Name: Nilesh Balotiya Exp_no:05 Roll_no:04

Experiment No:05

Aim: To apply navigation, routing and gestures in Flutter App Gesture

Theory:

Navigation and Routing:

Navigation refers to the process of moving between different screens or pages within an application. Flutter provides a robust navigation framework that allows developers to manage the navigation flow efficiently. This framework includes two primary components: Navigator and Routes.

- Navigator: The Navigator manages a stack of Route objects and is responsible for pushing new routes onto the stack, popping routes off the stack, and transitioning between routes.
- Routes: Routes represent individual screens or pages in the application. Each route is associated with a unique identifier and contains the UI elements specific to that screen.

Flutter offers various navigation methods, such as push, pop, pushNamed, popAndPushNamed, etc., to navigate between routes. Developers can also define named routes to facilitate navigation and provide a clear structure to the application.

Gestures:

Gestures play a crucial role in enhancing user interaction within Flutter applications. Flutter provides a rich set of gesture recognizers that enable developers to detect and respond to user gestures effectively. Some common gestures include tap, long-press, drag, swipe, pinch, and rotate.

- GestureDetector: The GestureDetector widget in Flutter allows developers to detect various gestures and attach corresponding event handlers. Developers can wrap any widget with GestureDetector to make it interactive and responsive to user input.
- Gesture Recognizers: Flutter offers pre-built gesture recognizers such as TapGestureRecognizer, LongPressGestureRecognizer, DragGestureRecognizer, etc., which can be configured and customized according to the application's requirements.

Implementing Navigation, Routing, and Gestures in Flutter:

To incorporate navigation, routing, and gestures into a Flutter application, developers typically follow these steps:

1. Define routes: Define named routes and corresponding widgets for each screen/page in the application.
2. Configure navigation: Use Navigator methods to navigate between routes, passing data if necessary.
3. Implement gestures: Wrap interactive widgets with GestureDetector or utilize pre-built gesture recognizers to detect user gestures.
4. Handle gestures: Attach event handlers to gesture detectors or gesture recognizers to respond to user input appropriately.

5. Ensure accessibility: Consider accessibility guidelines and ensure that navigation and gestures are intuitive and accessible to all users, including those with disabilities.

Output:

```
 @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        title: Text("Login"),
      ),
      body: Center(
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 15),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text(
                "Login",
                style: TextStyle(fontSize: 27, fontWeight: FontWeight.bold),
              ),
              SizedBox(
                height: 30,
              ),
              FormContainerWidget(
                controller: _emailController,
                hintText: "Email",
                isPasswordField: false,
              ),
              SizedBox(
                height: 10,
              ),
              FormContainerWidget(
                controller: _passwordController,
                hintText: "Password",
                isPasswordField: true,
              ),
              SizedBox(
                height: 30,
              ),
```

// Implementation of Gesture

```
GestureDetector(
  onTap: () {
    _signIn();
  },
  child: Container(
    width: double.infinity,
    height: 45,
    decoration: BoxDecoration(
      color: Colors.blue,
      borderRadius: BorderRadius.circular(10),
    ),
    child: Center(
      child: _isSigning
          ? CircularProgressIndicator(
              color: Colors.white,
            )
          : Text(
              "Login",
              style: TextStyle(
                color: Colors.white,
                fontWeight: FontWeight.bold,
              ),
            ),
    ),
  ),
),
SizedBox(
  height: 10,
),

SizedBox(
  height: 20,
),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Text("Don't have an account?"),
    SizedBox(
      width: 5,
                  ),

//Implementaion of Navigation
                  GestureDetector(
                    onTap: () {
                      Navigator.pushAndRemoveUntil(
                        context,
                        MaterialPageRoute(builder: (context) => SignUpPage()),
                        (route) => false,
                      );
                    },
                    child: Text(
                      "Sign Up",
                      style: TextStyle(
                        color: Colors.blue,
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                  ),
                ],
              ),
            ],
          ),
        ),
      ),
    );
  }
```

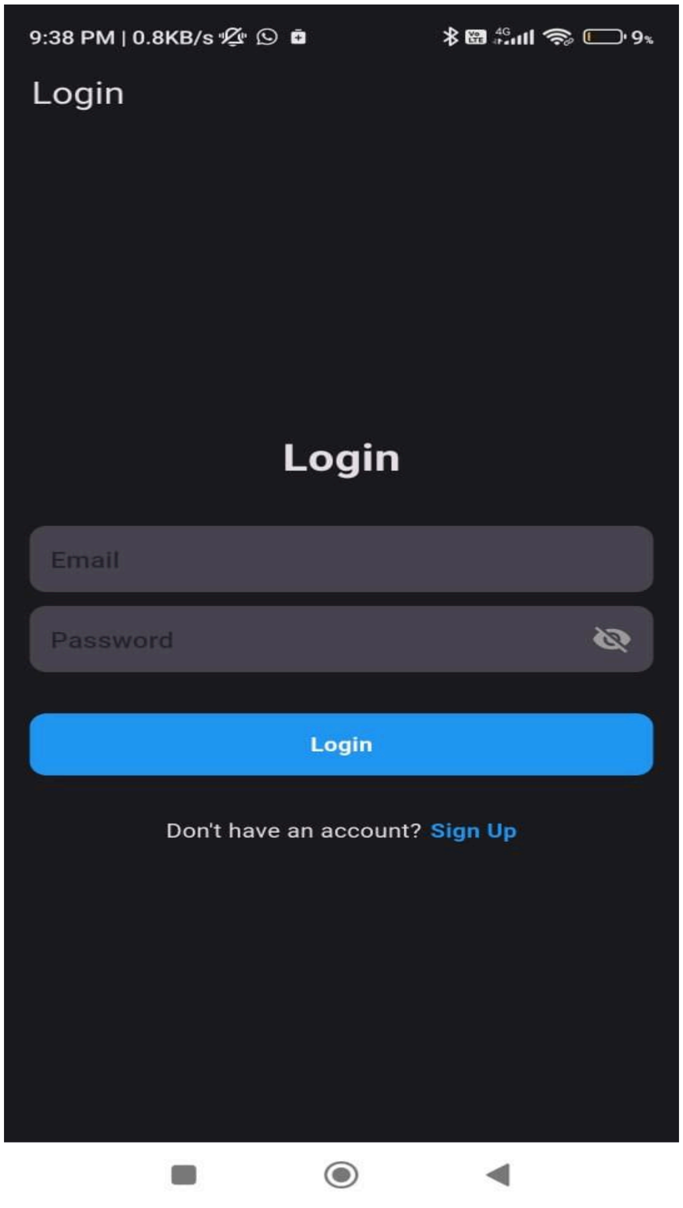
- **Navigation:**
  Navigation is performed when the user taps on the "Sign Up" text to navigate to the sign-up page.
  Here, the GestureDetector is used to detect the tap gesture, and when tapped, it navigates to the SignUpPage using Navigator.pushAndRemoveUntil.

- **Gesture:**
  Gestures are performed when the user taps on the "Login" button to initiate the sign-in process.
  In this section, the GestureDetector detects the tap gesture on the login button. When tapped, it calls the _signIn() function.

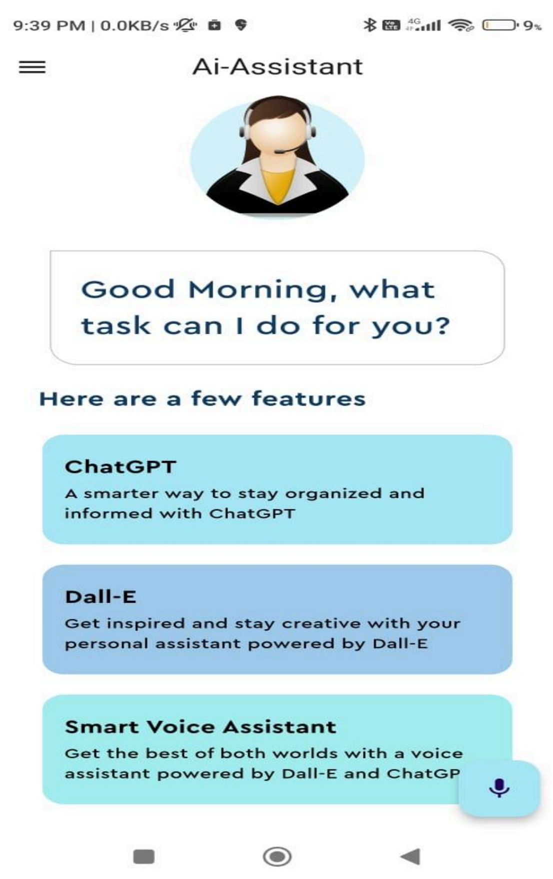
- **Routing:**
  Routing is implemented when setting up the navigation to the sign-up page.

Here, Navigator.pushAndRemoveUntil is used to navigate to the SignUpPage and remove all the previous routes from the stack, ensuring that the user cannot navigate back to the login screen using the back button.

Output:



Name: Nilesh Balotiya Exp_no:05 Roll_no:04


9:39 PM | 0.0KB/s

Ai-Assistant

Good Morning, what task can I do for you?

Here are a few features

ChatGPT
A smarter way to stay organized and informed with ChatGPT

Dall-E
Get inspired and stay creative with your personal assistant powered by Dall-E

Smart Voice Assistant
Get the best of both worlds with a voice assistant powered by Dall-E and ChatGP

Conclusion:

I have Successfully Implemented navigation, routing and gestures in Flutter App Gesture