Prerequisites

Name: Sharvari Kishor More

Class: D15B

Roll no.: 35

Lifeline: App

Objective:
Lifeline is designed to assist users in finding the most suitable hospital based on their specific medical conditions, symptoms, or emergencies. The app focuses on providing personalized, real-time, and reliable information to users, ensuring timely and appropriate medical attention.

Key Features:

1. **Symptom Checker and Disease Mapping:**
    - Users can input symptoms using a search bar or voice command.
    - The app uses a robust AI-based symptom checker to map possible diseases and suggest hospitals equipped to handle those conditions.
2. **Emergency Mode:**
    - A dedicated feature for emergencies. Users can select the type of emergency (e.g., cardiac, trauma, stroke), and the app will immediately display nearby hospitals with the required facilities.
3. **Hospital Details and Specializations:**
    - Comprehensive profiles for hospitals, including:
        - Specializations (e.g., cardiology, neurology).
        - Facilities available (e.g., ICU, diagnostic tools, surgical expertise).
        - Real-time bed and doctor availability.
        - Accreditation and patient reviews.
4. **Localized Support:**
    - Supports multiple Indian languages for accessibility across diverse user groups.
5. **Cost Transparency:**
    - Displays approximate treatment costs for common procedures to help users make informed decisions.

Societal Impact:

1. Empathy for Users in Distress:

- You've placed yourself in the shoes of users facing medical emergencies or confusion about their symptoms.

2. Leveraging Technology for Good:

- As an IT engineering student, you're aware of the power of AI, data integration, and real-time technology in creating impactful solutions.

3. Learning from Existing Gaps (Practo's Limitations):

- By critically analyzing apps like Practo, you've understood their limitations and conceptualized a solution that overcomes them.

4. Desire to Save Lives and Build a Healthier Society:

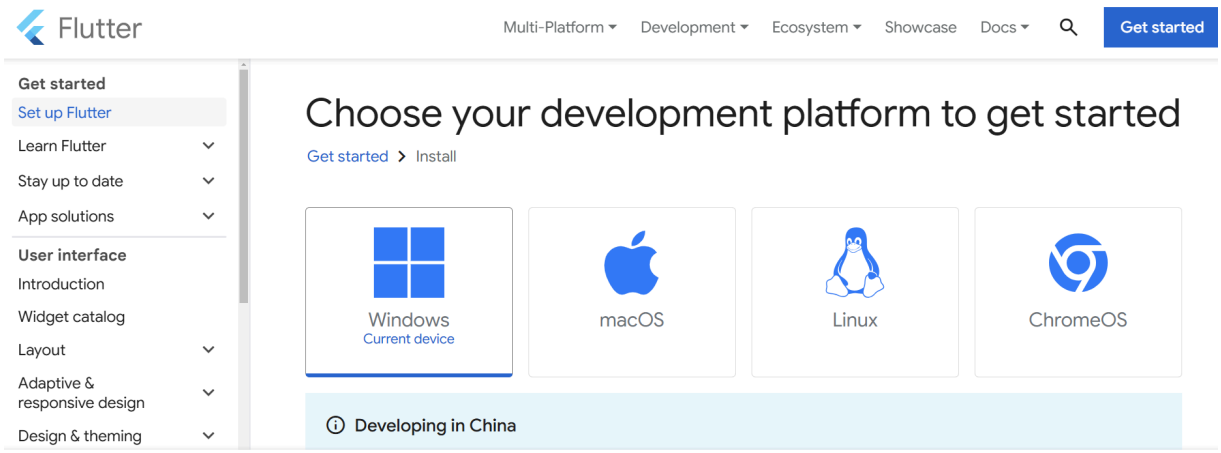
- You likely believe that every individual deserves access to the best possible healthcare resources, regardless of their location or situation.

Practical 1
Installation and Configuration of Flutter Environment

Name: Sharvari Kishor More
Class: D15B
Roll No.: 35

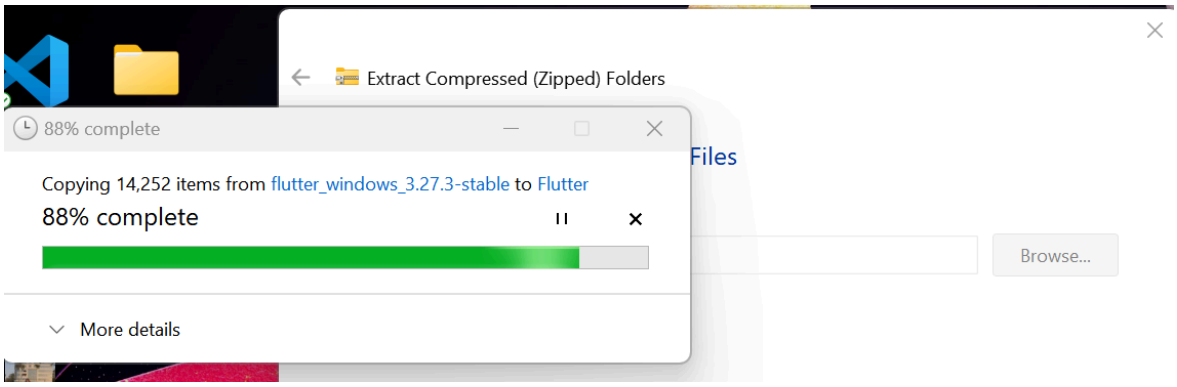
Aim: **Installation and Configuration of Flutter Environment**

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



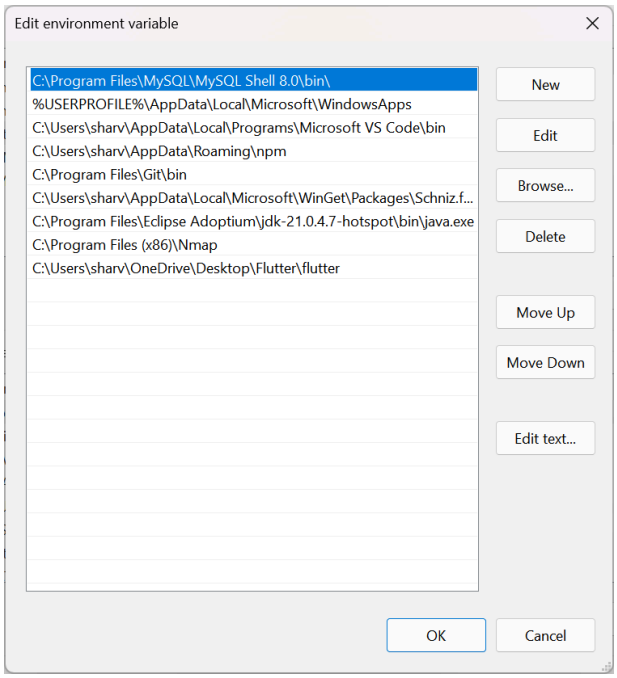
Step 2: Next, to download the latest Flutter SDK, click on the Windows icon. Here, you will find the download link for SDK.

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



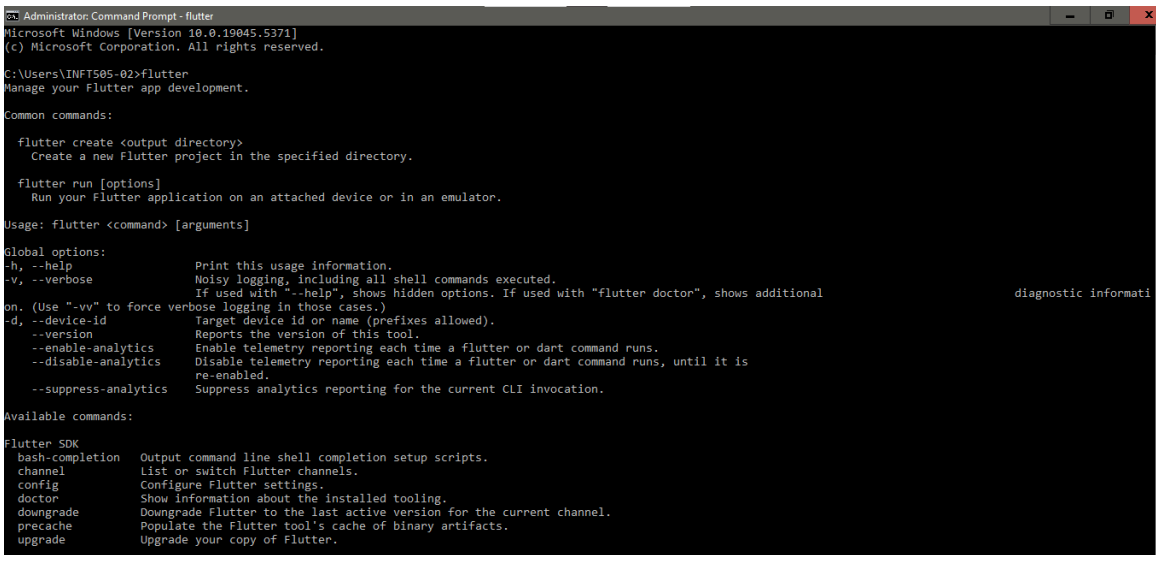
Step 4: To run the Flutter command in regular windows console, you need to update the system

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 5: Now, run the $ flutter command in 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.

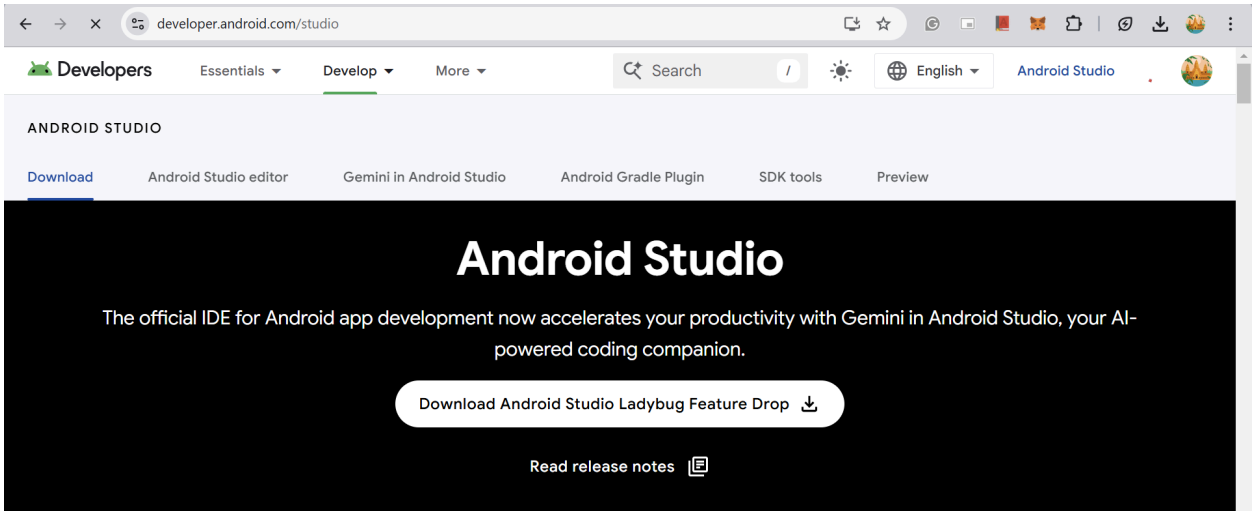


Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to

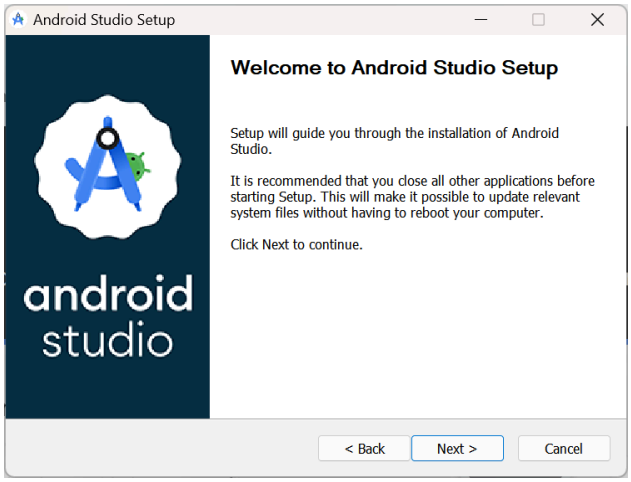
run Flutter as well as the development tools that are available but not connected with the device.

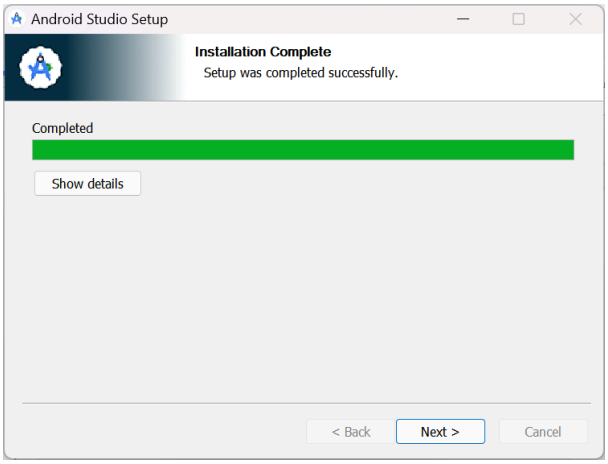
```
C:\Users\INFT505-02>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.27.3, on Microsoft Windows [Version 10.0.19045.5371], locale en-US)
[√] Windows Version (Installed version of Windows is version 10 or higher)
[√] Android toolchain - develop for Android devices (Android SDK version 33.0.1)
[√] Chrome - develop for the web
[X] Visual Studio - develop Windows apps
    X Visual Studio not installed; this is necessary to develop Windows apps.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++" workload, including all of its default components
[√] Android Studio (version 2021.3)
[√] VS Code (version 1.72.2)
[√] Connected device (3 available)
[√] Network resources

! Doctor found issues in 1 category.
```

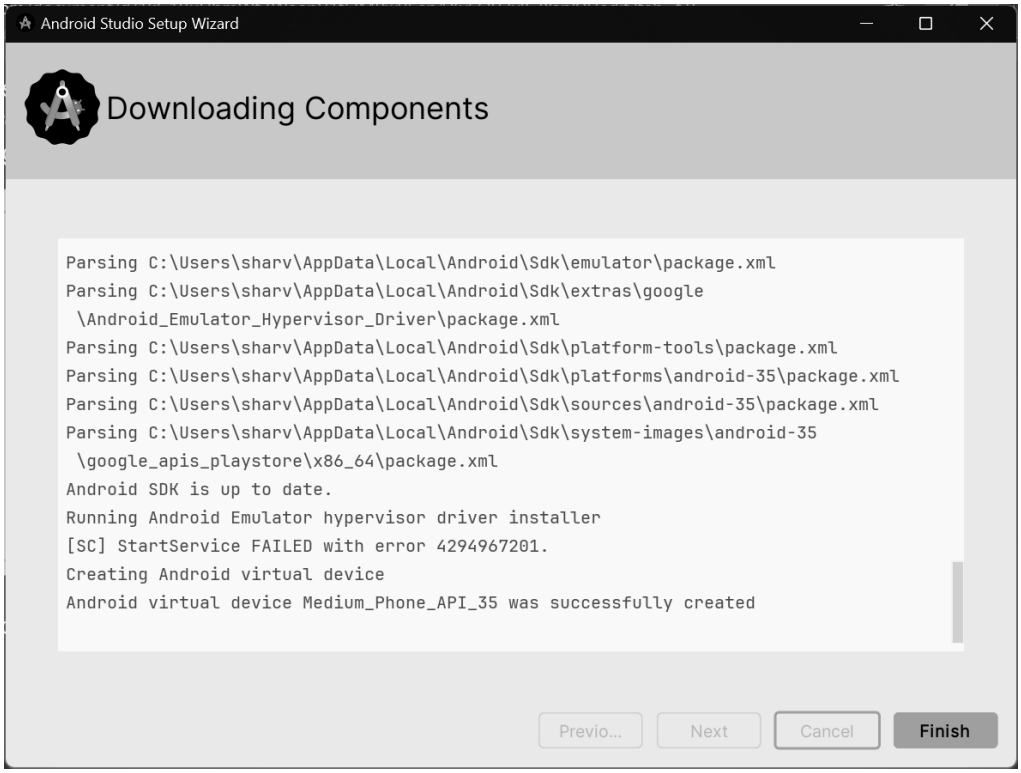
Step 7: Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.



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





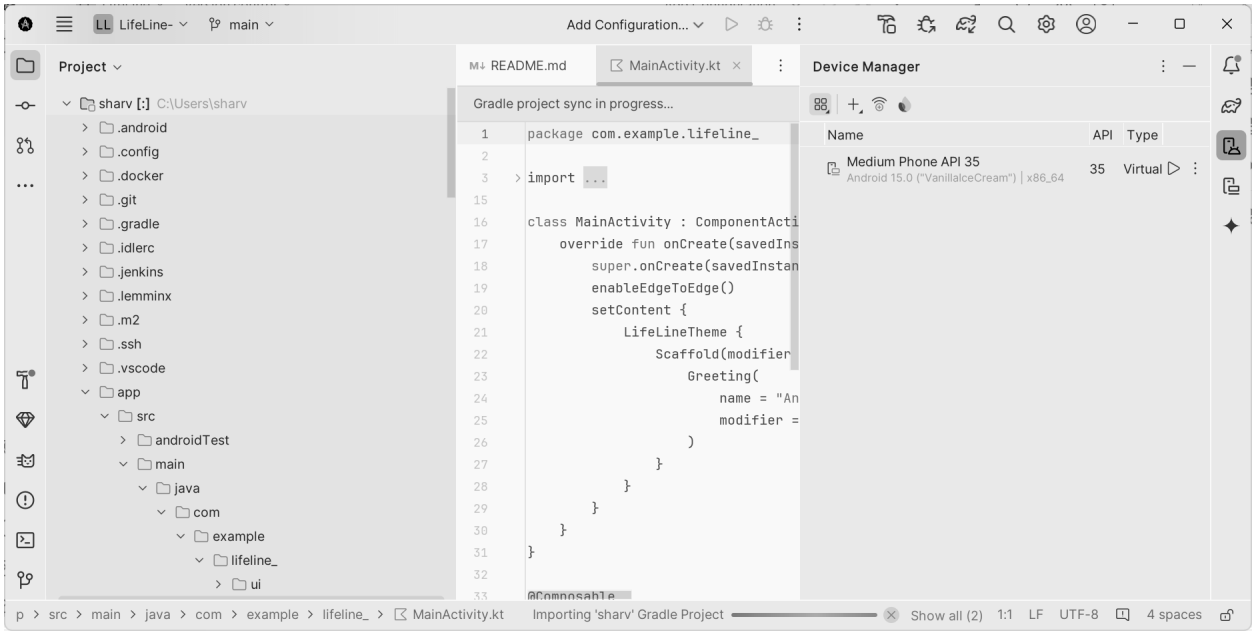


Step 7.5: run the $ flutter doctor command and Run flutter doctor --android-licenses command.

```
C:\Users\INFT505-02>flutter doctor --android-licenses
[========================               ] 61% Fetch remote repository...        etch remote repository...
Warning: Additionally, the fallback loader failed to parse the XML.
[============================           ] 73% Fetch remote repository...        etch remote repository...
Warning: Additionally, the fallback loader failed to parse the XML.
[======================================] 100% Computing updates...
All SDK package licenses accepted.
```

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

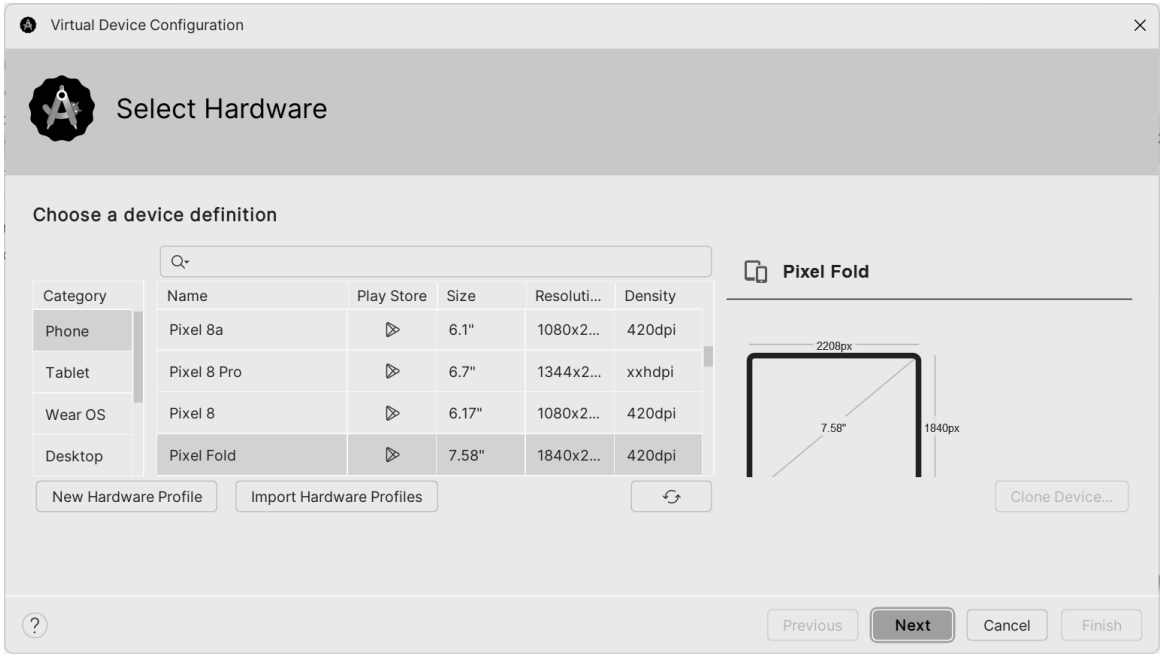
Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.



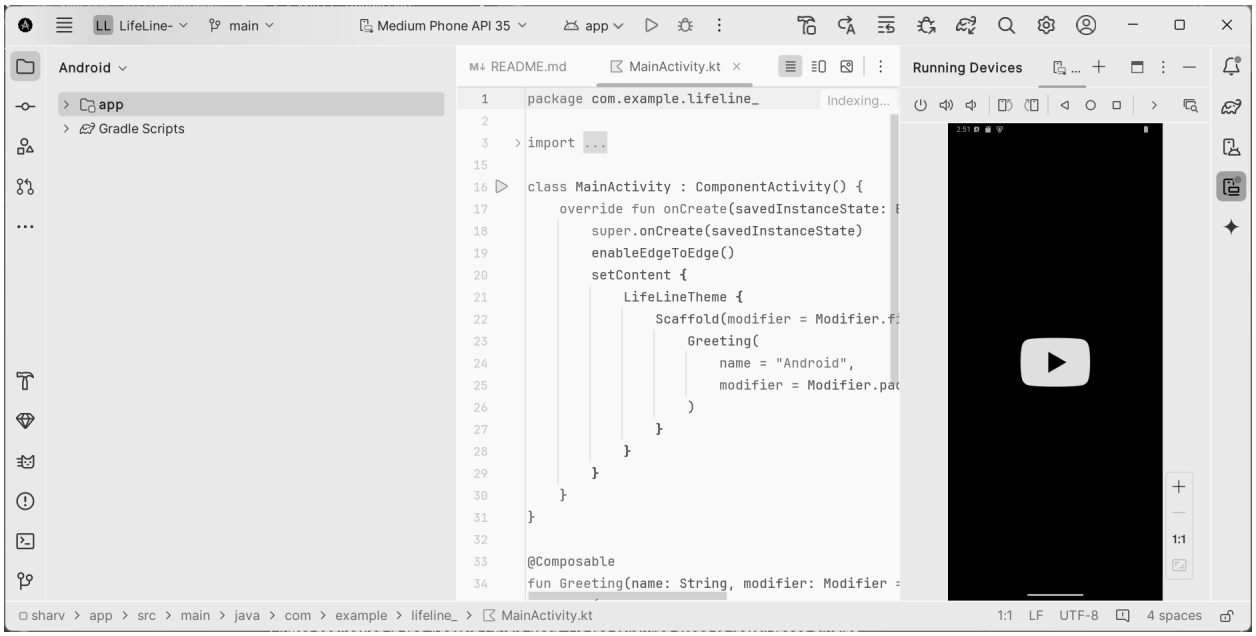
Step 8.2: Choose your device definition and click on Next.

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

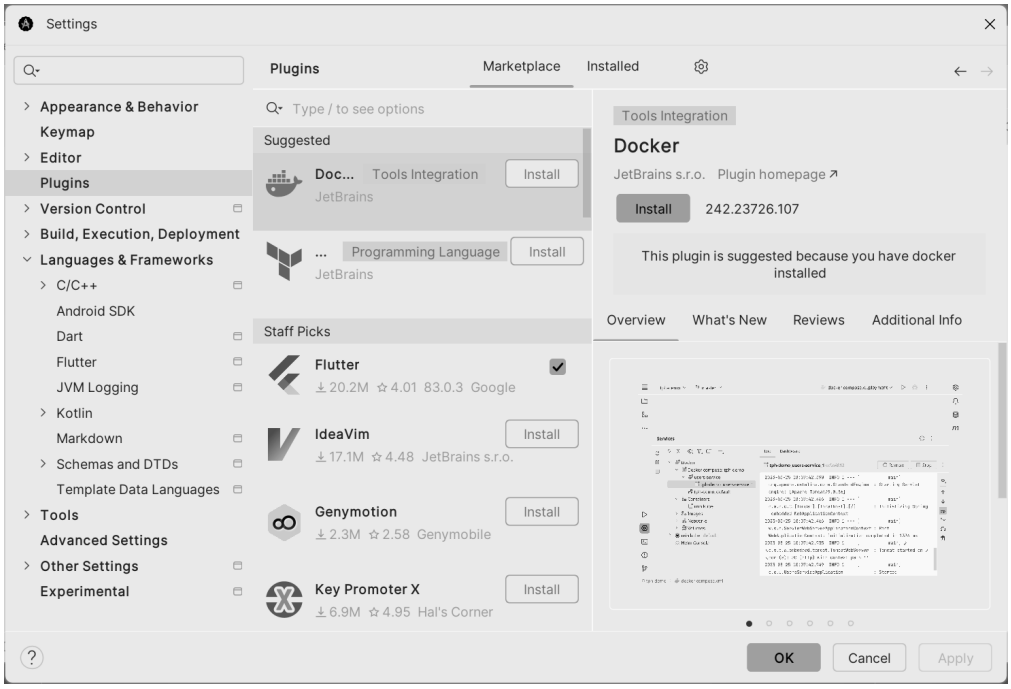
Step 8.4: Now, verify the all AVD configuration. If it is correct, click on Finish. The following screen appears.



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



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

Practical 2
Design a Flutter UI by including Common Widgets

Name: Sharvari Kishor More
Class: D15B
Roll No.: 35

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

Theory:

Introduction
Flutter is a popular framework for building cross-platform mobile applications. It uses a widget-based approach, where everything is a widget, making UI design flexible and efficient. Common widgets like AppBar, Container, TextField, GridView, and ListView help in creating responsive and visually appealing interfaces.

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

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

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

Implementation
In this experiment, we designed a Flutter UI for the Lifeline App, which helps users find hospitals based on their symptoms. The UI includes:

- App Bar: Displays the user's selected location and profile icon.
- Search Bar: Allows users to search for clinics.
- Appointment Section: Provides options for booking an in-clinic appointment or instant video consultation.
- Doctor Specialization Grid: Displays different medical specializations using GridView.
- Featured Services Section: Highlights important healthcare services.

We used StatelessWidget to create reusable components like AppointmentCard and DoctorCategoryCard. The UI follows a structured layout using Column, Row, and Padding for spacing and alignment.

CODE:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Lifeline',
      theme: ThemeData(
        primarySwatch: Colors.indigo,
      ),
      home: HomeScreen(),
    );
  }
}

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white,
      appBar: AppBar(
        backgroundColor: Colors.indigo,
        title: Row(
          children: [
            Icon(Icons.location_on, color: Colors.white),
            SizedBox(width: 5),
            Text("Bangalore", style: TextStyle(color: Colors.white)),
            Icon(Icons.keyboard_arrow_down, color: Colors.white),
          ],
        ),
        actions: [
          IconButton(
            icon: Icon(Icons.account_circle, color: Colors.white),
            onPressed: () {},
          ),
        ],
      ),
```

```
body: SingleChildScrollView(
  child: Padding(
    padding: EdgeInsets.all(16.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        // Search Bar
        Container(
          padding: EdgeInsets.symmetric(horizontal: 10),
          decoration: BoxDecoration(
            color: Colors.grey[200],
            borderRadius: BorderRadius.circular(10),
          ),
          child: TextField(
            decoration: InputDecoration(
              hintText: "Search for clinics",
              border: InputBorder.none,
              prefixIcon: Icon(Icons.search, color: Colors.grey),
            ),
          ),
        ),
        SizedBox(height: 20),

        // Appointment & Consultation
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            Expanded(
              child: AppointmentCard(
                title: "Book In-Clinic Appointment",
                imagePath: "assets/images/doctor_1.jpg",
              ),
            ),
            SizedBox(width: 10),
            Expanded(
              child: AppointmentCard(
                title: "Instant Video Consultation",
                imagePath: "assets/images/doctor_2.jpg",
              ),
            ),
          ],
        ),
        SizedBox(height: 20),
```

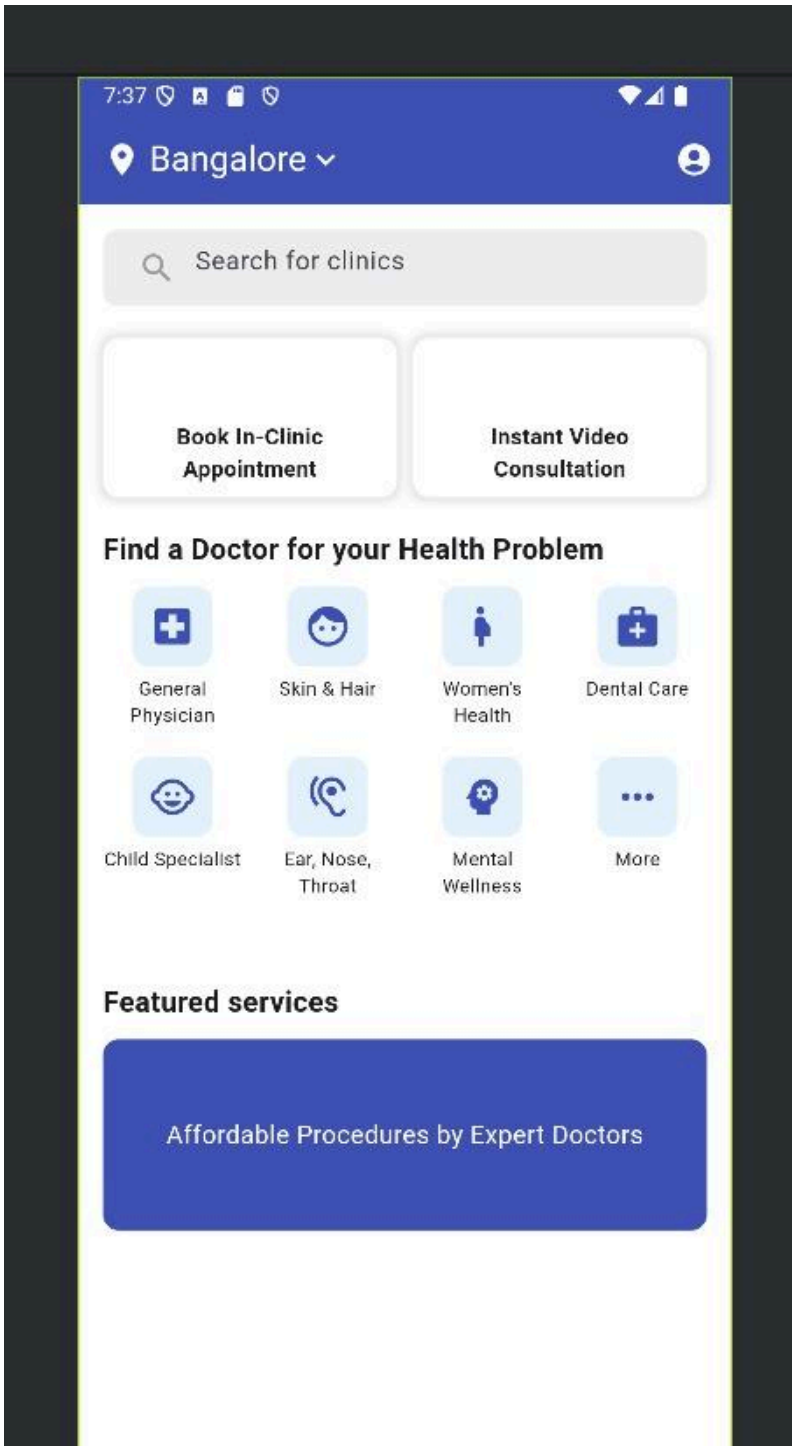
```
// Find a Doctor Section
Text(
  "Find a Doctor for your Health Problem",
  style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
),
SizedBox(height: 10),

// Categories
GridView.builder(
  shrinkWrap: true,
  physics: NeverScrollableScrollPhysics(),
  gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
    crossAxisCount: 4,
    crossAxisSpacing: 10,
    mainAxisSpacing: 10,
    childAspectRatio: 0.9,
  ),
  itemCount: doctorCategories.length,
  itemBuilder: (context, index) {
    return DoctorCategoryCard(doctorCategories[index]);
  },
),

SizedBox(height: 20),

// Featured Services
Text(
  "Featured services",
  style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
),
SizedBox(height: 10),

Container(
  width: double.infinity,
  height: 120,
  decoration: BoxDecoration(
    color: Colors.indigo,
    borderRadius: BorderRadius.circular(10),
  ),
  child: Center(
    child: Text(
      "Affordable Procedures by Expert Doctors",
      style: TextStyle(color: Colors.white, fontSize: 16),
      textAlign: TextAlign.center,
```

```
                ),
              ),
            ),
          ],
        ),
      ),
    ),
   );
  }
}

class AppointmentCard extends StatelessWidget {
  final String title;
  final String imagePath;

  AppointmentCard({required this.title, required this.imagePath});

  @override
  Widget build(BuildContext context) {
   return Container(
     height: 100,
     decoration: BoxDecoration(
       color: Colors.white,
       borderRadius: BorderRadius.circular(10),
       boxShadow: [
         BoxShadow(
           color: Colors.grey.shade300,
           blurRadius: 5,
           spreadRadius: 2,
         ),
       ],
     ),
     child: Column(
       children: [
         Expanded(
           child: Image.asset(imagePath, fit: BoxFit.cover),
         ),
         Padding(
           padding: const EdgeInsets.all(8.0),
           child: Text(
             title,
             textAlign: TextAlign.center,
             style: TextStyle(fontSize: 14, fontWeight: FontWeight.bold),
           ),
```

```
          ),
        ],
      ),
    );
  }
}

class DoctorCategoryCard extends StatelessWidget {
  final DoctorCategory category;

  DoctorCategoryCard(this.category);

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Container(
          padding: EdgeInsets.all(10),
          decoration: BoxDecoration(
            color: Colors.blue[50],
            borderRadius: BorderRadius.circular(10),
          ),
          child: Icon(category.icon, color: Colors.indigo, size: 30),
        ),
        SizedBox(height: 5),
        Text(category.title, textAlign: TextAlign.center, style: TextStyle(fontSize: 12)),
      ],
    );
  }
}

class DoctorCategory {
  final String title;
  final IconData icon;

  DoctorCategory({required this.title, required this.icon});
}

List<DoctorCategory> doctorCategories = [
    DoctorCategory(title: "General Physician", icon: Icons.local_hospital),  // Alternative for
stethoscope
  DoctorCategory(title: "Skin & Hair", icon: Icons.face),
  DoctorCategory(title: "Women's Health", icon: Icons.pregnant_woman),
  DoctorCategory(title: "Dental Care", icon: Icons.medical_services),
```

```
  DoctorCategory(title: "Child Specialist", icon: Icons.child_care),
  DoctorCategory(title: "Ear, Nose, Throat", icon: Icons.hearing),
  DoctorCategory(title: "Mental Wellness", icon: Icons.psychology),
  DoctorCategory(title: "More", icon: Icons.more_horiz),
];
```

Screenshot:



Conclusion:

In this experiment, we implemented a Flutter UI for the Lifeline App using common widgets like AppBar, GridView, and Column. We initially faced an issue with the Icons.stethoscope not being available, which we resolved by replacing it with alternative icons like Icons.local_hospital and Icons.medical_services.

Experiment 3
Flutter Widgets

Name: Sharvari Kishor More
Class: D15B
Roll no.: 35

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

Introduction
Mobile Application Development (MAD) involves designing and implementing applications that run on mobile devices. Flutter, an open-source UI software development kit by Google, is widely used for building natively compiled applications for mobile, web, and desktop from a single codebase.

Objective
The goal of this experiment is to design and develop a healthcare application interface using Flutter. This involves adding essential UI components such as buttons, icons, a search bar, and a dropdown list to enhance user interaction.

Concepts Covered
1. **Scaffold & AppBar:** Used to create the structure of the mobile application, including the navigation bar and main screen layout.
2. **Dropdown Button:** Allows users to select their city from a predefined list.
3. **Search Bar:** Enables users to search for clinics within the application.
4. **Buttons & Icons:** Enhance UI interactions, such as booking appointments and navigating the app.
5. **Grid View:** Displays doctor categories using an interactive grid layout.
6. **Custom Widgets:** Implemented reusable widgets for appointment booking and doctor categories.

Implementation
In this experiment, we implemented a Flutter UI with the following key features:
- Dropdown Menu in AppBar: Allows users to select a city.
- Search Bar Widget: Users can search for clinics conveniently.
- Appointment Booking Cards: Users can book in-clinic and video consultations.
- Doctor Categories in Grid View: Displays available medical specialties using interactive buttons.
- Profile Icon & Featured Services Section: Enhances UI usability and provides quick access to important features.

Code:

home_screen.dart

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

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

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

class _HomeScreenState extends State<HomeScreen> {
  String selectedCity = "Bangalore";
  final List<String> cities = ["Bangalore", "Pune", "Nashik",
"Mumbai"];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white,
      appBar: AppBar(
        backgroundColor: Colors.indigo,
        title: Row(
          children: [
            Icon(Icons.location_on, color: Colors.white),
            SizedBox(width: 5),
            DropdownButton<String>(
              value: selectedCity,
              dropdownColor: Colors.indigo,
              icon: Icon(Icons.keyboard_arrow_down, color:
Colors.white),
              underline: SizedBox(),
              style: TextStyle(color: Colors.white, fontSize: 16),
              onChanged: (String? newValue) {
                setState(() {
                  selectedCity = newValue!;
                });
              },
```

```
              items: cities.map<DropdownMenuItem<String>>((String
city) {
                return DropdownMenuItem<String>(
                  value: city,
                  child: Text(city, style: TextStyle(color:
Colors.white)),
                );
              }).toList(),
            ),
          ],
        ),
        actions: [
          IconButton(
            icon: Icon(Icons.account_circle, color: Colors.white),
            onPressed: () {},
          ),
        ],
      ),
      body: SingleChildScrollView(
        child: Padding(
          padding: EdgeInsets.all(16.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Container(
                padding: EdgeInsets.symmetric(horizontal: 10),
                decoration: BoxDecoration(
                  color: Colors.grey[200],
                  borderRadius: BorderRadius.circular(10),
                ),
                child: TextField(
                  decoration: InputDecoration(
                    hintText: "Search for clinics",
                    border: InputBorder.none,
                    prefixIcon: Icon(Icons.search, color:
Colors.grey),
                  ),
                ),
```

```
            ),
            SizedBox(height: 20),
            // Appointment & Consultation Section
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                Expanded(
                  child: AppointmentCard(
                    title: "Book In-Clinic Appointment",
                    imagePath: "assets/images/images.jpg",
                  ),
                ),
                SizedBox(width: 10),
                Expanded(
                  child: AppointmentCard(
                    title: "Instant Video Consultation",
                    imagePath: "assets/images/hospital.jpg",
                  ),
                ),
              ],
            ),
            SizedBox(height: 20),
            Text(
              "Find a Doctor for your Health Problem",
              style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold),
            ),
            SizedBox(height: 10),
            GridView.builder(
              shrinkWrap: true,
              physics: NeverScrollableScrollPhysics(),
              gridDelegate:
SliverGridDelegateWithFixedCrossAxisCount(
                crossAxisCount: 4,
                crossAxisSpacing: 10,
                mainAxisSpacing: 10,
                childAspectRatio: 0.9,
              ),
```

```
                itemCount: doctorCategories.length,
                itemBuilder: (context, index) {
                  return
DoctorCategoryButton(doctorCategories[index]);
                },
              ),
              SizedBox(height: 20),
              Text(
                "Featured services",
                style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold),
              ),
              SizedBox(height: 10),
              Container(
                width: double.infinity,
                height: 120,
                decoration: BoxDecoration(
                  color: Colors.indigo,
                  borderRadius: BorderRadius.circular(10),
                ),
                child: Center(
                  child: Text(
                    "Affordable Procedures by Expert Doctors",
                    style: TextStyle(color: Colors.white, fontSize:
16),
                    textAlign: TextAlign.center,
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}

class AppointmentCard extends StatelessWidget {
```

```
  final String title;
  final String imagePath;

  const AppointmentCard({
    required this.title,
    required this.imagePath,
    super.key,
  });

  @override
  Widget build(BuildContext context) {
    return Container(
      height: 120,
      decoration: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.circular(10),
        boxShadow: [
          BoxShadow(color: Colors.grey.shade300, blurRadius: 5,
spreadRadius: 2),
        ],
      ),
      child: Column(
        children: [
          Expanded(
            child: ClipRRect(
              borderRadius: BorderRadius.vertical(top:
Radius.circular(10)),
              child: Image.asset(imagePath, width: double.infinity,
fit: BoxFit.cover),
            ),
          ),
          Padding(
            padding: EdgeInsets.all(8.0),
            child: Text(title, textAlign: TextAlign.center, style:
TextStyle(fontSize: 14, fontWeight: FontWeight.bold)),
          ),
        ],
      ),
```

```
    );
  }
}

class DoctorCategoryButton extends StatelessWidget {
  final DoctorCategory category;
  const DoctorCategoryButton(this.category, {super.key});

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text("Selected: ${category.title}")),
        );
      },
      child: Column(
        children: [
          Container(
            padding: EdgeInsets.all(10),
            decoration: BoxDecoration(
              color: Colors.blue[50],
              borderRadius: BorderRadius.circular(10),
            ),
            child: Icon(category.icon, color: Colors.indigo, size:
30),
          ),
          SizedBox(height: 5),
          Text(category.title, textAlign: TextAlign.center, style:
TextStyle(fontSize: 12)),
        ],
      ),
    );
  }
}

class DoctorCategory {
  final String title;
```

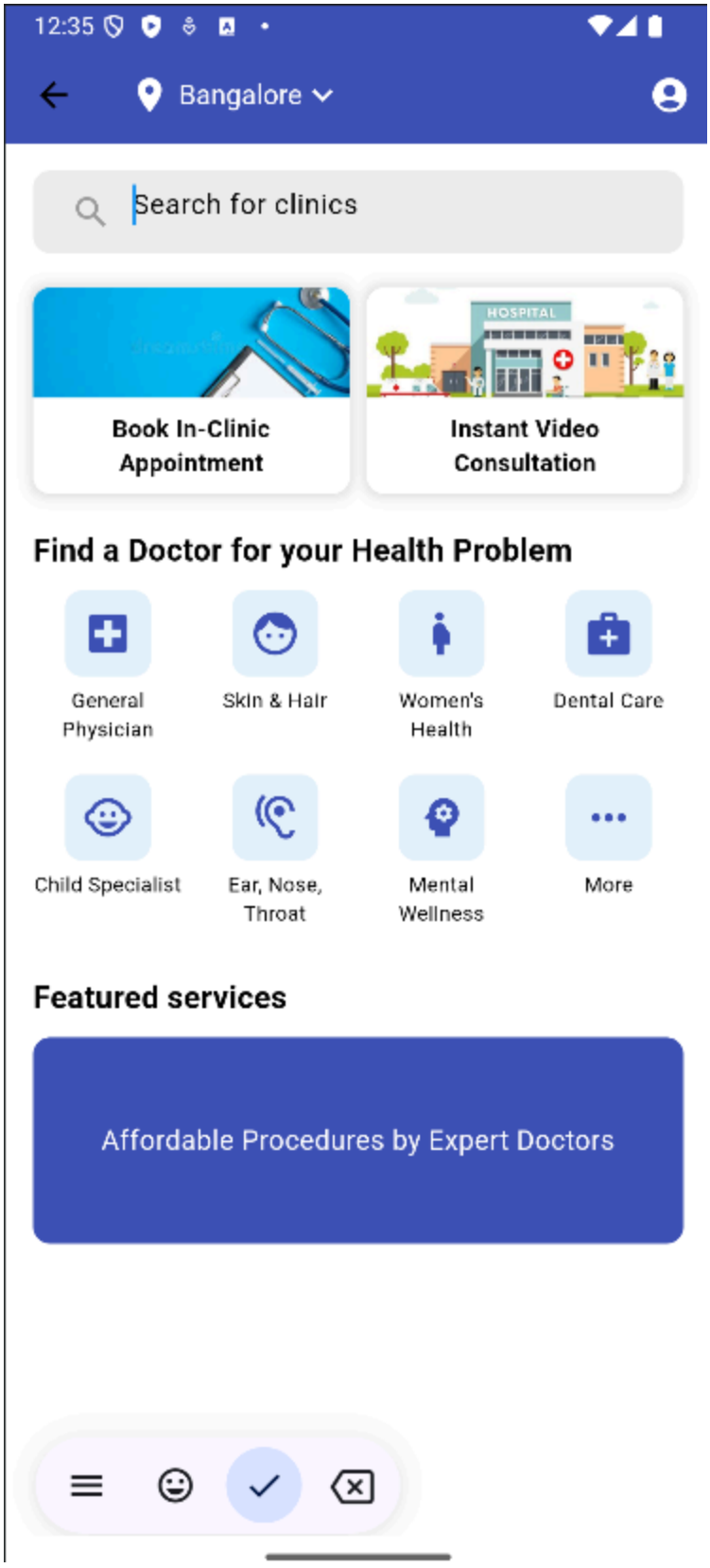
```
  final IconData icon;
  DoctorCategory({required this.title, required this.icon});
}

List<DoctorCategory> doctorCategories = [
  DoctorCategory(title: "General Physician", icon:
Icons.local_hospital),
  DoctorCategory(title: "Skin & Hair", icon: Icons.face),
  DoctorCategory(title: "Women's Health", icon:
Icons.pregnant_woman),
  DoctorCategory(title: "Dental Care", icon: Icons.medical_services),
  DoctorCategory(title: "Child Specialist", icon: Icons.child_care),
  DoctorCategory(title: "Ear, Nose, Throat", icon: Icons.hearing),
  DoctorCategory(title: "Mental Wellness", icon: Icons.psychology),
  DoctorCategory(title: "More", icon: Icons.more_horiz),
];
```

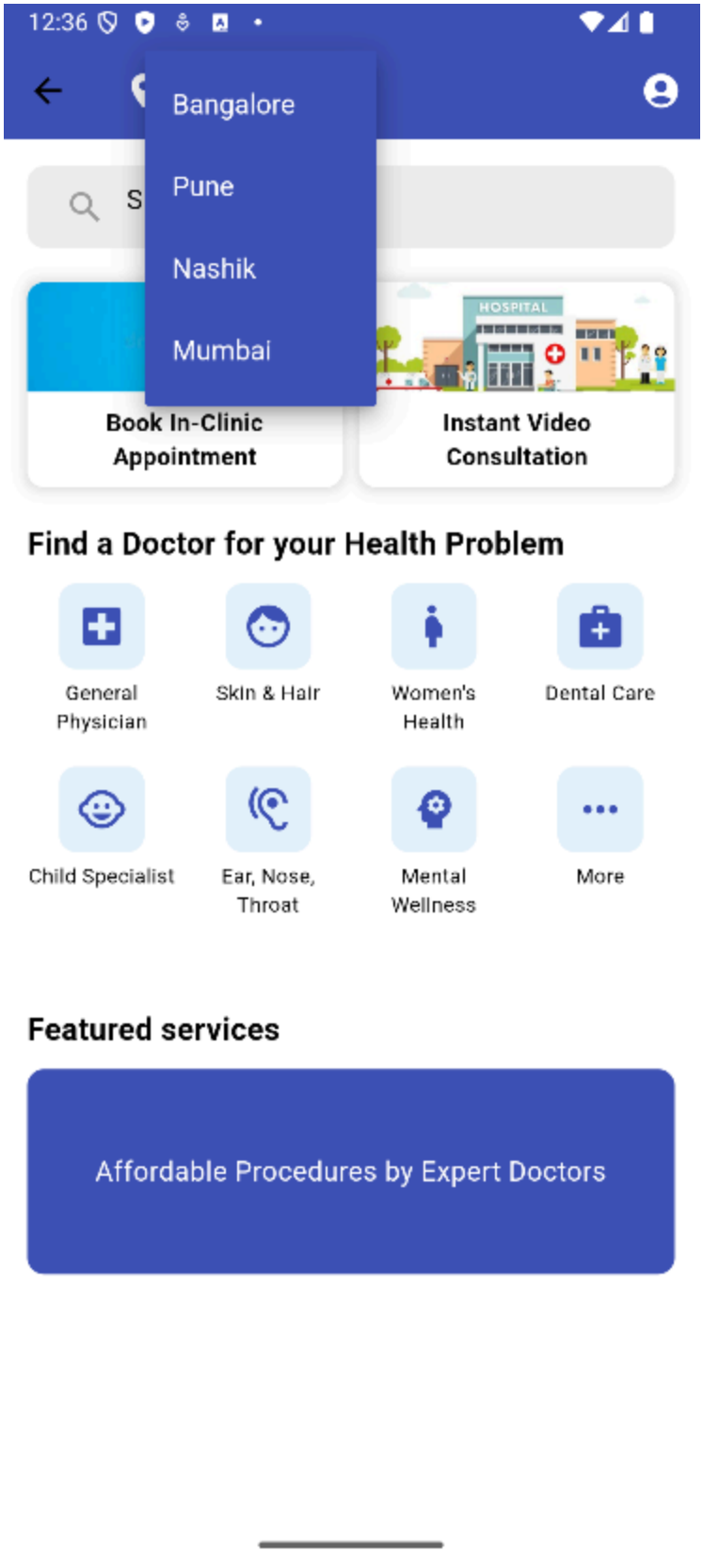
Code:

main.dart

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

void main() {
  runApp(const LifelineApp()); // Ensure const is used if applicable
}

class LifelineApp extends StatelessWidget {
  const LifelineApp({super.key}); // Add key parameter for best
practice

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Lifeline',
      theme: ThemeData(
```

```
        primaryColor: Colors.green,
        colorScheme: ColorScheme.fromSwatch().copyWith(secondary:
Colors.blue),
      ),
      initialRoute: '/login',
      routes: {
        '/login': (context) => LoginScreen(),
        '/signup': (context) => SignupScreen(),
        '/home': (context) => HomeScreen(),
      },
    );
  }
}
```


12:35
Bangalore
Search for clinics
Book In-Clinic Appointment
HOSPITAL
Instant Video Consultation
Find a Doctor for your Health Problem
General Physician
Skin & Hair
Women's Health
Dental Care
Child Specialist
Ear, Nose, Throat
Mental Wellness
More
Featured services
Affordable Procedures by Expert Doctors


12:36
Bangalore
Pune
Nashik
Mumbai
HOSPITAL
Book In-Clinic Appointment
Instant Video Consultation
Find a Doctor for your Health Problem
General Physician
Skin & Hair
Women's Health
Dental Care
Child Specialist
Ear, Nose, Throat
Mental Wellness
More
Featured services
Affordable Procedures by Expert Doctors

Conclusion:
In this experiment, we successfully implemented various UI components such as a dropdown list, search bar, buttons, and icons to enhance the application's usability. During development, we encountered minor issues such as dropdown styling inconsistencies and widget alignment problems, which were resolved through debugging, modifying the widget properties, and adjusting padding and alignment settings.

Experiment 4
Form Widgets

Name: Sharvari Kishor More
Class: D15B
Roll no.: 35

Aim: To create an interactive Form using form widget.

Introduction:
Mobile Application Development (MAD) focuses on creating applications that run on mobile devices, utilizing frameworks such as Flutter. We implemented fundamental concepts of mobile app development, including user input handling, validation, and navigation between screens. The primary objective was to enhance the user experience by implementing structured forms and ensuring seamless transitions between different pages.

General Overview:
In mobile app development, forms play a crucial role in collecting user input. A well-designed form includes input fields, validation checks, and submission mechanisms to ensure data integrity. TextFormField widgets in Flutter allow validation logic to be embedded directly within the UI components. Error messages guide users in entering correct information, ensuring smooth app interactions. Additionally, navigation between screens is essential for multi-page applications, achieved using Navigator methods in Flutter.

Implementation Details:
In this experiment, we designed a Login and Signup Screen using Flutter. The user interface was enhanced with a background image, translucent text fields, and a clean layout. A GlobalKey was used to manage the form state, ensuring proper validation of input fields.

- **Login Screen:** It includes email and password fields with validation logic. The password field has a toggle feature for visibility control. Upon successful validation, the user is redirected to the home screen.
- **Signup Screen:** It collects user details like name, email, and password. The password validation enforces security constraints such as length, special characters, and numeric values.
- **Navigation & UI Enhancements:** A visually appealing design was implemented, maintaining consistency in color schemes (blue, light blue, and white). The Navigator class was used to manage screen transitions seamlessly.

Code:
Login_screen.dart

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

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

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

class LoginScreenState extends State<LoginScreen> {
  final _formKey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  bool _obscurePassword = true;

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        fit: StackFit.expand,
        children: [
          Image.asset("assets/images/dr1.jpg", fit: BoxFit.cover),
          Container(color: Colors.black.withOpacity(0.4)),
          Center(
            child: Padding(
              padding: EdgeInsets.all(20),
              child: Form(
                key: _formKey,
                child: Column(
```

```
                mainAxisSize: MainAxisSize.min,
                children: [
                  Text("Welcome to Lifeline",
                      style: TextStyle(fontSize: 24, fontWeight:
FontWeight.bold, color: Colors.white)),
                  SizedBox(height: 20),
                  _buildTextField(_emailController, "Email",
Icons.email, false),
                  SizedBox(height: 10),
                  _buildTextField(_passwordController, "Password",
Icons.lock, true),
                  SizedBox(height: 20),
                  ElevatedButton(
                    style: ElevatedButton.styleFrom(backgroundColor:
Colors.blue),
                    onPressed: () {
                      if (_formKey.currentState!.validate()) {
                        Navigator.pushReplacement(context,
MaterialPageRoute(builder: (context) => HomeScreen()));
                      }
                    },
                    child: Text("Login", style: TextStyle(color:
Colors.white)),
                  ),
                  TextButton(
                    onPressed: () => Navigator.pushNamed(context,
'/signup'),
                    child: Text("Don't have an account? Sign Up", style:
TextStyle(color: Colors.white)),
                  ),
                ],
              ),
            ),
          ),
        ],
      ),
    );
  }
```

```
  Widget _buildTextField(TextEditingController controller, String label,
IconData icon, bool isPassword) {
    return TextFormField(
      controller: controller,
      obscureText: isPassword ? _obscurePassword : false,
      style: TextStyle(color: Colors.white),
      decoration: InputDecoration(
        labelText: label,
        labelStyle: TextStyle(color: Colors.white70),
        prefixIcon: Icon(icon, color: Colors.white),
        filled: true,
        fillColor: Colors.white.withOpacity(0.2),
        border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10)),
        suffixIcon: isPassword
            ? IconButton(
                icon: Icon(_obscurePassword ? Icons.visibility_off :
Icons.visibility, color: Colors.white),
                onPressed: () => setState(() => _obscurePassword =
!_obscurePassword),
              )
            : null,
      ),
      validator: (value) => value == null || value.isEmpty ? "$label
cannot be empty" : null,
    );
  }
}
```

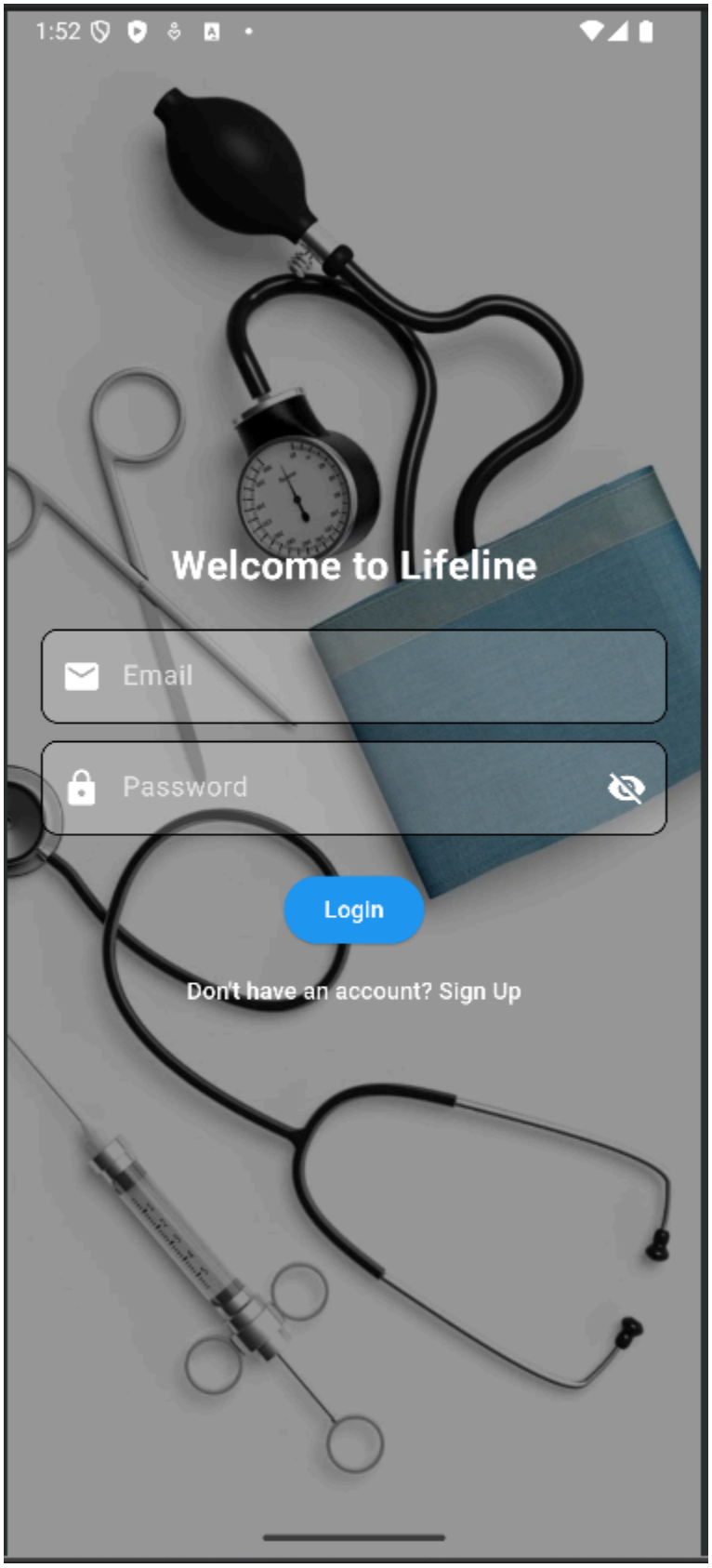
Signup_screen.dart

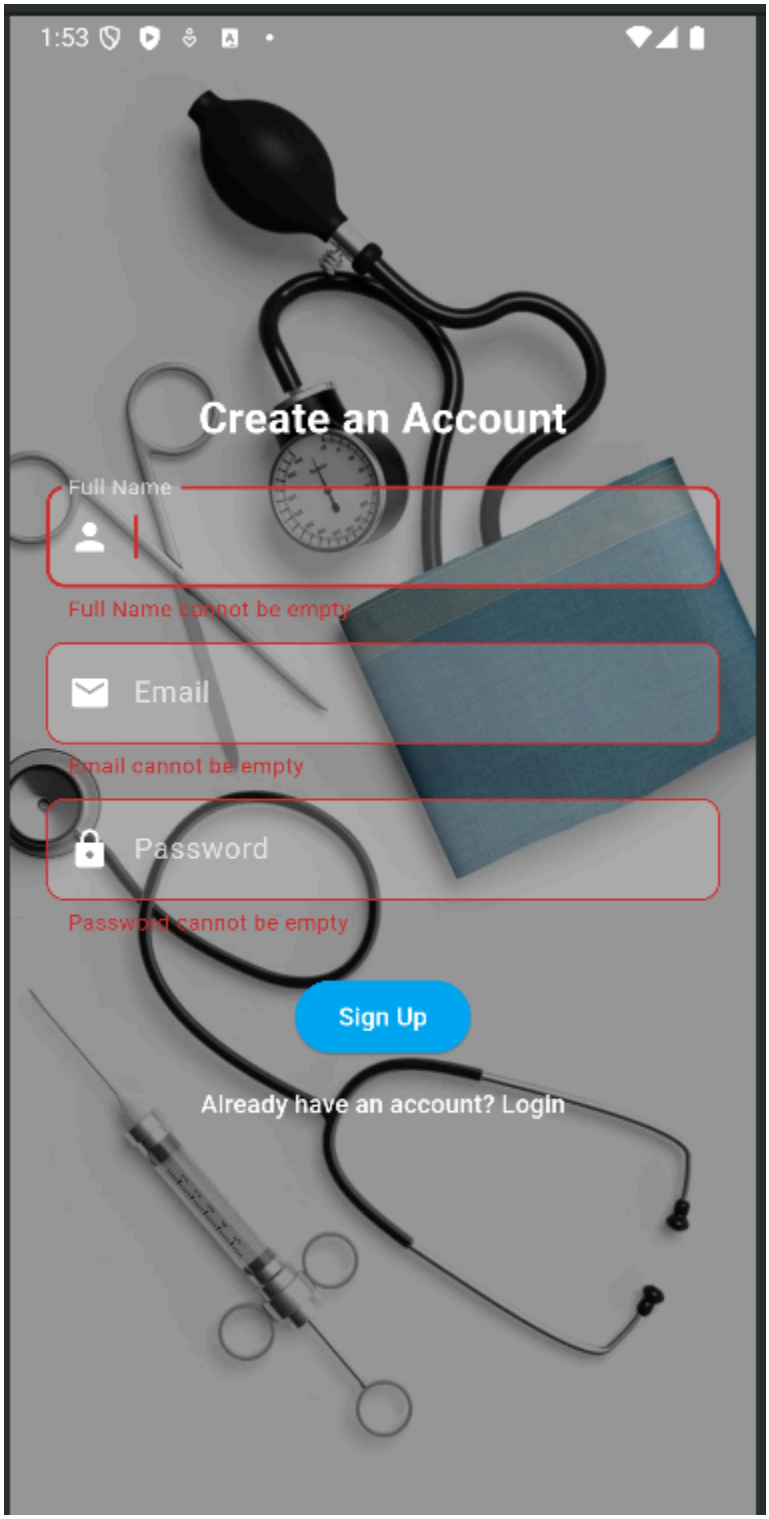
```
import 'package:flutter/material.dart';

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

  @override
  Widget build(BuildContext context) {
    final _formKey = GlobalKey<FormState>();
    final _nameController = TextEditingController();
    final _emailController = TextEditingController();
```

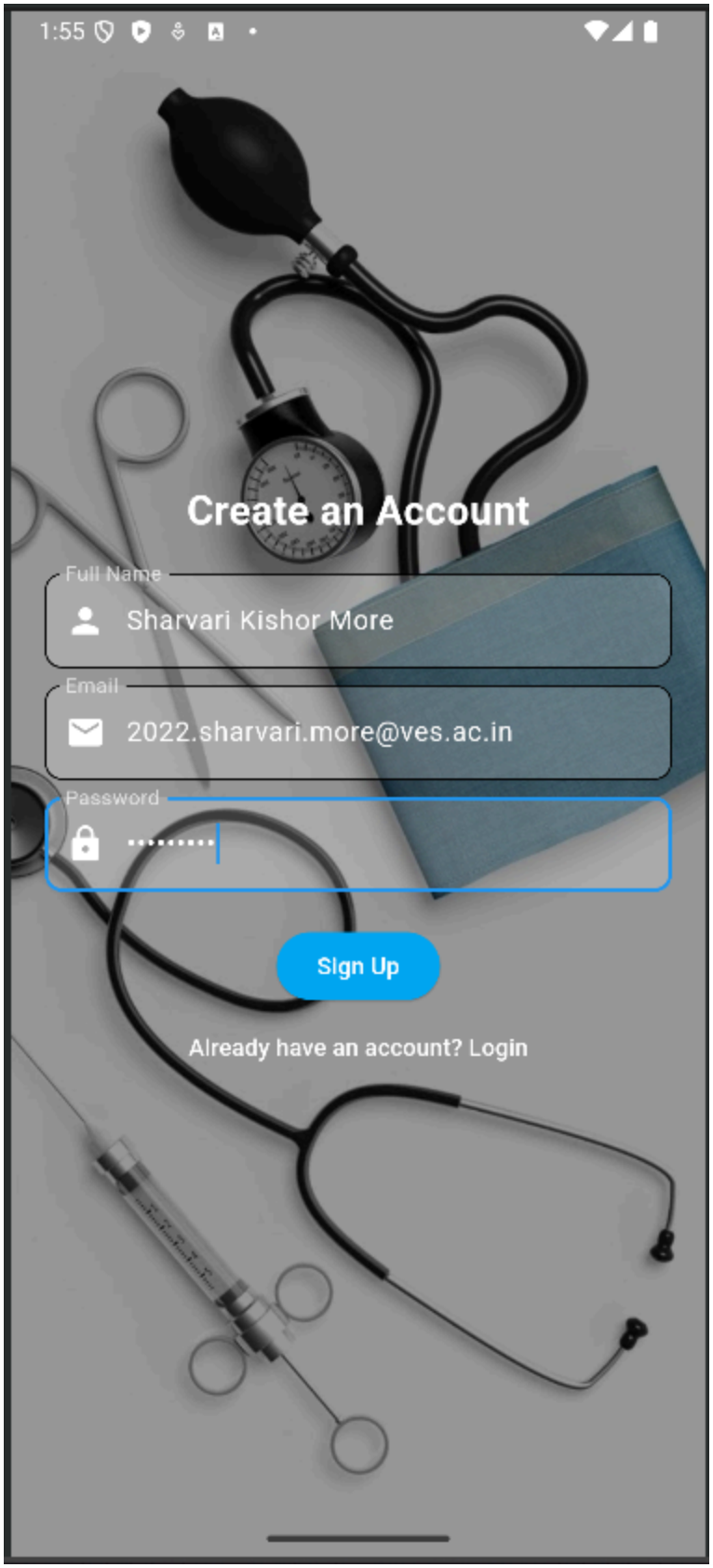
```
    final _passwordController = TextEditingController();

    return Scaffold(
      body: Stack(
        fit: StackFit.expand,
        children: [
          Image.asset("assets/images/dr1.jpg", fit: BoxFit.cover),
          Container(color: Colors.black.withOpacity(0.4)),
          Center(
            child: Padding(
              padding: EdgeInsets.all(20),
              child: Form(
                key: _formKey,
                child: Column(
                  mainAxisSize: MainAxisSize.min,
                  children: [
                    Text("Create an Account", style: TextStyle(fontSize:
24, fontWeight: FontWeight.bold, color: Colors.white)),
                    SizedBox(height: 20),
                    _buildTextField(_nameController, "Full Name",
Icons.person, false),
                    SizedBox(height: 10),
                    _buildTextField(_emailController, "Email",
Icons.email, false),
                    SizedBox(height: 10),
                    _buildTextField(_passwordController, "Password",
Icons.lock, true),
                    SizedBox(height: 20),
                    ElevatedButton(
                      style: ElevatedButton.styleFrom(backgroundColor:
Colors.lightBlue),
                      onPressed: () {
                        if (_formKey.currentState!.validate()) {
                          Navigator.pushReplacementNamed(context,
'/home');
                        }
                      },
                      child: Text("Sign Up",style: TextStyle(color:
Colors.white)),
                    ),
```

```
                    TextButton(
                      onPressed: () => Navigator.pushNamed(context,
'/login'),
                      child: Text("Already have an account? Login", style:
TextStyle(color: Colors.white)),
                    ),
                  ],
                ),
              ),
            ),
          ),
        ],
      ),
    );
  }

  Widget _buildTextField(TextEditingController controller, String label,
IconData icon, bool isPassword) {
    return TextFormField(
      controller: controller,
      obscureText: isPassword,
      style: TextStyle(color: Colors.white),
      decoration: InputDecoration(
        labelText: label,
        labelStyle: TextStyle(color: Colors.white70),
        prefixIcon: Icon(icon, color: Colors.white),
        filled: true,
        fillColor: Colors.white.withOpacity(0.2),
        border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10)),
      ),
      validator: (value) => value == null || value.isEmpty ? "$label
cannot be empty" : null,
    );
  }
}
```

Output:


1:52
Welcome to Lifeline
Email
Password
LogIn
Don't have an account? Sign Up


1:53
Create an Account
Full Name
Full Name cannot be empty
Email
Email cannot be empty
Password
Password cannot be empty
Sign Up
Already have an account? Login


1:55
Create an Account
Full Name
Sharvari Kishor More
Email
2022.sharvari.more@ves.ac.in
Password
Sign Up
Already have an account? Login

Conclusion:
This experiment strengthened our understanding of form validation, state management, and navigation in Flutter. We encountered challenges such as handling input errors and UI design consistency, which were resolved through structured debugging and iterative design improvements.

Experiment 5
Navigation , Routing and Gestures

Name: Sharvari Kishor More
Class: D15B
Roll no.: 35

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

Theory:

Flutter provides powerful tools to implement user interactions, navigation, and routing in mobile applications. This experiment focuses on three key concepts:

1. **Gestures**: Gestures allow users to interact with the UI using touch-based actions like tapping, swiping, and long pressing. Flutter provides the GestureDetector widget to handle such interactions.
2. **Navigation**: Navigation is the process of switching between different screens (pages) within an application. Flutter's Navigator class manages the navigation stack and allows users to move between screens using push and pop methods.
3. **Routing**: Routing is a way to define and manage the different screens in an application. Named routes can be used to simplify navigation and organize the app's structure better.

Implementation in the Given Code

In this experiment, we have implemented gestures, navigation, and routing as follows:

- **Gestures**: The **DoctorCategoryButton** widget uses **GestureDetector** to handle user interactions. When a user taps on a category, a SnackBar displays the selected category.
- **Navigation**: The login and signup pages utilize navigation to move between screens using the **Navigator.push() method.**
- **Routing**: The main.dart file contains the routing logic, which helps in switching between different pages smoothly.

Code:
Home_screen.dart

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

class HomeScreen extends StatefulWidget {

  const HomeScreen({super.key});

  @override

  _HomeScreenState createState() => _HomeScreenState();

}

class _HomeScreenState extends State<HomeScreen> {

  String selectedCity = "Bangalore";

  final List<String> cities = ["Bangalore", "Pune", "Nashik", "Mumbai"];

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      backgroundColor: Colors.white,

      appBar: AppBar(

        backgroundColor: Colors.indigo,

        title: Row(

          children: [

            Icon(Icons.location_on, color: Colors.white),

            SizedBox(width: 5),

            DropdownButton<String>(

              value: selectedCity,

              dropdownColor: Colors.indigo,
```

```
              icon: Icon(Icons.keyboard_arrow_down, color: Colors.white),
              underline: SizedBox(),
              style: TextStyle(color: Colors.white, fontSize: 16),
              onChanged: (String? newValue) {
                setState(() {
                  selectedCity = newValue!;
                });
              },
              items: cities.map<DropdownMenuItem<String>>((String city) {
                return DropdownMenuItem<String>(
                  value: city,
                  child: Text(city, style: TextStyle(color:
Colors.white)),
                );
              }).toList(),
            ),
          ],
        ),
        actions: [
          IconButton(
            icon: Icon(Icons.account_circle, color: Colors.white),
            onPressed: () {},
          ),
        ],
```

```
      ),
      body: SingleChildScrollView(
        child: Padding(
          padding: EdgeInsets.all(16.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Container(
                padding: EdgeInsets.symmetric(horizontal: 10),
                decoration: BoxDecoration(
                  color: Colors.grey[200],
                  borderRadius: BorderRadius.circular(10),
                ),
                child: TextField(
                  decoration: InputDecoration(
                    hintText: "Search for clinics",
                    border: InputBorder.none,
                    prefixIcon: Icon(Icons.search, color: Colors.grey),
                  ),
                ),
              ),
              SizedBox(height: 20),
              // Appointment & Consultation Section
```

```
                Row(
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  children: [
                    Expanded(
                      child: AppointmentCard(
                        title: "Book In-Clinic Appointment",
                        imagePath: "assets/images/images.jpg",
                      ),
                    ),
                    SizedBox(width: 10),
                    Expanded(
                      child: AppointmentCard(
                        title: "Instant Video Consultation",
                        imagePath: "assets/images/hospital.jpg",
                      ),
                    ),
                  ],
                ),
                SizedBox(height: 20),
                Text(
                  "Find a Doctor for your Health Problem",
                  style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold),
                ),
```

```
              SizedBox(height: 10),
              GridView.builder(
                shrinkWrap: true,
                physics: NeverScrollableScrollPhysics(),
                gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
                  crossAxisCount: 4,
                  crossAxisSpacing: 10,
                  mainAxisSpacing: 10,
                  childAspectRatio: 0.9,
                ),
                itemCount: doctorCategories.length,
                itemBuilder: (context, index) {
                  return DoctorCategoryButton(doctorCategories[index]);
                },
              ),
              SizedBox(height: 20),
              Text(
                "Featured services",
                style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold),
              ),
              SizedBox(height: 10),
              Container(
                width: double.infinity,
```

```
                height: 120,
                decoration: BoxDecoration(
                  color: Colors.indigo,
                  borderRadius: BorderRadius.circular(10),
                ),
                child: Center(
                  child: Text(
                    "Affordable Procedures by Expert Doctors",
                    style: TextStyle(color: Colors.white, fontSize: 16),
                    textAlign: TextAlign.center,
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}


class AppointmentCard extends StatelessWidget {
  final String title;
```

```
  final String imagePath;
  const AppointmentCard({
    required this.title,
    required this.imagePath,
    super.key,
  });
  @override
  Widget build(BuildContext context) {
    return Container(
      height: 120,
      decoration: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.circular(10),
        boxShadow: [
          BoxShadow(color: Colors.grey.shade300, blurRadius: 5,
spreadRadius: 2),
        ],
      ),
      child: Column(
        children: [
          Expanded(
            child: ClipRRect(
              borderRadius: BorderRadius.vertical(top:
Radius.circular(10)),
```

```
              child: Image.asset(imagePath, width: double.infinity, fit:
BoxFit.cover),

            ),

          ),

          Padding(

            padding: EdgeInsets.all(8.0),

            child: Text(title, textAlign: TextAlign.center, style:
TextStyle(fontSize: 14, fontWeight: FontWeight.bold)),

          ),

        ],

      ),

    );

  }

}

class DoctorCategoryButton extends StatelessWidget {

  final DoctorCategory category;

  const DoctorCategoryButton(this.category, {super.key});

  @override

  Widget build(BuildContext context) {

    return GestureDetector(

      onTap: () {

        ScaffoldMessenger.of(context).showSnackBar(

          SnackBar(content: Text("Selected: ${category.title}")),

        );
```

```
      },
      child: Column(
        children: [
          Container(
            padding: EdgeInsets.all(10),
            decoration: BoxDecoration(
              color: Colors.blue[50],
              borderRadius: BorderRadius.circular(10),
            ),
            child: Icon(category.icon, color: Colors.indigo, size: 30),
          ),
          SizedBox(height: 5),
          Text(category.title, textAlign: TextAlign.center, style:
TextStyle(fontSize: 12)),
        ],
      ),
    );
  }
}
class DoctorCategory {
  final String title;
  final IconData icon;
  DoctorCategory({required this.title, required this.icon});
}
```

```
List<DoctorCategory> doctorCategories = [
  DoctorCategory(title: "General Physician", icon: Icons.local_hospital),
  DoctorCategory(title: "Skin & Hair", icon: Icons.face),
  DoctorCategory(title: "Women's Health", icon: Icons.pregnant_woman),
  DoctorCategory(title: "Dental Care", icon: Icons.medical_services),
  DoctorCategory(title: "Child Specialist", icon: Icons.child_care),
  DoctorCategory(title: "Ear, Nose, Throat", icon: Icons.hearing),
  DoctorCategory(title: "Mental Wellness", icon: Icons.psychology),
  DoctorCategory(title: "More", icon: Icons.more_horiz),
];
```

Login_screen.dart

```
import 'package:flutter/material.dart';
import 'package:lifeline/home_screen.dart';
class LoginScreen extends StatefulWidget {
  const LoginScreen({super.key});
  @override
  LoginScreenState createState() => LoginScreenState();
}
class LoginScreenState extends State<LoginScreen> {
  final _formKey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  bool _obscurePassword = true;
```

```
  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        fit: StackFit.expand,
        children: [
          Image.asset("assets/images/dr1.jpg", fit: BoxFit.cover),
          Container(color: Colors.black.withOpacity(0.4)),
          Center(
            child: Padding(
              padding: EdgeInsets.all(20),
              child: Form(
                key: _formKey,
                child: Column(
                  mainAxisSize: MainAxisSize.min,
                  children: [
                    Text("Welcome to Lifeline",
```

```
                        style: TextStyle(fontSize: 24, fontWeight:
FontWeight.bold, color: Colors.white)),

                    SizedBox(height: 20),

                    _buildTextField(_emailController, "Email",
Icons.email, false),

                    SizedBox(height: 10),

                    _buildTextField(_passwordController, "Password",
Icons.lock, true),

                    SizedBox(height: 20),

                    ElevatedButton(

                      style: ElevatedButton.styleFrom(backgroundColor:
Colors.blue),

                      onPressed: () {

                        if (_formKey.currentState!.validate()) {

                          Navigator.pushReplacement(context,
MaterialPageRoute(builder: (context) => HomeScreen()));

                        }

                      },

                      child: Text("Login", style: TextStyle(color:
Colors.white)),

                    ),

                    TextButton(

                      onPressed: () => Navigator.pushNamed(context,
'/signup'),

                      child: Text("Don't have an account? Sign Up", style:
TextStyle(color: Colors.white)),

                    ),
```

```
                  ],
                ),
              ),
            ),
          ),
        ],
      ),
    );
  }

  Widget _buildTextField(TextEditingController controller, String label,
IconData icon, bool isPassword) {
    return TextFormField(
      controller: controller,
      obscureText: isPassword ? _obscurePassword : false,
      style: TextStyle(color: Colors.white),
      decoration: InputDecoration(
        labelText: label,
        labelStyle: TextStyle(color: Colors.white70),
        prefixIcon: Icon(icon, color: Colors.white),
        filled: true,
        fillColor: Colors.white.withOpacity(0.2),
        border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10)),
        suffixIcon: isPassword
```

```
            ? IconButton(

                icon: Icon(_obscurePassword ? Icons.visibility_off :
Icons.visibility, color: Colors.white),

                onPressed: () => setState(() => _obscurePassword =
!_obscurePassword),

              )

            : null,

      ),

      validator: (value) => value == null || value.isEmpty ? "$label
cannot be empty" : null,

    );

  }

}
```

Signup_screen.dart

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

class SignupScreen extends StatelessWidget {

  const SignupScreen({super.key});

  @override

  Widget build(BuildContext context) {

    final _formKey = GlobalKey<FormState>();

    final _nameController = TextEditingController();

    final _emailController = TextEditingController();

    final _passwordController = TextEditingController();

    return Scaffold(
```

```
      body: Stack(
        fit: StackFit.expand,
        children: [
          Image.asset("assets/images/dr1.jpg", fit: BoxFit.cover),
          Container(color: Colors.black.withOpacity(0.4)),
          Center(
            child: Padding(
              padding: EdgeInsets.all(20),
              child: Form(
                key: _formKey,
                child: Column(
                  mainAxisSize: MainAxisSize.min,
                  children: [
                    Text("Create an Account", style: TextStyle(fontSize:
24, fontWeight: FontWeight.bold, color: Colors.white)),
                    SizedBox(height: 20),
                    _buildTextField(_nameController, "Full Name",
Icons.person, false),
                    SizedBox(height: 10),
                    _buildTextField(_emailController, "Email",
Icons.email, false),
                    SizedBox(height: 10),
                    _buildTextField(_passwordController, "Password",
Icons.lock, true),
                    SizedBox(height: 20),
```

```
                    ElevatedButton(
                      style: ElevatedButton.styleFrom(backgroundColor:
Colors.lightBlue),
                      onPressed: () {
                        if (_formKey.currentState!.validate()) {
                          Navigator.pushReplacementNamed(context,
'/home');
                        }
                      },
                      child: Text("Sign Up",style: TextStyle(color:
Colors.white)),
                    ),
                    TextButton(
                      onPressed: () => Navigator.pushNamed(context,
'/login'),
                      child: Text("Already have an account? Login", style:
TextStyle(color: Colors.white)),
                    ),
                  ],
                ),
              ),
            ),
          ],
        ),
```

```
    );
  }
  Widget _buildTextField(TextEditingController controller, String label,
IconData icon, bool isPassword) {
    return TextFormField(
      controller: controller,
      obscureText: isPassword,
      style: TextStyle(color: Colors.white),
      decoration: InputDecoration(
        labelText: label,
        labelStyle: TextStyle(color: Colors.white70),
        prefixIcon: Icon(icon, color: Colors.white),
        filled: true,
        fillColor: Colors.white.withOpacity(0.2),
        border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10)),
      ),
      validator: (value) => value == null || value.isEmpty ? "$label
cannot be empty" : null,
    );
  }
}
```

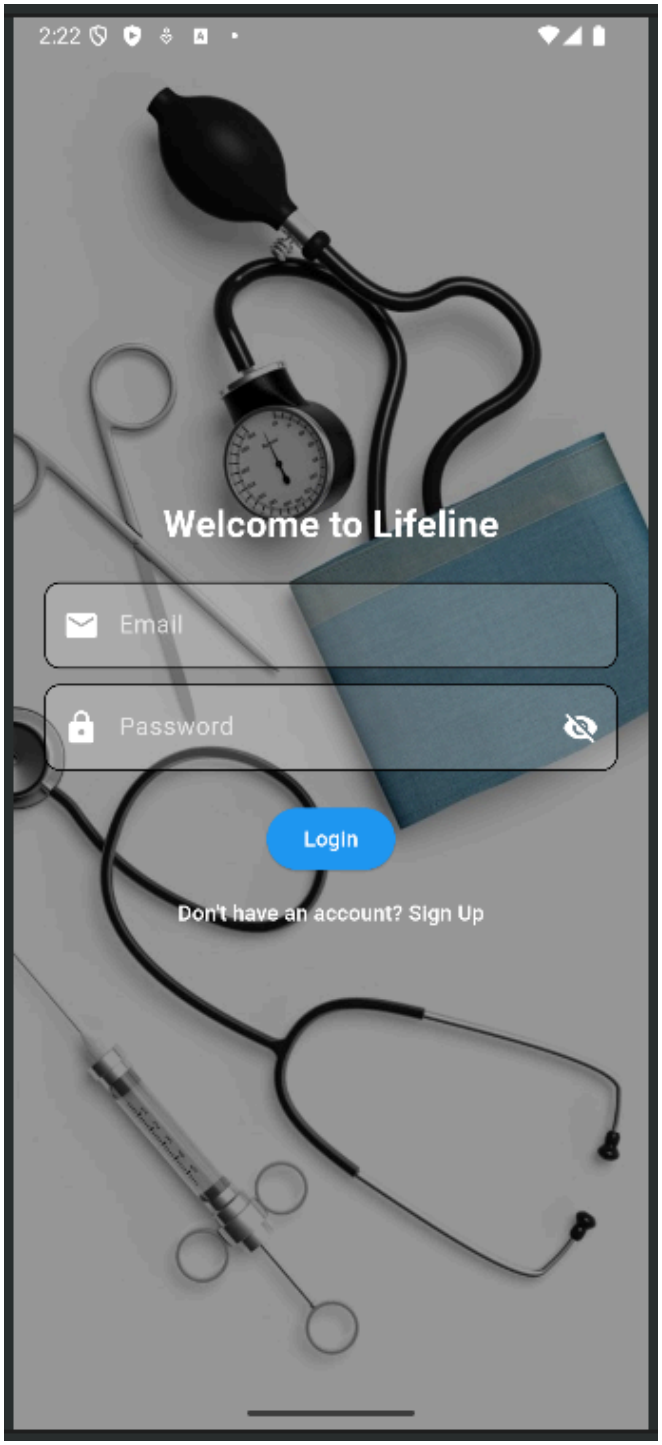
Main.dart

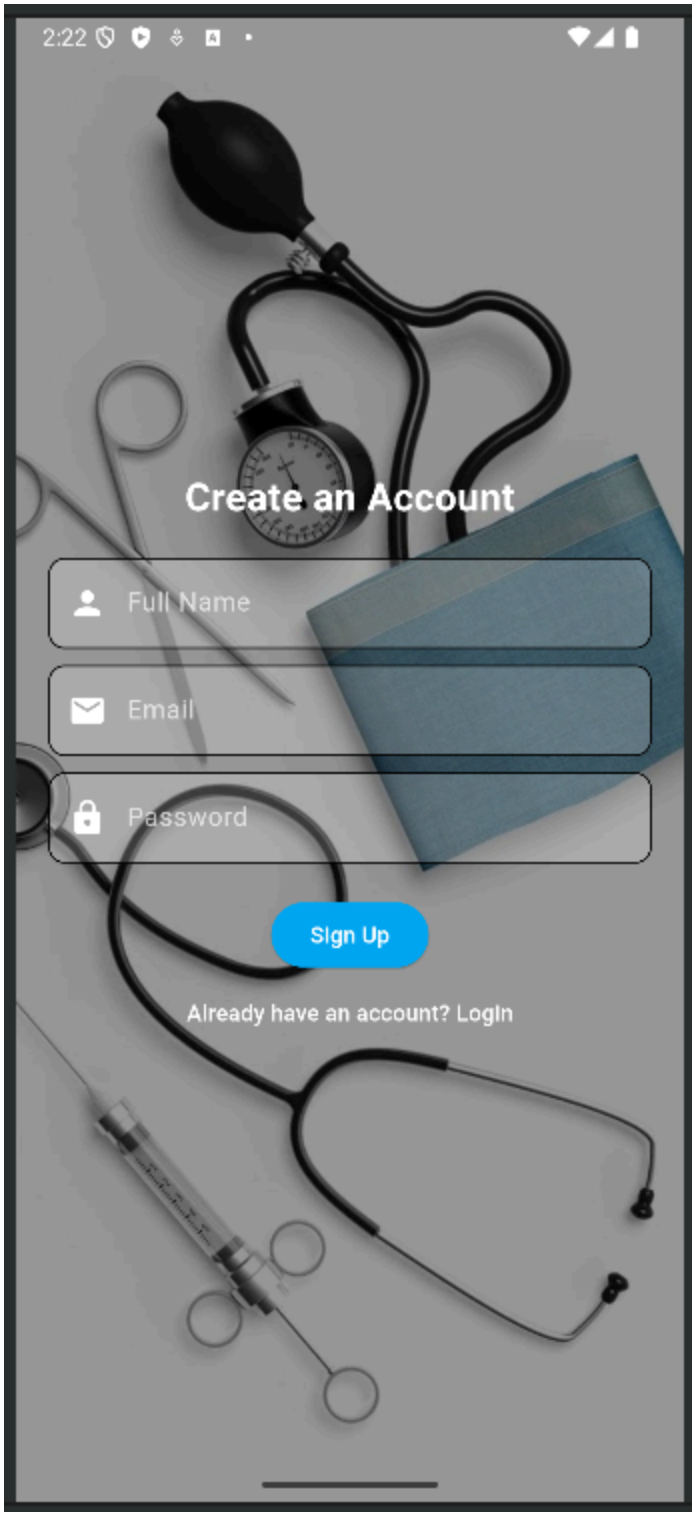
```
import 'package:flutter/material.dart';
```

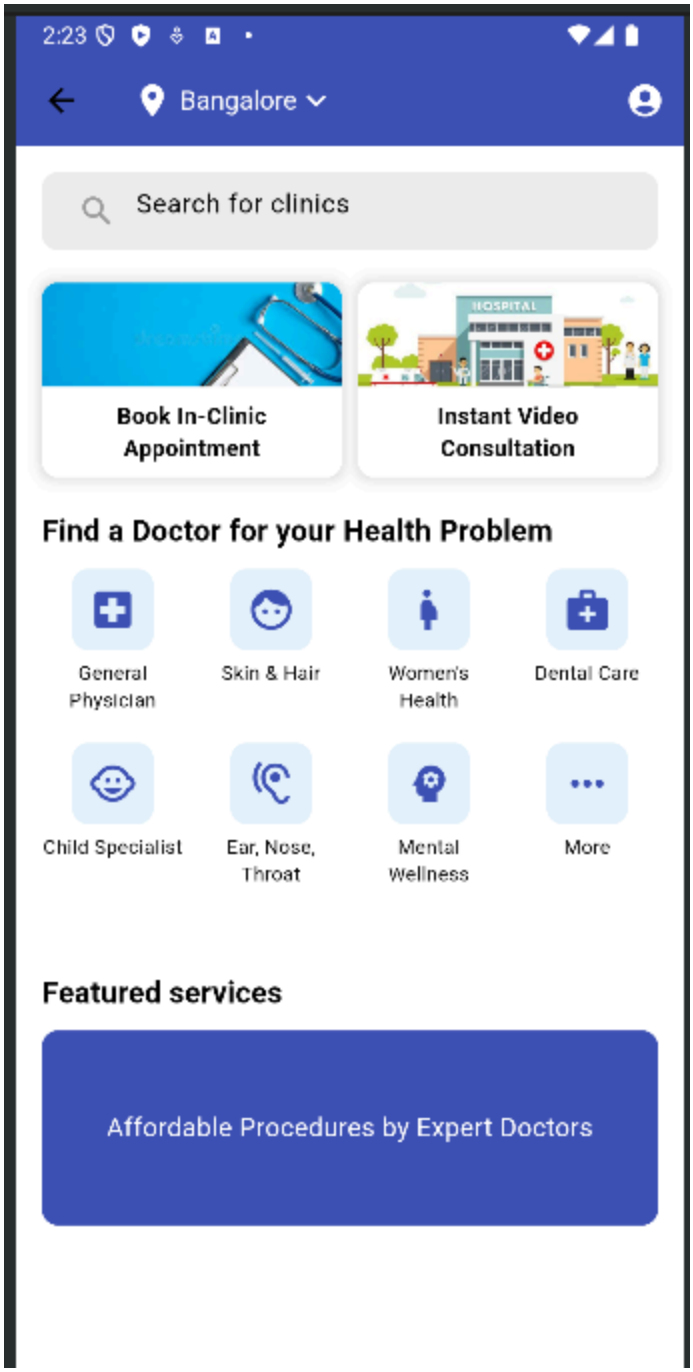
```
import 'home_screen.dart';

import 'login_screen.dart';

import 'signup_screen.dart';


void main() {

  runApp(const LifelineApp()); // Ensure const is used if applicable

}

class LifelineApp extends StatelessWidget {

  const LifelineApp({super.key}); // Add key parameter for best practice

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      title: 'Lifeline',

      theme: ThemeData(

        primaryColor: Colors.green,

        colorScheme: ColorScheme.fromSwatch().copyWith(secondary:
Colors.blue),

      ),

      initialRoute: '/login',

      routes: {

        '/login': (context) => LoginScreen(),

        '/signup': (context) => SignupScreen(),

        '/home': (context) => HomeScreen(),
```

```
        },
      );
    }
```

Output:


2:22
Welcome to Lifeline
Email
Password
Login
Don't have an account? Sign Up


2:22
Create an Account
Full Name
Email
Password
Sign Up
Already have an account? Login



Conclusion:

In this experiment, we successfully implemented gestures for user interaction, navigation for screen transitions, and routing for better app structure. Initially, we faced issues with state management in the dropdown menu and improper routing setup, which we resolved by properly using setState() and defining named routes correctly.

MPL Practical 06

Name: Sharvari Kishor More
Class: D15B
Roll no.: 37

Aim: To integrate Firebase Authentication in a Flutter app.

Theory:

Firebase Authentication in Our Code
User Registration – Users can sign up using their email and password. Upon successful registration, a verification email is sent.
Email Verification – Users must verify their email before they can log in.
User Login – Only verified users can log in to the application.
Google Sign-In – Users have the option to sign in using their Google account.
Password Reset – If users forget their password, they can request a reset link via email.
Logout – Users can securely log out from their session.

Firebase Integration Steps

- Configured a Firebase project and enabled authentication services.
- Initialized Firebase in the Fluttera project and added the required dependencies.
- Implemented authentication functions in auth_service.dart to handle user sign-up, login, password reset, and logout.

Code:

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

import 'package:flutter/material.dart';

import 'package:sharvari/reuseable_widgets/reusable_widgets.dart';


class ResetPassword extends StatefulWidget {

  const ResetPassword({super.key});


  @override

  _ResetPasswordState createState() => _ResetPasswordState();
```

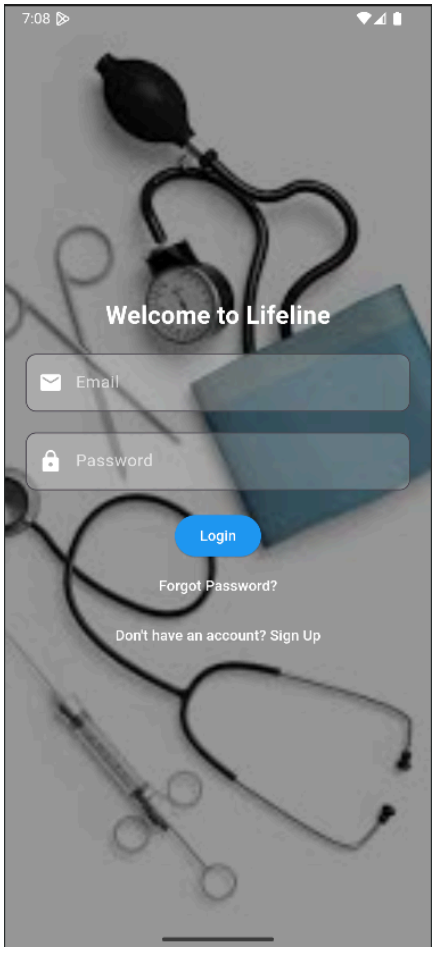
```
}

class _ResetPasswordState extends State<ResetPassword> {
  final TextEditingController _emailTextController = TextEditingController();
  final _formKey = GlobalKey<FormState>();
  bool _isLoading = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        fit: StackFit.expand,
        children: [
          Image.asset("assets/images/dr1.jpg", fit: BoxFit.cover),
          Container(color: Colors.black.withOpacity(0.4)),
          Center(
            child: Padding(
              padding: const EdgeInsets.all(20),
              child: Form(
                key: _formKey,
                child: Column(
                  mainAxisSize: MainAxisSize.min,
                  children: [
                    const Text(
                      "Reset Password",
                      style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold, color: Colors.white),
```

```
            ),
            const SizedBox(height: 20),
            reuseableTextField(
              "Enter Email Id",
              Icons.email,
              false,
              _emailTextController,
                (value) {
              if (value == null || value.isEmpty) {
                return "Please enter your email";
              }
              if
(!RegExp(r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$").hasMatch(value)) {
                return "Please enter a valid email address";
              }
              return null;
            },
            ),
            const SizedBox(height: 20),
            ElevatedButton(
              style: ElevatedButton.styleFrom(backgroundColor: Colors.lightBlue),
              onPressed: _isLoading ? null : _resetPassword,
              child: _isLoading
                  ? const CircularProgressIndicator(color: Colors.white)
                  : const Text("Reset Password", style: TextStyle(color: Colors.white)),
            ),
            TextButton(
```

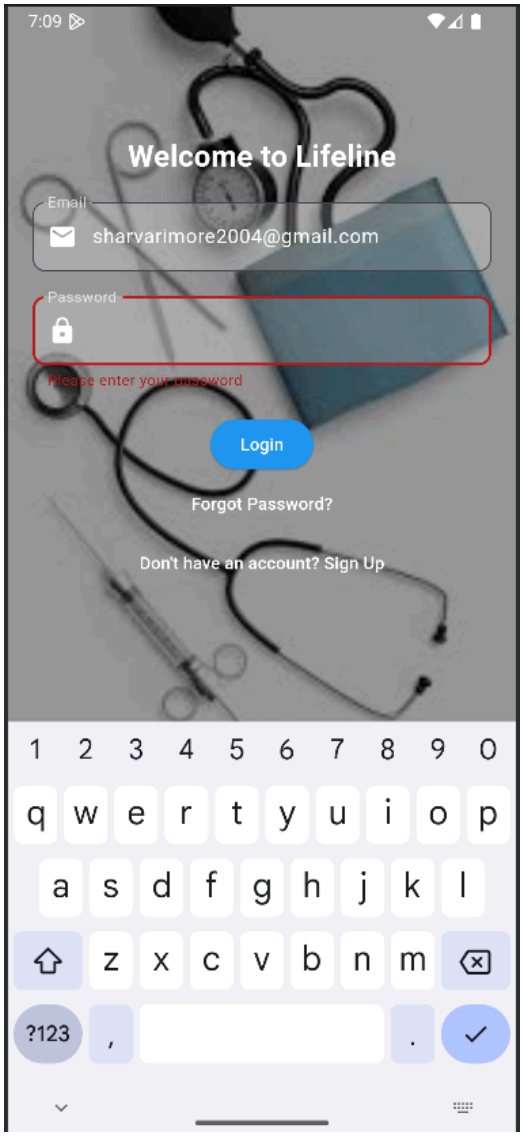
```
                onPressed: () => Navigator.pop(context),
                child: const Text("Back to Login", style: TextStyle(color: Colors.white)),
              ),
            ],
          ),
        ),
      ),
    ),
  ],
 ),
 );
}

void _resetPassword() async {
 if (!_formKey.currentState!.validate()) return;

 setState(() => _isLoading = true);
 String email = _emailTextController.text.trim();

 try {
  await FirebaseAuth.instance.sendPasswordResetEmail(email: email);
  setState(() => _isLoading = false);
  _showSnackbar("Password reset email sent successfully!");
  Navigator.of(context).pop();
 } on FirebaseAuthException catch (e) {
  setState(() => _isLoading = false);
```

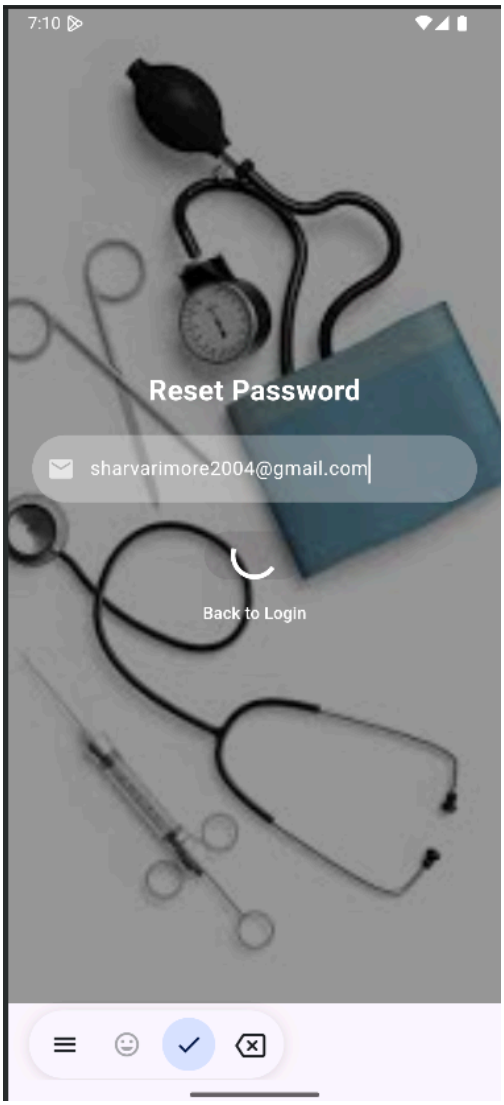
```
      _showSnackbar("Error: ${e.message}");
    } catch (e) {
      setState(() => _isLoading = false);
      _showSnackbar("An unexpected error occurred.");
    }
  }

  void _showSnackbar(String message) {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text(message)));
  }
}
```

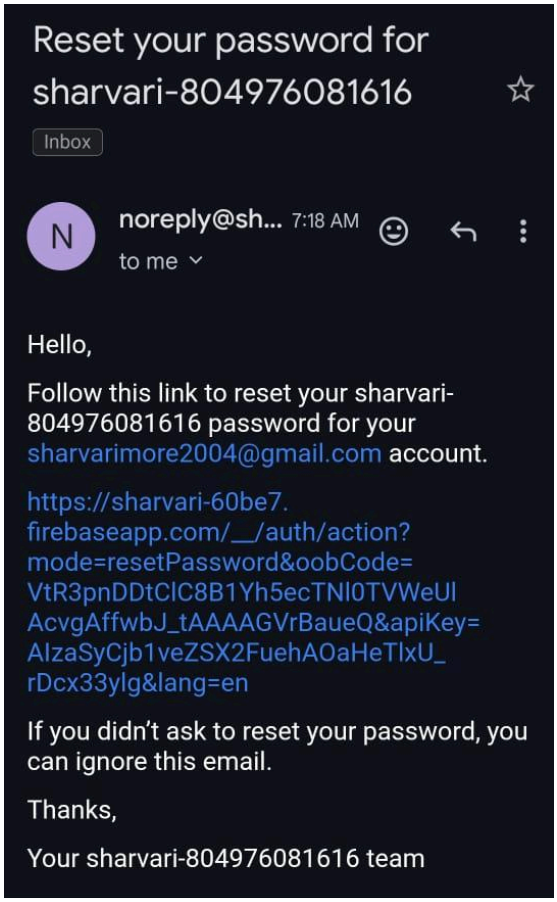
Screenshots:

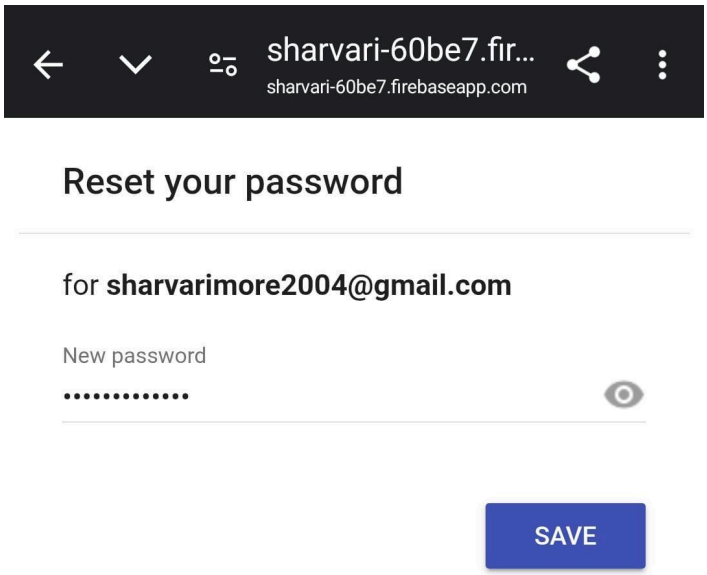


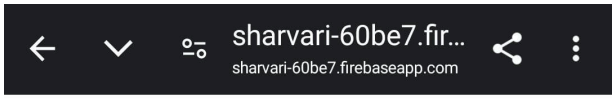
c


7:09
Welcome to Lifeline
Email
sharvarimore2004@gmail.com
Password
Please enter your password
Login
Forgot Password?
Don't have an account? Sign Up


7:10
Reset Password
sharvarimore2004@gmail.com
Back to Login

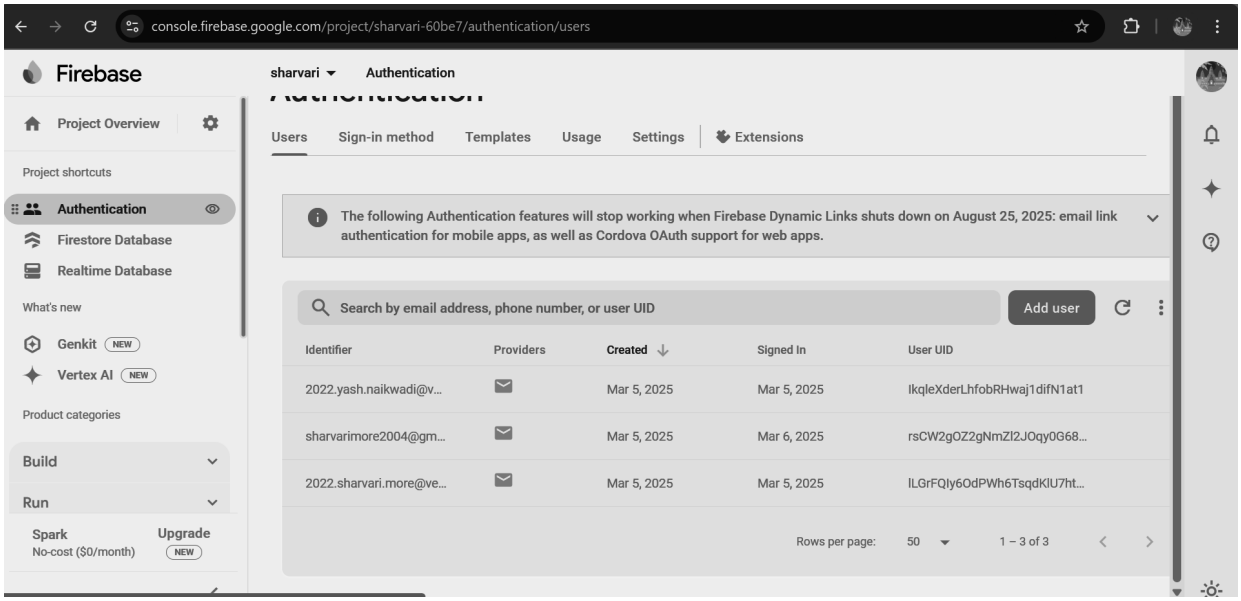

Reset your password for sharvari-804976081616
Inbox
noreply@sh... 7:18 AM
to me
Hello,
Follow this link to reset your sharvari-804976081616 password for your sharvarimore2004@gmail.com account.
https://sharvari-60be7.firebaseapp.com/__/auth/action?mode=resetPassword&oobCode=VtR3pnDDtClC8B1Yh5ecTNI0TVWeUlAcvgAffwbJ_tAAAAGVrBaueQ&apiKey=AIzaSyCjb1veZSX2FuehAOaHeTlxU_rDcx33ylg&lang=en
If you didn't ask to reset your password, you can ignore this email.
Thanks,
Your sharvari-804976081616 team


sharvari-60be7.fir...
sharvari-60be7.firebaseapp.com
Reset your password
for sharvarimore2004@gmail.com
New password
SAVE



Password changed

You can now sign in with your new password



Conclusion:

Integrating Firebase Authentication in our Flutter app enhanced security and user management. While implementing it, I encountered issues like delayed email verification updates and Firebase exceptions during sign-in. I resolved them by manually refreshing user authentication states and handling errors using try-catch blocks. Debugging authentication flows and ensuring proper dependency setup helped streamline the process, resulting in a smooth authentication system.

Practical 7

Name: Sharvari Kishor More
Class: D15B
Roll No.: 35

Aim:

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

Theory:

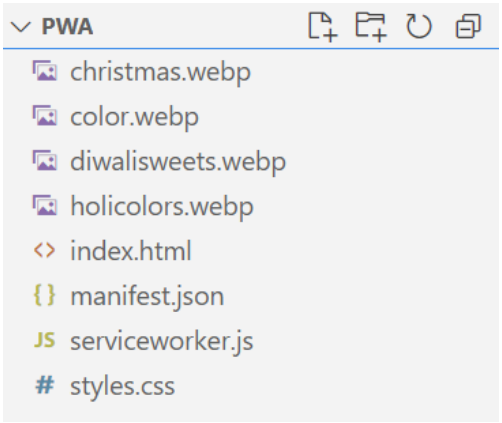
A **Progressive Web App (PWA)** enhances a regular web application by making it installable and capable of running offline like a native app. One of the key components that make a web app a PWA is the **Web App Manifest** file — a simple JSON file that provides essential metadata about the app.
In this experiment, we created a manifest.json file for an E-commerce web application named **Festival Combo Shop**. This file includes details like:

- **App name and short name** for display
- **Start URL**, **display mode** (standalone), **background color**, and **theme color**
- A **description** for the app
- A list of **icons** in required formats (PNG, 192x192 and 512x512, with purpose: "any")
- **Screenshots** of the app to showcase its interface

These details allow modern browsers to recognize the web app as a PWA and prompt users with an **“Add to Home Screen”** option.
We also fixed common validation issues such as:

- Ensuring icons are of minimum size 144x144
- Using proper image formats like PNG and WEBP
- Setting the required sizes and purpose attributes for icons

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">
   <title>Festival Combo Shop</title>
   <link rel="manifest" href="manifest.json">
   <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">
   <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">
   <style>
      .festival-card {
         transition: transform 0.3s ease, box-shadow 0.3s ease;
      }
      .festival-card:hover {
         transform: translateY(-5px);
         box-shadow: 0 10px 25px rgba(0, 0, 0, 0.1);
      }
      .btn-animation {
         transition: all 0.3s ease;
      }
      .btn-animation:hover {
         transform: translateY(-2px);
         box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
      }
      .hero-section {
   background-image: linear-gradient(rgba(0, 0, 0, 0.5), rgba(0, 0, 0, 0.5)), url('/images/color.webp');
}

   </style>
</head>

<script>
      if ('serviceWorker' in navigator) {
        window.addEventListener('load', () => {
          navigator.serviceWorker.register('/serviceworker.js')
            .then(reg => {
              console.log('Service Worker registered successfully:', reg.scope);
            })
```

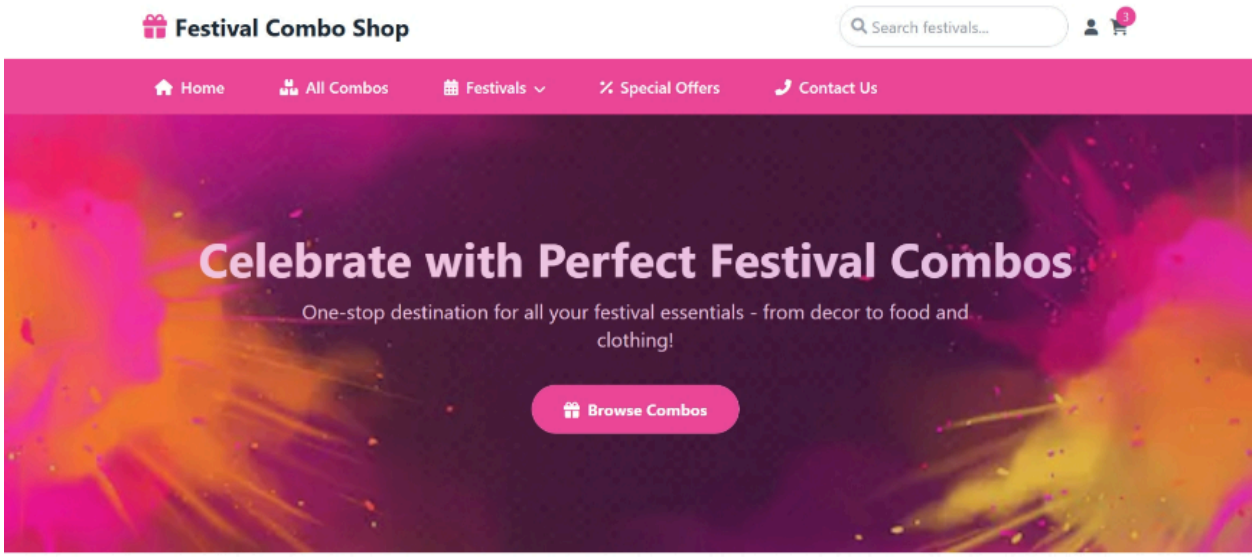
```
            .catch(err => {
              console.error('Service Worker registration failed:', err);
            });
        });
      }
    </script>
```

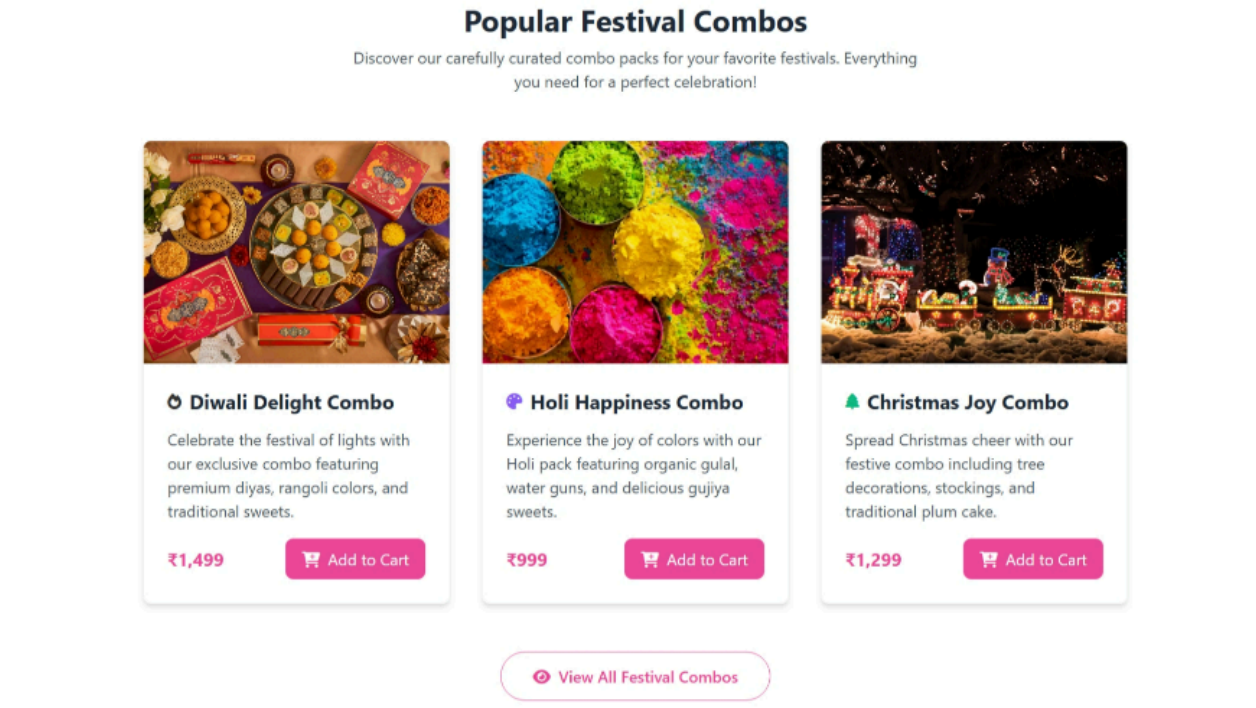
Manifest.json

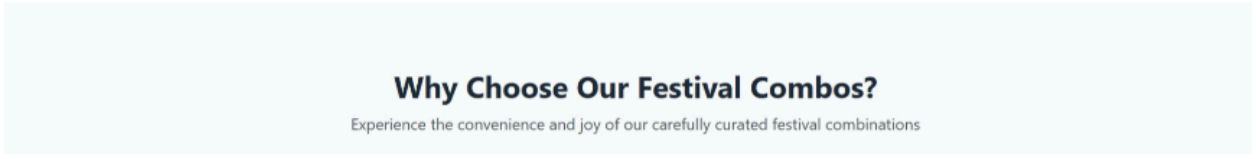
```
{
  "name": "Festival Combo Shop",
  "short_name": "Festival Combos",
  "start_url": "./index.html",
  "display": "standalone",
  "background_color": "#f9fafb",
  "theme_color": "#ec4899",
  "description": "Celebrate with perfect festival combos for Diwali, Holi, Christmas, and more!",
  "icons": [
    {
      "src": "/images/icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/images/icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "screenshots": [
    {
      "src": "/images/color.webp",
      "type": "image/webp",
      "sizes": "1280x720"
    },
    {
      "src": "/images/diwalisweets.webp",
      "type": "image/webp",
      "sizes": "800x600"
    },
```

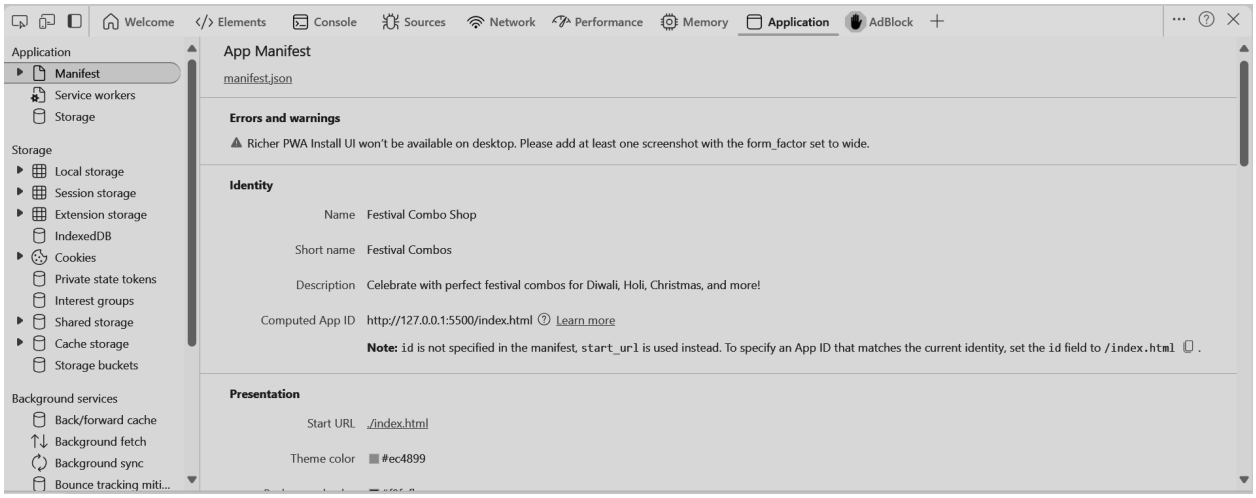
```
    {
      "src": "/images/holicolors.webp",
      "type": "image/webp",
      "sizes": "800x600"
    },
    {
      "src": "/images/christmas.webp",
      "type": "image/webp",
      "sizes": "800x600"
    }
  ]
}
```

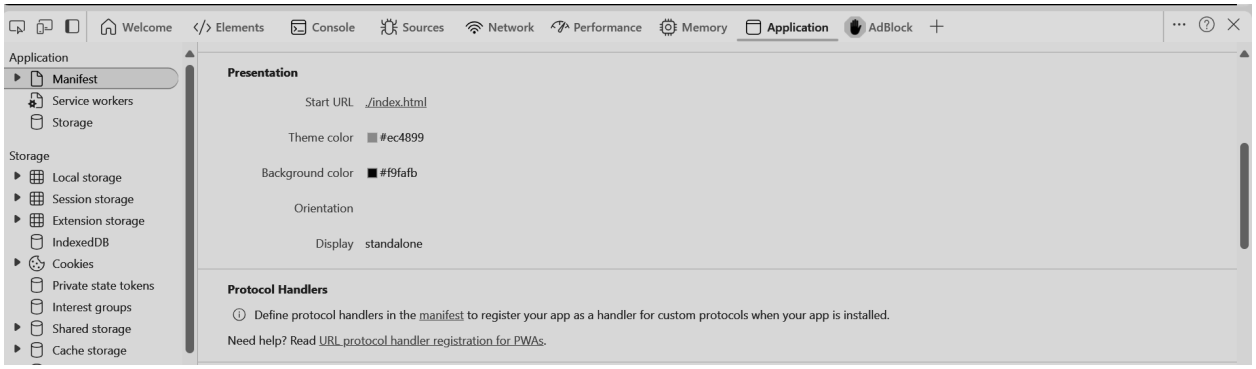
Output:


Festival Combo Shop
Search festivals...
Home
All Combos
Festivals
Special Offers
Contact Us
Celebrate with Perfect Festival Combos
One-stop destination for all your festival essentials - from decor to food and clothing!
Browse Combos

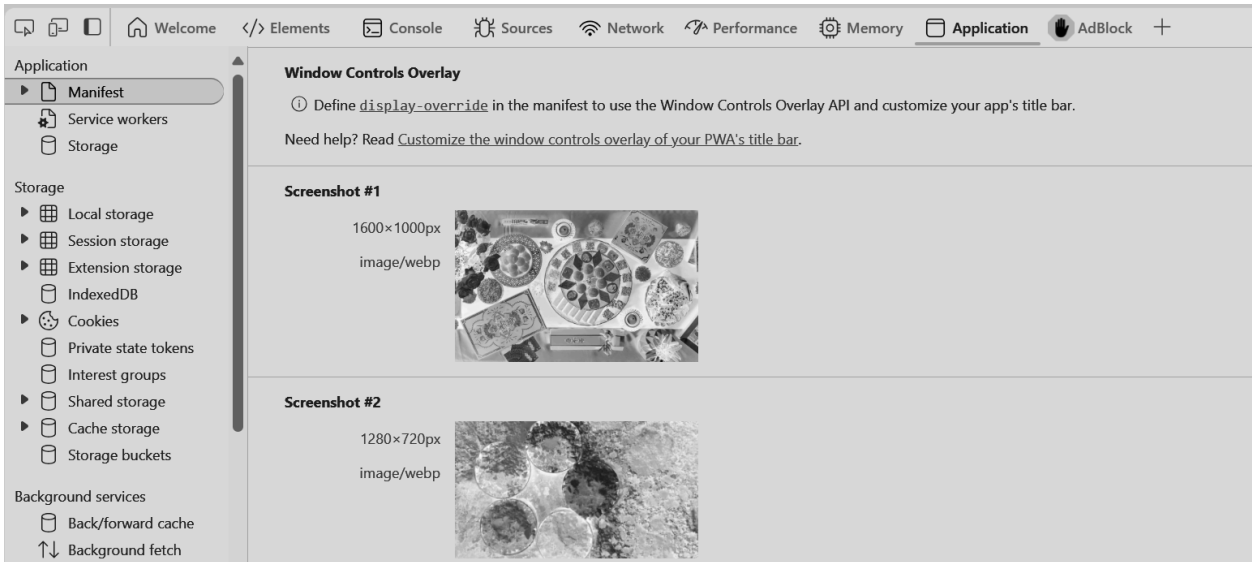

Popular Festival Combos
Discover our carefully curated combo packs for your favorite festivals. Everything you need for a perfect celebration!
Diwali Delight Combo
Celebrate the festival of lights with our exclusive combo featuring premium diyas, rangoli colors, and traditional sweets.
₹1,499
Add to Cart
Holi Happiness Combo
Experience the joy of colors with our Holi pack featuring organic gulal, water guns, and delicious gujiya sweets.
₹999
Add to Cart
Christmas Joy Combo
Spread Christmas cheer with our festive combo including tree decorations, stockings, and traditional plum cake.
₹1,299
Add to Cart
View All Festival Combos


Why Choose Our Festival Combos?
Experience the convenience and joy of our carefully curated festival combinations

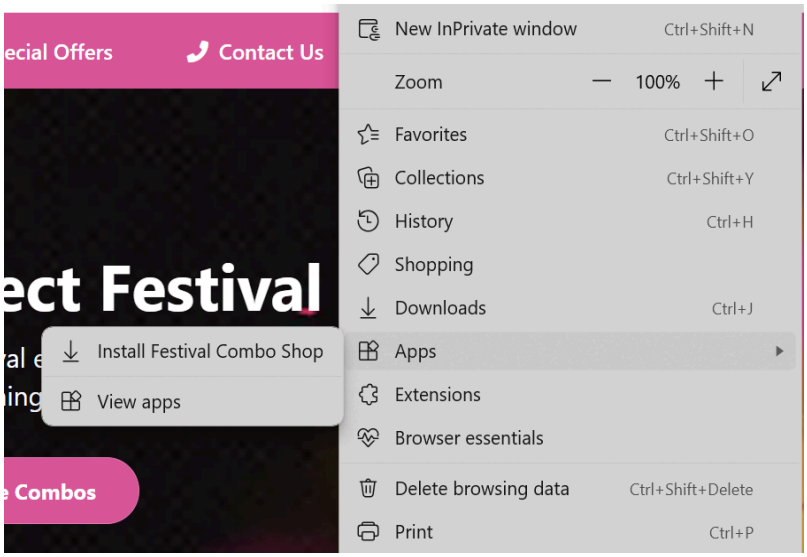

Welcome
Elements
Console
Sources
Network
Performance
Memory
Application
AdBlock
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
Back/forward cache
Background fetch
Background sync
Bounce tracking miti...
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.
Identity
Name Festival Combo Shop
Short name Festival Combos
Description Celebrate with perfect festival combos for Diwali, Holi, Christmas, and more!
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 .
Presentation
Start URL ./index.html
Theme color #ec4899

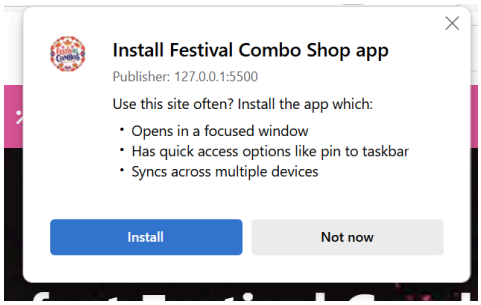

Welcome
Elements
Console
Sources
Network
Performance
Memory
Application
AdBlock
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Presentation
Start URL ./index.html
Theme color #ec4899
Background color #f9fafb
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.

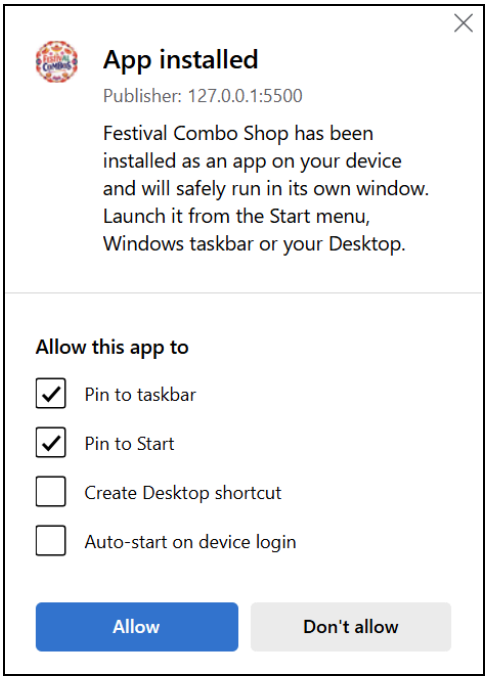

Welcome
Elements
Console
Sources
Network
Performance
Memory
Application
AdBlock
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
Back/forward cache
Background fetch
Background sync
Bounce tracking miti...
Icons
Show only the minimum safe area for maskable icons
Need help? Read the App Image Generator.
192×192px
image/png
512×512px
image/png

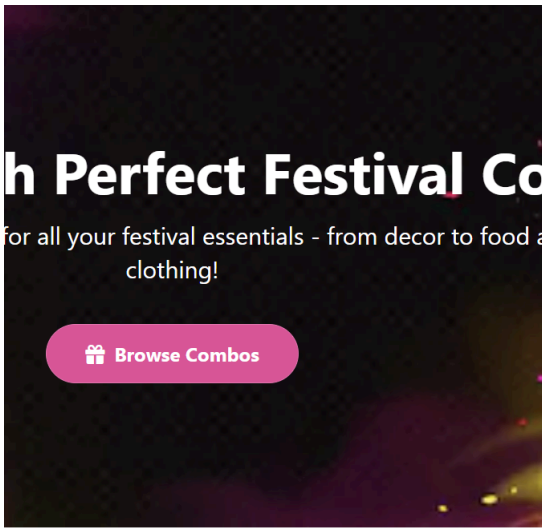

Welcome
Elements
Console
Sources
Network
Performance
Memory
Application
AdBlock
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
Back/forward cache
Background fetch
Window Controls Overlay
Define display-override in the manifest to use the Window Controls Overlay API and customize your app's title bar.
Need help? Read Customize the window controls overlay of your PWA's title bar.
Screenshot #1
1600×1000px
image/webp
Screenshot #2
1280×720px
image/webp


Shared storage
Cache storage
Storage buckets
Background services
Back/forward cache
Background fetch
Background sync
Screenshot #3
1600×1069px
image/webp


ecial Offers
Contact Us
ect Festival
Install Festival Combo Shop
View apps
Combos
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


Install Festival Combo Shop app
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





Conclusion:

By configuring the manifest.json correctly, the E-commerce web app becomes installable and behaves more like a native mobile application. This improves user engagement and accessibility, especially for mobile users. I faced errors while configuring the icons but is was easily solved by changing the width and height of the images.

Practical 8

Name: Sharvari Kishor More
Class: D15B
Roll No.: 35

Aim:

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

Theory:

A Service Worker is a background script that acts as a network proxy, allowing a web app to intercept network requests and serve cached content. It enhances performance and provides offline support.
In our serviceworker.js file, we implemented the following:

- **Install Event:**
  Caches important assets (index.html, CSS, images, manifest) during the installation of the service worker to make them available offline.
- **Activate Event:**
  Deletes old caches that don't match the current version to avoid storage clutter and ensure the latest files are used.
- **Fetch Event:**
  Intercepts all network requests. If the resource is in the cache, it returns the cached version; otherwise, it fetches it from the network.

This setup ensures our eCommerce PWA works smoothly even when offline and improves performance by loading resources from the cache.

Code:

Servicceworker.js

```
const CACHE_NAME = 'festival-combo-shop-v1';
const urlsToCache = [
  '/',
  '/index.html',
  '/styles.css',
  '/manifest.json',
  '/serviceworker.js',
  '/color.webp',
  '/diwalisweets.webp',
  '/holicolors.webp',
  '/christmas.webp',
];
```

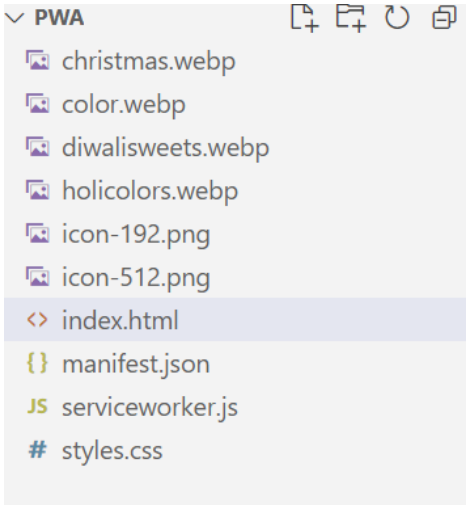
```
// Install service worker
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        console.log('[ServiceWorker] Caching app shell');
        return cache.addAll(urlsToCache);
      })
  );
});

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

// Fetch content
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then(response => {
        // Return cached version or fetch from network
        return response || fetch(event.request);
      })
  );
});
```

Index.html

```
<script>
    if ('serviceWorker' in navigator) {
      console.log('✅ Service workers are supported in this browser.');

      window.addEventListener('load', () => {
        console.log('🌐 Window loaded. Attempting to register service worker...');

        navigator.serviceWorker.register('/serviceworker.js')
          .then(reg => {
            console.log('🎉 Service Worker registered successfully!');
            console.log('🔗 Registration scope:', reg.scope);

            if (reg.installing) {
              console.log('📦 Service Worker is installing...');
            } else if (reg.waiting) {
              console.log('⏸️ Service Worker is installed and waiting...');
            } else if (reg.active) {
              console.log('✅ Service Worker is active.');
            }

            navigator.serviceWorker.ready.then(() => {
              console.log('🚀 Service Worker is ready and controlling the page.');
            });
          })
          .catch(err => {
            console.error('❌ Service Worker registration failed:', err);
          });
      });
      navigator.serviceWorker.addEventListener('controllerchange', () => {
        console.log('🔄 Controller changed: New service worker is now controlling the page.');
      });
      navigator.serviceWorker.addEventListener('message', (event) => {
        console.log('📨 Message received from Service Worker:', event.data);
      });
    } else {
      console.warn('⚠️ Service workers are not supported in this browser.');
    }
```

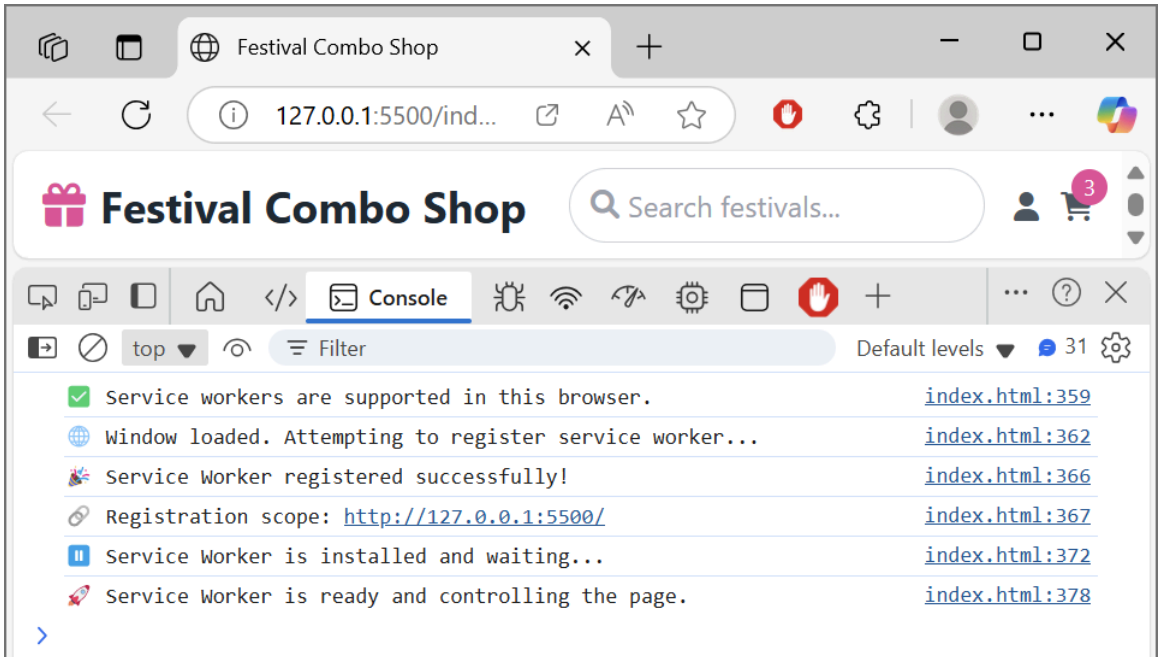
```
</script>
```

Folder Structure:

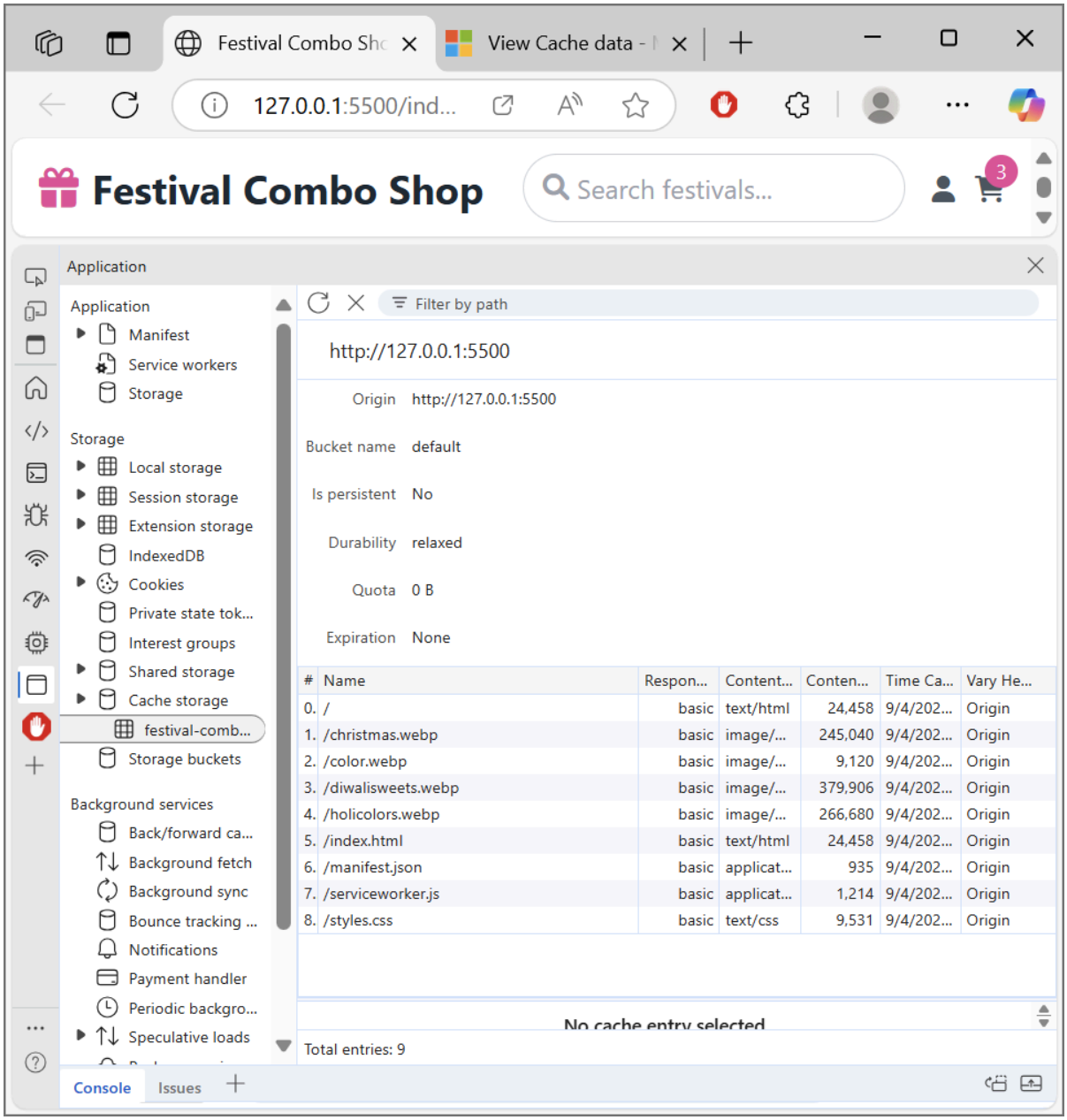


Output:

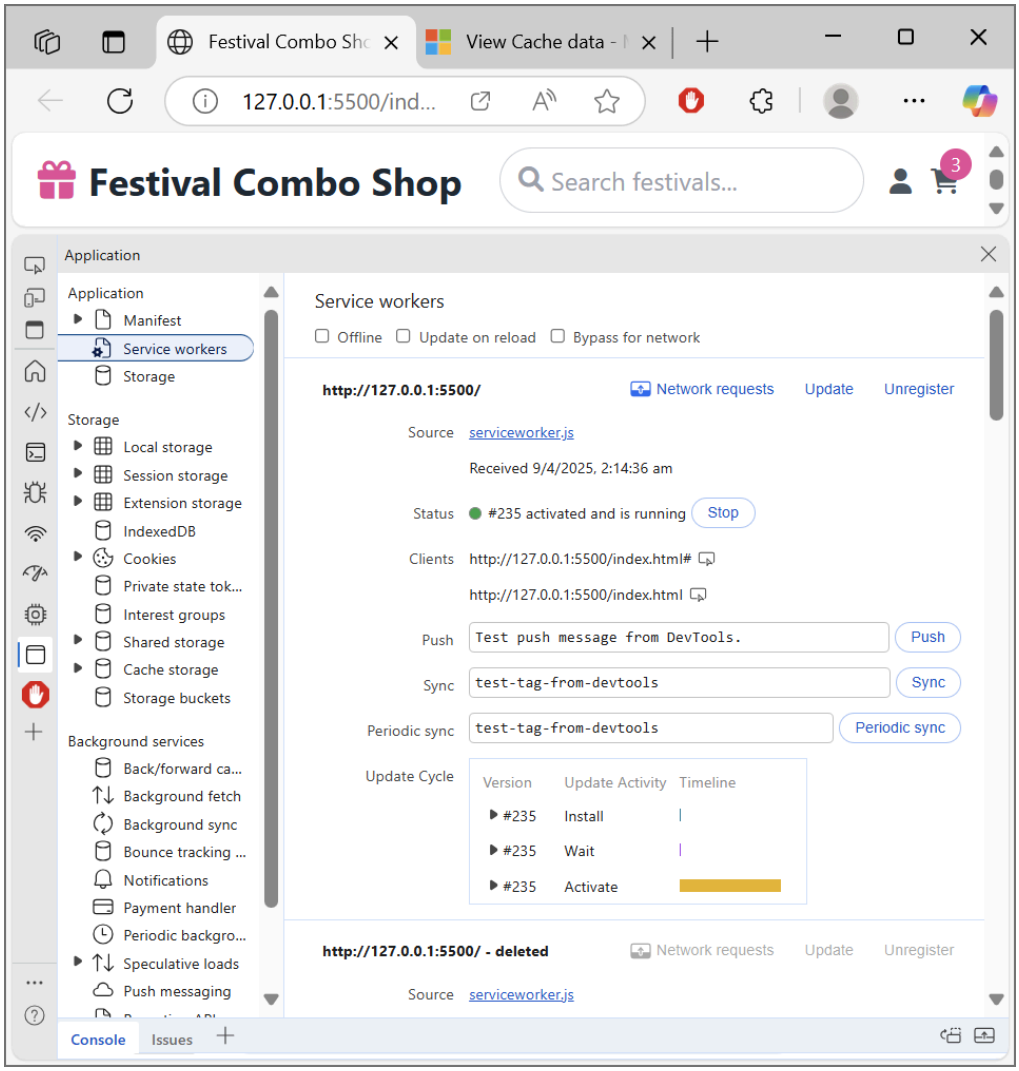
Console logs



Cache Storage



Service Worker running in the background.



Conclusion:

Through this experiment, we successfully implemented a Service Worker in our PWA, enabling offline functionality and improving load performance. It cached essential resources during installation, managed old caches on activation, and served content efficiently through fetch interception. This demonstrates how Service Workers play a crucial role in enhancing user experience in Progressive Web Apps. We faced an error when the cache storage was not being visible, so we had to delete the storage and reload the site.

Practical 9

Name: Sharvari Kishor More
Class: D15B
Roll No.: 35

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.

In this experiment, we enhanced our E-commerce Progressive Web App (PWA) by implementing advanced Service Worker events — **fetch**, **sync**, and **push**.

- The **install** and **activate** events handle caching of files and removal of old caches.
- The **fetch** event intercepts network requests and serves cached resources for better performance and offline support.
- The **sync** event simulates background synchronization when the app regains connectivity.
- The **push** event listens for push notifications and displays them, improving user engagement.

These events help in making PWAs more reliable, fast, and interactive even in low or no network conditions.

Code:

Index.html

```
<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>
```

Serviceworker.js

```
const CACHE_NAME = 'festival-combo-shop-v1';
const FILES_TO_CACHE = [
  '/',
  '/index.html',
  '/styles.css',
  '/manifest.json',
  '/serviceworker.js',
  '/color.webp',
  '/diwalisweets.webp',
  '/holicolors.webp',
  '/christmas.webp',
];
```

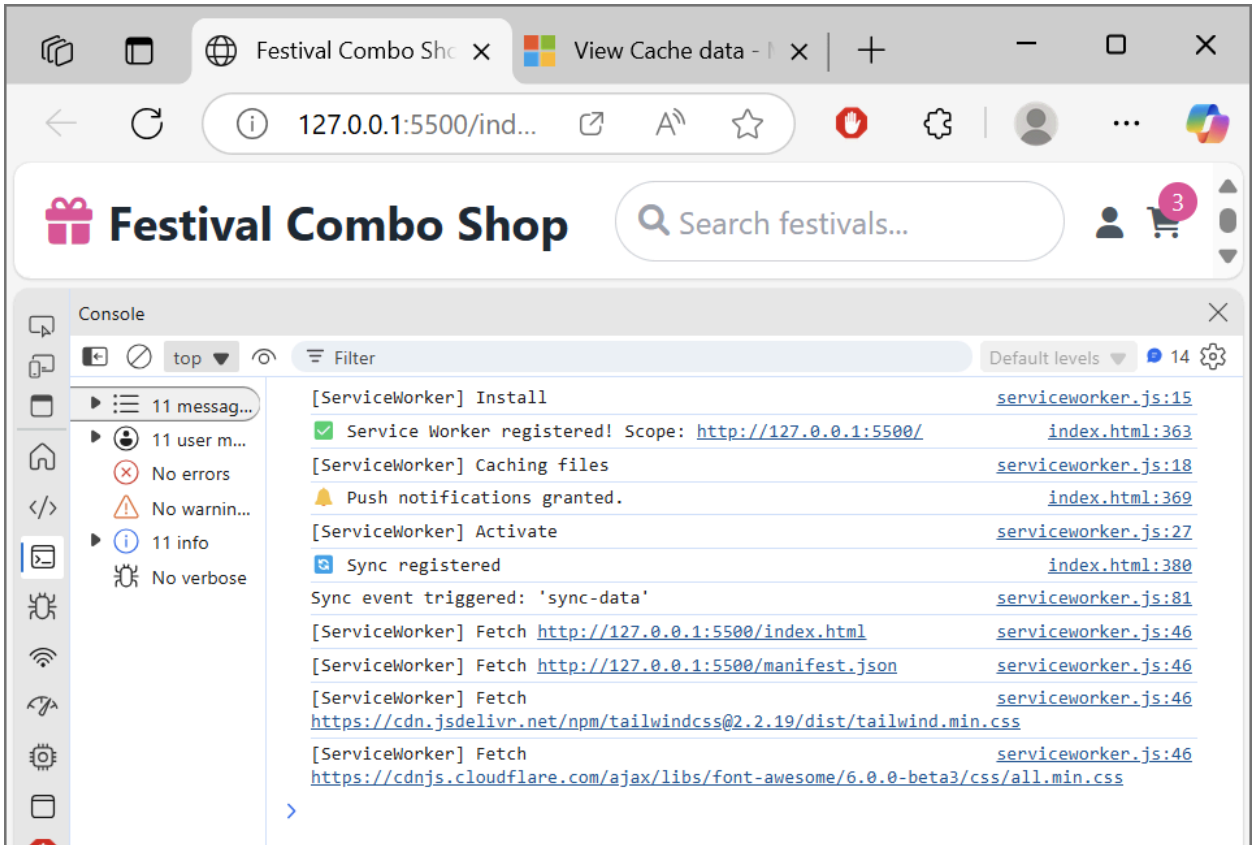
```
// Install Event
self.addEventListener("install", (event) => {
  console.log("[ServiceWorker] Install");
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
      console.log("[ServiceWorker] Caching files");
      return cache.addAll(FILES_TO_CACHE);
    })
  );
});

// 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 {
```

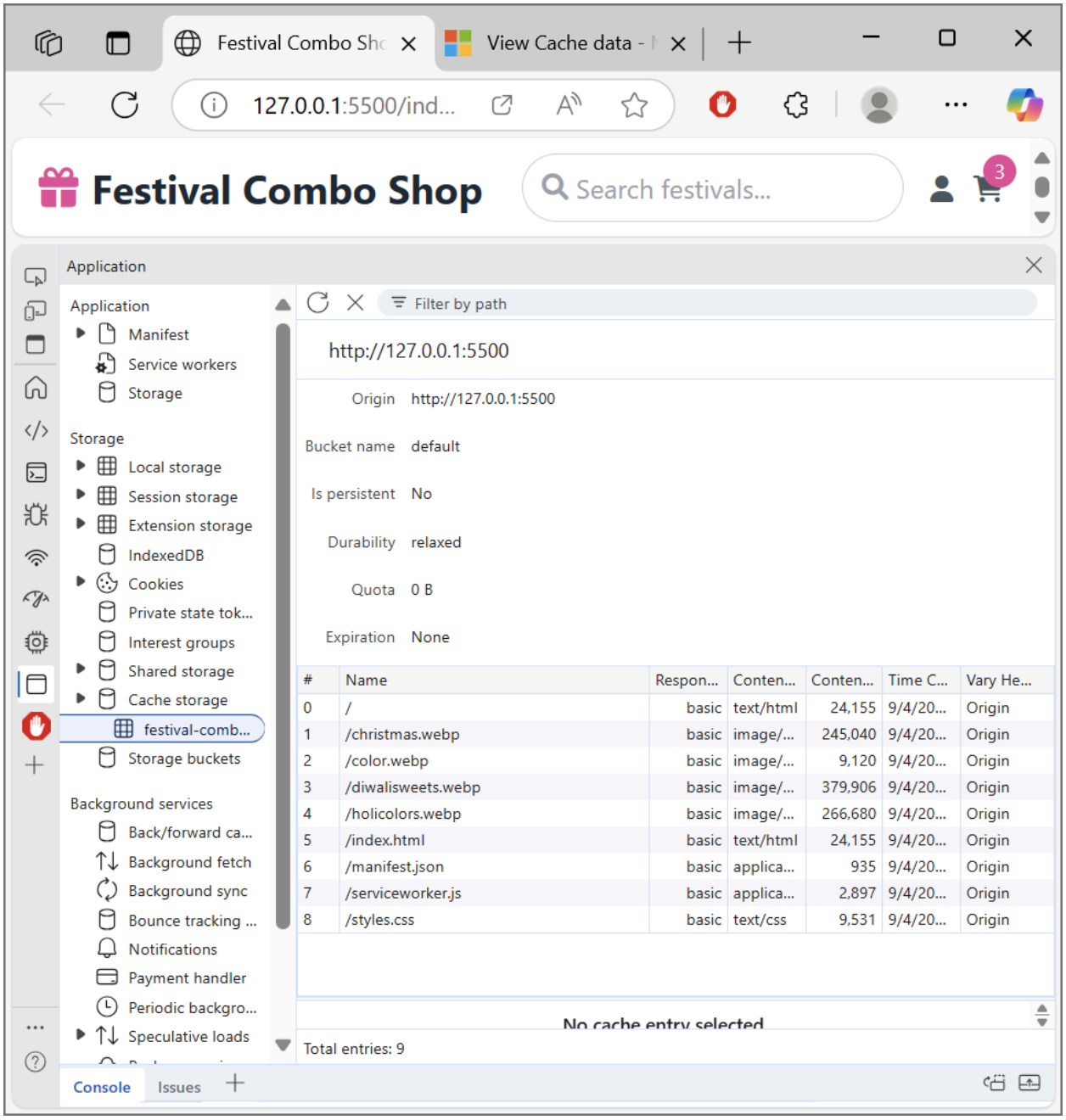
```
      data = event.data.json();
    } catch (e) {
      data = {
        method: "pushMessage",
        message: event.data.text(),
      };
    }

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

Output:

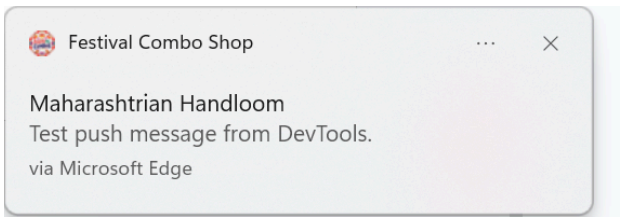
Working of service worker



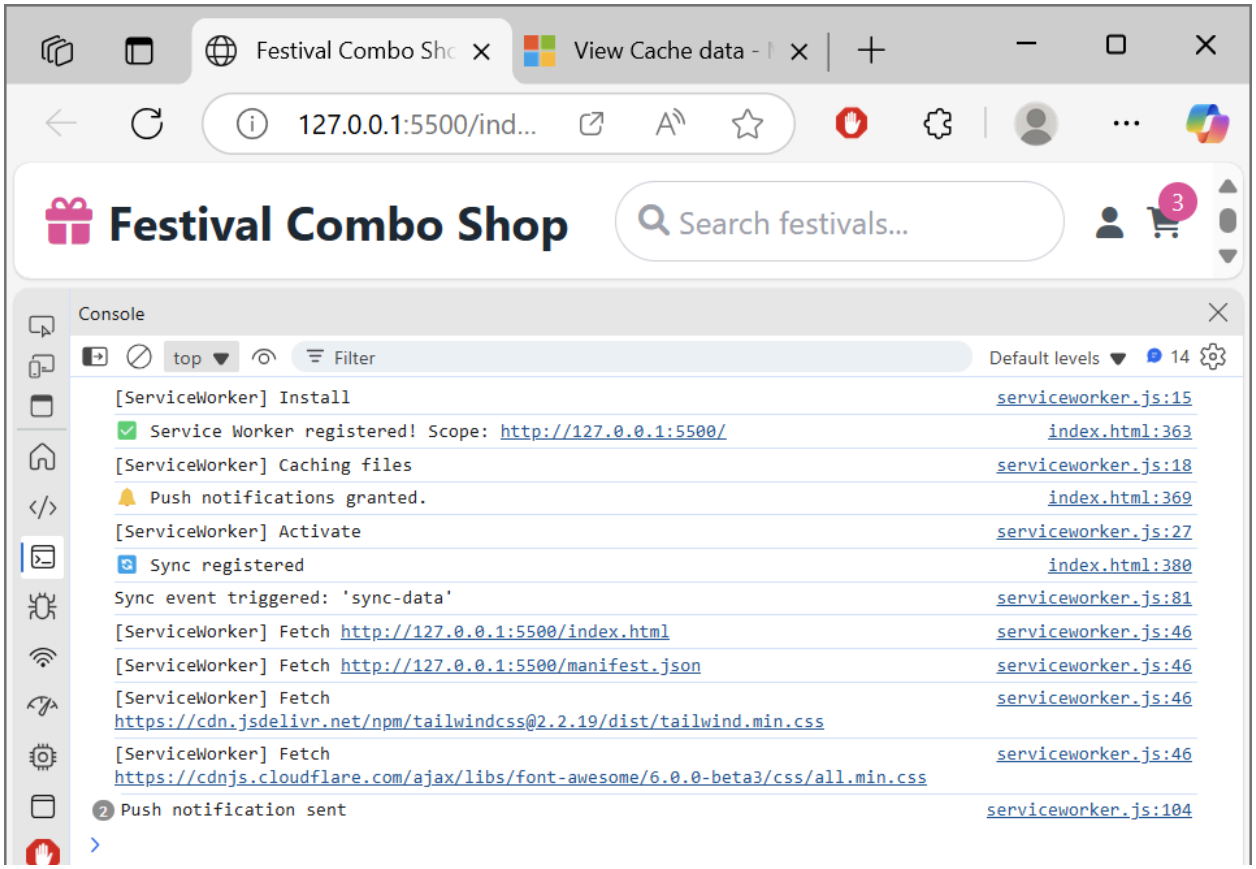
Cache



Sending of push notification.



Test: Push Notification sent



Conclusion:

By implementing fetch, sync, and push events in the Service Worker, we made our E-commerce PWA capable of working offline, syncing data in the background, and sending push notifications. This ensures a better user experience and highlights the power of PWAs in building modern web applications.

Practical 10

Name: Sharvari Kishor More
Class: D15B
Roll No.: 35

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.

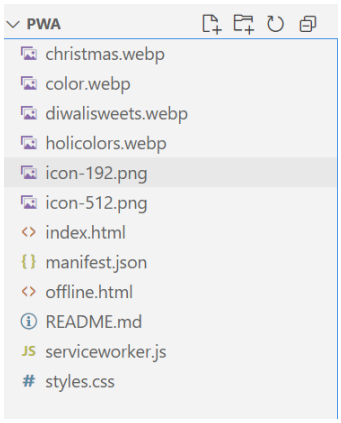
What we implemented in our code

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

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

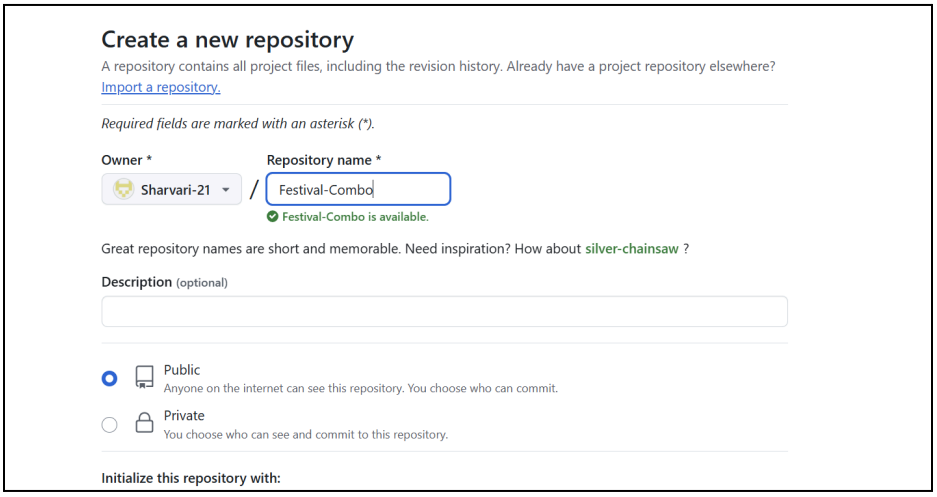
This deployment made our PWA fully functional and accessible online with service worker support, offline capabilities, and installability features.

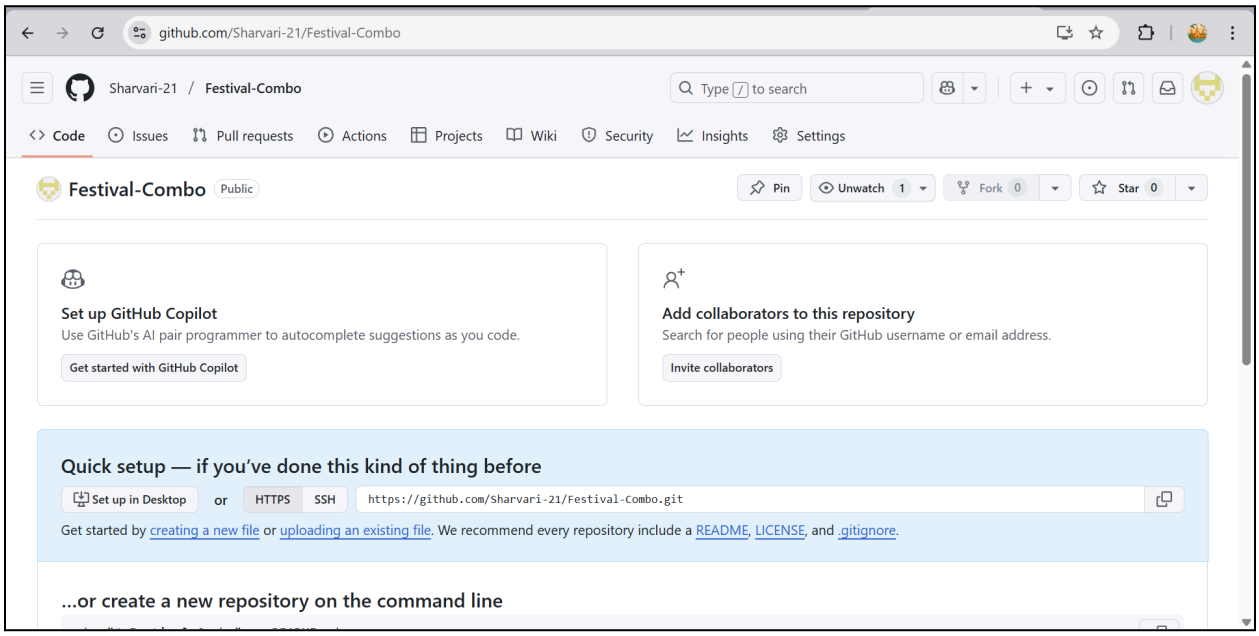
Folder Structure:

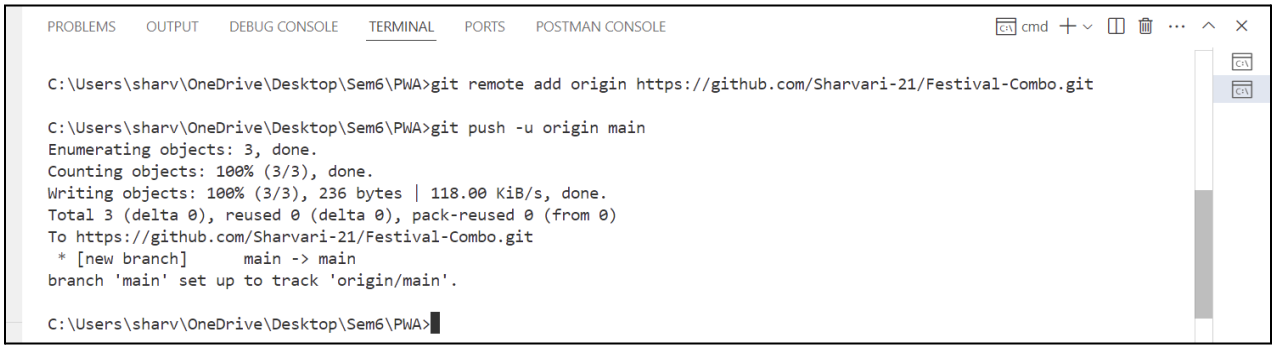


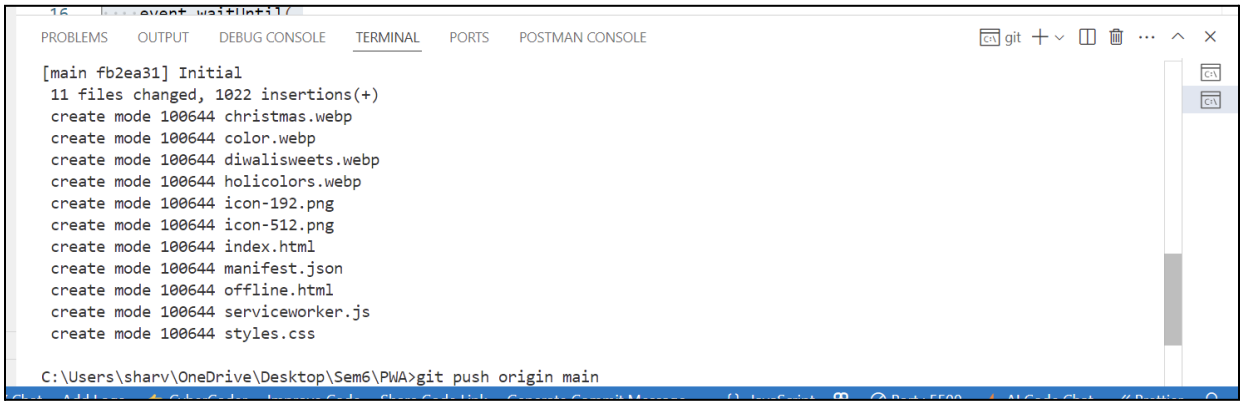
Screenshots:

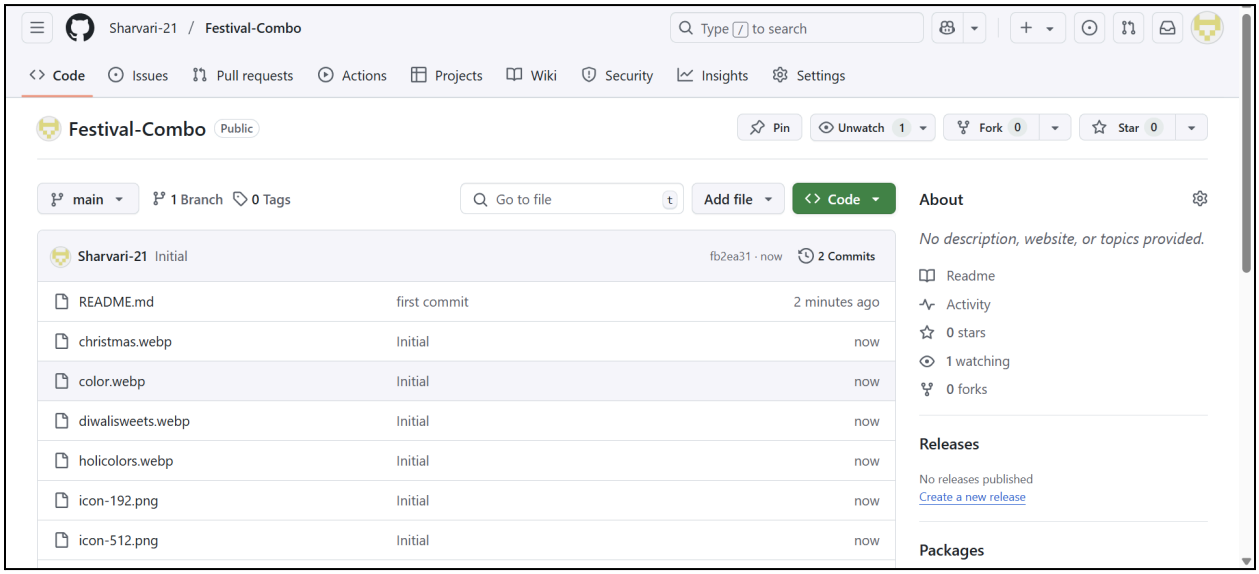
Creation of new repository in public settings.

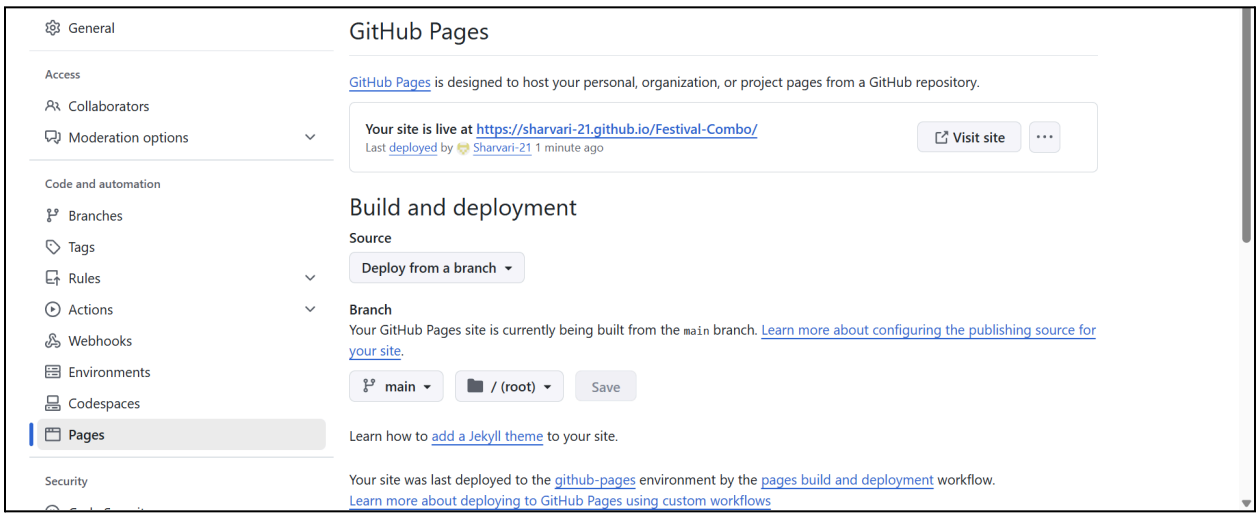


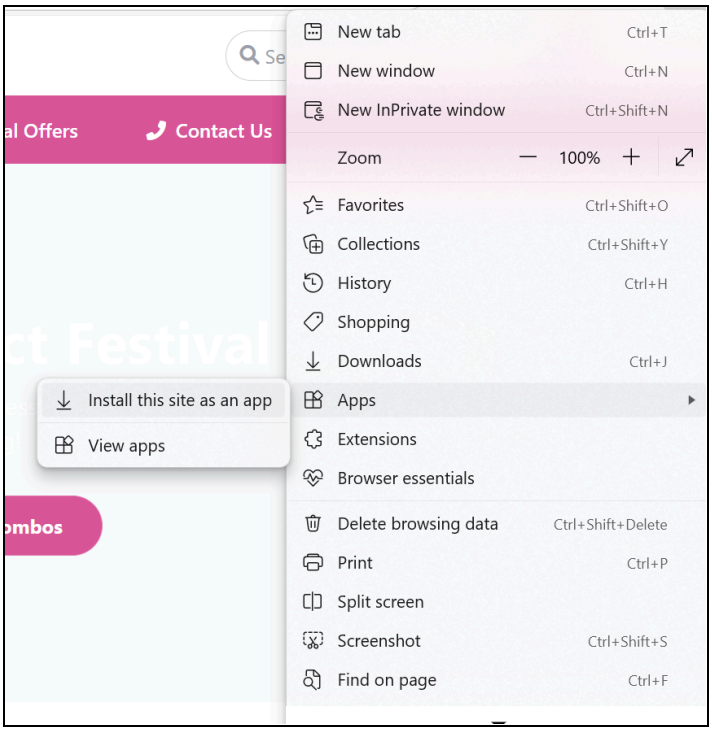


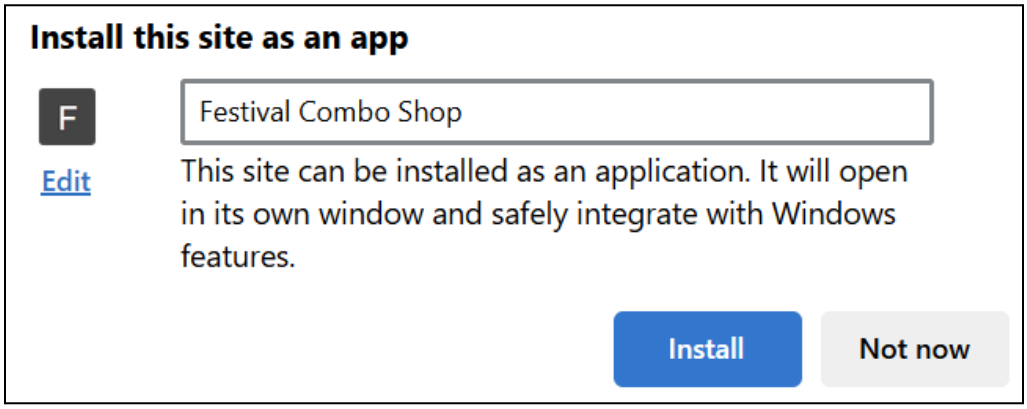



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
[main fb2ea31] Initial
11 files changed, 1022 insertions(+)
create mode 100644 christmas.webp
create mode 100644 color.webp
create mode 100644 diwalisweets.webp
create mode 100644 holicolors.webp
create mode 100644 icon-192.png
create mode 100644 icon-512.png
create mode 100644 index.html
create mode 100644 manifest.json
create mode 100644 offline.html
create mode 100644 serviceworker.js
create mode 100644 styles.css
C:\Users\sharv\OneDrive\Desktop\Sem6\PWA>git push origin main


Sharvari-21 / Festival-Combo
Code Issues Pull requests Actions Projects Wiki Security Insights Settings
Festival-Combo Public
Pin Unwatch 1 Fork 0 Star 0
main 1 Branch 0 Tags Go to file Add file Code
Sharvari-21 Initial fb2ea31 · now 2 Commits
README.md first commit 2 minutes ago
christmas.webp Initial now
color.webp Initial now
diwalisweets.webp Initial now
holicolors.webp Initial now
icon-192.png Initial now
icon-512.png Initial now
About
No description, website, or topics provided.
Readme
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages


General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Security
GitHub Pages
GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.
Your site is live at https://sharvari-21.github.io/Festival-Combo/
Last deployed by Sharvari-21 1 minute ago
Visit site
Build and deployment
Source
Deploy from a branch
Branch
Your GitHub Pages site is currently being built from the main branch. Learn more about configuring the publishing source for your site.
main / (root) Save
Learn how to add a Jekyll theme to your site.
Your site was last deployed to the github-pages environment by the pages build and deployment workflow.
Learn more about deploying to GitHub Pages using custom workflows





The system is available for downloading from the link

Conclusion:

In this experiment, we deployed the website PWA on the internet with the help of git pages. We took proper care of manifest.json file and serviceworker.js file. A link was generated by Git Pages. It took a lot of time for the link to be generated, but after the link creation, the website was visible and working.

Practical 11

Name: Sharvari Kishor More
Class: D15B
Roll No.: 35

Aim:

To use google Lighthouse PWA Analysis Tool to test the PWA functioning

Theory:

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

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

What we implemented:

- We deployed our PWA Festival-Combo on GitHub Pages.
- Then, we ran Lighthouse from Chrome DevTools on the live site.
- Lighthouse gave us excellent scores:
  Performance: 92
  Accessibility: 90
  Best Practices: 89
  SEO: 90

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

Code:
Manifest.json

```
{
  "name": "Festival Combo Shop",
  "short_name": "Festival Combos",
  "start_url": "./index.html",
  "display": "standalone",
  "background_color": "#f9fafb",
  "theme_color": "#ec4899",
  "description": "Celebrate with perfect festival combos for Diwali, Holi, Christmas, and more!",
  "icons": [
   {
```

```
      "src": "/icon-192.png",
      "sizes": "192x192",
      "type": "image/png",
      "purpose": "any-maskable"
    },
    {
      "src": "/icon-512.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "any-maskable"
    }
  ],
  "screenshots": [
    {
      "src": "/diwalisweets.webp",
      "type": "image/webp",
      "sizes": "1600x1000"
    },
    {
      "src": "/holicolors.webp",
      "type": "image/webp",
      "sizes": "1280x720"
    },
    {
      "src": "/christmas.webp",
      "type": "image/webp",
      "sizes": "1600x1069"
    }
  ]
}
```

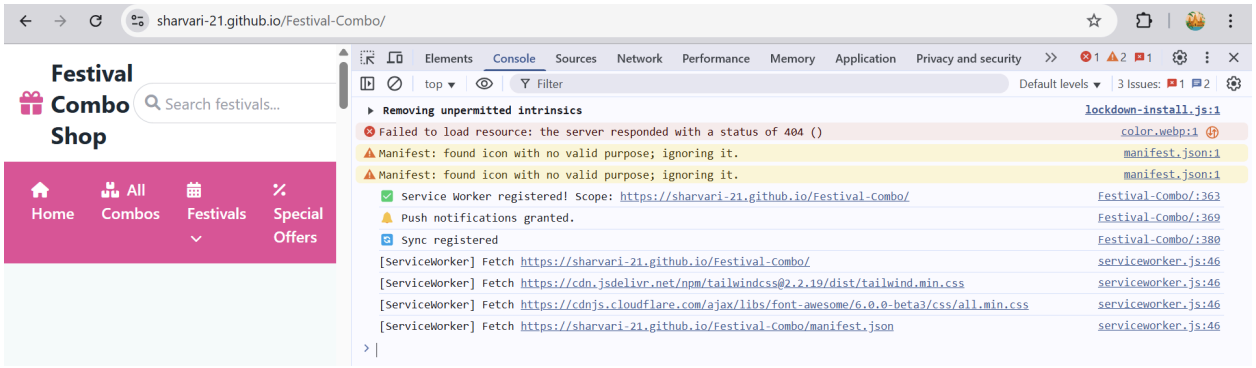
serverworker.js

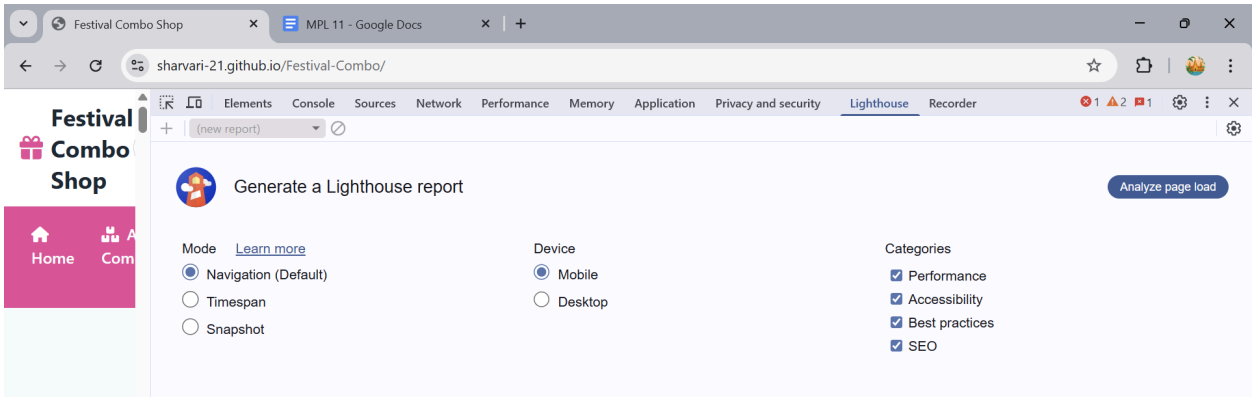
```
const CACHE_NAME = 'festival-combo-shop-v1';
const FILES_TO_CACHE = [
  '/Festival-Combo/',
  '/Festival-Combo/index.html',
  '/Festival-Combo/styles.css',
  '/Festival-Combo/manifest.json',
  '/Festival-Combo/serviceworker.js',
```

```
  '/Festival-Combo/color.webp',
  '/Festival-Combo/diwalisweets.webp',
  '/Festival-Combo/holicolors.webp',
  '/Festival-Combo/christmas.webp',
  ];
// Install Event
self.addEventListener("install", (event) => {
    console.log("[ServiceWorker] Install");
    event.waitUntil(
      caches.open(CACHE_NAME).then((cache) => {
        console.log("[ServiceWorker] Caching files");
        return cache.addAll(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,
        })
      );
    }
  }
});
```





Conclusion:

In this experiment, we tested the Festival Combo PWA using Google Lighthouse and achieved high scores in performance, accessibility, best practices, and SEO. Initially, we faced an error where all images weren't visible, which was resolved by adding the repository name into the serviceworker.js file before the image file.

Sharvari More
D15B 35

MPL
Assignment 1

8.19 Explain the key features and advantages of using Flutter for mobile app development

Answer: Key features of Flutter:
1. Single Codebase - Write one code for both android and IO
2. Fast performance - Uses dart language and a high-performance rendering engine.
3. Hot Reload - See changes instantly without restarting the app.
4. Rich UI Components - Comes with customizable widgets for smooth UI design
5. Native like experience - provides high-quality animations and fast execution.
6. Cross-platform Support - Can be used for mobile, web and desktop apps.
7. Open Source - Free to use and has a strong developer community.

Advantages of Using Flutter:
1. Saves Time and Effort - Single codebase for multiple platforms.
2. High speed Development - Hot reload feature speeds up coding.
3. Cost effective - Reduces development cost and time.
4. Attractive UI - Provides beautiful and customizable widgets
5. Good Performance - Uses dart and Skia for fast and

and smooth rendering.
6. Easy integration: Supports third-party plugins and native code integration.

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

Answer ...

Difference from traditional approaches
1. Single Codebase - Traditional methods need separate code for Android (Java/Kotlin) and iOS (Swift/Objective-C), but Flutter uses one code for both.
2. Hot reload - Traditional apps require full restart after changes, but flutter updates instantly.
3. UI rendering - traditional apps uses native components, while flutter has its own reddering engine (Skia) for faster performance.
4. Performance - Flutter compiles directly to native machine code, making it easier than frameworks that use a bridge (e.g. React Native)
5. Customization - Traditional UI design depends on platform-specific components, but Flutter provides fully customizable widgets.

It has popularity due to following reasons:
1. Fast Development - Hot reload and single codebase saves time.
2. Cross-Platform Support - Works on mobile, web and desktop.
3. Beautiful UI: Rich, Customizable widgets for modern design

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

2 a) Describe the concept of the widget tree in Flutter. Exp how widget composition is used to build complex interfa

Answer 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 the app, where parent widgets contain child widgets.

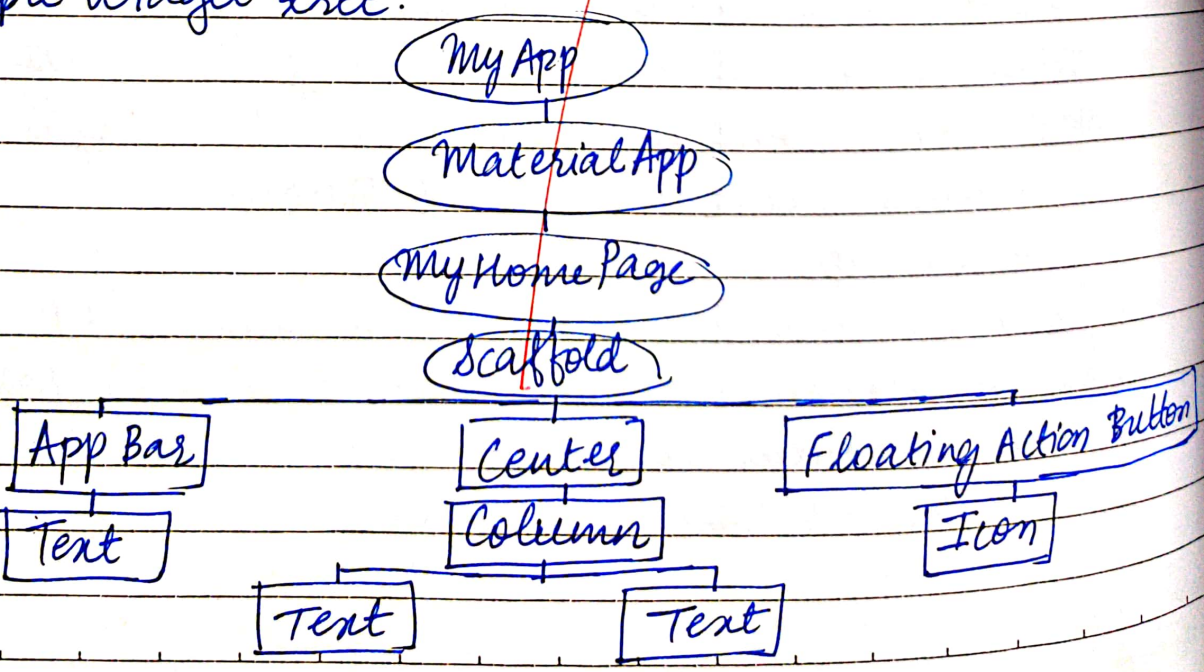
For example, a Scaffold widget can have a column widget, which contains Text and Button widgets. Changes in widgets update the tree dynamically.

Widget Composition for Complex UI
Flutter uses small, reusable widgets to build complex UI. Instead of creating a single large UI block, developers combine multiple small widgets like rows, columns, containers, and Buttons.
For example,
1. A ListView can contain multiple card widgets.
2. A Column can hold Text, Images and Buttons
This modular approach makes the UI visible, flexible, readable and easy to manage.

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

Answer: Commonly used widgets and their roles in a widget T
1. Scaffold - Provides the basic layout structure (AppBar, Body, Floating Button).
2. AppBar - Displays the top navigation bar with a title.
3. Text - Displays simple text on the screen.
4. Image - Shows images from assets or URLs
5. Containers - used for styling (background color, padding, margin).
6. Row - arranges child widgets horizontally.
7. Column - Arranges child widgets vertically.
8. ListView - Displays scrollable lists.
9. ElevatedButton - A clickable button with elevation.
10. TextField - Used for user input (typing text)
11. Card - creates a styled box for displaying content.
12. Stack - Overlays a widget on top of eachother.

Example Widget tree:



(3.29) Discuss the importance of state management in Flutte applications

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

1. Keeps UI updated - Ensure that the app reflects change (eg. button clicks, text inputs).
2. Improves performance - updates only necessary parts of UI instead of reloading everything.
3. Manages complex data - helps handle user inputs, API data and navigation efficiently.
4. Ensures ~~Smooth~~ User experience - keeps apps responsive and interactive.

Types of State in Flutter

1. Local state → managed within a single widget using Stateful widget.
2. Improves Performance → Updates only necessary parts of the UI instead of reloading everything
3. Manages Complex data → helps handle user inputs, API data, and navigation efficiently
4. Ensures smooth user experience → keeps the app responsive and interactive

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

b) Compare and contrast the different state management approaches available in Flutter, such as setState, Provider and Riverpod. Provide scenarios where each approach is suitable

Approach	How it works	When to use
setState	Updates UI by calling setState() in a Stateful Widget	Best for small apps managing state within a single widget. Example: Toggling a button color
Provider	Uses InheritedWidget to share state across widgets efficiently	Suitable for medium sized apps where data needs to be shared between multiple widgets. Example: managing user authentication.
Riverpod	An improved version of provider with better performance and simpler syntax	Best for large apps that need complex state management with dependency injection. Example: Handling API data and app-wide them

Choosing the Right Approach:
- Use setstate for simple UI updates.
- Use provider for moderate state sharing across widge
- Use Riverpod for scalable, well structured application

(2.a) Explain the process of integrating Firebase with a Flut application. Discuss the benefits of using Firebase as a backend solution.

Process of integrating Firebase with a Flutter Applicatio
1. Create a Firebase project - Go to (Firebase Console) (https://console.firebase.google.com/), create a new a
2. Add Firebase to Flutter App - Register the App (Android/ and download the google-services.json (Android) or Google Service-info.plist (iOS).
3. Install Firebase packages - Add dependencies like 'firebase_core' and 'firebase_auth' in 'pubspec.yaml'
4. Initialize Firebase - Import Firebase in 'main.dart' and call 'Firebase.initializeApp()'.
5. Use Firebase Services - Implement authentication, database, or cloud functions as needed.

Benefits of Using Firebase as a Backend Solution:
1. Real time Database - Syncs data instantly across devices
2. Authentication - provides ready-to-use sign in options (Google, Email, etc)
3. Cloud Firestore - stores structured data efficiently.
4. Hosting and storage - Hosts web apps and stores

securely
5. Scalability - Handles large user bases without manag servers
6. Push Notifications - sends alerts and updates to users

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

Answer. Common Firebase services used in Flutter development:
1. Firebase authentication - provides user sign-in method (Google, Email, Facebook, etc.)
2. Cloud Firestore - A NoSQL database that stores and syn data instantly across all connected devices.
3. Firebase Realtime database - stores and updates dat instantly across all connected devices.
4. Firebase cloud storage - used for storing and retrievin files like images and videos.
5. Firebase Cloud Messaging (FCM) - sends push notifications to users.
6. Firebase Hosting - deploys web apps with fast and secure hosting.
7. Firebase Analytics - Tracks user behaviour and app performance.

Data Synchronization
1. Real-time updates - Firestore and Realtime Database syne data across devices instantly.
2. Listeners and streams - Widgets listen for changes and update the UI automatically.

3. Offline Support - Firebase caches data, allowing apps t
work offline and sync when online

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

Sharvari Kishor More
D15B 35

05/03

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.

Answer: A progressive Web App (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 internet using cached data.
3. Fast performance - Loads quickly, even on slow networks.
4. No App Store Required → Users can install it directly from the browser
5. Lower Development Cost → One PWA can replace separate Android and iOS apps.

Difference between PWA and Traditional Apps.

	Feature	PWA	Traditional Mobile App
①	Install^n	Direct from browser	Download from app store.
②	Internet required	works offline with caching	Usually requires internet
③	Perform	Fast with service workers	Faster but needs installation
④	Updates	Automatic, no appstore approval	manual updates needed.

⑤	Development cost	lower (one codebase for all)	higher (separate for each platform

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

2. Define responsive web design and explain its importa in the context of Progressive web Apps. Compare and contrast responsive, fluid and adaptive web design approaches.

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

Importance of Responsive Design in PWAs
1. Better User experience - PWAs work smoothly on any device
2. Faster load time - Optimized design improves speed.
3. SEO Benefits - Google ranks responsive sites higher.
4. Cost-Effective - No need to build multiple versions for different screens.

Comparison of Web Design approaches:

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

Key Differences

- Responsive adapts dynamically to all screens.
- Fluid resizes smoothly but may not be fully optimized.
- Adaptive loads different layouts based on device type.

Responsive design is best for PWAs because it ensures a seamless experience on all devices.

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

Answer

Lifecycle of service workers

A service worker is a script that runs in the backgro and helps a web app work offline, load faster and send push notifications, its lifecycle has three main pha

1. Registration phase

- The browser registers the service worker using JavaScript

Code Example :

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

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

2. Installation Phase

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

Code Example :

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

```
    );
});
```

- This ensures the app loads even without the internet.

3. Activation Phase

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

Code Example:

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

- The service worker is now fully active and controls network requests.

Final step: Fetch and Sync.

- Once activates the service worker intercepts networks requests, serves cached files and syncs data when the internet is available
- This lifecycle makes PWAs faster, more reliable, and capable of working offline.

Q.4. Explain the use of indexedDB in Service Worker for Data storage

Indexed DB is a browser database that stores large amou of structured data like JSON objects. It helps PWAs work offline by saving and retrieving data efficiently.

Usage

1. Offline Support - stores data when offline and syncs it later.
2. Efficient Storage - Saves structured data like user settings, cart items or form inputs
3. Faster Access - Retrieves data quickly without needing a network request.
4. Persistent Data - Data remains saved even after the browser is closed.

Use of IndexedDB by Service Workers.

Opening the Database

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

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

Creating a store and Adding Data

```
request.onupgradeneeded = function(event) {
    let db = event.target.result;
    let store = db.createObjectStore('Users', {keyPath: 'id'});
    store.add({id: 1, name: 'Sharvari', age: 20});
};
```

Fetching Data in service worker

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

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