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

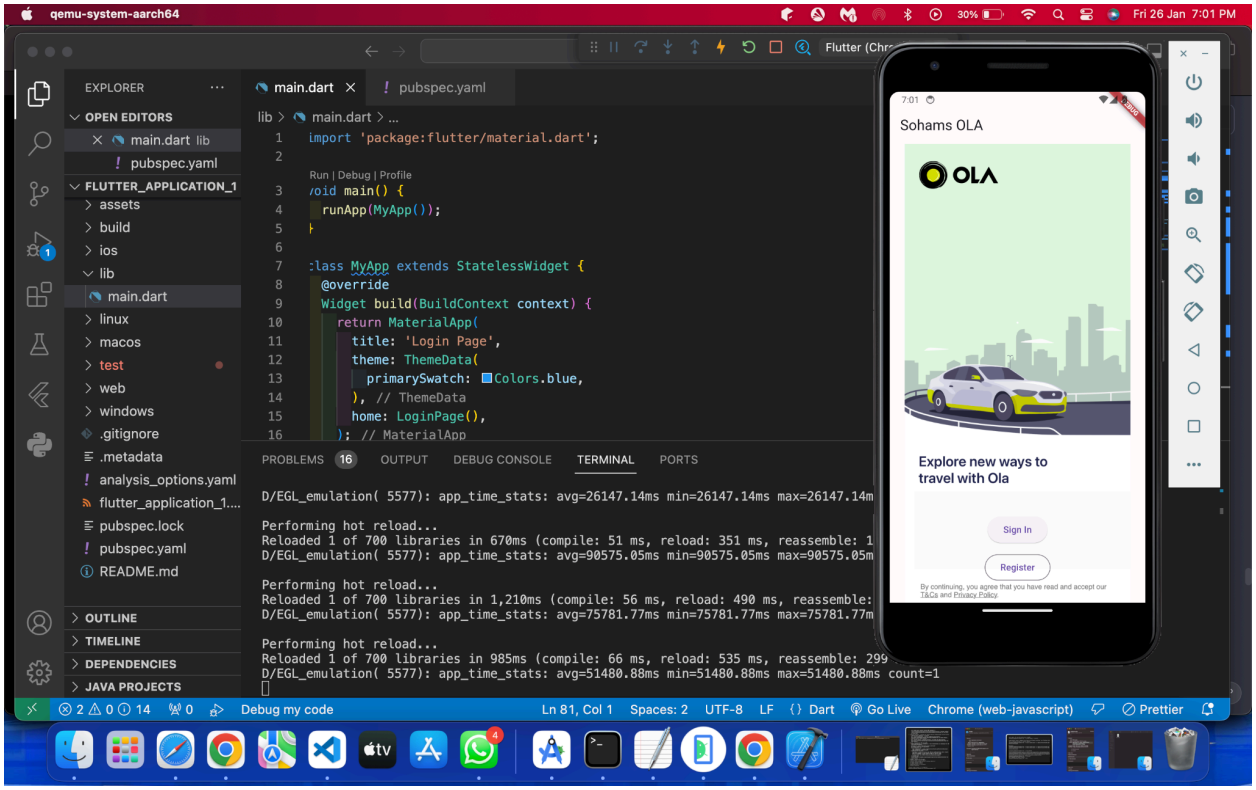
```
import 'package:flutter/material.dart';

void main() {
```

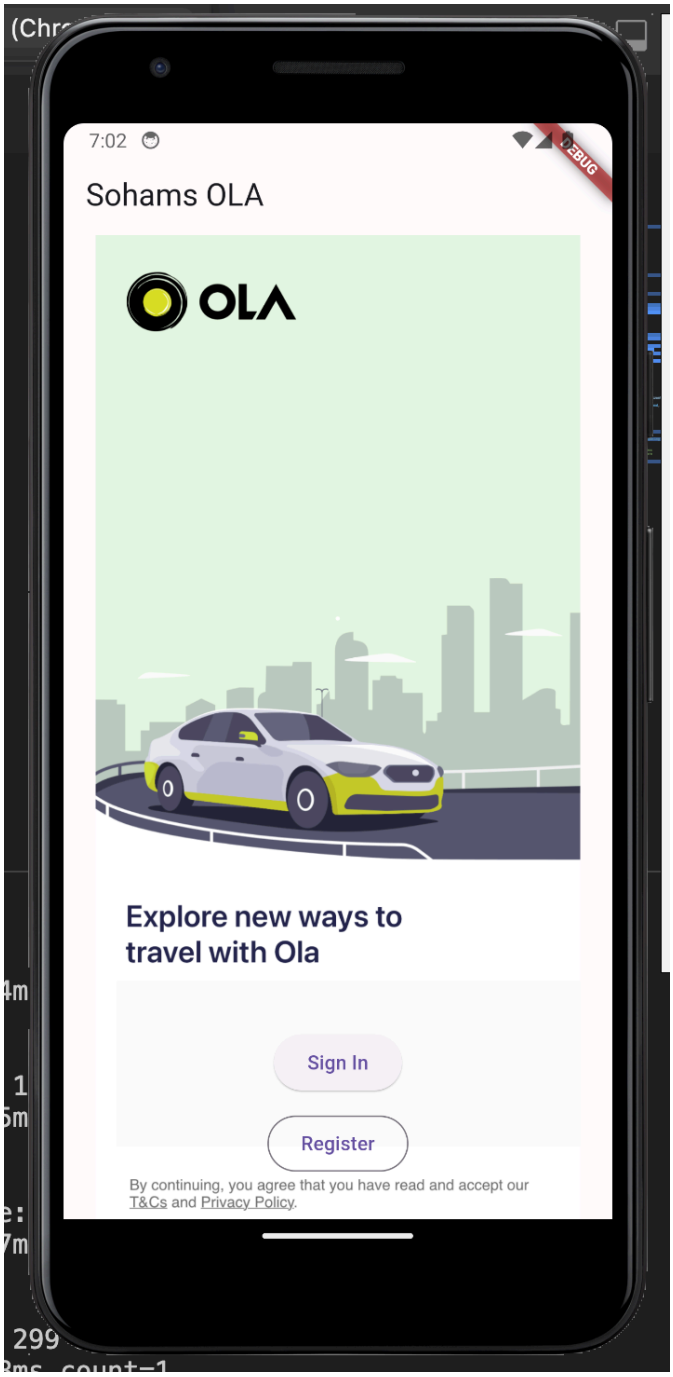
```
 runApp(MyApp());
}

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,
                  ),
```

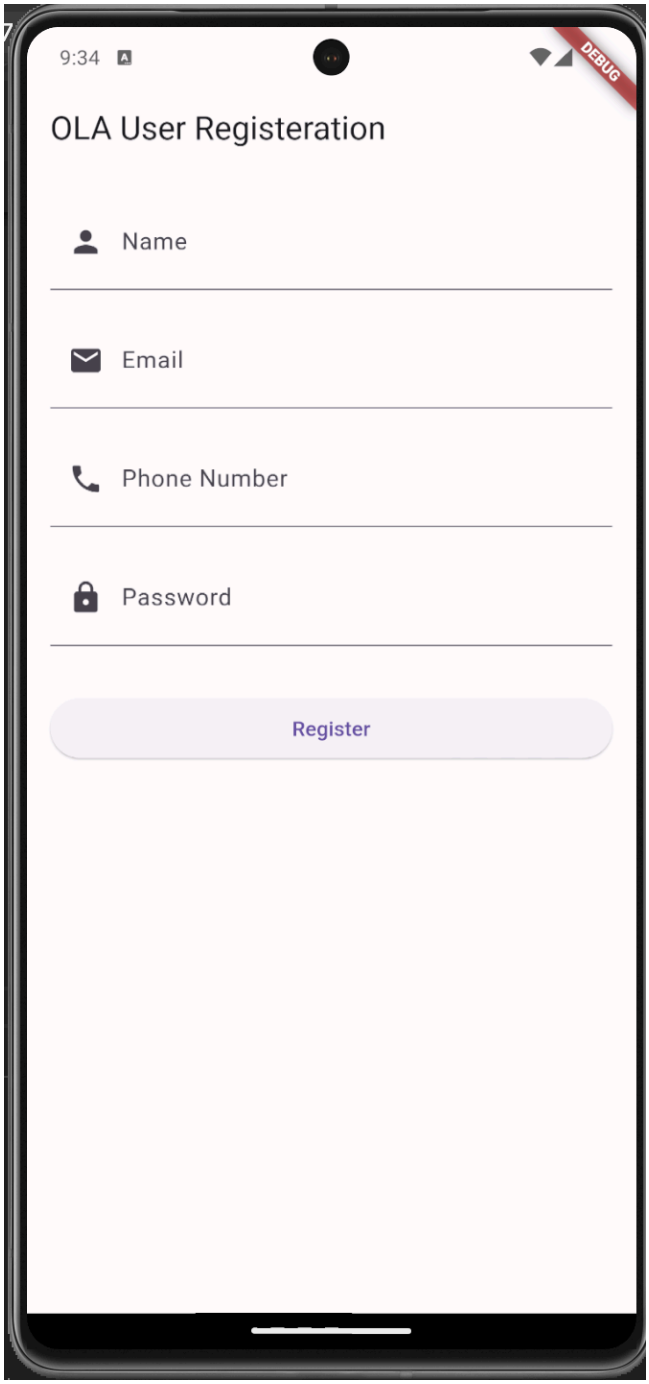
```
              ),
            ),
          ),
        ),
        Padding(
          padding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 30.0), // Adjust padding
here
          child: Column(
            mainAxisAlignment: MainAxisAlignment.end, // Align children at the bottom
            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
```


qemu-system-aarch64
Fri 26 Jan 7:01 PM
EXPLORER
OPEN EDITORS
main.dart lib
pubspec.yaml
FLUTTER_APPLICATION_1
assets
build
ios
lib
main.dart
linux
macos
test
web
windows
.gitignore
.metadata
analysis_options.yaml
flutter_application_1....
pubspec.lock
pubspec.yaml
README.md
OUTLINE
TIMELINE
DEPENDENCIES
JAVA PROJECTS
main.dart
pubspec.yaml
lib > main.dart > ...
import 'package:flutter/material.dart';
Run | Debug | Profile
void main() {
runApp(MyApp());
}
class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Login Page',
theme: ThemeData(
primarySwatch: Colors.blue,
), // ThemeData
home: LoginPage(),
); // MaterialApp
PROBLEMS 16
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
D/EGL_emulation(5577): app_time_stats: avg=26147.14ms min=26147.14ms max=26147.14m
Performing hot reload...
Reloaded 1 of 700 libraries in 670ms (compile: 51 ms, reload: 351 ms, reassemble: 1
D/EGL_emulation(5577): app_time_stats: avg=90575.05ms min=90575.05ms max=90575.05m
Performing hot reload...
Reloaded 1 of 700 libraries in 1,210ms (compile: 56 ms, reload: 490 ms, reassemble:
D/EGL_emulation(5577): app_time_stats: avg=75781.77ms min=75781.77ms max=75781.77m
Performing hot reload...
Reloaded 1 of 700 libraries in 985ms (compile: 66 ms, reload: 535 ms, reassemble: 299
D/EGL_emulation(5577): app_time_stats: avg=51480.88ms min=51480.88ms max=51480.88ms count=1
Debug my code
Ln 81, Col 1
Spaces: 2
UTF-8
LF
Dart
Go Live
Chrome (web-javascript)
Prettier
Sohams OLA
OLA
Explore new ways to travel with Ola
Sign In
Register


7:02
DEBUG
Sohams OLA
OLA
Explore new ways to travel with Ola
Sign In
Register
By continuing, you agree that you have read and accept our T&Cs and Privacy Policy.



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.