Experiment No.4
MAD & PWA LAB

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

- **Theory:**

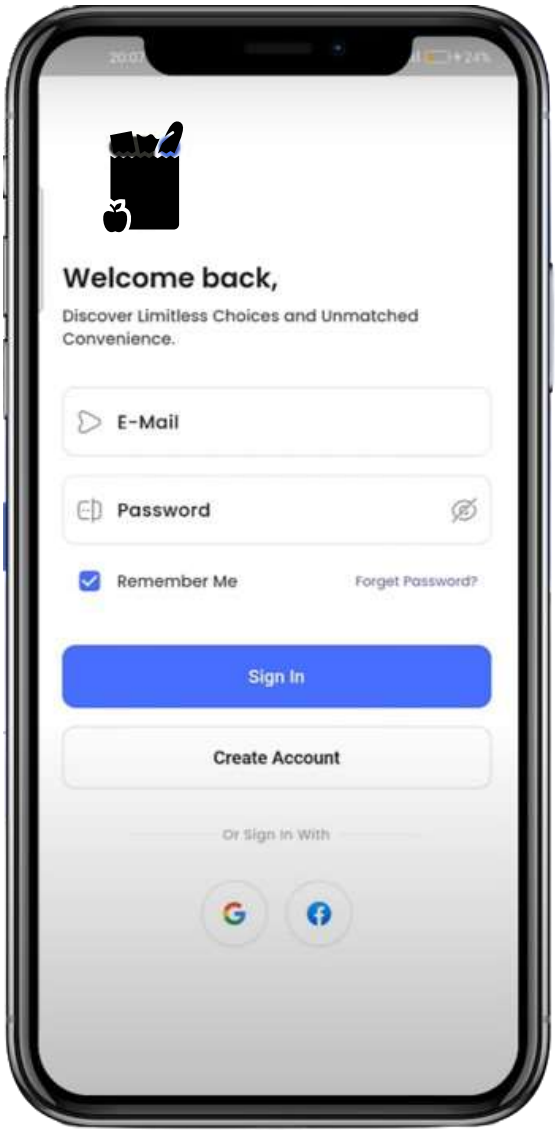
To enhance the security and user-friendliness of apps, it's common to incorporate Text Fields for user input, such as logging in with an Email Address and Password. Ensuring the validity of the provided information is crucial for a seamless experience. By creating a Form widget, you establish a structured container that facilitates the grouping and validation of multiple form fields.

When initiating the form, it's essential to assign a GlobalKey. This key uniquely identifies the Form, enabling subsequent validation of the form's content. If the user accurately completes the form, the information is processed. Conversely, if incorrect details are submitted, a user-friendly error message is displayed, offering clear guidance on rectifying the provided information.

→ **Form Widget: -**
The Form widget is a container for FormFields, which are widgets that represent a single form entry. The Form widget helps manage the form state and provides methods to save, reset, and validate the form.

→ **TextFormField:**
The TextFormField widget is commonly used within a Form. It is a material design text input field that allows the user to enter and edit text. It comes with built-in validation and formatting options.

→ **TextFormField Properties:**
- ✓ **controller:** A TextEditingController to control the text being edited.
- ✓ **validator:** A callback function to validate the input.
- ✓ **onSaved:** A callback function called when the form is saved.
- ✓ **decoration:** Defines the appearance of the input field.




```
Column(
  crossAxisAlignment:
CrossAxisAlignment.start,
  children: [
    Image(
      height: 150,
      image: AssetImage(dark ?
TImages.lightAppLogo :
TImages.darkAppLogo),
    ),
    Text(TTexts.loginTitle, style:
Theme.of(context).textTheme.headlineMe
dium),
    const SizedBox(height: TSizes.sm),
    Text(TTexts.loginSubTitle, style:
Theme.of(context).textTheme.bodyMediu
m),
  ],),
Column(
  children: [
    Form(
      child: Column(
        crossAxisAlignment:
CrossAxisAlignment.start,
        children: [
          TextFormField(
            decoration: const InputDecoration(
              prefixIcon:
Icon(Iconsax.direct_right),
              labelText: TTexts.email,
            ), ),
          const SizedBox(height:
TSizes.spaceBtwInputFields),
          TextFormField(
            decoration: const InputDecoration(
              prefixIcon:
Icon(Iconsax.password_check),
              labelText: TTexts.password,
              suffixIcon:
Icon(Iconsax.eye_slash),
            ),  ),
          const SizedBox(height:
TSizes.spaceBtwInputFields / 2),
        ],
      ),
    ),
  ],
),
```

→ **GlobalKey<FormState>:**
It is recommended to use a GlobalKey<FormState> to uniquely identify the Form and interact with it, such as validating the form, saving its state, or resetting it.

→ **Form Lifecycle:**
A form goes through various states in its lifecycle:
- ✓ **Initialization:** The Form is created with a key.
- ✓ **Validation:** The Form is validated using the validator callback on each TextFormField.
- ✓ **Saving:** The onSaved callback is invoked to save the form data.
- ✓ **Resetting:** The form can be reset using the FormState.reset() method.

→ **Handling Form Submission:**
After the user fills out the form and submits it, the data can be processed, such as sending it to a server or performing some action.

→ **Input Types:**
Flutter's TextFormField supports various input types, such as text, number, email, password, and more. You can use the keyboardType property to specify the keyboard type based on the expected input.

→ **Focus and FocusNode:**
Managing focus is crucial in forms. The FocusNode class helps manage the focus on different form fields. It can be used to control which form field has focus and customize the behavior when fields gain or lose focus.

→ **InputDecoration:**
The InputDecoration class is used to customize the appearance of the input field. It allows you to add labels, hints, icons, and borders to provide a better user experience.

→ **Form Validation:**
Flutter's form validation system is based on the validator callback provided in form fields. You can perform custom validation logic, such as checking for required fields or validating input patterns. The autovalidateMode property of the Form widget helps in auto-validating the form as the user types.

→ **Form Submission and Backend Integration:**
After form submission, you may want to send the data to a server. Flutter provides various methods, such as using the http package to make HTTP requests. You can handle the server response and update the UI accordingly.

→ **Benefits Of Forms:**

1) **Efficient Data Collection:** Forms provide a structured and organized method for collecting various types of data, streamlining the process of gathering information from users.
2) **Enhanced User Interaction:** Forms enable users to actively engage with applications by providing a means to input information, express preferences, and submit feedback or inquiries.
3) **Data Validation and Integrity:** Forms often include validation mechanisms that ensure accurate and properly formatted data entry, reducing errors and maintaining the integrity of the collected information.
4) **Personalization and Customization:** Through forms, users can personalize their experience by setting preferences and configuring application settings, contributing to a more tailored and user-centric environment.
5) **Workflow Management:** Forms play a key role in business applications by facilitating workflow and process management. Users can initiate actions or document steps in a structured manner, improving task efficiency.

- **Conclusion:**

  Therefore, we have learned how to construct an interactive form using a form widget, which has been applied to create the login screen for our application, TradeSphere's [E-Commerce].