Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Experiment 5

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

Theory:

Navigation

Navigation in Flutter involves transitioning between different screens or pages within an application. Flutter supports various navigation methods to suit different use cases:

- Imperative Navigation: Directly handles navigation actions using the Navigator class.
- Declarative Navigation: Manages navigation based on app state using the Router API.
- Named Routes: Uses predefined route names for cleaner and more scalable navigation.

Core Navigation Concepts

Navigator Widget

Acts as the core of Flutter's navigation system by maintaining a stack of route objects (screens). It provides methods like push() to add a new screen and pop() to return to the previous one. It also supports transition animations between routes.

Routes

Flutter offers two main types of routes:

- MaterialPageRoute: Uses Material Design transitions (Android-style).
- CupertinoPageRoute: Provides iOS-style transitions to match Apple's UI guidelines.

Navigation Methods

Basic Navigation

```
Navigator.push(context, MaterialPageRoute(builder: (context) => Screen2()));
```

Named Route Navigation

```
Navigator.pushNamed(context, '/screen2');
```

Returning Data from a Screen

```
Navigator.pop(context, returnValue);
```

Advanced Routing Techniques

Named Routes

Defined centrally within MaterialApp or CupertinoApp to organize navigation paths:

```
MaterialApp(
  routes: {
    '/': (context) => HomeScreen(),
    '/details': (context) => DetailsScreen(),
  },
)
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Route Guards
Custom logic can be added via onGenerateRoute to manage authentication, dynamic routing, and fallback for unknown routes (404 error handling).
Deep Linking
Flutter allows URL-based navigation to specific pages. This can be configured using onGenerateInitialRoutes for better integration with external links.

Gesture Detection
Common Gesture Widgets

- GestureDetector: Detects gestures like taps, drags, and pinch-to-zoom.
- InkWell: Adds visual feedback for taps, typically used with Material Design components.
- Dismissible: Enables swipe-to-dismiss functionality, often used with list items.

Gesture Types

- Tap: onTap, onDoubleTap
- Drag: onPanUpdate, onVerticalDrag
- Scale: onScaleUpdate

Custom Gestures
For more control, RawGestureDetector and custom GestureRecognizer classes can be used to define unique gesture interactions.

Navigation Patterns
Common Architectures

- Stack Navigation: Screens are placed on top of each other in a linear fashion.
- Tab Navigation: Allows switching between views using persistent tabs.
- Drawer Navigation: Provides a side panel menu for accessing various screens.
- Bottom Sheet Navigation: Displays modal screens from the bottom for temporary interaction.

State Management Integration
To maintain synchronization between UI and navigation logic, state management tools like Provider, Riverpod, or Bloc can be integrated effectively.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

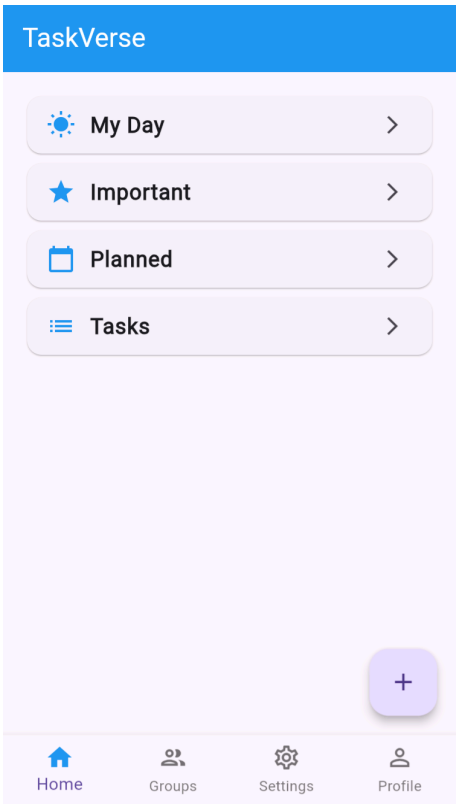
Division: D15C **Roll No: 28**

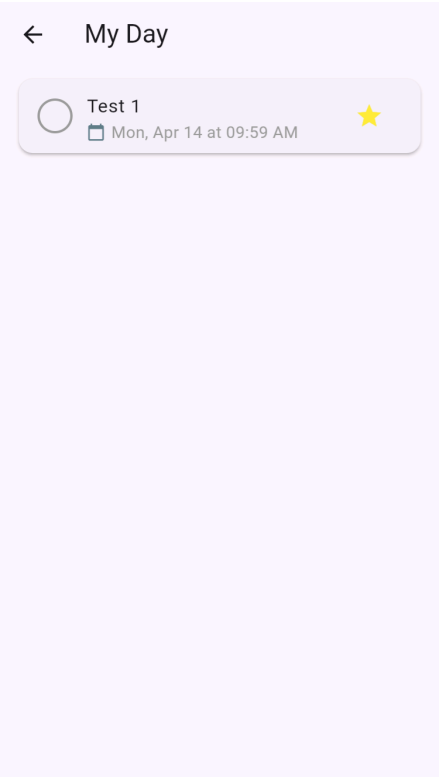
Code:

1. **Basic navigation**

```
class HomeScreen extends StatefulWidget {
 const HomeScreen({super.key});

 @override
 _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
 final FirestoreService _firestoreService = FirestoreService(); // Add Firestore service

 void _navigateTo(BuildContext context, Widget screen) {
  Navigator.push(
   context,
   MaterialPageRoute(builder: (context) => screen),
  );
 }
 void _addNewTask(Task newTask) {
     _firestoreService.saveTask(newTask); // Save to Firestore
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   body: ListView(
    padding: const EdgeInsets.all(16.0),
    children: [
     _buildSection(context, 'My Day', Icons.wb_sunny, MyDayScreen()),
     _buildSection(context, 'Important', Icons.star, ImportantScreen()),
     _buildSection(context, 'Planned', Icons.calendar_today, PlannedScreen()),
     _buildSection(context, 'Tasks', Icons.list, TasksScreen()),
    ],
   ),
   floatingActionButton: FloatingActionButton(
    onPressed: () {
     Navigator.push(
      context,
      MaterialPageRoute(
       builder: (context) => AddTaskScreen(
        onTaskAdded: _addNewTask,
       ),
```

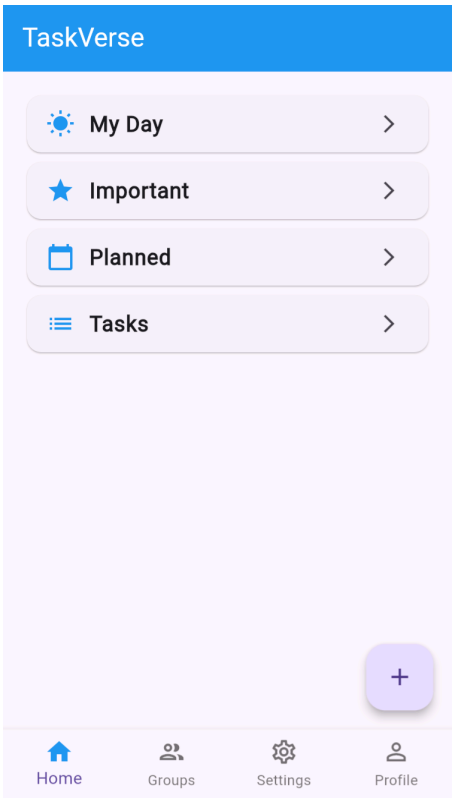
```
            ),
          );
        },
        child: const Icon(Icons.add),
      ),
    );
  }

  Widget _buildSection(BuildContext context, String title, IconData icon, Widget screen) {
    return Card(
      child: ListTile(
        leading: Icon(icon, color: Colors.blue),
                  title: Text(title, style: const TextStyle(fontSize: 18, fontWeight: FontWeight.w600)),
        trailing: const Icon(Icons.arrow_forward_ios, size: 18),
        onTap: () => _navigateTo(context, screen),
      ),
    );
  }
}
```

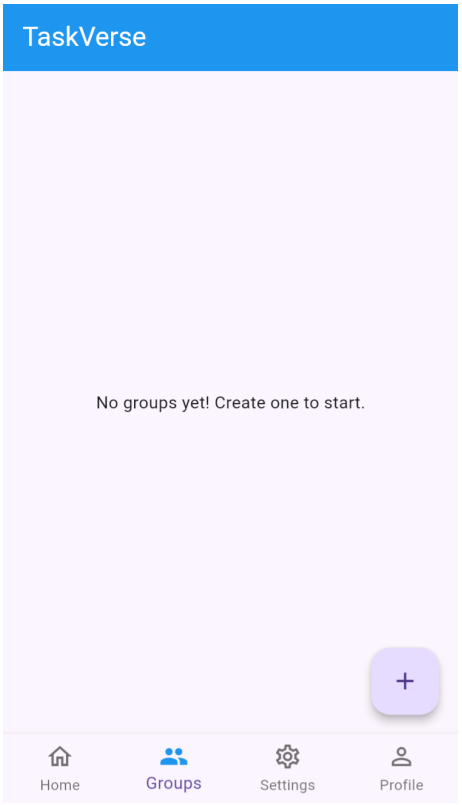




2. **Bottom Navigation**

```
bottomNavigationBar: Theme(
      data: Theme.of(context).copyWith(
        splashColor: Colors.transparent,
        highlightColor: Colors.transparent,
      ),
      child: BottomNavigationBar(
        type: BottomNavigationBarType.fixed,
        currentIndex: _selectedIndex,
        onTap: _onItemTapped,
        items: [
          BottomNavigationBarItem(
            icon: const Icon(Icons.home_outlined),
            activeIcon: const Icon(Icons.home, color: Colors.blue),
            label: AppLocalizations.of(context).home,
          ),
          BottomNavigationBarItem(
            icon: const Icon(Icons.group_outlined),
            activeIcon: const Icon(Icons.group, color: Colors.blue),
            label: AppLocalizations.of(context).groups,
          ),
          BottomNavigationBarItem(
            icon: const Icon(Icons.settings_outlined),
            activeIcon: const Icon(Icons.settings, color: Colors.blue),
            label: AppLocalizations.of(context).settings,
          ),
          BottomNavigationBarItem(
            icon: const Icon(Icons.person_outline),
            activeIcon: const Icon(Icons.person, color: Colors.blue),
            label: AppLocalizations.of(context).profile,
          ),
        ],
      ),
```



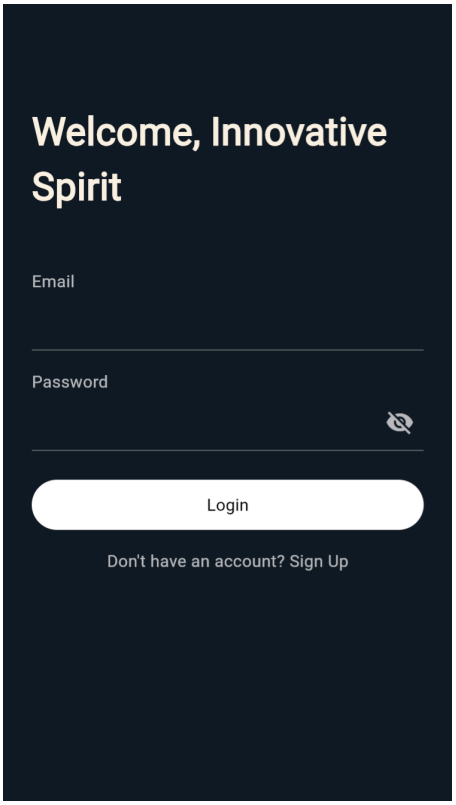


3. App Entry and loader screen navigation

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  await Future.delayed(Duration(seconds: 2)); // splash delay
  tz.initializeTimeZones();  //  Timezone setup


  runApp(const TaskVerseApp());
}

class TaskVerseApp extends StatefulWidget {
  const TaskVerseApp({super.key});

  @override
  State<TaskVerseApp> createState() => _TaskVerseAppState();
}

Future<bool> getLoginStatus() async {
  final prefs = await SharedPreferences.getInstance();
  return prefs.getBool('isLoggedIn') ?? false;
}
```

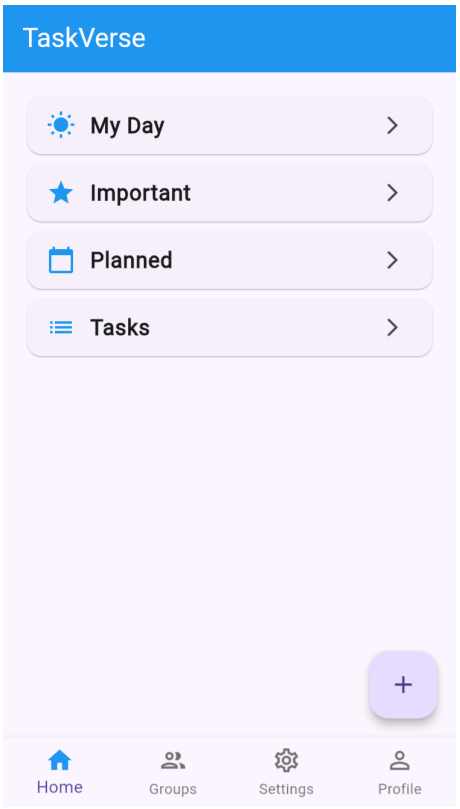
```
class _TaskVerseAppState extends State<TaskVerseApp> {
  Locale _appLocale = const Locale('en'); // Default language (English)
  ThemeMode _themeMode = ThemeMode.system; // Default to system theme

  void _changeLanguage(Locale locale) {
    setState(() {
      _appLocale = locale;
    });
  }

  void _changeTheme(ThemeMode themeMode) {
    setState(() {
      _themeMode = themeMode;
    });
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'TaskVerse',
      theme: ThemeData.light(),
      darkTheme: ThemeData.dark(),
      themeMode: _themeMode,
      locale: _appLocale,
      supportedLocales: const [
        Locale('en'),
        Locale('hi'),
        Locale('mr'),
      ],
      localizationsDelegates: const [
        AppLocalizations.delegate,
        GlobalMaterialLocalizations.delegate,
        GlobalWidgetsLocalizations.delegate,
        GlobalCupertinoLocalizations.delegate,
      ],
      home: FutureBuilder<bool>(
        future: getLoginStatus(),
        builder: (context, snapshot) {
          if (!snapshot.hasData) {
            return const SizedBox(); // Loading blank
```

```
}
return snapshot.data == true
   ? MainScreen(
       onLanguageChanged: _changeLanguage,
       onThemeChanged: _changeTheme,
     )
   : LoginPage(
       onLoginSuccess: () async {
         final prefs = await SharedPreferences.getInstance();
         await prefs.setBool('isLoggedIn', true);

         // Use pushAndRemoveUntil to clear stack and push new route
         Navigator.of(context).pushAndRemoveUntil(
           MaterialPageRoute(
             builder: (context) => MainScreen(
               onLanguageChanged: (_) {},
               onThemeChanged: (_) {},
             ),
           ),
           (Route<dynamic> route) => false, // remove all previous routes
         );
       },);},),);}}
```



Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**



GitHub Link: https://github.com/BKCODE2003/ToDo

Conclusion:

By applying navigation, routing, and gesture handling in a Flutter application, we have learned how to create smooth transitions between screens and build interactive user experiences. Understanding how to use Navigator, manage routes, and respond to gestures is essential for developing dynamic, multi-screen applications. These features play a key role in improving app usability and enabling intuitive user interactions.