Creando la navegación de una aplicación

En esta práctica, aprenderás que la navegación es un componente importante en una aplicación móvil. Una buena navegación crea una excelente experiencia de usuario (UX) al facilitar el acceso a la información. Por ejemplo, imagina que haces una entrada de diario y, cuando intentas seleccionar una etiqueta, no está disponible, por lo que debes crear una nueva. Cierra la entrada y ve a Configuración ➪ Etiquetas para agregar una nueva? Eso sería torpe. En cambio, el usuario necesita la capacidad de agregar una nueva etiqueta sobre la marcha y navegar apropiadamente para seleccionar o agregar una etiqueta desde su posición actual.

Al diseñar una aplicación, siempre ten en cuenta cómo el usuario navegaría a diferentes partes de la aplicación con la menor cantidad de toques.
La animación al navegar por diferentes páginas también es importante si ayuda a transmitir una acción, en lugar de simplemente ser una distracción. ¿Qué significa esto? El hecho de que puedas mostrar animaciones elegantes no significa que debas hacerlo. Utiliza animaciones para mejorar la UX, no frustrar al usuario.

Usando el navegador

El widget Navigator gestiona una pila de *rutas* para moverse entre páginas. Opcionalmente, puedes pasar datos a la página de destino y volver a la página original. Para comenzar a navegar entre páginas, utiliza los métodos Navigator.push, pushNamed y pop. (Aprenderás a utilizar el método pushNamed en la sección “Uso de la ruta del navegador con nombre” de esta práctica).

Navigator es increíblemente inteligente; muestra la navegación nativa en iOS o Android. Por ejemplo, en iOS cuando navegas a una nueva página, generalmente deslizas la página siguiente desde el lado derecho de la pantalla hacia la izquierda. En Android, cuando navegas a una nueva página, normalmente deslizas la página siguiente desde la parte inferior de la pantalla hacia la parte superior. Para resumir, en iOS, la nueva página se desliza desde la derecha y en Android, se desliza desde abajo.

El siguiente ejemplo te muestra cómo usar el método Navigator.push para navegar a la página Acerca de (About). El método push pasa los argumentos BuildContext y Route. Para enviar un nuevo argumento de ruta (Route), crea una instancia de la clase MaterialPageRoute que reemplaza la pantalla con la transición de animación de la plataforma adecuada (iOS o

Android). En el ejemplo, la propiedad fullscreenDialog se establece en true para presentar la página Acerca de, cómo un cuadro de diálogo modal de pantalla completa.
Al establecer la propiedad fullscreenDialog en true, la barra de la aplicación de la página Acerca de, incluye automáticamente un botón de cierre. En iOS, la transición de diálogo modal presenta la página deslizándose desde la parte inferior de la pantalla hacia la parte superior, y esto también es el predeterminado para Android.

```
Navigator.push(
   context,
   MaterialPageRoute(
       fullscreenDialog: true,
       builder: (context) => About(),
   ),
);
```

El siguiente ejemplo muestra cómo utilizar el método Navigator.pop para cerrar la página y volver a la página anterior. Tu llamas al método Navigator.pop (context) pasando el argumento BuildContext, y la página se cierra deslizándose desde la parte superior de la pantalla hacia la parte inferior. El segundo ejemplo muestra cómo pasar un valor a la página anterior.

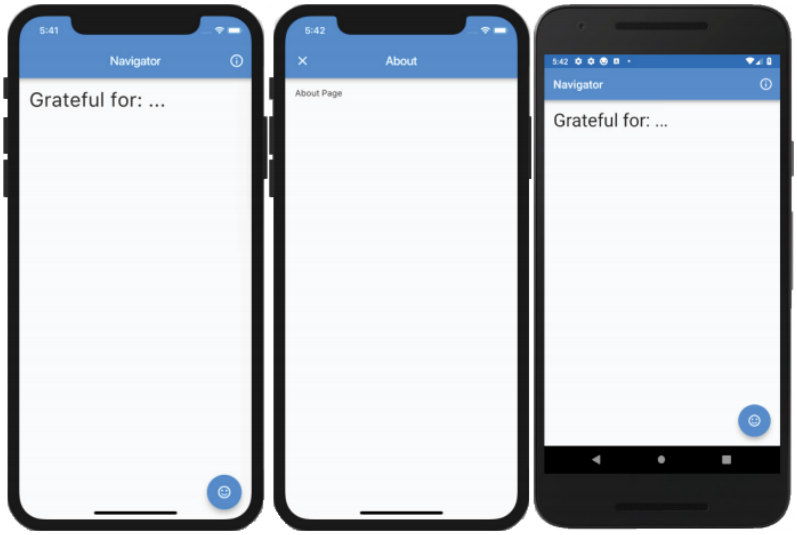
```
// Close page
Navigator.pop(context);

// Close page and pass a value back to previous page
Navigator.pop(context, 'Done');
```

Creación de la aplicación Navigator, parte 1: la página Acerca de

El proyecto tiene una página de inicio principal con FloatingActionButton para navegar a una página de gratitud pasando un valor de botón de opción seleccionado por defecto. La página de gratitud muestra tres botones de radio para seleccionar un valor y luego volver a pasarlo a la página de inicio y actualizar el widget de texto con el valor apropiado.

AppBar tiene un IconButton de acciones que navega a la página Acerca de, pasando un argumento fullscreenDialog establecido en true para crear un diálogo modal de pantalla completa. El diálogo modal muestra un botón de cierre en la parte superior izquierda de la página y se anima desde la parte inferior. En esta primera parte, desarrollarás la navegación desde la página principal a la página Acerca de, y viceversa.

1. Crea un nuevo proyecto Flutter y asígnale el nombre navigator. Consulta las instrucciones de la práctica 4. Para este proyecto, necesitas crear solo la carpeta de páginas.



2. Abre el archivo home.dart y agrega un IconButton a la lista de widgets de acciones de AppBar. La propiedad IconButton onPressed llamará al método _openPageAbout() y pasará argumentos de context y fullscreenDialog. No te preocupes por las líneas onduladas rojas debajo del nombre del método; crearás ese método en pasos posteriores. El widget Navigator necesita el argumento de context, y el argumento fullscreenDialog se establece en true para mostrar la página Acerca de, cómo modal de pantalla completa. Si estableces el argumento fullscreenDialog en falso, la página Acerca de, muestra una flecha hacia atrás en lugar de un icono de botón de cierre.

```
appBar: AppBar(
  title: Text('Navigator'),
  actions: <Widget>[
    IconButton(
      icon: Icon(Icons.info_outline),
      onPressed: () => _openPageAbout(
        context: context,
        fullscreenDialog: true,
      ),
    ),
  ],
),
```

3. Agrega un SafeArea con Padding como un hijo al cuerpo.

```
body: SafeArea(
  child: Padding(),
),
```

4. En el elemento secundario Padding, agrega un widget Text(), con el texto 'Greateful for: $_howAreYou'. Observa la variable $_howAreYou, que contendrá el valor

devuelto cuando navegues hacia atrás (Navigator.pop) desde la página de gratitud. Agrega una clase TextStyle con un valor de fontSize de 32,0 píxeles.

```
body: SafeArea(
  child: Padding(
    padding: EdgeInsets.all(16.0),
    child: Text('Grateful for: $_howAreYou', style: TextStyle(fontSize: 32.0),),
  ),
),
```

5. A la propiedad Scaffold floatingActionButton, agrega un widget FloatingActionButton() con un onPressed que llame al método _openPageGratitude (context: context), que pasa el argumento de contexto.

6. Para el elemento secundario FloatingActionButton(), agrega el icono llamado sentiment_satisfied, y para la información sobre herramientas (tooltip), agrega la descripción 'About'.

```
floatingActionButton: FloatingActionButton(
  onPressed: () => _openPageGratitude(context: context),
  tooltip: 'About',
  child: Icon(Icons.sentiment_satisfied),
)
```

7. En la primera línea después de la definición de la clase _HomeState, agrega una variable String llamada _howAreYou con un valor predeterminado de '...'.

```
String _howAreYou = "...";
```

8. Continúa agregando el método _openPageAbout() que acepta BuildContext y parámetros con nombre bool con los valores predeterminados establecidos en falso.

```
void _openPageAbout({BuildContext context, bool fullscreenDialog = false}) {}
```

9. En el método openPageAbout(), agrega un método Navigator.push() con un context y un segundo argumento de MaterialPageRoute(). La clase MaterialPageRoute() pasa el argumento fullscreenDialog y un constructor que llama a la página About() que tu crearas en pasos posteriores.

```
void _openPageAbout({BuildContext context, bool fullscreenDialog = false}) {
  Navigator.push(
    context,
    MaterialPageRoute(
      fullscreenDialog: fullscreenDialog,
      builder: (context) => About(),
    ),
  );
}
```

10. En la parte superior de la página home.dart, importa la página about.dart que crearás a continuación.

```
import 'about.dart';
```

Navigator es fácil de usar pero también poderoso. Examinemos cómo funciona. Navigator.push() pasa dos argumentos: context y MaterialPageRoute. Para el primer argumento, pasa el argumento de context. El segundo argumento, MaterialPageRoute(), le da la potencia necesaria para navegar a otra página usando una animación específica de la plataforma. Solo se requiere que el constructor navegue con el argumento opcional fullscreenDialog.

11. Crea un nuevo archivo llamado about.dart en la carpeta lib/pages. Dado que esta página solo muestra información, crea una clase StatelessWidget llamada About.

12. Para el cuerpo, agrega el SafeArea con relleno habitual (Padding) y la propiedad secundaria como un widget de texto (Text).

```
// about.dart
import 'package:flutter/material.dart';

class About extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('About'),
      ),
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Text('About Page'),
        ),
      ),
    );
  }
}
```

Agrega a AppBar un IconButton bajo la propiedad actions. La propiedad de icono para IconButton se establece en Icons.info_outline, el método _openPageAbout() pasa el context y el argumento fullscreenDialog se establece en verdadero. También agregas al Scaffold un FloatingActionButton que llama al método _openPageGratitude().

El método _openPageAbout() usa el navegador. push() para pasar el context y MaterialPageRoute. MaterialPageRoute pasa el argumento fullscreenDialog establecido en verdadero y el constructor llama a la página About(). La clase de página Acerca de, es un StatelessWidget con Scaffold y AppBar; la propiedad del cuerpo tiene un SafeArea con Padding como hijo que muestra un widget de texto con el texto "About Page".

Creación de la aplicación Navigator, parte 2: la página de gratitud

La segunda parte de la aplicación es navegar a la página de gratitud pasando un valor predeterminado para seleccionar el botón de opción apropiado. Una vez que regreses a la página de inicio, el valor del botón de opción recién seleccionado se devuelve y se muestra en el widget de texto.

1. Abre el archivo home.dart y, después del método _openPageAbout(), agrega el método _openPageGratitude(). El método _openPageGratitude() toma dos parámetros: un context y una variable bool fullscreenDialog con un valor predeterminado de falso. En este caso, la página de gratitud no es un diálogo de pantalla completa. Al igual que el MaterialPageRoute anterior, el constructor abre la página. En este caso, es la página de agradecimiento.

   Ten en cuenta que al pasar datos a la página de gratitud y esperar a recibir una respuesta, el método se marca como asíncrono para esperar una respuesta de Navigator.push utilizando la palabra clave await.

```
void _openPageGratitude(
    {BuildContext context, bool fullscreenDialog = false}) async {
  final String _gratitudeResponse = await Navigator.push(
```

   El constructor MaterialPageRoute crea el contenido de la ruta. En este caso, el contenido es la página de gratitud, que acepta un parámetro int radioGroupValue con un valor de -1. El valor -1 le dice a la página de la clase Gratitud que no seleccione ningún botón de opción. Si pasa un valor como 2, selecciona el botón de radio apropiado que corresponde a este valor.

```
builder: (context) => Gratitude(
    radioGroupValue: -1,
  ),
```

   Una vez que el usuario descarta la página de gratitud, se completa la variable _gratitudeResponse. Utiliza el ?? (doble signo de interrogación si es nulo) para comprobar el valor de _gratitudeResponse(no nulo) y completar la variable _howAreYou. El widget de texto se completa con el valor _howAreYou en la página de inicio con el valor de gratitud seleccionado apropiado o una cadena vacía.
   En otras palabras, si el valor de _gratitudeResponse no es nulo, la variable _howAreYou se rellena con el valor de_gratitudeResponse; de lo contrario, la variable _howAreYou se completa con una cadena vacía.

```
_howAreYou = _gratitudeResponse ?? '';
```

   Aquí está el código completo del método _openPageGratitude():

```
void _openPageGratitude(
    {BuildContext context, bool fullscreenDialog = false}) async {
  final String _gratitudeResponse = await Navigator.push(
    context,
    MaterialPageRoute(
      fullscreenDialog: fullscreenDialog,
      builder: (context) => Gratitude(
            radioGroupValue: -1,
          ),
    ),
  );
  _howAreYou = _gratitudeResponse ?? '';
}
```

2. En la parte superior de la página home.dart, agrega la página gratitude.dart que crearás a continuación.

```
import 'gratitude.dart';
```

3. Crea un nuevo archivo llamado gratitude.dart en la carpeta lib/pages. Dado que esta página modificará los datos (state), crea una clase StatefulWidget llamada Gratitude. Para recibir datos pasados desde la página de inicio, modifica la clase Gratitude agregando una variable int final llamada radioGroupValue. Ten en cuenta que la variable final no comienza con un guión bajo. Crea un constructor con nombre que requiera este parámetro. Se accede a la variable radioGroupValue mediante la clase _GratitudeState extends State<Gratitude> llamando a widget.radioGroupValue.

```
class Gratitude extends StatefulWidget {
  final int radioGroupValue;

  Gratitude({Key key, @required this.radioGroupValue}) : super(key: key);

  @override
  _GratitudeState createState() => _GratitudeState();
}
```

4. Para Scaffold AppBar, agrega un IconButton a la lista de acciones del Widget. Establece el icono IconButton en Icons.check con la propiedad onPressed llamando al Navigator.pop, que devuelve _selectedGratitude a la página de inicio.

```
appBar: AppBar(
  title: Text('Gratitude'),
  actions: <Widget>[
    IconButton(
      icon: Icon(Icons.check),
      onPressed: () => Navigator.pop(context, _selectedGratitude),
    ),
  ],
),
```

5. Para el cuerpo, agrega SafeArea y Padding con propiedad secundaria como una Fila (Row). La lista secundaria de Widget de Row contiene tres widgets de Radio y Text que se alternan. El widget Radio toma las propiedades value, groupValue y

onChanged. La propiedad de value es el valor de ID del botón de opción. La propiedad groupValue contiene el valor del botón de opción seleccionado actualmente. OnChanged pasa el valor de índice seleccionado al método personalizado _radioOnChanged() que maneja qué botón de opción está seleccionado actualmente. Después de cada botón de opción, hay un widget de texto que actúa como una etiqueta para el botón de opción.
Aquí está el código fuente de body completo:

```
body: SafeArea(
  child: Padding(
    padding: const EdgeInsets.all(16.0),
    child: Row(
      children: <Widget>[
        Radio(
          value: 0,
          groupValue: _radioGroupValue,
          onChanged: (index) => _radioOnChanged(index),
        ),
        Text('Family'),
        Radio(
          value: 1,
          groupValue: _radioGroupValue,
          onChanged: (index) => _radioOnChanged(index),
        ),
        Text('Friends'),
        Radio(
          value: 2,
          groupValue: _radioGroupValue,
          onChanged: (index) => _radioOnChanged(index),
        ),
        Text('Coffee'),
      ],
    ),
  ),
),
```

6. En la primera línea después de la definición de la clase _HomeState, agrega tres variables — _gratitudeList, _selectedGratitude y _radioGroupValue — y el método _radioOnChanged().
   - ➤ _gratitudeList es una List de valores String.
     List<String> _gratitudeList = List();
   - ➤ _selectedGratitude es una variable String que contiene el valor del botón Radio seleccionado.
     String _selectedGratitude;
   - ➤ _radioGroupValue es un int que contiene el ID del valor del botón Radio seleccionado.
     int _radioGroupValue;

7. Crea el método _radioOnChanged() tomando un int para el índice seleccionado del botón Radio. En el método, llama a setState() para que los widgets de Radio se actualicen con el value seleccionado. La variable _radioGroupValue se actualiza con el índice. La variable _selectedGratitude (valor de ejemplo Coffee) se actualiza tomando el valor de lista _gratitudeList[index] por el índice seleccionado (posición en la lista).

```
void _radioOnChanged(int index) {
  setState(() {
    _radioGroupValue = index;
    _selectedGratitude = _gratitudeList[index];
    print('_selectedRadioValue $_selectedGratitude');
  });
}
```

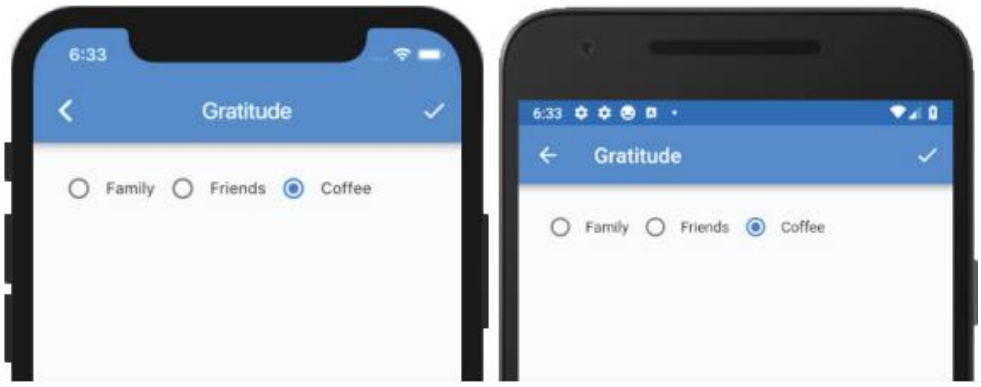
8. Anula initState() para inicializar _gratitudeList. Dado que _radioGroupValue se pasa desde la página de inicio, inicialízalo con widget.radioGroupValue, que es la última variable que se pasa desde la página de inicio.

```
gratitudeList..add('Family')..add('Friends')..add('Coffee');
_radioGroupValue = widget.radioGroupValue;
```

El siguiente es el código que declara todas las variables y métodos:

```
class _GratitudeState extends State<Gratitude> {
  List<String> _gratitudeList = List();
  String _selectedGratitude;
  int _radioGroupValue;

  void _radioOnChanged(int index) {
    setState(() {
      _radioGroupValue = index;
      _selectedGratitude = _gratitudeList[index];
      print('_selectedRadioValue $_selectedGratitude');
    });
  }

  @override
  void initState() {
    super.initState();

    _gratitudeList..add('Family')..add('Friends')..add('Coffee');
    _radioGroupValue = widget.radioGroupValue;
  }
```

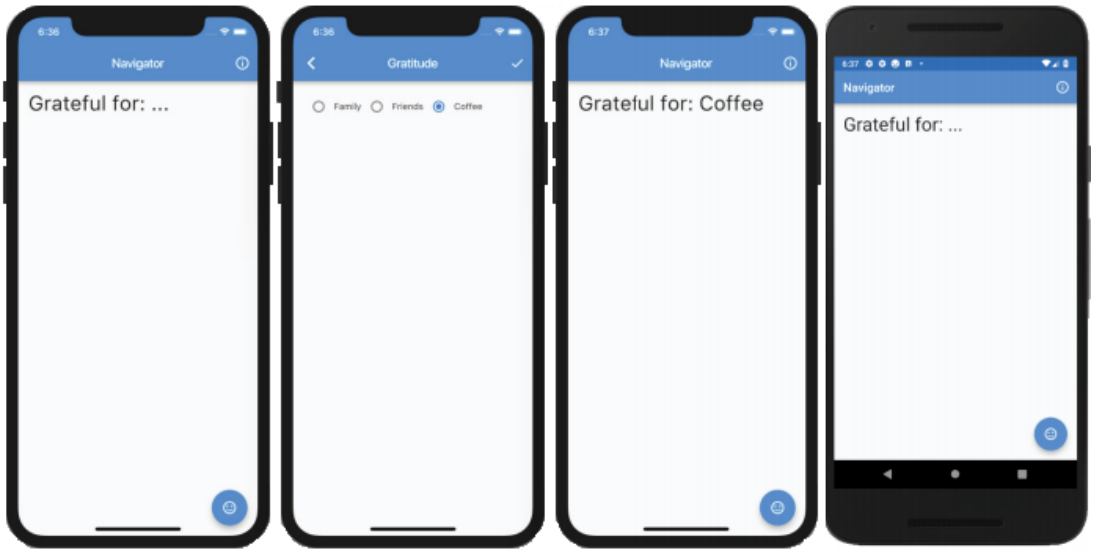
Aquí está todo el código fuente del archivo gratitude.dart (ver la siguiente página)



```
import 'package:flutter/material.dart';

class Gratitude extends StatefulWidget {
  final int radioGroupValue;

  Gratitude({Key key, @required this.radioGroupValue}) : super(key: key);

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

class _GratitudeState extends State<Gratitude> {
  List<String> _gratitudeList = List();
  String _selectedGratitude;
  int _radioGroupValue;

  void _radioOnChanged(int index) {
    setState(() {
      _radioGroupValue = index;
      _selectedGratitude = _gratitudeList[index];
      print('_selectedRadioValue $_selectedGratitude');
    });
  }
@override
void initState() {
  super.initState();

  _gratitudeList..add('Family')..add('Friends')..add('Coffee');
  _radioGroupValue = widget.radioGroupValue;
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Gratitude'),
      actions: <Widget>[
        IconButton(
          icon: Icon(Icons.check),
          onPressed: () => Navigator.pop(context, _selectedGratitude),
        ),
      ],
    ),
    body: SafeArea(
      child: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Row(
          children: <Widget>[
            Radio(
              value: 0,
              groupValue: _radioGroupValue,
              onChanged: (index) => _radioOnChanged(index),
            ),
            Text('Family'),
            Radio(
              value: 1,
              groupValue: _radioGroupValue,
              onChanged: (index) => _radioOnChanged(index),
            ),
            Text('Friends'),
            Radio(
              value: 2,
              groupValue: _radioGroupValue,
              onChanged: (index) => _radioOnChanged(index),
            ),
            Text('Coffee'),
          ],
        ),
      ),
    ),
  );
}
```

Tienes la aplicación completa creada con una página de inicio que puedes navegar a la página Acerca de, cómo un diálogo de pantalla completa. El fullscreenDialog le da a la página acerca de, un botón de acción de cierre predeterminado. Al tocar el FloatingActionButton de la página de inicio, el constructor Navigator MaterialPageRoute crea el contenido de la ruta, en este caso la página de gratitud. A través del constructor Gratitude, los datos se pasan a botones de opción no seleccionados.

Desde la página de gratitud, una lista de botones de radio ofrece la opción de seleccionar una gratitud. Al tocar el botón de acción AppBar (casilla de verificación IconButton), el método Navigator.pop devuelve el valor de gratitud seleccionado al widget de Texto de inicio. Desde la página de inicio, llamas al método Navigator.push utilizando la palabra clave await y el método ha estado esperando recibir un valor. Una vez que se llama al método Navigator.pop de la página Acerca de, devuelve un valor a la variable _gratitudeResponse de la página de inicio. El uso de la palabra clave await es una característica poderosa y sencilla de implementar.



Uso de la ruta del navegador con nombre

Una forma alternativa de utilizar Navigator es consultar la página a la que estás navegando por el nombre de la ruta. El nombre de la ruta comienza con una barra y luego viene el nombre de la ruta. Por ejemplo, el nombre de la ruta de la página Acerca de es '/about'. La lista de rutas está integrada en el widget MaterialApp().

Las rutas tienen un Mapa de String y WidgetBuilder donde String es el nombre de la ruta, y WidgetBuilder tiene un constructor para construir el contenido de la ruta por el Nombre de la clase (About) de la página para abrir.

```
routes: <String, WidgetBuilder>{
  '/about': (BuildContext context) => About(),
  '/gratitude': (BuildContext context) => Gratitude(),
},
```

Para llamar a la ruta, se llama al método Navigator.pushNamed() pasando dos argumentos. El primer argumento es el context y el segundo es el nombre de la ruta (route).

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

Usando Hero Animation

El widget Hero es una gran animación out-of-the-box para transmitir la acción de navegación de un widget que vuela a su lugar de una página a otra. La animación del héroe es una transición de elementos compartidos (animación) entre dos páginas diferentes.

Para visualizar la animación, imagina ver a un superhéroe volando en acción. Por ejemplo, tienes una lista de entradas del diario con una miniatura de foto, el usuario selecciona una entrada y ve la transición de la miniatura de la foto a la página de detalles moviéndose y creciendo a tamaño completo. La miniatura de la foto es el superhéroe y, cuando se toca, entra en acción al pasar de la página de la lista a la página de detalles y aterriza perfectamente en la ubicación correcta en la parte superior de la página de detalles que muestra la foto completa. Cuando se cierra la página de detalles, el widget Hero vuelve a la página, la posición y el tamaño originales. En otras palabras, la animación muestra la miniatura de la foto moviéndose y creciendo en su lugar desde la página de la lista a la página de detalles, y una vez que se descarta la página de detalles, la animación y el tamaño se invierten. El widget Hero tiene todas estas funciones integradas; no es necesario escribir código personalizado para manejar el tamaño y la animación entre páginas.

Para continuar con el escenario anterior, envuelve (wrap) el widget de imagen de la página de lista como un elemento secundario del widget de héroe y asigna un nombre de propiedad de etiqueta. Repite los mismos pasos para la página de detalles y asegúrate de que el valor de la propiedad de la etiqueta sea el mismo en ambas páginas.

```
// List page
Hero(
  tag: 'photo1',
  child: Image(
    image: AssetImage("assets/images/coffee.png"),
  ),
),

// Detail page
Hero(
  tag: 'photo1',
  child: Container(
    child: Image(
      image: AssetImage("assets/images/coffee.png"),
    ),
  ),
),
```

El widget secundario de héroe está marcado para la animación de héroe. Cuando el navegador presiona o abre un PageRoute, se reemplaza todo el contenido de la pantalla. Esto significa que, durante la transición de la animación, el widget Hero no se muestra en la posición original tanto en la ruta antigua como en la nueva, pero se mueve y cambia de tamaño de una página a otra. Cada etiqueta de héroe debe ser única y coincidir tanto en la página de origen como en la de destino.

Creación de la aplicación Hero Animation

En este ejemplo, el widget Hero tiene un Icon como hijo envuelto en un GestureDetector. También se podría usar un InkWell en lugar de GestureDetector para mostrar una animación de toque de material. El widget InkWell es un componente de material que responde a los gestos táctiles mostrando un efecto de salpicadura (ondulación).

Analizarás con más profundidad GestureDetector e InkWell más adelante. Cuando se toca el icono, se llama a Navigator.push para navegar a la pantalla de detalles, llamada Fly. Dado que la animación del widget Hero es como un superhéroe volando, la pantalla de detalles se llama Fly.

1. Crea un nuevo proyecto de Flutter y asígnale el nombre hero_animation. Nuevamente, puedes seguir las instrucciones de la práctica 4. Para este proyecto, solo necesitas crear la carpeta de páginas.

2. Abre el archivo home.dart y agregua al cuerpo un SafeArea con Padding como un hijo.

```
body: SafeArea(
  child: Padding(),
),
```

3. En el elemento secundario Padding, agregue el widget GestureDetector (), que creará a continuación.

```
body: SafeArea(
  child: Padding(
    padding: const EdgeInsets.all(16.0),
    child: GestureDetector(),
  ),
),
```

4. Agrega al hijo GestureDetector un widget Hero con la etiqueta 'format_paint'; La etiqueta puede ser cualquier ID único. El elemento secundario del widget Hero es un icono de format_paint con color verde claro y un valor de tamaño de 120.0 píxeles. Ten en cuenta que podrías haber utilizado un widget InkWell() en lugar de

GestureDetector(). El widget InkWell() muestra una retroalimentación de bienvenida cuando se toca, pero el widget GestureDetector() no mostrará la retroalimentación táctil. Para la propiedad onTap, llama a Navigator.push para abrir la página de detalles llamada Fly.

```
GestureDetector(
  child: Hero(
    tag: 'format_paint',
    child: Icon(
      Icons.format_paint,
      color: Colors.lightGreen,
      size: 120.0,
    ),
  ),
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => Fly()),
    );
  },
),
```

5. En la parte superior de la página home.dart, importa la página fly.dart que crearás a continuación.

```
import 'fly.dart';
```

Aquí está todo el código fuente del archivo Home.dart:

```
import 'package:flutter/material.dart';
import 'fly.dart';
class Home extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Hero Animation'),
      ),
      body: SafeArea(
        child: Padding(
          padding: EdgeInsets.all(16.0),
          child: GestureDetector(
            child: Hero(
              tag: 'format_paint',
              child: Icon(
                Icons.format_paint,
                color: Colors.lightGreen,
                size: 120.0,
              ),
            ),
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => Fly()),
              );
            },
          ),
        ),
      ),
    );
  }
}
```

6. Crea un nuevo archivo llamado fly.dart en la carpeta lib/pages. Dado que esta página solo muestra información, crea una clase StatelessWidget llamada Fly. Para el cuerpo, agrega el SafeArea habitual con un hijo configurado al widget Hero().

```
body: SafeArea(
  child: Hero(),
),
```

7. Para calcular el ancho del icono, después de la construcción del widget (BuildContext context) {, agrega una variable double llamada _width establecida por MediaQuery.of(context).size.shortestSide/2. La propiedad shortestSide devuelve el menor del ancho altura de la pantalla y se divide por dos para que sea la mitad del tamaño.

   La razón por la que calculas el ancho es solo para cambiar el tamaño del ancho del icono de acuerdo con el tamaño y la orientación del dispositivo. Si en su lugar utilizaras una imagen, este cálculo no sería necesario; se puede hacer usando BoxFit.fitWidth.

```
double _width = MediaQuery.of(context).size.shortestSide / 2;
```

8. Agrega al widget Hero una etiqueta de 'format_paint' con un contenedor para el hijo. Ten en cuenta que para que el widget Hero funcione correctamente, la etiqueta debe tener el mismo nombre que le diste en el widget Hero secundario de GestureDetector en el archivo home.dart. El elemento secundario del contenedor es un icono de pintura de formato con color verde claro y un valor de tamaño de la variable de ancho. Para la propiedad de alineación del contenedor, usa Alignment.bottomCenter. Puedes experimentar con diferentes valores de alineación para ver las variaciones de animación del héroe en funcionamiento.

```
Hero(
  tag: 'format_paint',
  child: Container(
    alignment: Alignment.bottomCenter,
    child: Icon(
      Icons.format_paint,
      color: Colors.lightGreen,
      size: _width,
    ),
  ),
)
```

   Aquí está todo el código fuente del archivo Fly.dart:

```
import 'package:flutter/material.dart';

class Fly extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    double _width = MediaQuery.of(context).size.shortestSide / 2;

    return Scaffold(
      appBar: AppBar(
        title: Text('Fly'),
      ),
      body: SafeArea(
        child: Hero(
          tag: 'format_paint',
          child: Container(
            alignment: Alignment.bottomCenter,
            child: Icon(
              Icons.format_paint,
              color: Colors.lightGreen,
              size: _width,
            ),
          ),
        ),
      ),
    );
  }
}
```

La animación del héroe es una poderosa animación incorporada para transmitir una acción animando automáticamente un widget de una página a otra al tamaño y posición correctos. En la página de inicio, declaras un GestureDetector con el widget Hero como widget secundario. El widget Hero establece un icono como hijo. El onTap llama al método Navigator.push (), que navega a la página Fly. Todo lo que necesitas hacer en la página Fly es declarar el widget que estás animando como hijo del widget Hero. Cuando regresas a la página de inicio, el héroe anima el icono a la posición original.

Uso de la barra de navegación inferior

BottomNavigationBar es un widget de Material Design que muestra una lista de BottomNavigationBarItems que contiene un icono y un título en la parte inferior de la página (figura 8.1). Cuando se selecciona BottomNavigationBarItem, se crea la página adecuada.

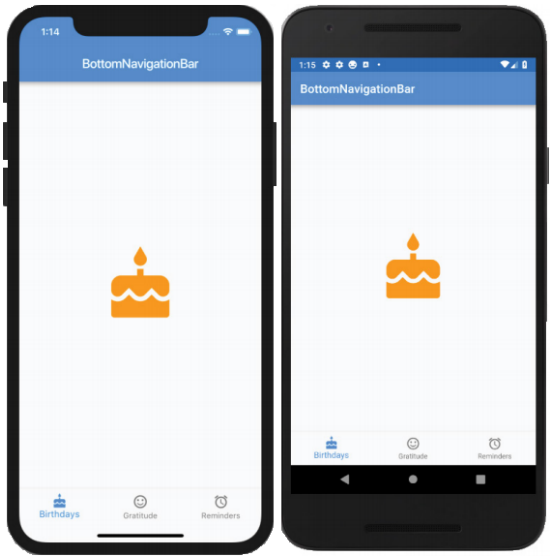


Figura 8.1: Barra de navegación inferior final con iconos y títulos.

Creación de la aplicación BottomNavigationBar

En este ejemplo, BottomNavigationBar tiene tres BottomNavigationBarItems que reemplazan la página actual por la seleccionada. Hay diferentes formas de mostrar la página seleccionada; utilizas una clase de widget como variable.

1. Crea un nuevo proyecto de Flutter y asígnale el nombre bottom_navigation_bar. Como siempre, puedes seguir las instrucciones del tema 4. Para este proyecto, solo necesitas crear la carpeta de páginas.

2. Abre el archivo home.dart y agrega al cuerpo un SafeArea con Padding cuando era hijo.

```
body: SafeArea(
   child: Padding(),
),
```

3. En el elemento secundario Padding, agrega la variable Widget _currentPage, que crearás a continuación. Ten en cuenta que esta vez estás utilizando una clase de widget para crear la variable _currentPage que contiene cada página seleccionada, ya sea la clase Gratitude, Reminders o Birthdays StatelessWidget.

```
body: SafeArea(
  child: Padding(
    padding: const EdgeInsets.all(16.0),
    child: _ currentPage,
  ),
),
```

4. Agrega a la propiedad bottomNavigationBar de Scaffold un widget BottomNavigationBar. Para la propiedad currentIndex, usa la variable _currentIndex, que se creará más adelante.

```
bottomNavigationBar: BottomNavigationBar(
        currentIndex: _currentIndex,
```

   La propiedad items es una List of BottomNavigationBarItems. Cada BottomNavigationBarItem toma una propiedad de icono y una propiedad de título (title).

5. Agrega a la propiedad items tres BottomNavigationBarItems con icon cake, sentiment_satisfied y access_alarm. Los títulos son 'Birthdays', 'Gratitude' y 'Reminders'.

```
items: [
  BottomNavigationBarItem(
    icon: Icon(Icons.cake),
    title: Text('Birthdays'),
  ),
```

6. Para la propiedad onTap, el callback devuelve el índice actual del elemento activo. Nombra la variable selectedIndex.

```
onTap: (selectedIndex) => _changePage(selectedIndex),
```

   Aquí está el código completo de BottomNavigationBar:

```
bottomNavigationBar: BottomNavigationBar(
  currentIndex: _currentIndex,
  items: [
    BottomNavigationBarItem(
      icon: Icon(Icons.cake),
      title: Text('Birthdays'),
    ),
    BottomNavigationBarItem(
      icon: Icon(Icons.sentiment_satisfied),
      title: Text('Gratitude'),
    ),
    BottomNavigationBarItem(
      icon: Icon(Icons.access_alarm),
      title: Text('Reminders'),
    ),
  ],
  onTap: (selectedIndex) => _changePage(selectedIndex),
),
```

7. Agrega el método _changePage(int selectedIndex) después de Scaffold(). El método _changePage() acepta un valor int del índice seleccionado. SelectedIndex se usa con el método setState() para establecer las variables _currentIndex y _currentPage. _CurrentIndex es igual a selectedIndex y _currentPage es igual a la página de List _listPages que corresponde al índice seleccionado.
   Es importante tener en cuenta que la variable Widget _currentPage muestra cada página seleccionada sin la necesidad de un widget Navigator. Este es un gran ejemplo del poder de personalizar los widgets según tus necesidades.

```
void _changePage(int selectedIndex) {
  setState(() {
    _currentIndex = selectedIndex;
    _currentPage = _listPages[selectedIndex];
  });
}
```

8. En la primera línea después de la definición de la clase _HomeState, agrega las variables _currentIndex, _listPages y _currentPage. La List _listPages contiene el nombre de clase de cada página.

```
int _currentIndex = 0;
List _listPages = List();
Widget _currentPage;
```

9. Anula initState() para agregar cada página a la List _listPages e inicializa _currentPage con la página de Birthdays(). Ten en cuenta el uso de la notación en cascada; los puntos dobles te permiten realizar una secuencia de operaciones en el mismo objeto.

```
@override
void initState() {
  super.initState();

  _listPages
    ..add(Birthdays())
    ..add(Gratitude())
    ..add(Reminders());
  _currentPage = Birthdays();
}
```

10. Agrega en la parte superior del archivo home.dart las importaciones de cada página que se creará a continuación.

```
import 'package:flutter/material.dart';
import 'gratitude.dart';
import 'reminders.dart';
import 'birthdays.dart';
```

    Aquí está el archivo home.dart completo:

```dart
import 'package:flutter/material.dart';
import 'gratitude.dart';
import 'reminders.dart';
import 'birthdays.dart';

class Home extends StatefulWidget {
  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  int _currentIndex = 0;
  List _listPages = List();
  Widget _currentPage;

  @override
  void initState() {
    super.initState();

    _listPages
      ..add(Birthdays())
      ..add(Gratitude())
      ..add(Reminders());
    _currentPage = Birthdays();
  }

  void _changePage(int selectedIndex) {
    setState(() {
      _currentIndex = selectedIndex;
      _currentPage = _listPages[selectedIndex];
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('BottomNavigationBar'),
      ),
      body: SafeArea(
        child: Padding(
          padding: EdgeInsets.all(16.0),
          child: _currentPage,
        ),
      ),
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: _currentIndex,
        items: [
          BottomNavigationBarItem(
            icon: Icon(Icons.cake),
            title: Text('Birthdays'),
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.sentiment_satisfied),
            title: Text('Gratitude'),
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.access_alarm),
            title: Text('Reminders'),
          ),
        ],
        onTap: (selectedIndex) => _changePage(selectedIndex),
      ),
    );
  }
}
```

11. Crea tres páginas de StatelessWidget y llámalas Birthdays, Gratitude y Reminders. Cada página tendrá un Scaffold con Center() para el cuerpo. El elemento secundario Center es un icono con un valor de tamaño de 120.0 píxeles y una propiedad de color. Aquí están los tres archivos de Dart, birthdays.dart, gratitude.dart y reminders.dart:

```
// birthdays.dart
import 'package:flutter/material.dart';

class Birthdays extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Icon(
          Icons.cake,
          size: 120.0,
          color: Colors.orange,
        ),
      ),
    );
  }
}

// gratitude.dart
import 'package:flutter/material.dart';

class Gratitude extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Icon(
          Icons.sentiment_satisfied,
          size: 120.0,
          color: Colors.lightGreen,
        ),
      ),
    );
  }
}

// reminders.dart
import 'package:flutter/material.dart';

class Reminders extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Icon(
          Icons.access_alarm,
          size: 120.0,
          color: Colors.purple,
        ),
      ),
    );
  }
}
```

La propiedad de elementos BottomNavigationBar tiene una lista de tres BottomNavigationBarItems. Para cada BottomNavigationBarItem, estableces una propiedad de icono (icon) y una propiedad de título (title). BottomNavigationBar onTap pasa el valor de índice seleccionado al método _changePage. El método _changePage usa setState() para configurar _currentIndex y _currentPage para mostrar. _CurrentIndex establece el BottomNavigationBarItem seleccionado, y _currentPage establece la página actual para que se muestre desde la lista _listPages.

Usando el BottomAppBar

El widget BottomAppBar se comporta de manera similar a BottomNavigationBar, pero tiene una muesca opcional en la parte superior. Al agregar un FloatingActionButton y habilitar la muesca, la muesca proporciona un agradable efecto 3D, por lo que parece que el botón está empotrado en la barra de navegación (figura 8.2).

Por ejemplo, para habilitar la muesca, establece la propiedad de forma BottomAppBar en una clase NotchedShape como la clase CircularNotchedRectangle() y establece la propiedad Scaffold floatingActionButtonLocation en FloatingActionButtonLocation.endDocked o centerDocked. Agrega a la propiedad Scaffold floatingActionButton un widget FloatingActionButton y el resultado mostrará el FloatingActionButton incrustado en el widget BottomAppBar, que es la muesca.

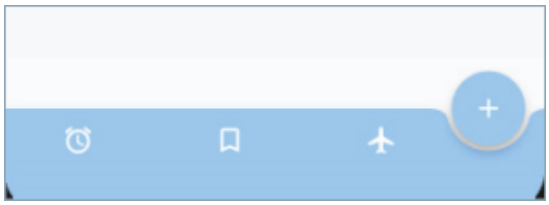


Figura 8.2: BottomAppBar con FloatingActionButton incrustado creando una muesca.

Creación de la aplicación BottomAppBar

En este ejemplo, BottomAppBar tiene una Fila (Row) como elemento secundario con tres IconButtons para mostrar los elementos de selección. El objetivo principal es utilizar un FloatingActionButton para acoplarlo al BottomAppBar con una muesca.
La muesca está habilitada por la propiedad de forma BottomAppBar establecida en CircularNotchedRectangle().

1. Crea un nuevo proyecto de Flutter y asígnale el nombre bottom_app_bar. Nuevamente, puedes seguir las instrucciones de la práctica 4. Para este proyecto, solo necesitas crear la carpeta de páginas.

2. Abre el archivo home.dart y agrega al cuerpo un SafeArea con un contenedor como hijo.

```
body: SafeArea(
  child: Container(),
),
```

3. Agrega un widget BottomAppBar() a la propiedad bottomNavigationBar de Scaffold.

```
bottomNavigationBar: BottomAppBar(),
```

4. Para habilitar la muesca, establece dos propiedades.
   - Primero establece la propiedad de forma BottomAppBar en CircularNotchedRectangle(). Establece la propiedad de color en Colors.blue.shade200 y agrega una Row como hijo.
   - A continuación, configura floatingActionButtonLocation, que manejarás en el paso 7.

```
bottomNavigationBar: BottomAppBar(
  color: Colors.blue.shade200,
  shape: CircularNotchedRectangle(),
  child: Row(),
),
```

5. Continúa agregando a la fila una propiedad mainAxisAlignment como MainAxisAlignment.spaceAround. La constante spaceAround permite que los IconButtons tengan un espacio uniforme entre ellos.

```
bottomNavigationBar: BottomAppBar(
  color: Colors.blue.shade200,
  shape: CircularNotchedRectangle(),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceAround,
    children: <Widget>[
    ],
```

```
    ),
  ),
```

6. Agrega tres IconButtons a la lista de hijos Row. Después del último IconButton, agrega un Divider() para agregar un espacio uniforme a la derecha, ya que FloatingActionButton está acoplado en el lado derecho de BottomAppBar. En lugar de un Divider(), podrías haber usado un contenedor con una propiedad de ancho (width).

```
bottomNavigationBar: BottomAppBar(
  color: Colors.blue.shade200,
  shape: CircularNotchedRectangle(),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceAround,
    children: <Widget>[
      IconButton(
        icon: Icon(Icons.access_alarm),
        color: Colors.white,
        onPressed: (){},
      ),
      IconButton(
        icon: Icon(Icons.bookmark_border),
        color: Colors.white,
        onPressed: (){},
      ),
      IconButton(
        icon: Icon(Icons.flight),
        color: Colors.white,
        onPressed: (){},
      ),
      Divider(),
    ],
  ),
),
```

7. Establece la ubicación de la muesca de la propiedad floatingActionButtonLocation en FloatingActionButtonLocation.endDocked. También puedes configurarlo en centerDocked.

```
floatingActionButtonLocation: FloatingActionButtonLocation.endDocked,
```

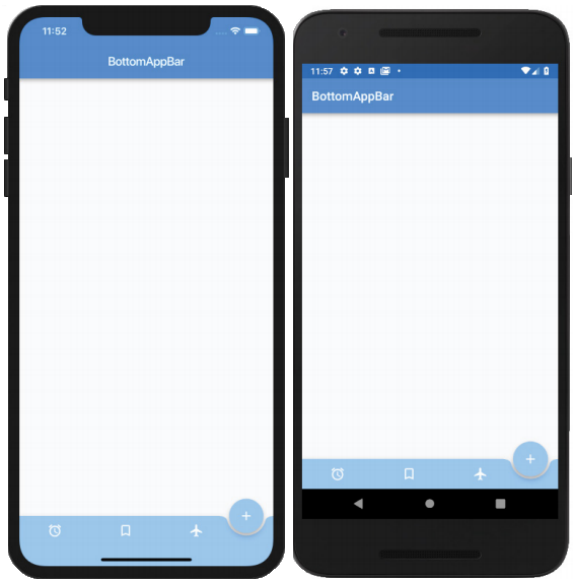
8. Agrega un FloatingActionButton a la propiedad floatingActionButton.

```
floatingActionButton: FloatingActionButton(
  backgroundColor: Colors.blue.shade200,
  onPressed: () {},
  child: Icon(Icons.add),
),
```

Aquí está todo el código fuente del archivo home.dart:

```
import 'package:flutter/material.dart';

class Home extends StatefulWidget {
  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('BottomAppBar'),
      ),
      body: SafeArea(
        child: Container(),
      ),
      bottomNavigationBar: BottomAppBar(
        color: Colors.blue.shade200,
        shape: CircularNotchedRectangle(),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceAround,
          children: <Widget>[
            IconButton(
              icon: Icon(Icons.access_alarm),
              color: Colors.white,
              onPressed: (){},
            ),
            IconButton(
              icon: Icon(Icons.bookmark_border),
              color: Colors.white,
              onPressed: (){},
            ),
            IconButton(
              icon: Icon(Icons.flight),
              color: Colors.white,
              onPressed: (){},
            ),
            Divider(),
          ],
        ),
      ),
      floatingActionButtonLocation: FloatingActionButtonLocation.endDocked,
      floatingActionButton: FloatingActionButton(
        backgroundColor: Colors.blue.shade200,
        onPressed: () {},
        child: Icon(Icons.add),
      ),
    );
  }
}
```

Para habilitar la muesca, se deben establecer dos propiedades para el widget Scaffold. La primera es usar un BottomAppBar con la propiedad de forma establecida en CircularNotchedRectangle(). La segunda es establecer la propiedad floatingActionButtonLocation en FloatingActionButtonLocation.endDocked o centerDocked.


11:52
BottomAppBar
11:57
BottomAppBar

Usando TabBar y TabBarView

El widget TabBar es un widget de Material Design que muestra una fila horizontal de pestañas. La propiedad de pestañas toma una lista de widgets y tu agregas pestañas usando el widget de pestaña. En lugar de usar el widget de pestaña, puedes crear un widget personalizado, que muestra el poder de Flutter. La pestaña seleccionada está marcada con una línea de selección inferior.

El widget TabBarView se utiliza junto con el widget TabBar para mostrar la página de la pestaña seleccionada. Los usuarios pueden deslizar el dedo hacia la izquierda o hacia la derecha para cambiar el contenido o tocar cada pestaña.

Tanto los widgets TabBar (figura 8.3) como TabBarView toman una propiedad de controlador de TabController. TabController es responsable de sincronizar las selecciones de pestañas entre TabBar y TabBarView. Dado que TabController sincroniza las selecciones de pestañas, debes declarar SingleTickerProviderStateMixin en la clase. En el tema 7, “Agregar animación a una aplicación”, aprendiste cómo implementar la clase Ticker que es impulsada (conducida) por el informe ScheduleBinding.scheduleFrameCallback una vez por cuadro de animación. Estás intentando sincronizar la animación para que sea lo más fluida posible.

Figura 8.3: TabBar en la propiedad bottomNavigationBar de Scaffold.

Creación de la aplicación TabBar y TabBarView

En este ejemplo, el widget TabBar es hijo de una propiedad bottomNavigationBar. Esto coloca la barra de pestañas en la parte inferior de la pantalla, pero también puedes colocarla en la barra de aplicaciones o en una ubicación personalizada. Cuando usas TabBar en combinación con TabBarView, una vez que se selecciona una pestaña, muestra automáticamente el contenido apropiado.
En este proyecto, el contenido está representado por tres páginas separadas. Crearás las mismas tres páginas que hiciste en el proyecto BottomNavigationBar.

1. Crea un nuevo proyecto de Flutter y asígnele el nombre tabbar. Una vez más, puedes seguir las instrucciones de la práctica 4. Para este proyecto, solo necesitas crear la carpeta de páginas.

2. Abre el archivo home.dart y agrega al cuerpo un SafeArea con TabBarView como hijo. La propiedad del controlador TabBarView es una variable TabController llamada _tabController. Agrega a la propiedad secundaria TabBarView las páginas de Birthdays(), Gratitude() y Reminders() que crearás en el paso 3

```
body: SafeArea(
  child: TabBarView(
    controller: _tabController,
    children: [
      Birthdays(),
      Gratitude(),
      Reminders(),
    ],
  ),
),
```

3. Esta vez, primero crearás las páginas a las que estás navegando. Como en la aplicación BottomNavigationBar, crea tres páginas StatelessWidget y llámalas Birthdays, Gratitude y Reminders. Cada página tiene un Scaffold con Center() para el cuerpo. El hijo del Center es un icono con el tamaño de 120.0 píxeles y un color. Los siguientes son los tres archivos de Dart, birthdays.dart, gratitude.dart y reminders.dart:

```
// birthdays.dart
import 'package:flutter/material.dart';

class Birthdays extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Icon(
          Icons.cake,
          size: 120.0,
          color: Colors.orange,
        ),
      ),
    );
  }
}

// gratitude.dart
import 'package:flutter/material.dart';

class Gratitude extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Icon(
          Icons.sentiment_satisfied,
          size: 120.0,
          color: Colors.lightGreen,
        ),
      ),
    );
  }
}
```

```dart
// reminders.dart
import 'package:flutter/material.dart';

class Reminders extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Icon(
          Icons.access_alarm,
          size: 120.0,
          color: Colors.purple,
        ),
      ),
    );
  }
}
```

4. Importa cada página en el archivo home.dart.

```dart
import 'package:flutter/material.dart';
import 'birthdays.dart';
import 'gratitude.dart';
import 'reminders.dart';
```

5. Declara el TickerProviderStateMixin a la clase _HomeState agregando with TickerProviderStateMixin. El argumento vsync AnimationController lo usará.

```dart
class _HomeState extends State<Home> with SingleTickerProviderStateMixin {...}
```

6. Declara una variable TabController con el nombre de _tabController. Reemplaza el método initState() para inicializar _tabController con el argumento vsync y un valor de longitud de 3.
   Esto hace referencia a vsync, es decir, esta referencia de la clase _HomeState. La longitud representa el número de pestañas que se mostrarán. Agrega un Oyente (Listener) al _tabController para detectar cuando se cambia una pestaña. Luego, anula el método dispose() para cuando la página se cierre para eliminar correctamente el _tabController.

   Ten en cuenta que en el método _tabChanged verifica indexIsChanging antes de mostrar qué pestaña se toca. Si no comprueba indexIsChanging, el código se ejecuta dos veces.

```
class _HomeState extends State<Home> with SingleTickerProviderStateMixin {
  TabController _tabController;

  @override
  void initState() {
    super.initState();

    _tabController = TabController(vsync: this, length: 3);
    _tabController.addListener(_tabChanged);
  }

  @override
  void dispose() {
    super.dispose();

  @override
  void dispose() {
    _tabController.dispose();
  super.dispose();
   }
     _tabController.dispose();
   }

   void _tabChanged() {
     // Check if Tab Controller index is changing, otherwise we get the notice twice
     if (_tabController.indexIsChanging) {
       print('tabChanged: ${_tabController.index}');
     }
   }
```

7. Agrega TabBar como elemento secundario de la propiedad bottomNavigationBar Scaffold.

```
bottomNavigationBar: SafeArea(
  child: TabBar(),
),
```

8. Pasa el _tabController para la propiedad del controlador TabBar. Personaliza labelColor y unselectedLabelColor usando, respectivamente, Colors.black54 y Colors.black38, pero siéntete libre de experimentar con diferentes colores.

```
bottomNavigationBar: SafeArea(
  child: TabBar(
    controller: _tabController,
    labelColor: Colors.black54,
    unselectedLabelColor: Colors.black38,
  ),
),
```

9. Agrega tres widgets de pestañas a la lista de widgets de pestañas. Personaliza cada ícono y texto de pestaña.

```
bottomNavigationBar: SafeArea(
  child: TabBar(
    controller: _tabController,
    labelColor: Colors.black54,
    unselectedLabelColor: Colors.black38,
    tabs: [
      Tab(
        icon: Icon(Icons.cake),
        text: 'Birthdays',
      ),
      Tab(
        icon: Icon(Icons.sentiment_satisfied),
        text: 'Gratitude',
      ),
      Tab(
        icon: Icon(Icons.access_alarm),
        text: 'Reminders',
      ),
    ],
  ),
),
```

Uso del Drawer y el Listview

Quizás te preguntes por qué estamos detallando ListView en esta práctica de navegación. Bueno, funciona muy bien con el widget Drawer. Los widgets ListView se utilizan con bastante frecuencia para seleccionar un elemento de una lista para navegar a una página detallada.

El Drawer es un Material Design que se desliza horizontalmente desde el borde izquierdo o derecho del Scaffold, la pantalla del dispositivo. Drawer se utiliza con la propiedad Scaffold drawer (left-side) o endDrawer (right-side). El Drawer se puede personalizar para cada necesidad individual, pero generalmente tiene un encabezado para mostrar una imagen o información fija y un ListView para mostrar una lista de páginas navegables. Normalmente, se utiliza un Drawer cuando la lista de navegación tiene muchos elementos.

Para configurar el encabezado Drawer, tienes dos opciones integradas, UserAccountsDrawerHeader o DrawerHeader. UserAccountsDrawerHeader está diseñado para mostrar los detalles del usuario de la aplicación estableciendo las propiedades currentAccountPicture, accountName, accountEmail, otherAccountsPictures y decoration.

```
// User details
UserAccountsDrawerHeader(
  currentAccountPicture: Icon(Icons.face,),
  accountName: Text('Sandy Smith'),
  accountEmail: Text('sandy.smith@domainname.com'),
  otherAccountsPictures: <Widget>[
    Icon(Icons.bookmark_border),
  ],
  decoration: BoxDecoration(
    image: DecorationImage(
      image: AssetImage('assets/images/home_top_mountain.jpg'),
      fit: BoxFit.cover,
    ),
  ),
),
```

DrawerHeader está diseñado para mostrar información genérica o personalizada estableciendo el relleno (padding), el elemento secundario (child), la decoración (decoration) y otras propiedades.

```
// Generic or custom information
DrawerHeader(
   padding: EdgeInsets.zero,
   child: Icon(Icons.face),
   decoration: BoxDecoration(color: Colors.blue),
),
```

El constructor estándar de ListView te permite crear una lista corta de elementos rápidamente. En la próxima práctica profundizaremos en cómo usar ListView. Ver la figura 8.4.

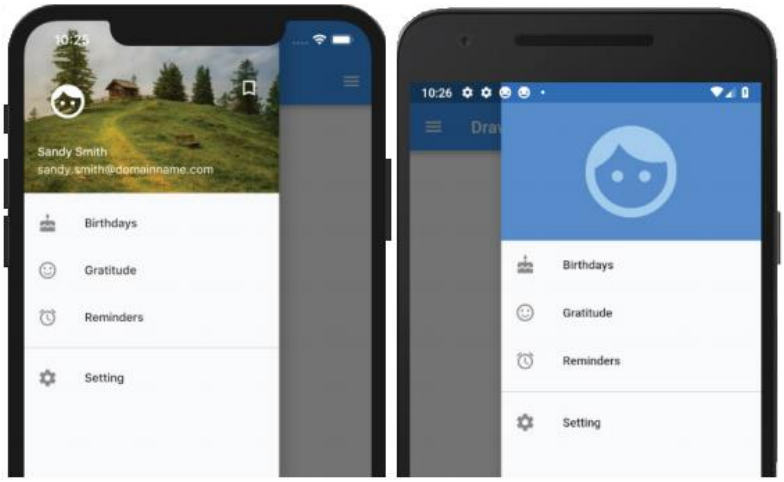


Figura 8.4: Drawer y ListView.

Creación de la aplicación Drawer

En este ejemplo, Drawer se agrega a la propiedad drawer o endDrawer del Scaffold. Las propiedades Drawer y endDrawer deslizan el Drawer de izquierda a derecha (TextDirection.ltr) o de derecha a izquierda (TextDirection.rtl). En este ejemplo, agregarás el drawer y el endDrawer para mostrar cómo usar ambos. Utiliza UserAccountsDrawerHeader para la propiedad de drawer (lado izquierdo) y DrawerHeader para la propiedad endDrawer (lado derecho).

Usa ListView para agregar el contenido de Drawer y ListTile para alinear el texto y los iconos de la lista del menú fácilmente. En este proyecto, usa el constructor ListView estándar ya que tiene una pequeña lista de elementos de menú.

1. Crea un nuevo proyecto de Flutter y asígnale el nombre drawer, siguiendo las instrucciones del tema 4. Para este proyecto, debes crear las páginas, los widgets y las carpetas de assets/images. Copia la imagen home_top_mountain.jpg en la carpeta assets/images.

2. Abre el archivo pubspec.yaml y en assets agrega la carpeta de imágenes.

```
# To add assets to your application, add an assets section, like this:
assets:
 - assets/images/
```

3. Agrega los recursos de la carpeta y las imágenes de la subcarpeta en la raíz del proyecto y luego copia el archivo home_top_mountain.jpg en la carpeta de imágenes.

4. Haz clic en el botón Guardar; dependiendo del editor que estés usando, automáticamente ejecuta los paquetes de flutter get. Una vez terminado, muestra un mensaje de Proceso terminado con el código de salida 0. Si no ejecuta automáticamente el comando tú mismo, abre la ventana Terminal (ubicada en la parte inferior de su editor) y escribe flutter packages get.

5. Crea las páginas a las que estás navegando primero. Crea tres páginas de StatelessWidget y llámalas Birthdays, Gratitude y Reminders. Cada página tiene un Scaffold con Center() para el cuerpo. El Center secundario es un icono con un valor de tamaño de 120.0 píxeles y una propiedad de color.

```
class Birthdays extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Icon(
          Icons.cake,
          size: 120.0,
          color: Colors.orange,
        ),
      ),
    );
  }
}
```

6. Agrega la AppBar al Scaffold, que es necesario para navegar de regreso a la página de inicio. A continuación se muestran los tres archivos de Dart, birthdays.dart, gratitude.dart y reminders.dart:

```
// birthdays.dart
import 'package:flutter/material.dart';

class Birthdays extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Birthdays'),
      ),
      body: Center(
        child: Icon(
          Icons.cake,
          size: 120.0,
          color: Colors.orange,
        ),
      ),
    );
  }
}
```

```
// gratitude.dart
import 'package:flutter/material.dart';

class Gratitude extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Gratitude'),
      ),
      body: Center(
        child: Icon(
          Icons.sentiment_satisfied,
          size: 120.0,
          color: Colors.lightGreen,
        ),
      ),
    );
  }
}

// reminders.dart
import 'package:flutter/material.dart';

class Reminders extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Reminders'),
      ),
      body: Center(
        child: Icon(
          Icons.access_alarm,
          size: 120.0,
          color: Colors.purple,
        ),
      ),
    );
  }
}
```

7. Los widgets de Drawer izquierdo y derecho comparten la misma lista de menú, y tú lo escribirás primero. Crea un nuevo archivo de Dart en la carpeta de widgets. Haz clic con el botón derecho en la carpeta de widgets y luego selecciona New ➪ Dart File, introduce menu_list_tile.dart y haz clic en el botón OK para guardar.

8. Importa las clases material.dart, birthdays.dart, gratitude.dart y reminders.dart (páginas). Agrega una nueva línea y luego comienza a escribir st; se abre la ayuda de autocompletado, así que selecciona la abreviatura completa y asígnale un nombre de MenuListTileWidget.

```
import 'package:flutter/material.dart';
import 'package:drawer/pages/birthdays.dart';
import 'package:drawer/pages/gratitude.dart';
import 'package:drawer/pages/reminders.dart';
```

9. La construcción del widget (BuildContext context) devuelve una columna. La lista de widgets de Column child contiene varios ListTiles que representan cada elemento

del menú. Agrega un widget Divider con la propiedad de color establecida en Colors.grey antes del último widget ListTile.

```
@override
Widget build(BuildContext context) {
  return Column(
    children: <Widget>[
      ListTile(),
      ListTile(),
      ListTile(),
      Divider(color: Colors.grey),
      ListTile(),
    ],
  );
}
```

10. Para cada ListTile, establece la propiedad inicial como un icono y la propiedad del título como un texto. Para la propiedad onTap, primero llama a Navigator.pop() para cerrar el cajón abierto y luego llama a Navigator.push() para abrir la página seleccionada.

```
@override
Widget build(BuildContext context) {
  return Column(
    children: <Widget>[
      ListTile(
        leading: Icon(Icons.cake),
        title: Text('Birthdays'),
        onTap: () {
          Navigator.pop(context);
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => Birthdays(),
            ),
          );
        },
      ),
      ListTile(
        leading: Icon(Icons.sentiment_satisfied),
        title: Text('Gratitude'),
        onTap: () {
          Navigator.pop(context);
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => Gratitude(),
            ),
          );
        },
      ),
      ListTile(
        leading: Icon(Icons.alarm),
        title: Text('Reminders'),
```

```
            onTap: () {
              Navigator.pop(context);
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => Reminders(),
                ),
              );
            },
          ),
          Divider(color: Colors.grey),
          ListTile(
            leading: Icon(Icons.settings),
            title: Text('Setting'),
            onTap: () {
              Navigator.pop(context);
            },
          ),
        ],
      );
    }
```

11. Crea un nuevo archivo de Dart en la carpeta de widgets. Haz clic con el botón derecho en la carpeta de widgets y luego selecciona New ➪ Dart File, introduce left_drawer.dart y haz clic en el botón OK para guardar.

12. Importa la biblioteca material.dart y la clase menu_list_tile.dart.

```
import 'package:flutter/material.dart';
import 'package:drawer/widgets/menu_list_tile.dart';
```

13. Agrega una nueva línea y luego comienza a escribir st; se abre la ayuda de autocompletado. Selecciona la abreviatura completa y asígnale un nombre de LeftDrawerWidget.

```
class LeftDrawerWidget extends StatelessWidget {
  const LeftDrawerWidget({
    Key key,
  }) : super(key: key);
  @override
  Widget build(BuildContext context) {
     return Drawer();
  }
}
```

14. La construcción del widget (BuildContext context) devuelve un Drawer. El elemento secundario Drawer es un widget de lista de elementos secundarios ListView de un widget UserAccountsDrawerHeader y una llamada a la clase de widget const MenuListTileWidget(). Para llenar todo el espacio del Drawer, establece la propiedad de relleno ListView en EdgeInsets.zero.

```
@override
Widget build(BuildContext context) {
  return Drawer(
    child: ListView(
      padding: EdgeInsets.zero,
      children: <Widget>[
        UserAccountsDrawerHeader(),
        const MenuListTileWidget(),
      ],
    ),
  );
}
```

15. Para UserAccountsDrawerHeader, establece las propiedades currentAccountPicture, accountName, accountEmail, otherAccountsPictures y decoración.

```
@override
Widget build(BuildContext context) {
  return Drawer(
    child: ListView(
      padding: EdgeInsets.zero,
      children: <Widget>[
        UserAccountsDrawerHeader(
          currentAccountPicture: Icon(
            Icons.face,
            size: 48.0,
            color: Colors.white,
          ),
          accountName: Text('Sandy Smith'),
          accountEmail: Text('sandy.smith@domainname.com'),
          otherAccountsPictures: <Widget>[
            Icon(
              Icons.bookmark_border,
              color: Colors.white,
            )
          ],
          decoration: BoxDecoration(
            image: DecorationImage(
              image: AssetImage('assets/images/home_top_mountain.jpg'),
              fit: BoxFit.cover,
            ),
          ),
        ),
        const MenuListTileWidget(),
      ],
    ),
  );
}
```

16. Crea un nuevo archivo de Dart en la carpeta de widgets. Haz clic con el botón derecho en la carpeta de widgets y luego selecciona New ⇨ Dart File, ingresa right_drawer.dart y haz clic en el botón OK para guardar. Importa la biblioteca material.dart y la clase menu_list_tile.dart.

```
import 'package:flutter/material.dart';
import 'package:drawer/widgets/menu_list_tile.dart';
```

17. Agrega una nueva línea y luego comienza a escribir st; se abre la ayuda de autocompletado. Selecciona la abreviatura completa y asígnale un nombre de RightDrawerWidget.

```
class RightDrawerWidget extends StatelessWidget {
  const RightDrawerWidget({
    Key key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Drawer();
  }
}
```

18. La construcción del widget (BuildContext context) devuelve un Drawer. El hijo Drawer es una lista de hijos ListView de un widget DrawerHeader y una llamada a la clase de widget const MenuListTileWidget(). Para llenar todo el espacio del Drawer, establece la propiedad de relleno ListView en EdgeInsets.zero.

```
@override
Widget build(BuildContext context) {
  return Drawer(
    child: ListView(
      padding: EdgeInsets.zero,
      children: <Widget>[
        DrawerHeader(),
        const MenuListTileWidget(),
      ],
    ),
  );
}
```

19. Para DrawerHeader, establece las propiedades padding, child y decoration.

```
@override
Widget build(BuildContext context) {
  return Drawer(
    child: ListView(
      padding: EdgeInsets.zero,
      children: <Widget>[
        DrawerHeader(
          padding: EdgeInsets.zero,
          child: Icon(
            Icons.face,
            size: 128.0,
            color: Colors.white54,
          ),
          decoration: BoxDecoration(color: Colors.blue),
        ),
        const MenuListTileWidget(),
      ],
    ),
  );
}
```

20. Abre el archivo home.dart e importa las clases material.dart, birthdays.dart, gratitude.dart y reminders.dart.

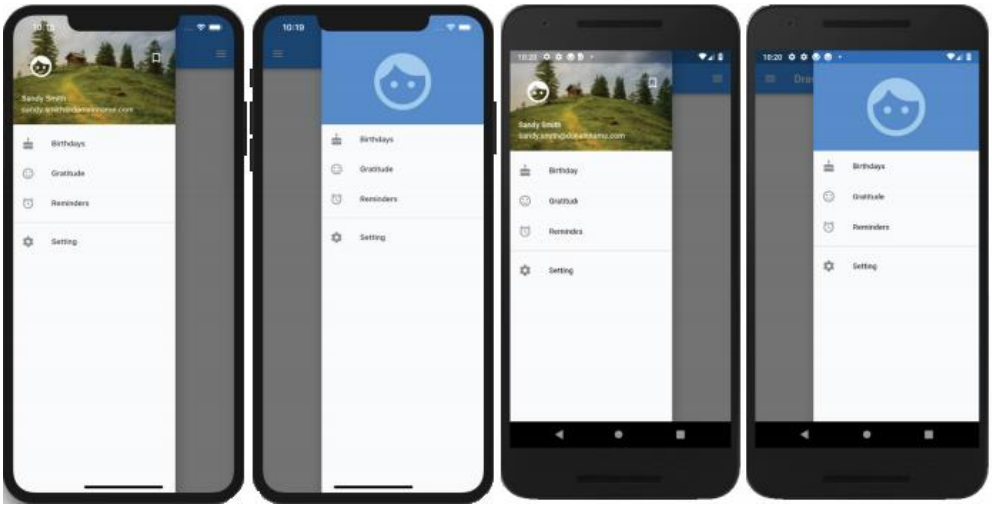
```
import 'package:flutter/material.dart';
import 'birthdays.dart';
import 'gratitude.dart';
import 'reminders.dart';
```

21. Agrega al cuerpo un SafeArea con un contenedor como un hijo.

```
body: SafeArea(
  child: Container(),
),
```

22. Agrega a la propiedad del Scaffold drawer una llamada a la clase de widget LeftDrawerWidget() y para la propiedad endDrawer una llamada a la clase de widget RightDrawerWidget().
    Ten en cuenta que usas la palabra clave const antes de llamar a cada clase de widget para aprovechar el almacenamiento en caché y la reconstrucción de subárboles para un mejor rendimiento.

```
return Scaffold(
  appBar: AppBar(
    title: Text('Drawer'),
  ),
  drawer: const LeftDrawerWidget(),
  endDrawer: const RightDrawerWidget(),
  body: SafeArea(
    child: Container(),
  ),
);
```



Para agregar un Drawer a una aplicación, establece la propiedad Scaffold drawer o endDrawer. Las propiedades Drawer y endDrawer deslizan el Drawer de izquierda a derecha (TextDirection.ltr) o de derecha a izquierda (TextDirection.rtl).

El widget Drawer toma una propiedad secundaria y le pasamos un ListView. El uso de ListView te permite crear una lista de elementos de menú desplazable. Para la lista de widgets secundarios de ListView, creaste dos clases de widgets, una para crear el encabezado Drawer y otra para crear la lista de elementos del menú. Para configurar el encabezado del Drawer, tienes dos opciones, UserAccountsDrawerHeader o DrawerHeader. Estos dos widgets te

permiten configurar fácilmente el contenido del encabezado según los requisitos. Observamos dos ejemplos para el encabezado Drawer llamando a la clase de widget apropiada LeftDrawerWidget() o RightDrawerWidget().

Para la lista de elementos del menú, usaste la clase de widget MenuListTileWidget(). Esta clase devuelve un widget de columna que usa ListTile para construir tu lista de menú. El widget ListTile te permite establecer el icono principal, el título y las propiedades onTap. La propiedad onTap llama a Navigator.pop() para cerrar Drawer y llama a Navigator.push() para navegar a la página seleccionada.

Esta aplicación es un excelente ejemplo de cómo crear un árbol de widgets poco profundo utilizando clases de widgets y separándolos en archivos individuales para una máxima reutilización. También anidaste clases de widgets con las clases de widgets izquierda y derecha, ambas llamando a la clase de lista de menú.

Resumen

En esta práctica, aprendiste a usar el widget Navigator para administrar una pila de rutas a fin de permitir la navegación entre páginas. Opcionalmente, pasaste datos a la página de navegación y volviste a la página original. La animación del héroe permite que una transición de widget se coloque en su lugar de una página a otra. El widget para animar desde y hacia está envuelto en un widget Hero mediante una llave única.

Usaste el widget BottomNavigationBar para mostrar una lista horizontal de BottomNavigationBarItems que contiene un icono y un título en la parte inferior de la página. Cuando el usuario toca cada BottomNavigationBarItem, se muestra la página correspondiente. Para mejorar el aspecto de una barra de navegación inferior, usaste el widget BottomAppBar y habilitaste la muesca opcional. La muesca es el resultado de incrustar un FloatingActionButton en un BottomAppBar estableciendo la forma BottomAppBar en una clase CircularNotchedRectangle() y estableciendo la propiedad Scaffold floatingActionButtonLocation en FloatingActionButtonLocation.endDocked.

El widget TabBar muestra una fila horizontal de pestañas. La propiedad de pestañas toma una lista de widgets y las pestañas se agregan usando el widget de pestaña. El widget TabBarView se utiliza junto con el widget TabBar para mostrar la página de la pestaña seleccionada. Los usuarios pueden deslizar el dedo hacia la izquierda o hacia la derecha para cambiar el contenido o tocar cada pestaña. La clase TabController manejó la sincronización de TabBar y seleccionó TabBarView. TabController requiere el uso de SingleTickerProviderStateMixin en la clase.

El widget Drawer permite al usuario deslizar un panel de izquierda a derecha. El widget Drawer se agrega configurando la propiedad Scaffold drawer o endDrawer. Para alinear fácilmente los elementos del menú en una lista, pasaste un ListView como hijo del Drawer. Dado que esta lista de menú es corta, usaste el constructor ListView estándar en lugar de un constructor ListView, que se trata en el siguiente tema. Tienes dos opciones de encabezado de drawer prediseñado, UserAccountsDrawerHeader o DrawerHeader. Cuando el usuario toca uno de los elementos del menú, la propiedad onTap llama a Navigator.pop() para cerrar Drawer y llama a Navigator.pop() para navegar a la página seleccionada.

En la próxima práctica, aprenderás a usar diferentes tipos de listas. Echarás un vistazo a ListView, GridView, Stack, Card y CustomScrollView para usar Slivers.