserCredential userCredential = await _auth.signInWithCredential(credential);
    User? user = userCredential.user;

    if (user != null) {
      // Check if user already exists in Firestore
                                                DocumentSnapshot userDoc = await
FirebaseFirestore.instance.collection('users').doc(user.uid).get();
```

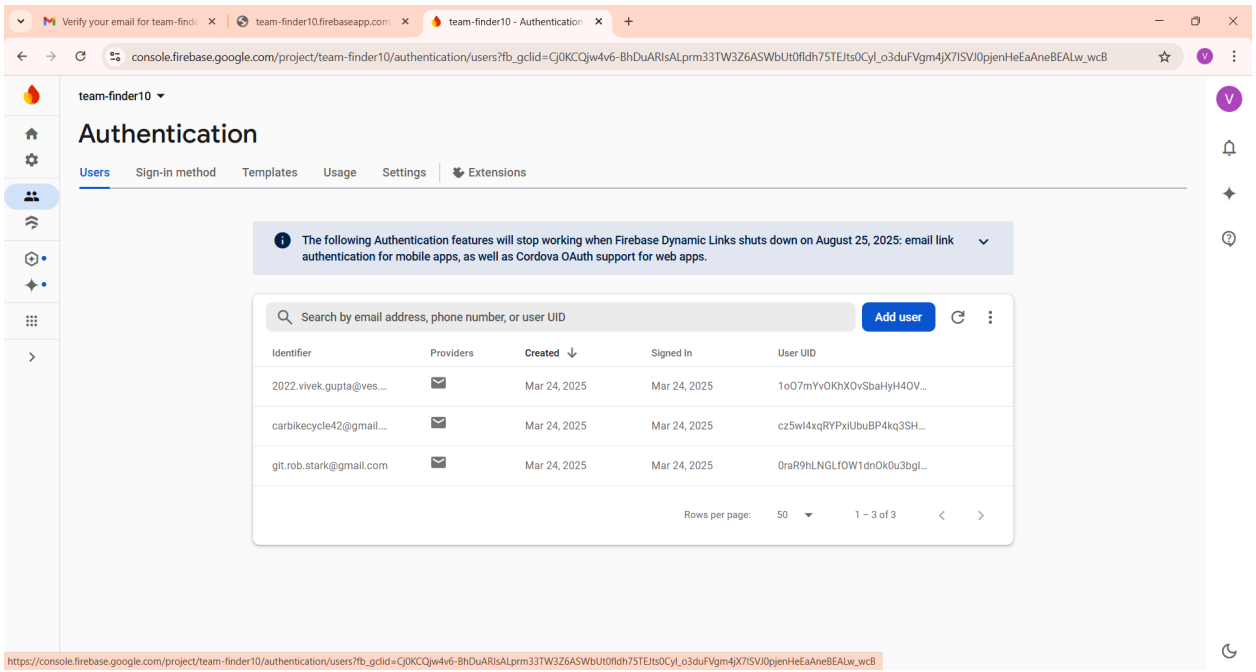
```
      if (!userDoc.exists) {
        // If new user, store their details in Firestore
        await FirebaseFirestore.instance.collection('users').doc(user.uid).set({
          'uid': user.uid,
          'fullName': user.displayName ?? "No Name",
          'email': user.email,
          'joinedTeams': [],
        });
      }
    }
    return user;
  } catch (e) {
    print("Google Sign-In Error: $e");
    return null;
  }
 }

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


Welcome Back!
Login to continue
Email
2022.vivek.gupta@ves.ac.in
Password
Forgot Password?
Login
Don't have an account? Sign Up
Create an Account
Join Team Finder today!
Full Name
Vivek Gupta
Email
2022.vivek.gupta@ves.ac.in
Password
Sign Up
Already have an account? Login
Invalid email or password. Please try again.


Verify your email for team-finder10
mail.google.com/mail/u/0/#inbox/FMfcgzQZTpvjbRqkdwTnxHBwJZXxkXfk
Gmail
Search mail
Compose
Inbox 1,507
Starred
Snoozed
Sent
Drafts 6
More
Labels
Verify your email for team-finder10 External Inbox
noreply@team-finder10.firebaseapp.com
Sun, Mar 23, 11:55 PM (9 hours ago)
Hello, Follow this link to verify your email address. https://team-finder10.firebaseapp.com/__/auth/action?mode=verifyEmail&oobCode=ik914K89AHtBNf0t31Rb3uls6l0w
noreply@team-finder10.firebaseapp.com
9:09 AM (3 minutes ago)
to me
Hello,
Follow this link to verify your email address.
https://team-finder10.firebaseapp.com/__/auth/action?mode=verifyEmail&oobCode=z_qvHQtGAzdIFcuTjCYfK5JuWZjl33C3xFCUX_3Uz0UAAAGVxjvKWA&apiKey=AIzaSyDRadLdPfE2uuyVSluu5-gMolwrhicmBmU&lang=en
Reply
Forward
Enable desktop notifications for Welcome to Vivekanand Education Society Mail.
OK
No thanks







Conclusion:

In this experiment, we successfully implemented navigation using named routes and integrated gestures for user interaction in the Team Finder app. Initially, we faced issues with incorrect route navigation and unresponsive gestures, which we resolved by debugging route names and ensuring GestureDetector was properly wrapped around interactive widgets.

Name : Vivek Gupta
Div : D15B
Roll No : 19

MPL Practical 07

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

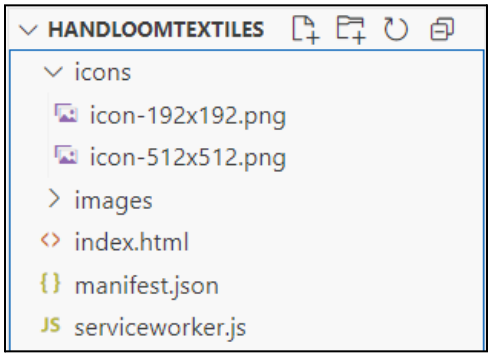
Theory:

The purpose of this experiment is to create a Web App Manifest file that provides metadata about the e-commerce website. This file helps browsers understand how the app should appear when installed on a user's device and enables the "Add to Home Screen" feature.
A manifest file is a JSON file that includes details like the app's name, short name, description, icons, start URL, background color, and display mode. By linking this file in the <head> section of the HTML, we allow the browser to treat our web app more like a native app.

In our project, we created a manifest.json file for the Maharashtrian Handloom & Textiles website. It includes a proper app name, custom theme and background colors matching the site design, and two icons (192x192 and 512x512) for different screen sizes. We also linked this manifest file in our index.html and added a theme-color meta tag to define the browser UI color when the site is launched from the home screen.

This experiment is important because it improves the user experience by making our website installable, giving it a more app-like feel on mobile devices.

Folder Structure:



index.html

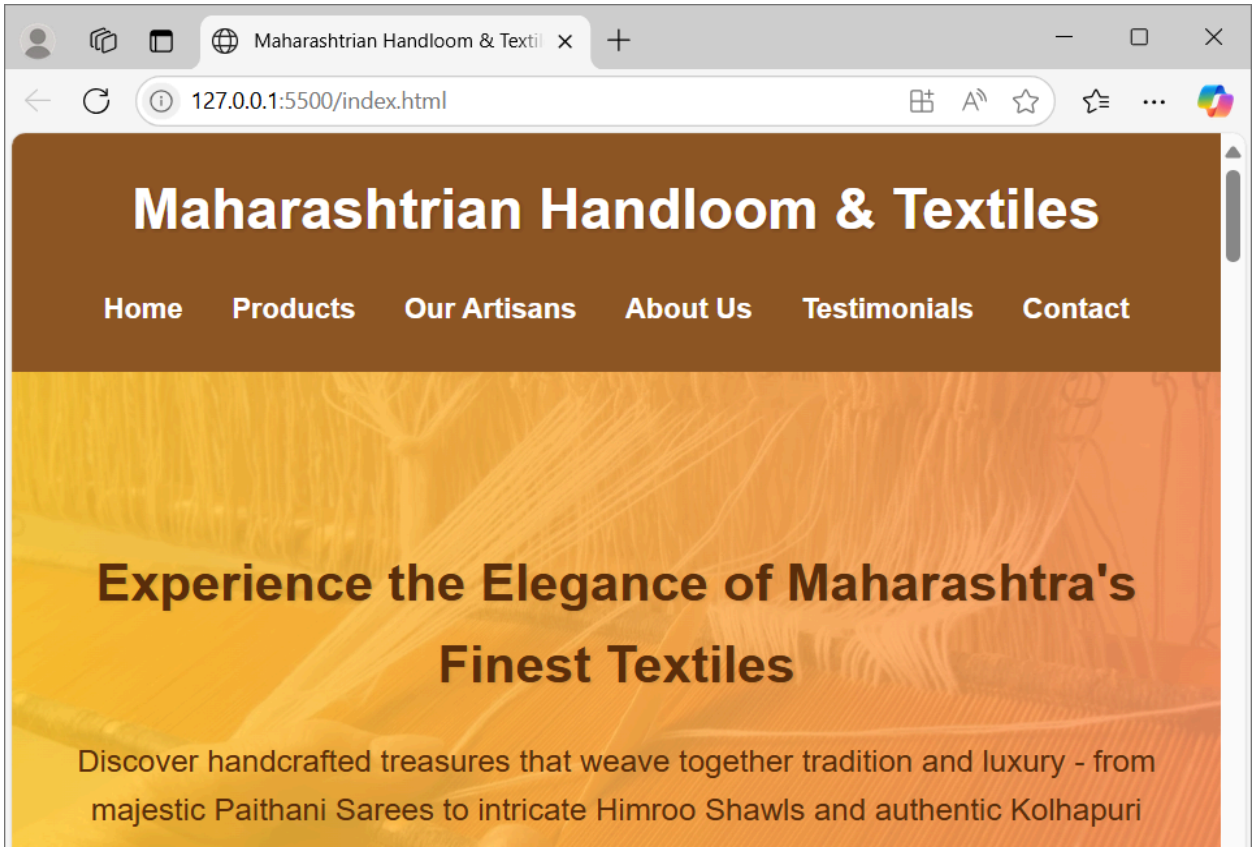
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- Theme Color -->
  <meta name="theme-color" content="#9e5c39">
  <title>Maharashtrian Handloom & Textiles</title>
  <style>
```

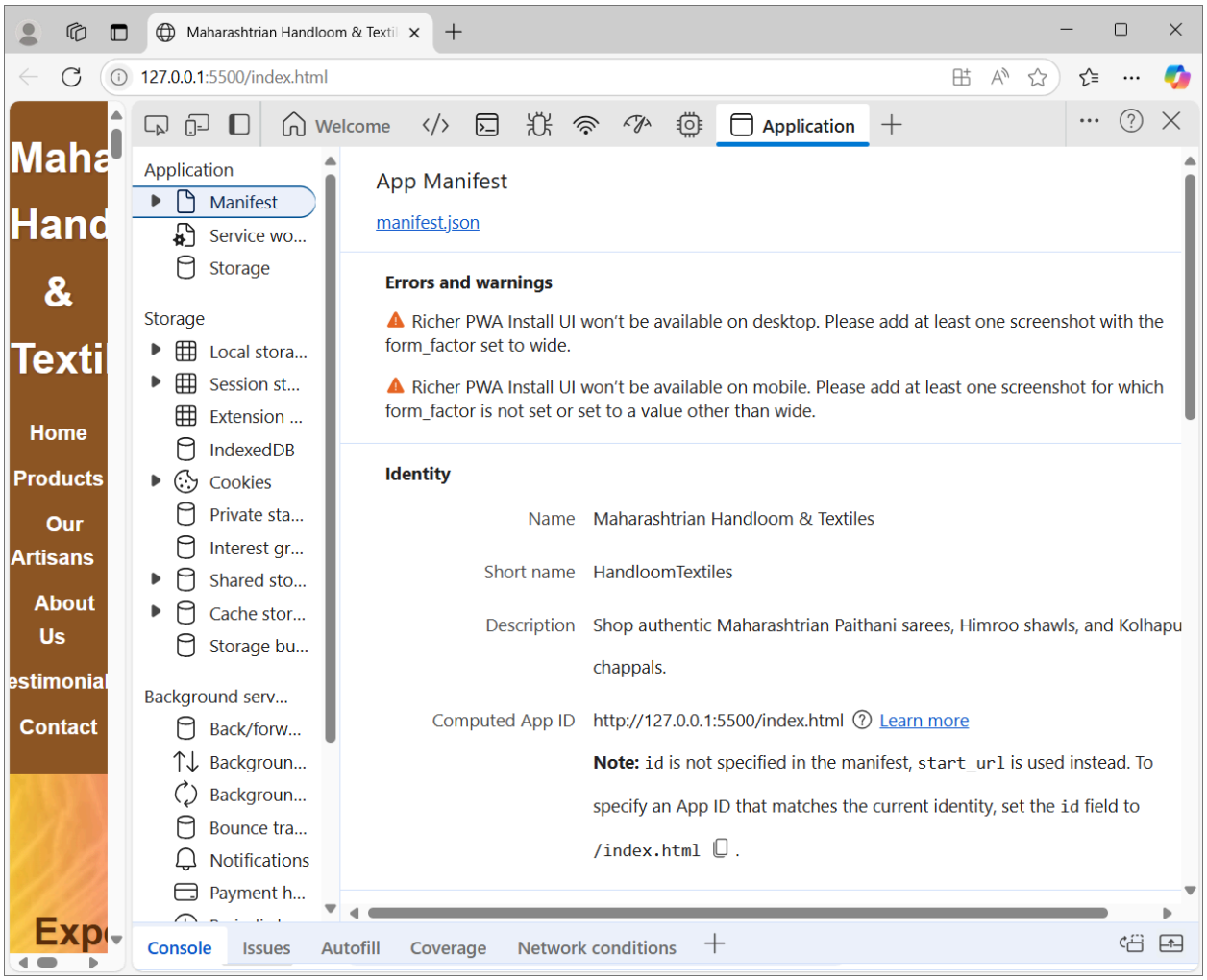
```
  </style>
  <!-- Web Manifest -->
  <link rel="manifest" href="manifest.json">
</head>
<body>


  <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>
```

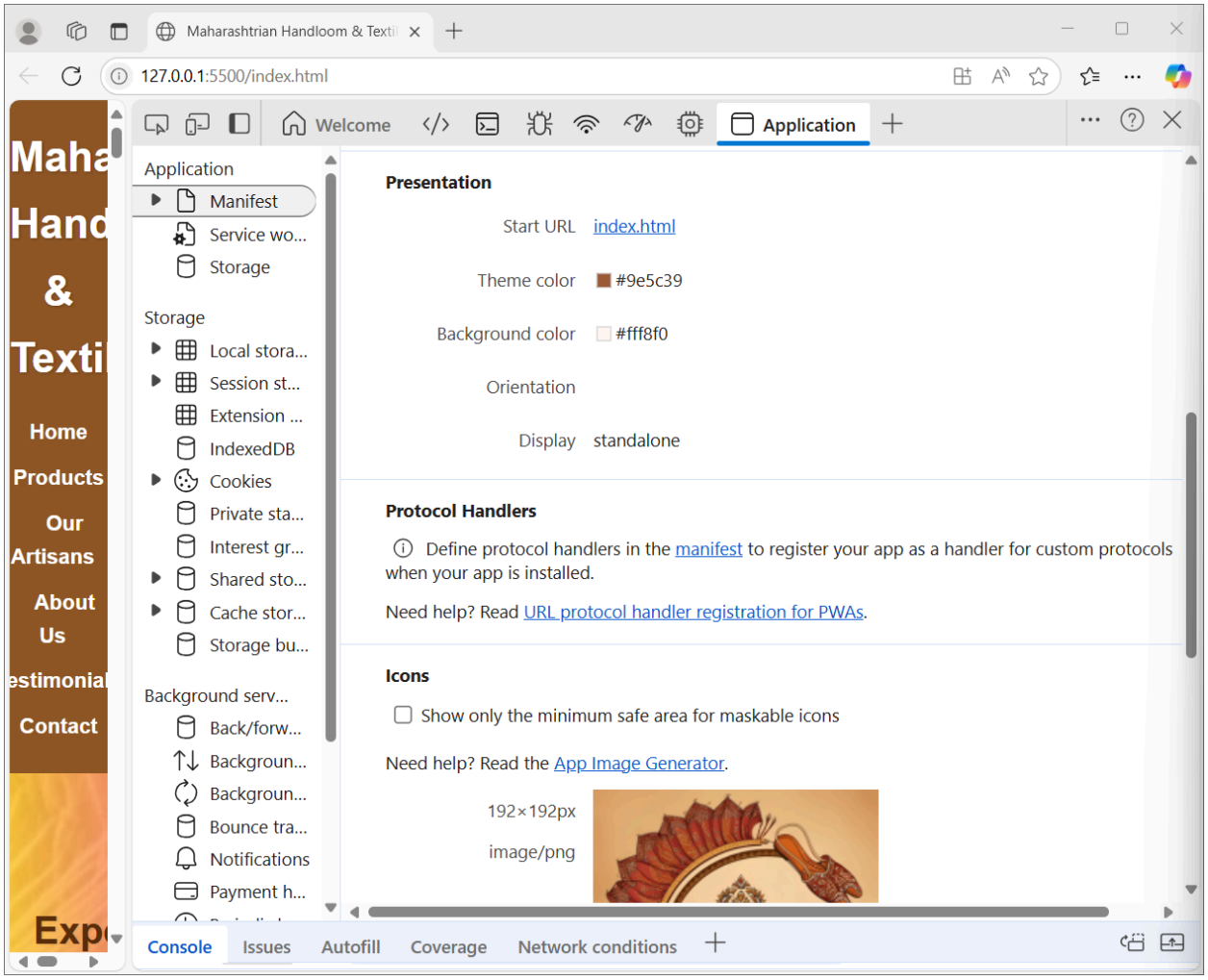
manifest.json

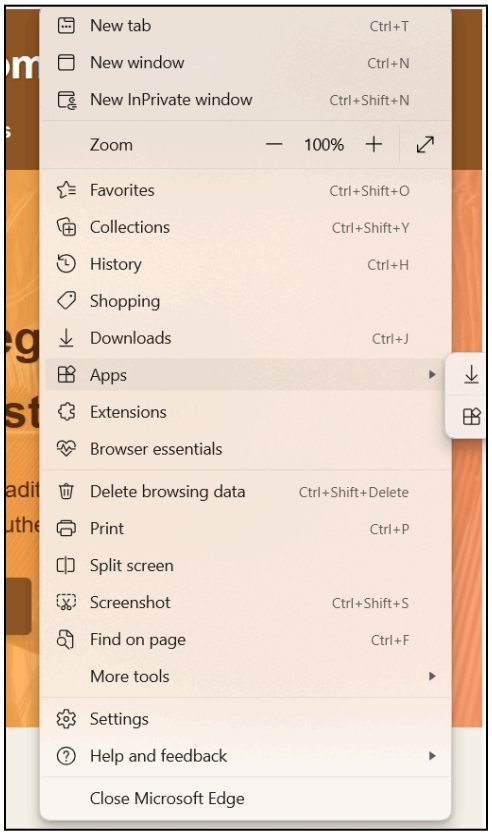
```
{
  "name": "Maharashtrian Handloom & Textiles",
  "short_name": "HandloomTextiles",
  "start_url": "index.html",
  "display": "standalone",
  "background_color": "#fff8f0",
  "theme_color": "#9e5c39",
  "description": "Shop authentic Maharashtrian Paithani sarees, Himroo shawls, and Kolhapuri chappals.",
  "icons": [
    {
      "src": "icons/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "icons/icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
 }
```

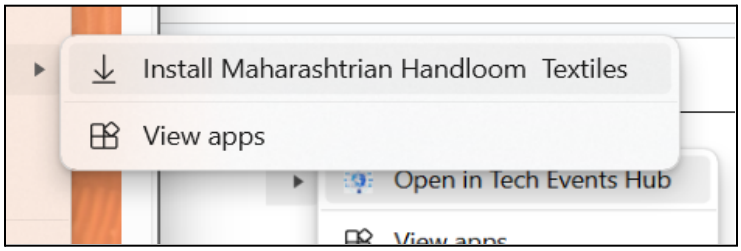
Screenshot:

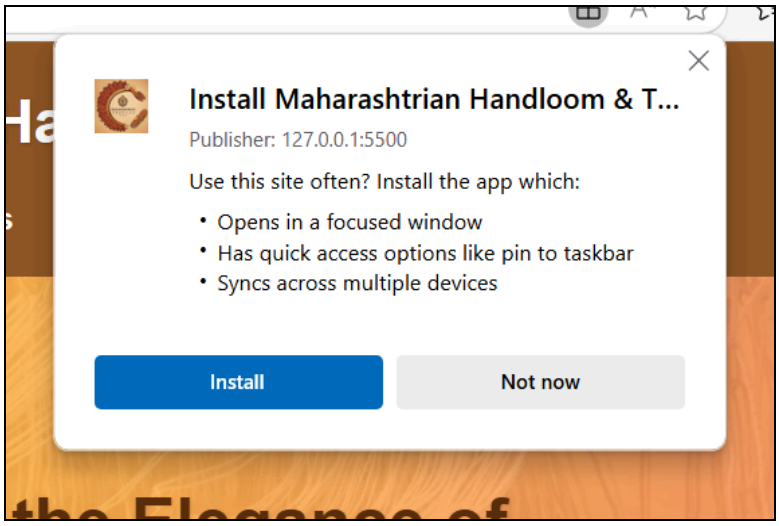

Maharashtrian Handloom & Textil
127.0.0.1:5500/index.html
Maharashtrian Handloom & Textiles
Home
Products
Our Artisans
About Us
Testimonials
Contact
Experience the Elegance of Maharashtra's Finest Textiles
Discover handcrafted treasures that weave together tradition and luxury - from majestic Paithani Sarees to intricate Himroo Shawls and authentic Kolhapuri

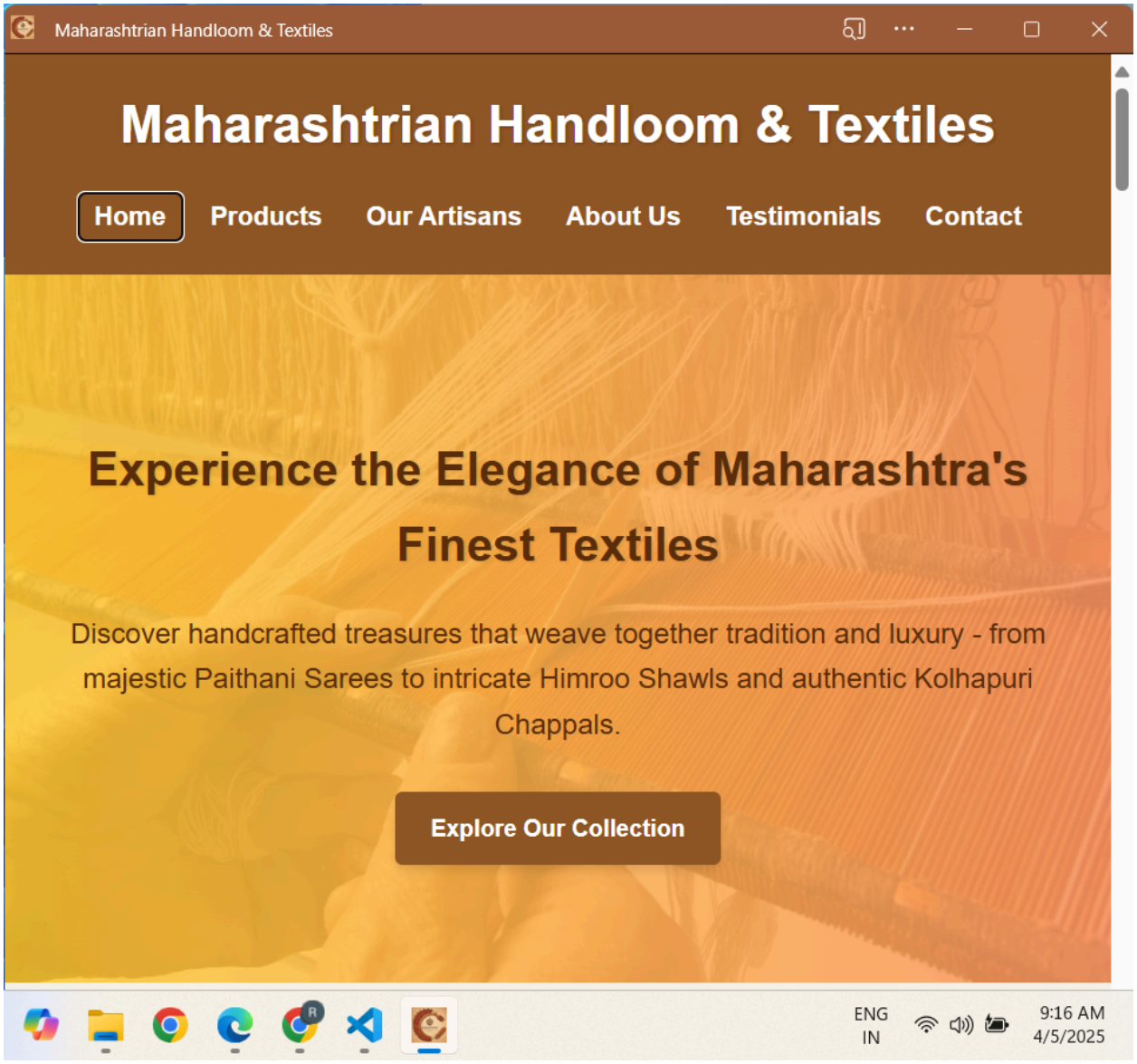

Welcome
Application
Application
Manifest
Service wo...
Storage
Storage
Local stora...
Session st...
Extension ...
IndexedDB
Cookies
Private sta...
Interest gr...
Shared sto...
Cache stor...
Storage bu...
Background serv...
Back/forw...
Backgroun...
Backgroun...
Bounce tra...
Notifications
Payment h...
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
Maharashtrian Handloom & Textiles
Short name
HandloomTextiles
Description
Shop authentic Maharashtrian Paithani sarees, Himroo shawls, and Kolhapu chappals.
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
Issues
Autofill
Coverage
Network conditions


Maharashtrian Handloom & Textil
127.0.0.1:5500/index.html
Welcome
Application
Application
Manifest
Service wo...
Storage
Storage
Local stora...
Session st...
Extension ...
IndexedDB
Cookies
Private sta...
Interest gr...
Shared sto...
Cache stor...
Storage bu...
Background serv...
Back/forw...
Backgroun...
Backgroun...
Bounce tra...
Notifications
Payment h...
Presentation
Start URL index.html
Theme color #9e5c39
Background color #fff8f0
Orientation
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 App Image Generator.
192×192px
image/png
Console
Issues
Autofill
Coverage
Network conditions
Home
Products
Our Artisans
About Us
Contact


New tab Ctrl+T
New window Ctrl+N
New InPrivate window Ctrl+Shift+N
Zoom 100%
Favorites Ctrl+Shift+O
Collections Ctrl+Shift+Y
History Ctrl+H
Shopping
Downloads Ctrl+J
Apps
Extensions
Browser essentials
Delete browsing data Ctrl+Shift+Delete
Print Ctrl+P
Split screen
Screenshot Ctrl+Shift+S
Find on page Ctrl+F
More tools
Settings
Help and feedback
Close Microsoft Edge


Install Maharashtrian Handloom Textiles
View apps
Open in Tech Events Hub


Install Maharashtrian Handloom & T...
Publisher: 127.0.0.1:5500
Use this site often? Install the app which:
• Opens in a focused window
• Has quick access options like pin to taskbar
• Syncs across multiple devices
Install
Not now


Maharashtrian Handloom & Textiles
Maharashtrian Handloom & Textiles
Home
Products
Our Artisans
About Us
Testimonials
Contact
Experience the Elegance of Maharashtra's Finest Textiles
Discover handcrafted treasures that weave together tradition and luxury - from majestic Paithani Sarees to intricate Himroo Shawls and authentic Kolhapuri Chappals.
Explore Our Collection
ENG
IN
9:16 AM
4/5/2025


Maharashtr...
Handloom ...

Conclusion

In this experiment, we successfully implemented the Web App Manifest for our Maharashtrian Handloom & Textiles website, enabling the “Add to Home Screen” feature with custom app name, theme color, and icons. Initially, we faced an issue where the app icon wasn't appearing, which we resolved by correcting the icon file path and verifying the JSON structure.

Name : Vivek Gupta
Div : D15B
Roll No : 19

MPL Practical 08

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:

In Progressive Web Applications (PWAs), a service worker is a script that runs in the background, separate from the web page. It plays a key role in enabling features like offline support, background sync, and push notifications.

This experiment focuses on understanding and implementing the service worker lifecycle, which includes:

- Installation: Caching essential files like HTML, CSS, JS, images, icons, and offline fallback pages.
- Activation: Cleaning up any old caches to ensure the updated service worker is active.
- Fetch Handling: Intercepting network requests to serve cached content when offline or on poor networks.

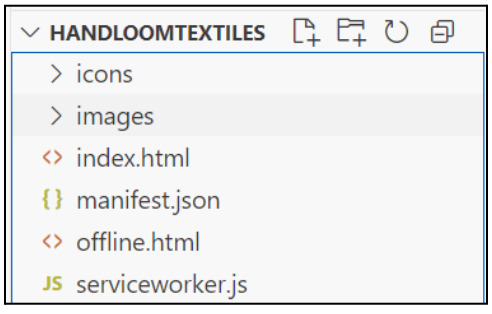
What We Implemented

In our E-commerce PWA:

- We registered a service worker (serviceworker.js) in our main JavaScript file.
- During the install event, we cached important files such as index.html, styles.css, app.js, images, and offline.html to allow offline usage.
- In the activate event, we removed older cache versions to keep the cache clean and updated.
- The fetch event handled all network requests. If the request failed (e.g., no internet), it served the cached version or showed the offline.html fallback page.

This implementation ensures the PWA loads faster and remains functional even without internet connectivity.

Folder Structure



index.html

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

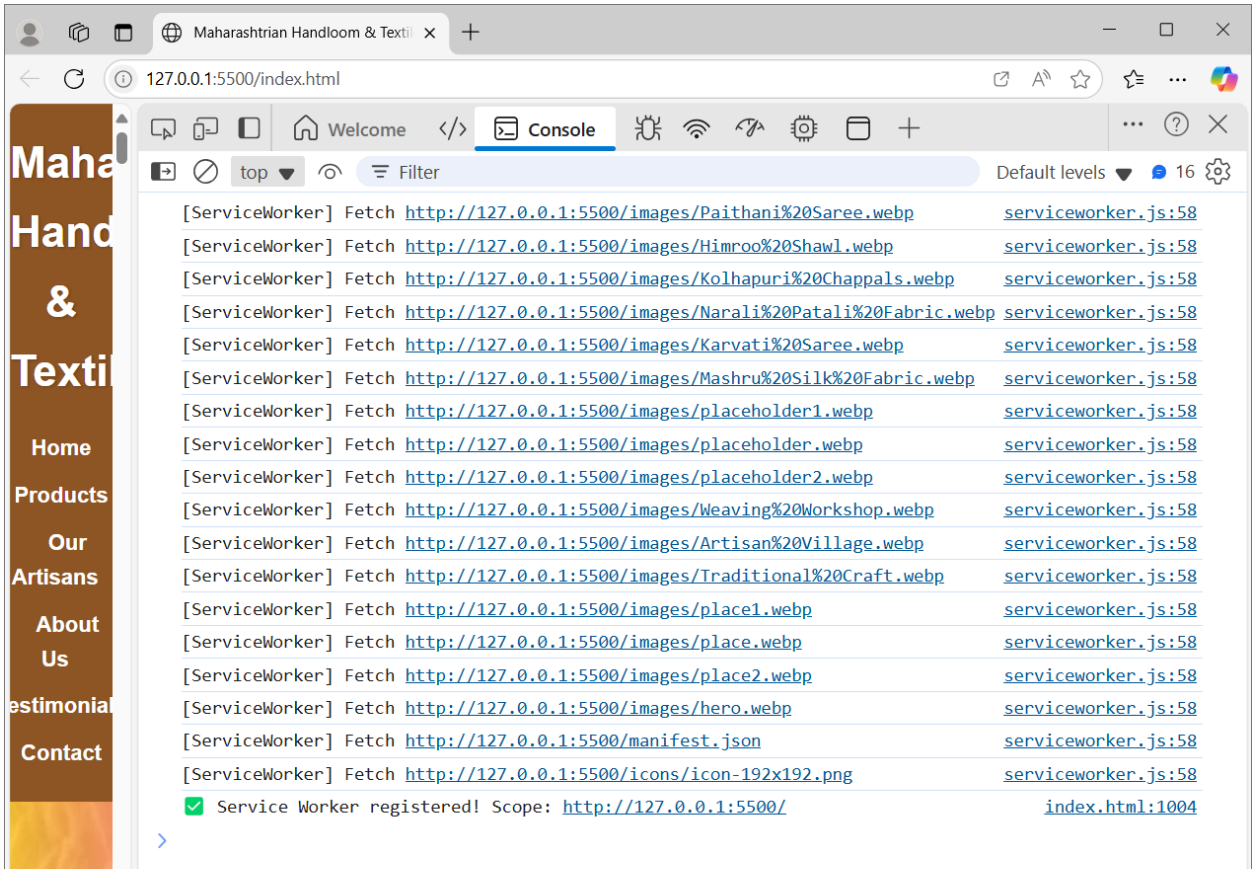
```
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <!-- Theme Color -->
  <meta name="theme-color" content="#9e5c39" />
  <title>Maharashtrian Handloom & Textiles</title>
  <style>

  </style>
  <!-- Web Manifest -->
  <link rel="manifest" href="manifest.json" />
</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>
```

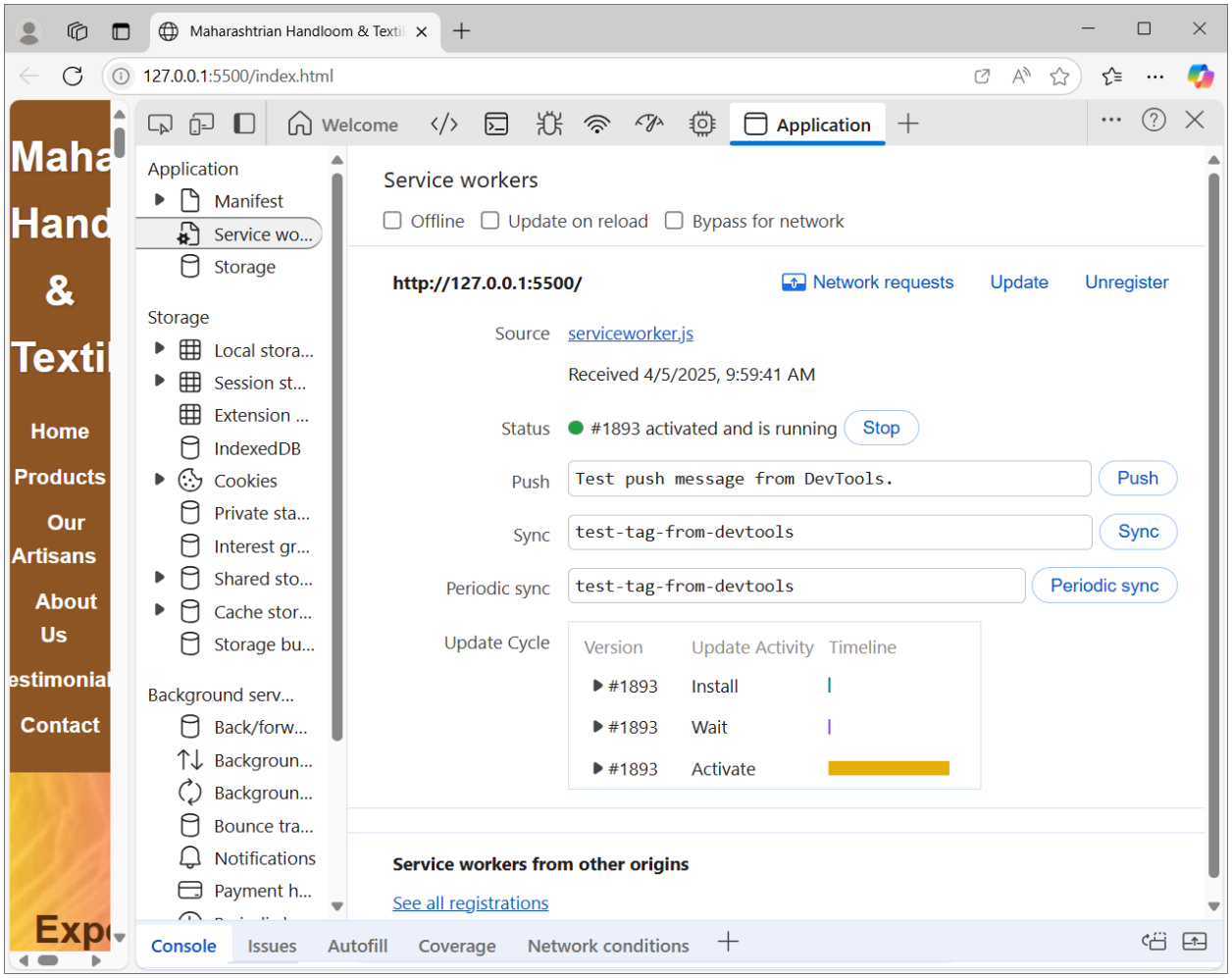
serviceworker.js

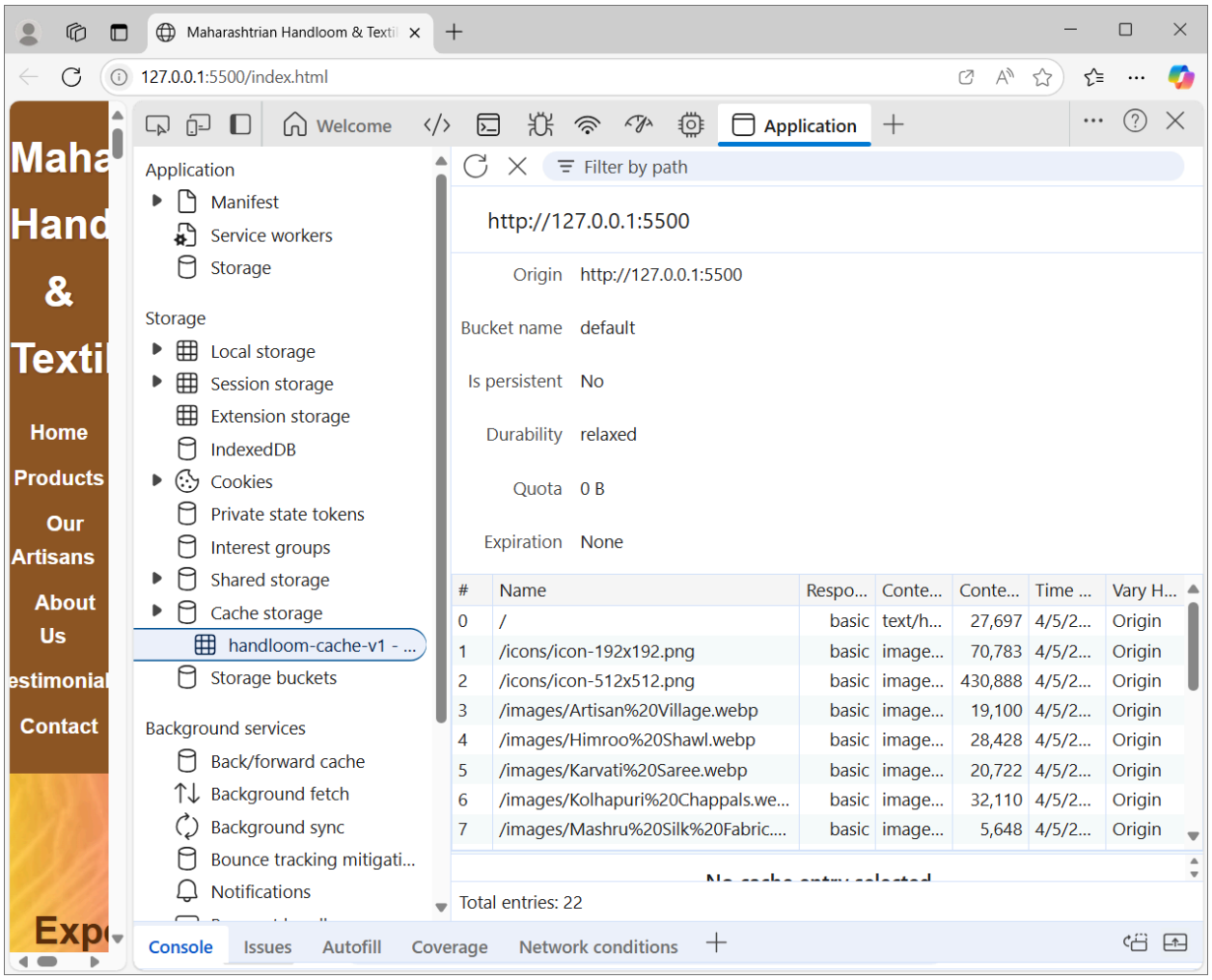
```
const CACHE_NAME = "handloom-cache-v1";
const FILES_TO_CACHE = [
  "/",
  "index.html",
  "manifest.json",
  "icons/icon-192x192.png",
  "icons/icon-512x512.png",
  "images/Artisan Village.webp",
  "images/hero.webp",
  "images/Himroo Shawl.webp",
  "images/Karvati Saree.webp",
  "images/Kolhapuri Chappals.webp",
  "images/Mashru Silk Fabric.webp",
```

```
  "images/Narali Patali Fabric.webp",
  "images/Paithani Saree.webp",
  "images/place.webp",
  "images/place1.webp",
  "images/place2.webp",
  "images/placeholder.webp",
  "images/placeholder1.webp",
  "images/placeholder2.webp",
  "images/Traditional Craft.webp",
  "images/Weaving Workshop.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(FILES_TO_CACHE);
    })
  );
});

// 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();
});
```

Screenshot


Maharashtrian Handloom & Textil
127.0.0.1:5500/index.html
Welcome
Console
top
Filter
Default levels
16
[ServiceWorker] Fetch http://127.0.0.1:5500/images/Paithani%20Saree.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/Himroo%20Shawl.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/Kolhapuri%20Chappals.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/Narali%20Patali%20Fabric.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/Karvati%20Saree.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/Mashru%20Silk%20Fabric.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/placeholder1.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/placeholder.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/placeholder2.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/Weaving%20Workshop.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/Artisan%20Village.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/Traditional%20Craft.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/place1.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/place.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/place2.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/images/hero.webp serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/manifest.json serviceworker.js:58
[ServiceWorker] Fetch http://127.0.0.1:5500/icons/icon-192x192.png serviceworker.js:58
Service Worker registered! Scope: http://127.0.0.1:5500/ index.html:1004
Maha
Hand
&
Texti
Home
Products
Our Artisans
About Us
estimonial
Contact


Maharashtrian Handloom & Textil
127.0.0.1:5500/index.html
Welcome
Application
Application
Manifest
Service wo...
Storage
Storage
Local stora...
Session st...
Extension ...
IndexedDB
Cookies
Private sta...
Interest gr...
Shared sto...
Cache stor...
Storage bu...
Background serv...
Back/forw...
Backgroun...
Backgroun...
Bounce tra...
Notifications
Payment h...
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5500/
Network requests
Update
Unregister
Source serviceworker.js
Received 4/5/2025, 9:59:41 AM
Status #1893 activated and is running Stop
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
#1893 Install
#1893 Wait
#1893 Activate
Service workers from other origins
See all registrations
Console
Issues
Autofill
Coverage
Network conditions
Maha
Hand
&
Texti
Home
Products
Our Artisans
About Us
estimonial
Contact
Expe



Conclusion

In this experiment, we successfully implemented a service worker to enable offline access for the Maharashtrian Handloom & Textiles website by caching essential files and handling fetch events. Initially, we faced issues like 404 errors due to incorrect file paths and missing files, but we resolved them by verifying the paths and ensuring all referenced assets were present in the project directory.

Name : Vivek Gupta
Div : D15B
Roll No : 19

MPL Practical 09

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

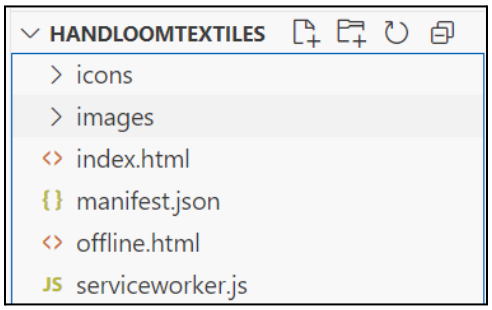
Theory:
Service workers are powerful scripts that run in the background of Progressive Web Apps (PWAs), enabling features like offline access, background sync, and push notifications. In this experiment, we focused on three key service worker events: fetch, sync, and push.

1. Fetch Event
The fetch event allows the service worker to intercept network requests. In our code, we used it to serve cached files when the network is unavailable, helping our E-commerce website work offline. This improves performance and reliability by loading essential resources from cache.

2. Sync Event
The sync event enables background data synchronization when the user regains internet connectivity. In our implementation, we registered a sync task named 'sync-data'. Once the device is back online, the service worker gets triggered and performs background tasks like logging or syncing cart data—ensuring smooth user experience even after going offline.

3. Push Event
The push event is used to receive and display push notifications sent from the server. In our code, we handled this event by showing a browser notification with custom text. We also included error handling to support both JSON and plain text messages.

Folder Structure:



index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <!-- Theme Color -->
  <meta name="theme-color" content="#9e5c39" />
  <title>Maharashtrian Handloom & Textiles</title>
```

```
  <style>

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

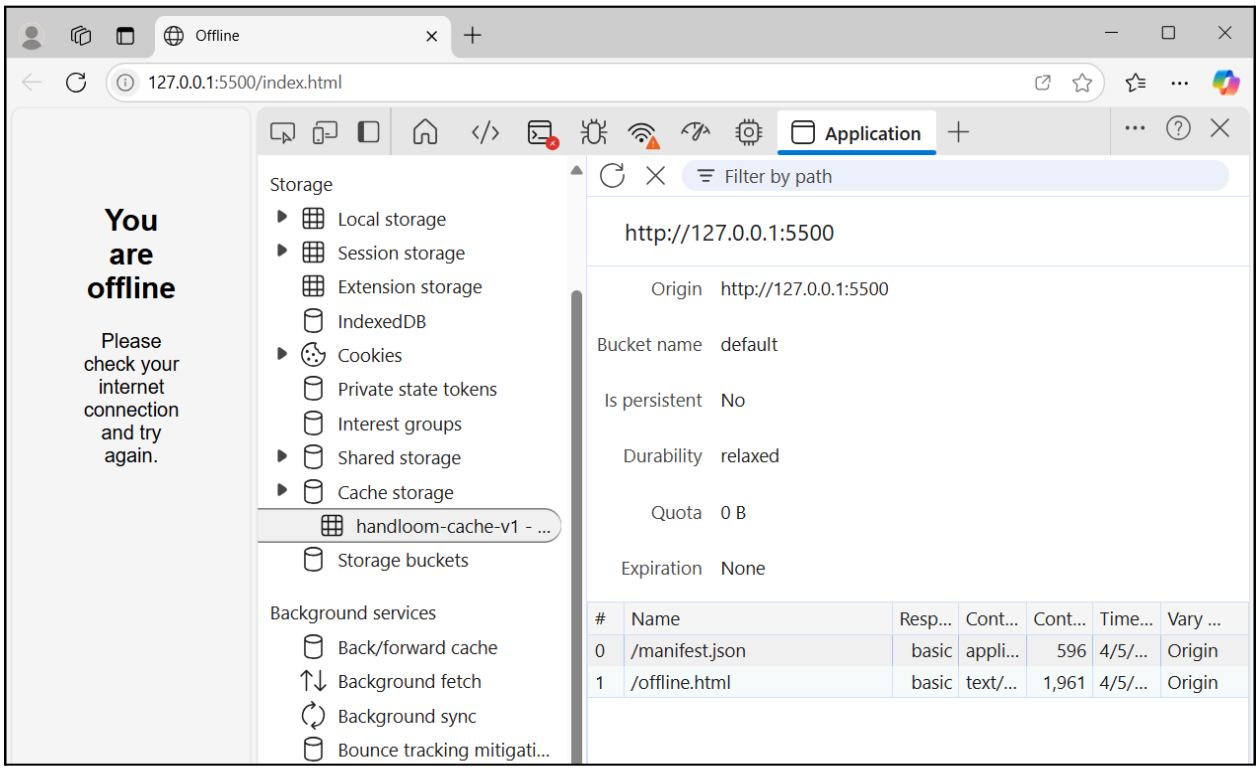
```
const CACHE_NAME = "handloom-cache-v1";
const FILES_TO_CACHE = [
 "/",
 "index.html",
 "manifest.json",
 "icons/icon-192x192.png",
 "icons/icon-512x512.png",
 "images/Artisan Village.webp",
 "images/hero.webp",
 "images/Himroo Shawl.webp",
 "images/Karvati Saree.webp",
 "images/Kolhapuri Chappals.webp",
 "images/Mashru Silk Fabric.webp",
 "images/Narali Patali Fabric.webp",
 "images/Paithani Saree.webp",
 "images/place.webp",
 "images/place1.webp",
 "images/place2.webp",
 "images/placeholder.webp",
 "images/placeholder1.webp",
 "images/placeholder2.webp",
 "images/Traditional Craft.webp",
 "images/Weaving Workshop.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(FILES_TO_CACHE);
  })
 );
});

// 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(
```

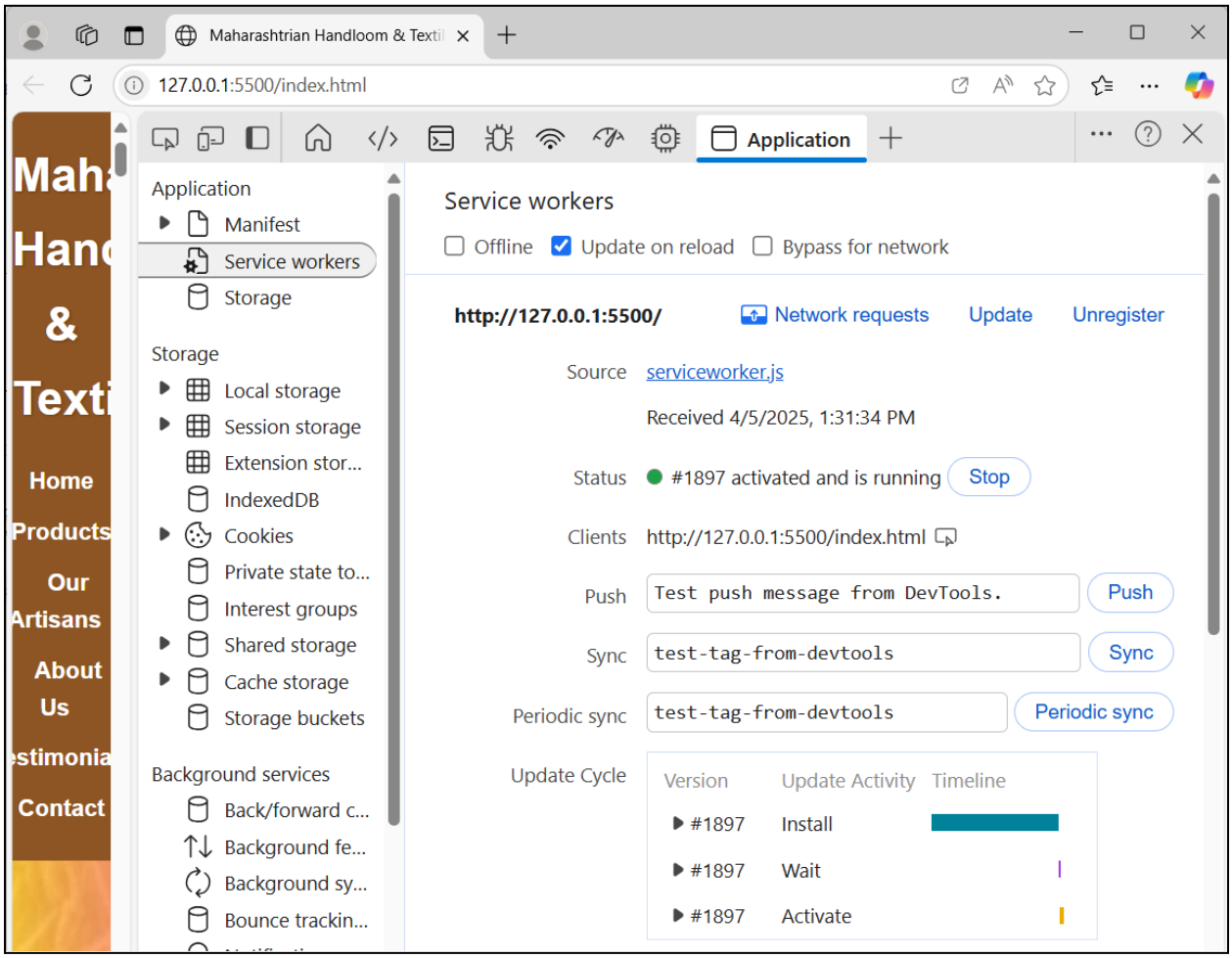
```
    fetch(event.request)
      .then((response) => {
        return response;
      })
      .catch(() =>
        caches.match(event.request).then((res) => {
          return res || caches.match("offline.html");
        })
      )
  );
 }
});

// 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("Maharashtrian Handloom", {
        body: data.message,
      })
    );
  }
 }
});
```

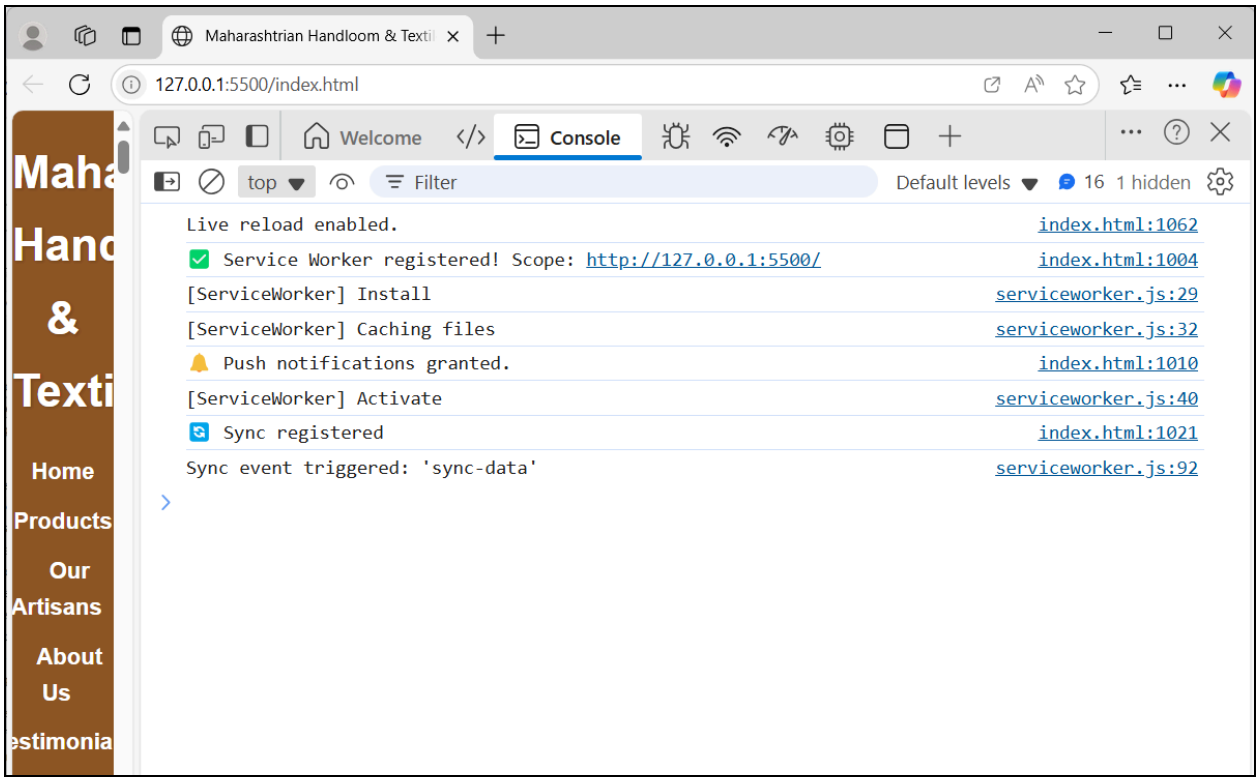
Screenshot

1. Test: Fetch Event (Offline Support)

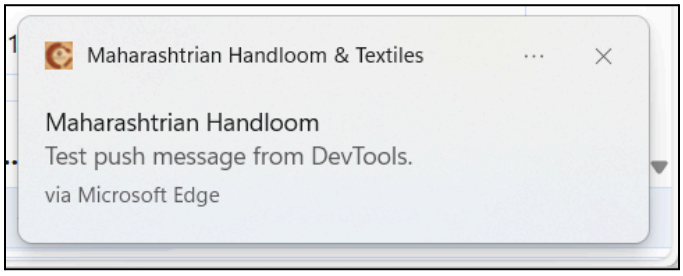


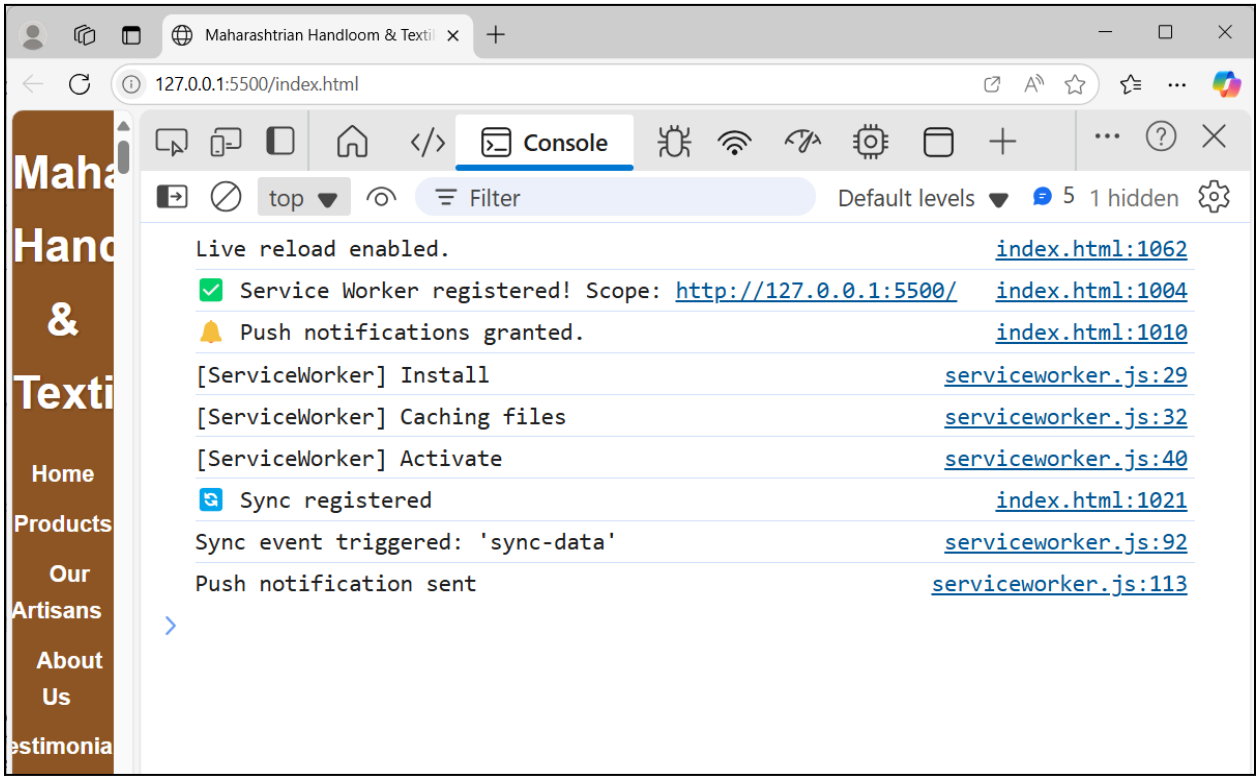
2. Test: Background Sync Event





3. Test: Push Notification Event





Conclusion

In this experiment, we successfully implemented fetch, sync, and push events in the service worker for our *Maharashtrian Handloom & Textiles* PWA. While testing push notifications, we initially faced a JSON parsing error, but we resolved it by updating the code to handle both JSON and plain text payloads properly.

Name : Vivek Gupta
Div : D15B
Roll No : 19

MPL Practical 10

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

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.

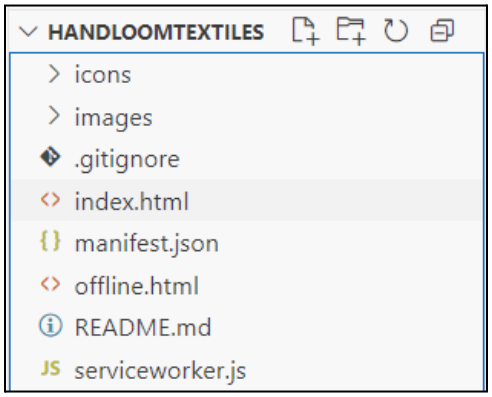
What we implemented in our code

In this experiment, we deployed our E-commerce PWA project, "Maharashtrian Handloom & Textiles", to GitHub Pages using these steps:

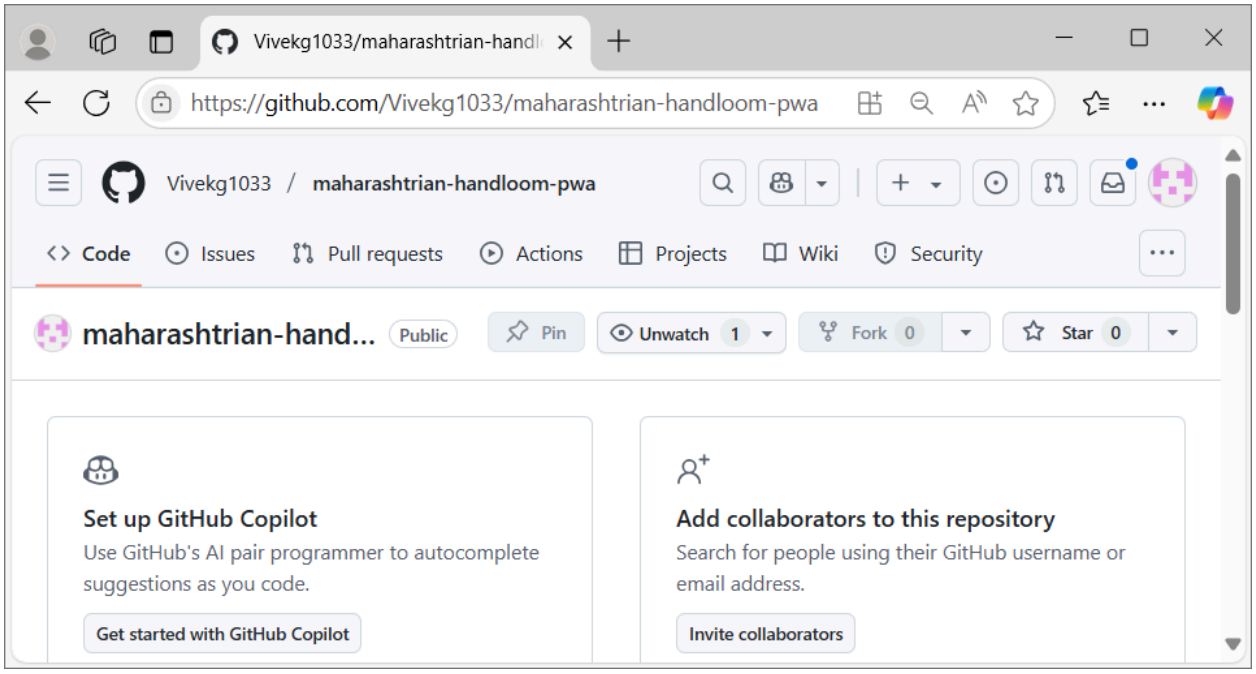
- We ensured all necessary PWA files were in place, such as index.html, manifest.json, serviceworker.js, and offline assets.
- The index.html file used relative paths correctly so resources load properly after deployment.
- We registered the service worker to enable offline access and caching of essential 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.

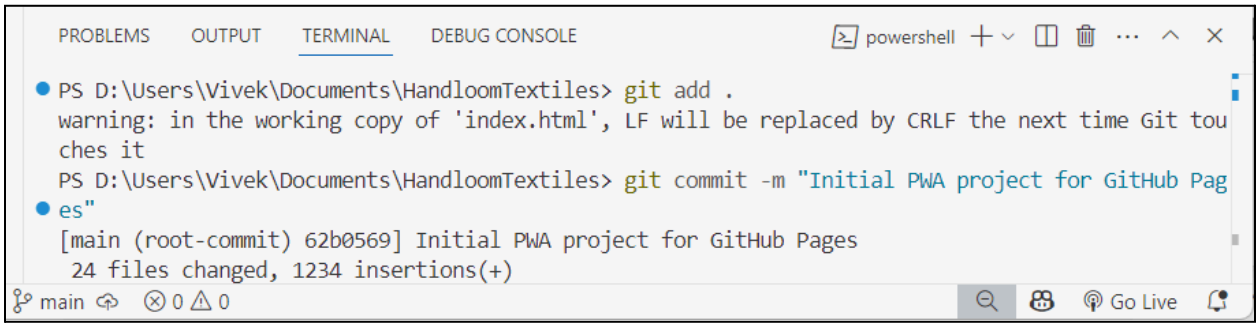
Folder Structure

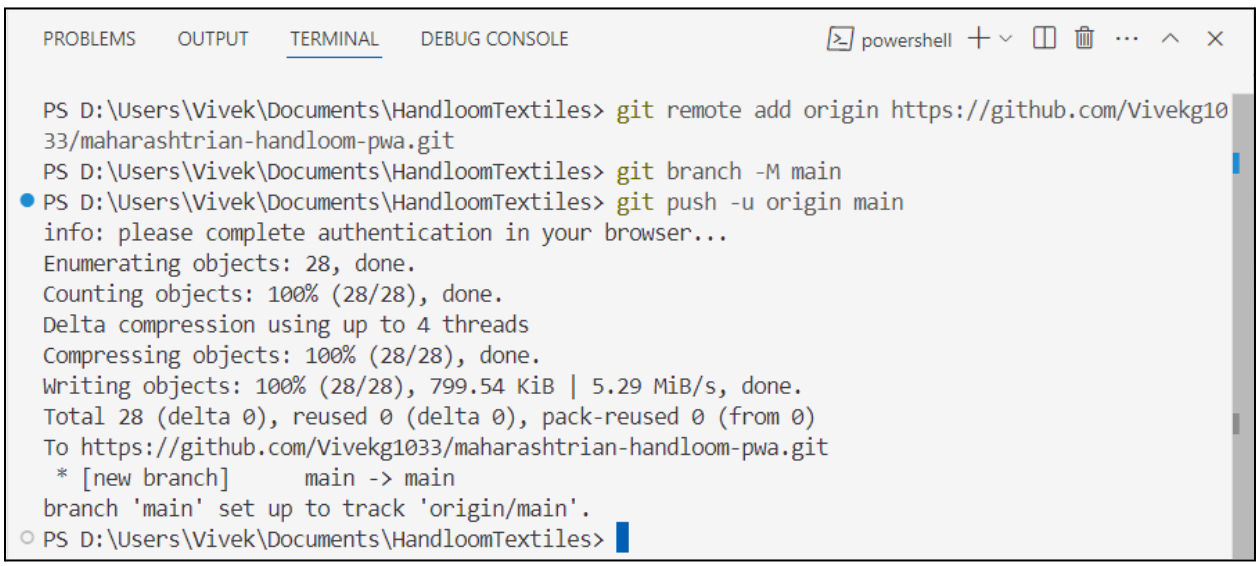


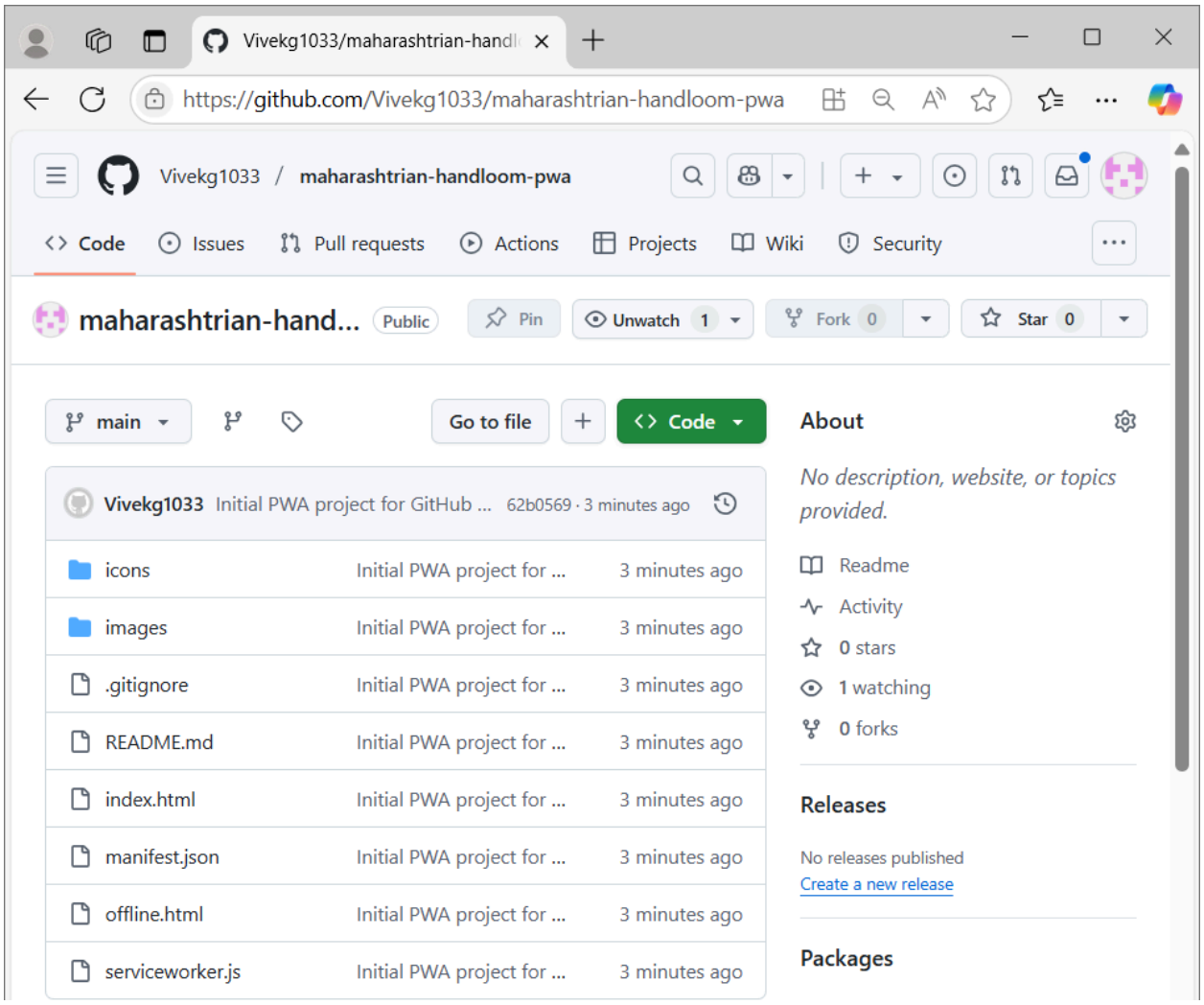
Create a GitHub Repository

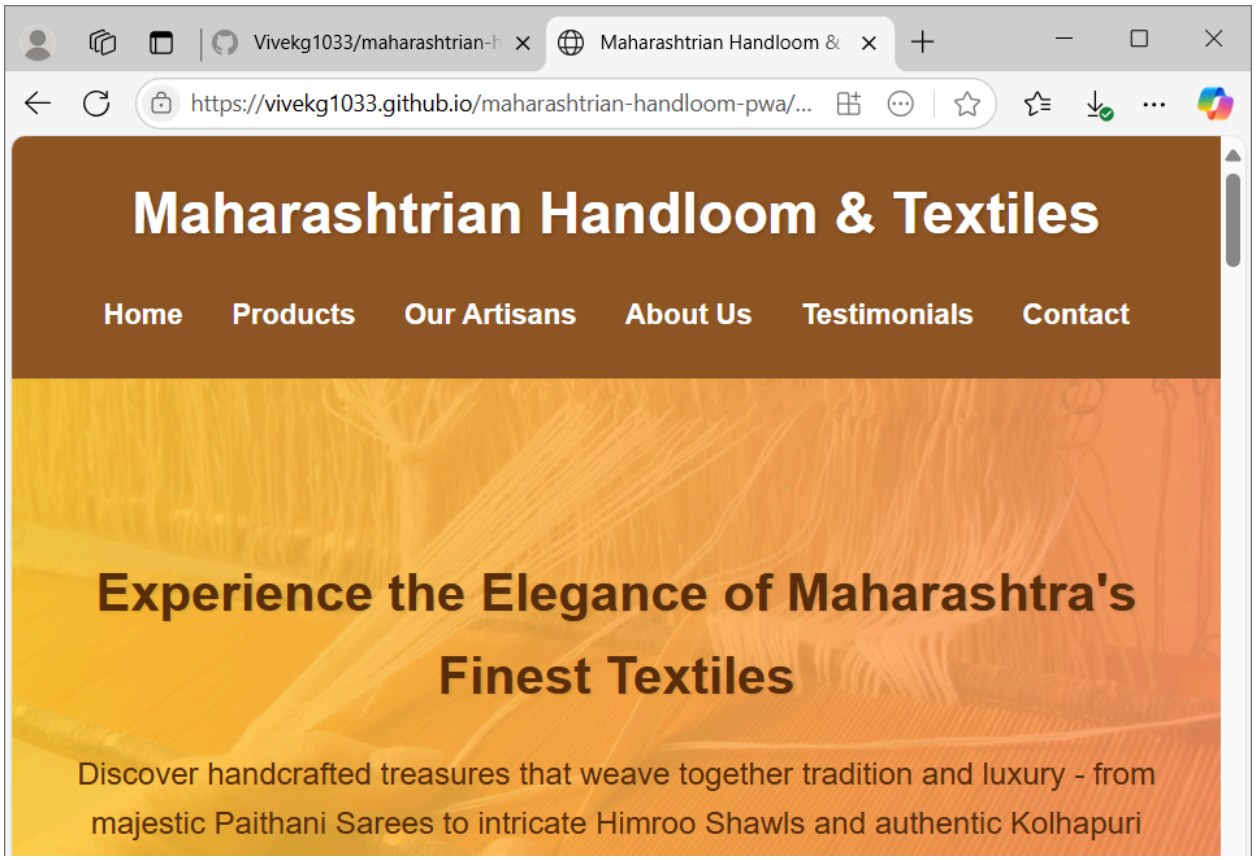


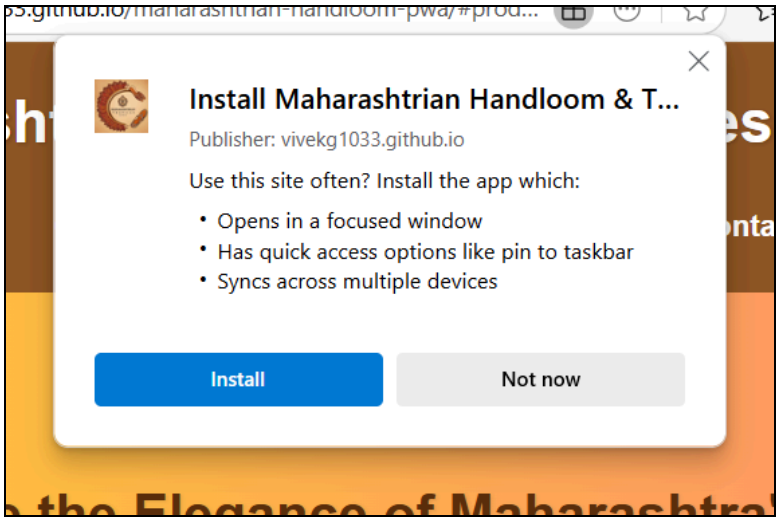
Link Your Local Project to GitHub and Push Changes.






Vivekg1033/maharashtrian-handl
https://github.com/Vivekg1033/maharashtrian-handloom-pwa
Vivekg1033 / maharashtrian-handloom-pwa
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
maharashtrian-hand...
Public
Pin
Unwatch 1
Fork 0
Star 0
main
Go to file
Code
About
No description, website, or topics provided.
Readme
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
Vivekg1033 Initial PWA project for GitHub ... 62b0569 · 3 minutes ago
icons Initial PWA project for ... 3 minutes ago
images Initial PWA project for ... 3 minutes ago
.gitignore Initial PWA project for ... 3 minutes ago
README.md Initial PWA project for ... 3 minutes ago
index.html Initial PWA project for ... 3 minutes ago
manifest.json Initial PWA project for ... 3 minutes ago
offline.html Initial PWA project for ... 3 minutes ago
serviceworker.js Initial PWA project for ... 3 minutes ago


Vivekg1033/maharashtrian-h
Maharashtrian Handloom &
https://vivekg1033.github.io/maharashtrian-handloom-pwa/...
Maharashtrian Handloom & Textiles
Home
Products
Our Artisans
About Us
Testimonials
Contact
Experience the Elegance of Maharashtra's Finest Textiles
Discover handcrafted treasures that weave together tradition and luxury - from majestic Paithani Sarees to intricate Himroo Shawls and authentic Kolhapuri



Conclusion

In this experiment, we successfully deployed the Maharashtrian Handloom & Textiles 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.

Name : Vivek Gupta
Div : D15B
Roll No : 19

MPL Practical 11

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

Theory:

Google Lighthouse is an open-source tool provided by Google that helps developers analyze the quality of web applications, including Progressive Web Apps (PWAs). It checks key performance aspects like speed, accessibility, SEO, best practices, and PWA standards.

Lighthouse simulates a mobile device and runs various audits to give scores (out of 100) for different categories. This helps identify how fast and optimized a web app is, and whether it meets PWA requirements like offline support, responsiveness, secure HTTPS hosting, and manifest availability.

What we implemented:
In this experiment, we tested our Maharashtrian Handloom & Textiles PWA using Lighthouse. The app was hosted on GitHub Pages and included PWA features such as:

- A valid web app manifest
- A registered and active service worker
- Offline caching of assets
- Responsive design and mobile-friendly layout

After running Lighthouse, we achieved the following scores:

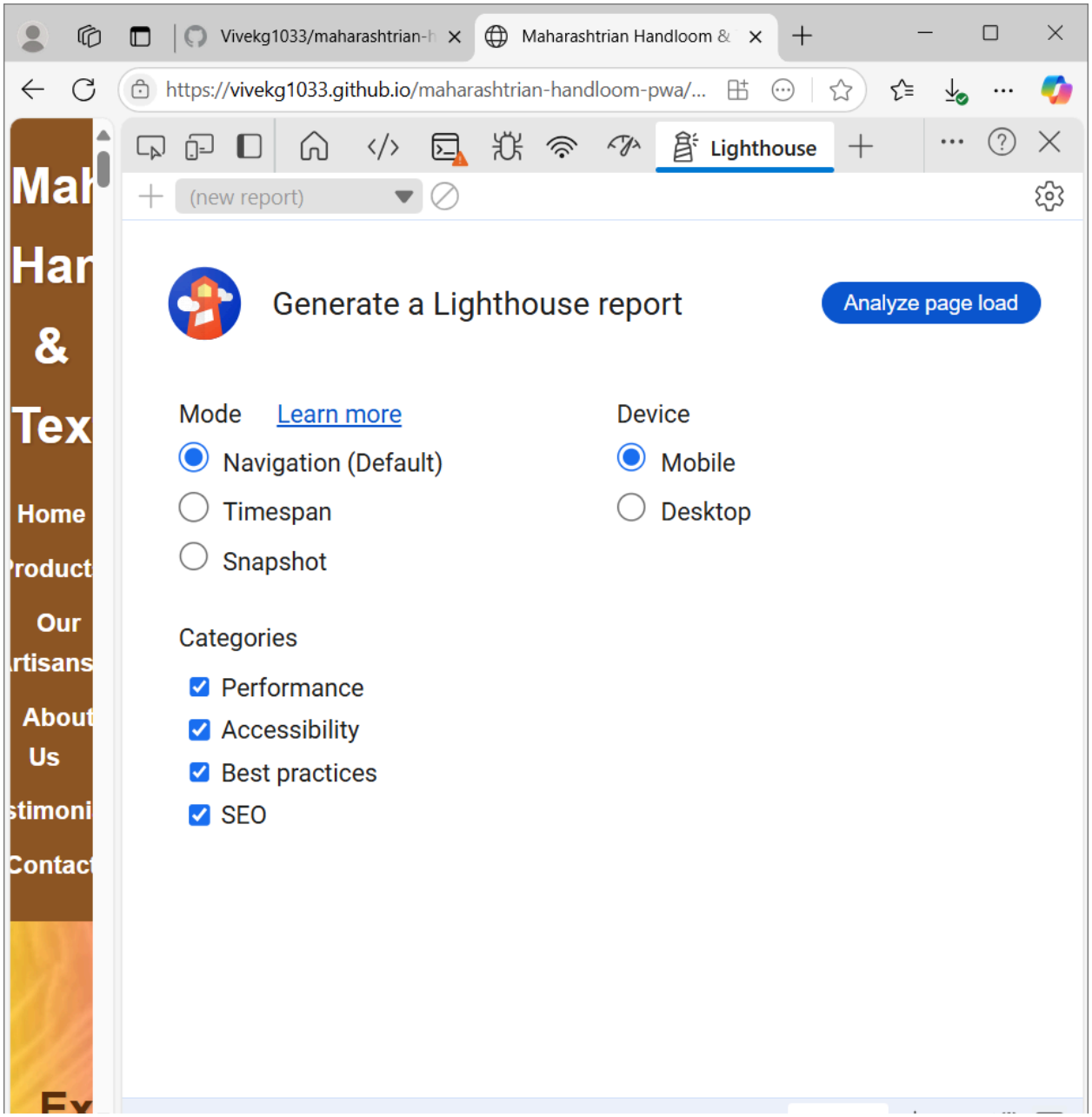
- Performance: 87 – Fast load time with some optimization scope (e.g., reduce main-thread blocking).
- Accessibility: 100 – All content is easily accessible and screen reader-friendly.
- Best Practices: 96 – Follows security and coding standards well.
- SEO: 91 – Good search engine discoverability with minor improvements needed (e.g., add meta description).

These results confirm that our PWA meets the major requirements and performs well across devices.

manifest.json

```
{
  "name": "Maharashtrian Handloom & Textiles",
  "short_name": "HandloomTextiles",
  "start_url": "index.html",
  "display": "standalone",
  "background_color": "#fff8f0",
  "theme_color": "#9e5c39",
  "description": "Shop authentic Maharashtrian Paithani sarees, Himroo shawls, and Kolhapuri
chappals.",
```

```
 "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 successfully tested our Maharashtrian Handloom & Textiles PWA using Google Lighthouse and achieved high scores by implementing a valid manifest, service worker, and responsive design. Initially, we faced issues like missing theme color and improper icon formats, which we resolved by updating the manifest.json and adding necessary meta tags in index.html.

https://vivekg1033.github.io/maharashtrian-handloom-pwa/#products

Performance	Accessibility	Best Practices	SEO
87	100	96	91

87

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: 1.3 s
- Total Blocking Time: 520 ms
- Cumulative Layout Shift: 0
- Speed Index: 1.8 s

View Treemap

Show audits relevant to: All LCP TBT

DIAGNOSTICS

- Minimize main-thread work — 2.8 s
- Serve static assets with an efficient cache policy — 16 resources found
- Avoid long main-thread tasks — 8 long tasks found
- Avoid chaining critical requests — 1 chain found
- Largest Contentful Paint element — 1,330 ms

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

PASSED AUDITS (33) Show

100

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.

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 (15) Show

NOT APPLICABLE (42) Show

96

Best Practices

TRUST AND SAFETY

- ▲ Requests the notification permission on page load
- ○ 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 (14) 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.

CONTENT BEST PRACTICES

- ▲ Document does not have a meta description

Format your HTML in a way that enables crawlers to better understand your app's content.

ADDITIONAL ITEMS TO MANUALLY CHECK (1) Show

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

PASSED AUDITS (7) Show

NOT APPLICABLE (2) Show

Captured at Apr 6, 2025, 10:59 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

Name :- Divesh S. Gupta
SPIV :- DISB
Roll no :- Z19

SDS | Page No. | Date

MPL Assignment 1

9.1 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 Component - Comes with customizable widget for smooth UI design.
5. Open source - Free to use and has a strong developer community.

Advantages of Using Flutter:

1. Saves time & Effort - Single codebase for multiple platform
2. High Speed Development - Hot Reload feature.
3. Cost Effective - Reduces development cost
4. Attractive UI - Provides beautiful and customizable widget.

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

→ How flutter Differs from Traditional Approaches.

1. Single Codebase - Traditional methods need seperate code for Android and iOS, but flutter uses one code for both.
2. Hot Reload - Traditional apps require full restart after changes.

3. UI Rendering - Traditional apps use native components, while flutter has its own rendering engine for faster performance.
4. Performance - Flutter compiles directly to native machine code, making it faster than frameworks.

Why Flutter is Popular Among developers.
1. Fast Development - Hot Reload and single codebase
2. Cross Platform Support - Works on mobile, web and desktop
3. Beautiful UI - Rich, customizable widgets for modern
4. High Performance - Runs smoothly without a bridge

9.2.3] Describe the concept of widget tree in flutter. Explain how widget composition is used to build complex interface.
→. Concept of Widget Tree in flutter:
In flutter, everything is a widget, widgets are arranged in a tree structure, called the widget tree. This tree represents the UI of app.

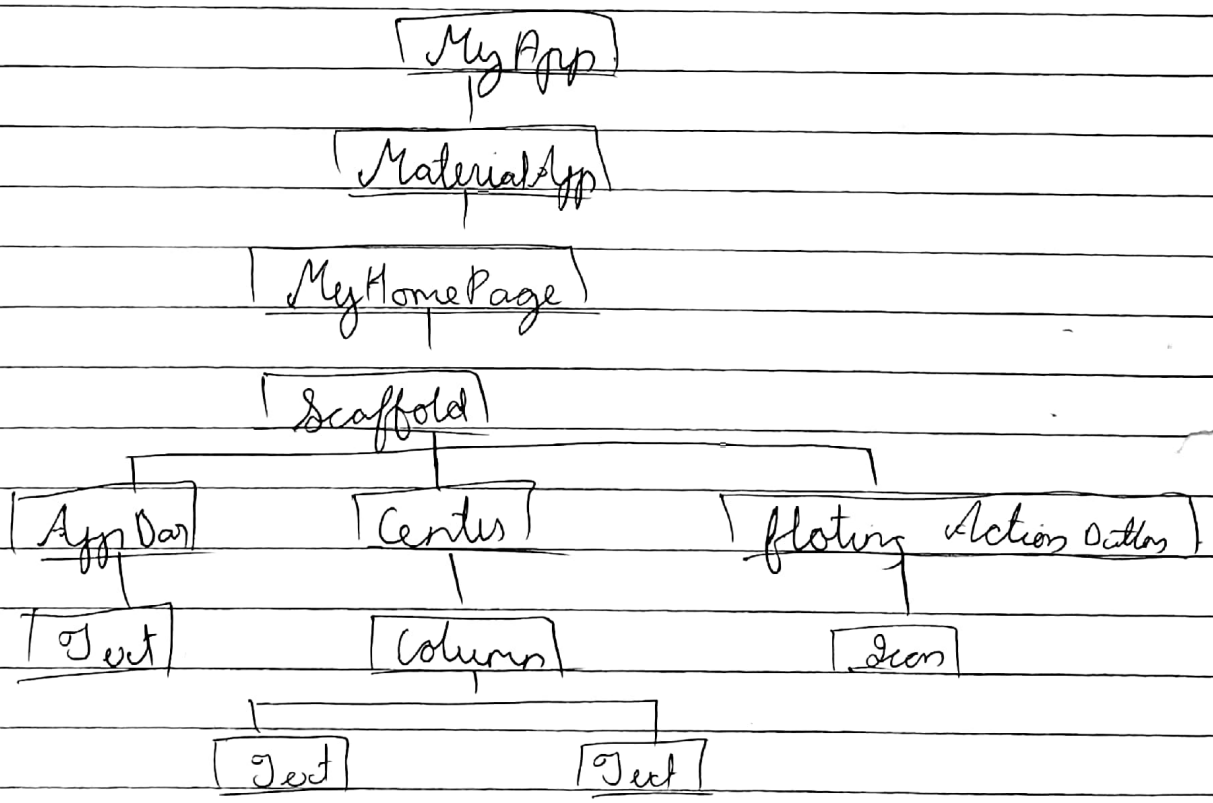
Widget Composition for Complex UI:
flutter uses small, reusable widgets to build complex UI. Instead of creating a single large UI block, developers combine multiple widgets

eg. 1] A ListView can contain multiple Card Widgets
2] A Column can hold text, Images and Buttons

b) Provide examples of commonly used widgets.

→ 1). Scaffold - Provides the basic layout structure
2). AppBar - Displays the top navigation bar with title
3) Text - Displays single text
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.

Example Widget tree



SDS Page No.
Date

3. a]. Discuss the importance of state management in flutter.

→ State management is important because it control how the app store, update, and display data when the user interact with it.

Why State Management is Needed?

1). Keeps UI Updated - Ensures that app reflect change.
2) Improves Performance - Update only necessary part of UI instead of reloading everything.
3). Manages Complex Data - Helps handle D user input
4). Ensures Smooth User Experience - Keeps the app responsive

Types of State in Flutter:

1. Local State - Managed within a single widget
2. Global State - Shared across multiple screens.

b]. Compare and contrast the different state management approaches available in Flutter, such as setState, Providers and Riverpod.

→

Approach	How it Works	When to Use
setState	Update UI by calling setState() in a StatefulWidget	Best for small apps or managing state.
Providers	Uses InheritedWidget to share state across widgets	Suitable for medium sized apps where data
Riverpod	A improved version of Providers with better performance single syntax.	Best for large apps that need complex state

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.

9.4.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]
2. Add firebase to flutter App - Register the app and download the google services.json.
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 apps.
3. Scalability - Handles large bases without managing
4. Push Notification - Sends alerts and updates to users

SDS Page No.
Data / /

b) Highlight the Firebase services commonly used in flutter development and provide a brief overview of how data synchronization is achieved.

→ Common firebase services used in flutter Development.

1. Firebase Authentication - Provides user sign-in method
2. Cloud firestore - A NoSQL database that store and sync data in real time.
3. Firebase Realtime Database - Stores and updates data instantly across all connected devices
4. Firebase cloud storage - Used for storing and retrieving files like images and videos.
5. firebase Cloud Messaging (FCM) - Sends push notifications.
6. Firebase hosting - Deploys web apps with fast and secure hosting.

How data synchronization is Achieved.

1. Real time Update - Firestore and Realtime Database sync data across device
2. Listeners & Streams - Widget listen for changes
3. Offline Support - firebase caches data, allowing apps to work offline and sync when online.

This ensures fast, smooth and automatic data update in flutter apps.

Name :- Vivek S Gupta
Div :- D15B
Roll no :- 19

MPL assignment 2.

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

→. 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 provide 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 seperate Android and IOS apps.

Key difference between PWA and Traditional Mobile Apps -

Feature	PWA	Traditional App
Installation	Direct from browser	Download from App store
Internet Required	Works offline with caching	Usually requires internet.

Performance	Fast with service workers	Usually Faster but needs installation.
Updates	Automatic, no app store approval.	Manual updates needed.
Development cost	lower (one codebase for all)	Higher (seperate apps for each platform)

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

9. 2. Define responsive web design and explain its importance.

→

Definition of Responsive Web design:

Responsive web design is a technique that makes web pages adjust automatically to different screen sizes and devices. It ensures a good user experience on mobile, to tablet and desktop without needing seperate versions.

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.

Comparison of Web design Approaches :-

Approach	How it works	Pros	Cons
Responsive	Uses flexible grids and CSS media queries.	Works on all devices	Can be complex to design
fluid	Uses percent based widths instead of fixed pixels so elements resize.	Works well on different screen size.	less control over layout on large screens

3) Describe the lifecycle of Service 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

1) Registration process :-

→ The browser registers the Service worker using JS.

eg :

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

2. Installation phase.

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

eg

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

3) Activation phase.

- The old service worker is replaced with the new one.

eg.

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

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

→ Use of IndexedDB in Service Worker 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.

Why Use IndexedDB in Service workers.
1. Offline support - stores data when offline.
2. Efficient storage - saves structured data.
3. Fast access - Retrieves data quickly

How Service Workers use IndexedDB?

Opening the database.

```
let db;
let request = indexedDB.open('MyDatabase', 1);
request.onsuccess = function(event) {
    db = event.target.result;
};
```

fetching data in Service Worker.

```
let transaction = db.transaction('Users', 'readonly');
let store = transaction.objectStore('Users');
```

```
var getUser = store1.get(1);

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