Vivekanand Education Society's Institute of Technology

An Autonomous Institute Affiliated to University of Mumbai
Hashu Advani Memorial Complex, Collector Colony, Chembur East, Mumbai - 400074.

Department of Information Technology

CERTIFICATE

This is to certify that **Soham Shetye** of **D15A** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Name of the Course : MAD & PWA Lab

Course Code : ITL604

Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**

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

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers **:** Mrs. Kajal Jewani.

Email **:** kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**

Lab Objectives:

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

Lab Outcomes:

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

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1	19/01	02/02	15
2.	To design Flutter UI by including common widgets.	LO2	26/01	02/02	15
3.	To include icons, images, fonts in Flutter app	LO2	02/02	09/02	15
4.	To create an interactive Form using form widget	LO2	09/02	16/02	15
5.	To apply navigation, routing and gestures in Flutter App	LO2	16/02	23/02	15
6.	To Connect Flutter UI with fireBase database	LO3	23/02	08/03	15
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4	08/03	22/03	15
8.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA	LO5	15/03	22/03	15
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	22/03	01/04	15
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	29/03	01/04	15
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	29/03	01/04	15
12.	Assignment-1	LO1,LO2 ,LO3	28/01	05/02	05
13.	Assignment-2	LO4,LO5 ,LO6	14/03	21/03	04

MAD & PWA Lab
Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	15

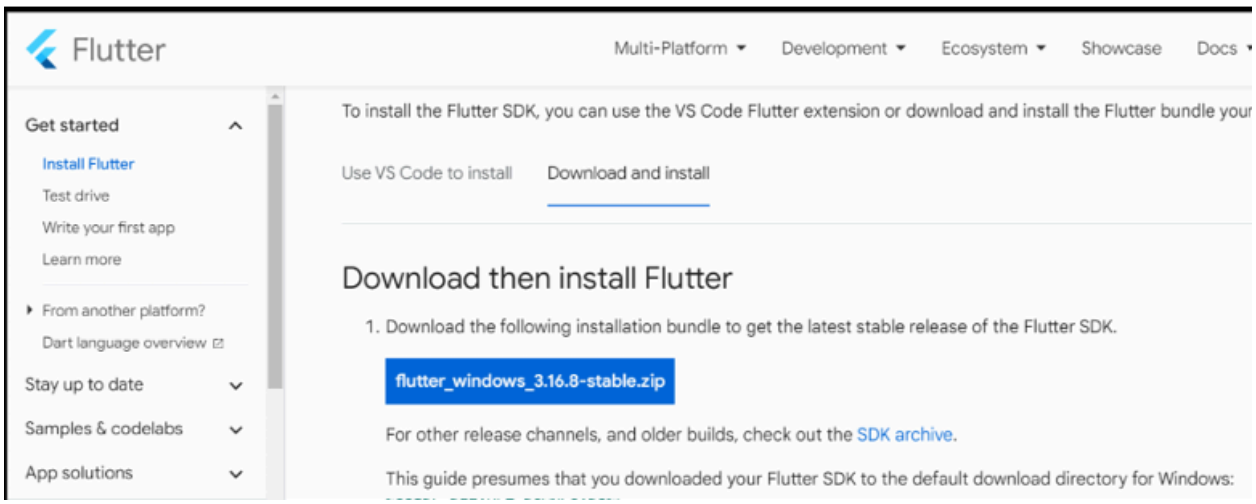
NAME : SOHAM SHETYE ROLL NO. : 54
DIV : D15A BATCH : C

MAD PWA LAB

LAB 1

Aim: To install flutter and Android Studio and create a 'Hello World App' using Flutter

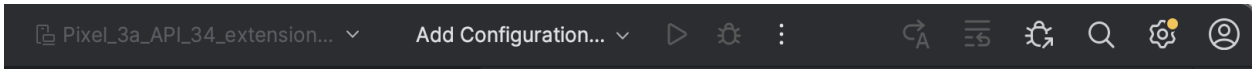
Step 1 :
Create the app
1. Open the IDE and select Create New Flutter Project.
2. Select Flutter Application as the project type. Then click Next.
3. Verify the Flutter SDK path specifies the SDK‟s location (select Install SDK... if the
text field is blank).
4. Enter a project name (for example, myapp). Then click Next.
5. Click Finish.
6. Wait for Android Studio to install the SDK and create the project.

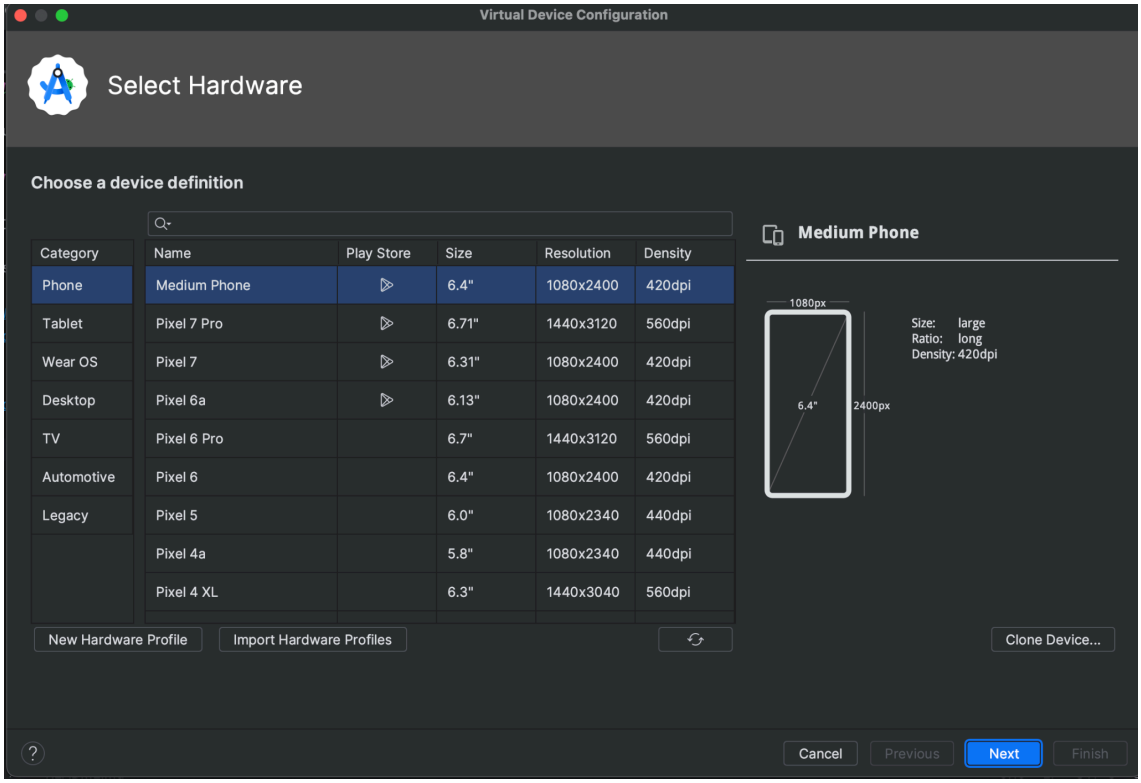


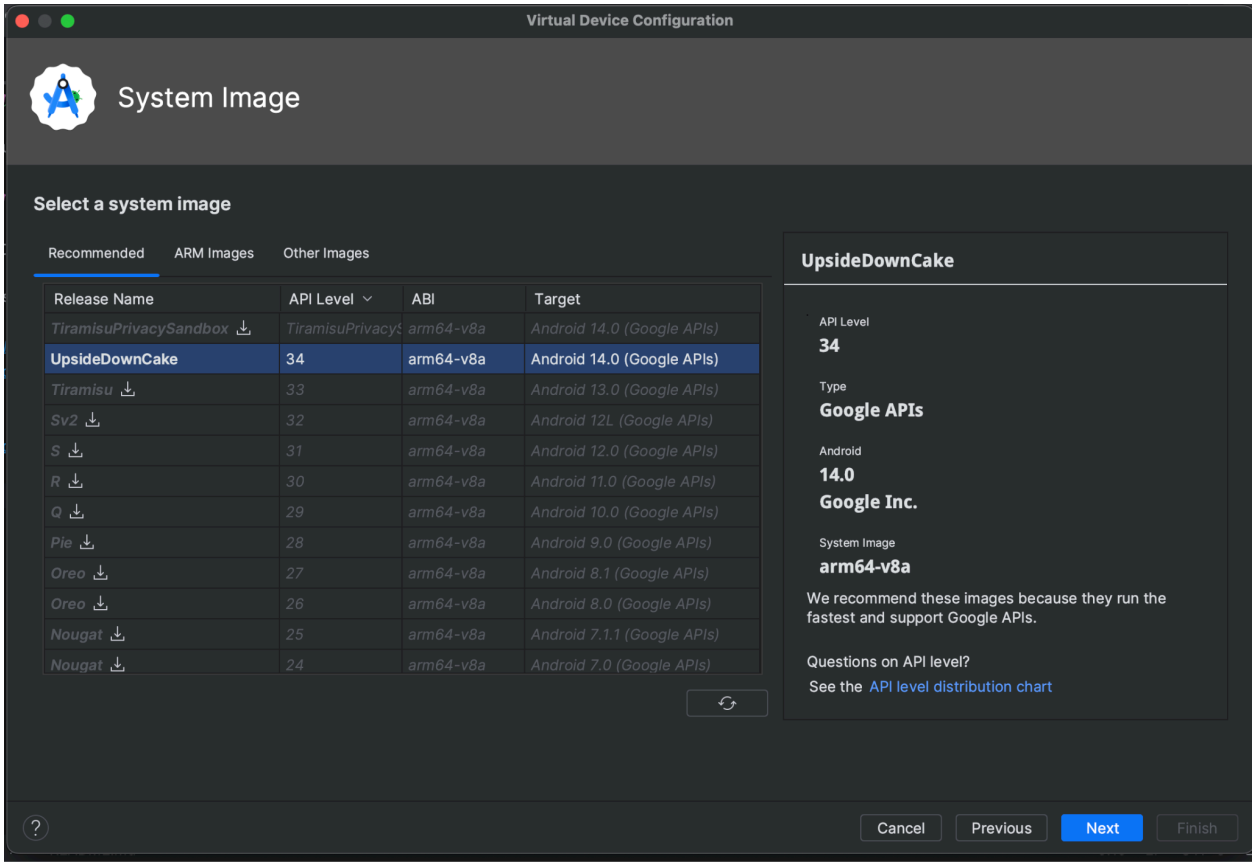
```
C:\Users\acer>flutter --version

┌──────────────────────────────────────────────────────────┐
│ A new version of Flutter is available!                   │
│                                                          │
│ To update to the latest version, run "flutter upgrade".  │
└──────────────────────────────────────────────────────────┘
Flutter 3.16.7 • channel stable • https://github.com/flutter/flutter.git
Framework • revision ef1af02aea (8 days ago) • 2024-01-11 15:19:26 -0600
Engine • revision 4a585b7929
Tools • Dart 3.2.4 • DevTools 2.28.5
```

```
C:\Users\acer>flutter devices
Found 3 connected devices:
  Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.3007]
  Chrome (web)      • chrome  • web-javascript • Google Chrome 120.0.6099.225
  Edge (web)        • edge    • web-javascript • Microsoft Edge 120.0.2210.121

Run "flutter emulators" to list and start any available device emulators.

If you expected another device to be detected, please run "flutter doctor" to diagnose potenti
try increasing the time to wait for connected devices with the "--device-timeout" flag. Visit
for troubleshooting tips.
```

Main Android Studio Toolbar





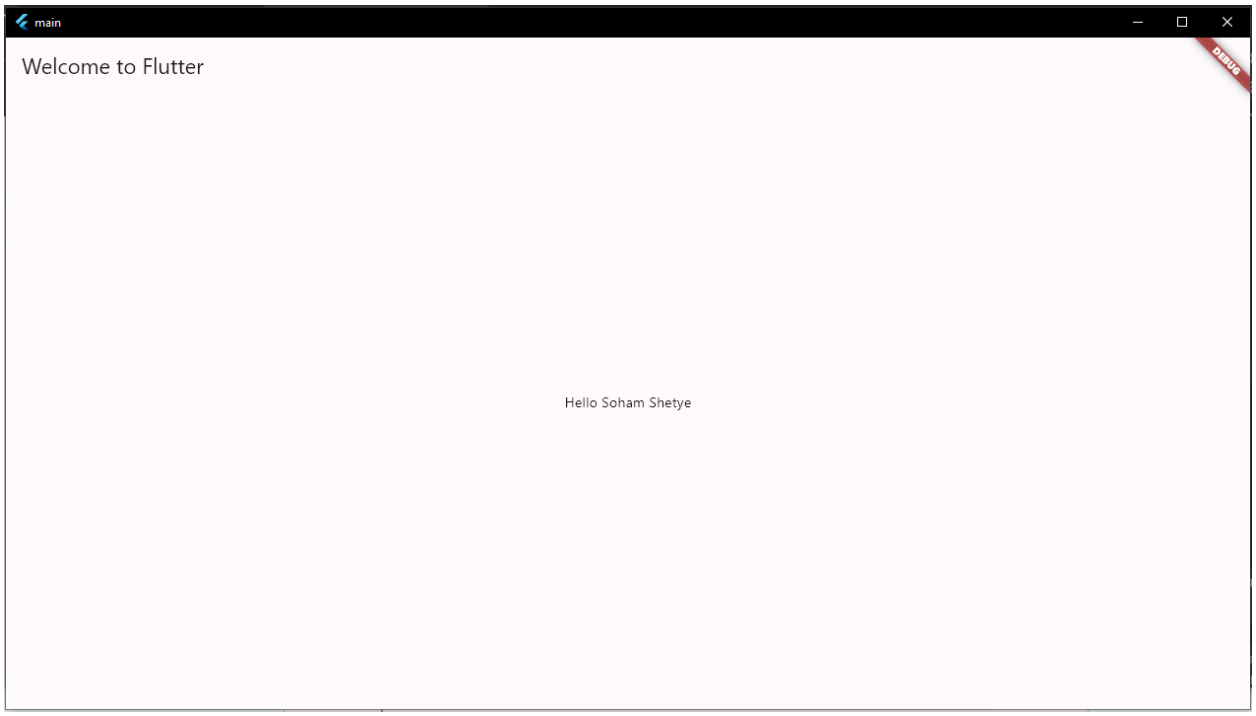


1. Replace the contents of lib/main.dart. - Delete all of the code from lib/main.dart.
2. Replace with the following code, which displays “Hello World” in the center of the
screen.

```
import 'package:flutter/material.dart';
void main() {
runApp(const MyApp());
}
class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Welcome to Flutter',
home: Scaffold(
```

```
appBar: AppBar(
title: const Text('Welcome to Flutter'),
),
body: const Center(
child: Text('Hello World'),
),
),
);
}
}
```

3. Run the app by selecting Run> Run „main.dart" and see the output in emulator device.



Conclusion: Thus,we installed flutter and android studio and created our first flutter application.

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

NAME : SOHAM SHETYE ROLL NO. : 54
DIV : D15A BATCH : C

MAD PWA LAB

LAB 2

Aim: To design Flutter UI by including common widgets.

THEORY :

Designing Flutter UI involves using a variety of widgets to create a visually appealing and functional layout. Key widgets include:

1. Structural Widgets:
   - `Container`: Box model for customization.
   - `Row`, `Column`: Arranging widgets horizontally or vertically.
   - `Stack`: Overlapping widgets for layering.

2. Text and Styling:
   - `Text`: Displaying text with style.
   - `RichText`: Inline text styling.
   - `TextStyle`: Defining text appearance.

3. Images:
   - `Image`: Displaying images from various sources.

4. User Input and Interaction:
   - `GestureDetector`: Capturing user gestures.
   - `TextField`: Accepting text input.
   - Button widgets for triggering actions.

5. Navigation:
   - `Navigator`: Managing navigation stack.
   - `PageRoute`: Defining transitions between pages.

6. Lists and Scrollable Widgets:
   - `ListView`, `GridView`: Scrollable lists and grids.
   - `SingleChildScrollView`: Scrolling a single widget.

7. Material Design Widgets:
   - `Scaffold`: Basic material design structure.
   - `AppBar`, `BottomNavigationBar`: Common material design components.

8. Themes and Styles:
   - `Theme`, `ThemeData`: Defining visual themes.

9. Animations:
   - Animated widgets for creating animations.
   - `Hero`: Transition animations between routes.

10. State Management:
    - `StatefulWidget`: Managing mutable state.
    - External packages for advanced state management.

11. Internationalization and Localization:
    - Using the `intl` package for internationalization.

12. Testing and Debugging:
    - Flutter DevTools for profiling and debugging.
    - Widget testing for UI components.

13. Custom Widgets:
    - Creating reusable widgets by combining existing ones.
    - `CustomPainter` for custom graphics.

14. Accessibility:
    - `Semantics`: Enhancing accessibility.

15. Performance Optimization:
    - Efficient use of keys and `const` constructors.

16. Package Integration:
    - Incorporating third-party packages for additional features.

17. Documentation and Best Practices:
    - Referencing Flutter documentation and best practices.

Thoughtful combination of these widgets, adherence to design principles, and user experience considerations contribute to effective Flutter UI design. Regular testing and iteration are crucial for refinement.

#main.dart

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

void main() {
```

```
 runApp(MyApp());
}

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

class LoginPage extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Sohams OLA'),
    ),
    body: Container(
      decoration: BoxDecoration(
        image: DecorationImage(
          image: AssetImage('assets/projects.png'), // Replace with your image file
          fit: BoxFit.cover,
        ),
      ),
      child: Column(
        children: [
          Expanded(
            child: Align(
              alignment: Alignment.center,
              child: Padding(
                padding: const EdgeInsets.all(16.0),
                child: Text(
                  '.',
                  style: TextStyle(
                    fontSize: 20,
                    fontWeight: FontWeight.bold,
                    color: Colors.white,
                  ),
```

```
              ),
            ),
          ),
        ),
        Padding(
          padding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 30.0), // Adjust padding
here
          child: Column(
            mainAxisAlignment: MainAxisAlignment.end, // Align children at the bottom
            children: [
              ElevatedButton(
                onPressed: () {
                  // Handle sign-in button press
                  // Add your sign-in logic here
                },
                child: Text('Sign In'),
              ),
              SizedBox(height: 10), // Adjust spacing here
              OutlinedButton(
                onPressed: () {
                  // Handle register button press
                  // Add your register logic here
                },
                child: Text('Register'),
              ),
            ],
          ),
        ),
      ],
    ),
  ),
 );
 }
}
```

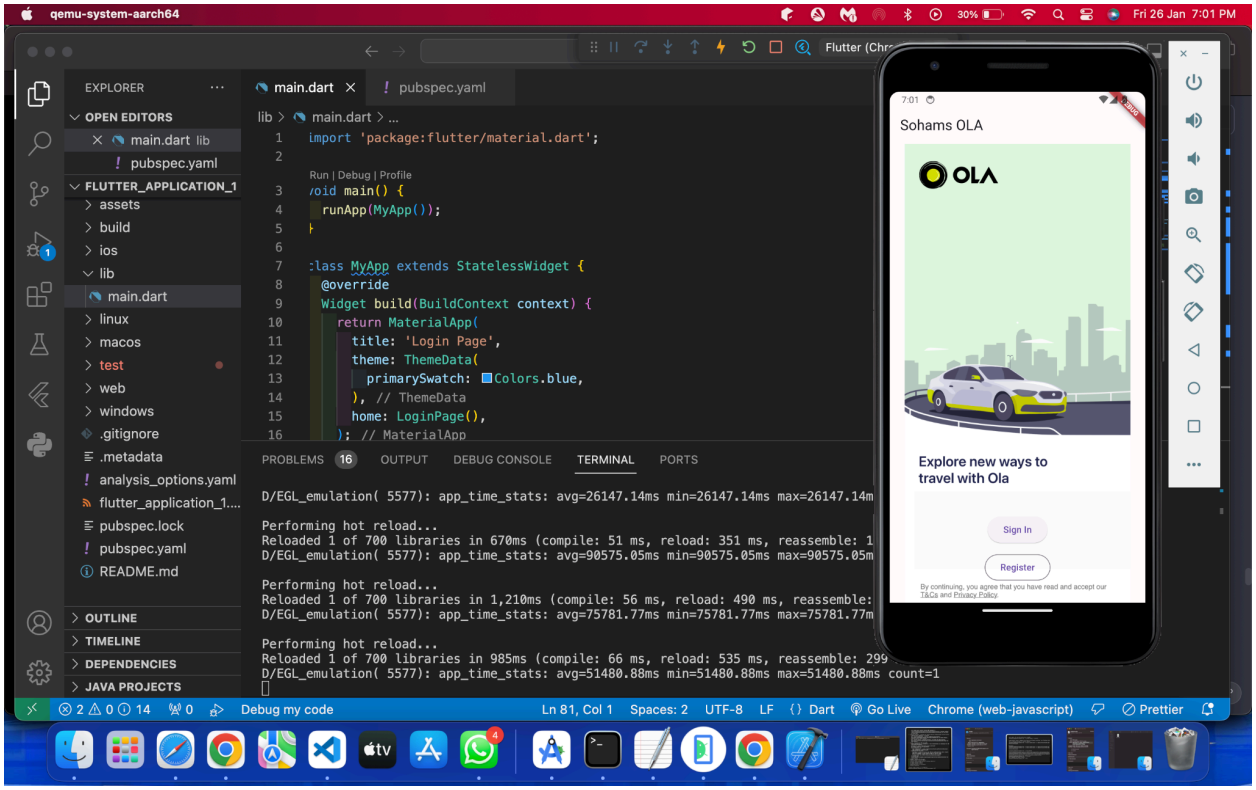
#pubspec.yaml

```
name: flutter_application_1
description: "A new Flutter project."
# The following line prevents the package from being accidentally published to
# pub.dev using `flutter pub publish`. This is preferred for private packages.
publish_to: 'none' # Remove this line if you wish to publish to pub.dev

# The following defines the version and build number for your application.
# A version number is three numbers separated by dots, like 1.2.43
```

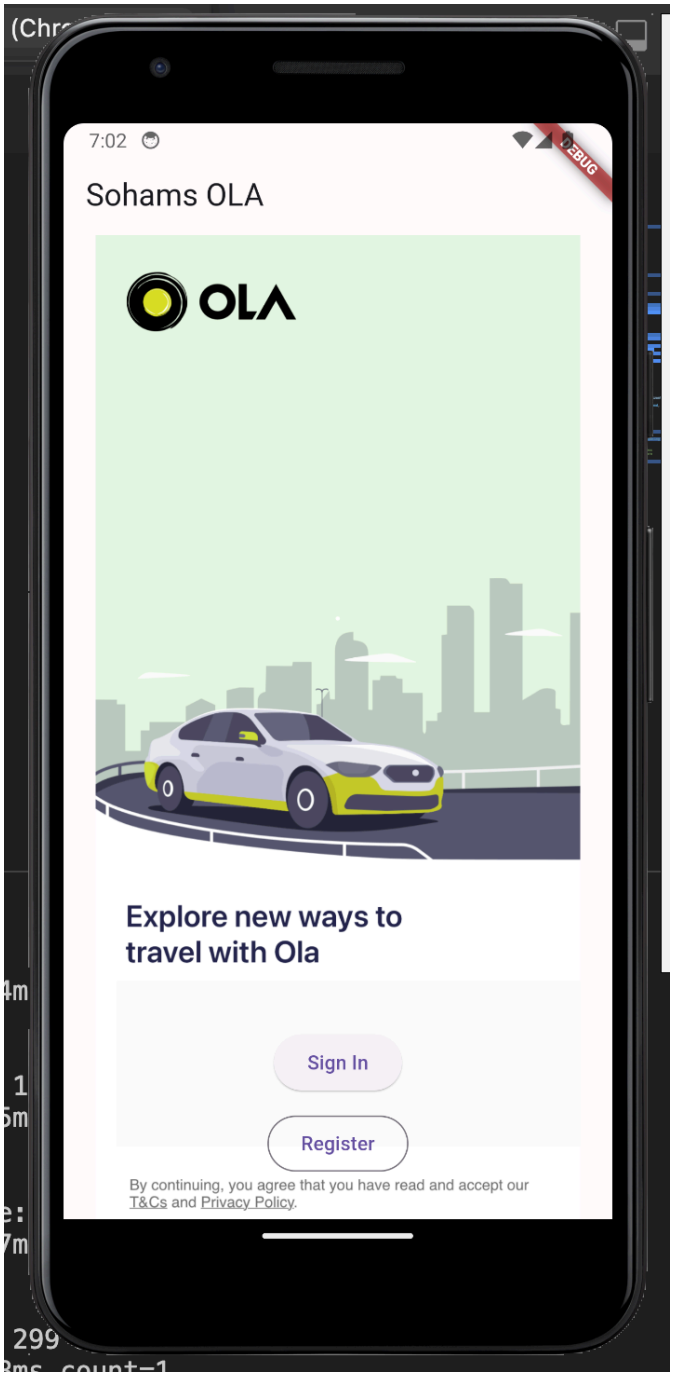
```
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as versionCode.
# Read more about Android versioning at
https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-number is used as
CFBundleVersion.
# Read more about iOS versioning at
#
https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyRefer
ence/Articles/CoreFoundationKeys.html
# In Windows, build-name is used as the major, minor, and patch parts
# of the product and file versions while build-number is used as the build suffix.
version: 1.0.0+1

environment:
 sdk: '>=3.2.6 <4.0.0'

# Dependencies specify other packages that your package needs in order to work.
# To automatically upgrade your package dependencies to the latest versions
# consider running `flutter pub upgrade --major-versions`. Alternatively,
# dependencies can be manually updated by changing the version numbers below to
# the latest version available on pub.dev. To see which dependencies have newer
# versions available, run `flutter pub outdated`.
dependencies:
 flutter:
   sdk: flutter
 font_awesome_flutter: ^9.0.2


 # The following adds the Cupertino Icons font to your application.
 # Use with the CupertinoIcons class for iOS style icons.
 cupertino_icons: ^1.0.2

dev_dependencies:
 flutter_test:
   sdk: flutter

 # The "flutter_lints" package below contains a set of recommended lints to
 # encourage good coding practices. The lint set provided by the package is
 # activated in the `analysis_options.yaml` file located at the root of your
 # package. See that file for information about deactivating specific lint
 # rules and activating additional ones.
```

```
 flutter_lints: ^2.0.0

# For information on the generic Dart part of this file, see the
# following page: https://dart.dev/tools/pub/pubspec

# The following section is specific to Flutter packages.
flutter:

 # The following line ensures that the Material Icons font is
 # included with your application, so that you can use the icons in
 # the material Icons class.
 uses-material-design: true

 # To add assets to your application, add an assets section, like this:
 assets:
     - assets/projects.png
 #   - images/a_dot_ham.jpeg

 # An image asset can refer to one or more resolution-specific "variants", see
   #https://flutter.dev/assets-and-images/#resolution-aware

 # For details regarding adding assets from package dependencies, see
   #https://flutter.dev/assets-and-images/#from-packages
```

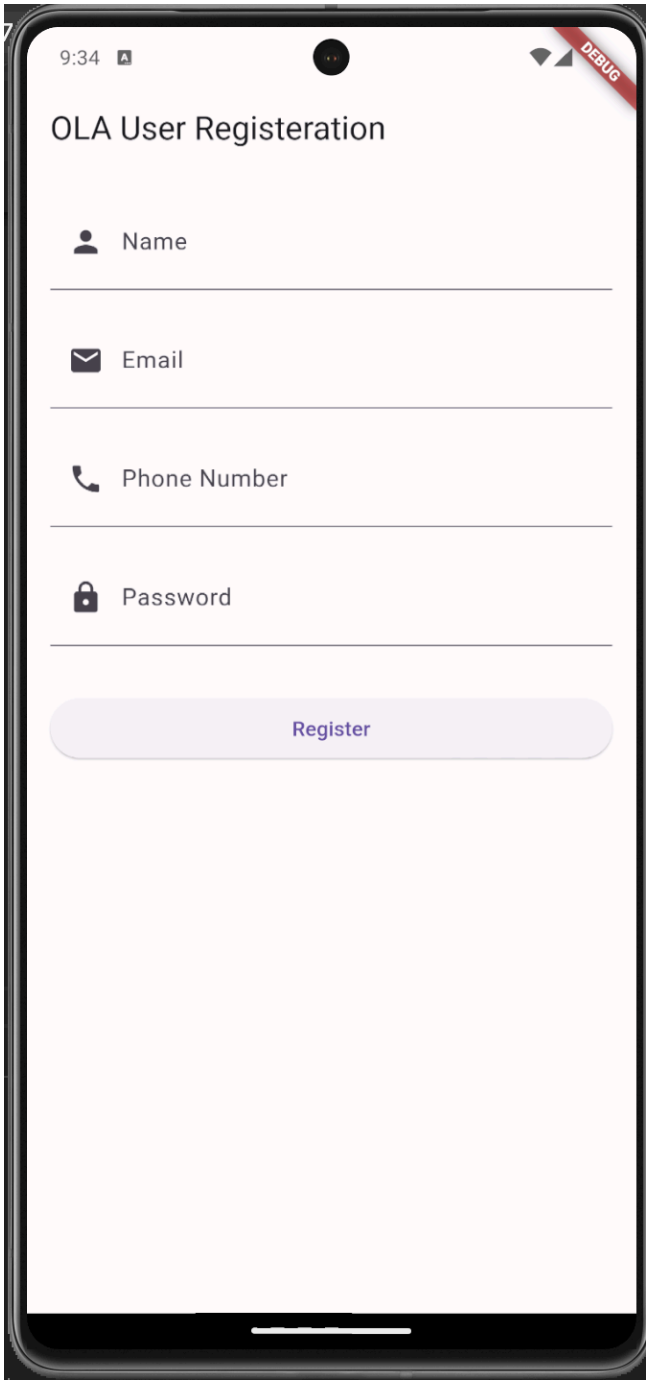
Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**



Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**



Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**



Widgets used :-

MaterialApp:

- The top-level widget that represents the entire application. It provides the overall theme and configuration.

Scaffold:

- Represents the basic structure of the visual interface, including the app bar and body.

AppBar:

- Displays the top app bar with the title 'Sohams OLA'.

Container:

- A box model widget that contains other widgets and is used here for the background image.

Decoration:

- The BoxDecoration with DecorationImage is used to set the background image for the container.

Column:

- Arranges its children vertically. Used to structure the UI with a column of widgets.

Expanded:

- Takes up the available space in the vertical direction. Used to make the first child of the column (containing the dot) take up the remaining space.

Align:

- Aligns its child within itself. Used to center the dot within the expanded space.

Padding:

- Adds padding around its child. Used to create space around the dot and adjust its position.

Text:

- Displays text on the screen. Used to display the dot with specific styling.

ElevatedButton and OutlinedButton:

- Widgets for creating buttons with different visual styles. Used for the "Sign In" and "Register" buttons.

SizedBox:

- Creates a box with a specified size. Used to add spacing between the buttons.

CONCLUSION :

Designing a Flutter UI encompasses leveraging a diverse set of widgets for structure, styling, interaction, and responsiveness, coupled with adherence to design principles, resulting in a visually appealing and user-friendly application.

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

NAME : SOHAM SHETYE ROLL NO. : 54
DIV : D15A BATCH : C

MAD PWA LAB

LAB 3

Aim: To include icon images and fonts in flutter.

THEORY :

Custom Fonts in Flutter:

What is a Font?
A font is a set of characters with a specific style and size. Fonts are used to define the visual appearance of text in your application.

Custom Fonts in Flutter:
In Flutter, you can use custom fonts to give a unique look to the text in your application. Custom fonts are often used to match a specific brand or design aesthetic.

- How to Add Custom Fonts:
  1. Font Files: Custom fonts are typically provided as font files (e.g., TrueType Font - `.ttf` or OpenType Font - `.otf`).
  2. Directory: Place your font files in the `assets/fonts/` directory in your Flutter project.
  3. Update `pubspec.yaml`: Add an entry in your `pubspec.yaml` file specifying the font file and its family name.

Example:
```yaml
flutter:
  fonts:
    - family: YourCustomFont
      fonts:
        - asset: assets/fonts/your_custom_font.ttf
```

- Using Custom Fonts in Text Widget:
  Use the `fontFamily` property in the `TextStyle` to apply your custom font to a `Text` widget.

Example:
```dart
Text(
  'Hello, World!',
  style: TextStyle(fontFamily: 'YourCustomFont'),
)
```

Images in Flutter:

What is an Image?
An image is a visual representation or graphic in a digital form. In Flutter, images are used to display visual content.

Images in Flutter:
Flutter supports different types of images, including local assets and network images.

- How to Add Images:
  1. Image Files: Images can be PNG, JPEG, GIF, etc.
  2. Directory:*Place your image files in the `assets/images/` directory in your Flutter project.
  3. Update `pubspec.yaml`:** Add an entry to the `assets` section in your `pubspec.yaml` file to include the image files.

Example:
```yaml
flutter:
  assets:
    - assets/images/your_image.png
```

- Using Images in Flutter:
  You can use the `Image` widget or `Image.asset` to display images.

Example:
```dart
Image.asset('assets/images/your_image.png')
```

```

Icons in Flutter:

What is an Icon?
An icon is a visual symbol representing an object, concept, or action. Icons are used to provide a quick and recognizable representation of functionality in an application.

Icons in Flutter:
Flutter provides a set of built-in icons that you can use directly. Additionally, you can use custom icons by importing image assets or using custom icon font libraries.

- Using Built-in Icons:
  Flutter's `Icons` class provides a variety of built-in icons that you can use directly.

Example:
```dart
Icon(Icons.home)
```

- Using Custom Icons:
  1. Image Assets: Import custom icons as image assets.

  Example:
  ```dart
  Image.asset('assets/icons/custom_icon.png')
  ```

  2. Icon Fonts: Use custom icon fonts (e.g., Font Awesome) by importing and using the corresponding icons.

  Example:
  ```dart
  Icon(FontAwesomeIcons.someIcon)
  ```

These concepts collectively contribute to the visual design and aesthetics of your Flutter application, allowing you to create a visually appealing and brand-consistent user interface.
  ```

CODE :

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: PhoneNumberInputScreen(),
    );
  }
}

class PhoneNumberInputScreen extends StatefulWidget {
  @override
  _PhoneNumberInputScreenState createState() => _PhoneNumberInputScreenState();
}

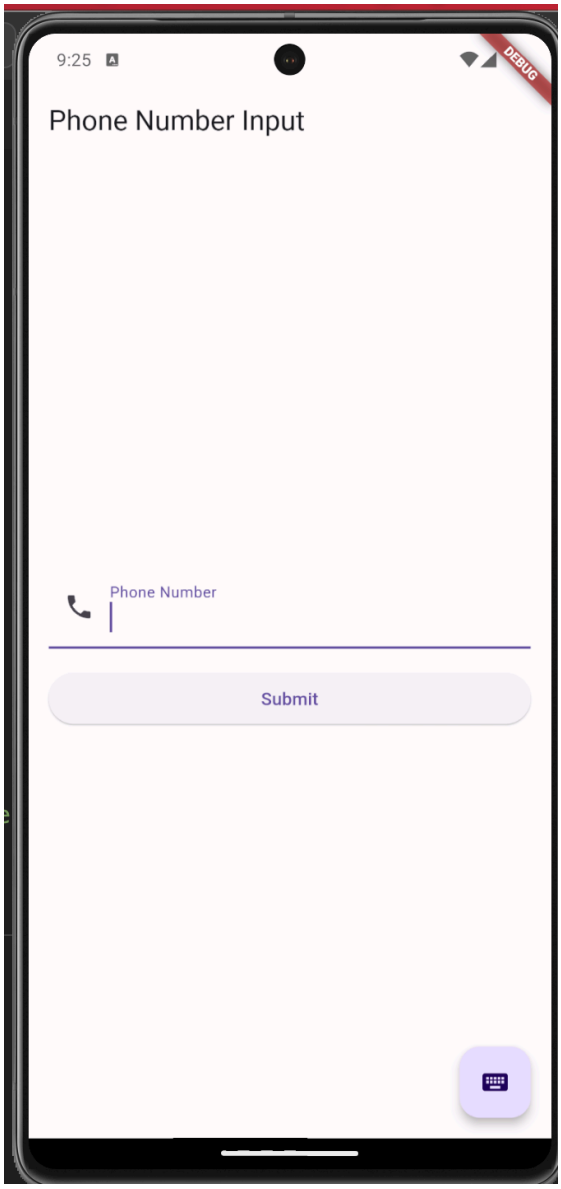
class _PhoneNumberInputScreenState extends State<PhoneNumberInputScreen> {
  TextEditingController _phoneNumberController = TextEditingController();

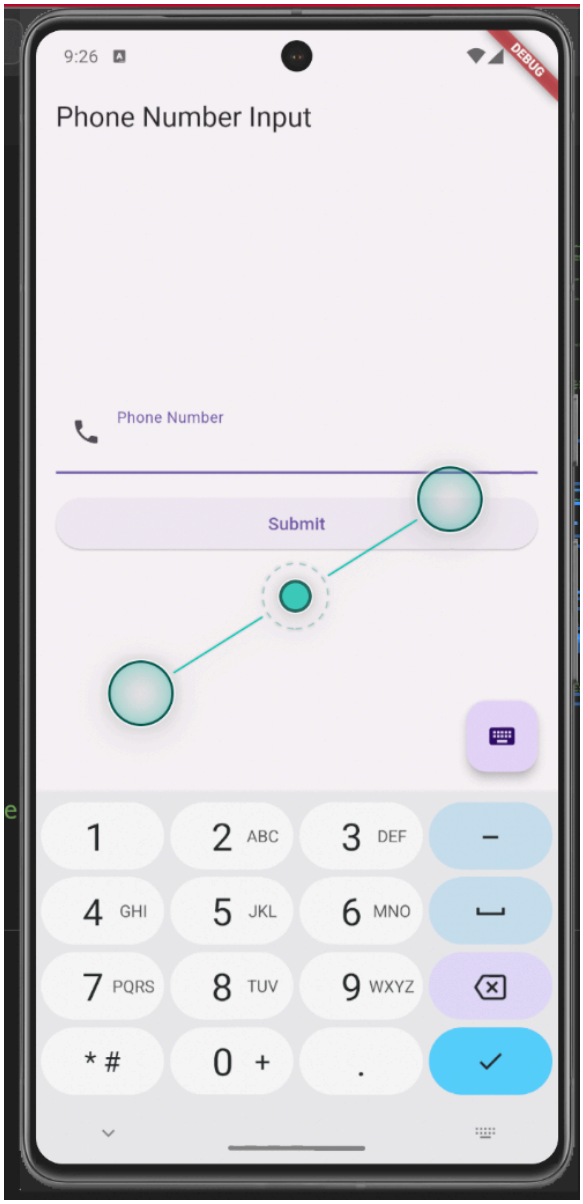
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Enter Phone number for verification'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            TextField(
              controller: _phoneNumberController,
              keyboardType: TextInputType.phone,
              decoration: InputDecoration(
```

```
            labelText: 'Phone Number',
            prefixIcon: Icon(Icons.phone),
          ),
        ),
        SizedBox(height: 16.0),
        ElevatedButton(
          onPressed: () {
            // Handle phone number submission
            String phoneNumber = _phoneNumberController.text;
            print('Phone Number: $phoneNumber');
          },
          child: Text('Submit'),
        ),
      ],
    ),
  ),
  floatingActionButton: FloatingActionButton(
    onPressed: () {
      // Show floating keypad
      // You can implement your own keypad logic or use a package
      // like 'flutter_numeric_keyboard' for a numeric keypad.
      // For simplicity, we'll just print a message here.
      print('Floating Keypad Pressed');
    },
    tooltip: 'Floating Keypad',
    child: Icon(Icons.keyboard),
  ),
 );
 }
}
```

OUTPUT :

Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**





The code uses built-in Flutter icons from the Icons class.

- Icons.person for the name input.
- Icons.email for the email input.
- Icons.phone for the phone number input.
- Icons.lock for the password input.
- ppBar: The app bar in the PhoneNumberInputScreen uses the Icons.phone icon for the prefix of the phone number input.
- ElevatedButton: The submit button uses the default Icons.check icon.
- FloatingActionButton: The floating action button (FAB) uses the Icons.keyboard icon.

- AppBar Action: An additional app bar action is included with the Icons.info icon, which is triggered when the user presses the info button.

Conclusion :

The provided Flutter registration page code utilizes built-in Flutter icons (`Icons.person`, `Icons.email`, `Icons.phone`, and `Icons.lock`) for user input fields, and it doesn't include specific images or custom fonts, allowing flexibility for customization based on individual design preferences.

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

MAD PWA LAB

LAB 4

Aim: To create interactive form using widgets

THEORY :

Interactive forms go beyond the static text boxes and radio buttons. They leverage widgets to create engaging and user-friendly experiences that improve data collection and user satisfaction.

Here's a theoretical dive into the world of interactive forms with widgets:

What are Widgets?

Widgets are pre-built user interface elements that add specific functionalities to your forms. They act as building blocks, offering a variety of interactive features like:

- **Dropdowns:** Allow users to select from a list of options.
- **Sliders:** Enable users to choose a value within a specific range.
- **Date pickers:** Facilitate easy date selection.
- **Image uploaders:** Permit users to submit pictures.
- **Progress bars:** Show the form completion progress.
- **Conditional logic:** Display or hide fields based on previous answers.
- **Interactive maps:** Allow users to pinpoint locations.

Benefits of using Widgets:

- **Enhanced User Experience:** Widgets make forms more engaging and intuitive,leading to higher completion rates.
- **Improved Data Quality:** Interactive controls like sliders and date pickers reduce errors and capture precise data.
- **Streamlined Workflows:** Conditional logic helps personalize the form experience and guide users efficiently.
- **Increased Accessibility:** Features like image uploaders and location pickers cater to diverse user needs.

Key Principles for designing Interactive Forms:

- **Identify your goals:** What data do you want to collect? How will you use it?
- **Target your audience:** Consider their comfort level with technology and specific needs.
- **Keep it simple:** Use clear instructions and avoid overwhelming users with complex widgets.
- **Mobile-friendliness:** Ensure your form adapts seamlessly to different devices.
- **Accessibility:** Make your form usable for everyone, regardless of abilities.

Popular Platforms for building Interactive Forms:

- **Typeform:** Drag-and-drop builder with various widgets and customizable themes.
- **Google Forms:** Easy-to-use platform with basic widgets and integration with other Google services.
- **JotForm:** Wide range of widgets, conditional logic, and advanced customization options.
- **Zoho Forms:** Extensive features, including payment integrations and data analysis tools.

Beyond the Theory:

Remember, theory is just the foundation. To fully grasp interactive forms, consider:

- **Exploring real-world examples:** Look at forms from different industries to see how they use widgets effectively.
- **Trying out different platforms:** Test drive various form builders to discover what suits your needs.
- **Gathering user feedback:** Pay attention to user experience and make adjustments based on their input.

By understanding the theory and exploring the practical aspects, you can create interactive forms that captivate your audience and collect valuable data effortlessly.

Code :

```
class DestinationPage extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   TextEditingController pickupController = TextEditingController();
```

```
  TextEditingController destinationController = TextEditingController();

  List<String> recommendedPlaces = [
    'CSM International Airport',
    'Panvel Railway Station',
    'Bandra Terminus',
    'Chembur Colony',
    'Vashi Central Park',
    'Seawoods Grand Central',
    'Hakkasan Mumbai',
    'Juhu Beach',
    'Gateway Of India',
    'CSMT Station',
  ];

  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();

  return Scaffold(
    appBar: AppBar(
      title: Text('Pickup and Drop Destination'),
      backgroundColor: const Color.fromARGB(
          255, 168, 249, 171), // Set app bar background color
    ),
    body: SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Form(
          key: _formKey,
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text(
                'Enter Locations',
                style: TextStyle(
                  fontSize: 24.0,
                  fontWeight: FontWeight.bold,
                  color: Color.fromARGB(255, 0, 129, 4), // Set text color
                ),
              ),
              SizedBox(height: 20.0),
              TextFormField(
                controller: pickupController,
                decoration: InputDecoration(
                  labelText: 'Enter Pickup Spot',
```

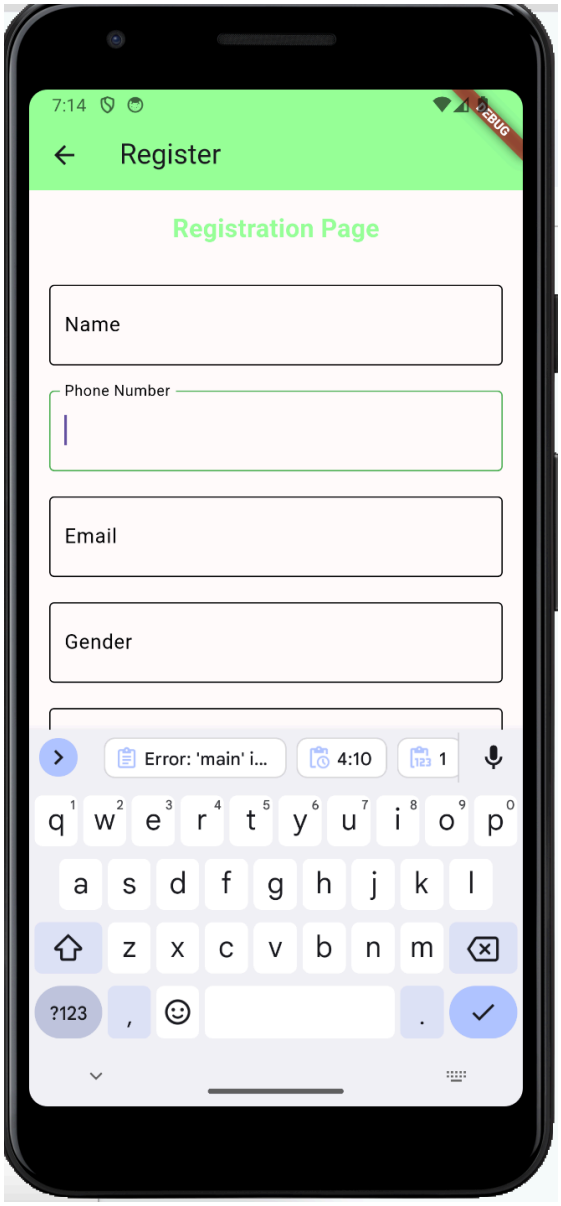
```
    border: OutlineInputBorder(),
    filled: true,
    fillColor: Colors.white, // Set text field background color
   ),
   validator: (value) {
    if (value == null || value.isEmpty) {
     return 'Please enter pickup spot';
    }
    return null;
   },
  ),
  SizedBox(height: 16.0),
  TextFormField(
   controller: destinationController,
   decoration: InputDecoration(
    labelText: 'Enter Destination',
    border: OutlineInputBorder(),
    filled: true,
    fillColor: Colors.white, // Set text field background color
   ),
   validator: (value) {
    if (value == null || value.isEmpty) {
     return 'Please enter destination';
    }
    return null;
   },
  ),
  SizedBox(height: 16.0),
  ElevatedButton(
   onPressed: () {
    if (_formKey.currentState?.validate() ?? false) {
     Navigator.push(
      context,
      MaterialPageRoute(
       builder: (context) => DriverListPage(
        pickupSpot: pickupController.text,
        destination: destinationController.text,
       ),
      ),
     );
    } else {
     ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
       content:
```

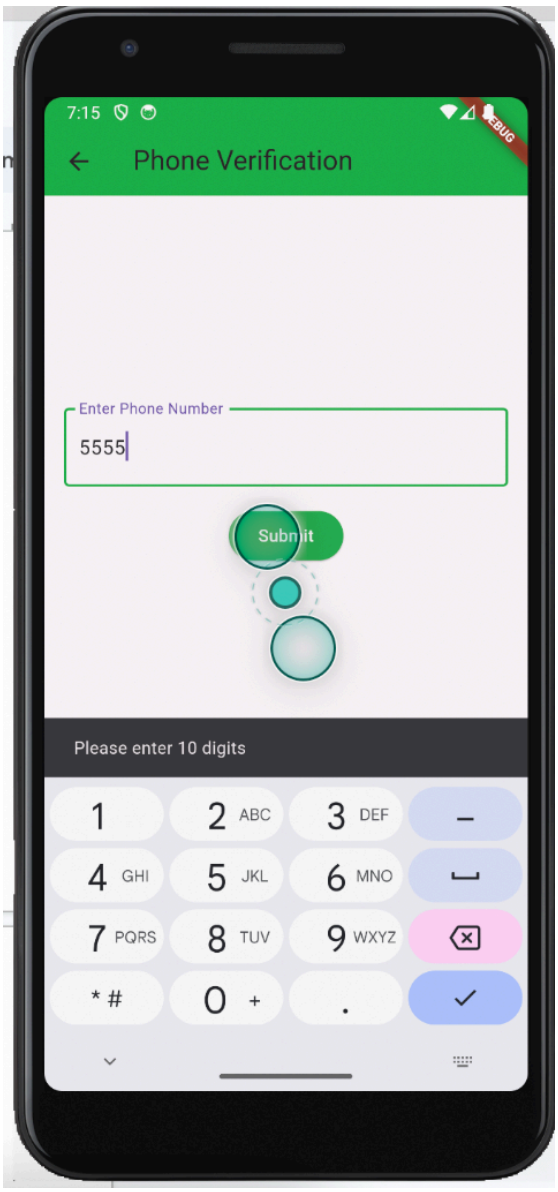
```
            Text('Please fill the required information.'),
          backgroundColor:
            Colors.red, // Set snackbar background color
        ),
       );
    }
  },
  style: ElevatedButton.styleFrom(
    primary: Colors.green, // Set button background color
  ),
  child: Text(
    'Search',
    style:
      TextStyle(color: Colors.white), // Set button text color
  ),
),
SizedBox(height: 20.0),
Text(
  'Popular Places:',
  style: TextStyle(
    fontSize: 18.0,
    fontWeight: FontWeight.bold,
    color: Colors.green, // Set text color
  ),
),
SizedBox(height: 10.0),
Container(
  height: 150.0,
  child: ListView.builder(
    scrollDirection: Axis.vertical,
    itemCount: recommendedPlaces.length,
    itemBuilder: (context, index) {
      return Padding(
        padding: const EdgeInsets.all(8.0),
        child: ElevatedButton(
          onPressed: () {
            pickupController.text = recommendedPlaces[index];
          },
          style: ElevatedButton.styleFrom(
            primary:
              Colors.black, // Set button background color
          ),
          child: Text(
            recommendedPlaces[index],
```

```
                        style: TextStyle(
                            color: Colors.white), // Set button text color
                      ),
                    ),
                  );
                },
              ),
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

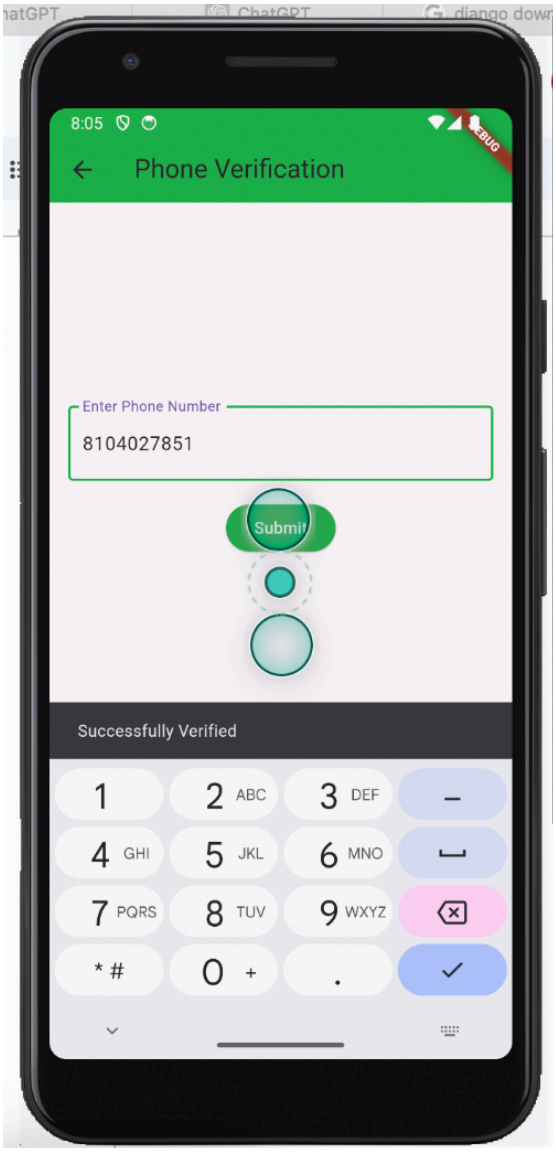
Screenshots :

Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**

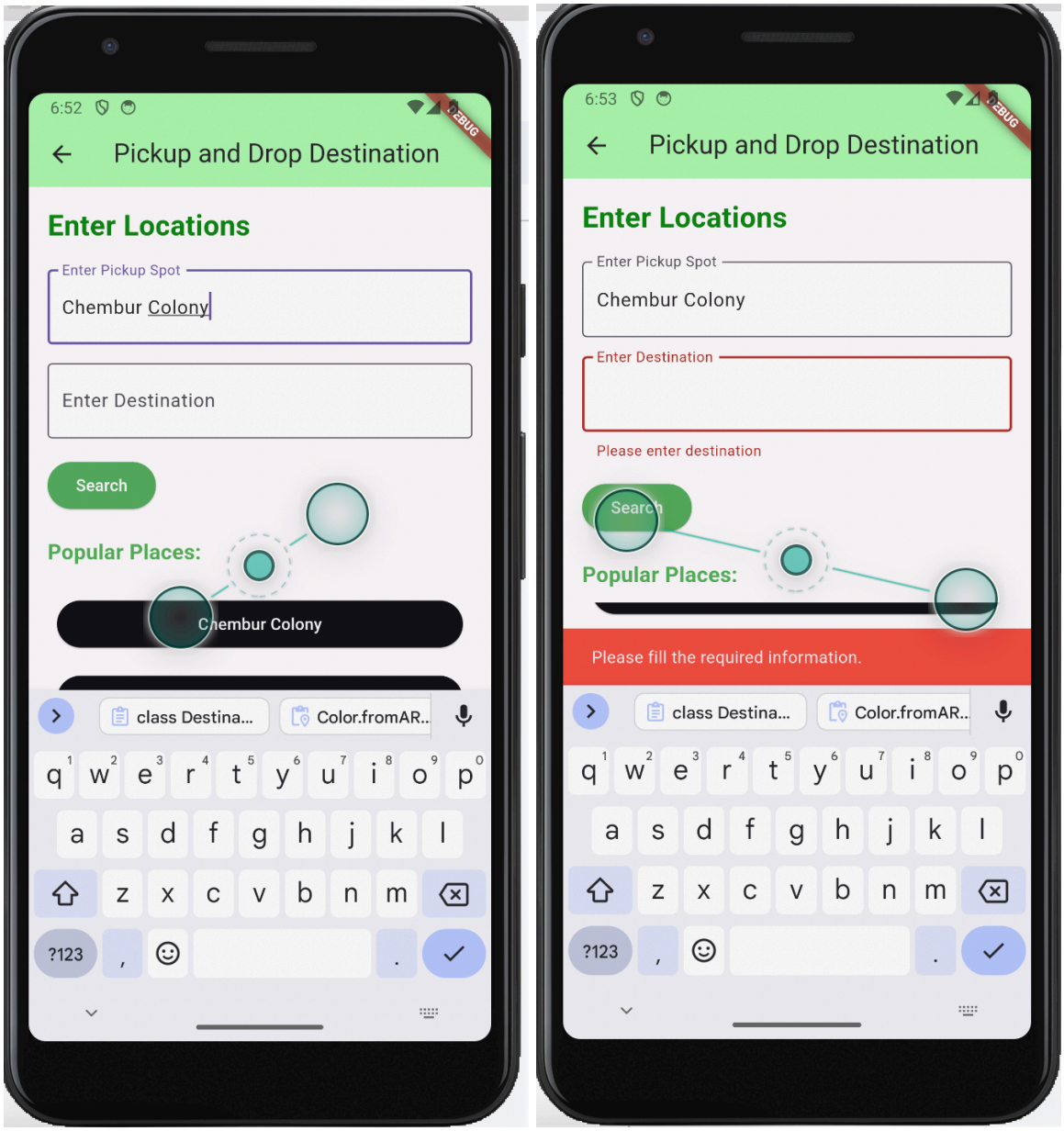




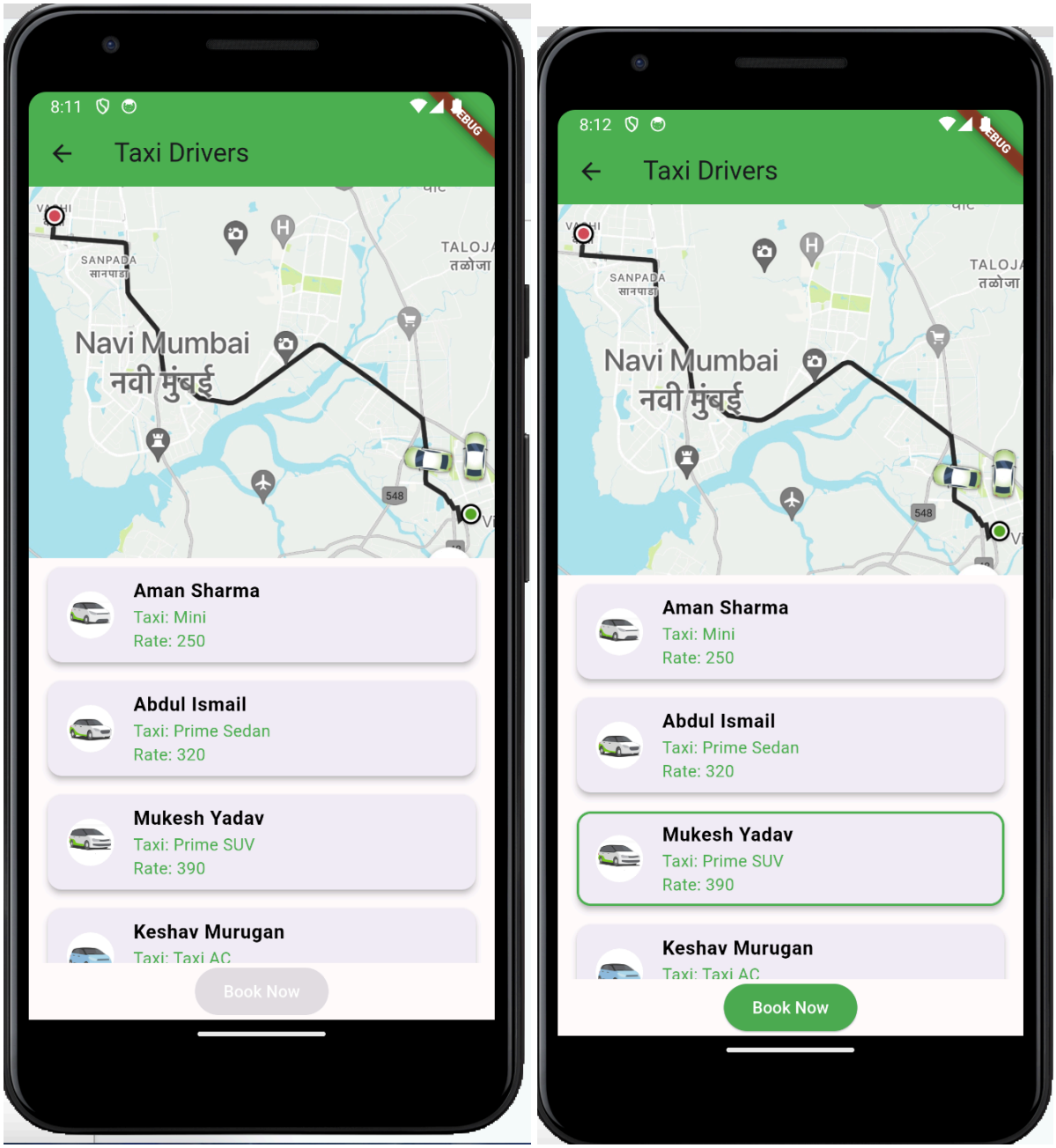
Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**

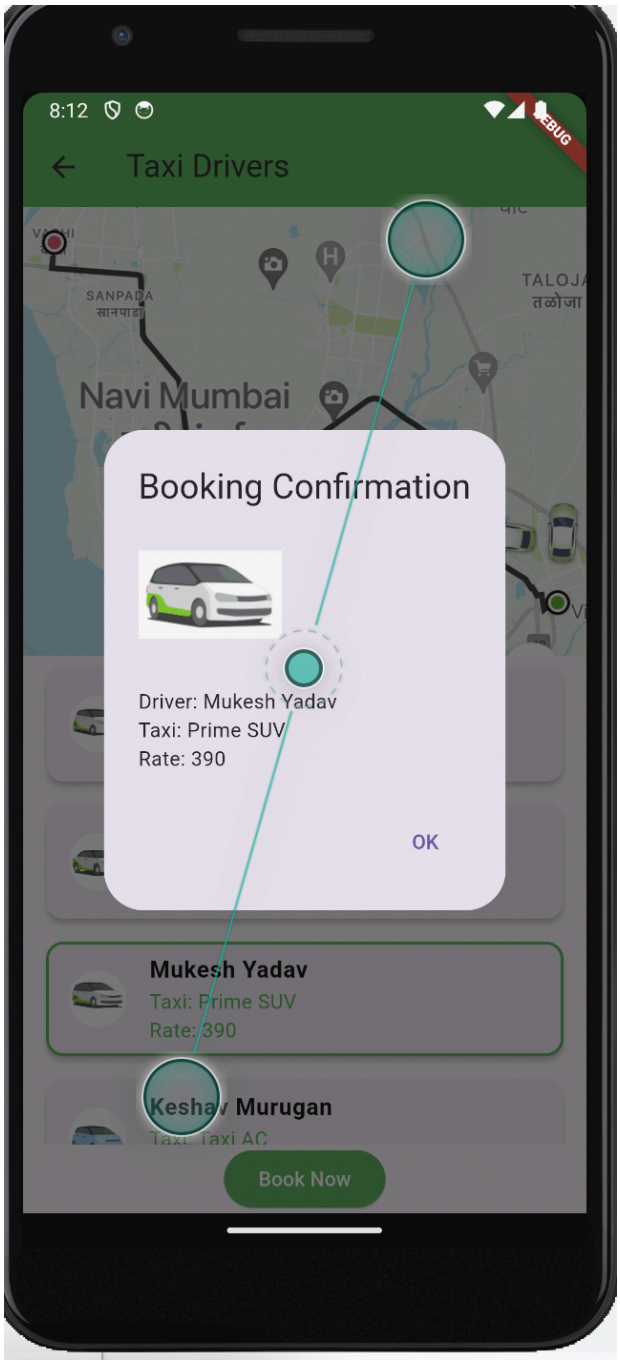


Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**



Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**





CONCLUSION :

Interactive forms with widgets: Boost data collection and user experience by leveraging pre-built features like sliders, maps, and conditional logic. Remember, user-friendliness, accessibility, and clear goals are key for success.

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

NAME : SOHAM SHETYE ROLL NO. : 54
DIV : D15A BATCH : C

MAD PWA LAB

LAB 5

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

THEORY :

Flutter provides robust tools for crafting intuitive and user-friendly app navigation experiences. Here's a theoretical breakdown of the key concepts:

Navigation:

- Refers to the app's structure and how users move between different screens.
- Involves concepts like hierarchical navigation (think nested screens) and bottom navigation bars.

Routing:

- Defines the mechanism for transitioning between screens, handling the "how" of navigation.
- Utilizes widgets like Navigator and methods like push and pop to manage the navigation stack.
- Can use named routes for clarity and easier maintenance.

Gestures:

- Provide ways for users to interact with the app interface and trigger navigation actions.
- Common gestures include taps, swipes, drags, and long presses.
- Flutter's gesture recognition system allows you to define custom gestures for specific behaviors.

Understanding the interplay:

- Gestures trigger routing actions managed by the Navigator.
- Navigation decisions define the app's flow and screen transitions.
- Consider:

- Use relevant gestures based on platform conventions and user expectations.
- Design smooth and intuitive transitions between screens.
- Implement accessibility features for diverse user needs.

Further Exploration:

- Dive deeper into the official Flutter documentation for navigation and routing:https://docs.flutter.dev/ui/navigation
- Explore gesture recognition in Flutter: <invalid URL removed>
- Experiment with different navigation patterns and gesture interactions to find the perfect fit for your app.

By actively exploring and practicing these concepts, we"ll create a truly engaging and user-friendly navigation experience for your Flutter app.

CODE :

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'OLA Taxi Booking App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: LoginPage(),
      routes: {
        '/home': (context) => HomePage(),
        '/destination': (context) => DestinationPage(),
        '/account': (context) => AccountPage(),
        '/register': (context) => RegisterPage(),
        '/phone_verification': (context) => PhoneVerificationPage(),
      },
    );
```

```
  }
}

class LoginPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('OLA Account Login'),
      ),
      body: GestureDetector(
        onTap: () {
          // Dismiss the keyboard when tapping outside the text field
          FocusScope.of(context).unfocus();
        },
        child: Container(
          decoration: BoxDecoration(
            image: DecorationImage(
              image: AssetImage('assets/images/projects.png'),
              fit: BoxFit.cover,
            ),
          ),
          child: Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Expanded(
                  child: SizedBox(),
                ),
                Text(
                  'Login Page',
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: 24,
                    fontWeight: FontWeight.bold,
                  ),
                ),
                SizedBox(height: 20),
                ElevatedButton(
                  onPressed: () {
                    Navigator.pushNamed(context, '/home');
```

```
            },
            style: ElevatedButton.styleFrom(
              primary: Colors.green,
              onPrimary: Colors.white,
            ),
            child: Padding(
              padding: EdgeInsets.symmetric(horizontal: 20),
              child: Text(
                'Login',
                style: TextStyle(fontSize: 18),
              ),
            ),
          ),
          SizedBox(height: 10),
          TextButton(
            onPressed: () {
              Navigator.pushNamed(context, '/register');
            },
            style: TextButton.styleFrom(
              primary: Color.fromARGB(255, 5, 185, 29),
            ),
            child: Text(
              'Register',
              style: TextStyle(
                fontSize: 16,
                decoration: TextDecoration.underline,
              ),
            ),
          ),
        ],
      ),
    ),
  ),
 );
 }
}

class RegisterPage extends StatefulWidget {
 @override
 _RegisterPageState createState() => _RegisterPageState();
```

```
}

class _RegisterPageState extends State<RegisterPage> {
  TextEditingController nameController = TextEditingController();
  TextEditingController phoneNumberController = TextEditingController();
  TextEditingController emailController = TextEditingController();
  TextEditingController genderController = TextEditingController();
  TextEditingController addressController = TextEditingController();
  TextEditingController ageController = TextEditingController();

  @override
  Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
      title: Text('Register'),
      backgroundColor: const Color.fromARGB(255, 150, 255, 154),
     ),
     body: GestureDetector(
      onTap: () {
       // Dismiss the keyboard when tapping outside the text field
       FocusScope.of(context).unfocus();
      },
      child: SingleChildScrollView(
       child: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
         mainAxisAlignment: MainAxisAlignment.center,
         children: [
          Text(
           'Registration Page',
           style: TextStyle(
            fontSize: 20.0,
            fontWeight: FontWeight.bold,
            color: const Color.fromARGB(255, 150, 255, 154),
           ),
          ),
          SizedBox(height: 20.0),
          buildTextField(nameController, 'Name'),
          buildTextField(phoneNumberController, 'Phone Number'),
          buildTextField(emailController, 'Email'),
          buildTextField(genderController, 'Gender'),
```

```
            buildTextField(addressController, 'Address'),
            buildTextField(ageController, 'Age'),
            SizedBox(height: 20.0),
            ElevatedButton(
              onPressed: () {
                Navigator.pushNamed(context, '/phone_verification');
              },
              style: ElevatedButton.styleFrom(
                primary: Color.fromARGB(255, 155, 255, 255, 159),
              ),
              child: Text('Continue to Phone Verification'),
            ),
            SizedBox(height: 10.0),
            TextButton(
              onPressed: () {
                Navigator.pop(context); // Navigate back to the previous screen
              },
              style: TextButton.styleFrom(
                primary: Colors.white,
              ),
              child: Text(
                'Back',
                style: TextStyle(
                  fontSize: 16,
                  decoration: TextDecoration.underline,
                ),
              ),
            ),
          ],
        ),
      ),
    ),
  ),
 );
}

Widget buildTextField(TextEditingController controller, String labelText) {
 return Padding(
   padding: const EdgeInsets.symmetric(vertical: 10.0),
   child: TextField(
     controller: controller,
```

```
      decoration: InputDecoration(
       labelText: labelText,
       labelStyle: TextStyle(color: Color.fromARGB(255, 0, 0, 0)),
       focusedBorder: OutlineInputBorder(
        borderSide: BorderSide(color: Colors.green),
       ),
       enabledBorder: OutlineInputBorder(
        borderSide: BorderSide(color: Colors.black),
       ),
      ),
     ),
    );
  }
}

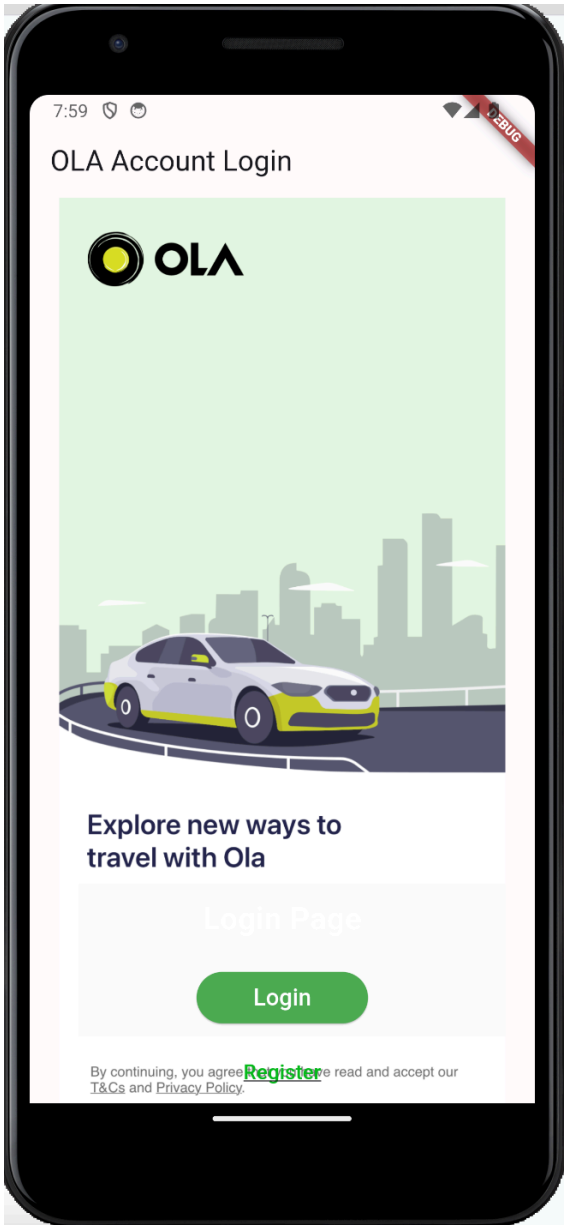
class PhoneVerificationPage extends StatelessWidget {
 final TextEditingController _phoneNumberController = TextEditingController();

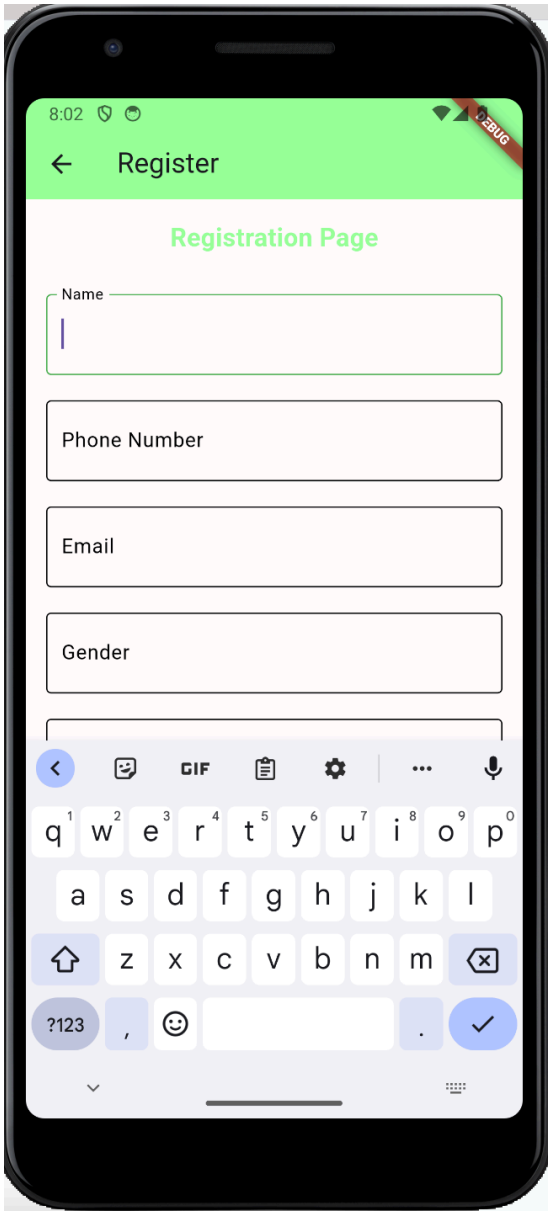
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('Phone Verification'),
    backgroundColor: Colors.green,
   ),
   body: GestureDetector(
    onTap: () {
     // Dismiss the keyboard when tapping outside the text field
     FocusScope.of(context).unfocus();
    },
    child: Padding(
     padding: const EdgeInsets.all(16.0),
     child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
       TextField(
        controller: _phoneNumberController,
        keyboardType: TextInputType.phone,
        maxLength: 10,
        decoration: InputDecoration(
         labelText: 'Enter Phone Number',
```

```
            hintText: 'e.g., 1234567890',
            counterText: '',
            enabledBorder: OutlineInputBorder(
              borderSide: BorderSide(color: Colors.green),
            ),
            focusedBorder: OutlineInputBorder(
              borderSide: BorderSide(color: Colors.green, width: 2.0),
            ),
          ),
        ),
        SizedBox(height: 16.0),
        ElevatedButton(
          onPressed: () {
            if (_phoneNumberController.text.length == 10) {
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(
                  content: Text('Successfully Verified'),
                  duration: Duration(seconds: 2),
                ),
              );
            } else {
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(
                  content: Text('Please enter 10 digits'),
                  duration: Duration(seconds: 2),
                ),
              );
            }
          },
          style: ElevatedButton.styleFrom(
            primary: Colors.green,
            onPrimary: Colors.white,
          ),
          child: Text('Submit'),
        ),
      ],
    ),
  ),
 );
}
```

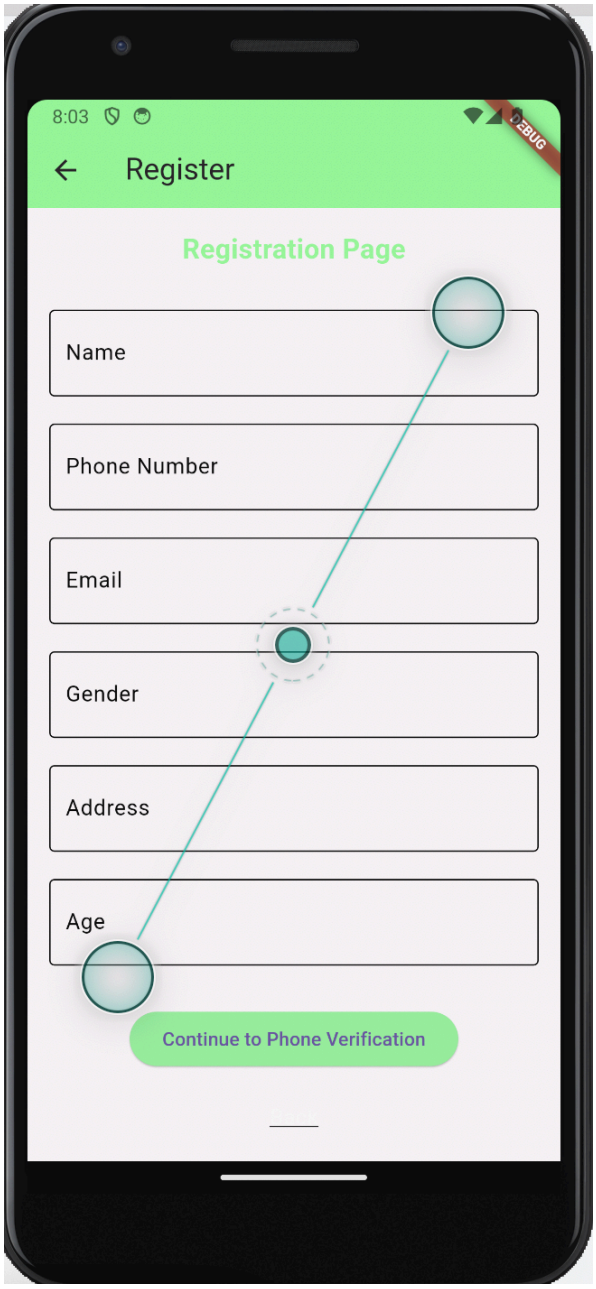
}

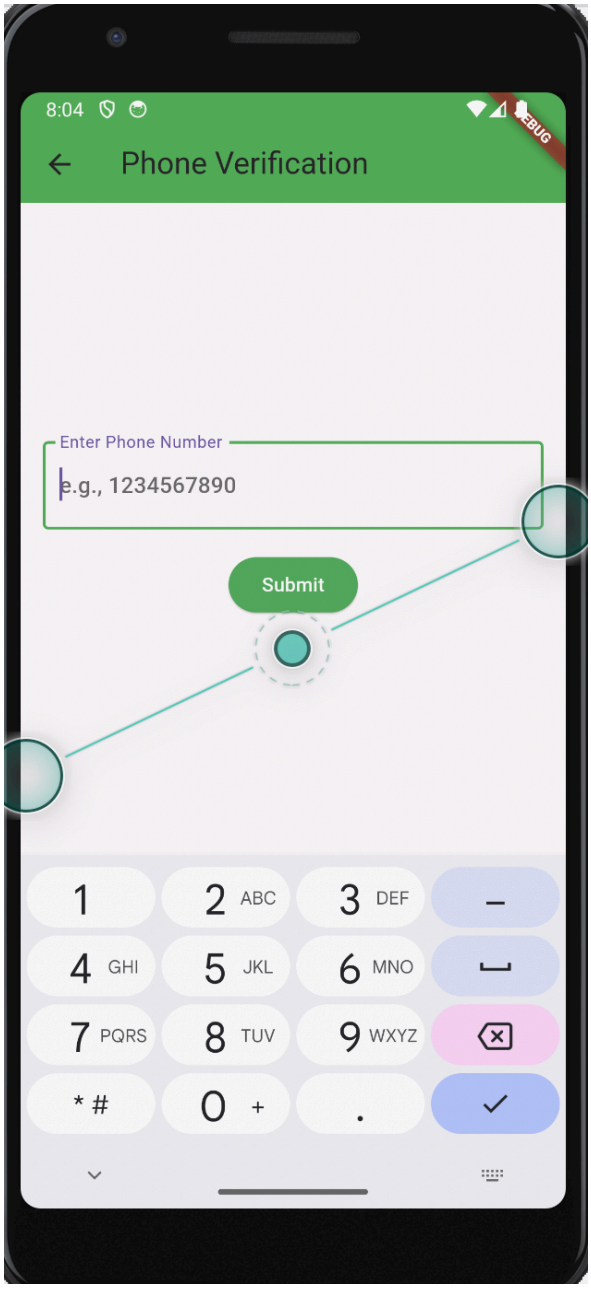
Screenshots :



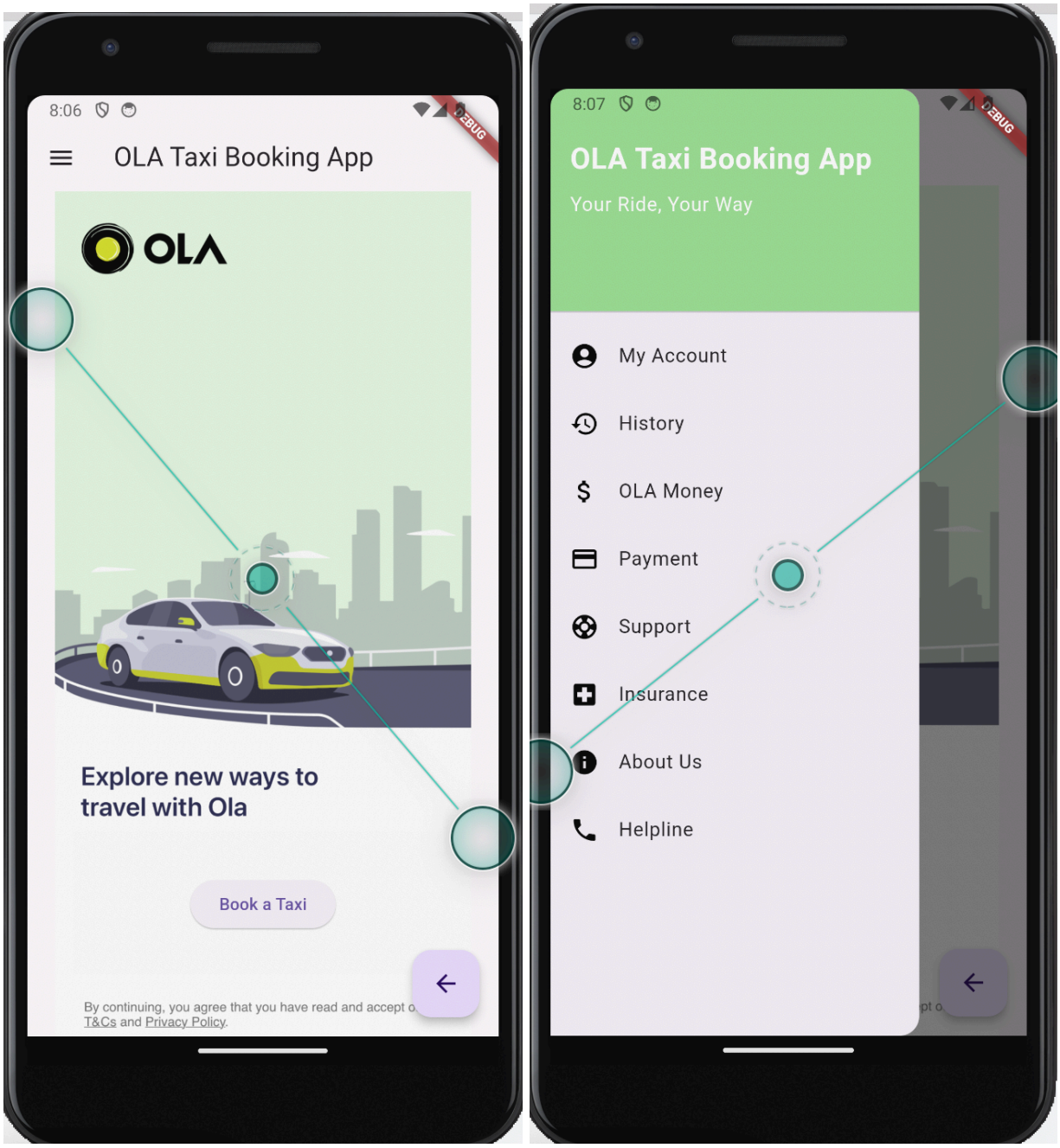


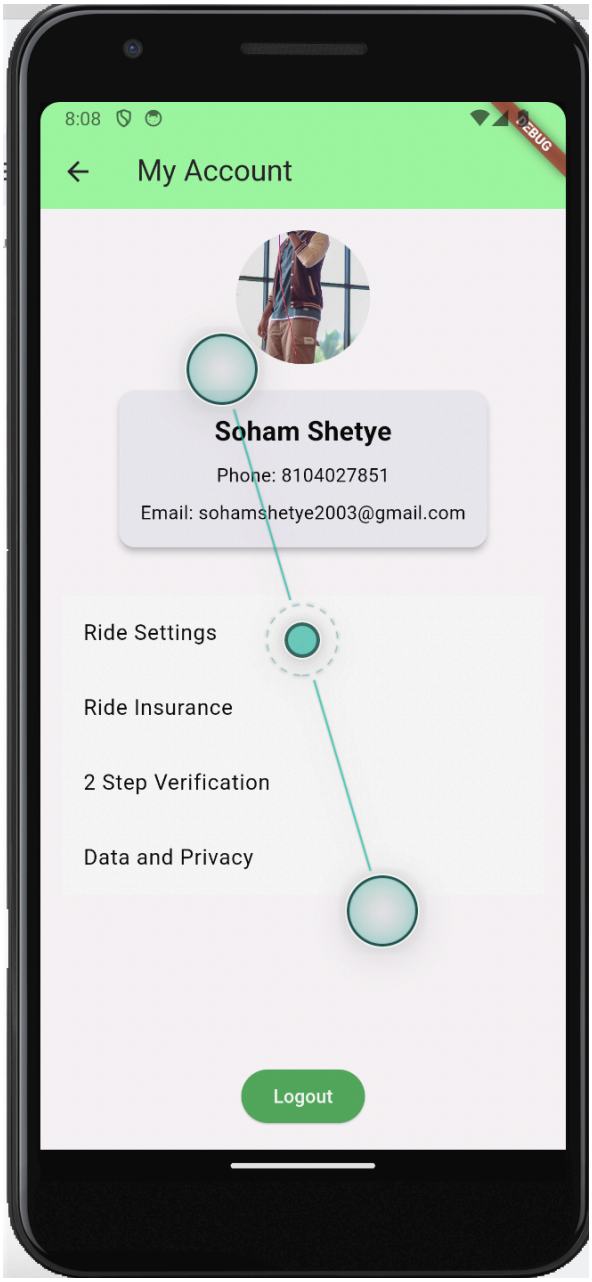
Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**





Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**





Navigation and Routing:

Main Route Configuration (MyApp):

- The `MaterialApp` widget is used to define the main navigation configuration.
- The `routes` property is used to define named routes and their corresponding widgets.

Login Page (LoginPage):

- The `ElevatedButton` on the login page is configured to navigate to the `/home` route using `Navigator.pushNamed(context, '/home')`.

Register Page (RegisterPage):

- The `ElevatedButton` on the register page is configured to navigate to the `/phone_verification` route using `Navigator.pushNamed(context, '/phone_verification')`.
- The `TextButton` is configured to navigate back to the previous screen using `Navigator.pop(context)`.

Phone Verification Page (PhoneVerificationPage):

- No explicit navigation in this page, but it is part of the overall route configuration.

Gestures:

Login Page (LoginPage):

- The `GestureDetector` wraps the entire `Container` to handle gestures. The `onTap` callback is used to dismiss the keyboard when tapping outside the text field.

Register Page (RegisterPage):

- The `GestureDetector` wraps the entire `SingleChildScrollView` to handle gestures. The `onTap` callback is used to dismiss the keyboard when tapping outside the text field.

Phone Verification Page (PhoneVerificationPage):

- The `GestureDetector` wraps the entire `Padding` widget to handle gestures. The `onTap` callback is used to dismiss the keyboard when tapping outside the text field.

CONCLUSION :

Navigation and routing are used when transitioning between different pages using named routes (Navigator.pushNamed). Gestures are used to dismiss the keyboard when tapping outside text fields (GestureDetector with onTap callback).

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	15

NAME : SOHAM SHETYE ROLL NO. : 54
DIV : D15A BATCH : C

MAD PWA LAB

LAB 6

Aim: To connect flutter UI with firebase database

THEORY :

Using Firebase with Flutter provides several advantages, making it a popular choice for mobile app development. Here are some reasons why Firebase is often chosen as a backend solution for Flutter applications:

Real-Time Database:

- Firebase provides a real-time NoSQL database that allows seamless data synchronization across devices. This is particularly useful for apps that require live updates, such as messaging apps, collaborative tools, or any application where real-time data is crucial.

Authentication:

- Firebase Authentication simplifies the process of user authentication with various sign-in methods, including email/password, Google Sign-In, Facebook Login, and more. Integrating Firebase Authentication with Flutter is straightforward and provides a secure way to manage user identities.

Cloud Functions:

- Firebase allows you to deploy serverless functions in the cloud using Cloud Functions for Firebase. This enables you to run backend code in response to events triggered by changes in the database, authentication, or other cloud services. These functions can be written in JavaScript, TypeScript, or Dart.

Cloud Storage:

- Firebase offers Cloud Storage for storing and serving user-generated content, such as images, videos, and other files. Integrating Cloud Storage with Flutter enables efficient handling of multimedia content without the need for a separate server.

Firebase Hosting:

- Firebase Hosting provides a simple and secure way to deploy web apps. If you're building a Flutter web application, Firebase Hosting can be used to deploy and serve your app, ensuring a reliable and scalable hosting solution.

Analytics and Crash Reporting:

- Firebase Analytics helps you understand user behavior, track app performance, and gain insights into how users interact with your app. Firebase Crashlytics provides real-time crash reporting, allowing you to identify and address issues quickly.

Cloud Firestore:

- While the real-time database is a key feature, Firebase also offers Cloud Firestore, a more powerful and scalable NoSQL database. Firestore provides richer query capabilities, hierarchical data structures, and better support for large-scale applications compared to the original real-time database.

Easy Integration with Flutter:

- Google provides official plugins for Flutter that make integrating Firebase services seamless. These plugins are well-maintained, well-documented, and designed to work seamlessly with Flutter apps.

Scalability and Reliability:

- Firebase is a fully managed platform, ensuring scalability and reliability without the need for developers to manage servers or infrastructure. This allows developers to focus more on building features and less on managing backend services.

Community Support:

- Due to its popularity, there is a vibrant community around both Flutter and Firebase. This means a wealth of resources, tutorials, and community support are available, making it easier for developers to find help when needed.

CODE :-

```
class LoginPage extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: Text('Login Page'),
     ),
     body: Stack(
       fit: StackFit.expand,
       children: [
         // Background Image
         Image.asset(
           'assets/images/projects.png', // Replace with your image asset path
           fit: BoxFit.cover,
         ),
         Column(
           mainAxisAlignment: MainAxisAlignment.center,
```

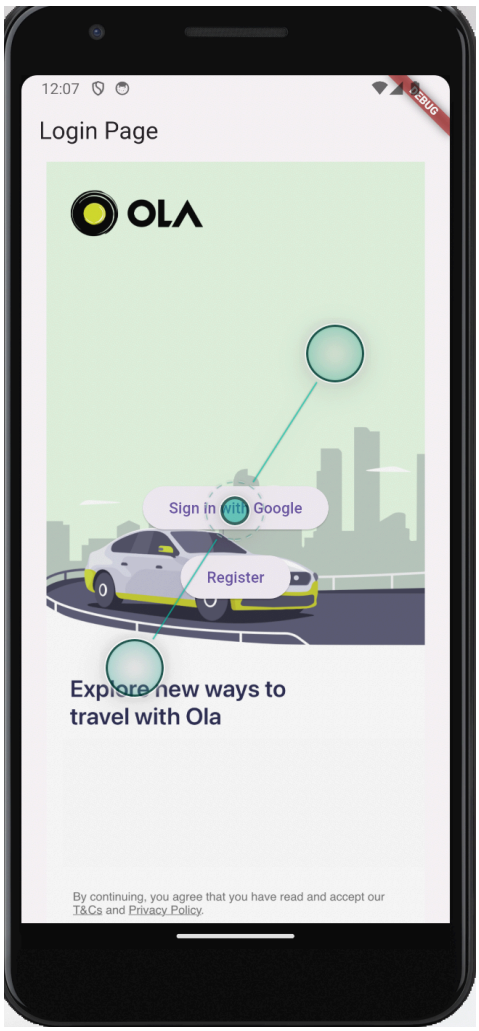
```
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            ElevatedButton(
              onPressed: () async {
                // Sign out the current Google user
                await GoogleSignIn().signOut();

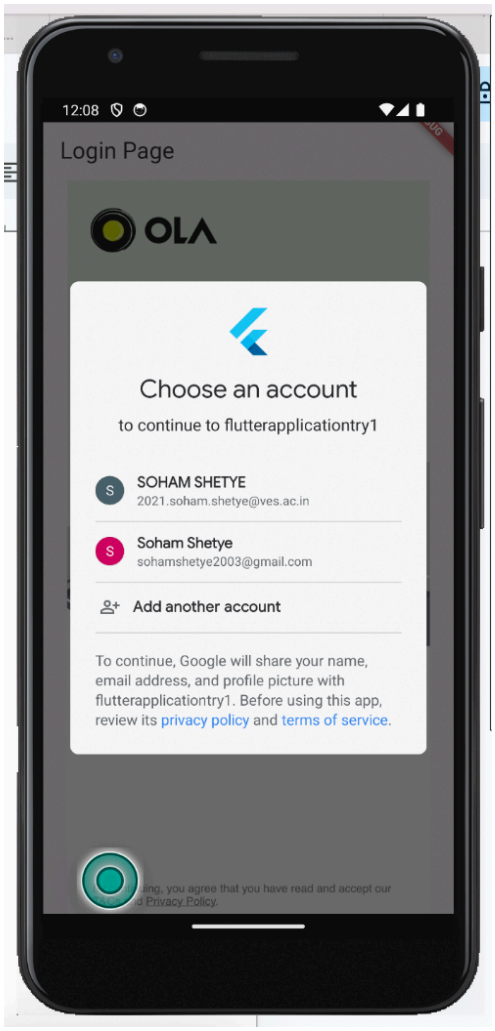
                // Open the Google Sign-In page
                await signInWithGoogle(context);
              },
              child: Text('Sign in with Google'),
            ),
            SizedBox(height: 16.0), // Add some spacing
            ElevatedButton(
              onPressed: () {
                // Navigate to the RegisterPage
                Navigator.pushNamed(context, '/register');
              },
              child: Text('Register'),
            ),
          ],
        ),
      ],
    ),
  );
 }
}

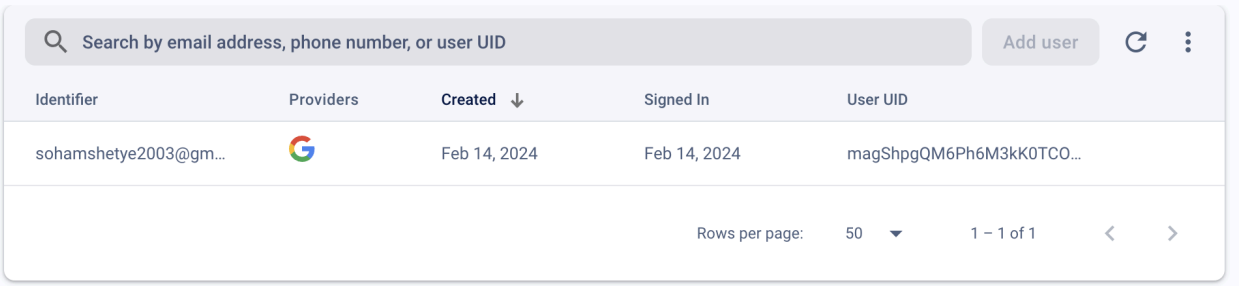
Future<void> signInWithGoogle(BuildContext context) async {
 try {
  final GoogleSignInAccount? googleUser = await GoogleSignIn().signIn();
  if (googleUser == null) {
    // User canceled the sign-in
    return;
  }

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

```
    // Navigate to the home page after successful sign-in
    Navigator.pushReplacementNamed(context, '/home');
  } catch (e) {
    print('Error signing in with Google: $e');
  }
}
```








```
class DriverListPage extends StatefulWidget {
```

```
 final String pickupSpot;
 final String destination;

 DriverListPage({required this.pickupSpot, required this.destination});

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

class _DriverListPageState extends State<DriverListPage> {
 int? selectedDriverIndex;
 List<TaxiDriver> taxiDrivers = [];

 @override
 void initState() {
  super.initState();
  // Fetch data from Firestore when the widget is initialized
  fetchData();
 }

 Future<void> fetchData() async {
  try {
    QuerySnapshot<Map<String, dynamic>> querySnapshot =
        await FirebaseFirestore.instance.collection('taxiDrivers').get();

    setState(() {
      taxiDrivers = querySnapshot.docs
          .map((doc) => TaxiDriver.fromMap(doc.data() as Map<String, dynamic>))
          .toList();
    });
  } catch (e) {
    print('Error fetching data: $e');
  }
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Taxi Drivers'),
      backgroundColor: Colors.green,
    ),
    body: Column(
      children: [
```

```
Container(
  height: MediaQuery.of(context).size.height * 0.4,
  decoration: BoxDecoration(
    image: DecorationImage(
      image: AssetImage("assets/images/map.jpg"),
      fit: BoxFit.cover,
    ),
  ),
),
Expanded(
  child: taxiDrivers.isEmpty
      ? Center(
          child: CircularProgressIndicator(),
        )
      : ListView.builder(
          itemCount: taxiDrivers.length,
          itemBuilder: (context, index) {
            return GestureDetector(
              onTap: () {
                setState(() {
                  selectedDriverIndex = index;
                });
              },
              child: Card(
                elevation: 4.0,
                margin: EdgeInsets.symmetric(
                    vertical: 8.0, horizontal: 16.0),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(12.0),
                  side: BorderSide(
                    color: selectedDriverIndex == index
                        ? Colors.green
                        : Colors.transparent,
                    width: 2.0,
                  ),
                ),
                child: ListTile(
                  title: Text(
                    taxiDrivers[index].name,
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Colors.black,
                    ),
                  ),
```

```
                subtitle: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      'Taxi: ${taxiDrivers[index].taxiName}',
                      style: TextStyle(
                        color: Colors.green,
                      ),
                    ),
                    Text(
                      'Rate: ${taxiDrivers[index].rate}',
                      style: TextStyle(
                        color: Colors.green,
                      ),
                    ),
                  ],
                ),
                leading: CircleAvatar(
                  backgroundImage:
                      AssetImage(taxiDrivers[index].carPhoto),
                  backgroundColor: Colors.white,
                ),
              ),
            ),
          );
        },
      ),
),
ElevatedButton(
  onPressed: selectedDriverIndex != null
      ? () {
          // Implement booking logic here
          // You can use taxiDrivers[selectedDriverIndex] to get the selected driver details
          // For example: taxiDrivers[selectedDriverIndex].name
          showDialog(
            context: context,
            builder: (context) => AlertDialog(
              title: Text('Booking Confirmation'),
              content: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                mainAxisSize: MainAxisSize.min,
                children: [
                  Image.asset(
                    taxiDrivers[selectedDriverIndex!].carPhoto,
```

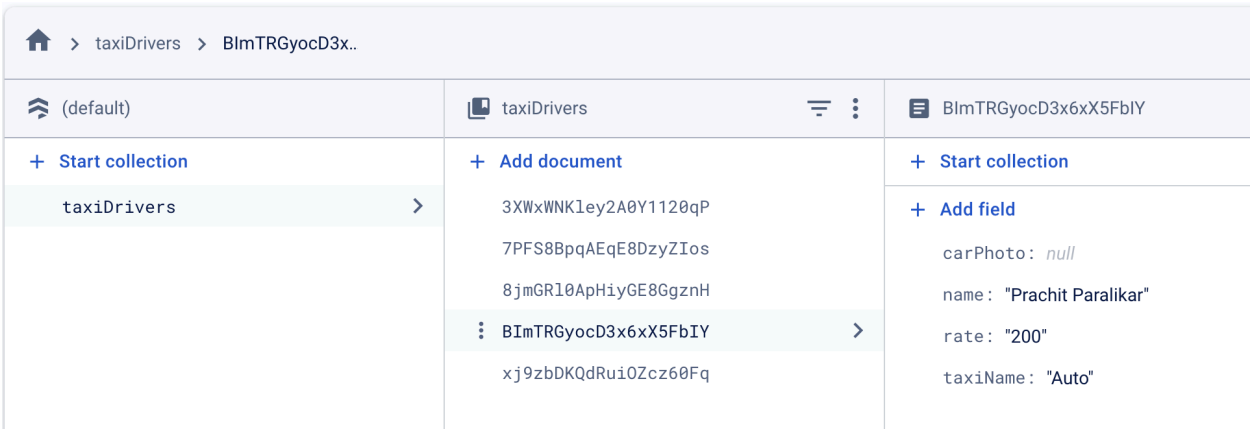
```
                    height: 100,
                    width: 100,
                    fit: BoxFit.cover,
                  ),
                  SizedBox(height: 8),
                  Text(
                     'Driver: ${taxiDrivers[selectedDriverIndex!].name}'),
                  Text(
                     'Taxi: ${taxiDrivers[selectedDriverIndex!].taxiName}'),
                  Text(
                     'Rate: ${taxiDrivers[selectedDriverIndex!].rate}'),
                ],
              ),
              actions: [
                TextButton(
                  onPressed: () {
                    Navigator.pop(context);
                  },
                  child: Text('OK'),
                ),
              ],
            ),
          );
        }
      : null,
    style: ElevatedButton.styleFrom(
      primary: Colors.green,
    ),
    child: Text(
      'Book Now',
      style: TextStyle(color: Colors.white),
    ),
  ),
 ],
 ),
 );
 }
}

class TaxiDriver {
 final String name;
 final String taxiName;
 final String rate;
 final String carPhoto;
```

```
 TaxiDriver({
   required this.name,
   required this.taxiName,
   required this.rate,
   required this.carPhoto,
 });

 factory TaxiDriver.fromMap(Map<String, dynamic> map) {
   print('Mapping Firestore data: $map');
   return TaxiDriver(
     name: map['name'] ?? '',
     taxiName: map['taxiName'] ?? '',
     rate: map['rate'] ?? '',
     carPhoto: map['carPhoto'] ?? '',
   );
 }
}
```

Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**



CONCLUSION :-

Firebase provides a comprehensive set of backend services that complement Flutter's capabilities, making it a convenient and powerful choice for building mobile and web applications with real-time features, authentication, and more.

MAD & PWA Lab
Journal

Experiment No.	07
Experiment Title.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

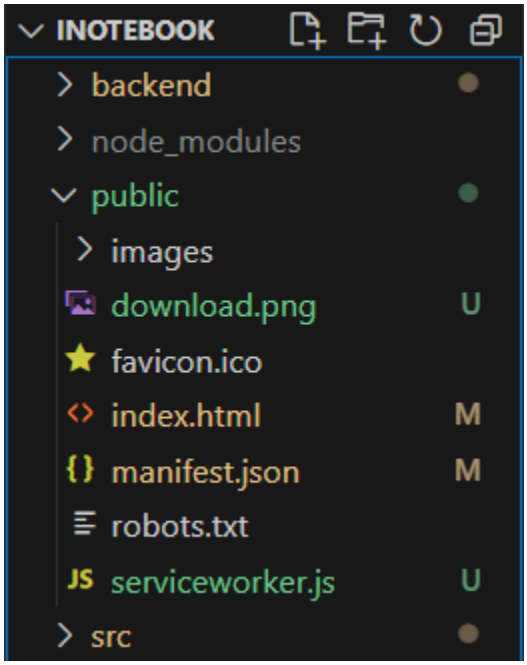
Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**

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

THEORY:

Regular Web App

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



Progressive Web App

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

manifest.json

```
{
   "short_name": "React App",
   "name": "Create React App Sample",
   "icons": [
     {
       "src": "favicon.ico",
       "sizes": "64x64 32x32 24x24 16x16",
       "type": "image/x-icon"
     },
     {
       "src": "logo192.png",
       "type": "image/png",
       "sizes": "192x192"
     },
     {
       "src": "logo512.png",
       "type": "image/png",
       "sizes": "512x512"
     }
   ],
   "start_url": ".",
   "display": "standalone",
   "theme_color": "#000000",
   "background_color": "#ffffff"
 }
```

serviceworker.js

```
var staticCacheName = "pwa";

self.addEventListener("install", function (e) {
e.waitUntil(
   caches.open(staticCacheName).then(function (cache) {
   return cache.addAll(["/"]);
   })
);
});

self.addEventListener("fetch", function (event) {
console.log(event.request.url);

event.respondWith(
   caches.match(event.request).then(function (response) {
   return response || fetch(event.request);
```

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

public/index.html

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

<head>
  <meta charset="utf-8" />
  <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <meta name="theme-color" content="#000000" />
  <meta name="description" content="Web site created using create-react-app" />
  <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
  <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
  <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
  <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->

  <!-- Bootstrap CSS -->
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.rtl.min.css"

integrity="sha384-PJsj/BTMqILvmcej7ulplguok8ag4xFTPryRq8xevL7eBYSmpXKcbNVuy+P0
RMgq" crossorigin="anonymous">

  <title>iNoteBook</title>
</head>
```

```
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
     This HTML file is a template.
     If you open it directly in the browser, you will see an empty page.

     You can add webfonts, meta tags, or analytics to this file.
     The build step will place the bundled scripts into the <body> tag.

     To begin the development, run `npm start` or `yarn start`.
     To create a production bundle, use `npm run build` or `yarn build`.
    -->

  <!-- Option 1: Bootstrap Bundle with Popper -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"

integrity="sha384-geWF76RCwLtnZ8qwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy
3Te4Bkz"
    crossorigin="anonymous"></script>

    <script src="https://kit.fontawesome.com/e7a7193d30.js" crossorigin="anonymous"></script>

    <script>
        window.addEventListener("load", () => {
          registerSW();
        });

        // Register the Service Worker
        async function registerSW() {
          if ("serviceWorker" in navigator) {
            try {
              await navigator;
              ServiceWorker;
              register("serviceworker.js");
            } catch (e) {
              console.log("SW Registration Failed.");
            }
          }
        }
```

```
    </script>
</body>
</html>
```

COMMAND: **npm run start**

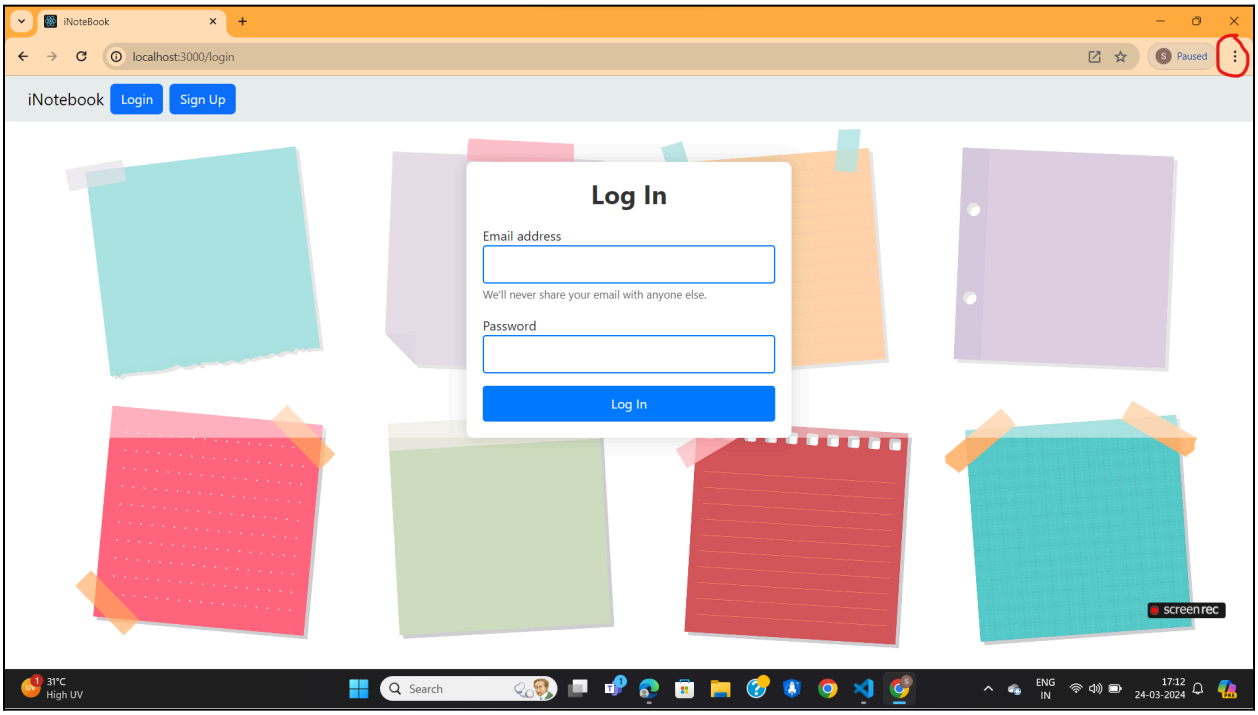
```
You can now view inotebook in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.104:3000

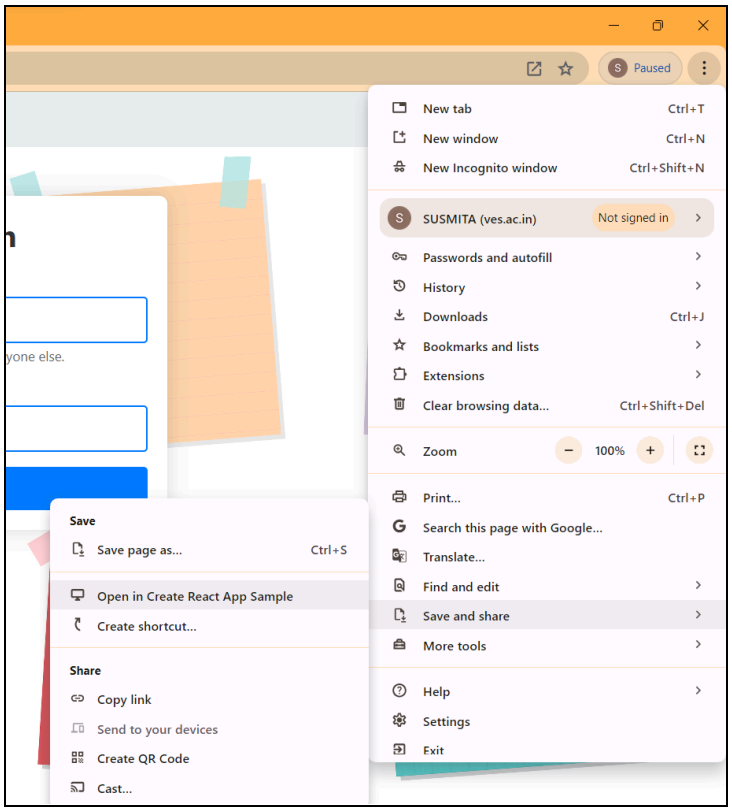
Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

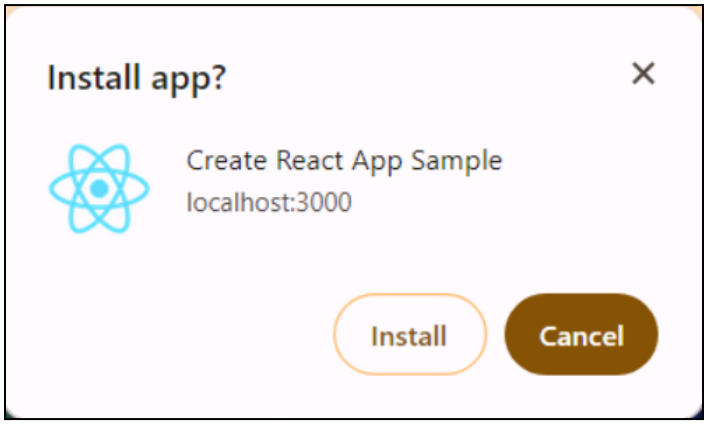
Open localhost:3000 on the browser



Click on 3 dots on the top right corner of the browser → go to **Save and Share** → Click on **Install Create React App Sample**



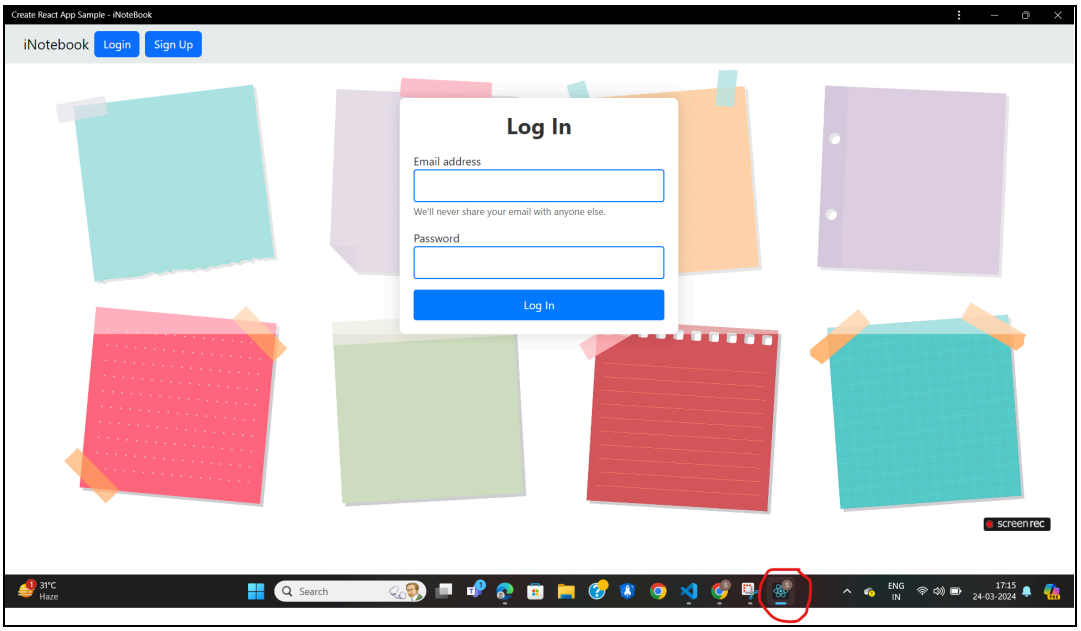
You will get the following pop-up:



Click on **Install**.

App Icon will appear at the bottom:

Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**



CONCLUSION:

Hence, we learnt how to write a metadata of our E-commerce website PWA in a Web App Manifest File to enable add to homescreen feature.

MAD & PWA Lab
Journal

Experiment No.	08
Experiment Title.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**

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

Theory:

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

Things to note about Service Worker:

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

What can we do with Service Workers?

- You can dominate **Network Traffic**
  You can manage all network traffic of the page and do any manipulations. For example, when the page requests a CSS file, you can send plain text as a response or when the page requests an HTML file, you can send a png file as a response. You can also send a true response too.

- You can **Cache**
  You can cache any request/response pair with Service Worker and Cache API and you can access these offline content anytime.

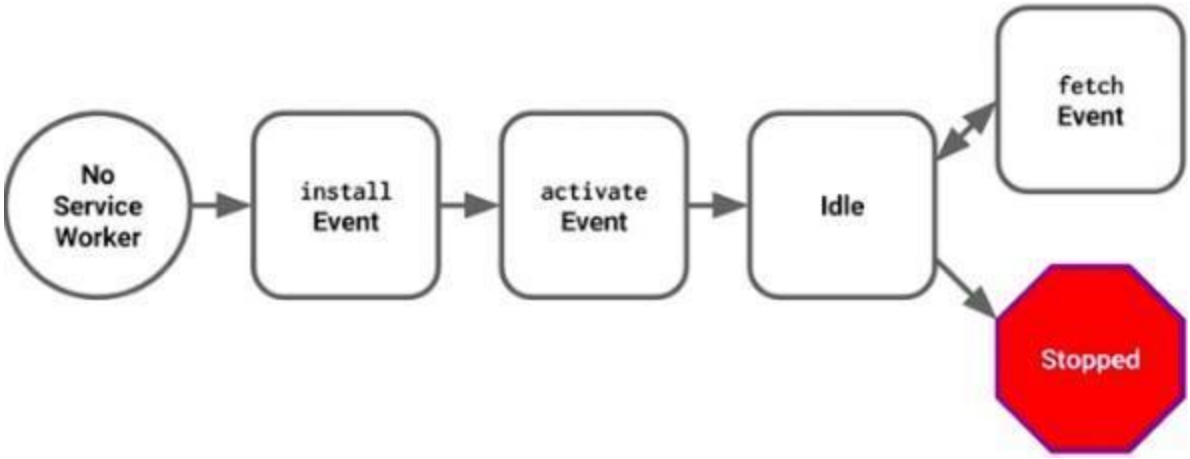
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can’t we do with Service Workers?

- You can't access the **Window**
  You can't access the window, therefore, You can't manipulate DOM elements. But, you can communicate to the window through post Message and manage processes that you want.

- You can't work it on **80 Port**
  Service Worker just can work on HTTPS protocol. But you can work on localhost during development.

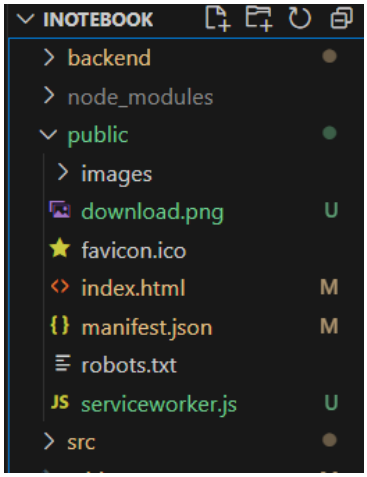
Service Worker Cycle



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

- Registration
- Installation
- Activation

My Project's folder structure:



Registration

To install a service worker, you need to register it in your main JavaScript code. Registration tells the browser where your service worker is located, and to start installing it in the background.

CODE:

public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->

    <!-- Bootstrap CSS -->
    <link
      rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.rtl.min.css"

integrity="sha384-PJsj/BTMqILvmcej7ulplguok8ag4xFTPryRq8xevL7eBYSmpXKcbNVuy+P0
RMgq"
```

```
    crossorigin="anonymous"
   />

   <title>iNoteBook</title>
  </head>

  <body>
   <noscript>You need to enable JavaScript to run this app.</noscript>
   <div id="root"></div>
   <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
   -->

   <!-- Option 1: Bootstrap Bundle with Popper -->
   <script
    src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"

integrity="sha384-geWF76RCwLtnZ8qwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy
3Te4Bkz"
    crossorigin="anonymous"
   ></script>

   <script
    src="https://kit.fontawesome.com/e7a7193d30.js"
    crossorigin="anonymous"
   ></script>

   <script>
    window.addEventListener("load", () => {
     registerSW();
    });

    // Register the Service Worker
```

```
    async function registerSW() {
      if ("serviceWorker" in navigator) {
        try {
          const registration = await navigator.serviceWorker.register(
            "serviceworker.js"
          );
          console.log("Registered Service Worker:", registration);
        } catch (error) {
          console.log("SW Registration Failed:", error);
        }
      }
    }
  </script>
 </body>
</html>
```

Installation

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

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

```
self.addEventListener("install", function (e) {
 e.waitUntil(
  caches.open(staticCacheName).then(function (cache) {
   return cache.addAll(["/"]);
  })
 );
});
```

~This is done in serviceworker.js

Activation

Once a service worker has successfully installed, it transitions into the activation stage. If there are any open pages controlled by the previous service worker, the new service worker enters a waiting state. The new service worker only activates when there are no longer any pages loaded that are still using the old service worker. This ensures that only one version of the service worker is running at any given time.

When the new service worker activates, an activate event is triggered in the activating service worker. This event listener is a good place to clean up outdated caches.

```
self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(cacheNames => {
      return Promise.all(
        cacheNames.filter(name => {
          return name !== staticCacheName;
        }).map(name => {
          return caches.delete(name);
        })
      );
    })
  );
});
```

~This is done in serviceworker.js

Therefore, serviceworker.js finally looks like this:

```
var staticCacheName = "pwa";

self.addEventListener("install", function (e) {
 e.waitUntil(
  caches.open(staticCacheName).then(function (cache) {
   return cache.addAll(["/"]);
  })
 );
});

self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(cacheNames => {
      return Promise.all(
```

```
            cacheNames.filter(name => {
               return name !== staticCacheName;
            }).map(name => {
               return caches.delete(name);
            })
         );
      })
   );
});

self.addEventListener("fetch", function (event) {
  console.log(event.request.url);

  event.respondWith(
    caches.match(event.request).then(function (response) {
      return response || fetch(event.request);
    })
  );
});
```

We now run the app using the following command:
> npm run start

```
You can now view inotebook in the browser.

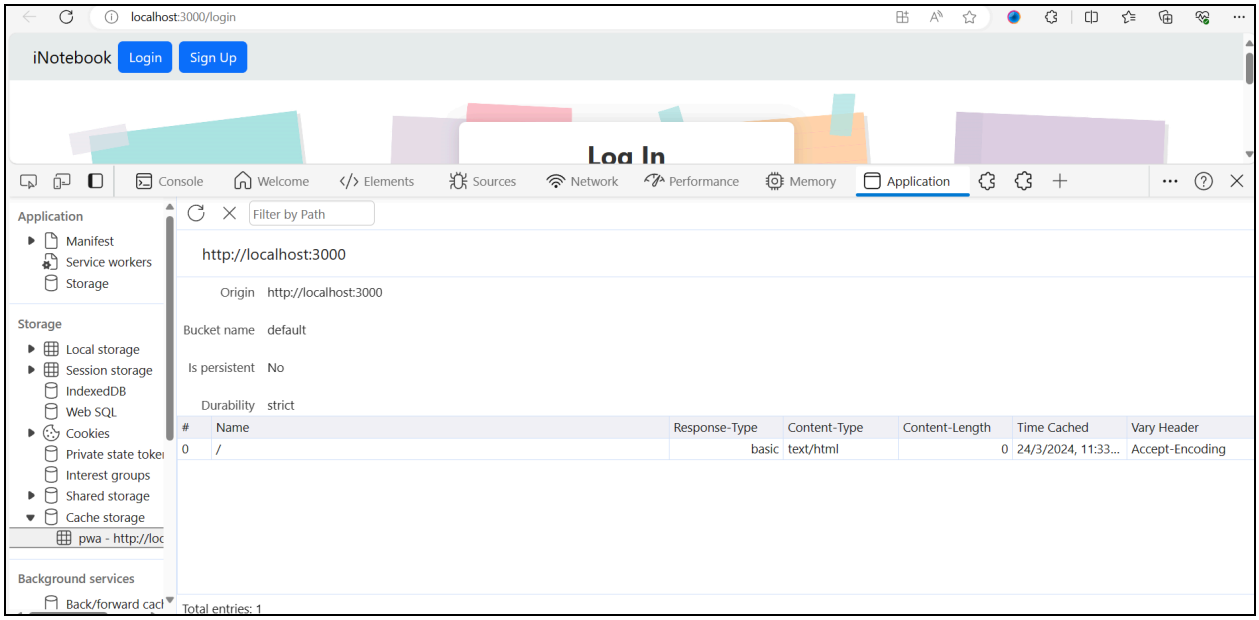
  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.104:3000

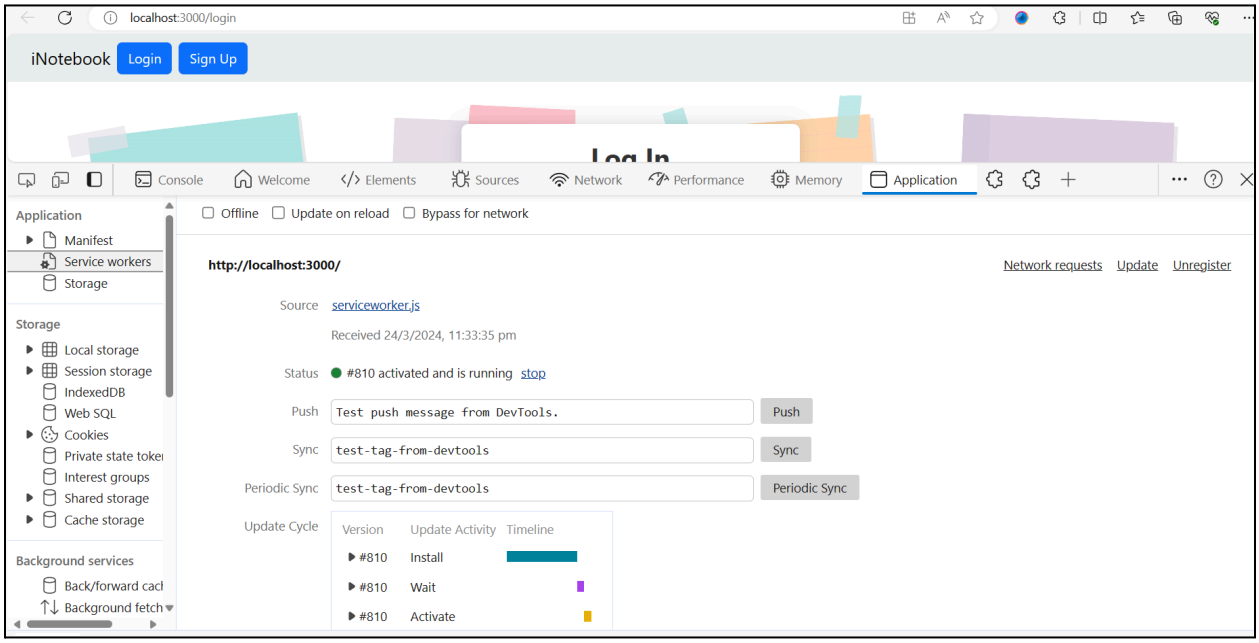
Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

On the browser:

Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**





Conclusion:

The aim was to implement a service worker for aProgressive Web Application (PWA), enabling offline functionality and improving performance. This involved coding and registering the service worker (serviceworker.js) to intercept network requests, cache essential resources, and enhance the offline experience. By completing the install and activation process, the PWA gained features like offline access and faster page loads, enhancing user experience and resilience to network issues.

MAD & PWA Lab
Journal

Experiment No.	09
Experiment Title.	To implement Service worker events like fetch, sync and push for E-commerce PWA
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

MAD PWA LAb 9

NAME: Soham Shetye **Batch :** C

Class : D15A **Roll No.** : 54

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

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

"cacheFirst" but if you request a targeted external URL, this is called "networkFirst".

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

Sync Event

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

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

1. When we click the "send" button, email content will be saved to IndexedDB.
2. Background Sync registration.
3. **If the Internet connection is available**, all email content will be read and sent to Mail Server.

   If the Internet connection is unavailable, the service worker waits until the connection is available even though the window is closed. When it is available, email content will be sent to Mail Server.

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

Push Event

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

We can check in the following example.
"Notification.requestPermission();" is the necessary line to show notification to the

user. If you don't want to show any notification, you don't need this line.

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

serviceworker.js

```
var staticCacheName = "pwa";

self.addEventListener("install", function (event) {
  event.waitUntil(preLoad());
});

self.addEventListener("fetch", function (event) {
  event.respondWith(
    checkResponse(event.request).catch(function () {
      console.log("Fetch from cache successful!");
      return returnFromCache(event.request);
    })
  );
  console.log("Fetch successful!");
  event.waitUntil(addToCache(event.request));
});

self.addEventListener("sync", (event) => {
  if (event.tag === "syncMessage") {
    console.log("Sync successful!");
  }
});

self.addEventListener("push", function (event) {
    if (event && event.data) {
      var data = event.data.json();
      if (data.method == "pushMessage") {
        console.log("Push notification requested");

        if (Notification.permission === "granted") {
          event.waitUntil(
            self.registration.showNotification("To Do List", {
              body: data.message,
```

```
          }).catch(function(error) {
            console.error("Error showing notification:", error);
          })
        );
      } else if (Notification.permission === "denied") {
        console.error("Notification permission denied.");
        // Handle the case where notification permission is denied, such as showing a message to
the user.
      } else {
        Notification.requestPermission().then(function(permission) {
          if (permission === "granted") {
            console.log("Notification permission granted, showing notification");
            self.registration.showNotification("To Do List", {
              body: data.message,
            });
          } else {
            console.error("Notification permission denied.");
            // Handle the case where notification permission is denied after requesting.
          }
        });
      }
    }
  }
 });


var filesToCache = ["/login", "/Home", "/About", "/index.html"];

var preLoad = function () {
  return caches.open("index").then(function (cache) {
    return cache.addAll(filesToCache);
  });
};

var checkResponse = function (request) {
  return new Promise(function (fulfill, reject) {
    fetch(request)
      .then(function (response) {
        if (response.status !== 404) {
          fulfill(response);
        } else {
          reject();
        }
      })
```

```
      .catch(reject);
  });
};

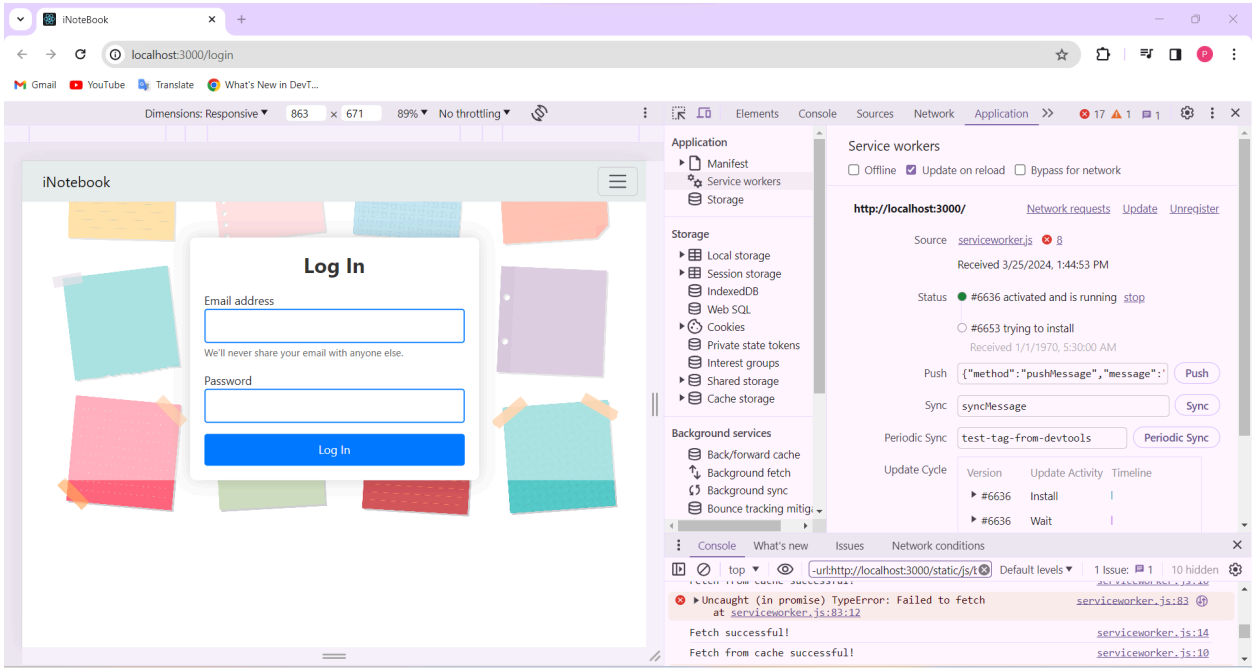
var addToCache = function (request) {
  return caches.open("index").then(function (cache) {
    return fetch(request).then(function (response) {
      return cache.put(request, response);
    });
  });
};

var returnFromCache = function (request) {
  return caches.open("index").then(function (cache) {
    return cache.match(request).then(function (matching) {
      if (!matching || matching.status == 404) {
        return cache.match("index.html");
      } else {
        return matching;
      }
    });
  });
};

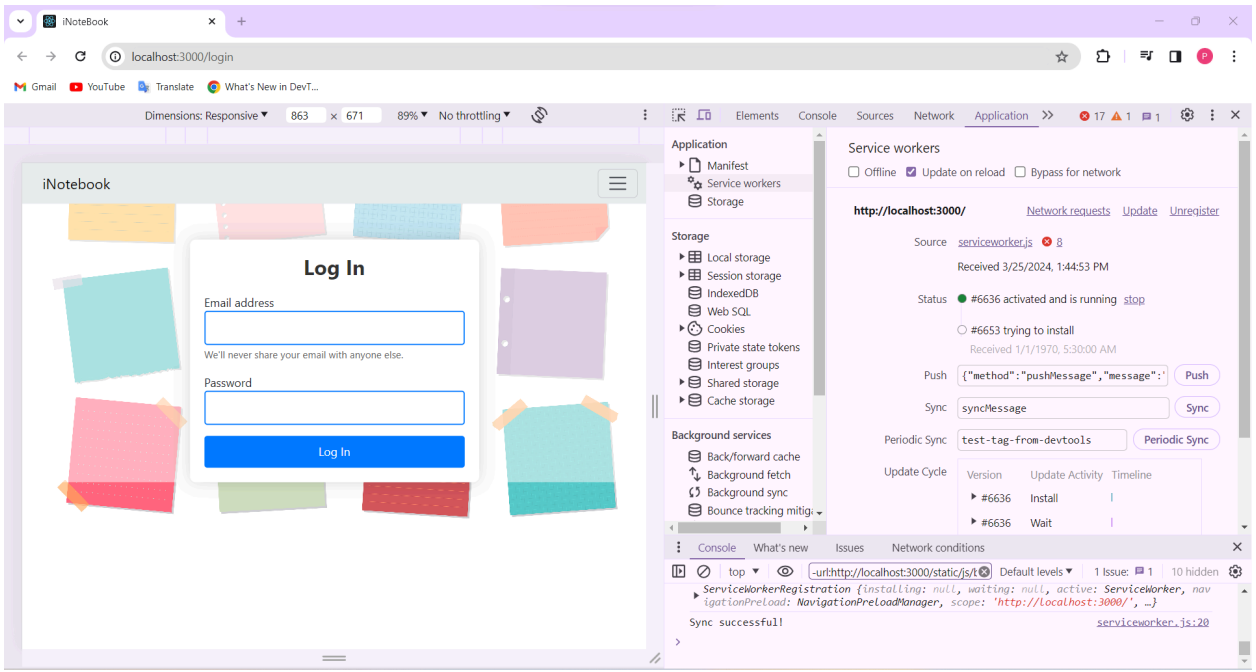
self.addEventListener("activate", (event) => {
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames
          .filter((name) => {
            return name !== staticCacheName;
          })
          .map((name) => {
            return caches.delete(name);
          })
      );
    })
  );
});
```

Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**

Fetch Event :



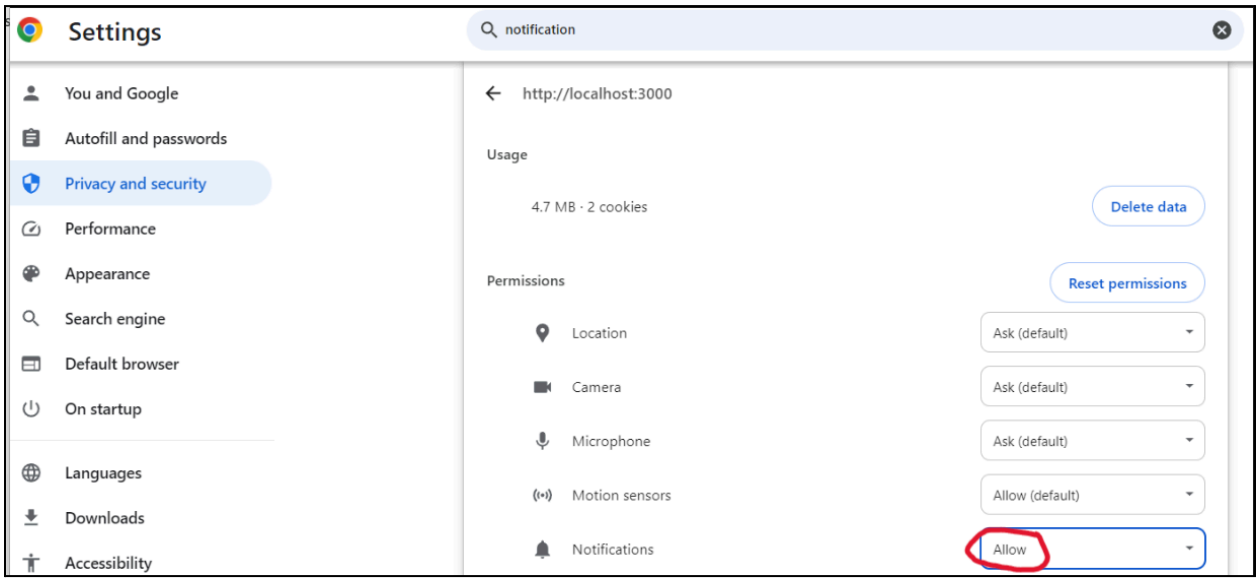
Sync Event :



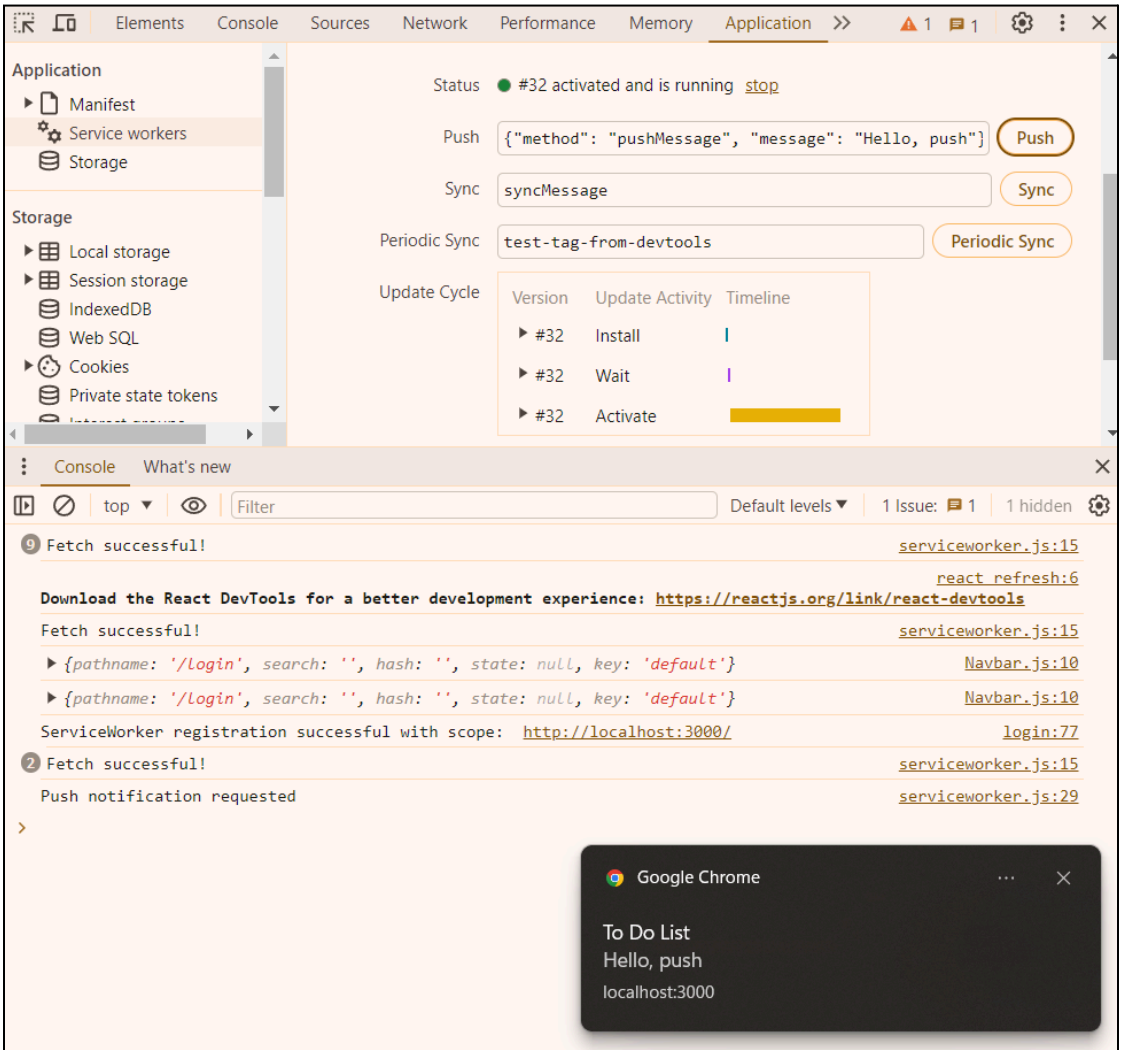
Push Event:

```
Push notification requested                                    serviceworker.js:29
▼Notification permission denied.                               serviceworker.js:42
(anonymous) @ serviceworker.js:42
```

Go to Site Settings and allow Notifications:



Now, we can see the desired result:



CONCLUSION:

- The aim is to enhance the functionality and user experience of a Progressive Web App (PWA) by implementing service worker events such as fetch, sync, and push.
- These events enable the app to efficiently cache resources for offline access, synchronize data in the background, and deliver timely notifications to users, thereby improving performance, reliability, and engagement.

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

AIM:

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

THEORY:

GitHub Pages

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

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

Pros

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

Cons

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

Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

Pros

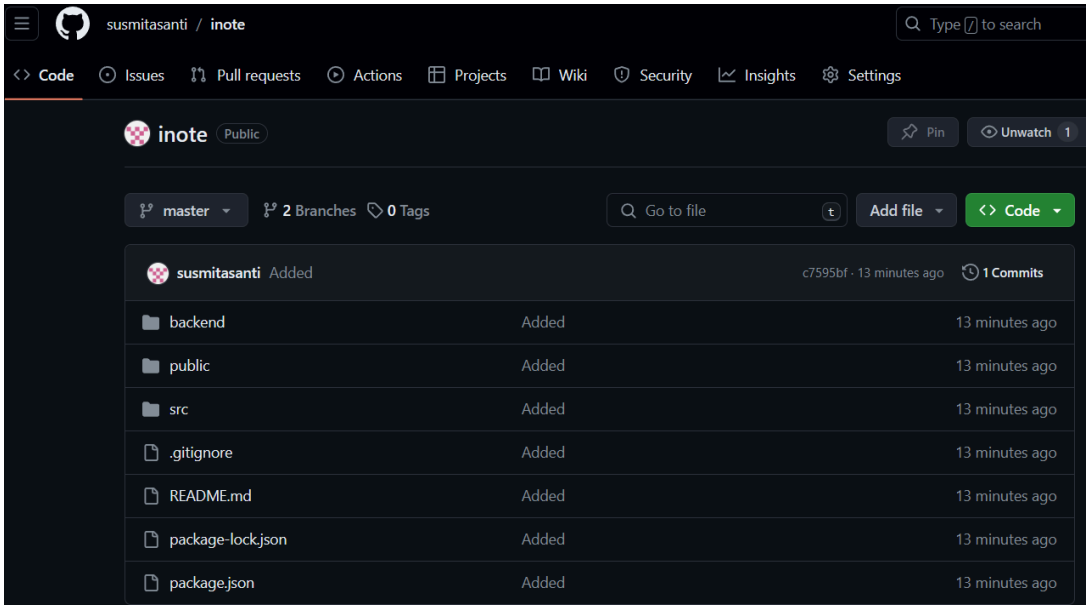
1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5.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

Cons

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

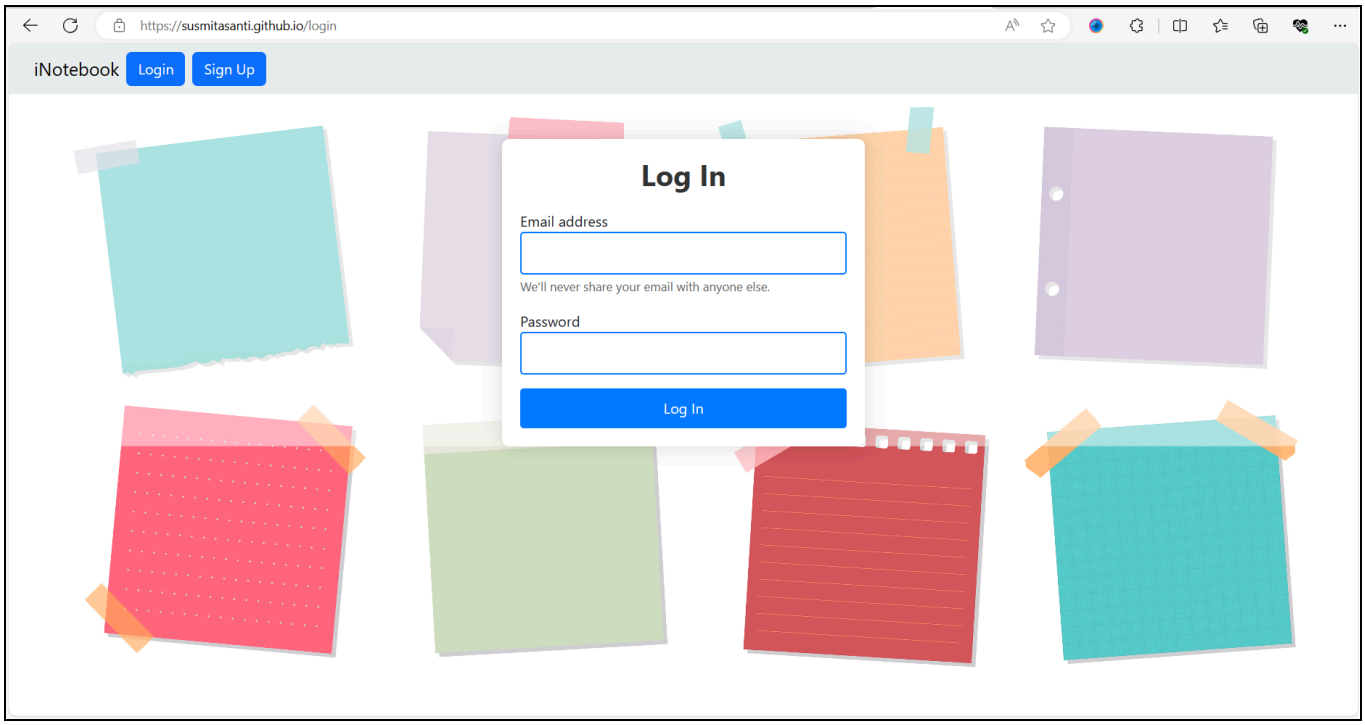
Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**

Link to our GitHub repository: [susmitasanti/inote (github.com)]



HOSTED LINK:
[iNoteBook (susmitasanti.github.io)]

HOSTED IMAGE:



Conclusion :-

In summary, GitHub Pages offers a straightforward, free, and easily accessible platform for hosting static websites, while Firebase provides a robust real-time backend solution with additional services and support from Google.

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

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

THEORY:

Google Lighthouse :

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

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

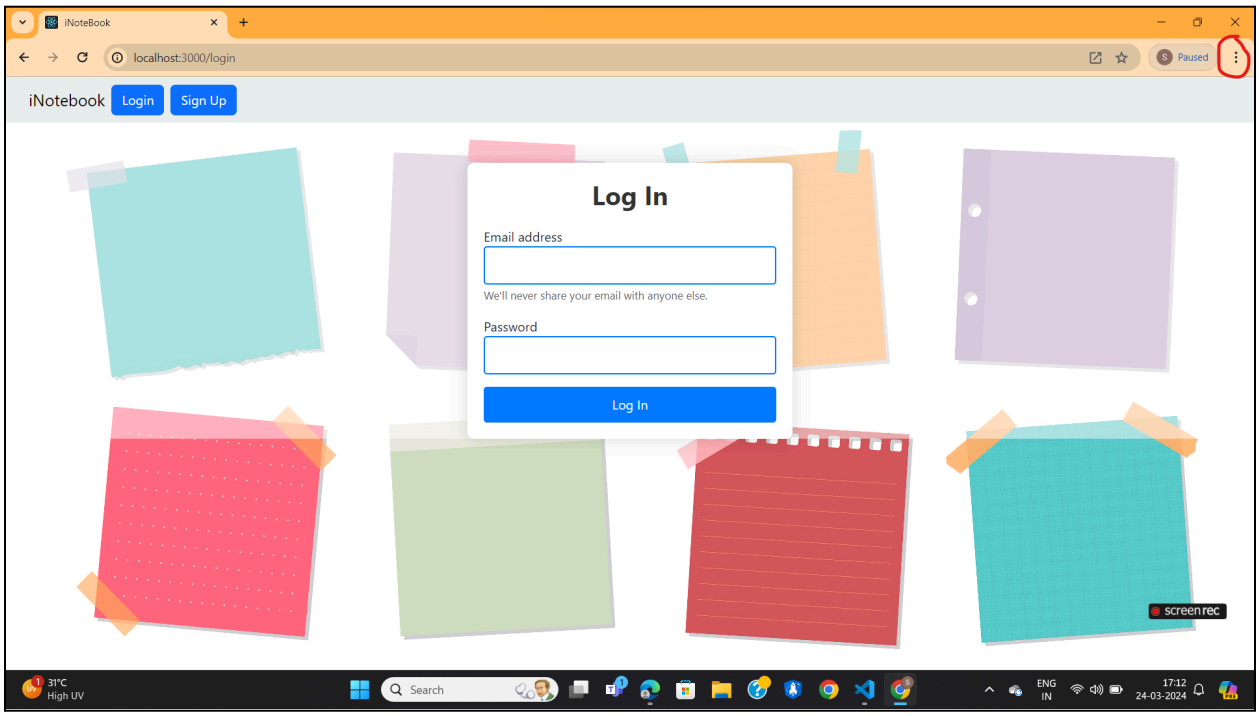
Key Features and Audit Metrics

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

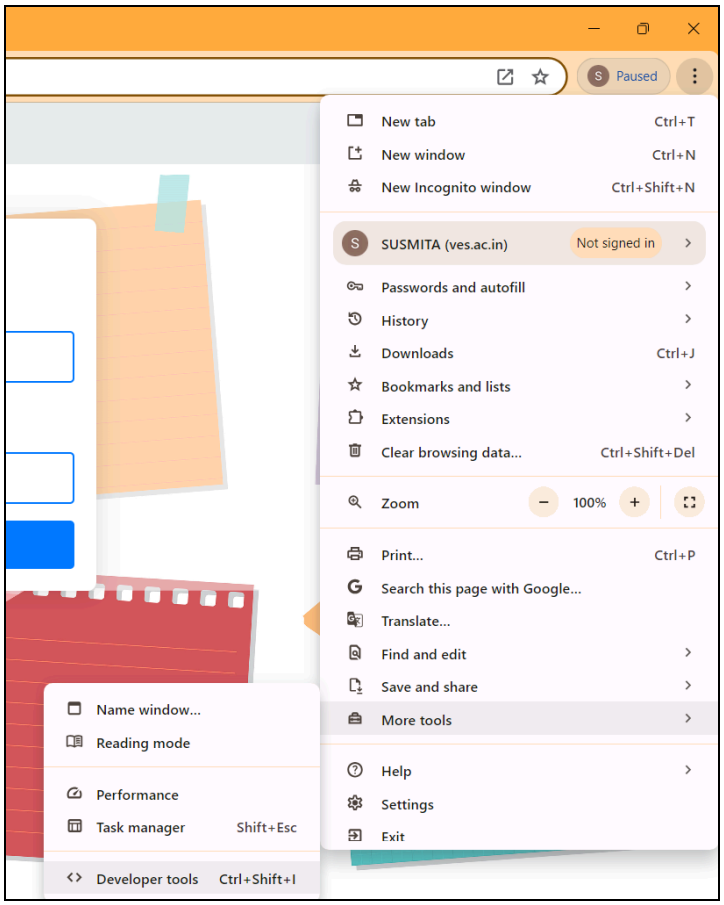
1. **Performance:** This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th percentile and so on.
2. **PWA Score (Mobile):** Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards the PWA paradigm, where the objective is to make the application behave as close as possible to native mobile applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled environments, etc.
3. **Accessibility:** As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually

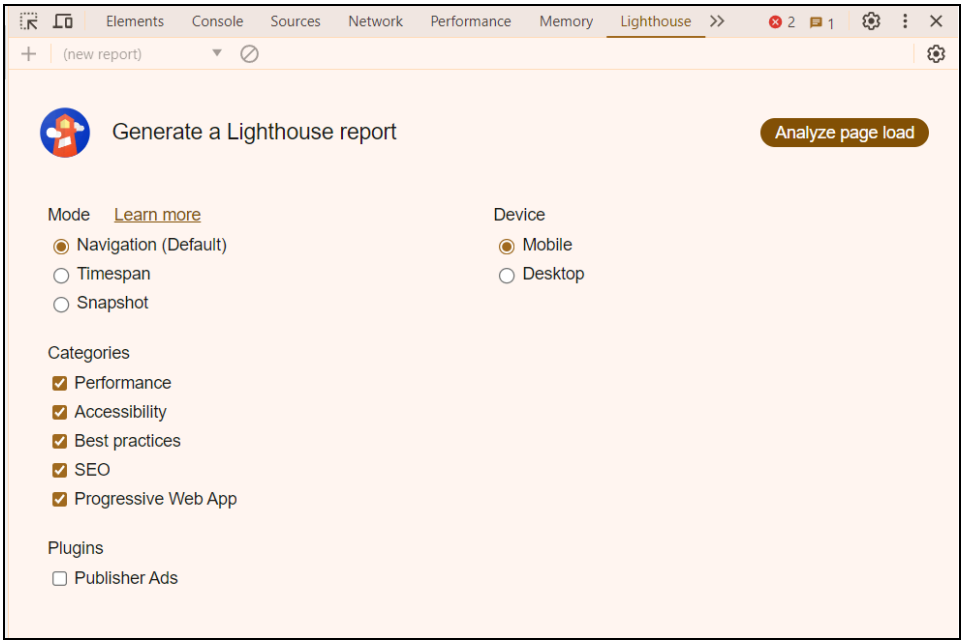
challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.

4. **Best Practices:** As any developer would know, there are a number of practices that have been deemed 'best' based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS
   Avoiding the use of deprecated code elements like tags, directives, libraries, etc. Password input with paste-into disabled
   Geo-Location and cookie usage alerts on load, etc.



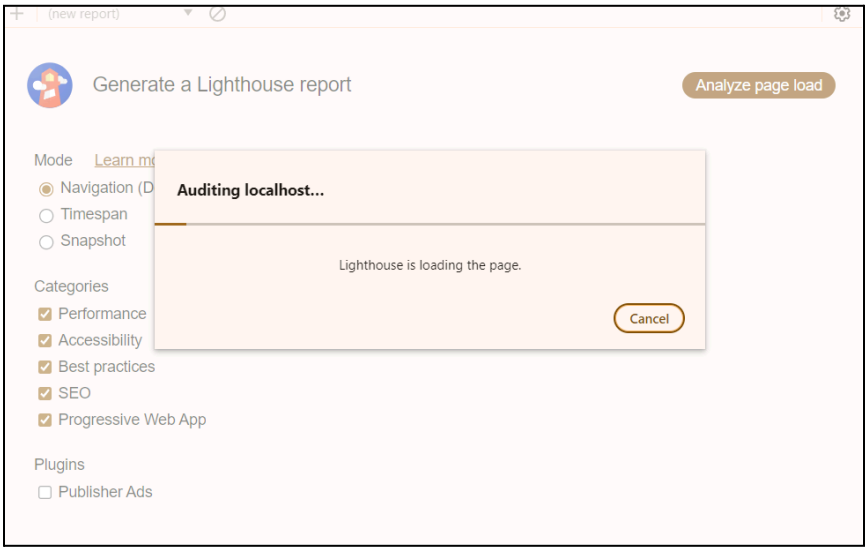
Click on the 3 dots on the top-right corner of the page → Click on **More Tools** → Click on **Developer Tools**

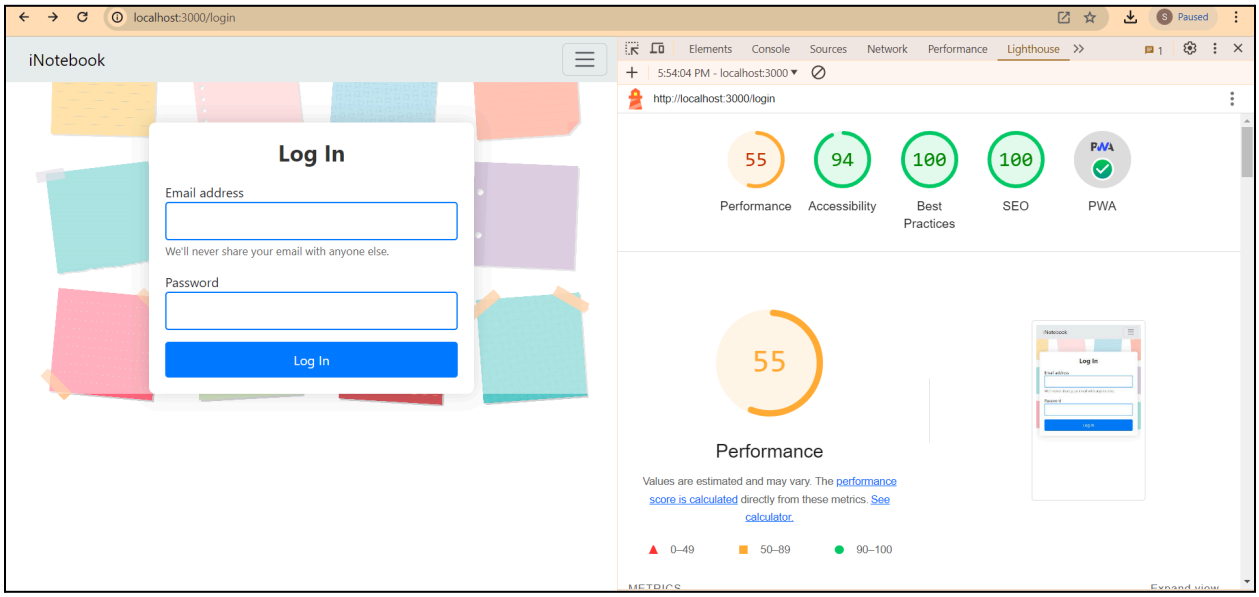




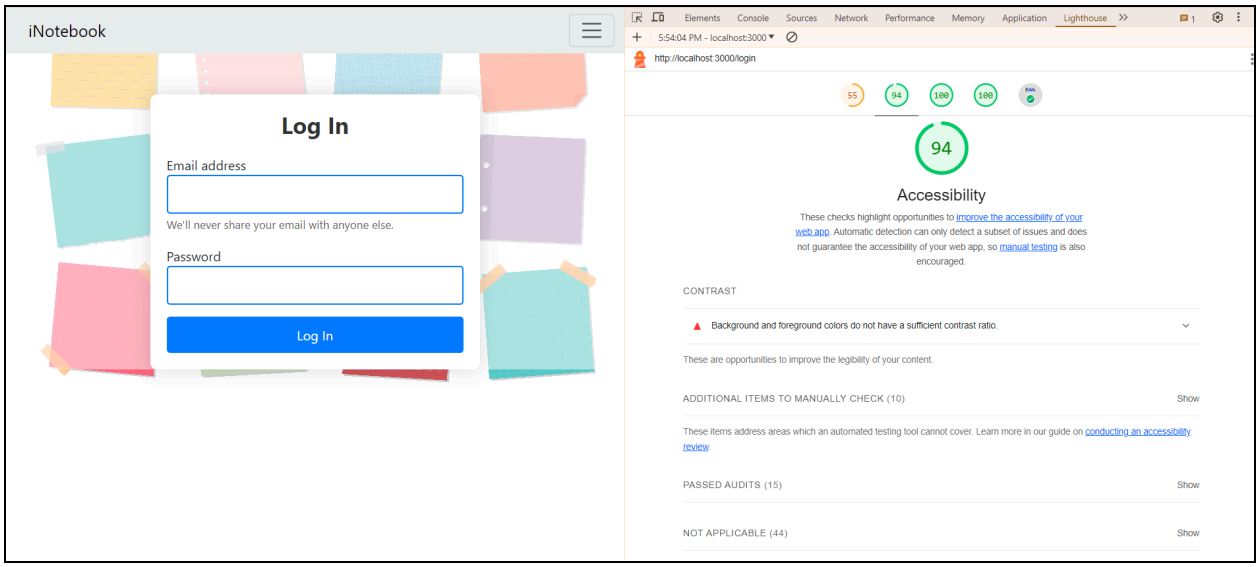
Click on **Analyze page load**

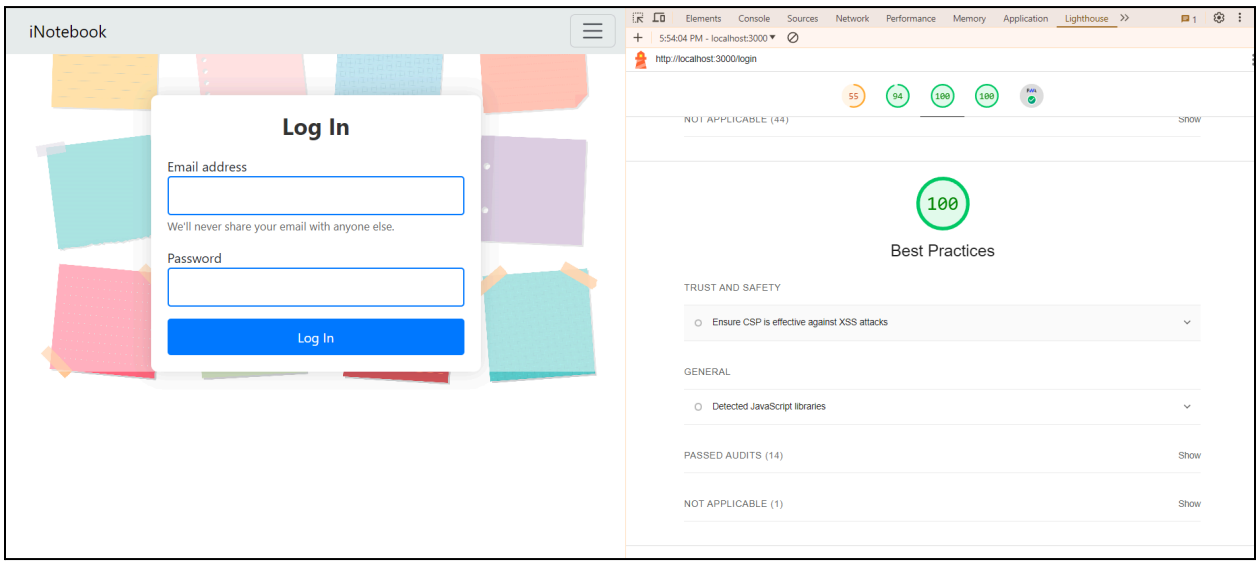
Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**

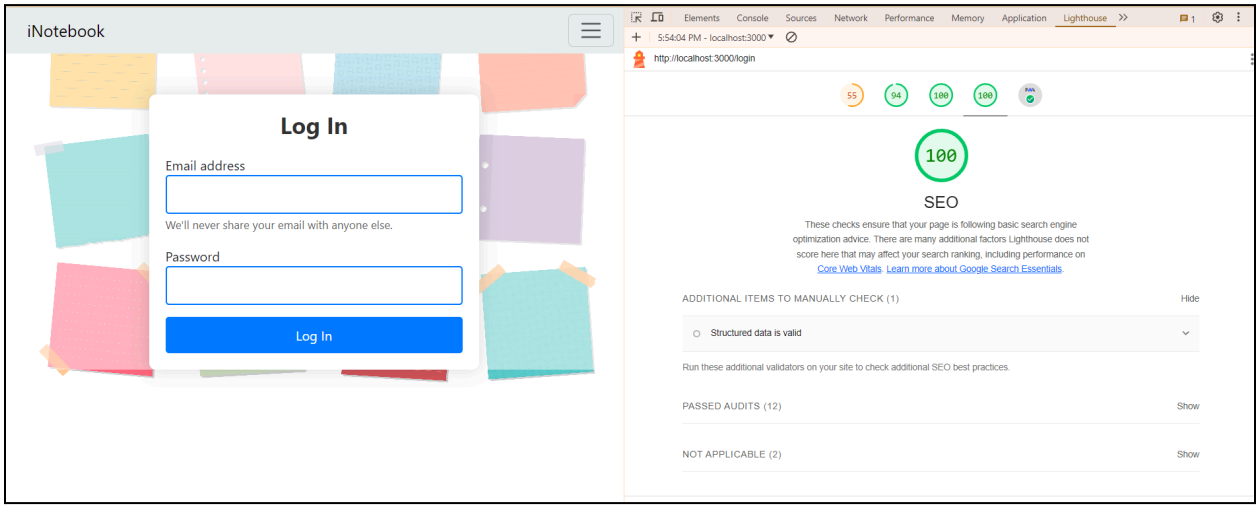


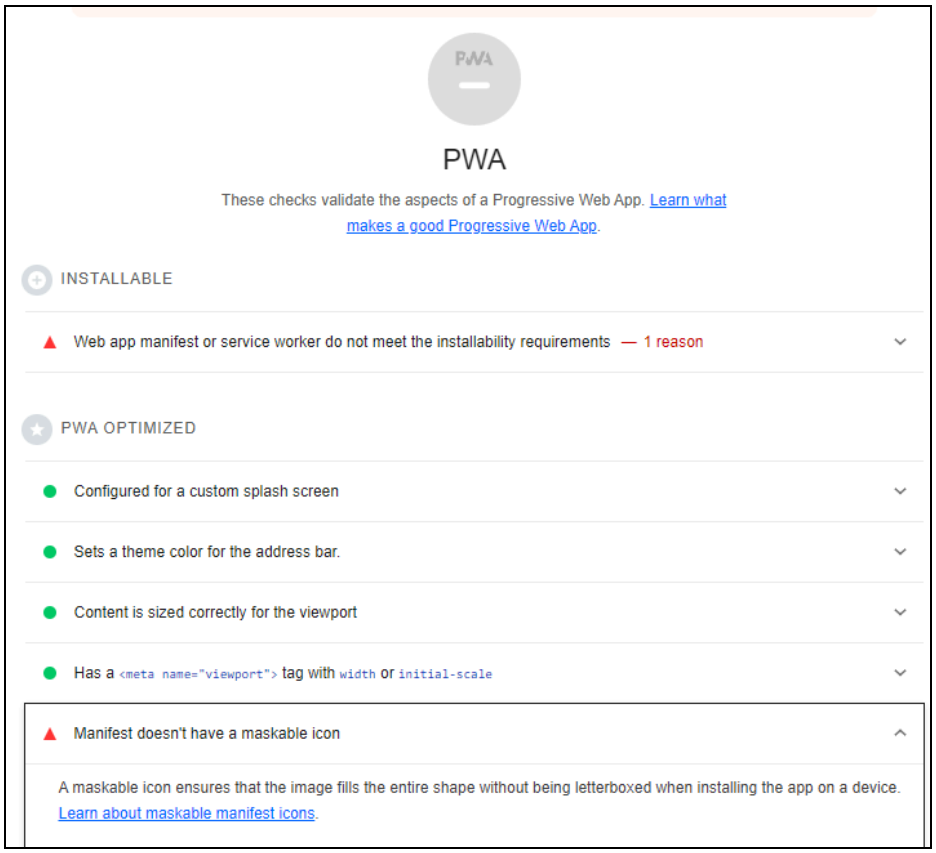


Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**





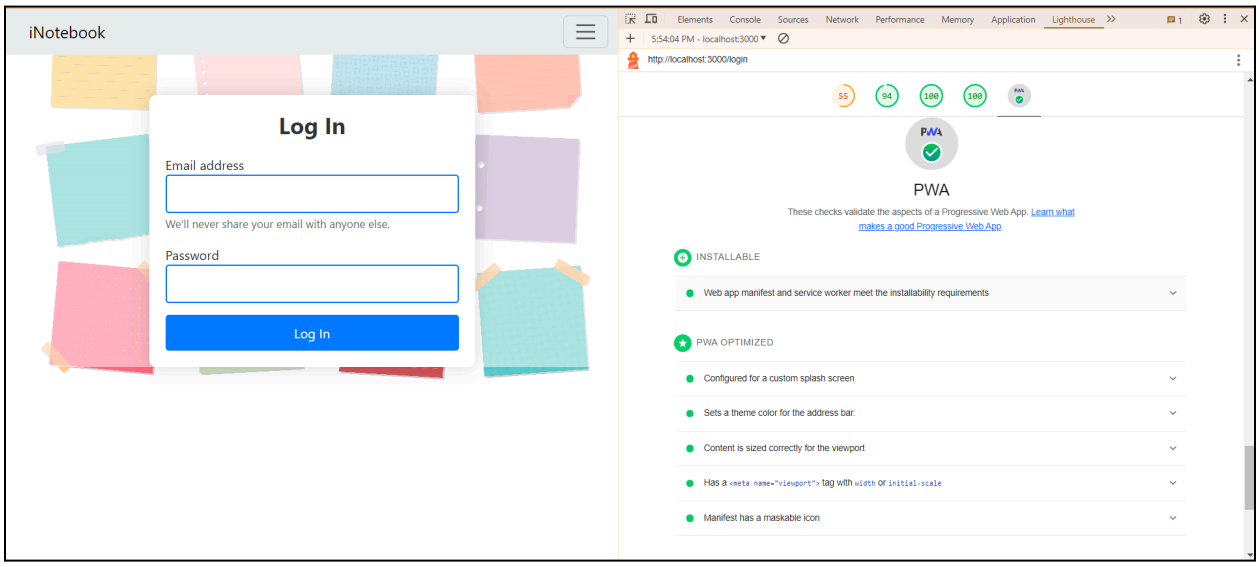




Made the following changes in **manifest.json**

```
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512",
      "purpose":"maskable"
    },
```

```
    {
      "src": "download.png",
      "type": "image/png",
      "sizes": "144x144",
      "purpose": "any"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```



Conclusion: Thus we successfully used google Lighthouse PWA Analysis Tool for testing the PWA functioning.

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	5

Assignment-1

Soham Shitye
D15A 54

Page : flutter Date :

Q.1. flutter Overview: Explain the key features and advantages of using flutter for traditional approaches, and why it has gained popularity in developer community.

Ans Key features of flutter:-

① Cross-platform development
→ . With single codebase, flutter allows you to build apps for both iOS and Android simultaneously.

② Hot Reload
→ . Reloads changes made to the code within the running app almost instantly.

③ Customizable widgets
→ We have full control over every pixel allowing unique and creative designs

④ Open Source and large community.
→ flutter is backed by Google and has extensive resources, libraries, plugins etc.

Advantages of flutter. 8 5

① . faster development
→ . Hot reload and pre built widgets reduce the development time.

② Reduced maintenance
→ . Maintaining a single codebase is easier and cost-effective.

③ future-proof.
→ Constantly evolving with new updates.

Page : Date :

→ Flutter eliminates the need for separate codebase and platform-specific expertise by using Dart.

→ This contrasts with traditional mobile app development where native languages and frameworks for iOS and Android lead to slower development and higher maintenance costs.

→ Flutter's popularity has surged, due to its user-friendly nature, strong performance and cost-effectiveness.

→ Key factor in its widespread adoption include rising demand for cross-platform apps.

→ Flutter's continous evolution and Google's support make it a future-proof technology.

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

Ans Widget tree is a fundamental concept behind Flutter's UI structure. It's a hierarchical organization of widgets.

Page :

Date :

where each widget represents a visual element or behaviour in our app.

Widget composition:

- Parent - child relationships.

→ Child widgets inherit properties and behaviours from their parent.

- Nesting and stacking.

→ Allows to build complex layouts by combining basic building blocks.

- Customization.

→ Each widget can be customized with specific properties, appearance, behaviour and data.

Common widgets and Their Roles:

① Basic widgets

→ Text, Image, container, Row & Column.

② Layout widgets.

→ Stack, Padding & Margin, Center & Align.

③ Interactive widgets

→ Elevated Button & Icon Button, TextField & Dropdown Button, List View & Grid View

Building Complex UIs Example

→ Imagine creating profile list screen. We will start with Container as root widget. Add a column inside it to stack elements vertically

Page : Date :

Each product item becomes child wi
- Row for horizontal layout.
- Image widget for product image.
- Text widget for product name, price and other detail.
- IconButton for adding cart functionality

Benefits of Widget Tree.

(1) Modular and Reusable.
→ Reused across different platforms.

(2) Declarative approach.
→ Flutter handles rendering and updates

(3) Dynamic and flexible.
→ Tree can be dynamically updated. based on user interaction or data changes.

Q.3 State Management in flutter: Discuss the importance of state management in flutter app. Compare and contrast different state management approaches available in flutter, such as setState, Provider, and Riverpod. Provide scenarios where an approch is suitable.

Ans Importance:-

(i) → Maintaining UI consistency.
Keeping the UI in sync with the underlying data ensures responsiveness and prevents displaying outdated information.

Page : Date :

② · Sharing data across widgets.
→ · Improves code organization and reduces redundancy.
③ · Handling complex interaction.
→ · Vital for intricate app features like multiple screens, user action etc.
④ · Building scalable apps.
→ · facilitates modularity and maintainability.

Comparing state Management approach.
① · SetState
→ · Managing state within individual widgets.
→ · Suitable for simple UIs.
→ · Becomes cumbersome for sharing data across screens.

② · provider.
→ · Based on dependency injection.
→ · provides global state container.
→ · Well suitable for sharing data between multiple screens and widgets.

③ · Riverpod.
→ Newer package inspired by provider.
→ · leverages state providers and autoDispose mechanism.
→ smaller community and ecosystem compared to provider.

Page : Date :

Choosing Right Approach.

① Simple UIs → setState.
- Sufficient for basic state management

② Shared data & medium complexity → provider
- Excels at manging shared data.

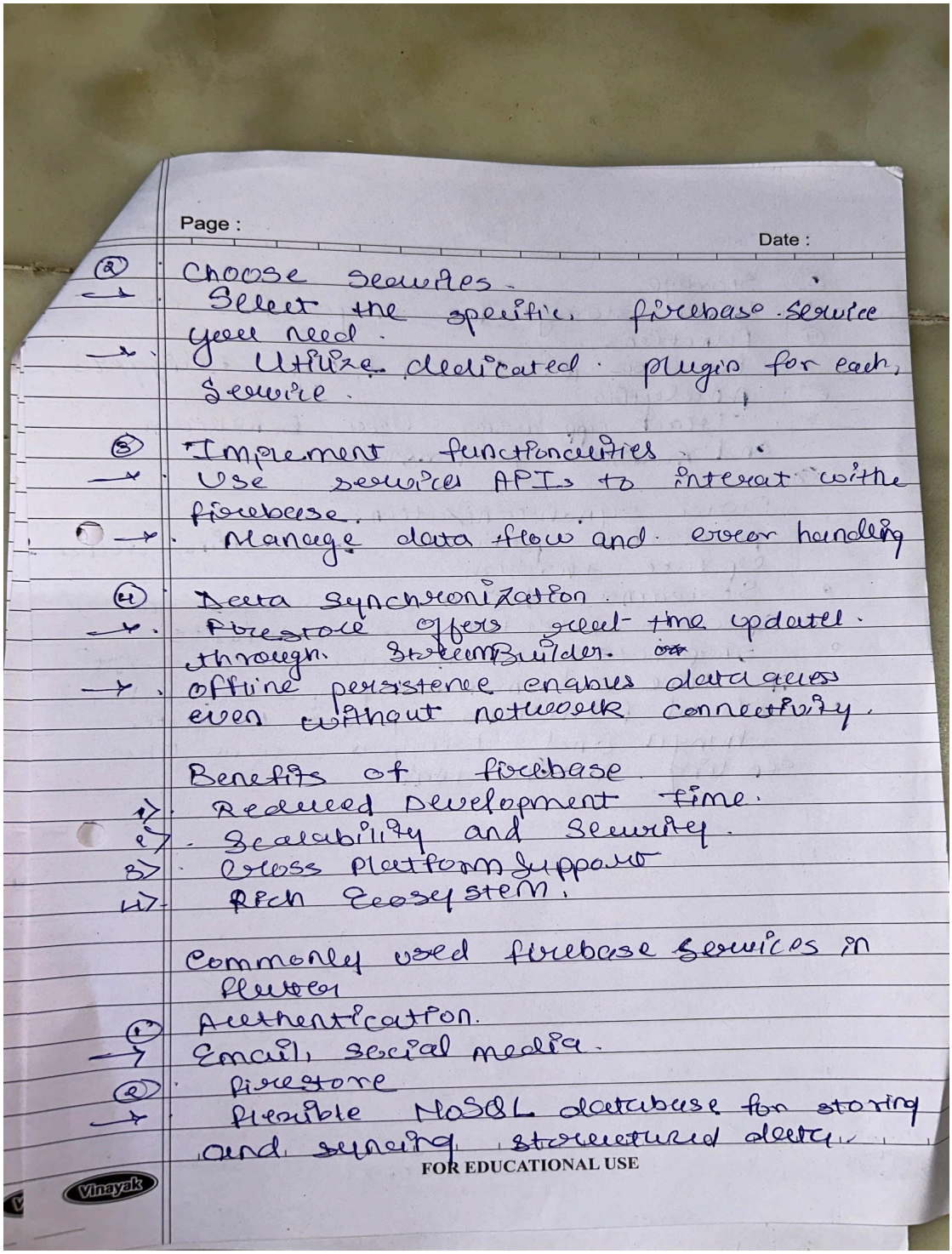
③ Large & complex apps → Riverpod, Clarity and memory management might be beneficial.

Q.4. Firebase Integration in flutter: Explain the process of integrating firebase with flutter application. Discuss the benefits of using firebase as a backend solution. Highlight the firebase services commonly used in flutter development and provide a brief overview of how data synchronization is achieved.

Ans

Integrating firebase with flutter.

① Setup:
- Install flutterfire plugins for desired firebase services.
- configure your firebase project in flutter app using flutterfire configure.
- Initialize firebase in your app's main.dart file.



Page :

Date :

② Choose services.

→ Select the specific firebase service you need.

→ Utilize dedicated plugin for each service.

③ Implement functionalities.

→ Use service API's to interact with firebase.

→ Manage data flow and error handling

④ Data synchronization.

→ Firestore offers real-time updates through StreamBuilder.

→ Offline persistence enables data access even without network connectivity.

Benefits of firebase

1> Reduced development time.

2> Scalability and security.

3> Cross platform support

4> Rich ecosystem.

Commonly used firebase services in flutter

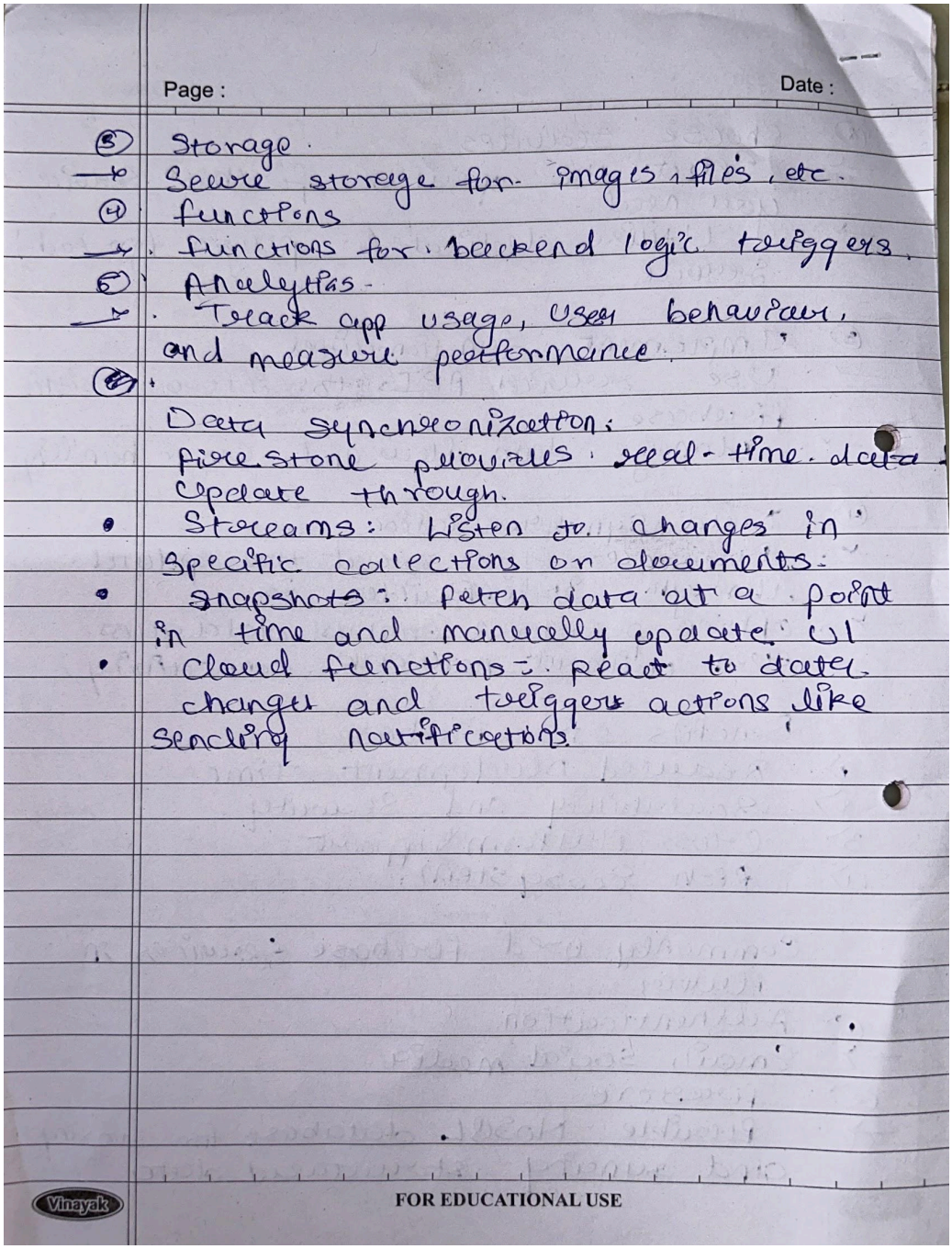
① Authentication.

→ Email, social media.

② Firestore

→ Flexible NoSQL database for storing and syncing structured data.

Vinayak



Page : Date :

(3) Storage.
→ Secure storage for images, files, etc.
(4) functions
→ functions for backend logic triggers.
(5) Analytics.
→ Track app usage, user behaviour, and measure performance.
(6)
Data synchronization:
firestore provides real-time data update through.
- Streams: Listen to changes in specific collections or documents.
- Snapshots: Retch data at a point in time and manually update UI
- Cloud functions: React to data changes and triggers actions like sending notifications

Project Title: OLA Flutter Clone/ PWA: To do list **Roll No. 54**

MAD & PWA Lab
Journal

Experiment No.	Assignment-2
Assignment 2 Questions	1. Define Progressive Web App (PWA) and explain its significance in modern web development.Discuss the key characteristics that differentiate PWAs from traditional mobile apps 2. Define responsive web design and explain its importance in the context of Progressive Web Apps.Compare and contrast responsive, fluid, and adaptive web design approaches. 3. Describe the lifecycle of Service Workers, including registration, installation, and activation phases. 4. Explain the use of IndexedDB in the Service Worker for data storage.
Roll No.	54
Name	Soham Shetye
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	4

MAD PWA LAB.

Soham Shetye
D15A 54

Page : Assignment - 2 Date :

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

Ans → A Progressive web App (PWA) is a type of web application that utilizes modern web technologies to deliver a user experience similar to that of a native mobile app.

→ PWAs are designed to work across various platforms and devices providing users with consistent and reliable experience regardless of device they are using.

→ They combine best features of both mobile and web applications.

Key characteristics :-

(1) Cross-platform compatible.

→ Platforms like desktops, smartphones, tablets.

(2) Responsive design.

→ Adapt different screen sizes and orientation.

(3) Offline functionality

→ Cache resource and content allowing users to access them while offline.

(4) Push notification.

→ Provide timely updates and information

Page : Date :

(5) Installation without app store.
→ Directly install from web browser.
(6) Improved performance
→ Faster load time and smoother interactions.

Q.2 Define responsive web design and explain its importance in context of progressive web apps. Compare and contrast responsive, fluid and adaptive web design approaches.

Ans→ Responsive web design (RWD) dynamically adapts content to various screen sizes and orientations, ensuring optimal user experience
→ RWD is crucial for progressive web Apps to maintain consistency across devices, unlike fluid and Adaptive Approaches,
Comparison.
(1) Responsive Web Design:
→ Uses flexible grids and layouts
→ Utilizes CSS media to apply different styles based on device characteristics.
→ provide seamless and consistent user experience
→ Offers optimal viewing experience.
(2) Fluid web Design,

Page : Date :

→ focuses more on fluidity and flexibility
→ Elements resize proportionally based on size of browser window.
→ Provides smooth transition between screens
→ Content adapts fluidly to fit the available space.

(5) Adaptive web Design.
→ Creating multiple versions of website
→ Detects user's device or screen size and serves the appropriate version of website
→ More customized experience
→ More upfront development effort.

Q.3 Describe the life cycle of service workers, including registerations, installation and activation phases.

ANS (1) Registeration
→ Registering a service worker script with browser.
→ Do this by calling 'navigator.serviceWorker.register()' method in JavaScript code.
→ Registeration occurs in main JavaScript file of web application.
→ During registeration the browser checks if the service worker file is valid and then starts downloading in the background.

Page : Date :

(7) Installation.

→ Once service worker script is succesfully downloaded, the installation phase begins.

→ The browser installs the service worker by parsing and executing its script.

→ Perform task such as caching static assets.

→ Developer can define logic for caching and other tasks in 'install' event listener inside service worker script.

(8) Activation:

→ Activation occurs when there are no active instances of service worker controlling any clients.

→ During activation, the service worker can clean up outdated caches, update any necessary resources.

→ Developer can define the activation logic in 'activate' event listener inside service worker script.

→ Once activated, the service worker can start intercepting network requests, managing cache storage and handling other tasks as defined by developer.

7.4 Explain use of IndexedDB in the service workers for data storage.

Page : Date :

Use of IndexedDB

(1) Offline Caching
- Cache responses for offline access.
- Enables web applications to provide seamless offline experience by serving cached content from IndexedDB when network connectivity is unavailable.

(2) Data Synchronization.
- Service workers can perform background sync operations to synchronize data between server and client.
- IndexedDB can be used to store data locally during synchronization attempts
- If synchronization fails, service worker can retry the operation later.

(3) Persistent Storage.
- Storing user-generated data, such as notes, messages or application settings
- Provides a structured storage solution that allows developers to store and retrieve complex data objects efficiently

(4) Performance Optimization.
- Reducing the need to fetch data from network repeatedly.
- Service workers minimize latency associated with network request, resulting in faster and responsive user experience.