MPL Idea Summary

MedLink – Medical Assistant App

Objective

The aim of the MedLink application is to provide users with instant mobile access to nearby doctors and essential healthcare services.

Key Features

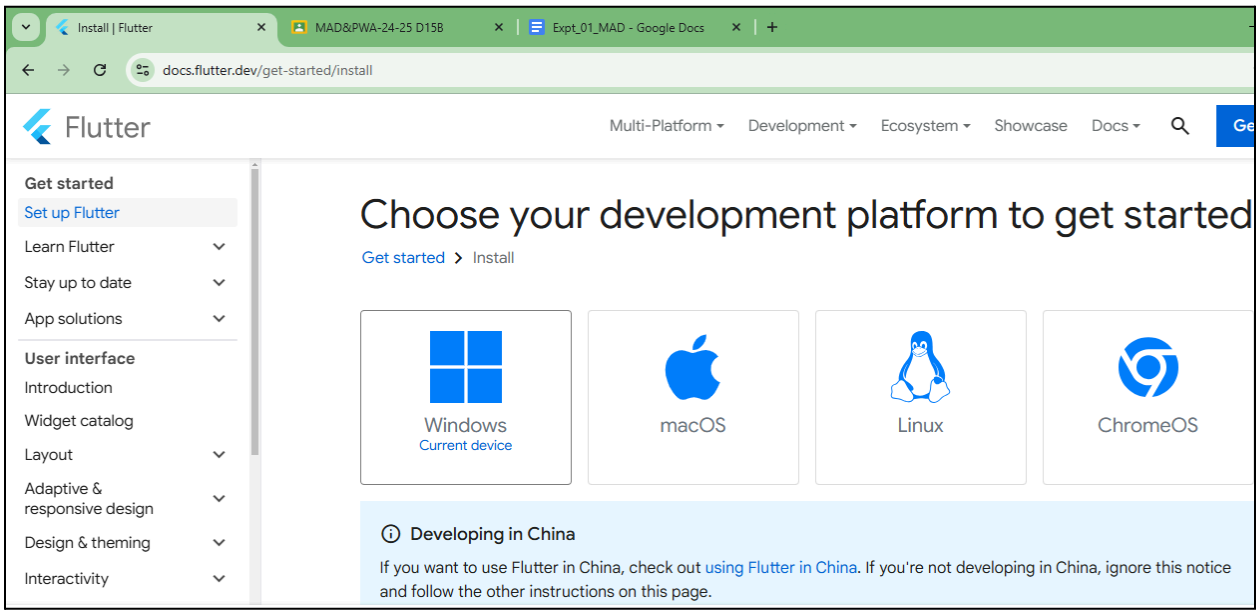
1. Offline Map with Nearby Doctors & Emergency Centers
    - Provides access to a map with locations of nearby doctors, hospitals, and emergency centers, even without an internet connection.
2. AI-Powered Symptom Checker
    - An intelligent symptom checker that helps users assess their health condition and receive preliminary guidance.
3. Emergency SOS Location Button with Live Sharing
    - A one-touch SOS button that shares the user's real-time location with emergency contacts for immediate assistance.
4. Smart Medicine Reminder with Pill Recognition
    - A smart reminder system that not only schedules medication alerts but also includes pill recognition for easy identification.

MPL Practical 01

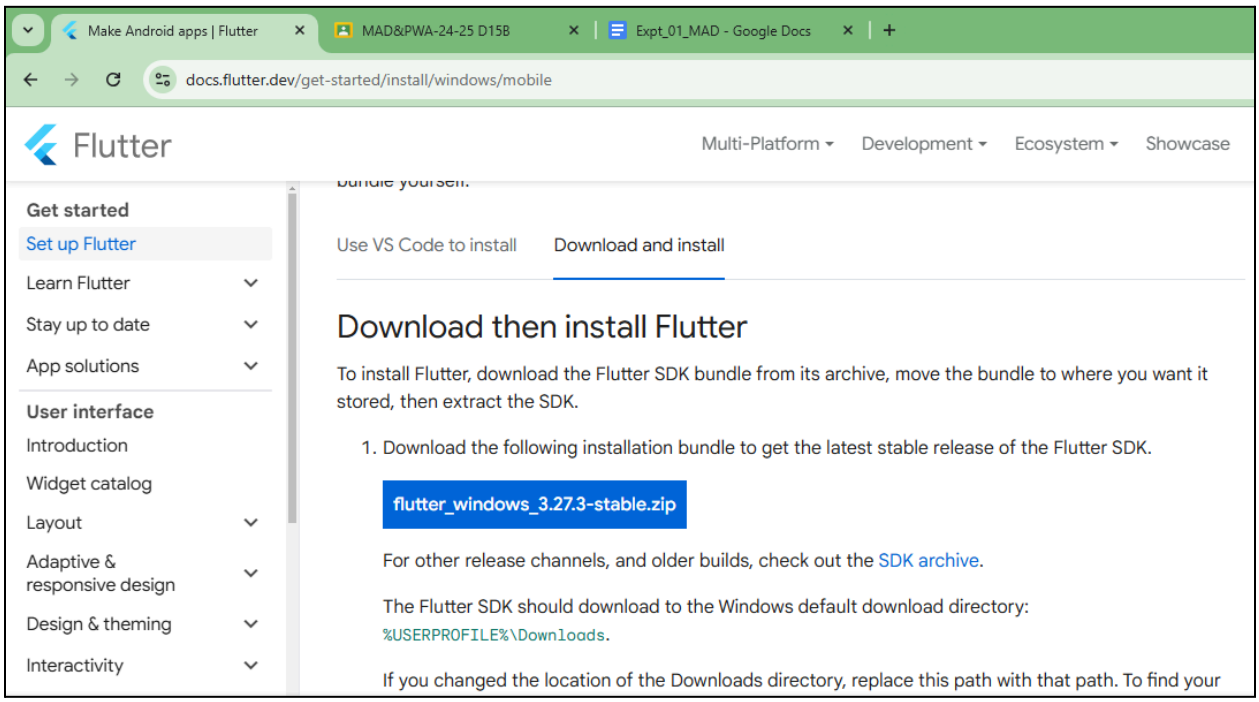
Aim: Installation and Configuration of Flutter Environment.

Install the Flutter SDK

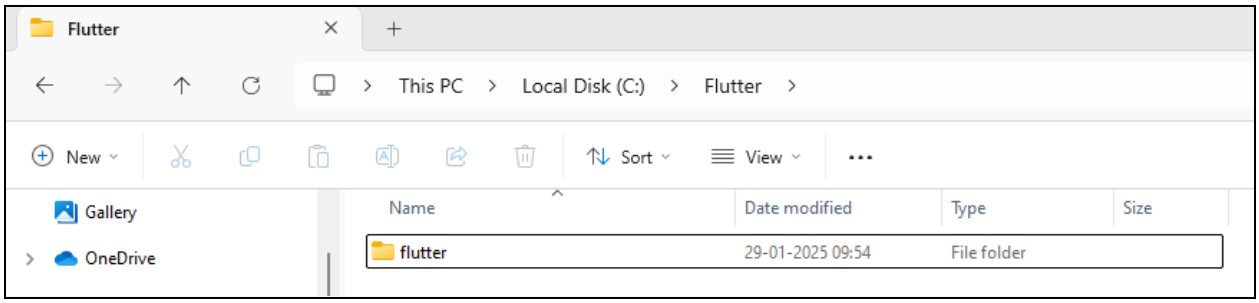
Step 1: Download the installation bundle of the Flutter Software Development Kit for windows. To download Flutter SDK, Go to its official website https://docs.flutter.dev/get-started/install , you will get the following screen.



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

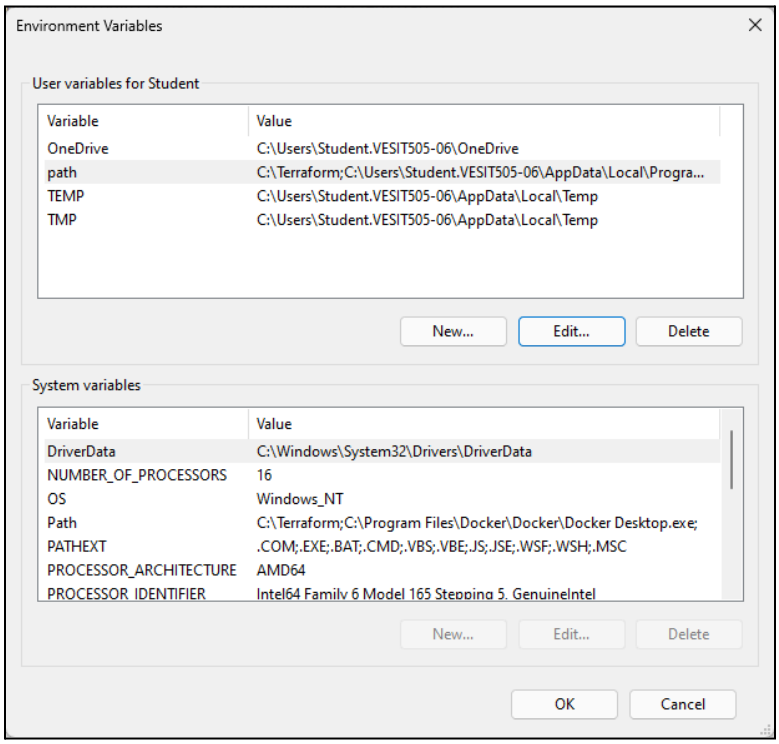


Step 3: When your download is complete, extract the zip file and place it in the desired installation folder or location, for example, C: /Flutter.



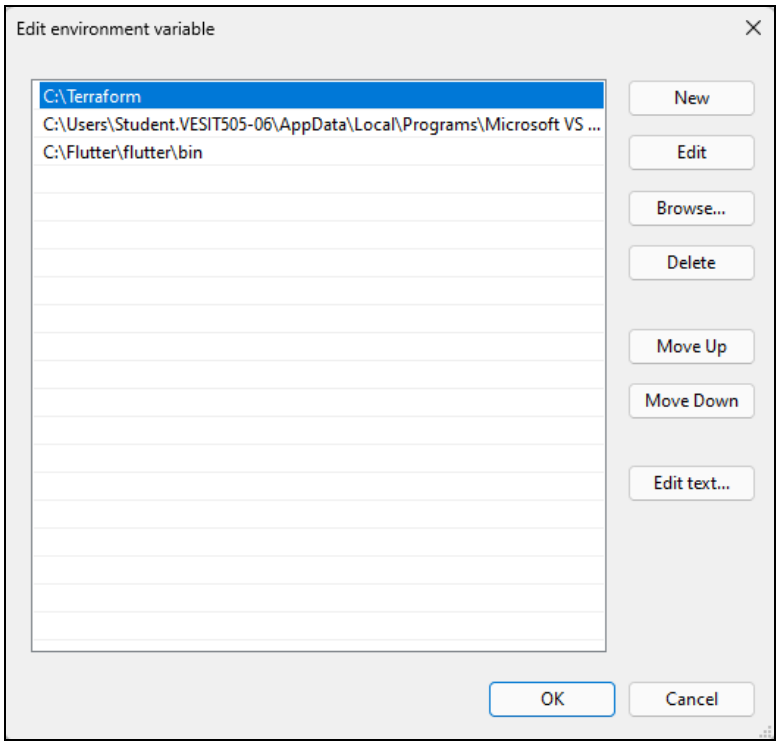
Step 4: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:

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

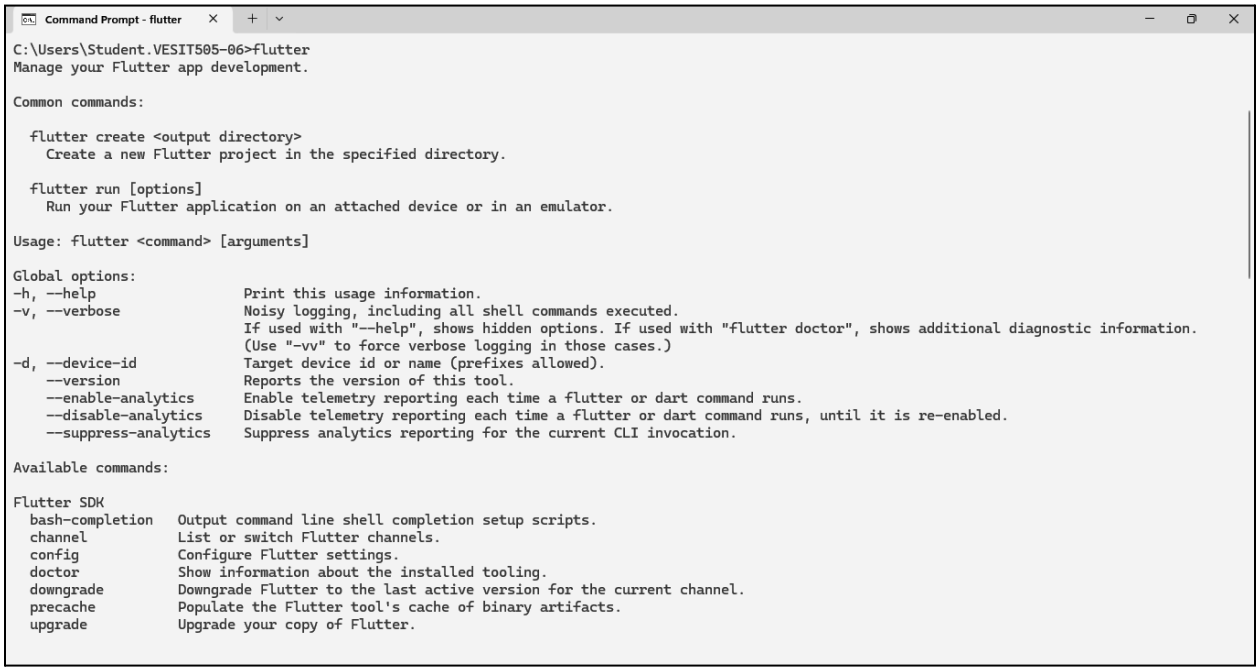


Step 4.2: Now, select path -> click on edit. The following screen appears

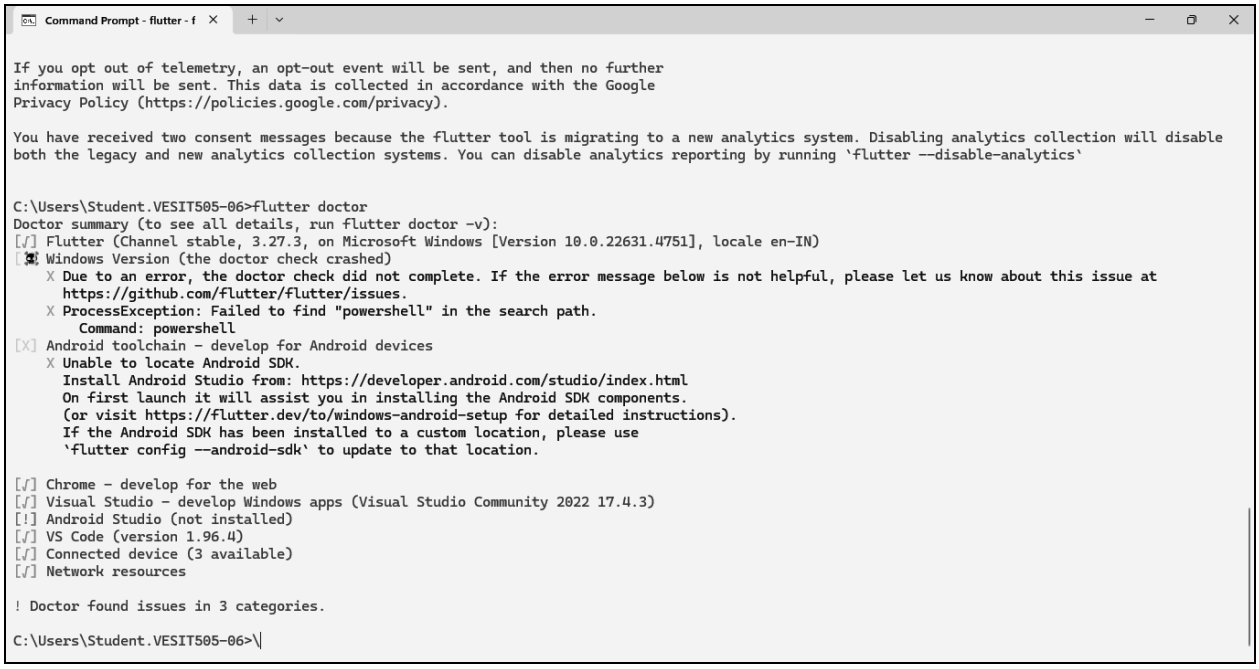
Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value -> ok -> ok -> ok.



Step 5: Now, run the $ flutter command in the command prompt.
Now, run the $ flutter doctor command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation.

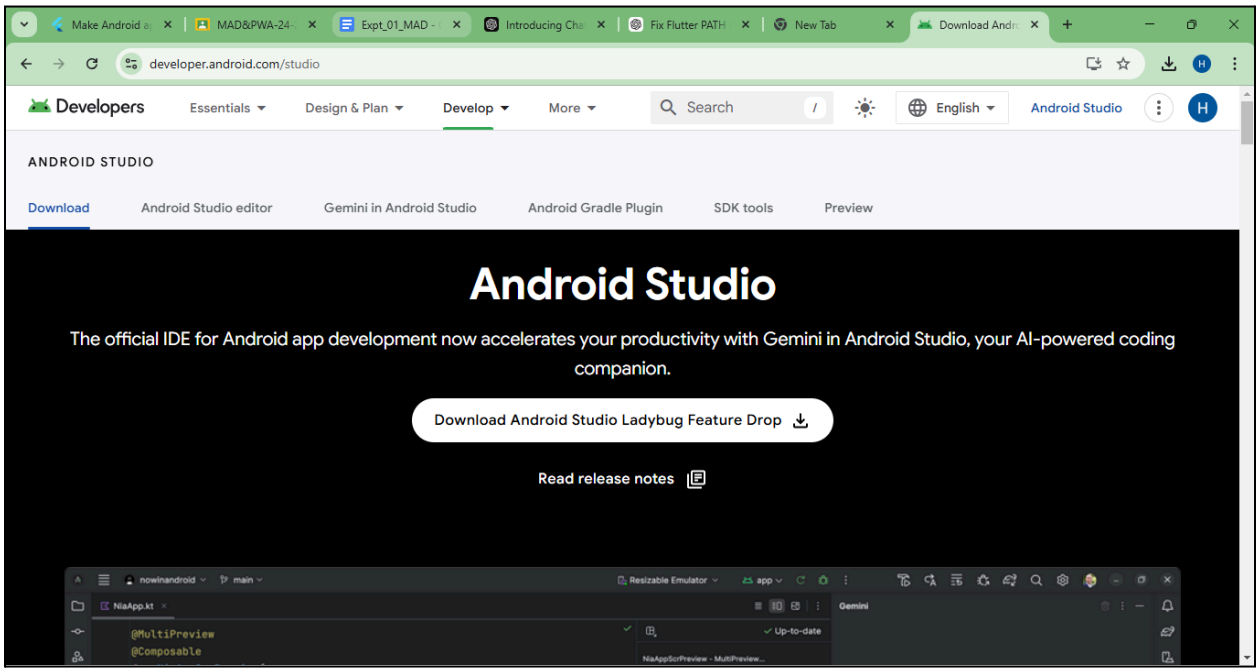


Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

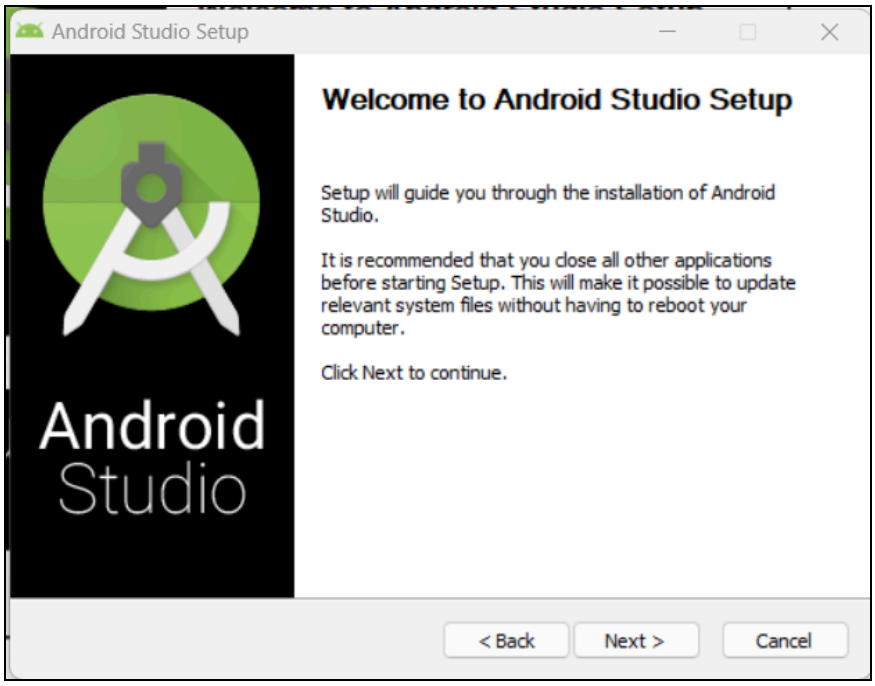


Step 7: Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.

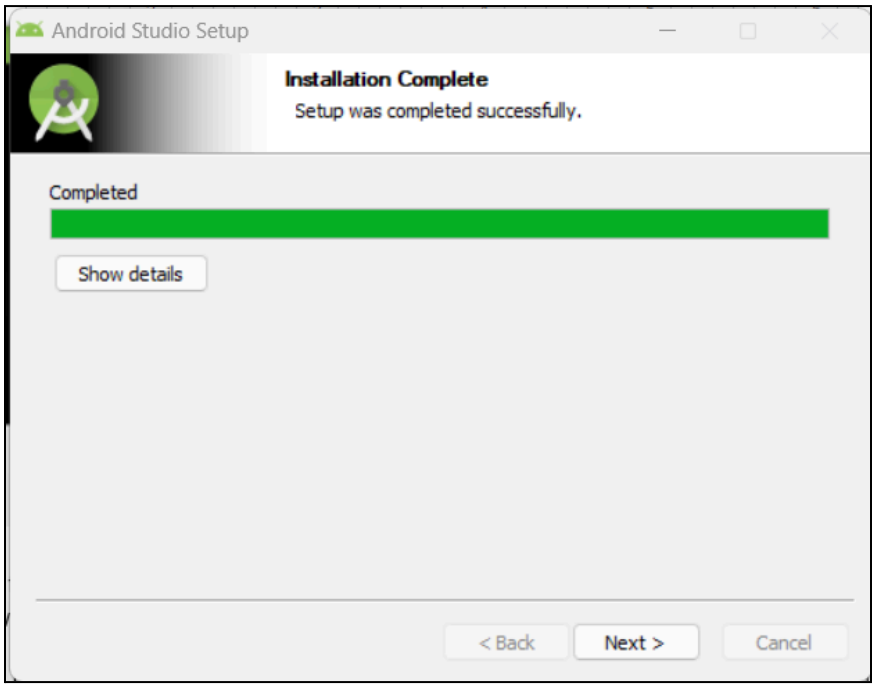
Step 7.1: Download the latest Android Studio executable or zip file from the official site.



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



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

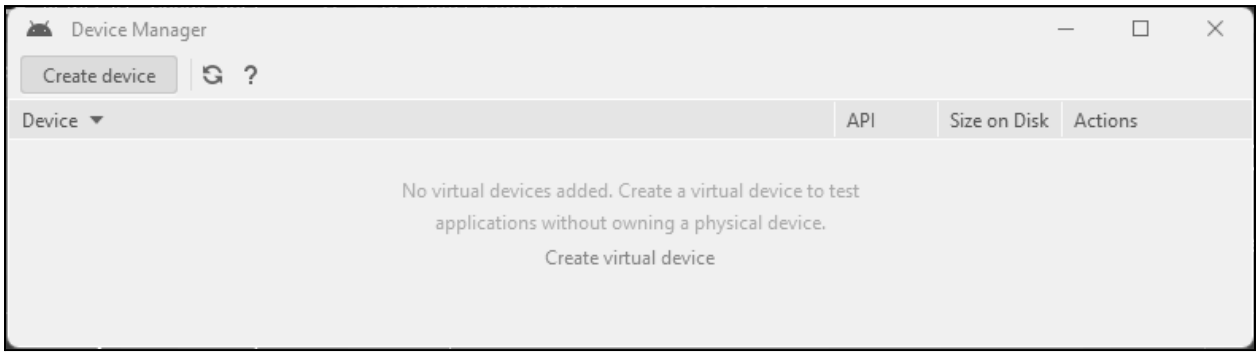


Step 7.4: In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to choose the 'Don't import Settings option' and click OK. It will start the Android Studio.

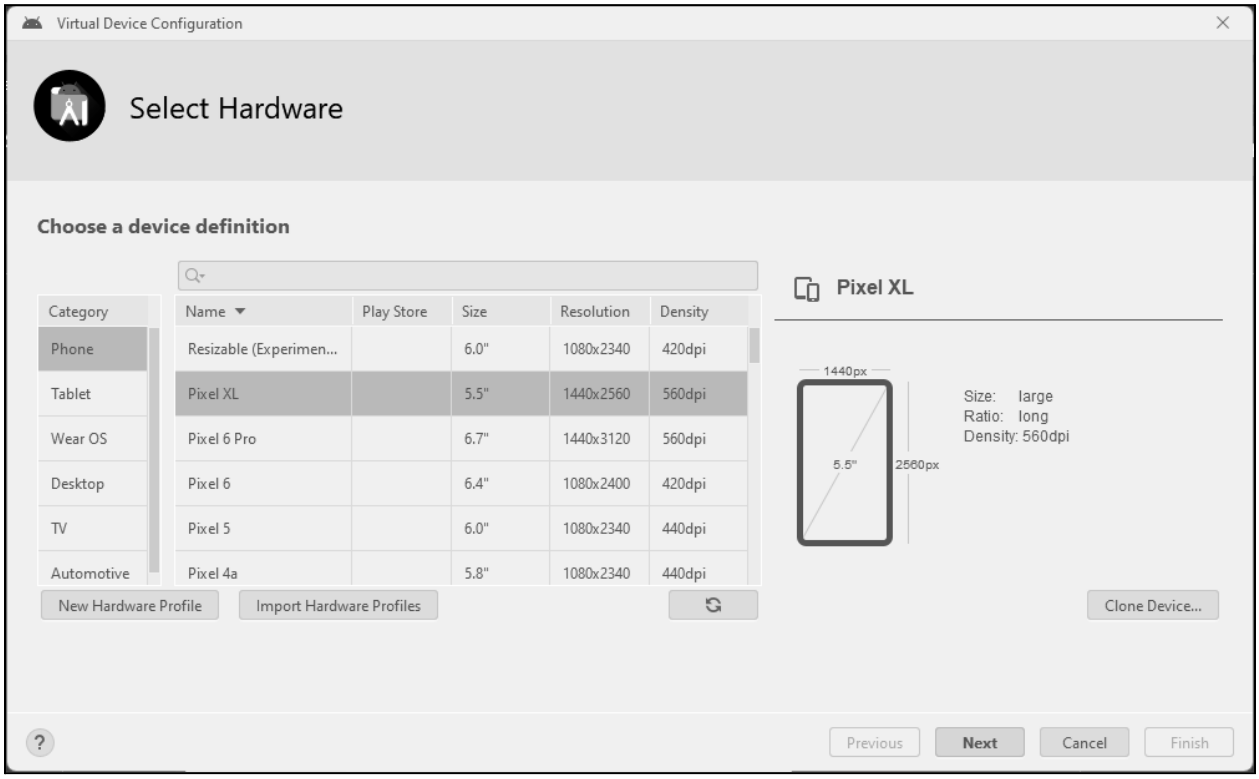
Step 7.5: run the $ flutter doctor command and Run flutter doctor --android-licenses command.

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

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

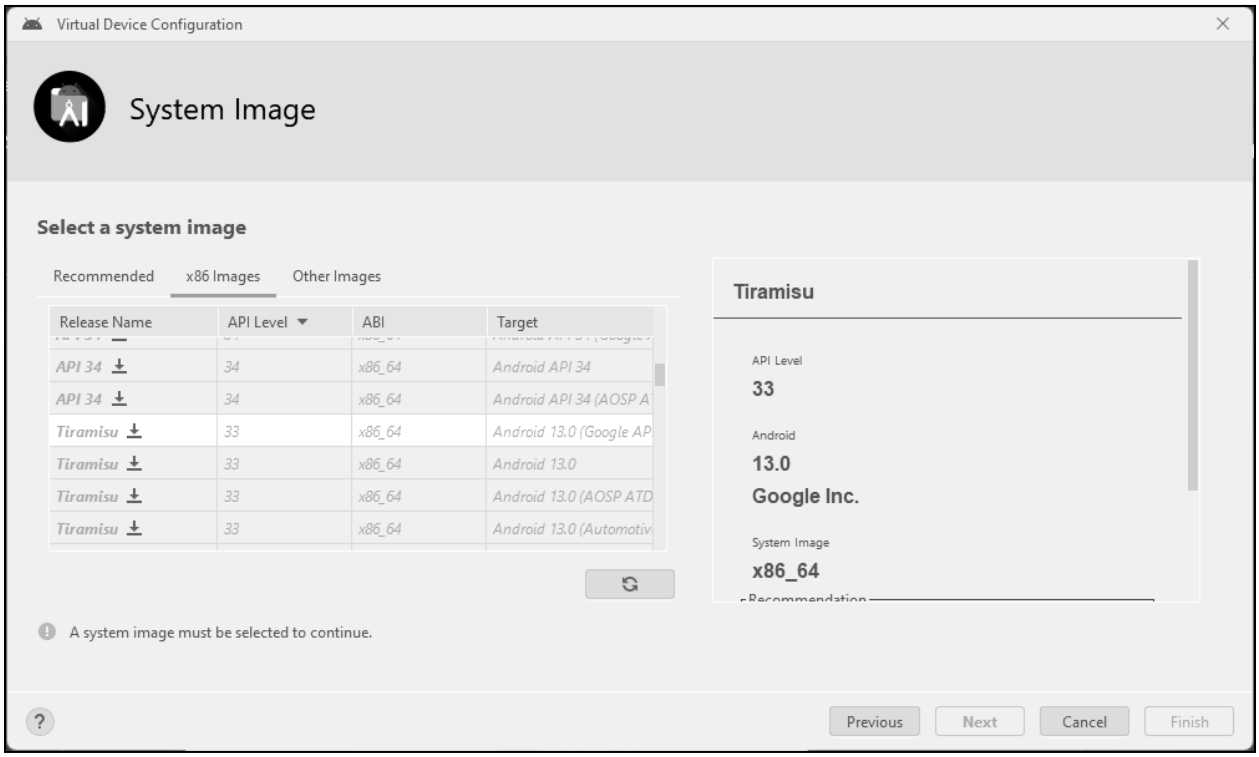


Step 8.2: Choose your device definition and click on Next.

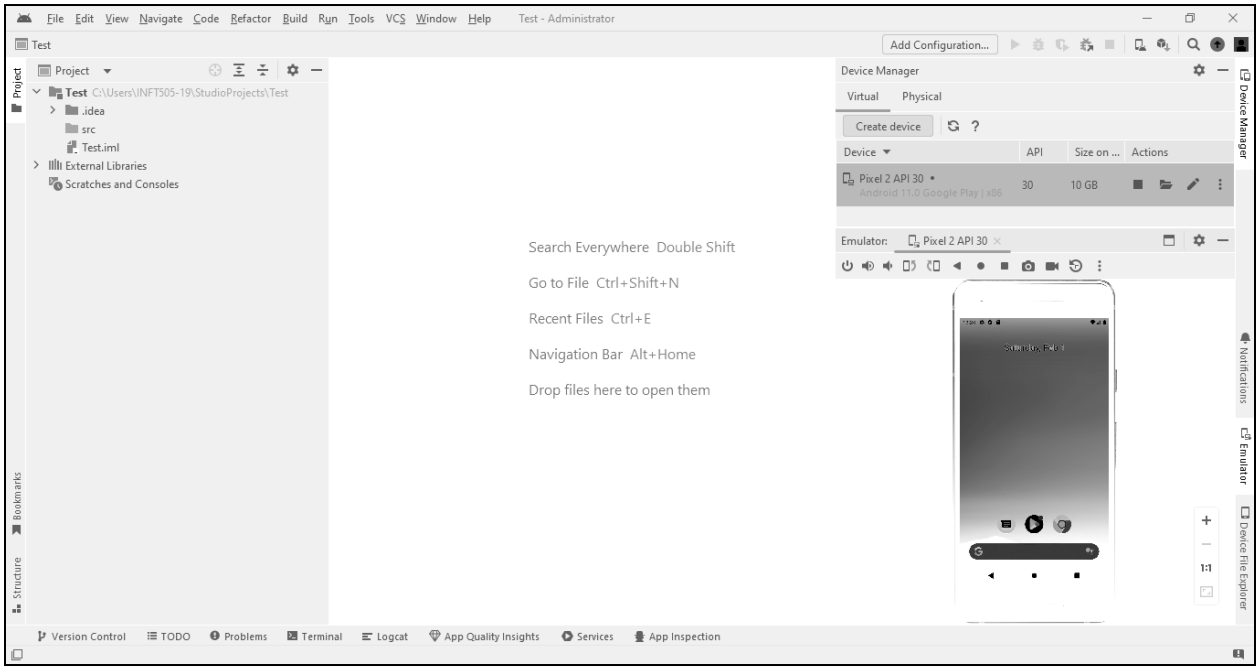


Step 8.3: Select the system image for the latest Android version and click on Next.

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

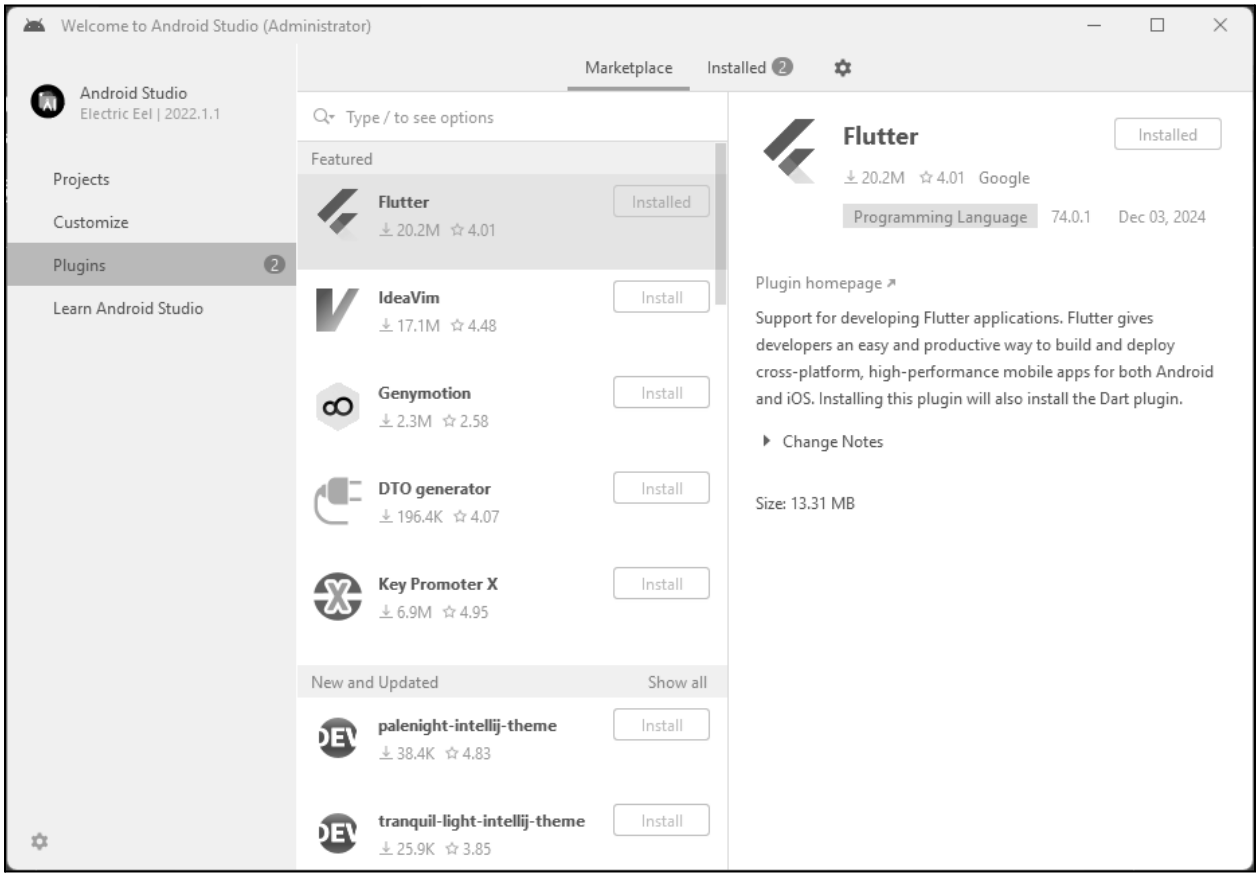


Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

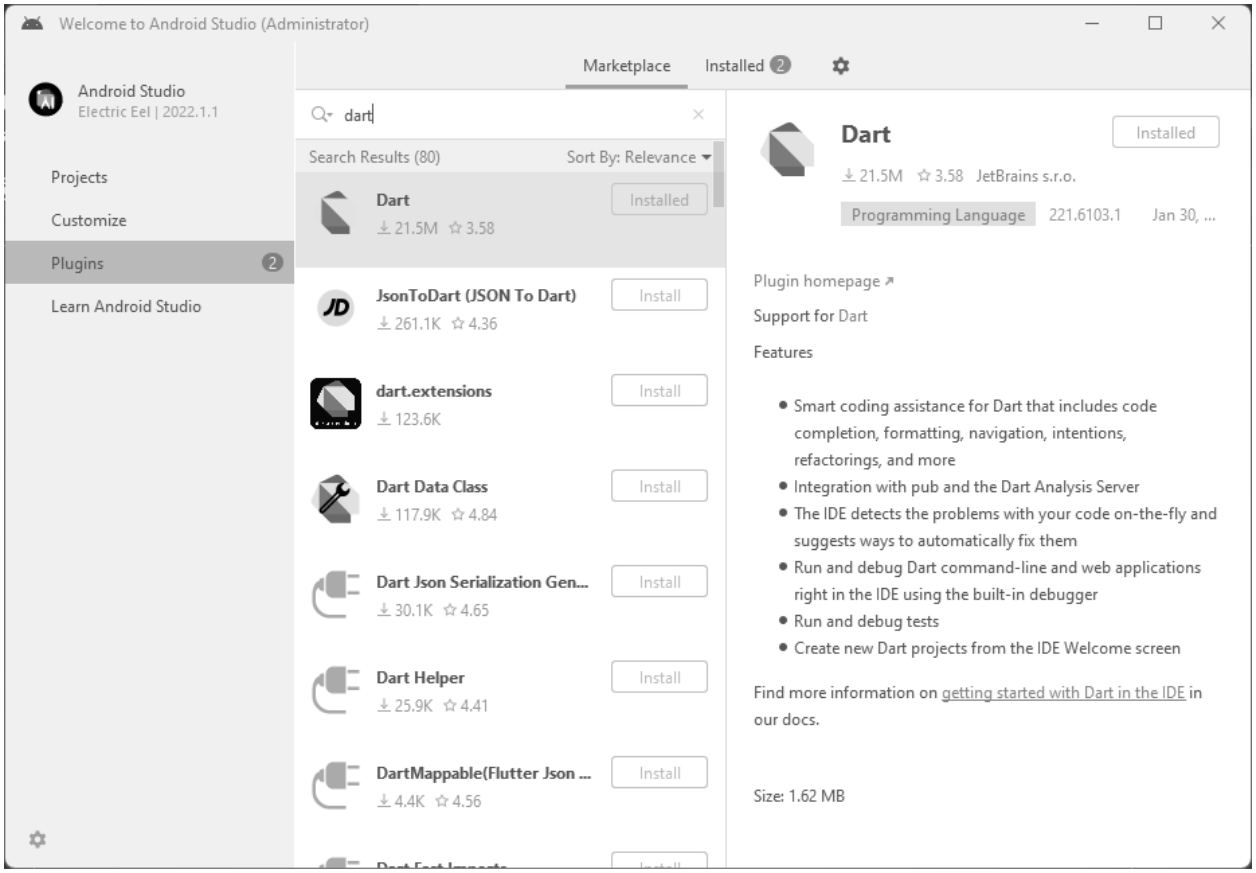


Step 9: Now, install the Flutter and Dart plugin for building Flutter applications in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.

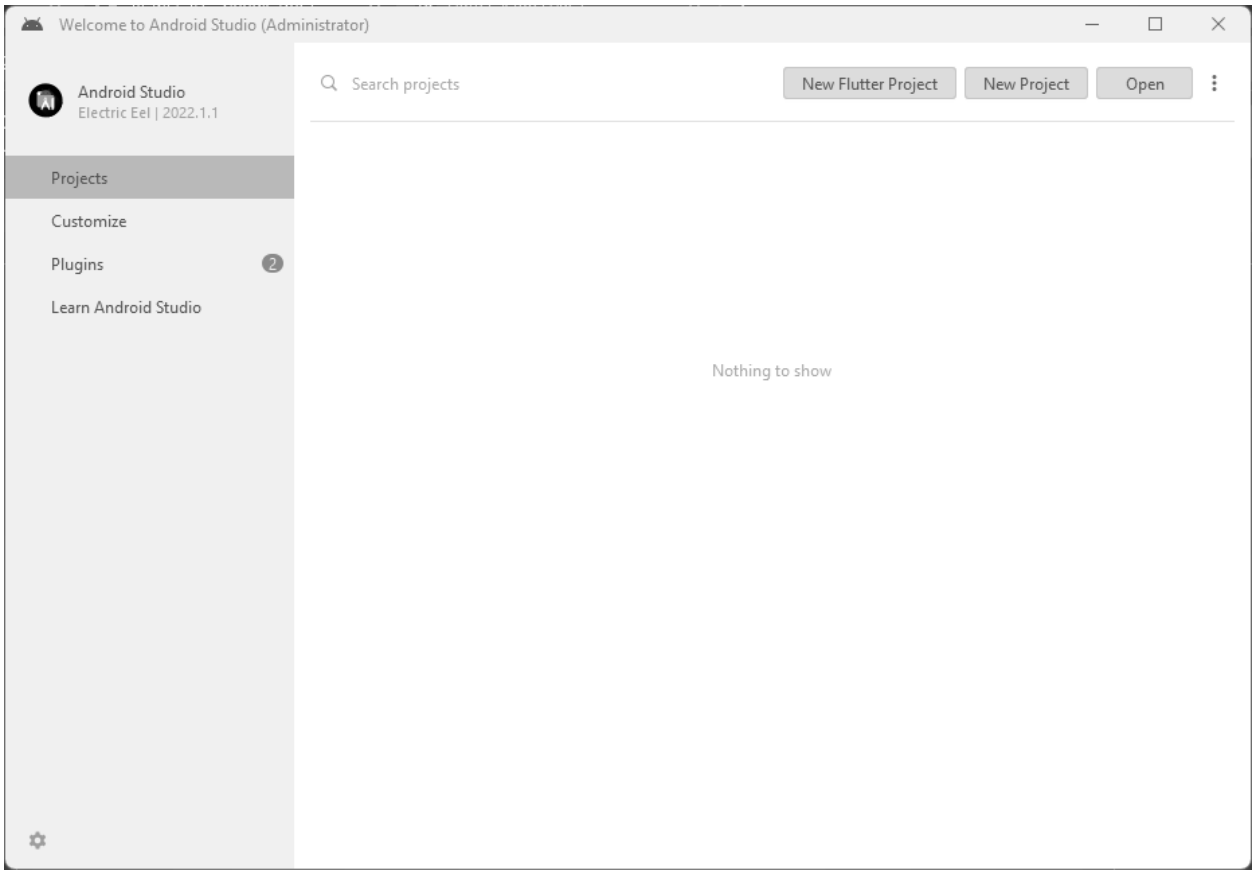
Step 9.1: Open the Android Studio and then go to File->Settings->Plugins.



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



Step 9.3: Restart the Android Studio.



Conclusion:

MPL Practical 02

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

Theory:

Introduction

MedLink is a mobile application designed to provide quick access to essential healthcare services. The app includes features like a symptom checker, medicine reminder, and an emergency location button with an offline map of nearby doctors. The goal is to make healthcare information and emergency support easily accessible.

Objective

The objective of this app is to assist users in managing their health by providing:

- An AI-powered symptom checker to analyze common health issues.
- A medicine reminder to help users take their medications on time.
- An emergency map to locate nearby hospitals and doctors, even offline.

Technologies Used

- Flutter: A cross-platform mobile app development framework.
- Dart: The programming language used for Flutter development.
- Material Design Widgets: Prebuilt UI components for a smooth user interface.
- Navigation in Flutter: Used to switch between different screens of the app.

Implementation Details

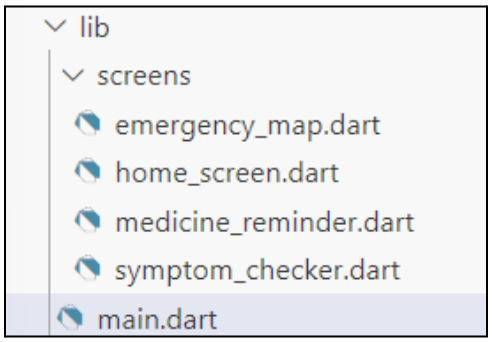
1. Home Screen
   - This is the main page of the app, which contains buttons to navigate to different features.
   - Three primary features are provided: Symptom Checker, Medicine Reminder, and Emergency Map.
   - Each feature is accessed using a simple ElevatedButton wrapped inside a Column widget.

2. Symptom Checker
   - This screen will analyze user-input symptoms and provide possible health conditions.
   - In this implementation, we have created a placeholder screen for future AI integration.

3. Medicine Reminder
   - This feature helps users keep track of their medicine schedules.
   - Currently, it is a placeholder, but in future versions, we can use notifications and local storage for reminders.

4. Emergency Map
   - This screen will help users locate nearby doctors and hospitals.
   - Right now, we have created a placeholder, but later we can integrate Google Maps API for real-time location tracking.

Folder Structure:



lib/main.dart

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

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

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

lib/screens/home_screen.dart

```
import 'package:flutter/material.dart';
import 'symptom_checker.dart';
import 'medicine_reminder.dart';
import 'emergency_map.dart';

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('MedLink - Medical Assistant')),
      body: Center(
```

```
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          FeatureButton(
            title: 'Symptom Checker',
            icon: Icons.health_and_safety,
            onTap: () => Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => SymptomCheckerScreen()),
            ),
          ),
          FeatureButton(
            title: 'Medicine Reminder',
            icon: Icons.alarm,
            onTap: () => Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => MedicineReminderScreen()),
            ),
          ),
          FeatureButton(
            title: 'Emergency Map',
            icon: Icons.map,
            onTap: () => Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => EmergencyMapScreen()),
            ),
          ),
        ],
      ),
    ),
  );
 }
}

class FeatureButton extends StatelessWidget {
 final String title;
 final IconData icon;
 final VoidCallback onTap;

 FeatureButton({required this.title, required this.icon, required this.onTap});

 @override
 Widget build(BuildContext context) {
  return Padding(
   padding: const EdgeInsets.all(8.0),
   child: ElevatedButton.icon(
    onPressed: onTap,
```

```
      icon: Icon(icon),
      label: Text(title),
      style: ElevatedButton.styleFrom(
        padding: EdgeInsets.symmetric(vertical: 12, horizontal: 20),
      ),
    ),
  );
 }
}
```

lib/screens/symptom_checker.dart

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

class SymptomCheckerScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(title: Text('Symptom Checker')),
   body: Center(child: Text('Symptom Checker Coming Soon!')),
  );
 }
}
```

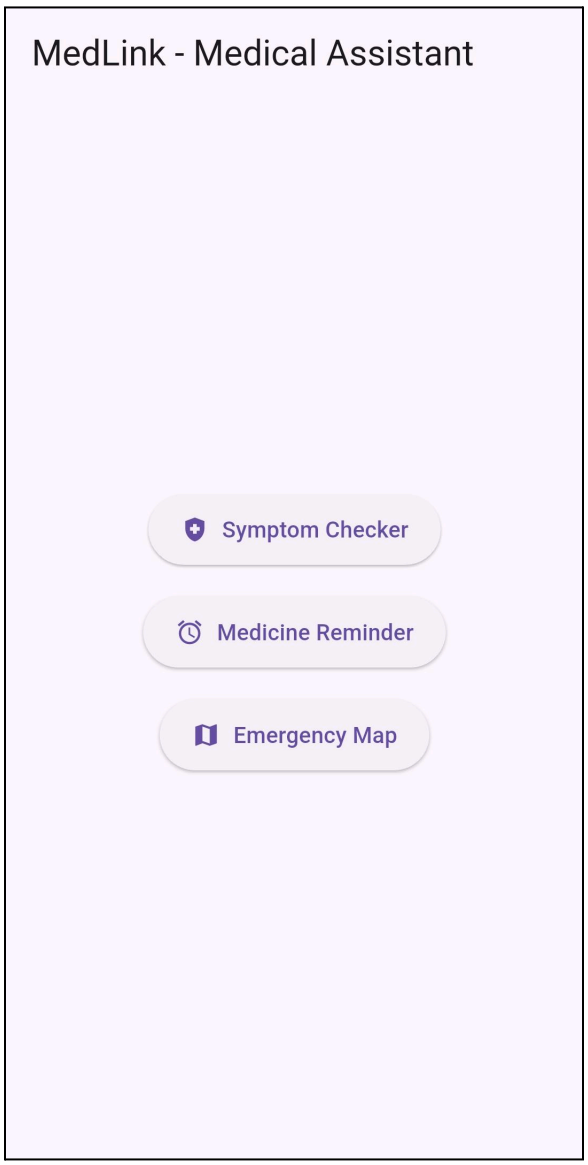
lib/screens/medicine_reminder.dart

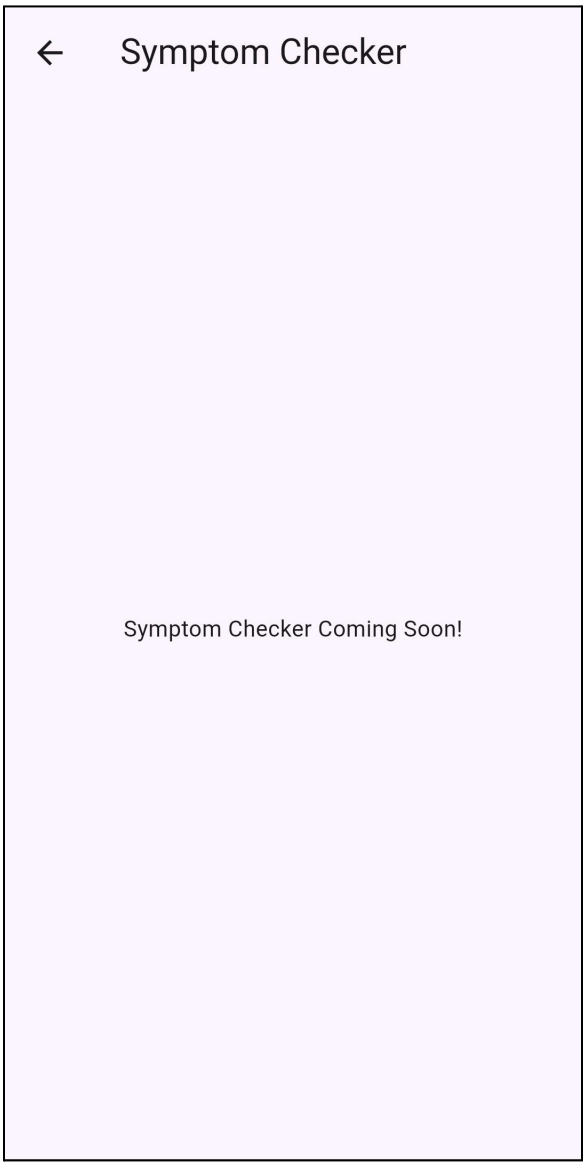
```
import 'package:flutter/material.dart';

class MedicineReminderScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(title: Text('Medicine Reminder')),
   body: Center(child: Text('Medicine Reminder Feature Coming Soon!')),
  );
 }
}
```

lib/screens/emergency_map.dart

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

class EmergencyMapScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(title: Text('Emergency Map')),
```

```
    body: Center(child: Text('Emergency Map Feature Coming Soon!')),
   );
  }
}
```





Conclusion

In this project, we implemented the basic UI for MedLink using Flutter, including navigation between screens for the Symptom Checker, Medicine Reminder, and Emergency Map. During development, we faced minor issues like incorrect widget placement and navigation errors, which we resolved by understanding Flutter's widget structure and using proper navigation techniques.

MPL Practical 03

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

Theory:

Introduction

Flutter allows developers to enhance the UI of an app by incorporating icons, images, and custom fonts. This improves the app's visual appeal and usability.

1. Icons in Flutter

Icons are used to represent actions, status, or navigation elements in an app. Flutter provides built-in icons using the Icons class and allows adding custom icons.

Implementation in Our Code:

- We used the Icons.health_and_safety for analysis results in the dialog box.
- The AppBar also includes an icon for navigation.

2. Images in Flutter

Images enhance UI by adding visual elements. Flutter supports images from assets, network, and memory.

Implementation in Our Code:

- We added an asset image (symptom_checker.png) at the top of the symptom checker screen.
- Used Image.asset() to load images stored in the assets/images directory.

3. Custom Fonts in Flutter

Custom fonts improve the app's appearance and branding. Fonts are added via the pubspec.yaml file and applied using the TextStyle widget.

Implementation in Our Code:

- We used Poppins and Montserrat fonts for headings and text elements to enhance readability and design.

lib/main.dart

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

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

class MedLinkApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'MedLink',
      theme: ThemeData(
        // Custom colors from your theme
        primaryColor: Color(0xFF007BFF),
        colorScheme: ColorScheme.light(
          primary: Color(0xFF007BFF),
          secondary: Color(0xFF20C997),
          background: Color(0xFFF5F5F5),
          surface: Colors.white,
        ),
        // Custom font
        fontFamily: 'Montserrat',
        textTheme: TextTheme(
          headlineLarge: TextStyle(
            fontFamily: 'Poppins',
            fontWeight: FontWeight.bold,
          ),
          headlineMedium: TextStyle(
            fontFamily: 'Poppins',
            fontWeight: FontWeight.w600,
          ),
          bodyLarge: TextStyle(fontFamily: 'Montserrat'),
          bodyMedium: TextStyle(fontFamily: 'Montserrat'),
        ),
        // Button theme
        elevatedButtonTheme: ElevatedButtonThemeData(
          style: ElevatedButton.styleFrom(
            backgroundColor: Color(0xFF007BFF),
```

```
          foregroundColor: Colors.white,
          padding: EdgeInsets.symmetric(vertical: 12, horizontal: 20),
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(10),
          ),
        ),
      ),
    ),
    home: HomeScreen(),
  );
 }
}
```

lib/screens/home_screen.dart

```
import 'package:flutter/material.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';
import 'symptom_checker.dart';
import 'medicine_reminder.dart';
import 'emergency_map.dart';

class HomeScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text(
     'MedLink',
     style: TextStyle(
      fontFamily: 'Poppins',
      fontWeight: FontWeight.bold,
     ),
    ),
    backgroundColor: Theme.of(context).colorScheme.primary,
    elevation: 0,
   ),
   body: Container(
    decoration: BoxDecoration(
     color: Theme.of(context).colorScheme.background,
    ),
    child: Column(
     children: [
```

```
// Header with logo image
Container(
  padding: EdgeInsets.all(20),
  color: Theme.of(context).colorScheme.primary,
  child: Center(
    child: Column(
      children: [
        Icon(
          FontAwesomeIcons.stethoscope, // Adjusted Icon
          size: 80,
          color: Colors.white,
        ),
        SizedBox(height: 16),
        Text(
          'Your Medical Assistant',
          style: TextStyle(
            color: Colors.white,
            fontSize: 18,
            fontFamily: 'Montserrat',
          ),
        ),
      ],
    ),
  ),
),

Expanded(
  child: Padding(
    padding: const EdgeInsets.all(20.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(
          'Medical Services',
          style: Theme.of(context).textTheme.headlineMedium?.copyWith(
                color: Theme.of(context).colorScheme.primary,
              ),
        ),
        SizedBox(height: 24),
        Expanded(
          child: GridView.count(
            crossAxisCount: 2,
```

```
            crossAxisSpacing: 16,
            mainAxisSpacing: 16,
            children: [
              FeatureTile(
                title: 'Symptom Checker',
                icon: FontAwesomeIcons.diagnoses, // Adjusted Icon
                color: Theme.of(context).colorScheme.primary,
                onTap: () => Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => SymptomCheckerScreen()),
                ),
              ),
              FeatureTile(
                title: 'Medicine Reminder',
                icon: FontAwesomeIcons.pills, // Adjusted Icon
                color: Theme.of(context).colorScheme.secondary,
                onTap: () => Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => MedicineReminderScreen()),
                ),
              ),
              FeatureTile(
                title: 'Emergency Map',
                icon: FontAwesomeIcons.mapMarkedAlt, // Adjusted Icon
                color: Color(0xFFFF6B6B),
                onTap: () => Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => EmergencyMapScreen()),
                ),
              ),
              FeatureTile(
                title: 'SOS Alert',
                icon: FontAwesomeIcons.exclamationTriangle, // Adjusted Icon
                color: Color(0xFFFFB347),
                onTap: () => _showSOSDialog(context),
              ),
            ],
          ),
        ),
      ],
    ),
  ),
);
```

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

 void _showSOSDialog(BuildContext context) {
  showDialog(
   context: context,
   builder: (BuildContext context) {
    return AlertDialog(
     title: Text('SOS Alert'),
     content: Text('This feature will be available soon!'),
     actions: [
      TextButton(
       child: Text('OK'),
       onPressed: () {
        Navigator.of(context).pop();
       },
      ),
     ],
    );
   },
  );
 }
}

class FeatureTile extends StatelessWidget {
 final String title;
 final IconData icon;
 final Color color;
 final VoidCallback onTap;

 const FeatureTile({
  required this.title,
  required this.icon,
  required this.color,
  required this.onTap,
 });

 @override
```

```
Widget build(BuildContext context) {
  return InkWell(
    onTap: onTap,
    borderRadius: BorderRadius.circular(16),
    child: Container(
      decoration: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.circular(16),
        boxShadow: [
          BoxShadow(
            color: Colors.black.withOpacity(0.05),
            blurRadius: 10,
            offset: Offset(0, 4),
          ),
        ],
      ),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Container(
            padding: EdgeInsets.all(12),
            decoration: BoxDecoration(
              color: color.withOpacity(0.1),
              shape: BoxShape.circle,
            ),
            child: Icon(
              icon,
              size: 40,
              color: color,
            ),
          ),
          SizedBox(height: 12),
          Text(
            title,
            style: TextStyle(
              fontFamily: 'Montserrat',
              fontWeight: FontWeight.w600,
              fontSize: 14,
            ),
            textAlign: TextAlign.center,
          ),
        ],
```

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

lib/screens/symptom_checker.dart

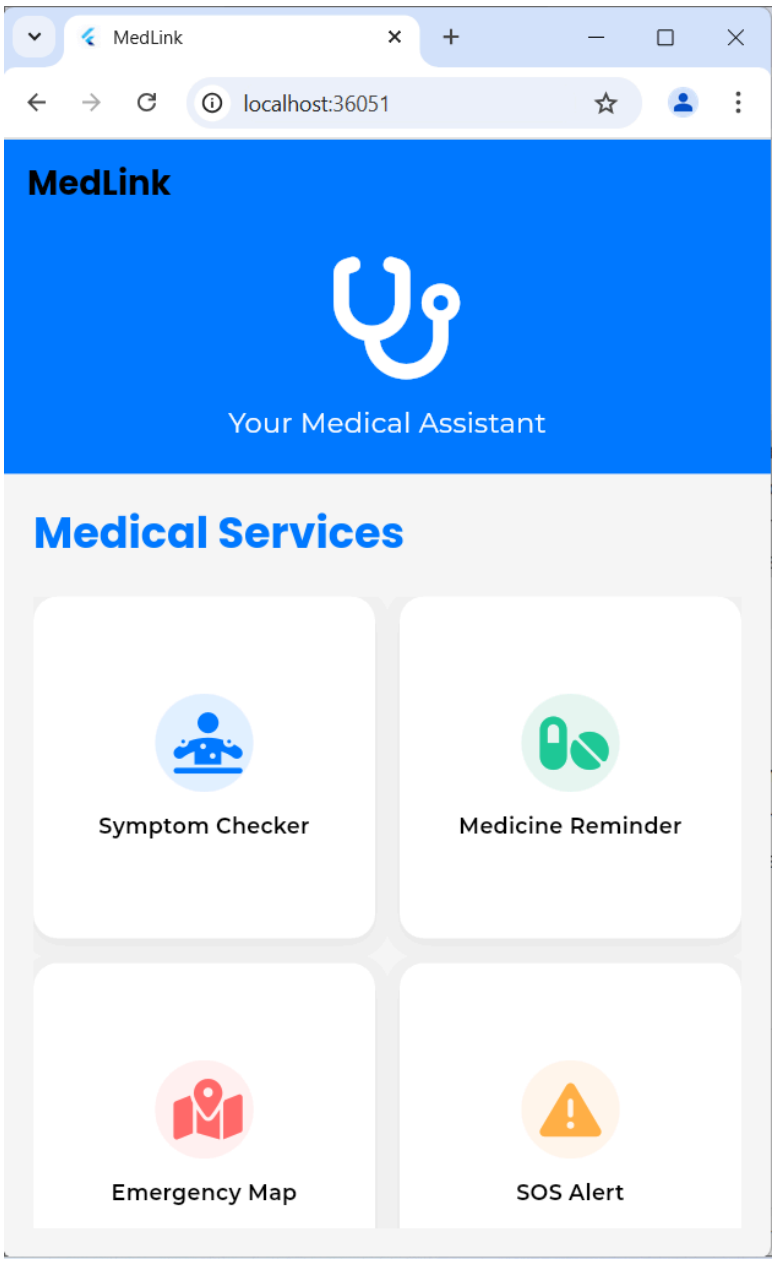
```
import 'package:flutter/material.dart';

class SymptomCheckerScreen extends StatefulWidget {
 @override
 _SymptomCheckerScreenState createState() => _SymptomCheckerScreenState();
}

class _SymptomCheckerScreenState extends State<SymptomCheckerScreen> {
 final List<String> commonSymptoms = [
  'Headache',
  'Fever',
  'Cough',
  'Sore Throat',
  'Fatigue',
  'Shortness of Breath',
  'Nausea'
 ];

 final List<bool> selectedSymptoms = List.filled(7, false);

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('Symptom Checker'),
    backgroundColor: Theme.of(context).colorScheme.primary,
   ),
   body: SingleChildScrollView(
    child: Padding(
     padding: const EdgeInsets.all(16.0),
     child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
       // Header with image
```

```
Center(
  child: Column(
    children: [
      Image.asset(
        'assets/images/symptom_checker.png',
        height: 150,
      ),
      SizedBox(height: 16),
      Text(
        'How are you feeling today?',
        style: TextStyle(
          fontFamily: 'Poppins',
          fontSize: 20,
          fontWeight: FontWeight.bold,
          color: Theme.of(context).colorScheme.primary,
        ),
      ),
      SizedBox(height: 8),
      Text(
        'Select your symptoms below',
        style: TextStyle(
          fontFamily: 'Montserrat',
          fontSize: 16,
          color: Colors.grey[600],
        ),
      ),
    ],
  ),
),

SizedBox(height: 24),

// Symptoms list
Text(
  'Common Symptoms',
  style: TextStyle(
    fontFamily: 'Poppins',
    fontSize: 18,
    fontWeight: FontWeight.w600,
  ),
),
SizedBox(height: 12),
```

```
ListView.builder(
  shrinkWrap: true,
  physics: NeverScrollableScrollPhysics(),
  itemCount: commonSymptoms.length,
  itemBuilder: (context, index) {
    return CheckboxListTile(
      title: Text(
        commonSymptoms[index],
        style: TextStyle(fontFamily: 'Montserrat'),
      ),
      value: selectedSymptoms[index],
      activeColor: Theme.of(context).colorScheme.primary,
      onChanged: (bool? value) {
        setState(() {
          selectedSymptoms[index] = value ?? false;
        });
      },
    );
  },
),

SizedBox(height: 24),

// Check button
Center(
  child: ElevatedButton(
    onPressed: () {
      // Show a dialog with a placeholder result
      _showResultDialog(context);
    },
    child: Padding(
      padding: const EdgeInsets.symmetric(horizontal: 32.0, vertical: 12.0),
      child: Text(
        'Check Symptoms',
        style: TextStyle(
          fontFamily: 'Montserrat',
          fontSize: 16,
          fontWeight: FontWeight.bold,
        ),
      ),
    ),
  ),
```

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

 void _showResultDialog(BuildContext context) {
  showDialog(
   context: context,
   builder: (BuildContext context) {
    return AlertDialog(
     title: Row(
      children: [
       Icon(
        Icons.health_and_safety,
        color: Theme.of(context).colorScheme.primary,
       ),
       SizedBox(width: 8),
       Text(
        'Analysis Result',
        style: TextStyle(
         fontFamily: 'Poppins',
         fontWeight: FontWeight.bold,
         color: Theme.of(context).colorScheme.primary,
        ),
       ),
      ],
     ),
     content: Text(
        'Based on your symptoms, you may have a common cold. Please consult with a healthcare professional for proper diagnosis.',
      style: TextStyle(
       fontFamily: 'Montserrat',
      ),
     ),
     actions: [
      TextButton(
       child: Text('OK'),
       onPressed: () {
```

```
            Navigator.of(context).pop();
          },
        ),
      ],
    );
  },
 );
 }
}
```



Conclusion

In this experiment, we successfully implemented icons, images, and custom fonts to enhance our Flutter app's UI. Initially, we faced issues with icons not displaying properly, which we resolved by ensuring the correct icon usage and asset paths in our project.

MPL Practical 04

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

Theory:

Forms are a fundamental part of mobile applications, enabling users to input and submit data. In Flutter, the Form widget provides a structured way to handle user inputs along with validation. The TextFormField widget is commonly used within a form to accept inputs such as email and password. Validation ensures that user credentials follow the required format before submission.

Implementation in MedLink App
In this experiment, we implemented a Login and Signup form for the MedLink app. The app:

- Redirects users to the Login page on first launch.
- Validates user credentials using the email_validator package.
- Ensures passwords contain at least one uppercase letter, one lowercase letter, one number, and one special character.
- Allows users to switch between Login and Signup screens.
- Navigates to the Home screen after successful login.

---

lib/screens/login_screen.dart

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

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

class _LoginScreenState extends State<LoginScreen> {
  final _formKey = GlobalKey<FormState>();
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();
  void _login() {
    if (_formKey.currentState!.validate()) {
      // Simulating authentication (you can replace it with actual authentication logic)
      Navigator.pushReplacement(
        context,
```

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

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Theme.of(context).colorScheme.background,
    body: Center(
      child: Padding(
        padding: EdgeInsets.all(20),
        child: Form(
          key: _formKey,
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text(
                'MedLink Login',
                style: TextStyle(
                  fontFamily: 'Poppins',
                  fontSize: 24,
                  fontWeight: FontWeight.bold,
                  color: Theme.of(context).colorScheme.primary,
                ),
              ),
              SizedBox(height: 20),
              TextFormField(
                controller: emailController,
                keyboardType: TextInputType.emailAddress,
                decoration: InputDecoration(labelText: 'Email'),
                validator: (value) {
                  if (value == null || value.isEmpty) {
                    return 'Enter your email';
                  } else if (!EmailValidator.validate(value)) {
                    return 'Enter a valid email';
                  }
                  return null;
                },
              ),
```

```
            SizedBox(height: 10),
            TextFormField(
              controller: passwordController,
              obscureText: true,
              decoration: InputDecoration(labelText: 'Password'),
              validator: (value) {
                if (value == null || value.isEmpty) {
                  return 'Enter your password';
                                                                    } else if
(!RegExp(r'^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[!@#\$&*~]).{6,}$').hasMatch(value)) {
                    return 'Password must have 1 uppercase, 1 lowercase, 1 number, and 1 special
character';
                }
                return null;
              },
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: _login,
              child: Text('Login'),
            ),
            SizedBox(height: 10),
            TextButton(
              onPressed: () {
                Navigator.push(context, MaterialPageRoute(builder: (context) => SignupScreen()));
              },
              child: Text('Don\'t have an account? Sign up'),
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

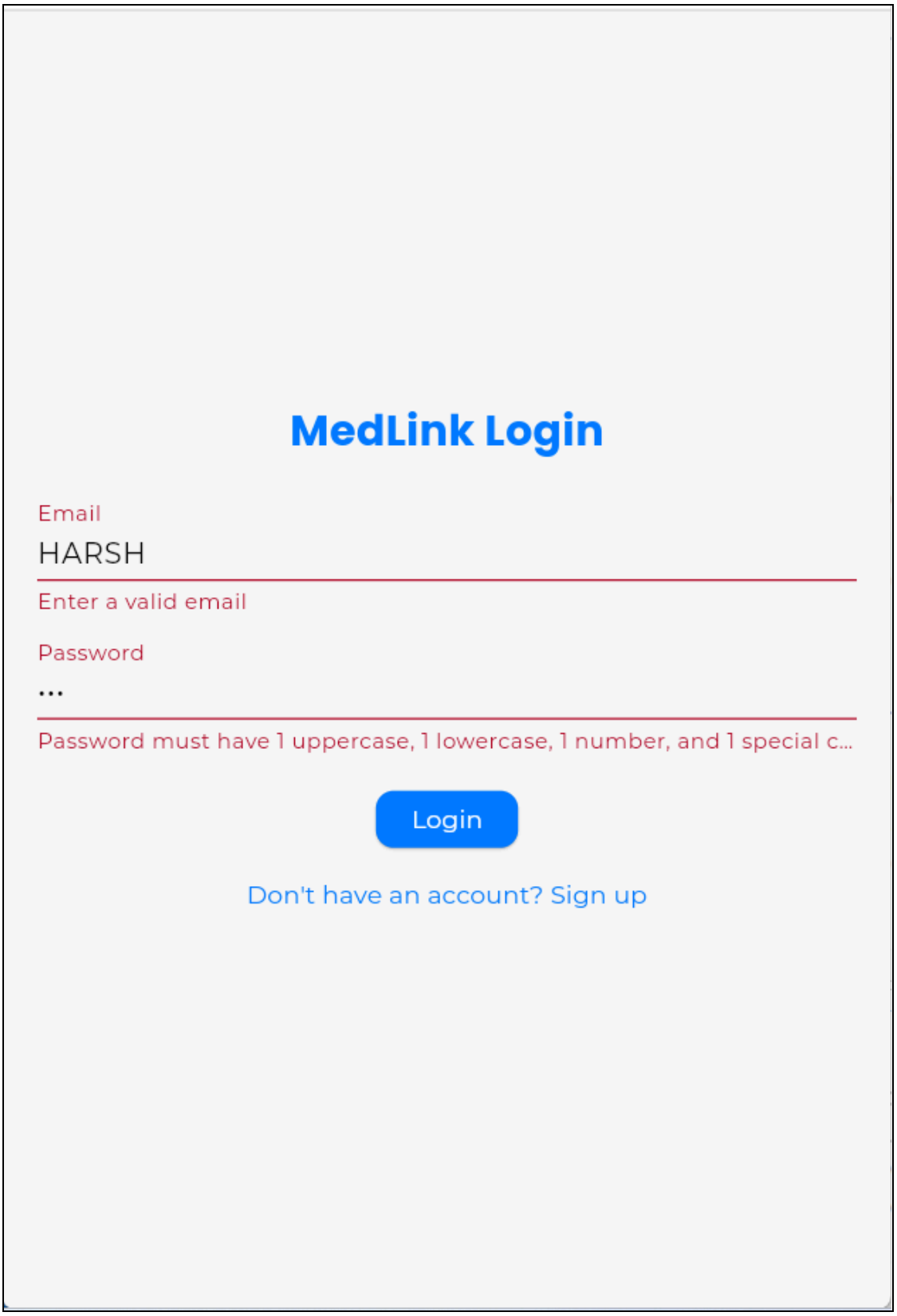
lib/screens/signup_screen.dart

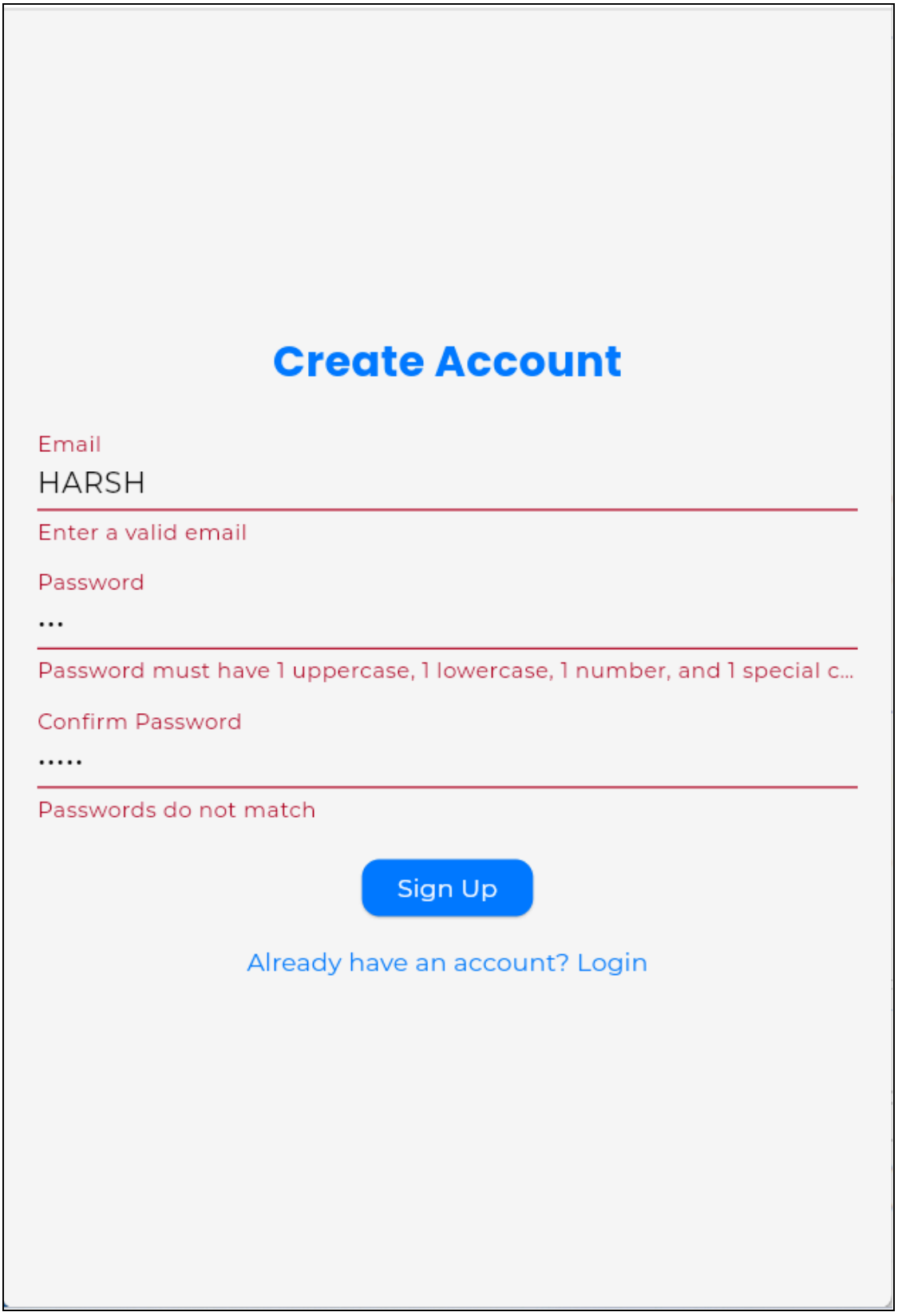
```
import 'package:flutter/material.dart';
import 'package:email_validator/email_validator.dart';
```

```
import 'login_screen.dart';

class SignupScreen extends StatefulWidget {
  @override
  _SignupScreenState createState() => _SignupScreenState();
}

class _SignupScreenState extends State<SignupScreen> {
  final _formKey = GlobalKey<FormState>();
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();
  final TextEditingController confirmPasswordController = TextEditingController();

  void _signup() {
    if (_formKey.currentState!.validate()) {
      // Simulating user registration (replace with actual logic)
                  Navigator.pushReplacement(context,   MaterialPageRoute(builder:   (context)   =>
LoginScreen()));
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Theme.of(context).colorScheme.background,
      body: Center(
        child: Padding(
          padding: EdgeInsets.all(20),
          child: Form(
            key: _formKey,
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Text(
                  'Create Account',
                  style: TextStyle(
                    fontFamily: 'Poppins',
                    fontSize: 24,
                    fontWeight: FontWeight.bold,
                    color: Theme.of(context).colorScheme.primary,
```

```
            ),
          ),
          SizedBox(height: 20),
          TextFormField(
            controller: emailController,
            keyboardType: TextInputType.emailAddress,
            decoration: InputDecoration(labelText: 'Email'),
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Enter your email';
              } else if (!EmailValidator.validate(value)) {
                return 'Enter a valid email';
              }
              return null;
            },
          ),
          SizedBox(height: 10),
          TextFormField(
            controller: passwordController,
            obscureText: true,
            decoration: InputDecoration(labelText: 'Password'),
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Enter a password';
                                                                    } else if
(!RegExp(r'^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[!@#\$&*~]).{6,}$').hasMatch(value)) {
                  return 'Password must have 1 uppercase, 1 lowercase, 1 number, and 1 special
character';
              }
              return null;
            },
          ),
          SizedBox(height: 10),
          TextFormField(
            controller: confirmPasswordController,
            obscureText: true,
            decoration: InputDecoration(labelText: 'Confirm Password'),
            validator: (value) {
              if (value != passwordController.text) {
                return 'Passwords do not match';
```

```
                }
                return null;
              },
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: _signup,
              child: Text('Sign Up'),
            ),
            SizedBox(height: 10),
            TextButton(
              onPressed: () {
                Navigator.push(context, MaterialPageRoute(builder: (context) => LoginScreen()));
              },
              child: Text('Already have an account? Login'),
            ),
          ],
        ),
      ),
    ),
  ),
 );
 }
}
```


MedLink Login
Email
HARSH
Enter a valid email
Password
...
Password must have 1 uppercase, 1 lowercase, 1 number, and 1 special c...
Login
Don't have an account? Sign up



Conclusion

In this experiment, we successfully implemented a Login and Signup form with proper validation using the Form widget in Flutter. Initially, we faced issues with email validation and password constraints, but we resolved them by integrating the email_validator package and using regular expressions for password validation, ensuring a smooth authentication process.

MPL Practical 05

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

Theory:

Navigation and routing are essential in Flutter applications to allow users to move between different screens. Flutter provides Navigator for managing screen transitions using push(), pop(), and pushNamed(). Named routes help in maintaining clean and structured navigation.

Gestures enhance user interaction by detecting taps, swipes, long presses, and double taps using the GestureDetector widget. This makes the app more user-friendly and dynamic.

Implementation in MedLink App
In this experiment, we implemented:

- Named Routes for structured navigation between screens.
- Swipe Right Gesture to navigate back.
- Tap, Double Tap & Long Press gestures for interactive elements.
- Snackbar Feedback to provide real-time responses.

---

lib/main.dart

```
import 'package:flutter/material.dart';
import 'screens/home_screen.dart';
import 'screens/login_screen.dart';
import 'screens/signup_screen.dart';
import 'screens/symptom_checker.dart';
import 'screens/medicine_reminder.dart';
import 'screens/emergency_map.dart';

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

class MedLinkApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'MedLink',
      theme: ThemeData(
```

```
      primaryColor: Color(0xFF007BFF),
      colorScheme: ColorScheme.light(
        primary: Color(0xFF007BFF),
        secondary: Color(0xFF20C997),
        background: Color(0xFFF5F5F5),
        surface: Colors.white,
      ),
      fontFamily: 'Montserrat',
    ),
    initialRoute: '/',
    routes: {
      '/': (context) => LoginScreen(),
      '/signup': (context) => SignupScreen(),
      '/home': (context) => HomeScreen(),
      '/symptom_checker': (context) => SymptomCheckerScreen(),
      '/medicine_reminder': (context) => MedicineReminderScreen(),
      '/emergency_map': (context) => EmergencyMapScreen(),
    },
  );
 }
}
```

lib/screens/home_screen.dart

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

class HomeScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    leading: Icon(
     FontAwesomeIcons.stethoscope, // Use FontAwesome if it's the correct icon
     color: Colors.white,
    ),
    title: Text(
     'MedLink',
     style: TextStyle(
      fontFamily: 'Poppins',
      fontWeight: FontWeight.bold,
```

```
        ),
      ),
      backgroundColor: Theme.of(context).colorScheme.primary,
    ),
    body: GestureDetector(
      onHorizontalDragEnd: (details) {
        if (details.primaryVelocity! > 0) {
          Navigator.pop(context); // Swipe right to go back
        }
      },
      child: Padding(
        padding: const EdgeInsets.all(20.0),
        child: GridView.count(
          crossAxisCount: 2,
          crossAxisSpacing: 16,
          mainAxisSpacing: 16,
          children: [
            FeatureTile(
              title: 'Symptom Checker',
              icon: FontAwesomeIcons.diagnoses,
              color: Theme.of(context).colorScheme.primary,
              route: '/symptom_checker',
            ),
            FeatureTile(
              title: 'Medicine Reminder',
              icon: FontAwesomeIcons.pills,
              color: Theme.of(context).colorScheme.secondary,
              route: '/medicine_reminder',
            ),
            FeatureTile(
              title: 'Emergency Map',
              icon: FontAwesomeIcons.mapMarkedAlt,
              color: Color(0xFFFF6B6B),
              route: '/emergency_map',
            ),
          ],
        ),
      ),
    ),
  );
```

```
  }
}

class FeatureTile extends StatelessWidget {
  final String title;
  final IconData icon;
  final Color color;
  final String route;

  const FeatureTile({
    required this.title,
    required this.icon,
    required this.color,
    required this.route,
  });

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        Navigator.pushNamed(context, route);
      },
      child: Container(
        decoration: BoxDecoration(
          color: Colors.white,
          borderRadius: BorderRadius.circular(16),
          boxShadow: [
            BoxShadow(
              color: Colors.black.withOpacity(0.05),
              blurRadius: 10,
              offset: Offset(0, 4),
            ),
          ],
        ),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(
              icon,
              size: 40,
```

```
            color: color,
          ),
          SizedBox(height: 12),
          Text(
            title,
            style: TextStyle(
              fontFamily: 'Montserrat',
              fontWeight: FontWeight.w600,
              fontSize: 14,
            ),
            textAlign: TextAlign.center,
          ),
        ],
      ),
    ),
  );
 }
}
```

lib/screens/symptom_checker.dart

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

class SymptomCheckerScreen extends StatefulWidget {
 @override
 _SymptomCheckerScreenState createState() => _SymptomCheckerScreenState();
}

class _SymptomCheckerScreenState extends State<SymptomCheckerScreen> {
 final List<String> commonSymptoms = [
  'Headache',
  'Fever',
  'Cough',
  'Sore Throat',
  'Fatigue',
  'Shortness of Breath',
  'Nausea'
 ];

 final List<bool> selectedSymptoms = List.filled(7, false);
```

```
@override
Widget build(BuildContext context) {
 return WillPopScope(
  onWillPop: () async {
   Navigator.of(context).pushReplacementNamed('/home');
   return false;
  },
  child: Scaffold(
   appBar: AppBar(
    leading: BackButton(
     color: Colors.white, // Ensure it's visible
     onPressed: () {
      Navigator.of(context).pushReplacementNamed('/home');
     },
    ),
    title: Text(
     'Symptom Checker',
     style: TextStyle(
      color: Colors.white,
      fontWeight: FontWeight.bold,
     ),
    ),
    backgroundColor: Color(0xFF007BFF),
   ),
   body: SingleChildScrollView(
    child: Padding(
     padding: const EdgeInsets.all(16.0),
     child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
       // Header with image
       Center(
        child: Column(
         children: [
          GestureDetector(
           onDoubleTap: () {
            _showHint(context);
           },
           child: Container(
            height: 180,
```

```
            width: 280,
            decoration: BoxDecoration(
              color: Colors.grey.shade100,
              borderRadius: BorderRadius.circular(12),
            ),
            child: Image.asset(
              'assets/images/symptom_checker.png',
              fit: BoxFit.contain,
            ),
          ),
        ),
        SizedBox(height: 24),
        Text(
          'How are you feeling today?',
          style: TextStyle(
            fontFamily: 'Poppins',
            fontSize: 22,
            fontWeight: FontWeight.bold,
            color: Color(0xFF007BFF),
          ),
        ),
        SizedBox(height: 12),
        Text(
          'Select your symptoms below',
          style: TextStyle(
            fontFamily: 'Montserrat',
            fontSize: 16,
            color: Colors.grey[600],
          ),
        ),
      ],
    ),
  ),

  SizedBox(height: 32),

  // Symptoms list
  Container(
    padding: EdgeInsets.symmetric(vertical: 8, horizontal: 4),
    child: Text(
```

```
    'Common Symptoms',
    style: TextStyle(
     fontFamily: 'Poppins',
     fontSize: 20,
     fontWeight: FontWeight.w600,
    ),
   ),
  ),
 ),
 SizedBox(height: 8),

 Container(
  decoration: BoxDecoration(
   color: Colors.white,
   borderRadius: BorderRadius.circular(12),
   boxShadow: [
    BoxShadow(
     color: Colors.grey.withOpacity(0.1),
     spreadRadius: 1,
     blurRadius: 2,
     offset: Offset(0, 1),
    ),
   ],
  ),
  child: ListView.separated(
   shrinkWrap: true,
   physics: NeverScrollableScrollPhysics(),
   itemCount: commonSymptoms.length,
   separatorBuilder: (context, index) => Divider(height: 1),
   itemBuilder: (context, index) {
    return GestureDetector(
     onLongPress: () {
      ScaffoldMessenger.of(context).showSnackBar(
       SnackBar(
        content: Text('Symptom: ${commonSymptoms[index]}'),
        duration: Duration(seconds: 2),
       ),
      );
     },
     child: CheckboxListTile(
      title: Text(
```

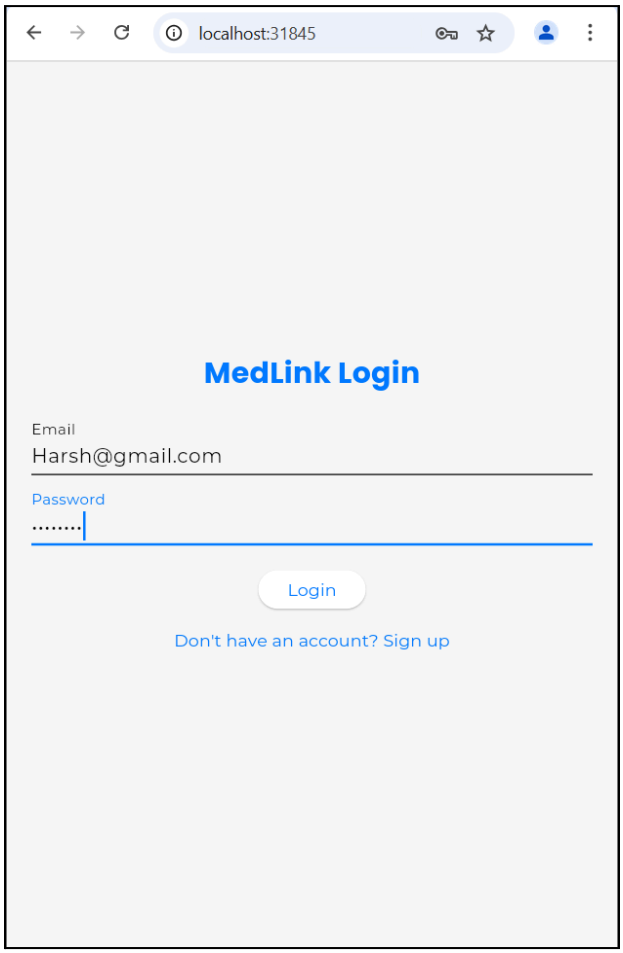
```
                commonSymptoms[index],
                style: TextStyle(
                  fontFamily: 'Montserrat',
                  fontSize: 16,
                  fontWeight: FontWeight.w500,
                ),
              ),
              value: selectedSymptoms[index],
              activeColor: Color(0xFF007BFF),
              checkColor: Colors.white,
              onChanged: (bool? value) {
                setState(() {
                  selectedSymptoms[index] = value ?? false;
                });
              },
            ),
          );
        },
      ),
    ),

    SizedBox(height: 36),

    // Check button
    Center(
      child: ElevatedButton(
        onPressed: () {
          _showResultDialog(context);
        },
        style: ElevatedButton.styleFrom(
          backgroundColor: Color(0xFF007BFF),
          foregroundColor: Colors.white,
          elevation: 2,
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(10),
          ),
          padding: EdgeInsets.symmetric(horizontal: 32, vertical: 16),
        ),
        child: Text(
          'Check Symptoms',
```

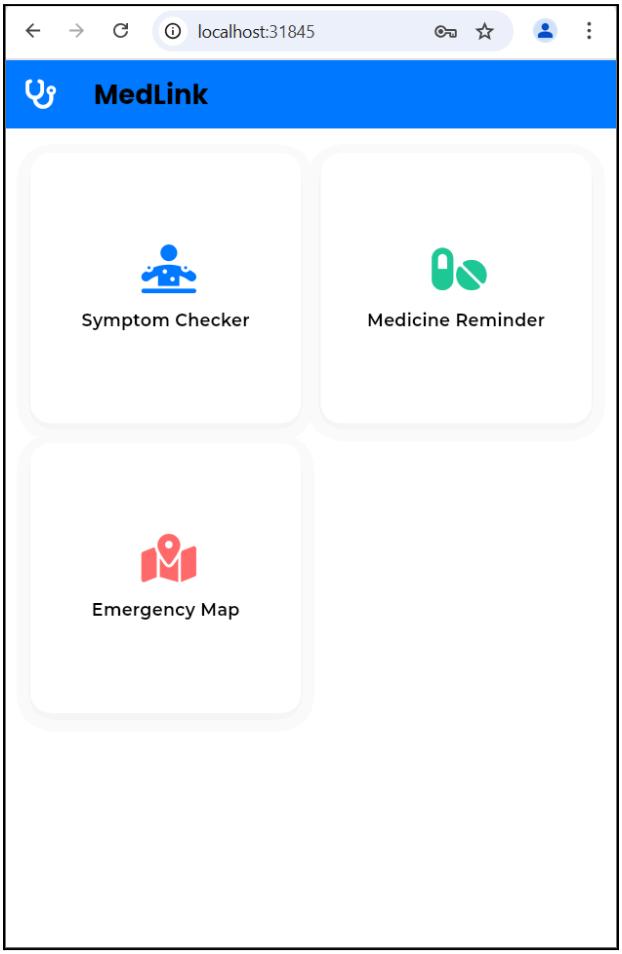
```
                style: TextStyle(
                  fontFamily: 'Montserrat',
                  fontSize: 16,
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),
          ),
        ],
      ),
    ),
  ),
 );
}

void _showResultDialog(BuildContext context) {
 showDialog(
  context: context,
  builder: (BuildContext context) {
   return AlertDialog(
    shape: RoundedRectangleBorder(
     borderRadius: BorderRadius.circular(16),
    ),
    title: Row(
     children: [
      Icon(
       Icons.health_and_safety,
       color: Color(0xFF007BFF),
       size: 28,
      ),
      SizedBox(width: 12),
      Text(
       'Analysis Result',
       style: TextStyle(
        fontFamily: 'Poppins',
        fontWeight: FontWeight.bold,
        fontSize: 20,
        color: Color(0xFF007BFF),
       ),
```

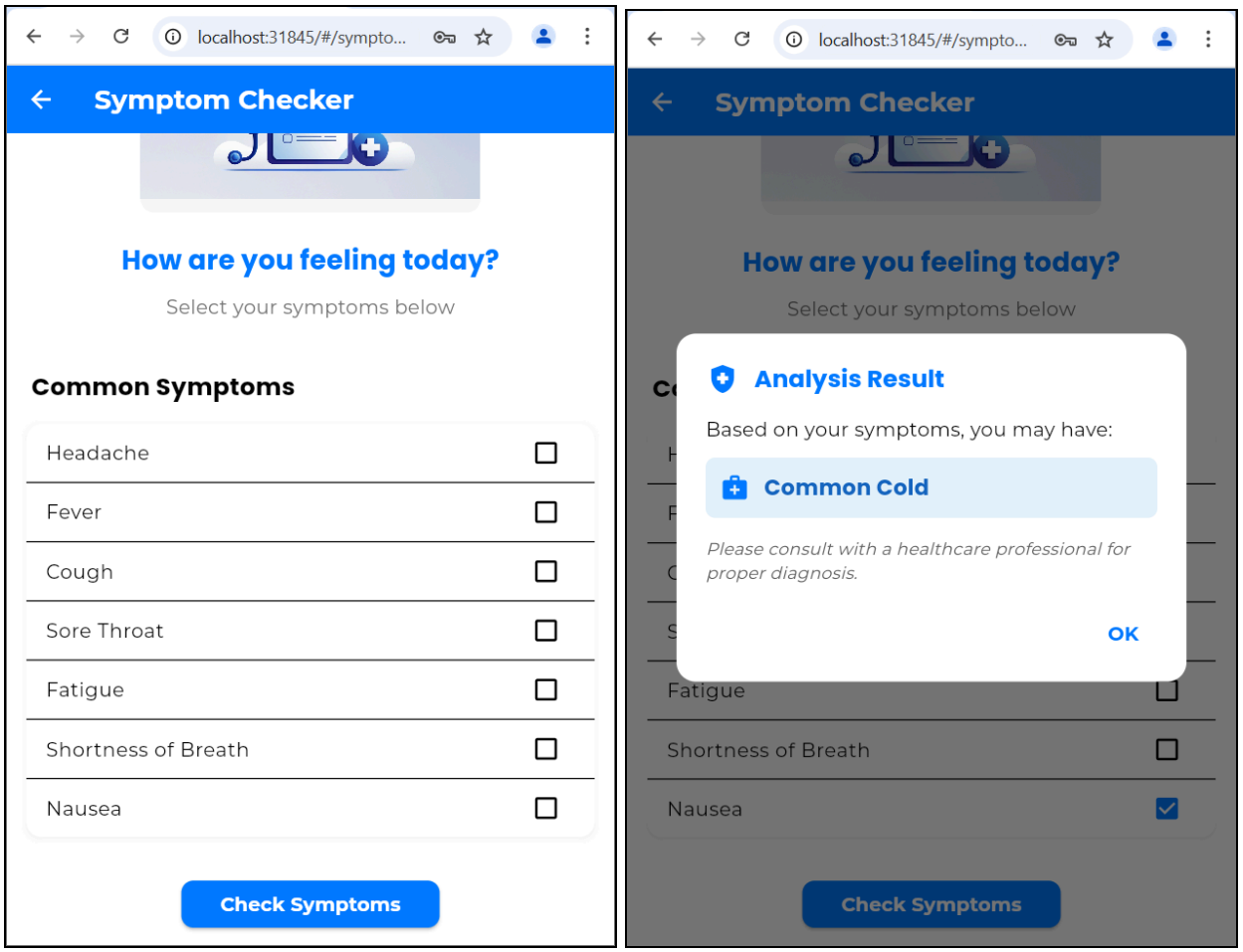
```
          ),
        ],
      ),
      content: Column(
        mainAxisSize: MainAxisSize.min,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            'Based on your symptoms, you may have:',
            style: TextStyle(
              fontFamily: 'Montserrat',
              fontWeight: FontWeight.w500,
              fontSize: 16,
            ),
          ),
          SizedBox(height: 12),
          Container(
            padding: EdgeInsets.all(12),
            decoration: BoxDecoration(
              color: Colors.blue.shade50,
              borderRadius: BorderRadius.circular(8),
            ),
            child: Row(
              children: [
                Icon(Icons.medical_services, color: Color(0xFF007BFF)),
                SizedBox(width: 12),
                Expanded(
                  child: Text(
                    'Common Cold',
                    style: TextStyle(
                      fontFamily: 'Poppins',
                      fontWeight: FontWeight.w600,
                      fontSize: 18,
                      color: Colors.blue.shade800,
                    ),
                  ),
                ),
              ],
            ),
          ),
        ),
```

```
            SizedBox(height: 16),
            Text(
              'Please consult with a healthcare professional for proper diagnosis.',
              style: TextStyle(
                fontFamily: 'Montserrat',
                fontStyle: FontStyle.italic,
                color: Colors.grey.shade700,
              ),
            ),
          ],
        ),
        actions: [
          TextButton(
            style: TextButton.styleFrom(
              foregroundColor: Color(0xFF007BFF),
            ),
            child: Text(
              'OK',
              style: TextStyle(
                fontFamily: 'Montserrat',
                fontWeight: FontWeight.bold,
                fontSize: 16,
              ),
            ),
            onPressed: () {
              Navigator.of(context).pop();
            },
          ),
        ],
      );
    },
  );
}

 void _showHint(BuildContext context) {
   ScaffoldMessenger.of(context).showSnackBar(
     SnackBar(
       content: Text('Double-tap on the image for a health tip!'),
       duration: Duration(seconds: 2),
       backgroundColor: Color(0xFF007BFF),
```

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


localhost:31845
MedLink Login
Email
Harsh@gmail.com
Password
Login
Don't have an account? Sign up


localhost:31845
MedLink
Symptom Checker
Medicine Reminder
Emergency Map



Conclusion

In this experiment, we successfully implemented named routes, swipe gestures for navigation, and tap interactions to enhance user experience in the MedLink app. Initially, we faced issues with gesture detection conflicts and incorrect route navigation, but we resolved them by properly handling GestureDetector callbacks and using structured named routes for smooth transitions.

MPL Practical 06

Aim: To Connect Flutter UI with fireBase database.

Theory:

Firebase is a cloud-based backend service that provides features such as authentication, real-time database, cloud storage, and hosting for mobile applications. In Flutter, Firebase is integrated using the firebase_auth and cloud_firestore packages, enabling seamless communication between the app and Firebase services.

Implementation in Our Project
In this experiment, we integrated Firebase into our Flutter application, MedLink, to enable user authentication. This includes email/password authentication, Google Sign-In, password reset functionality, and email verification.

Key Features Implemented

1. User Registration with Email & Password
    - Users can sign up using their email and password.
    - After registration, an email verification is sent to the user.
    - Only verified users are allowed to log in.

2. User Login
    - Users can log in with their registered email and password.
    - The system checks if the email is verified before granting access.

3. Google Sign-In
    - Users can sign in with their Google accounts using Firebase Authentication.

4. Password Reset
    - If a user forgets their password, they can request a password reset email.

5. Logout Functionality
    - A logout button allows users to sign out from Firebase authentication.

---

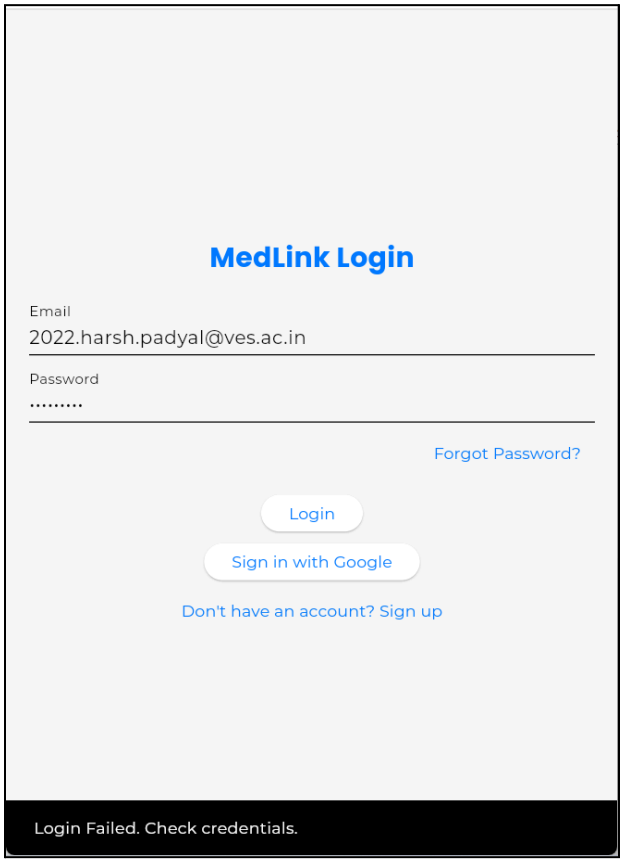
utils/auth_service.dart

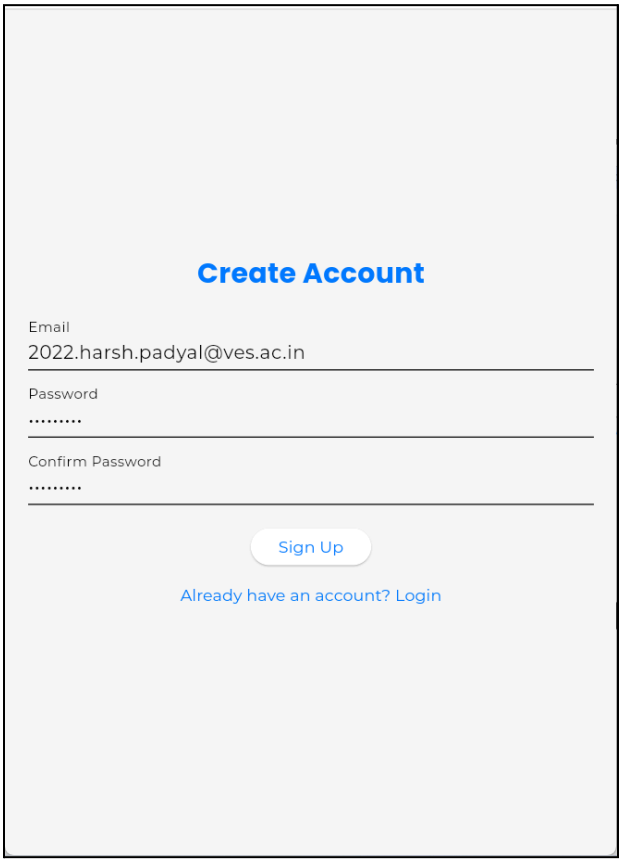
```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:google_sign_in/google_sign_in.dart';

class AuthService {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final GoogleSignIn _googleSignIn = GoogleSignIn(
    clientId: "641097014683-pr9jdi0s76of2nb3suugi6ualiij0lq1.apps.googleusercontent.com",
```

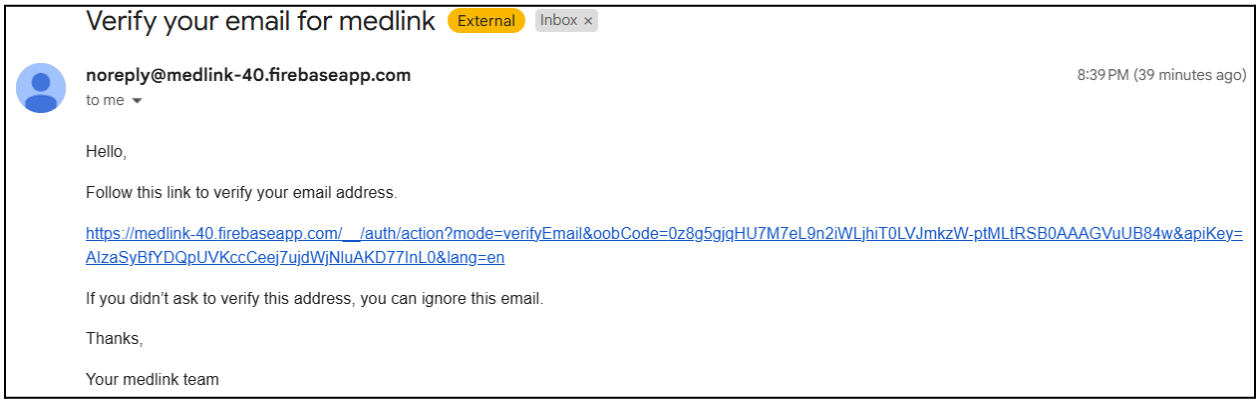
```
);

// Sign Up with Email & Password
Future<User?> signUp(String email, String password) async {
 try {
  print("Attempting to sign up user: $email");
  UserCredential userCredential = await _auth.createUserWithEmailAndPassword(
   email: email,
   password: password,
  );

  // Send email verification
  await userCredential.user?.sendEmailVerification();
  print("Verification email sent to: ${userCredential.user?.email}");

  return userCredential.user;
 } catch (e) {
  print("Sign Up Error: $e");
  return null;
 }
}

// Login with Email & Password
Future<User?> signIn(String email, String password) async {
 try {
  print("Attempting to sign in user: $email");
  UserCredential userCredential = await _auth.signInWithEmailAndPassword(
   email: email,
   password: password,
  );
  if (userCredential.user?.emailVerified == true) {
   print("Login Successful for: ${userCredential.user?.email}");
   return userCredential.user;
  } else {
   print("Login failed: Email not verified");
   await _auth.signOut();
   return null;
  }
 } catch (e) {
  print("Login Error: $e");
```

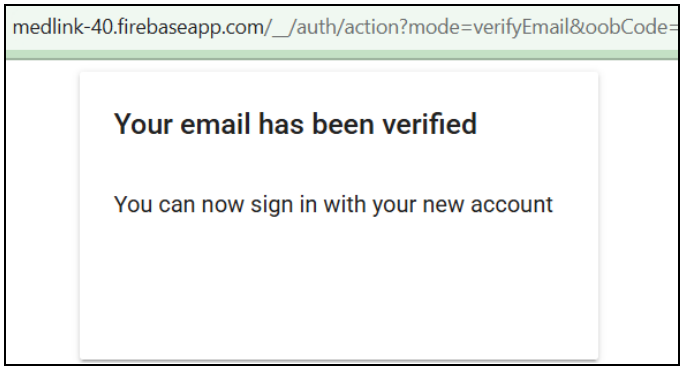
```
    return null;
  }
}

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

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

    UserCredential userCredential = await _auth.signInWithCredential(credential);
    return userCredential.user;
  } catch (e) {
    print("Google Sign-In Error: $e");
    return null;
  }
}

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

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

// Reset Password
Future<bool> resetPassword(String email) async {
  try {
    await _auth.sendPasswordResetEmail(email: email);
    return true; // Email sent successfully
```

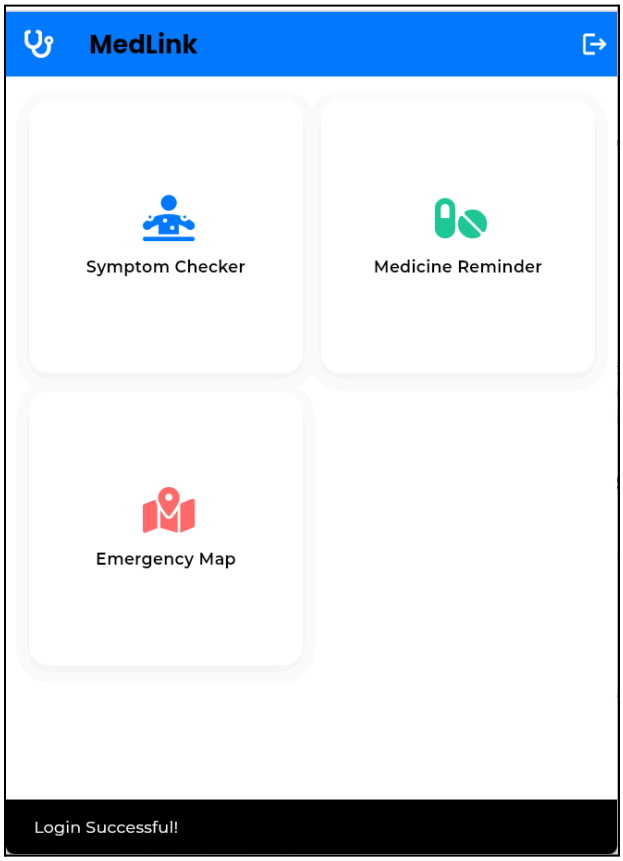
```
    } catch (e) {
      print("Password Reset Error: $e");
      return false; // Failed to send email
    }
  }
}
```

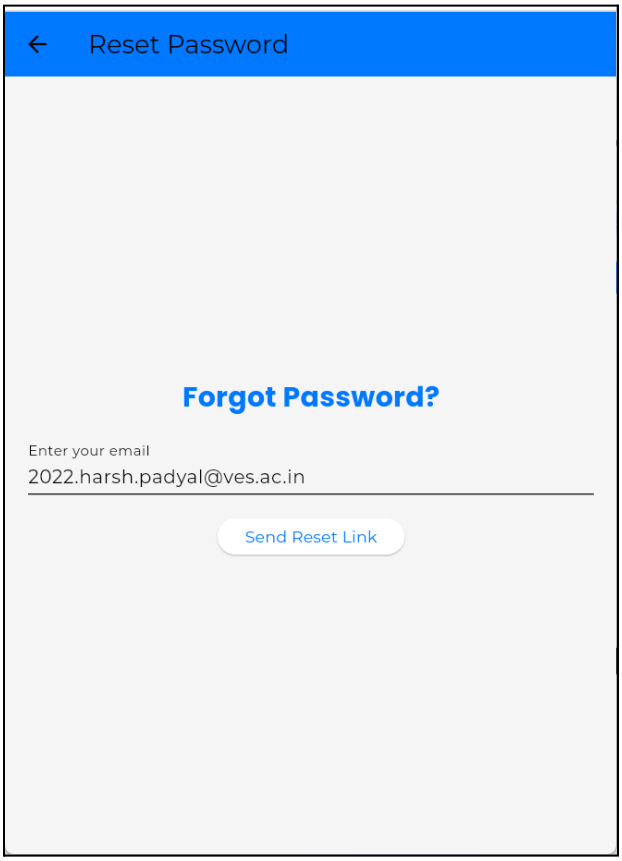


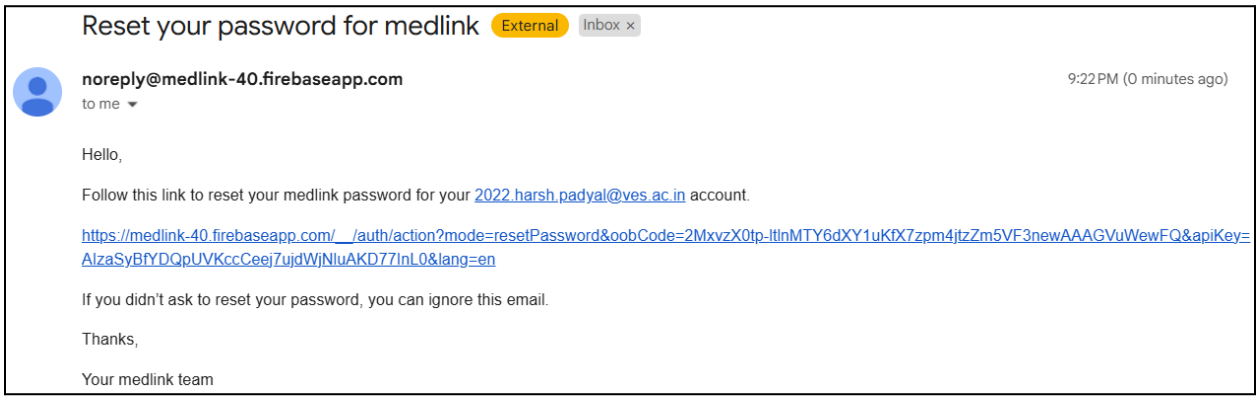


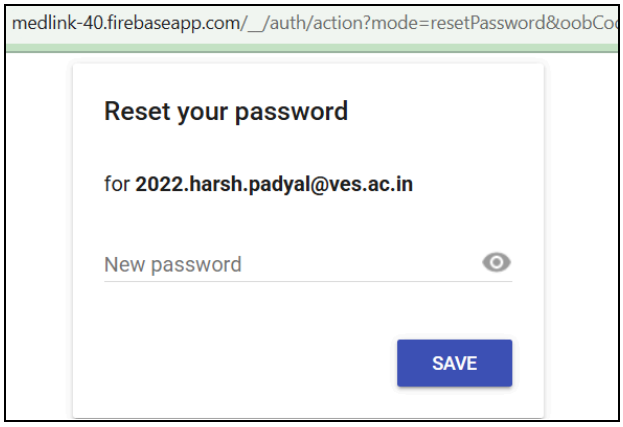


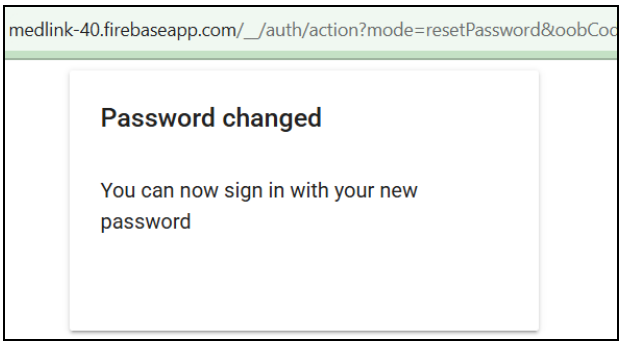

medlink-40.firebaseapp.com/__/auth/action?mode=verifyEmail&oobCode=
Your email has been verified
You can now sign in with your new account

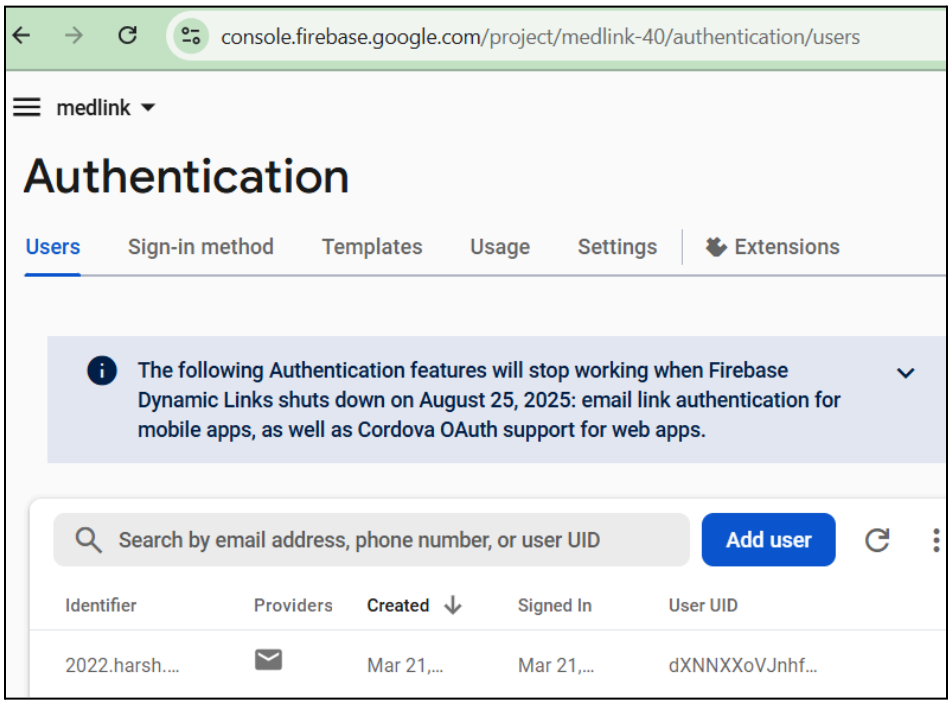

MedLink
Symptom Checker
Medicine Reminder
Emergency Map
Login Successful!

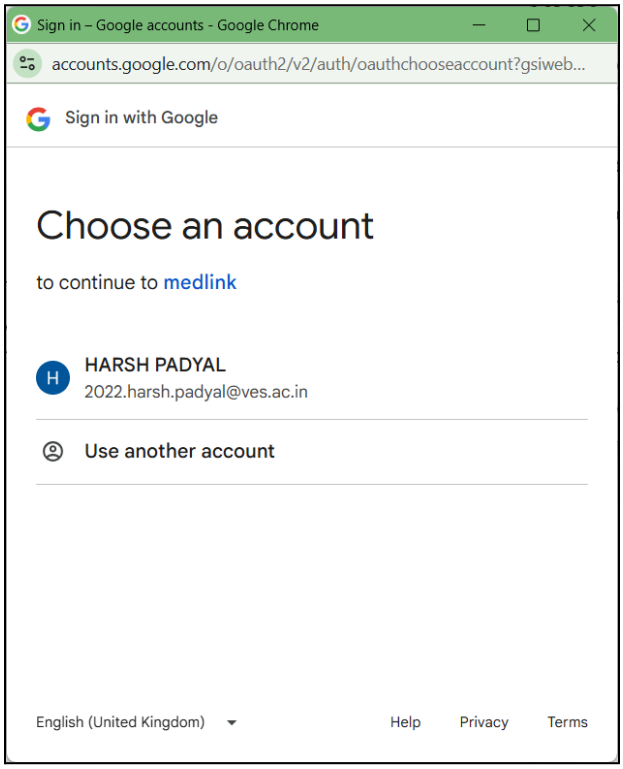

Reset Password
Forgot Password?
Enter your email
2022.harsh.padyal@ves.ac.in
Send Reset Link

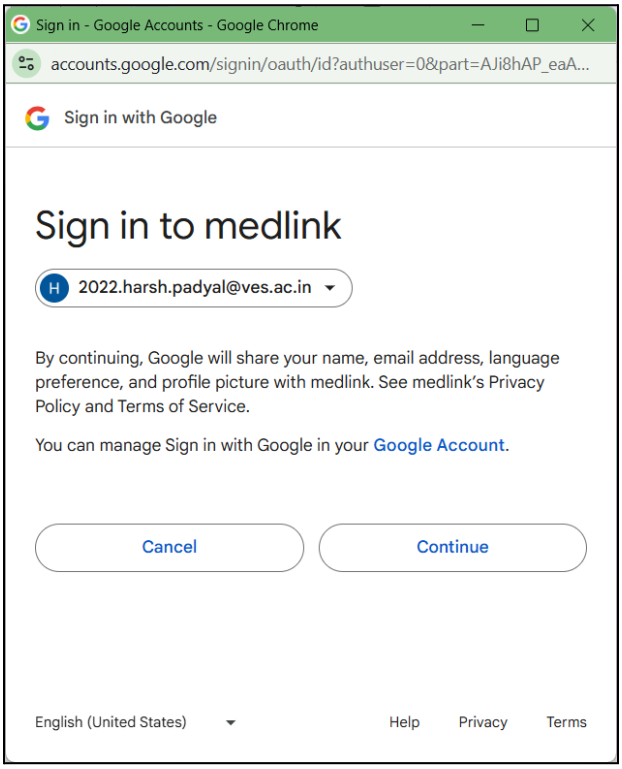

Reset your password for medlink
External
Inbox ×
noreply@medlink-40.firebaseapp.com
to me
9:22 PM (0 minutes ago)
Hello,
Follow this link to reset your medlink password for your 2022.harsh.padyal@ves.ac.in account.
https://medlink-40.firebaseapp.com/__/auth/action?mode=resetPassword&oobCode=2MxvzX0tp-ltlnMTY6dXY1uKfX7zpm4jtzZm5VF3newAAAGVuWewFQ&apiKey=AIzaSyBfYDQpUVKccCeej7ujdWjNluAKD77InL0&lang=en
If you didn't ask to reset your password, you can ignore this email.
Thanks,
Your medlink team

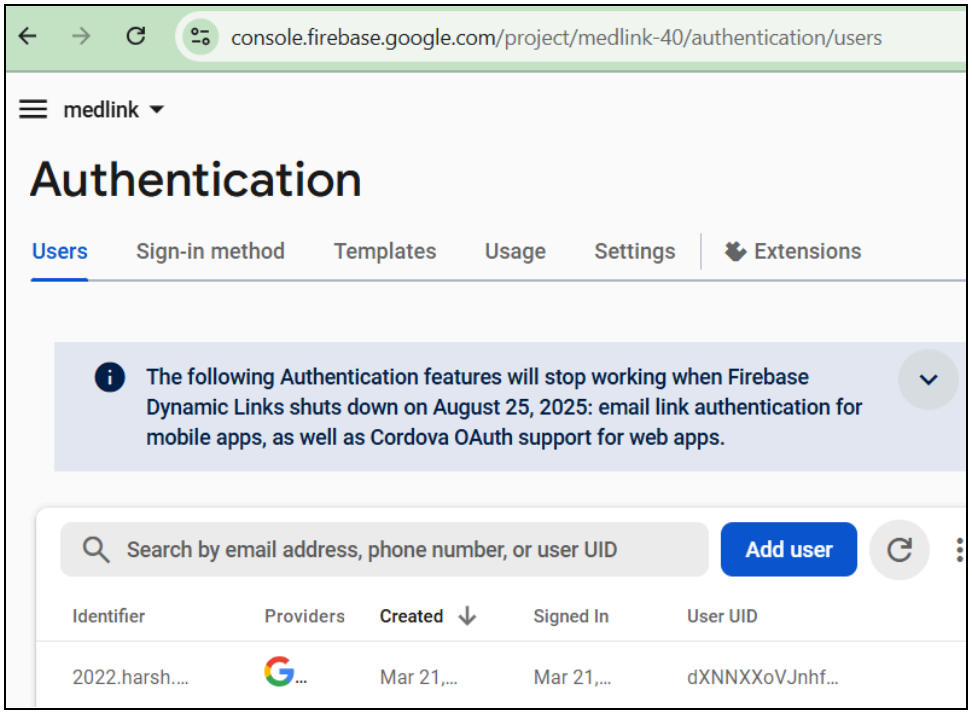

medlink-40.firebaseapp.com/__/auth/action?mode=resetPassword&oobCo
Reset your password
for 2022.harsh.padyal@ves.ac.in
New password
SAVE


medlink-40.firebaseapp.com/__/auth/action?mode=resetPassword&oobCo
Password changed
You can now sign in with your new password


console.firebase.google.com/project/medlink-40/authentication/users
medlink
Authentication
Users
Sign-in method
Templates
Usage
Settings
Extensions
The following Authentication features will stop working when Firebase Dynamic Links shuts down on August 25, 2025: email link authentication for mobile apps, as well as Cordova OAuth support for web apps.
Search by email address, phone number, or user UID
Add user
Identifier
Providers
Created
Signed In
User UID
2022.harsh....
Mar 21,...
Mar 21,...
dXNNXXoVJnhf...


Sign in – Google accounts - Google Chrome
accounts.google.com/o/oauth2/v2/auth/oauthchooseaccount?gsiweb...
Sign in with Google
Choose an account
to continue to medlink
HARSH PADYAL
2022.harsh.padyal@ves.ac.in
Use another account
English (United Kingdom)
Help
Privacy
Terms


Sign in - Google Accounts - Google Chrome
accounts.google.com/signin/oauth/id?authuser=0&part=AJi8hAP_eaA...
Sign in with Google
Sign in to medlink
2022.harsh.padyal@ves.ac.in
By continuing, Google will share your name, email address, language preference, and profile picture with medlink. See medlink's Privacy Policy and Terms of Service.
You can manage Sign in with Google in your Google Account.
Cancel
Continue
English (United States)
Help
Privacy
Terms



Conclusion

We successfully integrated Firebase Authentication with our Flutter app, implementing email/password signup, Google Sign-In, email verification, and password reset features. During development, we encountered issues such as email verification not updating immediately and Google Sign-In configuration errors, which we resolved by manually refreshing user authentication state and correctly setting up Firebase credentials.

MPL Practical 07

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

Theory

A Web App Manifest is a JSON file that contains metadata about a web application, allowing browsers to treat it like a native app. It helps enable the “Add to Home Screen” feature on mobile devices, giving users quicker access to the web app with an icon and splash screen similar to native apps.

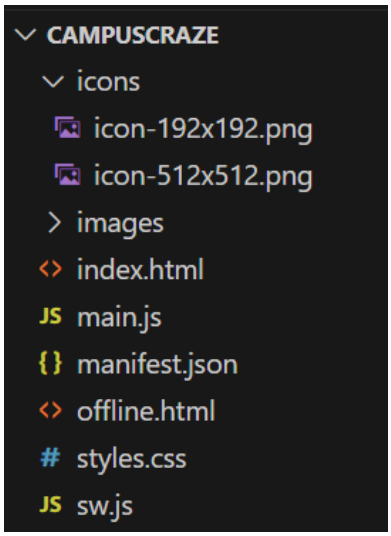
In this experiment, we created a manifest file for our e-commerce PWA CampusCraze, targeted at college students. We included essential metadata like:

- name and short_name to define the app's identity,
- description to describe its purpose,
- start_url to define the launch point,
- display: standalone to make it open without browser UI,
- theme_color and background_color for brand consistency,
- orientation as portrait-primary for mobile optimization,
- icons of different sizes for home screen and splash visuals.

We linked this manifest properly in the <head> of the index.html file, added theme color, and defined an Apple touch icon for broader device compatibility. This ensures that CampusCraze can be installed on user devices and accessed like a native shopping app.

By implementing the manifest, we’ve made CampusCraze PWA-ready with a smoother, app-like experience on mobile devices.

Folder Structure



index.html

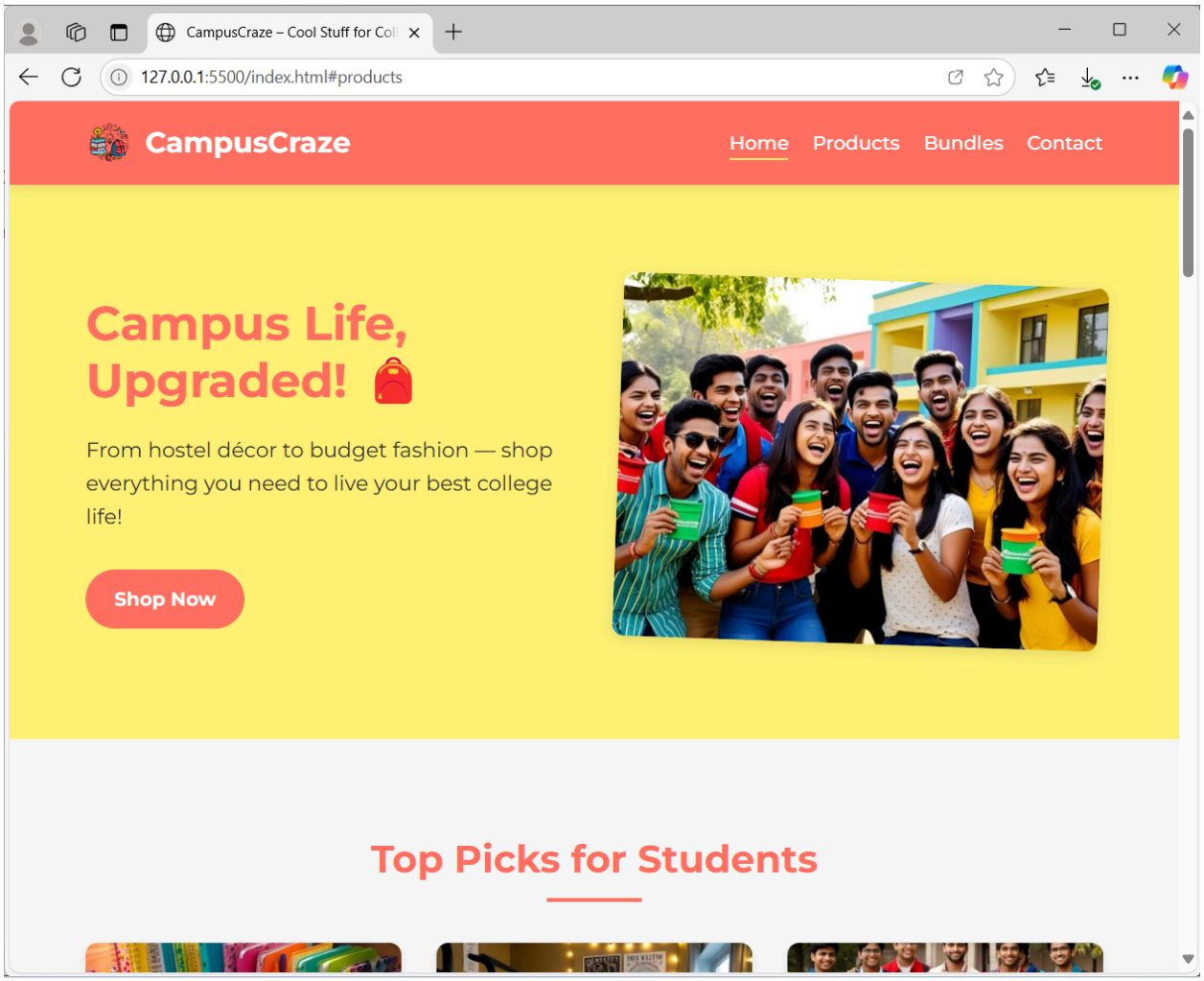
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="CampusCraze - Budget-friendly stationery, hostel decor and fashion for college students">
  <meta name="theme-color" content="#ff6f61">

  <title>CampusCraze – Cool Stuff for College Life</title>

  <link rel="stylesheet" href="styles.css">
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link href="https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;700&display=swap" rel="stylesheet">

  <!-- PWA requirements -->
  <link rel="manifest" href="manifest.json">
  <link rel="apple-touch-icon" href="icons/icon-192x192.png">

  <!-- For better SEO and sharing -->
  <meta property="og:title" content="CampusCraze – Cool Stuff for College Life">
  <meta property="og:description" content="From hostel décor to budget fashion — shop everything you need to live your best college life!">
  <meta property="og:image" content="images/og-image.jpg">
</head>
<body>

</body>
</html>
```

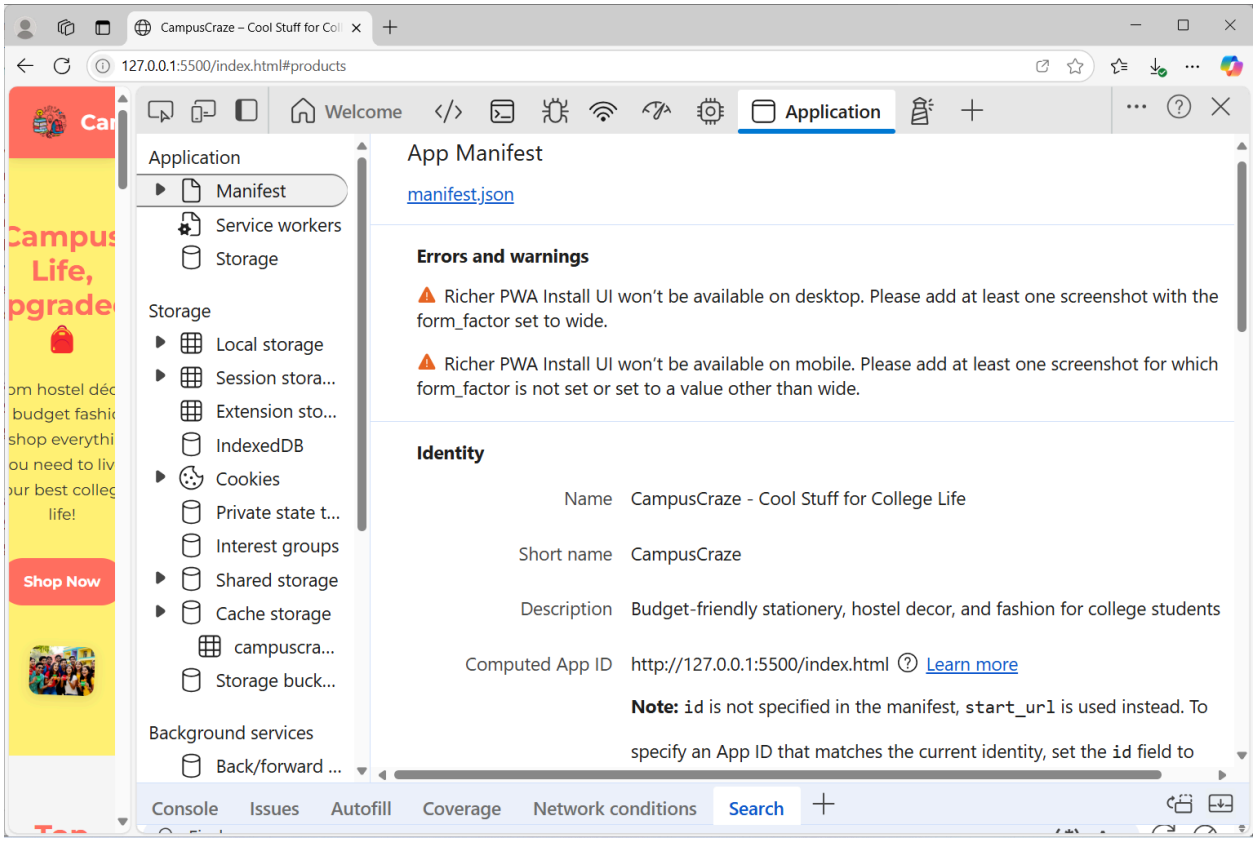
manifest.json

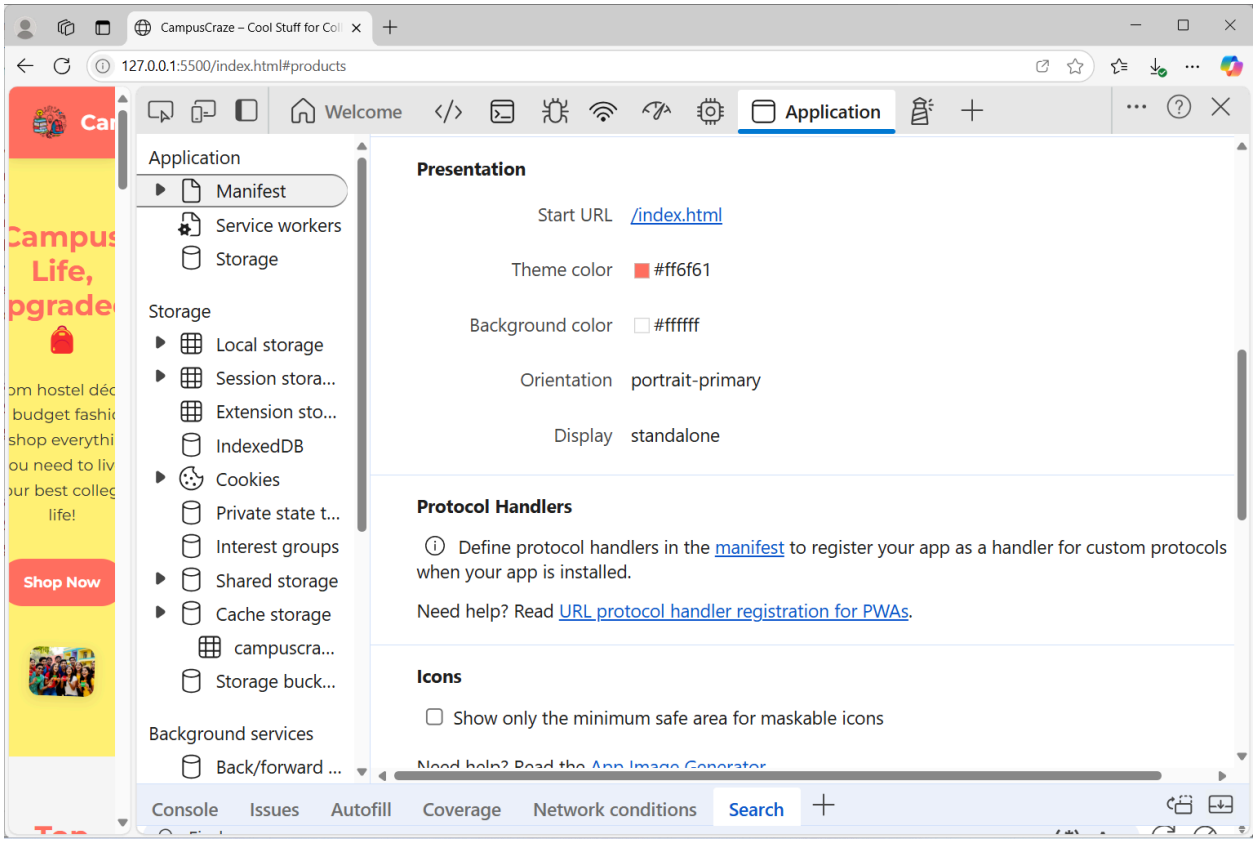
```
{
  "name": "CampusCraze - Cool Stuff for College Life",
  "short_name": "CampusCraze",
  "description": "Budget-friendly stationery, hostel decor, and fashion for college students",
  "start_url": "/index.html",
```

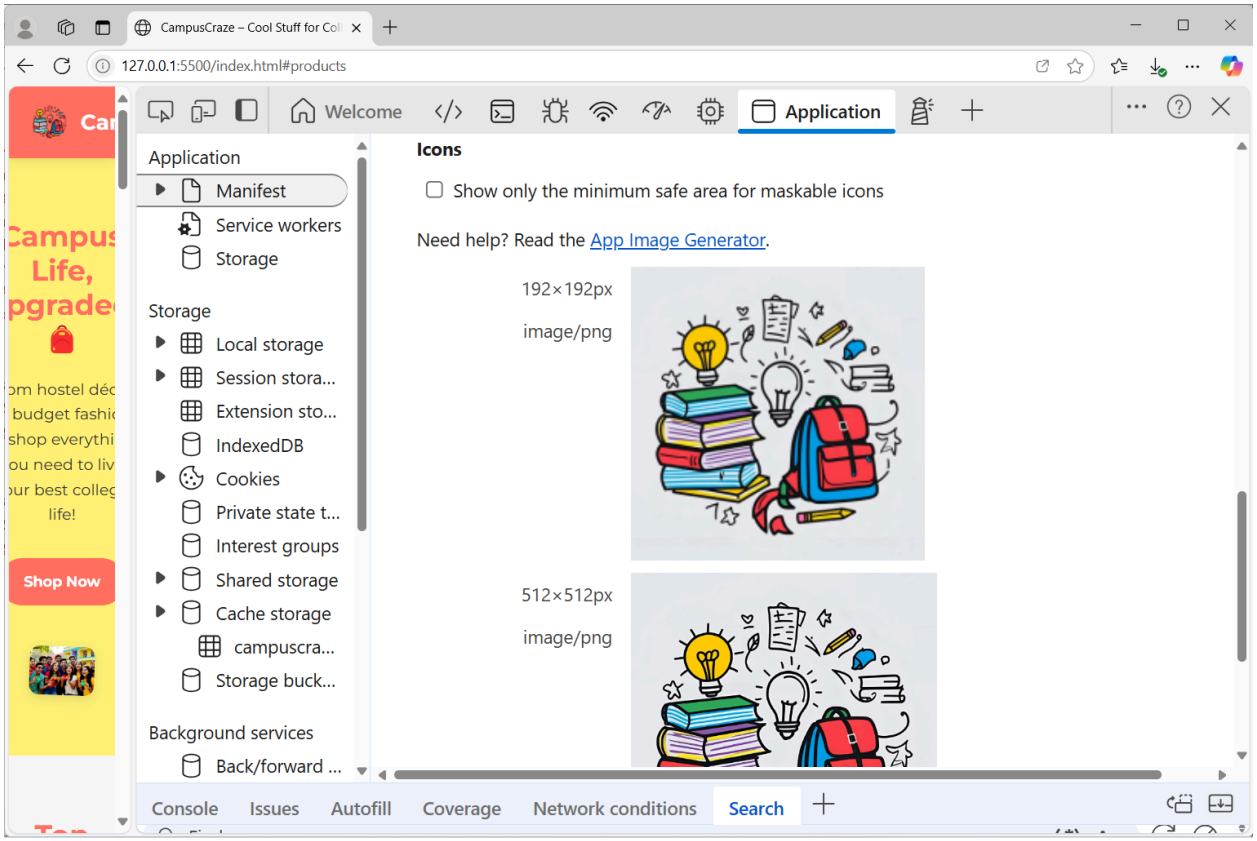
```
 "display": "standalone",
 "background_color": "#ffffff",
 "theme_color": "#ff6f61",
 "orientation": "portrait-primary",
 "icons": [
  {
   "src": "icons/icon-192x192.png",
   "sizes": "192x192",
   "type": "image/png"
  },
  {
   "src": "icons/icon-512x512.png",
   "sizes": "512x512",
   "type": "image/png"
  }
 ]
}
```

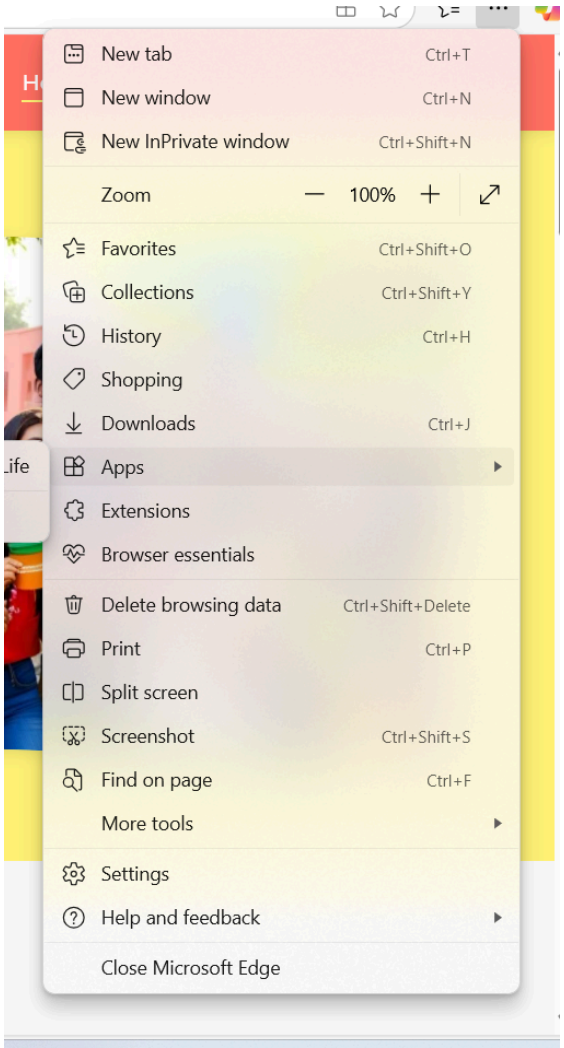
Screenshot

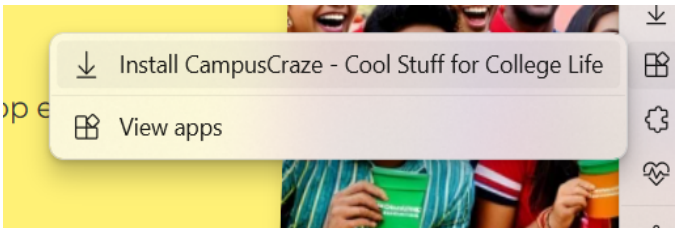


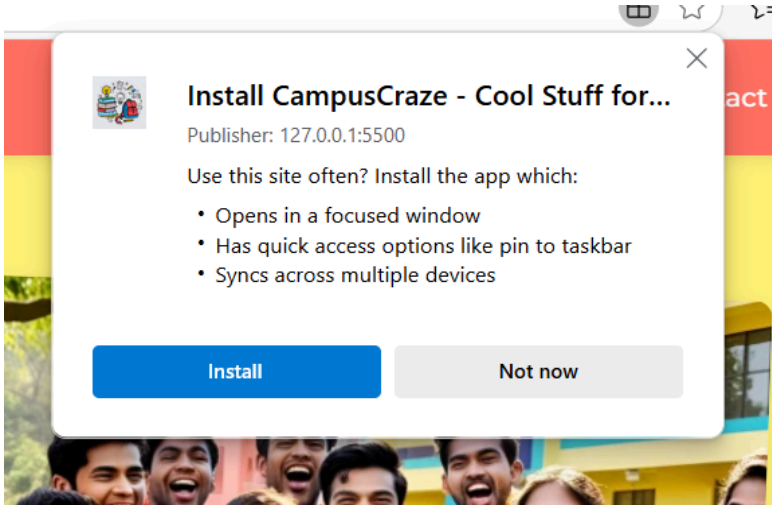

CampusCraze – Cool Stuff for Col
127.0.0.1:5500/index.html#products
Welcome
Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session stora...
Extension sto...
IndexedDB
Cookies
Private state t...
Interest groups
Shared storage
Cache storage
campuscra...
Storage buck...
Background services
Back/forward ...
App Manifest
manifest.json
Errors and warnings
Richer PWA Install UI won't be available on desktop. Please add at least one screenshot with the form_factor set to wide.
Richer PWA Install UI won't be available on mobile. Please add at least one screenshot for which form_factor is not set or set to a value other than wide.
Identity
Name CampusCraze - Cool Stuff for College Life
Short name CampusCraze
Description Budget-friendly stationery, hostel decor, and fashion for college students
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
Console
Issues
Autofill
Coverage
Network conditions
Search
Shop Now

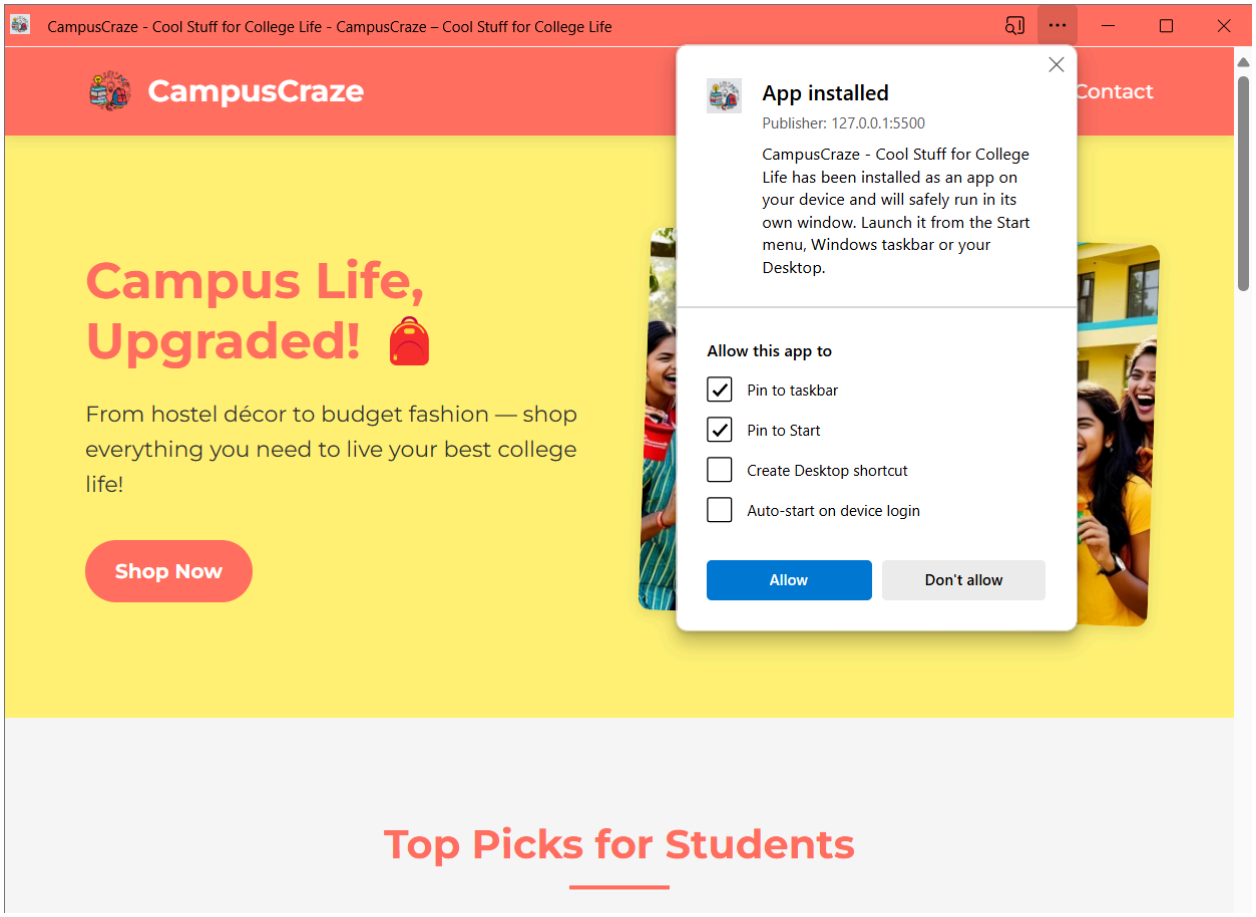

CampusCraze – Cool Stuff for Col
127.0.0.1:5500/index.html#products
Welcome
Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session stora...
Extension sto...
IndexedDB
Cookies
Private state t...
Interest groups
Shared storage
Cache storage
campuscra...
Storage buck...
Background services
Back/forward ...
Presentation
Start URL /index.html
Theme color #ff6f61
Background color #ffffff
Orientation portrait-primary
Display standalone
Protocol Handlers
Define protocol handlers in the manifest to register your app as a handler for custom protocols when your app is installed.
Need help? Read URL protocol handler registration for PWAs.
Icons
Show only the minimum safe area for maskable icons
Console
Issues
Autofill
Coverage
Network conditions
Search
Shop Now


CampusCraze – Cool Stuff for Col
127.0.0.1:5500/index.html#products
Welcome
Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session stora...
Extension sto...
IndexedDB
Cookies
Private state t...
Interest groups
Shared storage
Cache storage
campuscra...
Storage buck...
Background services
Back/forward ...
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
Console
Issues
Autofill
Coverage
Network conditions
Search
Shop Now


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


Install CampusCraze - Cool Stuff for College Life
View apps





Conclusion

In this experiment, we successfully implemented the Web App Manifest for CampusCraze, enabling the “Add to Home Screen” feature with proper metadata and icons. Initially, the manifest didn’t load due to a wrong file path, which we fixed by placing it in the root directory and correctly linking it in the HTML <head>.

MPL Practical 08

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

Theory

A Service Worker is a JavaScript file that runs in the background and helps make a PWA work offline by caching important files. It also allows features like background sync, push notifications, and faster loading even with a slow or no internet connection.

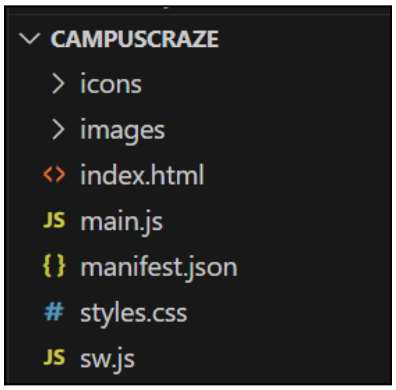
In this experiment, we created and registered a service worker for our e-commerce PWA CampusCraze. The service worker listens to two main events:

- The install event, where it caches essential files like HTML, CSS, images, icons, and fonts to make the app available offline.
- The activate event, where it removes old cached data to keep the app updated and clean.

We also wrote JavaScript in the index.html file to register the service worker when the page loads. Once registered, it caches all listed assets and allows the app to work smoothly even without an internet connection.

This setup helps improve the performance and offline capability of the CampusCraze PWA.

Folder Structure



index.html

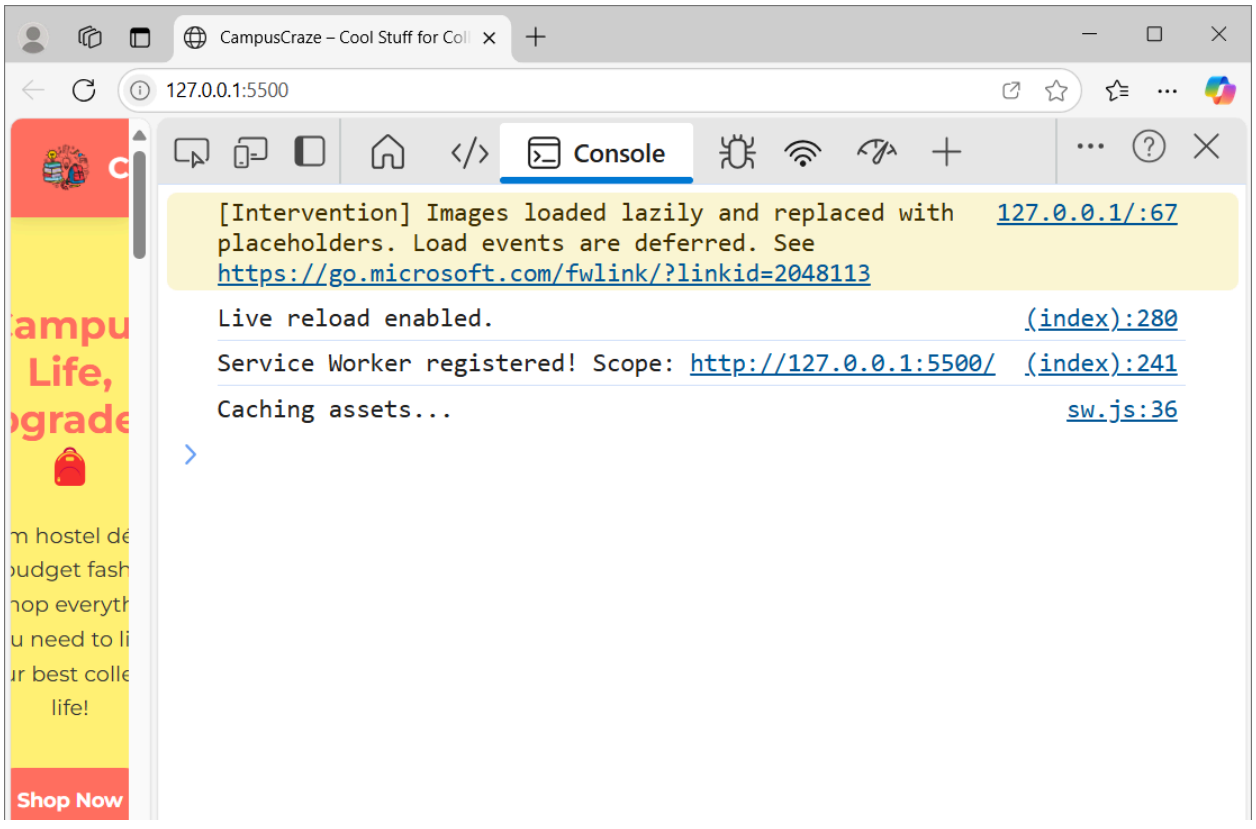
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="CampusCraze - Budget-friendly stationery, hostel decor and fashion for college students">
  <meta name="theme-color" content="#ff6f61">

  <title>CampusCraze – Cool Stuff for College Life</title>
```

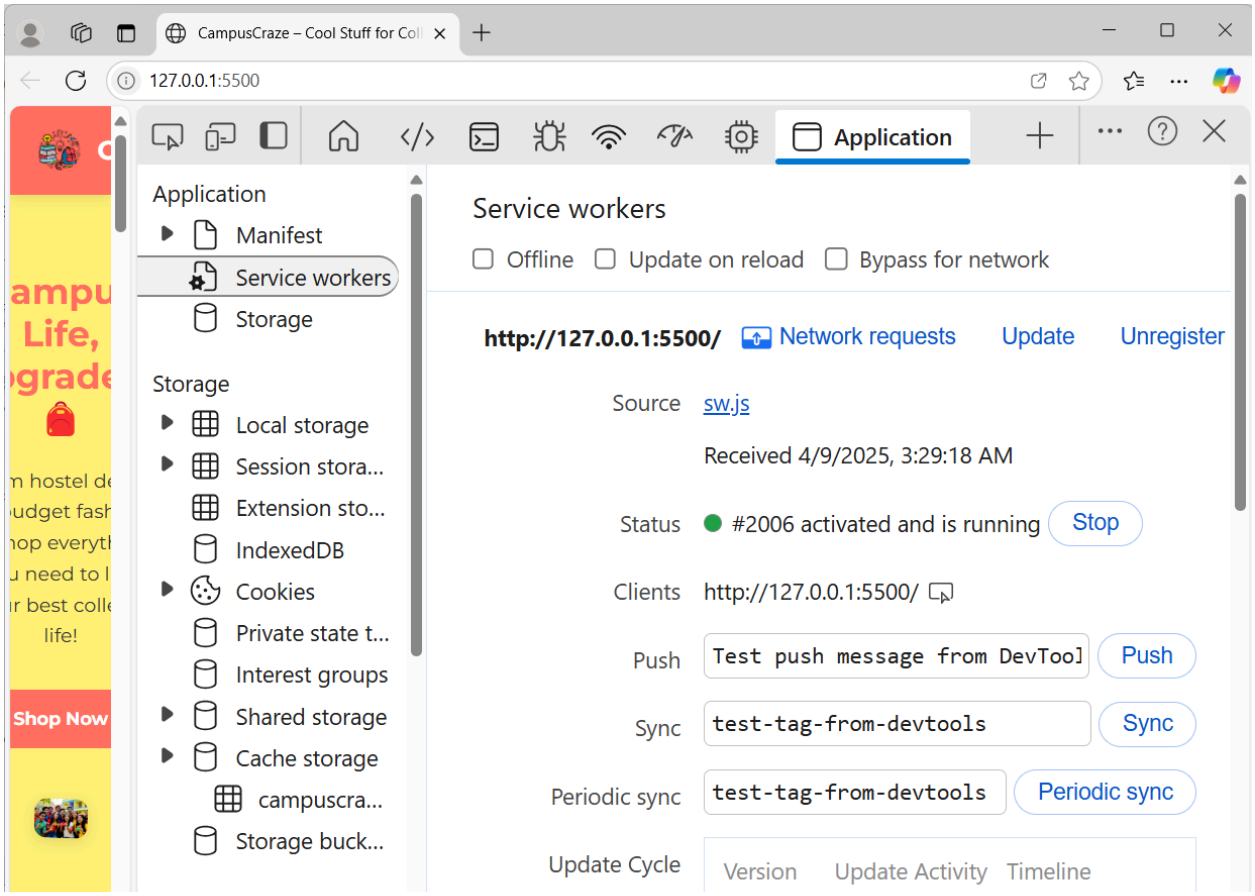
```
  <link rel="stylesheet" href="styles.css">
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link
href="https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;700&display=swap"
rel="stylesheet">

  <!-- PWA requirements -->
  <link rel="manifest" href="manifest.json">
  <link rel="apple-touch-icon" href="icons/icon-192x192.png">

  <!-- For better SEO and sharing -->
  <meta property="og:title" content="CampusCraze – Cool Stuff for College Life">
  <meta property="og:description" content="From hostel décor to budget fashion — shop
everything you need to live your best college life!">
  <meta property="og:image" content="images/og-image.jpg">
</head>
<body>

  <script>
    if ("serviceWorker" in navigator) {
      window.addEventListener("load", () => {
        navigator.serviceWorker
          .register("/sw.js")
          .then((registration) => {
            console.log(
              "Service Worker registered! Scope:",
              registration.scope
            );
          })
          .catch((error) => {
            console.log("Service Worker registration failed:", error);
          });
      });
    }
  </script>
</body>
</html>
```

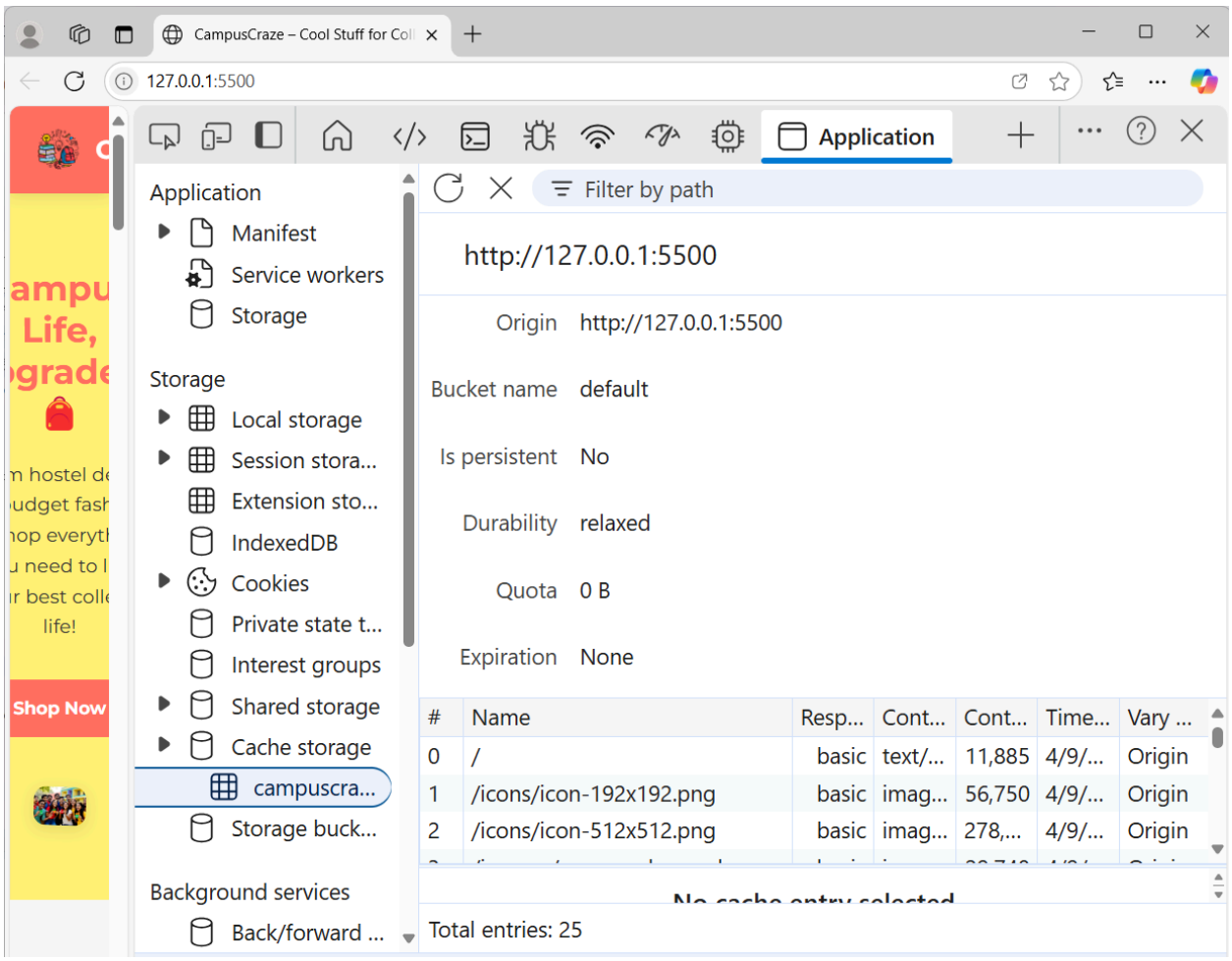
serviceworker.js

```
// CampusCraze Service Worker for PWA functionality
const CACHE_NAME = 'campuscraze-v1';
const ASSETS = [
  '/',
  '/index.html',
  '/styles.css',
  '/main.js',
  '/manifest.json',
  '/images/logo.svg',
  '/images/college-vibes.webp',
  '/images/stationery.webp',
  '/images/decor.webp',
  '/images/fashion.webp',
  '/images/tech.webp',
  '/images/study-bundle.webp',
  '/images/decor-bundle.webp',
  '/images/app-mockup.webp',
  '/images/google-play.svg',
  '/images/app-store.svg',
  '/images/avatar-simran.webp',
  '/images/avatar-rishi.webp',
  '/images/avatar-priya.webp',
  '/images/instagram.svg',
  '/images/youtube.svg',
  '/images/twitter.svg',
  '/icons/icon-192x192.png',
  '/icons/icon-512x512.png',
  'https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;700&display=swap'
];

// Install event - cache assets
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        console.log('Caching assets...');
        return cache.addAll(ASSETS);
      })
      .then(() => self.skipWaiting())
```

```
  );
});

// Activate event - clean up old caches
self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(cacheNames => {
      return Promise.all(
        cacheNames.map(cache => {
          if (cache !== CACHE_NAME) {
            console.log('Clearing old cache:', cache);
            return caches.delete(cache);
          }
        })
      );
    })
    .then(() => self.clients.claim())
  );
});
```




CampusCraze – Cool Stuff for Col
127.0.0.1:5500
Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session stora...
Extension sto...
IndexedDB
Cookies
Private state t...
Interest groups
Shared storage
Cache storage
campuscra...
Storage buck...
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5500/
Network requests
Update
Unregister
Source
sw.js
Received 4/9/2025, 3:29:18 AM
Status
#2006 activated and is running
Stop
Clients
http://127.0.0.1:5500/
Push
Test push message from DevTool
Push
Sync
test-tag-from-devtools
Sync
Periodic sync
test-tag-from-devtools
Periodic sync
Update Cycle
Version
Update Activity
Timeline


CampusCraze – Cool Stuff for Col
127.0.0.1:5500
Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session stora...
Extension sto...
IndexedDB
Cookies
Private state t...
Interest groups
Shared storage
Cache storage
campuscra...
Storage buck...
Background services
Back/forward ...
Filter by path
http://127.0.0.1:5500
Origin
http://127.0.0.1:5500
Bucket name
default
Is persistent
No
Durability
relaxed
Quota
0 B
Expiration
None
Name Resp... Cont... Cont... Time... Vary ...
0 / basic text/... 11,885 4/9/... Origin
1 /icons/icon-192x192.png basic imag... 56,750 4/9/... Origin
2 /icons/icon-512x512.png basic imag... 278,... 4/9/... Origin
Total entries: 25

Conclusion

In this experiment, we successfully registered a service worker and implemented the install and activate events to cache assets for offline use in CampusCraze. Initially, the service worker failed to load because the file was named incorrectly (sw.js vs serviceworker.js), which we fixed by renaming the file to match the registered path.

MPL Practical 09

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

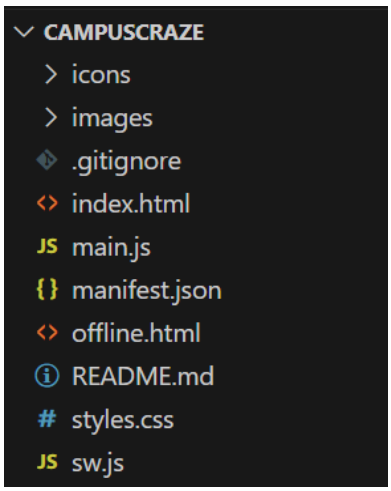
Theory

A **service worker** can handle more than just caching — it can also manage **fetch**, **sync**, and **push** events to improve a web app's performance and user experience. These events allow the app to work offline, sync data when internet is back, and send notifications to users.
In this experiment, we enhanced our CampusCraze PWA by implementing:

- A **fetch event** to serve cached files when offline, and show a custom offline.html page if the network fails.
- A **sync event** to register background sync using SyncManager, which lets the app send data to the server when the internet connection is restored.
- A **push event** that listens for push messages and shows custom notifications even when the app is not open.

These features make CampusCraze more reliable, responsive, and engaging for users, especially in low or no network conditions.

Folder Structure



---

index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>EcoGadgetHub – Sustainable Tech Gadgets</title>
  <link rel="manifest" href="manifest.json">
```

```
   </style>
  </head>
  <body>
   <script>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="CampusCraze - Budget-friendly stationery, hostel decor
and fashion for college students">
  <meta name="theme-color" content="#ff6f61">

  <title>CampusCraze – Cool Stuff for College Life</title>

  <link rel="stylesheet" href="styles.css">
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link
href="https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;700&display=swap"
rel="stylesheet">

  <!-- PWA requirements -->
  <link rel="manifest" href="manifest.json">
  <link rel="apple-touch-icon" href="icons/icon-192x192.png">

  <!-- For better SEO and sharing -->
  <meta property="og:title" content="CampusCraze – Cool Stuff for College Life">
  <meta property="og:description" content="From hostel décor to budget fashion — shop
everything you need to live your best college life!">
  <meta property="og:image" content="images/og-image.jpg">
</head>
<body>

  <script>
     if ("serviceWorker" in navigator) {
       window.addEventListener("load", () => {
         navigator.serviceWorker
           .register("/sw.js")
           .then((registration) => {
```

```
          console.log(
            "Service Worker registered! Scope:",
            registration.scope
          );


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


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

offline.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <title>CampusCraze - Offline</title>
 <style>
 </style>
</head>
<body>
   <div class="message">
    <p>Looks like your internet connection is pulling an all-nighter! 💤</p>
    <p>Check your Wi-Fi or data connection and try again.</p>
   </div>
</body>
</html>
```

serviceworker.js

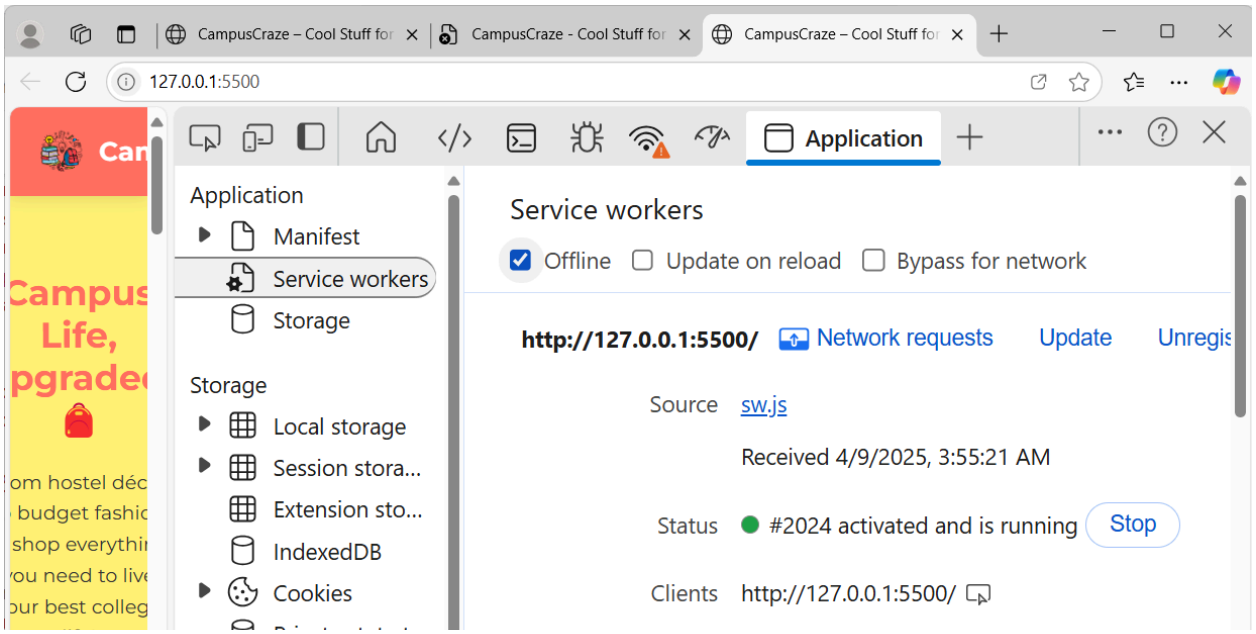
```
// CampusCraze Service Worker for PWA functionality
const CACHE_NAME = 'campuscraze-v1';
const ASSETS = [
 '/',
 '/index.html',
 '/styles.css',
 '/main.js',
 '/manifest.json',
 '/images/logo.svg',
 '/images/college-vibes.webp',
 '/images/stationery.webp',
 '/images/decor.webp',
 '/images/fashion.webp',
 '/images/tech.webp',
 '/images/study-bundle.webp',
 '/images/decor-bundle.webp',
 '/images/app-mockup.webp',
 '/images/google-play.svg',
 '/images/app-store.svg',
 '/images/avatar-simran.webp',
```

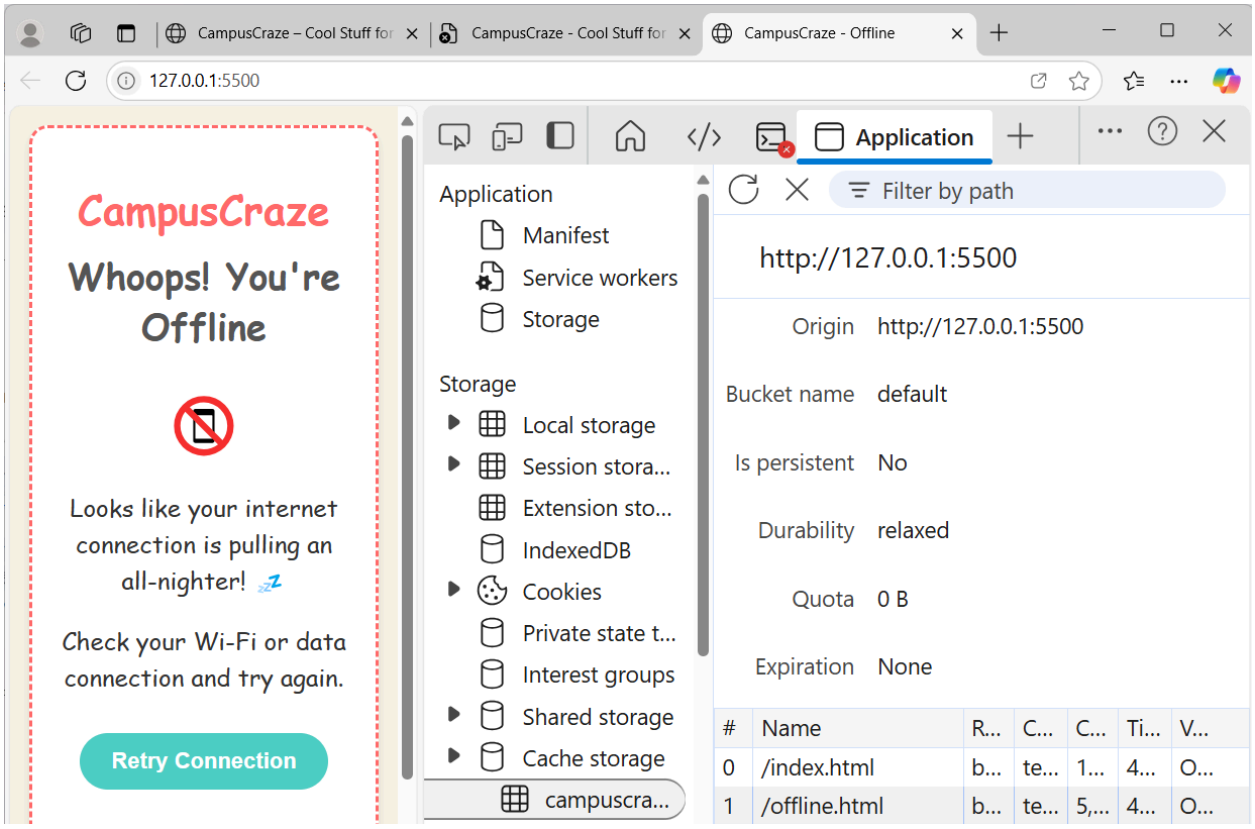
```
  '/images/avatar-rishi.webp',
  '/images/avatar-priya.webp',
  '/images/instagram.svg',
  '/images/youtube.svg',
  '/images/twitter.svg',
  '/icons/icon-192x192.png',
  '/icons/icon-512x512.png',
  'offline.html',
  'https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;700&display=swap'
];

// 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(ASSETS);
    })
  );
});

// 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
      })()
    );
  }
```

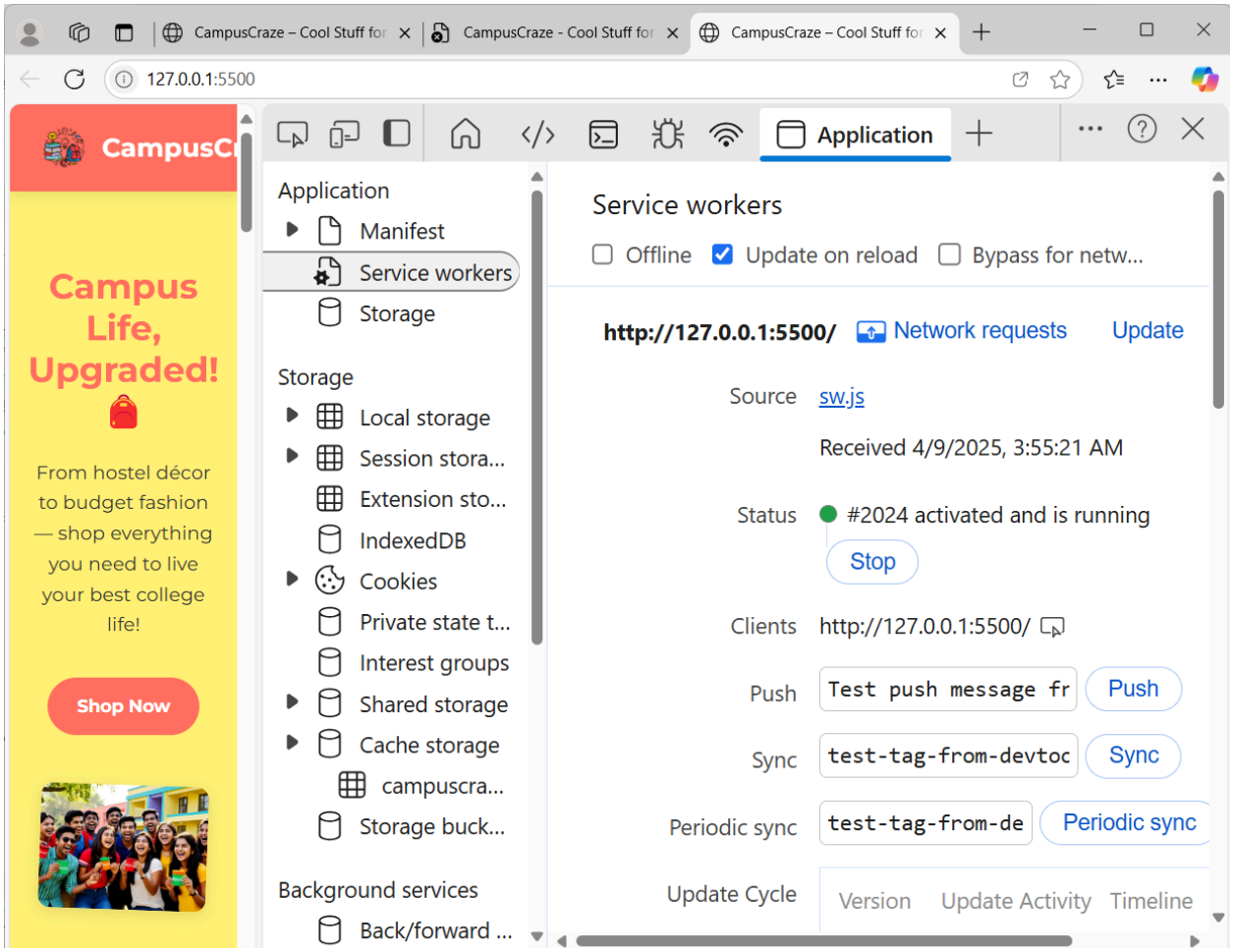
```
});

// 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("CampusCraze - Cool Stuff for College Life", {
          body: data.message,
        })
      );
    }
  }
});
```

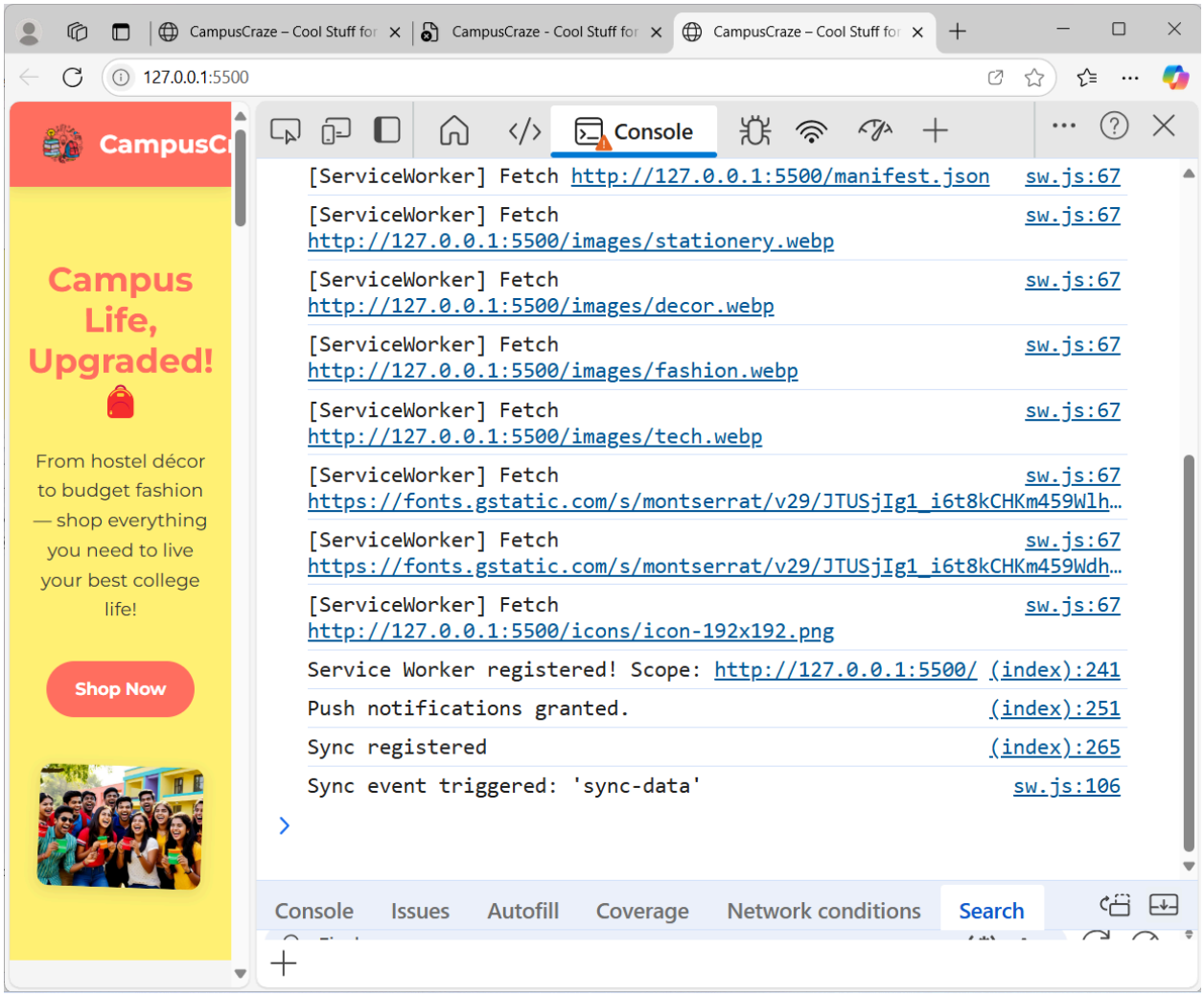
1. Test: Fetch Event (Offline Support)



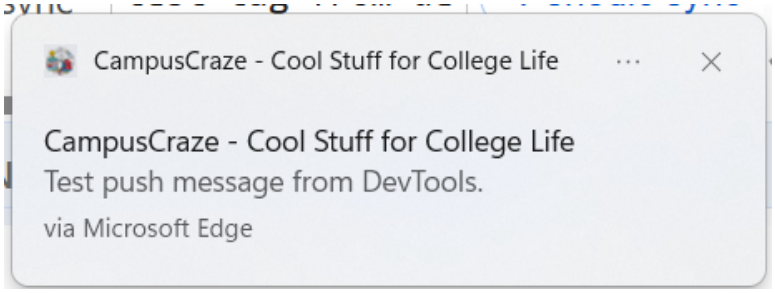


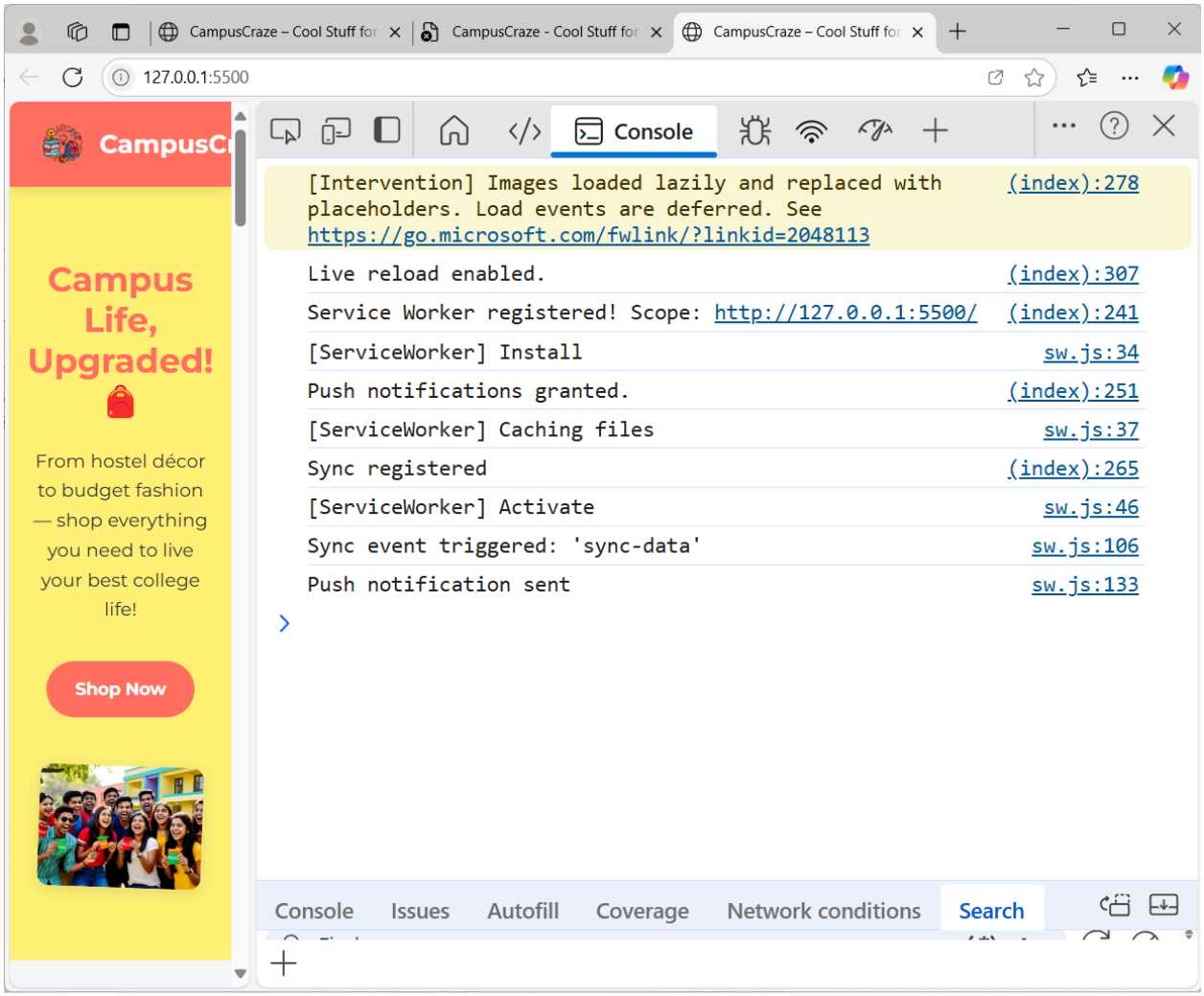
2. Test: Background Sync Event





3. Test: Push Notification Event





Conclusion

In this experiment, we implemented fetch, sync, and push events in the CampusCraze service worker to enable offline access, background syncing, and push notifications. Initially, the sync event wasn't triggering because we forgot to check for navigator.serviceWorker.ready, which we fixed by properly chaining the sync registration inside it.

MPL Practical 10

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

Github Link: https://harshpadyal.github.io/PWA-CAMPUS-CRAZE/

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.

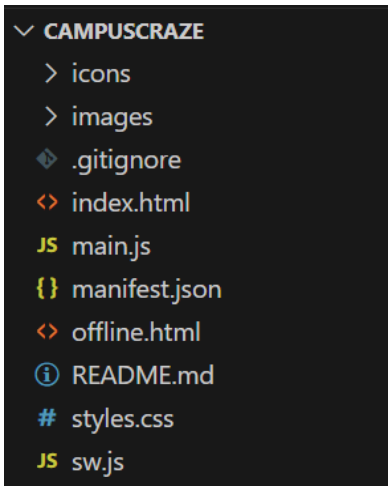
What we implemented in our code

In this experiment, we deployed our E-commerce PWA project, "CampusCraze - Budget-friendly stationery, hostel decor, and fashion for college students
", to GitHub Pages using these steps:

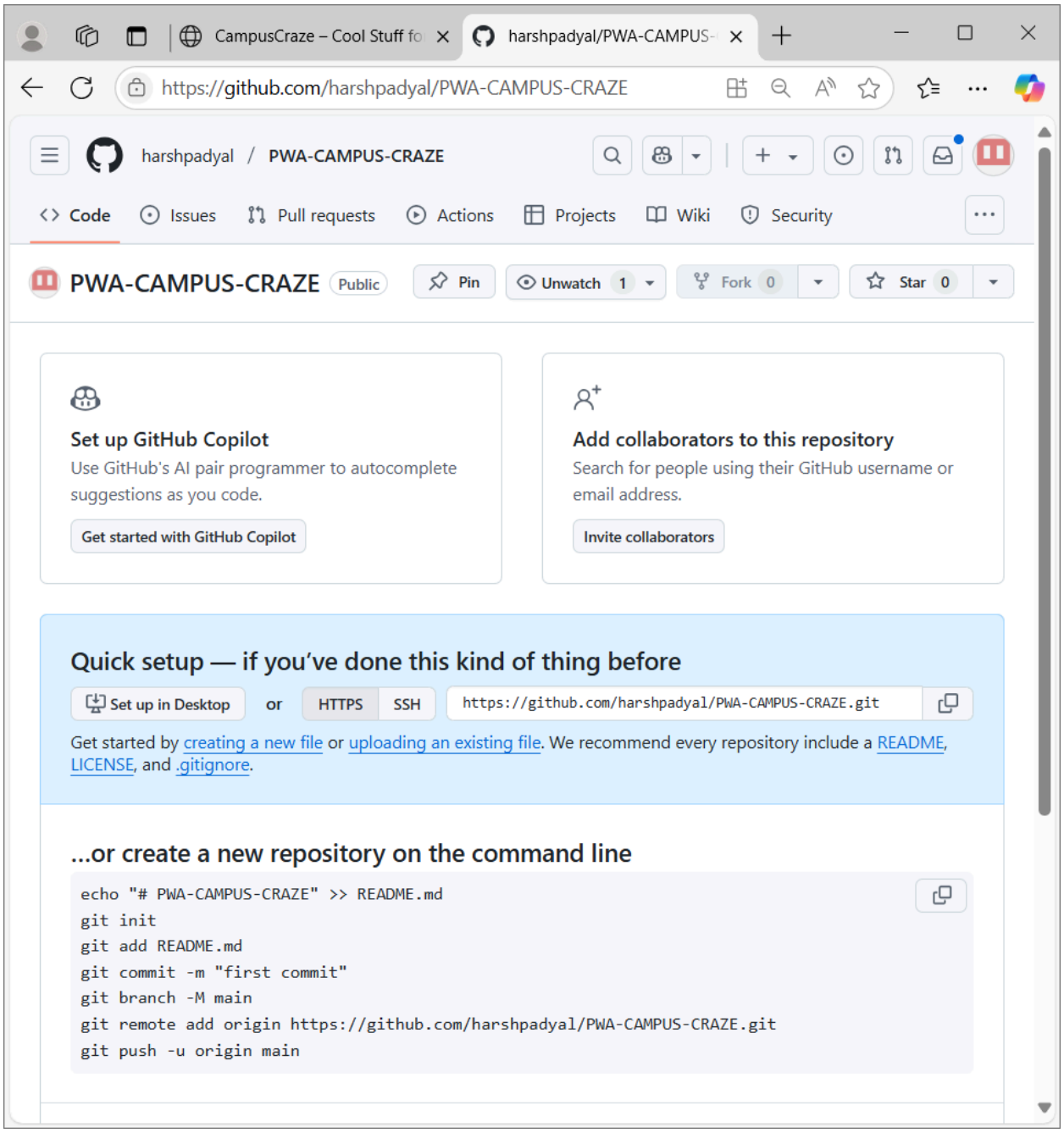
- We ensured all necessary PWA files were in place, such as index.html, manifest.json, sw.js, and offline assets.
- The index.html file used relative paths correctly so resources load properly after deployment.
- We registered the service worker to enable offline access and caching of essential files.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

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

Folder Structure



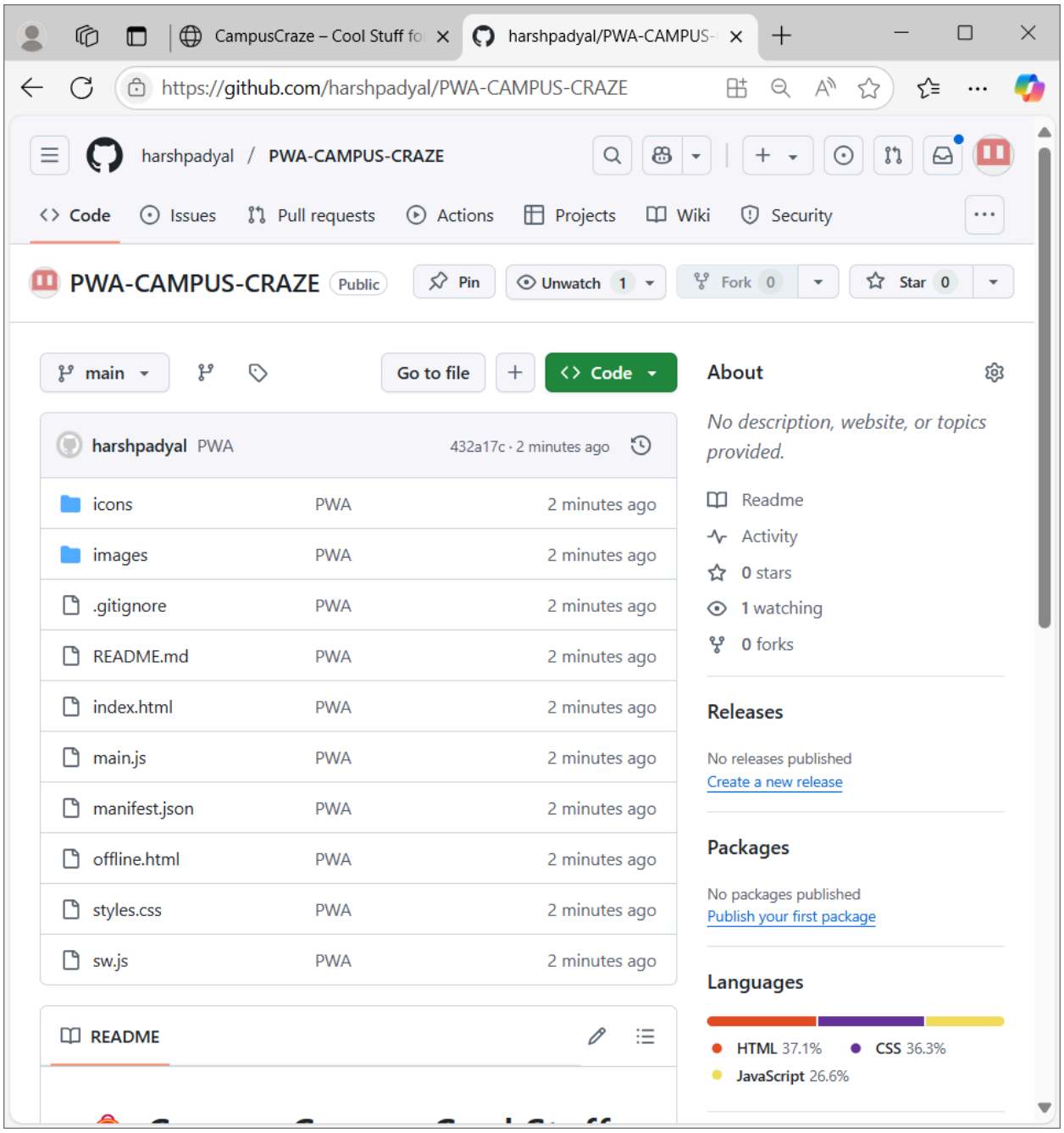
Create a GitHub Repository

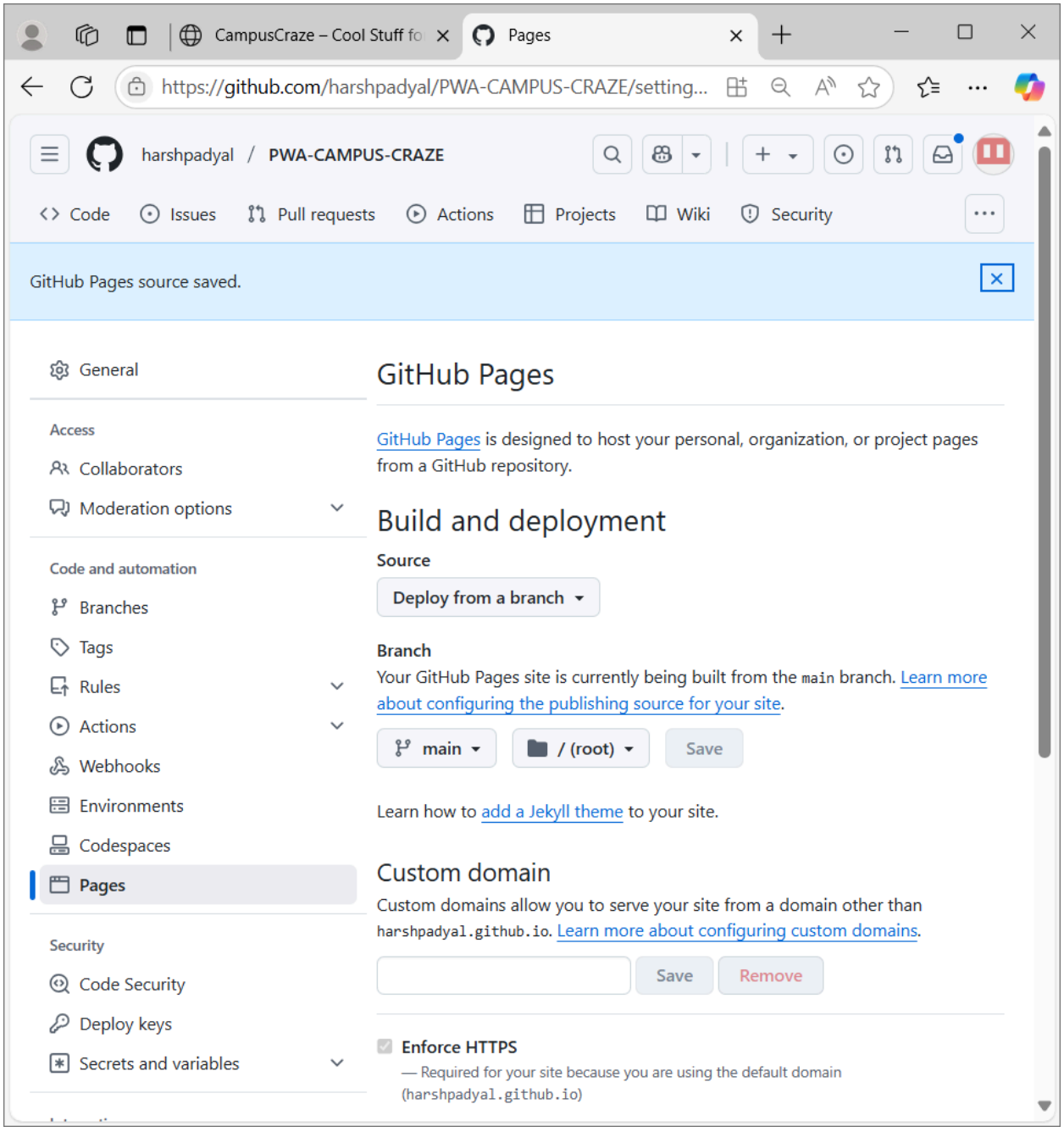


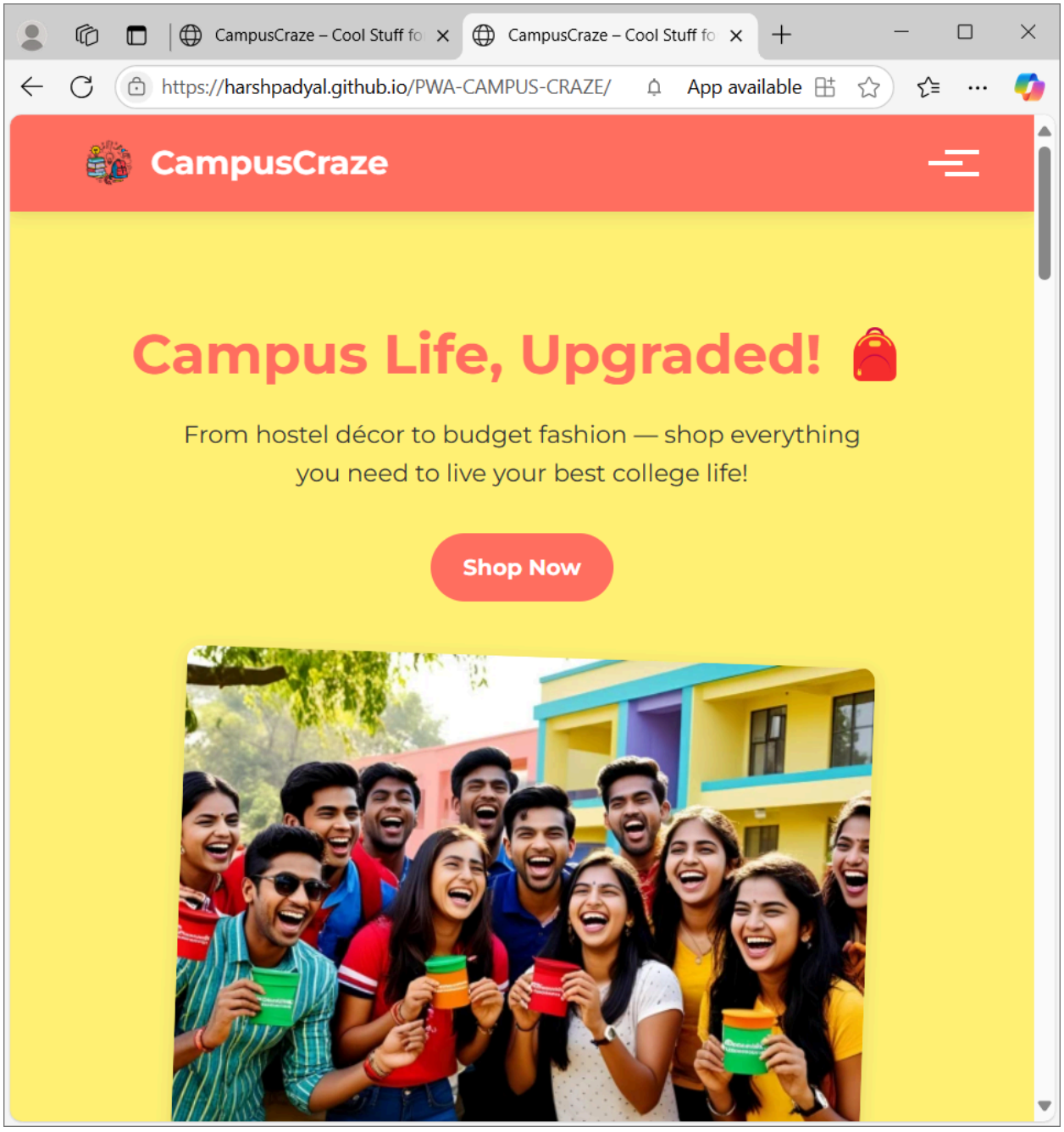
Link Your Local Project to GitHub and Push Changes.

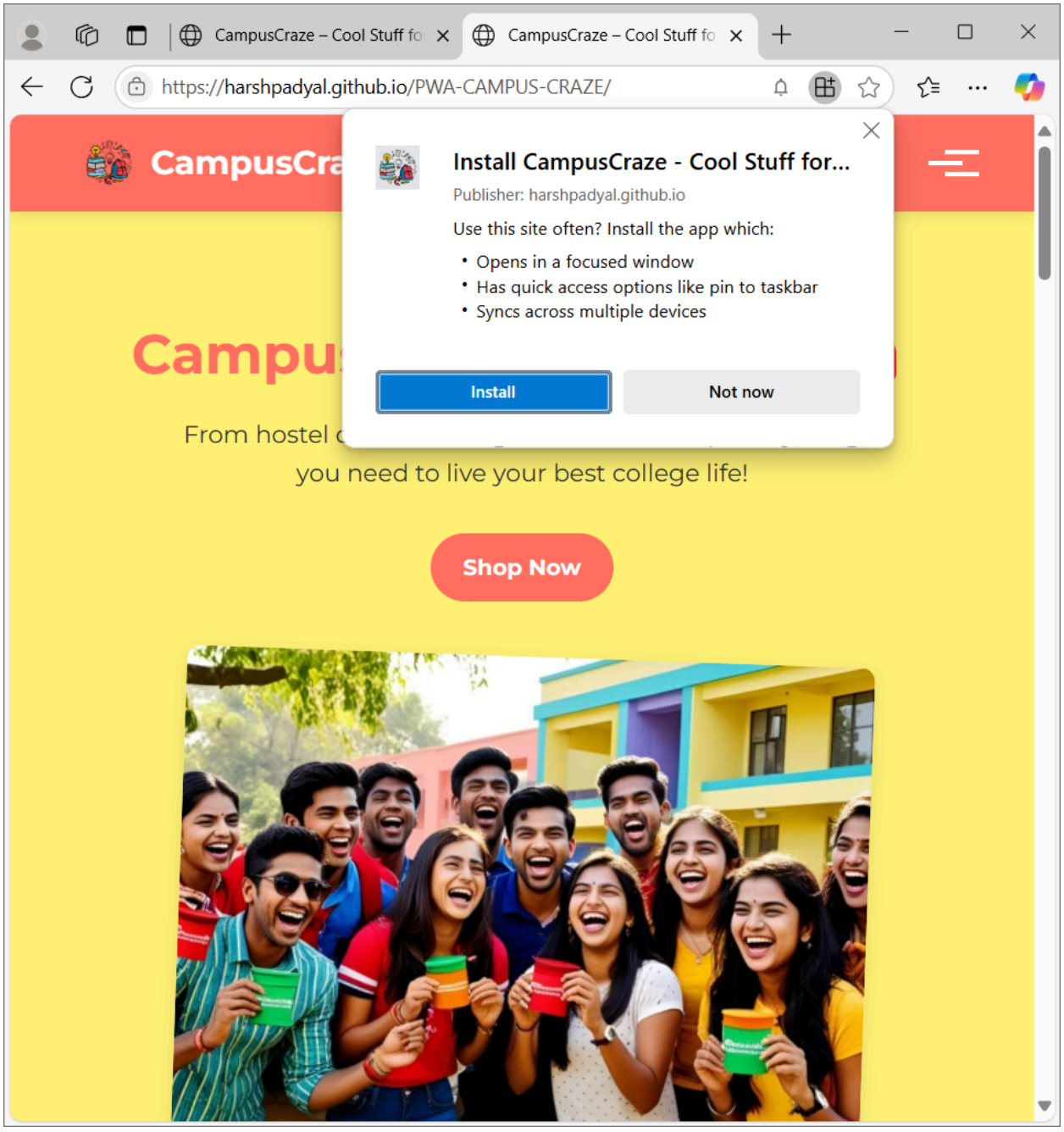
```
PS D:\Users\harsh\Documents\CampusCraze> git init
Initialized empty Git repository in D:/Users/harsh/Documents/CampusCraze/.git/
PS D:\Users\harsh\Documents\CampusCraze> git add .
PS D:\Users\harsh\Documents\CampusCraze> git commit -m "PWA"
[main (root-commit) 432a17c] PWA
 27 files changed, 1774 insertions(+)
```

```
PS D:\Users\harsh\Documents\CampusCraze> git remote add origin https://github.com/harshpadyal/
PS D:\Users\harsh\Documents\CampusCraze> git push -u origin main
info: please complete authentication in your browser...
Enumerating objects: 31, done.
Counting objects: 100% (31/31), done.
Delta compression using up to 4 threads
Compressing objects: 100% (31/31), done.
Writing objects: 100% (31/31), 879.62 KiB | 10.35 MiB/s, done.
Total 31 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/harshpadyal/PWA-CAMPUS-CRAZE.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS D:\Users\harsh\Documents\CampusCraze>
```




CampusCraze – Cool Stuff fo
Pages
https://github.com/harshpadyal/PWA-CAMPUS-CRAZE/setting...
harshpadyal / PWA-CAMPUS-CRAZE
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
GitHub Pages source saved.
General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Security
Code Security
Deploy keys
Secrets and variables
GitHub Pages
GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.
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.
Custom domain
Custom domains allow you to serve your site from a domain other than harshpadyal.github.io. Learn more about configuring custom domains.
Save
Remove
Enforce HTTPS
— Required for your site because you are using the default domain (harshpadyal.github.io)


CampusCraze – Cool Stuff fo
CampusCraze – Cool Stuff fo
https://harshpadyal.github.io/PWA-CAMPUS-CRAZE/
App available
CampusCraze
Campus Life, Upgraded!
From hostel décor to budget fashion — shop everything you need to live your best college life!
Shop Now



Conclusion

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

https://harshpadyal.github.io/PWA-CAMPUS-CRAZE/#products

99 95 96 100

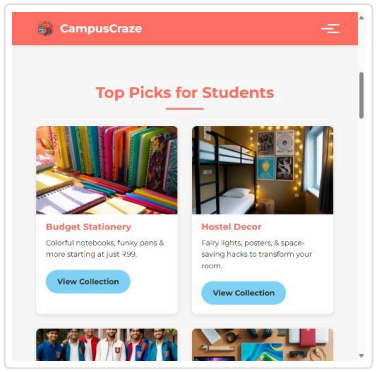
99	95	96	100
Performance	Accessibility	Best Practices	SEO

99

Performance

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

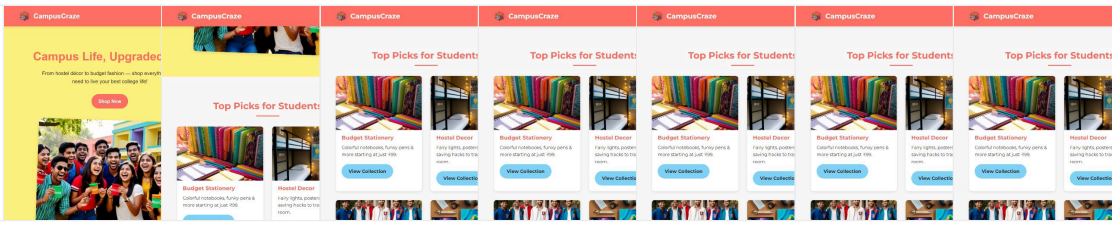
▲ 0–49 ■ 50–89 ● 90–100



METRICS Expand view

- First Contentful Paint 0.7 s
- Largest Contentful Paint 0.8 s
- Total Blocking Time 0 ms
- Cumulative Layout Shift 0.006
- Speed Index 0.8 s

View Treemap



Show audits relevant to: All FCP LCP TBT CLS

DIAGNOSTICS

- ▲ Eliminate render-blocking resources — Potential savings of 420 ms
- ■ Serve static assets with an efficient cache policy — 12 resources found
- ○ Avoid large layout shifts — 1 layout shift found
- ○ Avoid chaining critical requests — 3 chains found
- ○ Minimize third-party usage — Third-party code blocked the main thread for 0 ms
- ○ Largest Contentful Paint element — 820 ms
- ○ Avoid long main-thread tasks — 3 long tasks found

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

PASSED AUDITS (31) Show

Accessibility

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

CONTRAST

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

These are opportunities to improve the legibility of your content.

ADDITIONAL ITEMS TO MANUALLY CHECK (10) Show

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

PASSED AUDITS (18) Show

NOT APPLICABLE (38) Show

Best Practices

TRUST AND SAFETY

- ▲ Requests the notification permission on page load
- ○ Ensure CSP is effective against XSS attacks
- ○ Use a strong HSTS policy
- ○ Ensure proper origin isolation with COOP
- ○ Mitigate clickjacking with XFO or CSP

PASSED AUDITS (13) Show

NOT APPLICABLE (3) Show

SEO

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

ADDITIONAL ITEMS TO MANUALLY CHECK (1) Show

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

PASSED AUDITS (8) Show

NOT APPLICABLE (2) Show

Captured at Apr 9, 2025, 4:36 AM GMT+5:30

Initial page load

Emulated Desktop with Lighthouse 12.4.0

Custom throttling

Single page session

Using Chromium 135.0.0.0 with devtools

MPL Practical 11

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

Theory:

Google Lighthouse is a powerful open-source tool built into Chrome DevTools that helps test the quality of a Progressive Web App (PWA). It evaluates key aspects like performance, accessibility, SEO, and best practices, and provides a score along with suggestions for improvement.

In this experiment, we tested the CampusCraze PWA using Lighthouse at my github deployed website. The results showed excellent performance with a Performance score of 99, Accessibility 95, Best Practices 96, and SEO 100. The service worker was detected, manifest was valid, and offline support was working correctly. Minor suggestions like improving contrast and caching policies were noted for future enhancement.

This analysis confirmed that our PWA implementation is highly optimized and provides a reliable, fast, and installable user experience.

---

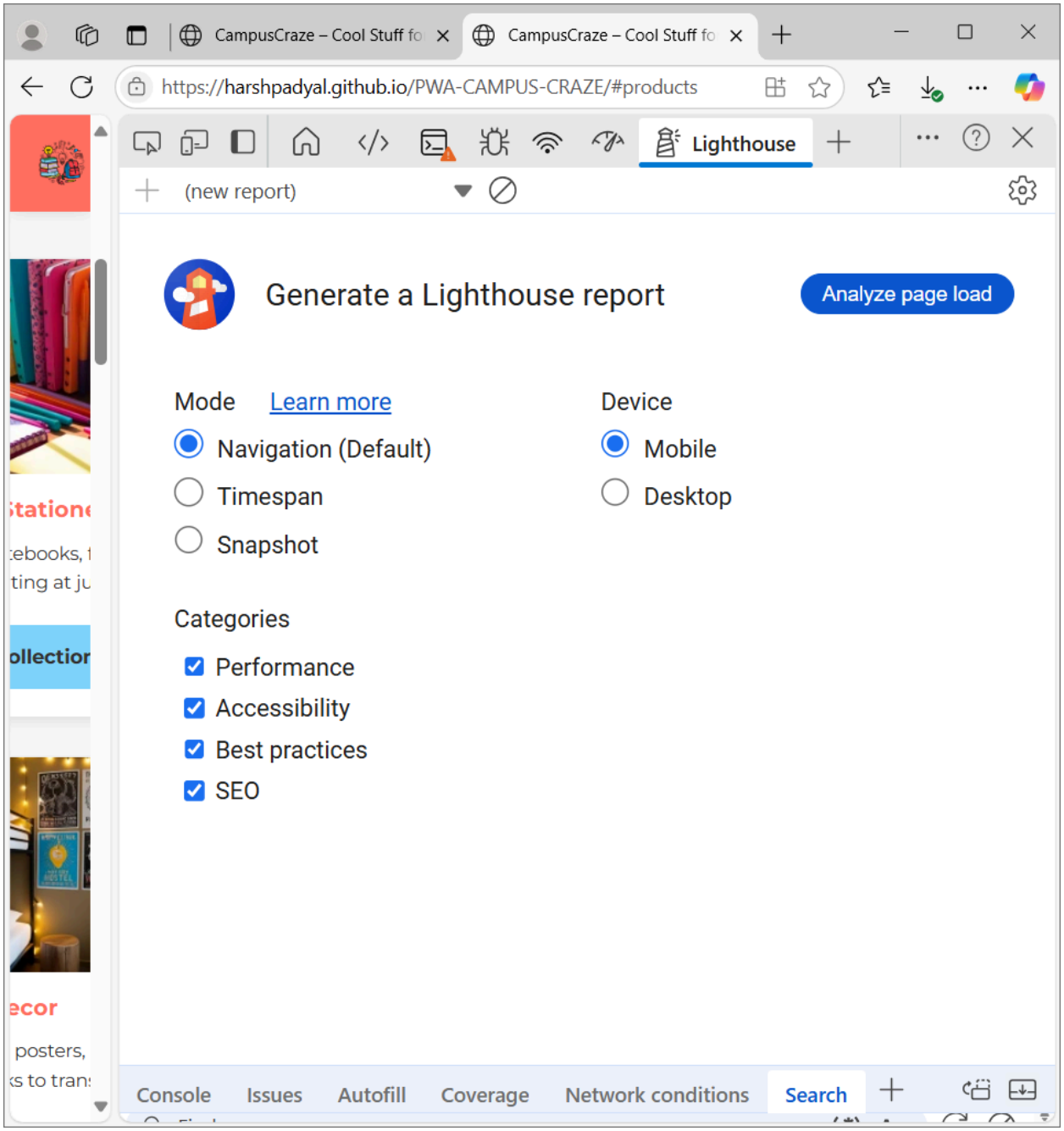
sw.js

```
// CampusCraze Service Worker for PWA functionality
const CACHE_NAME = 'campuscraze-v1';
const ASSETS = [
  '/PWA-CAMPUS-CRAZE/',
  '/PWA-CAMPUS-CRAZE/index.html',
  '/PWA-CAMPUS-CRAZE/styles.css',
  '/PWA-CAMPUS-CRAZE/main.js',
  '/PWA-CAMPUS-CRAZE/manifest.json',
  '/PWA-CAMPUS-CRAZE/images/logo.svg',
  '/PWA-CAMPUS-CRAZE/images/college-vibes.webp',
  '/PWA-CAMPUS-CRAZE/images/stationery.webp',
  '/PWA-CAMPUS-CRAZE/images/decor.webp',
  '/PWA-CAMPUS-CRAZE/images/fashion.webp',
  '/PWA-CAMPUS-CRAZE/images/tech.webp',
  '/PWA-CAMPUS-CRAZE/images/study-bundle.webp',
  '/PWA-CAMPUS-CRAZE/images/decor-bundle.webp',
  '/PWA-CAMPUS-CRAZE/images/app-mockup.webp',
  '/PWA-CAMPUS-CRAZE/images/google-play.svg',
  '/PWA-CAMPUS-CRAZE/images/app-store.svg',
  '/PWA-CAMPUS-CRAZE/images/avatar-simran.webp',
  '/PWA-CAMPUS-CRAZE/images/avatar-rishi.webp',
  '/PWA-CAMPUS-CRAZE/images/avatar-priya.webp',
  '/PWA-CAMPUS-CRAZE/images/instagram.svg',
  '/PWA-CAMPUS-CRAZE/images/youtube.svg',
```

```
  '/PWA-CAMPUS-CRAZE/images/twitter.svg',
  '/PWA-CAMPUS-CRAZE/icons/icon-192x192.png',
  '/PWA-CAMPUS-CRAZE/icons/icon-512x512.png',
  '/PWA-CAMPUS-CRAZE/offline.html',
  'https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;700&display=swap'
];

// 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(ASSETS);
    })
  );
});

// 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("CampusCraze - Cool Stuff for College Life", {
        body: data.message,
      })
    );
  }
 }
});
```



Conclusion

In this experiment, we used Google Lighthouse to analyze the PWA performance of CampusCraze and achieved near-perfect scores, confirming our implementation was correct. Initially, the service worker wasn't detected because of a wrong file path during GitHub Pages deployment, which we fixed by updating all asset links with the /PWA-CAMPUS-CRAZE/ prefix.

Name:- Harsh Pramod Padyal D15B 40

Page No.
Date

MPL Assign 01

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

Ans.

Key Features of Flutter:-

1. Single codebase.
2. Fast performance
3. Hot Reload.
4. Rich UI components.
5. Native-like Experience
6. Cross-platform support
7. Open-source.

Advantages of Using Flutter:-

1. Saves Time & Effort.
2. High Speed Development.
3. Cost-Effective.
4. Attractive UI
5. Good Performance
6. Easy Integration.

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

Ans How Flutter Differs from Traditional Approaches:-

1. Single Codebase.
2. Hot Reload - Traditional apps require full restart after changes, But flutter updates instantly.
3. UI Rendering - Traditional apps require full restart after changes, but flutter updates instantly.

4. Performance:- Flutter compiles directly to native machine code, making it faster than frameworks that use a bridge (eg. React Native)

5. Customization:- Traditional UI design depends on platform-specific components, but Flutter provides fully customizable widgets.

Why Flutter is Popular Among Developers:

1. Fast Development - Hot Reload and Single codebase save time.

2. Cross-Platform support - works on mobile, web and desktop.

3. Beautiful UI - Rich, customizable widgets for modern designs.

4. High Performance - Runs smoothly without a bridge like React Native.

SDS Page No.
Date / /

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

Ans.

Concept of widget Tree in Flutter :-

In Flutter, everything is a widget. widgets are arranged in a tree structure, called the widgets 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.

For eg :-

1. A ListView can contain multiple Card widgets.

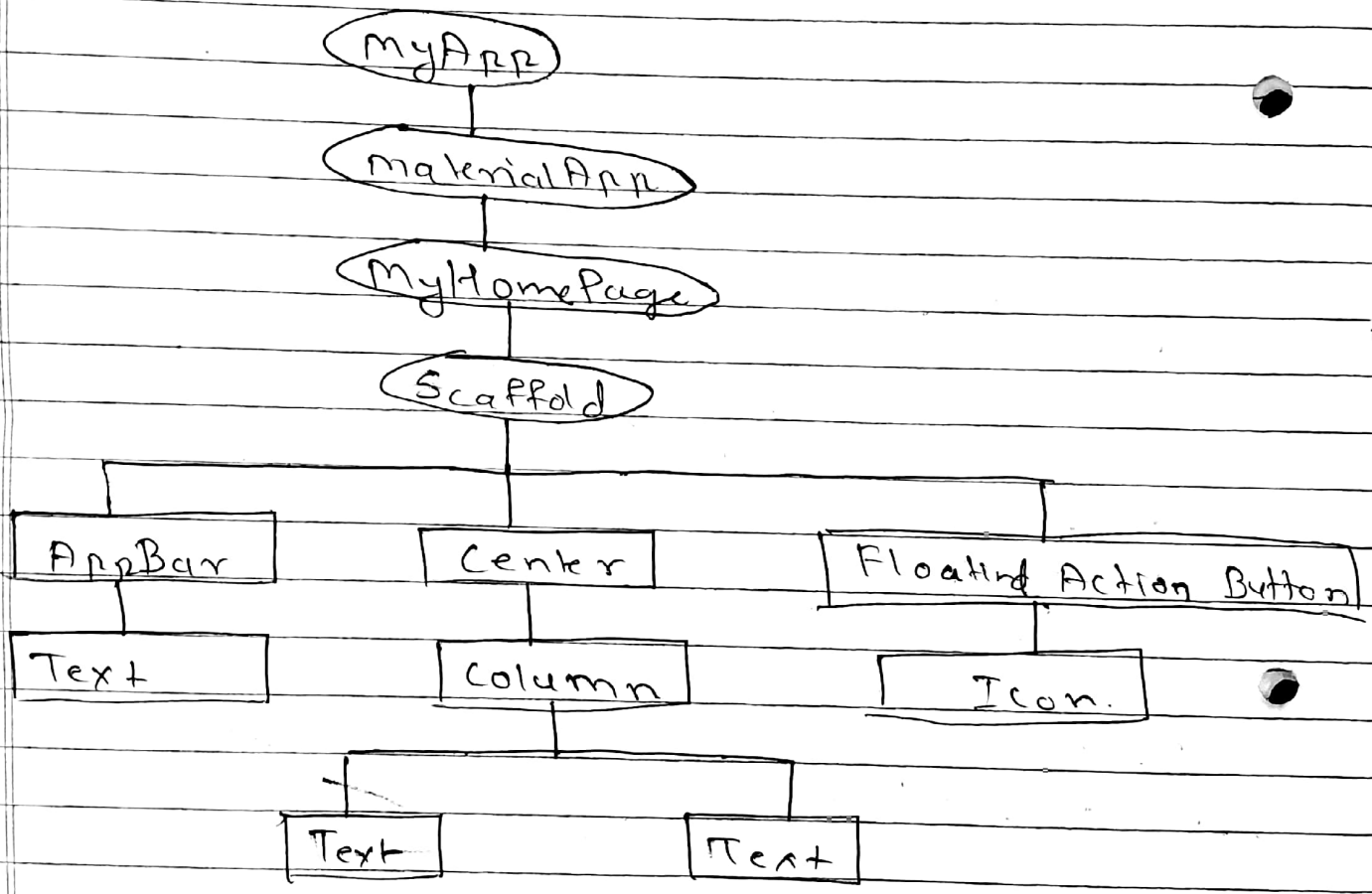
2. A column can hold Texts, Images, and Buttons.

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

Ans. 1. Scaffold
2. Appbar
3. Text
4. Image
5. container
6. Row.

7. column
8. ListView
9. Elevated Button
10. TextField
11. Card
12. Stack.

Tree :-



Q-3)a) Discuss the importance of state management in Flutter application

Ans Importance of state Management in Flutter Applications:-

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

Why state Management is needed?

1. Keeps UI updated
2. Improves Performance
3. manage Complex Data
4. Ensures Smooth User Experience.

Types of State in Flutter:-

1. Local State.
2. Global state.

b) Compares and contrast the different state management

Ans

Approach	How it works	when to use
SetState	updates UI by calling setState() in a StatefulWidget	Best for small Apps or 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 themes.

SDS Page No.
Date / /

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

Ans

Process of Integrating Firebase with a Flutter Application:-

1. Create a Firebase Project -
   Go to [Firebase Console](https://console.firebase.google.com/), create a new project

2. Add Firebase to Flutter App -
   Register the app (Android / iOS) and download the google-services.json (Android) or GoogleService-Info.plist (iOS).

3. Install Firebase Packages -

4. Initialize Firebase - Import Firebase in 'main.dart' and call 'Firebase.initializeApp()'.

5. Use Firebase Services - Implement authentication, database, or cloud functions as needed.

SDS Page No.
Date / /

Q.4) b) Highlight the Firebase Services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.

Ans.

Common Firebase Services Used in Flutter Development:-

1. Firebase Authentication - Provides user sign-in methods (Google, Email, Facebook, etc.)

2. Cloud Firestore - A NoSQL database

3. Firebase Realtime Database - Stores and updates data instantly across all connected database.

4. Firebase Cloud Storage - Usage for storing and retrieving files like images and videos.

5. Firebase Hosting - Deploys web apps with fast and secure hosting.

SDS Page No.
Date / /

How Data synchronization is Achieved:-

1. Real-time Updates - Firestore and Realtime Database sync data across devices instantly.

2. Listeners and streams - widgets listen for changes and update the UI automatically.

3. offline support - Firebase caches data, allowing app to work offline and sync when online.

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

Name:- Harsh Pramod Padyal DISB 40

MPL Assign 02

05/05

Page No.
Date

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

Ans

A progressive web App (PWA) is a type of web application that works like a mobile app but runs in a browser

Significance of PWA

1. Cross-Platform Compatibility.
2. Offline Support.
3. Fast Performance.
4. No App store Required.
5. Lower Development cost.

Key Differences betwn PWA and Traditional mobile Apps :-

Feature	PWA	Traditional Mobile App
Installation	Direct from browser	Download from App store
Internet Required	works offline with caching	Usually requires internet
Performance	Fast with service workers.	Faster but needs installation.
Updates	Automatic, no app store approval	Manual updates require

Feature	PWA	Traditional Mobile App.
Development cost	Lower (one codebase for all)	Higher (seperate apps for each platform)

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

Ans.

Defination of Responsive web Design:
Responsive web Design (RWD) is a technique that makes web pages adjust automatically to different screen sizes and devices.

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 Benifits - Google ranks responsive sites higher.

4. Cost-Effective - No need to build multiple versions for different screens.

Page No.
Date

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 types.

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

Ans

Lifecycle of service Workers

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

1. Registration Phase :-

- The brower 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:' error));
}
```

2. Installation Phase :-

- The service workers downloads necessary files (HTML, CSS, JS) and stores them in cache

code Example:

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

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(key => {
      return Promise.all(keys.map(key => {
        if (key !== 'app-cache'){
          return caches.delete(key);
        }
      }));
    })
  );
});
```

Page No.
Date

Final step: Fetch and Sync

Once activated, the Service Worker intercepts network requests, serves cached files, and Syncs data when the internet available.

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

Ans

Use of IndexedDB in Service Worker for Data Storage :-

IndexedDB is a browser database that stores large amounts of structured data like JSON objects.

Why Use IndexedDB in Service Workers? :-

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.

How Service Workers Use IndexedDB?

Opening the Database :-

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

request.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: 'John Doe', age: 25});
};
```

Fetching Data in Service Worker :-

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

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