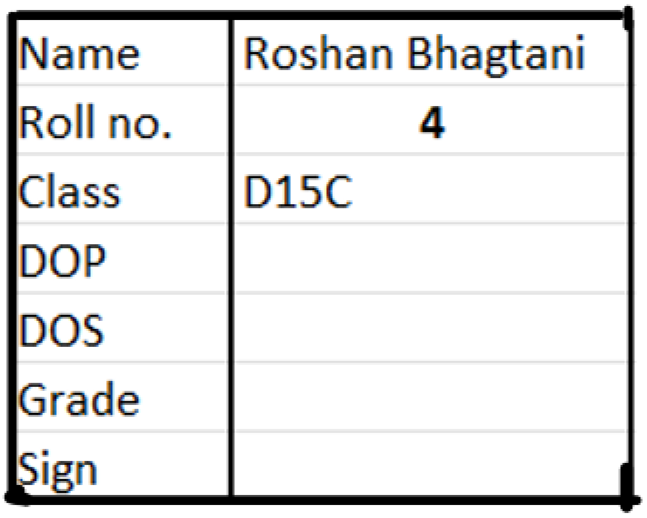
Experiment 04



Name	Roshan Bhagtani
Roll no.	**4**
Class	D15C
DOP	
DOS	
Grade	
Sign	

Aim: To create an interactive Form using form widget

Theory:

In Flutter, creating an interactive form typically involves using the `Form` widget, which is specifically designed to manage the state of form fields. The `Form` widget requires a `GlobalKey<FormState>` to control its state, making it easy to validate and save the form data. This key enables actions such as form validation, saving form data, and resetting the form. Within the `Form`, individual input fields are usually defined with the `TextFormField` widget, which provides features like `controller`, `validator`, and `onSaved`.

- **`TextFormField`**: This widget is used to capture user input. The `controller` property is used to manage the value of the input field, while the `validator` property ensures that the input satisfies certain conditions (e.g., non-empty, correct format, etc.). The `onSaved` property is used to store the value when the form is saved.

- **Form Validation**: The `validator` function within `TextFormField` checks if the input is valid. It returns a string message if the input is invalid, which is displayed as an error under the text field. For example, a simple check could validate whether a phone number is entered correctly or if a password meets specific requirements.

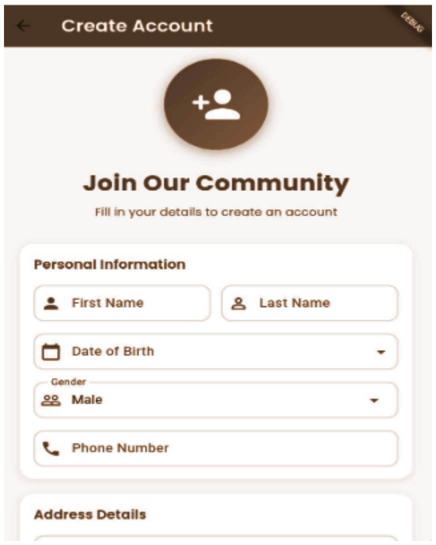
- **Saving Form Data**: When the form is submitted, the `validate()` method is invoked, which triggers the validation process. If all fields are valid, the `save()` method is called to store the values. The `onSaved` callback is useful to pass data from the form fields to variables or backend services.

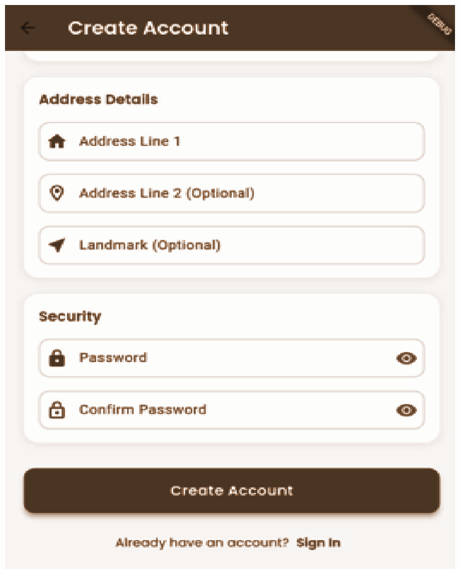
- **Handling State**: The `FormState` is responsible for managing the internal state of the form, such as whether fields are valid, whether they have been touched, and whether the form has been saved. It provides methods like `reset()`, `validate()`, and `save()` to manage these states effectively.

By using these tools, Flutter allows developers to build forms that are interactive, handle user input efficiently, and ensure data is valid before submission, creating a robust user experience for applications.

Implementation: date of birth, gender, phone number, and address. It includes input validation to ensure that required fields are filled in before submission. The form also handles password validation, ensuring that the password and confirmation password match before submitting. The `GlobalKey<FormState>` is used to manage the form's state, and animations are applied to make the user interface more engaging. The data entered in the form is then saved to Firebase Firestore, with error handling in place to display relevant messages in case of failure. The design is visually appealing, with custom input decorations, a hero animation for the logo, and a fade-in effect on the form for a smooth user experience.

Output:





Conclusion: The `SignupScreen` form implementation effectively combines user-friendly design with robust functionality for a seamless user registration experience. The use of `TextFormField` widgets, along with validation and custom input decorations, ensures that users can easily enter their information while the app maintains data integrity.