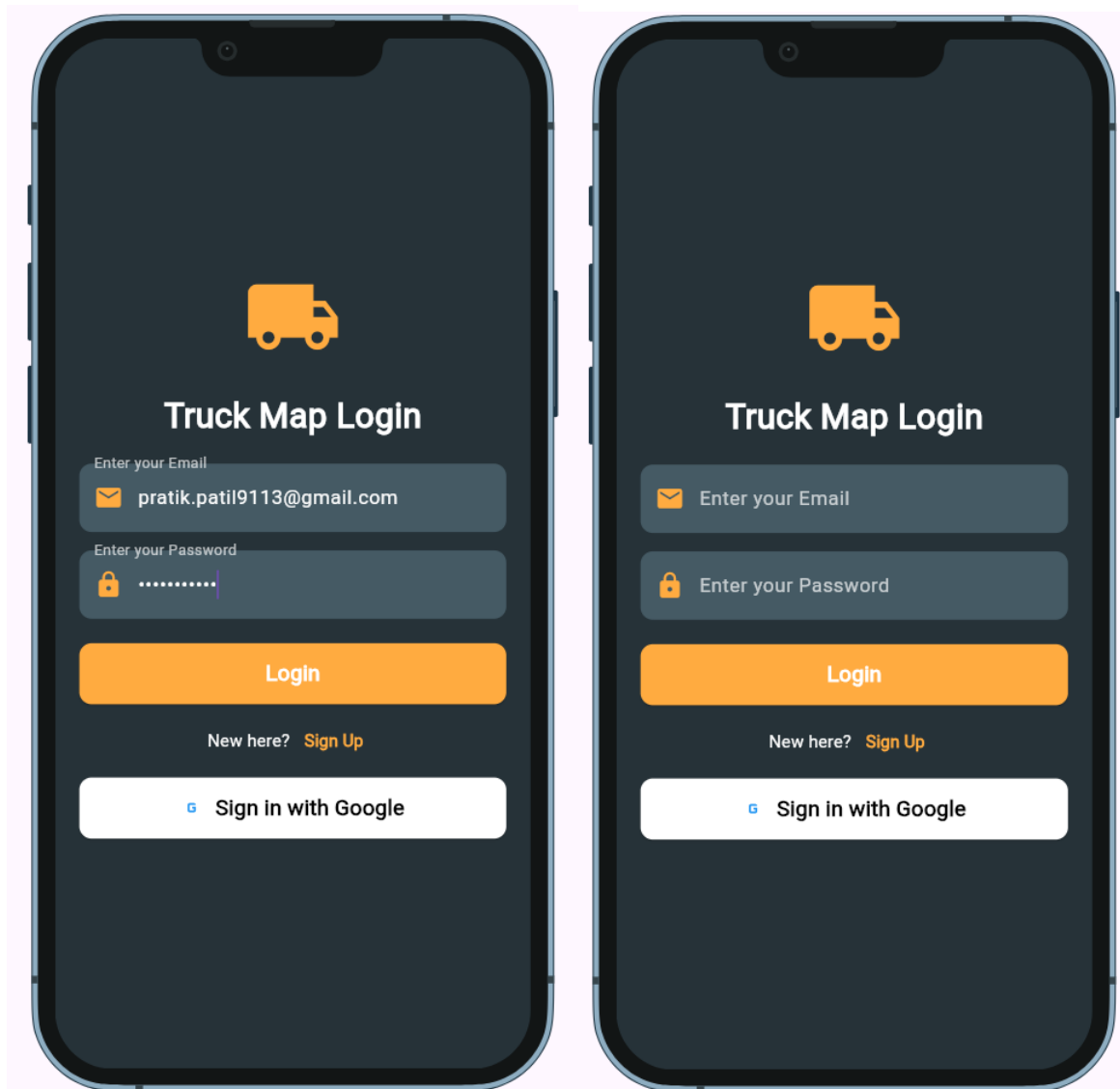


**Aim:** create an interactive form using form widgets in flutter.

**Theory:**



Flutter provides a comprehensive set of widgets for building forms, allowing developers to create interactive user interfaces for collecting user input. These form widgets streamline the process of handling user input validation, submission, and data processing.

### **Form Widget:**

The Form widget is the foundation for creating forms in Flutter. It acts as a container for form fields and provides methods for form validation and submission. The Form widget

manages the form state internally and provides access to the `FormState` object, which can be used to interact with the form fields.

### TextFormField Widget:

The `TextFormField` widget is used to create text input fields within a form. It provides various properties for customizing the appearance and behavior of the input field, such as decoration, validation, and input formatting. Developers can specify validators to enforce input constraints and error messages to provide feedback to users when input validation fails.

```
const SizedBox(height: 20),
Text(
  isLogin ? 'Truck Map Login' : 'Truck Map Sign Up',
  style: const TextStyle(
    fontSize: 28,
    fontWeight: FontWeight.bold,
    color: Colors.white,
  ), // TextStyle
), // Text
const SizedBox(height: 20),
_buildTextField(emailController, 'Enter your Email', Icons.email),
const SizedBox(height: 15),
_buildTextField(
  passwordController, 'Enter your Password', Icons.lock,
  isPassword: true),
const SizedBox(height: 20),
_buildButton(isLogin ? 'Login' : 'Sign Up',
  createUserWithEmailAndPassword),
const SizedBox(height: 10),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Text(
      isLogin ? 'New here?' : 'Already registered?',
      style: const TextStyle(color: Colors.white),
    ), // Text
    TextButton(
      onPressed: () {
        setState(() {
          isLogin = !isLogin;
        });
      },
    ),
  ],
),
```

```
Widget _buildTextField(
  TextEditingController controller, String hint, IconData icon,
  {bool isPassword = false}) {
  return TextField(
    controller: controller,
    obscureText: isPassword,
    decoration: InputDecoration(
      labelText: hint,
      prefixIcon: Icon(icon, color: Colors.orangeAccent),
      filled: true,
      fillColor: Colors.blueGrey[700],
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(10),
        borderSide: BorderSide.none,
      ), // OutlineInputBorder
      labelStyle: const TextStyle(color: Colors.white70),
    ), // InputDecoration
    style: const TextStyle(color: Colors.white),
  ); // TextField
}
```

### Form Validation:

Flutter provides built-in support for form validation using the `validator` property of form fields. Developers can define validator functions that evaluate the input value and return an error message if the input does not meet the specified criteria. The `Form` widget automatically triggers validation when the form is submitted, and displays error messages for invalid fields.

```
bool isLogin = true;
final TextEditingController emailController = TextEditingController();
final TextEditingController passwordController = TextEditingController();

@override
void dispose() {
  emailController.dispose();
  passwordController.dispose();
  super.dispose();
}
```

### Form Submission:

Form submission in Flutter involves handling user input after the form is validated. Developers typically use the `onPressed` callback of a submit button to trigger form submission. Within the submit callback, developers can access the current state of the form using the `FormState` object and retrieve the values of individual form fields for further processing, such as data submission to a server or local storage.

### Feedback and Error Handling:

Providing feedback to users during form interaction is crucial for a positive user

experience. Flutter allows developers to display error messages and visual indicators to guide users when input validation fails. Error messages can be displayed inline with form fields or in a separate section of the form, depending on the design requirements.

Additionally, developers can use dialogs or snack bars to provide feedback upon successful form submission or error handling.

By leveraging these form widgets and techniques, developers can create intuitive and responsive forms in Flutter applications, enabling seamless interaction with users and efficient data collection and processing.

## **Conclusion:**

Creating interactive forms in Flutter using form widgets is essential for building user-friendly applications that collect and process user input effectively. By utilizing form widgets such as Form and TextFormField, along with form validation and submission techniques, developers can design robust and responsive forms that enhance the overall user experience. With Flutter's flexibility and rich set of features, developers have the tools they need to create dynamic and interactive forms tailored to their application's requirements.