Creación de efectos y listas de desplazamiento

En esta práctica, aprenderás a crear listas de desplazamiento que ayuden a los usuarios a ver y seleccionar información.

Comenzarás con el widget de tarjeta (Card) en esta práctica porque se usa comúnmente junto con widgets con capacidad de lista para mejorar la interfaz de usuario (UI) y los datos de grupo. En la práctica anterior, analizaste el uso del constructor básico para ListView y, en esta, usarás ListView.builder para personalizar los datos. El widget GridView es un widget fantástico que muestra una lista de datos por un número fijo de mosaicos (grupos de datos) en el eje transversal.

El widget Stack se usa comúnmente para superponer, colocar y alinear widgets para crear una apariencia personalizada. Un buen ejemplo es un carrito de compras con el número de artículos a comprar en la parte superior derecha.

El widget CustomScrollView te permite crear efectos de desplazamiento personalizados mediante el uso de una lista de widgets de astillas (slivers). Las astillas son útiles, por ejemplo, si tienes una entrada de diario con una imagen en la parte superior de la página y la descripción del diario a continuación. Cuando el usuario desliza el dedo para leer más, el desplazamiento de la descripción es más rápido que el desplazamiento de la imagen, creando un efecto de paralaje.

Usando la tarjeta

El widget Card es parte de Material Design y tiene esquinas y sombras redondeadas mínimas. Para agrupar y diseñar datos, la tarjeta es un widget perfecto para mejorar el aspecto de la interfaz de usuario.

El widget Card se puede personalizar con propiedades como elevación, forma, color, margen y otras. La propiedad de elevación es un valor de double y cuanto mayor sea el número, mayor será la sombra proyectada.

Aprendiste previamente, que un double es un número que requiere precisión del punto decimal, como 8.50. Para personalizar la forma y los bordes del widget Card, modifica la propiedad de la forma. Algunas de las propiedades de forma son StadiumBorder, UnderlineInputBorder, OutlineInputBorder y otras.

```
Card(
  elevation: 8.0,
  color: Colors.white,
  margin: EdgeInsets.all(16.0),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      Text('
        'Barista',
        textAlign: TextAlign.center,
        style: TextStyle(
          fontWeight: FontWeight.bold,
          fontSize: 48.0,
          color: Colors.orange,
        ),
      '),
      Text(
        'Travel Plans',
        textAlign: TextAlign.center,
        style: TextStyle(color: Colors.grey),
      ),
    ],
  ),
),
```

Las siguientes son algunas formas de personalizar la propiedad de forma de la tarjeta (figura 9.1).

```
// Create a Stadium Border
shape: StadiumBorder(),

// Create Square Corners Card with a Single Orange Bottom Border
shape: UnderlineInputBorder(borderSide: BorderSide(color: Colors.deepOrange)),

// Create Rounded Corners Card with Orange Border
shape: OutlineInputBorder(borderSide: BorderSide(color: Colors.deepOrange.
withOpacity(0.5)),),
```

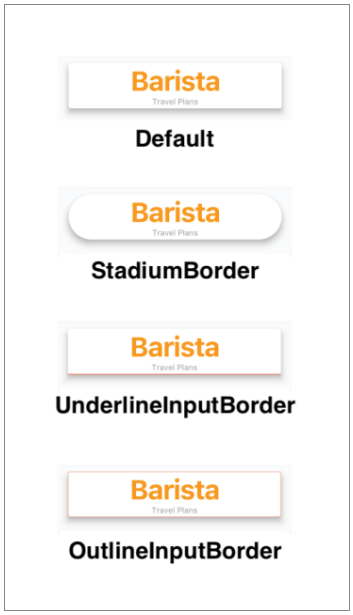


Figura 9.1: Personalizaciones de tarjeta.

Usando ListView y ListTile

El constructor ListView.builder se utiliza para crear una lista de widgets desplazable lineal bajo demanda (figura 9.2). Cuando tienes un gran conjunto de datos, se llama al constructor solo para los widgets visibles, lo que es excelente para el rendimiento. Dentro del constructor, utilizas la devolución de llamada (callback) de itemBuilder para crear la lista de widgets secundarios. Ten en cuenta que solo se llama a itemBuilder si el argumento itemCount es mayor que cero y se llama tantas veces como el valor itemCount. Recuerda, la Lista comienza en la fila 0, no 1. Si tienes 20 elementos en la Lista, se repite desde la fila 0 a la 19. El argumento scrollDirection predeterminado es Axis.vertical pero puede cambiarse a Axis.horizontal.

El widget ListTile se usa comúnmente con el widget ListView para formatear y organizar fácilmente iconos, títulos y descripciones en un diseño lineal. Entre las propiedades principales se encuentran las propiedades iniciales, finales, de título y de subtítulos, pero hay otras. También puedes usar las devoluciones de llamada (callback) onTap y onLongPress para ejecutar una acción cuando el usuario toca ListTile. Por lo general, las propiedades iniciales y finales se implementan con iconos, pero puedes agregar cualquier tipo de widget.

En la figura 9.2, los primeros tres ListTiles muestran un icono para la propiedad inicial, pero para la propiedad final, un widget Text muestra un valor porcentual. Los ListTiles restantes muestran las propiedades iniciales y finales como iconos. Otro escenario es usar la propiedad de subtítulos para mostrar una barra de progreso en lugar de una descripción de texto adicional, ya que estas propiedades aceptan widgets.

El primer ejemplo de código aquí muestra una tarjeta con la propiedad secundaria como ListTile para darle un bonito efecto de marco y sombra. El segundo ejemplo muestra un ListTile base.

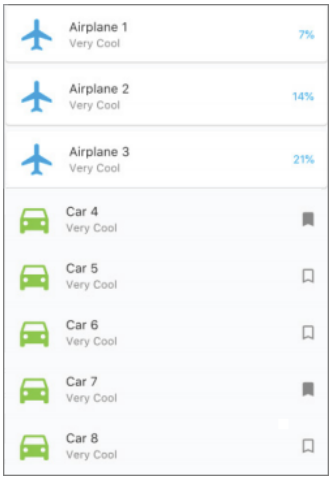


Figura 9.2: Diseño lineal de ListView usando ListTile.

```
// Card with a ListTile widget
Card(
  child: ListTile(
    leading: Icon(Icons.flight),
    title: Text('Airplane $index'),
    subtitle: Text('Very Cool'),
    trailing: Text('${index * 7}%'),
    onTap: ()=> print('Tapped on Row $index'),
  ),
);

// ListTile
ListTile(
  leading: Icon(Icons.directions_car),
  title: Text('Car $index'),
  subtitle: Text('Very Cool'),
  trailing: Icon(Icons.bookmark),
  onTap: ()=> print('Tapped on Row $index'),
);
```

Creación de la aplicación ListView

En este ejemplo, el widget ListView usa el constructor para mostrar una tarjeta para el encabezado y dos variaciones de ListTile para la lista de datos. ListTile puede mostrar widgets iniciales y finales. La propiedad inicial muestra un icono, pero podría haber mostrado una imagen. Para la propiedad final, el primer tipo de ListTile muestra datos como un porcentaje y el segundo ListTile muestra un icono de marcador seleccionado o no seleccionado. También establece un título y subtítulo, y para onTap usa la declaración de impresión para mostrar el valor del índice de la fila tocada.

1. Crea un nuevo proyecto de Flutter y asígnale el nombre listview. Puedes seguir las instrucciones de la práctica 4. Para este proyecto, necesitas crear solo las páginas y las carpetas de widgets. Crea la clase de inicio (Home Class) como un widget sin estado (StatelessWidget), ya que los datos no requieren cambios.

2. Abre el archivo home.dart y agrega al cuerpo un SafeArea con ListView.builder() como hijo.

```
body: SafeArea(
   child: ListView.builder(),
),
```

3. Configura ListView.builder con el argumento itemCount establecido en 20. Para este ejemplo, especifica 20 filas de datos. Para el callback de itemBuilder, pasa el BuildContext y el índice del widget como un valor int.
   Para mostrar cómo crear diferentes tipos de widgets para enumerar cada fila de datos, verifiquemos en la primera fila el valor de índice de cero y llamemos a la clase de widget HeaderWidget (index: index). Esta clase muestra una tarjeta con el texto "Barista Travel Plan".

Para la primera, segunda y tercera filas, se llama a la clase de widget RowWithCardWidget (index: index) para mostrar un ListTile como hijo de una Card. Para el resto de las filas, llama a la clase de widget RowWidget (index: index) para mostrar un ListTile predeterminado.
Es importante comprender que las clases de widgets a las que llamas desde itemBuilder crean un widget único con el valor de índice pasado. ItemBuilder repite el valor itemCount, en este ejemplo 20 veces.
Crearás las tres clases de widgets en el paso 5.

```
body: SafeArea(
  child: ListView.builder(
    itemCount: 20,
    itemBuilder: (BuildContext context, int index) {
      if (index == 0) {
        return HeaderWidget(index: index);
      } else if (index >= 1 && index <= 3) {
        return RowWithCardWidget(index: index);
      } else {
        return RowWidget(index: index);
      }
    },
  ),
),
```

4. Agrega en la parte superior del archivo las declaraciones de importación para las clases de widgets header.dart, row_with_card.dart y row.dart que crearás a continuación.

```
import 'package:flutter/material.dart';
import 'package:listview/widgets/header.dart';
import 'package:listview/widgets/row_with_card.dart';
import 'package:listview/widgets/row.dart';
```

5. 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 header.dart y haz clic en el botón OK para guardar.

6. Importa la biblioteca material.dart, agrega una nueva línea y luego comienza a escribir st; se abre la ayuda de autocompletado, así que selecciona la abreviatura stless (StatelessWidget) y asígnale un nombre de HeaderWidget.

7. Modifica la clase de widget HeaderWidget para devolver un contenedor. Importa la biblioteca material.dart.
   El contenedor hijo es una carta con una elevación de 8.0 para mostrar una sombra profunda. La lista de widgets secundarios de la tarjeta devuelve dos widgets de texto. Dejamos a propósito tres tipos de formas comentadas para que las pruebes y veas cómo cambian la forma y los bordes de la Tarjeta.

Queríamos señalar que ListView itemBuilder en el archivo main.dart llama a esta clase, el HeaderWidget (index: index) para cada elemento de fila. Para cada fila, se crea un widget y se agrega al árbol de widgets.
Ten en cuenta que comentamos tres formas diferentes de personalizar la forma predeterminada de la tarjeta para tus pruebas.

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

class HeaderWidget extends StatelessWidget {
  const HeaderWidget({
    Key key,
    @required this.index,
  }) : super(key: key);

  final int index;

  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.all(16.0),
      height: 120.0,
      child: Card(
        elevation: 8.0,
        color: Colors.white,
        //shape: StadiumBorder(),
        //shape: UnderlineInputBorder(borderSide: BorderSide(color: Colors
.deepOrange)),
        //shape: OutlineInputBorder(borderSide: BorderSide(color:
Colors.deepOrange.withOpacity(0.5)),),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'Barista',
              textAlign: TextAlign.center,
              style: TextStyle(
                fontWeight: FontWeight.bold,
                fontSize: 48.0,
                color: Colors.orange,
              ),
            ),
            Text(
              'Travel Plans',
              textAlign: TextAlign.center,
              style: TextStyle(color: Colors.grey),
            ),
          ],
        ),
      ),
    );
  }
}
```

8. 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 row_with_card.dart y haz clic en el botón OK para guardar.

9. Importa la biblioteca material.dart, agrega una nueva línea y luego comienza a escribir st; se abre la ayuda de autocompletado, así que selecciona la abreviatura stless (StatelessWidget) y asígnale el nombre RowWithCardWidget.

10. Modifica el widget de clase RowWithCardWidget para devolver una Card. Importa la biblioteca material.dart. El hijo de la tarjeta es un ListTile, que es ideal para alinear contenido fácilmente. Para la propiedad principal (leading), devuelve un icono (Icon). La propiedad final (trailing) devuelve un widget de texto con interpolación de cadenas que toma el índice multiplicado por siete para obtener un número. La propiedad del título devuelve un widget de texto con el valor del índice. La propiedad de subtítulos devuelve un widget de texto. Para la propiedad onTap, usa una declaración de impresión para mostrar el índice de fila tocada.
    Como recordatorio, se crea un widget y se agrega al árbol de widgets para cada fila.

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

class RowWithCardWidget extends StatelessWidget {
  const RowWithCardWidget({
    Key key,
    @required this.index,
  }) : super(key: key);

  final int index;

  @override
  Widget build(BuildContext context) {
    return Card(
      child: ListTile(
        leading: Icon(
          Icons.flight,
          size: 48.0,
          color: Colors.lightBlue,
        ),
        title: Text('Airplane $index'),
        subtitle: Text('Very Cool'),
        trailing: Text(
          '${index * 7}%',
          style: TextStyle(color: Colors.lightBlue),
        ),
        //selected: true,
        onTap: () {
          print('Tapped on Row $index');
        },
      ),
    );
  }
}
```

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, ingresa row.dart y haz clic en el botón OK para guardar.

12. Importa la biblioteca material.dart, agrega una nueva línea y luego comienza a escribir st; se abre la ayuda de autocompletado, así que selecciona la abreviatura stless (StatelessWidget) y asígnale un nombre de RowWidget.

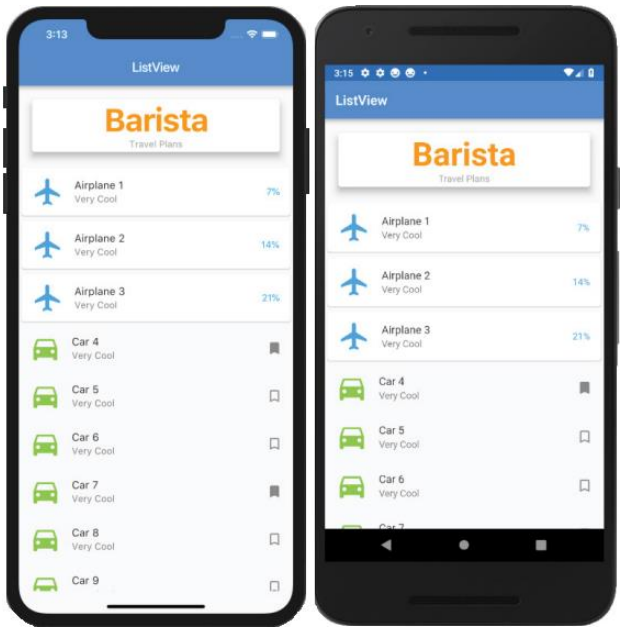
13. Modifica la clase de widget RowWidget para devolver un ListTile. Importa la biblioteca material.dart.

14. Para la propiedad leading, devuelva un icono. Para la propiedad trailing, devuelve un bookmark_border o un bookmark Icon. Para aleatorizar qué ícono devolver, usa un operador ternario para calcular el módulo de índice (%) de 3 y verifica si es un número par. Si el número es par o impar, se muestra el icono correspondiente. La propiedad del título devuelve un widget de texto con el valor del índice. La propiedad de subtítulos devuelve un widget de texto.

    Para onTap, usa una declaración de impresión para mostrar el índice de la fila marcada.

    Ten en cuenta que para cada fila se agrega un widget al árbol de widgets.

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

class RowWidget extends StatelessWidget {
  const RowWidget({
    Key key,
    @required this.index,
  }) : super(key: key);

  final int index;

  @override
  Widget build(BuildContext context) {
    return ListTile(
      leading: Icon(
        Icons.directions_car,
        size: 48.0,
        color: Colors.lightGreen,
      ),
      title: Text('Car $index'),
      subtitle: Text('Very Cool'),
      trailing: (index % 3).isEven
          ? Icon(Icons.bookmark_border)
          : Icon(Icons.bookmark),
      selected: false,
      onTap: () {
        print('Tapped on Row $index');
      },
    );
  }
}
```



El constructor ListView.builder toma un itemCount y usa el itemBuilder para construir un widget para cada registro secundario. Cada widget secundario se agrega al árbol de widgets con los valores adecuados. A medida que se agrega cada widget secundario, puedes personalizar las filas de ListView de acuerdo con las especificaciones de la aplicación.

El uso de ListTile hace que sea extremadamente fácil alinear los widgets. ListTile tiene propiedades iniciales (leading), finales (trailing), título (title), subtítulo (subtitle), onTap y otras.

Usando GridView

GridView (figura 9.3) muestra mosaicos de widgets desplazables en formato de cuadrícula (rejilla). Los tres constructores en los que nos enfocamos son GridView.count, GridView.extent y GridView.builder.

GridView.count y GridView.extent se utilizan normalmente con un conjunto de datos fijo o más pequeño. El uso de estos constructores significa que todos los datos, no solo los widgets visibles, se cargan en init. Si tienes un gran conjunto de datos, el usuario no ve GridView hasta que se cargan todos los datos, lo cual no es una gran experiencia de usuario (UX). Por lo general, usas GridView.count cuando necesitas un diseño con un número fijo de mosaicos en el eje transversal. Por ejemplo, se muestran tres mosaicos en los modos vertical y horizontal. Utiliza GridView.extent cuando necesites un diseño con los mosaicos que necesitan una extensión máxima de eje transversal.

Por ejemplo, de dos a tres mosaicos encajan en el modo vertical y de cinco a seis mosaicos en el modo horizontal; en otras palabras, se adapta a tantos mosaicos como sea posible según el tamaño de la pantalla.

El constructor GridView.builder se usa con un conjunto de datos de tamaño mayor, infinito o desconocido. Como nuestro ListView.builder, cuando tienes un gran conjunto de datos, se llama al constructor solo para widgets visibles, lo cual es excelente para el rendimiento.

Dentro del constructor, utilizas el callback del itemBuilder para crear la lista de widgets secundarios. Ten en cuenta que solo se llama a itemBuilder si el argumento itemCount es mayor que cero y se llama tantas veces como el valor itemCount. Recuerda, la Lista comienza en la fila 0, no 1. Si tienes 20 elementos en la Lista, se repite desde la fila 0 a la 19. El argumento scrollDirection predeterminado es Axis.vertical pero puede cambiarse a Axis.horizontal (figura 9.3).

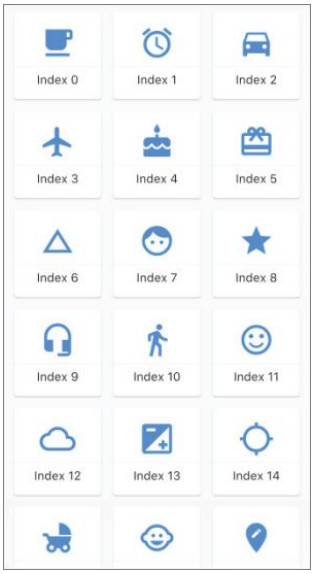


Figura 9.3: Diseño de GridView.

Usando GridView.count

GridView.count requiere establecer crossAxisCount y el argumento de los hijos. CrossAxisCount establece el número de mosaicos que se mostrarán (figura 9.4), y los elementos secundarios son una lista de widgets.

El argumento scrollDirection establece la dirección del eje principal para que la cuadrícula se desplace, ya sea Axis.vertical o Axis.horizontal, y el valor predeterminado es vertical.

Para los hijos, usa List.generate para crear tus datos de muestra, una lista de valores. Dentro del argumento de los hijos, agregamos una declaración de impresión para mostrar que toda la lista de valores se construye al mismo tiempo, no solo las filas visibles como GridView.builder. Ten en cuenta que para el siguiente código de muestra, se generan 7,000 registros para mostrar que GridView.count no muestra ningún dato hasta que todos los registros se procesan primero.

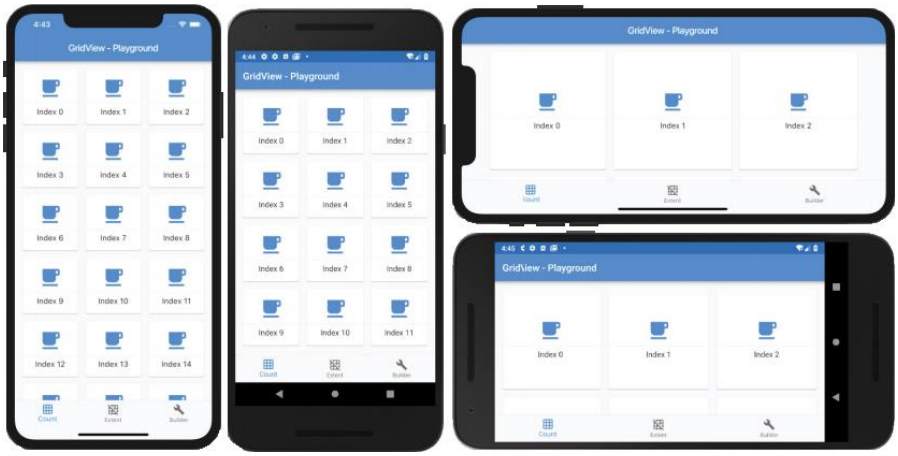


Figura 9.4: GridView cuenta con tres mosaicos en modo vertical y horizontal.

```
GridView.count(
  crossAxisCount: 3,
  padding: EdgeInsets.all(8.0),
  children: List.generate(7000, (index) {
    print('_buildGridView $index');

    return Card(
      margin: EdgeInsets.all(8.0),
      child: InkWell(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Icon(
              _iconList[0],
              size: 48.0,
              color: Colors.blue,
            ),
            Divider(),
            Text(
              'Index $index',
              textAlign: TextAlign.center,
              style: TextStyle(
                fontSize: 16.0,
              ),
            )
          ],
        ),
        onTap: () {
          print('Row $index');
        },
      ),
    );
  },),
)
```

Usando GridView.extent

GridView.extent requiere que establezcas el argumento maxCrossAxisExtent y children. El argumento maxCrossAxisExtent establece el tamaño máximo de cada mosaico para el eje. Por ejemplo, en vertical, pueden caber de dos a tres mosaicos, pero cuando se gira a horizontal, pueden caber de cinco a seis según el tamaño de la pantalla (figura 9.5). El argumento scrollDirection establece la dirección del eje principal para que la cuadrícula (rejilla) se desplace, ya sea Axis.vertical o Axis.horizontal, y el valor predeterminado es vertical.

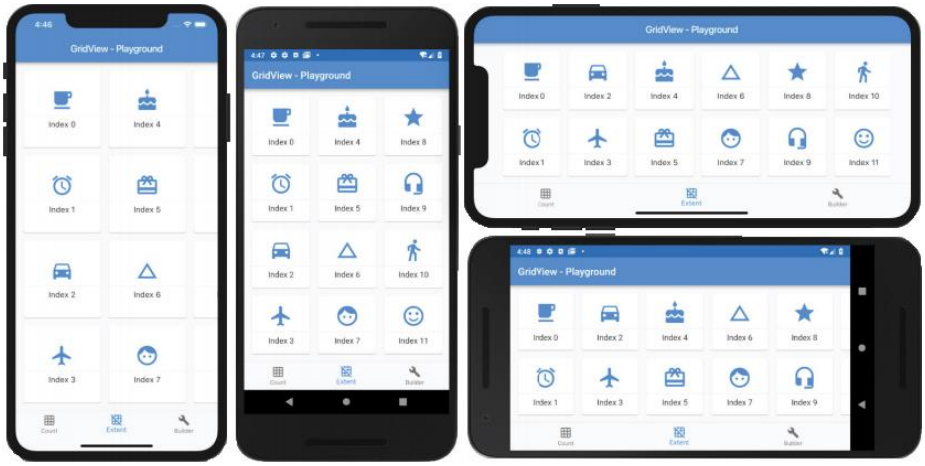


Figura 9.5: Extensión de GridView que muestra el número máximo de mosaicos que pueden caber según el tamaño de la pantalla.

Para los hijos, usa List.generate para crear tus datos de muestra, que es una lista de valores. Dentro del argumento de los hijos, agrega una declaración de impresión para mostrar que toda la lista de valores se construye al mismo tiempo, no solo las filas visibles como GridView.builder. Ver el código en la siguiente página.

Usando GridView.builder

GridView.builder requiere que establezcas los argumentos itemCount, gridDelegate y itemBuilder. ItemCount establece el número de mosaicos a construir. GridDelegate es un SilverGridDelegate responsable de diseñar la lista secundaria de widgets para GridView. El argumento gridDelegate no puede ser nulo; debe pasar el tamaño maxCrossAxisExtent, por ejemplo, 150.0 píxeles.

Por ejemplo, para mostrar tres mosaicos en la pantalla, especifique el argumento gridDelegate con la clase SliverGridDelegateWithFixedCrossAxisCount para crear un diseño de cuadrícula con un número fijo de mosaicos para el eje transversal. Si necesita mostrar

mosaicos que tienen un ancho máximo de 150.0 píxeles, especifique el argumento gridDelegate con la clase SliverGridDelegateWithMaxCrossAxisExtent para crear un diseño de cuadrícula con mosaicos que tienen una extensión máxima de eje transversal, el ancho máximo de cada mosaico.

```
GridView.extent(
  maxCrossAxisExtent: 175.0,
  scrollDirection: Axis.horizontal,
  padding: EdgeInsets.all(8.0),
  children: List.generate(20, (index) {
    print('_buildGridViewExtent $index');

    return Card(
      margin: EdgeInsets.all(8.0),
      child: InkWell(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Icon(
              _iconList[index],
              size: 48.0,
              color: Colors.blue,
            ),
            Divider(),
            Text(
              'Index $index',
              textAlign: TextAlign.center,
              style: TextStyle(
                fontSize: 16.0,
              ),
            )
          ],
        ),
        onTap: () {
          print('Row $index');
        },
      ),
    );
  }),
)
```

Código para GridView.extent

GridView.builder se usa cuando tienes un gran conjunto de datos porque el constructor se llama solo para mosaicos visibles, lo cual es excelente para el rendimiento. El uso del constructor GridView.builder da como resultado la construcción perezosa de una lista de mosaicos visibles, y cuando el usuario se desplaza a los siguientes mosaicos visibles, se construyen de manera perezosa según sea necesario.

Creación de la aplicación GridView.builder

En este ejemplo, el widget GridView usa el constructor para mostrar una tarjeta que muestra cada elemento de la rejilla con un icono y un texto que muestra la ubicación del índice. El onTap imprimirá el índice del elemento Grid tocado.

1. Crea un nuevo proyecto Flutter y asígnale el nombre gridview, siguiendo las instrucciones de la práctica 4. Para este proyecto, necesitas crear solo las carpetas de páginas, clases y widgets. Crea la clase de inicio como un widget sin estado, ya que los datos no requieren cambios.

2. Abre el archivo home.dart y agrega al cuerpo un SafeArea con la clase de widget GridViewBuilderWidget() como hijo.

```
body: SafeArea(
  child: const GridViewBuildWidget(),
),
```

3. Agrega en la parte superior del archivo la declaración de importación para la clase de widget gridview_builder.dart que crearás a continuación.

```
import 'package:flutter/material.dart';
import 'package:gridview/widgets/gridview_builder.dart';
```

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

5. Importa la biblioteca material.dart, agrega una nueva línea y crea la clase GridIcons. La clase GridIcons contiene una lista de IconData llamada iconList.

6. Crea el método getIconList() que crea la Lista de IconData que se utiliza más adelante en GridView.builder.

```
class GridIcons {
  List<IconData> iconList = [];

  List<IconData> getIconList() {
    iconList
      ..add(Icons.free_breakfast)
      ..add(Icons.access_alarms)
      ..add(Icons.directions_car)
      ..add(Icons.flight)
      ..add(Icons.cake)
      ..add(Icons.card_giftcard)
      ..add(Icons.change_history)
      ..add(Icons.face)
      ..add(Icons.star)
      ..add(Icons.headset_mic)
      ..add(Icons.directions_walk)
      ..add(Icons.sentiment_satisfied)
      ..add(Icons.cloud_queue)
      ..add(Icons.exposure)
      ..add(Icons.gps_not_fixed)
      ..add(Icons.child_friendly)
      ..add(Icons.child_care)
      ..add(Icons.edit_location)
      ..add(Icons.event_seat)
      ..add(Icons.lightbulb_outline);
    return iconList;
  }
}
```

7. 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 gridview_builder.dart y haza clic en el botón OK para guardar.

8. Importa la biblioteca material.dart, agrega una nueva línea y luego comienza a escribir st; se abre la ayuda de autocompletado, así que selecciona la abreviatura stless (StatelessWidget) y asígnale el nombre GridViewBuilderWidget.

9. Modifica la clase de widget GridViewBuilderWidget para devolver un GridView.builder con el argumento itemCount establecido en 20. Para este ejemplo, especifica que se enumeren 20 filas de datos.

10. Para el argumento gridDelegate, usa SliverGridDelegateWithMaxCrossAxisExtent (maxCross AxisExtent: 150.0).
    Tu otra opción es usar SliverGridDelegateWithFixedCrossAxisCount en su lugar, que funciona de la misma manera que el constructor GridView.count, donde pasas el número de mosaicos para mostrar.

11. Para el callback de itemBuilder, pasa el BuildContext y el índice del widget como un valor int. En la primera línea de itemBuilder, coloca una declaración de impresión para mostrar el índice de cada elemento que se está construyendo de acuerdo con el espacio visible.

12. Devuelve una tarjeta con el hijo como InkWell. InkWell onTap tiene una declaración de impresión para mostrar el elemento de Card marcado, con la Fila seleccionada.

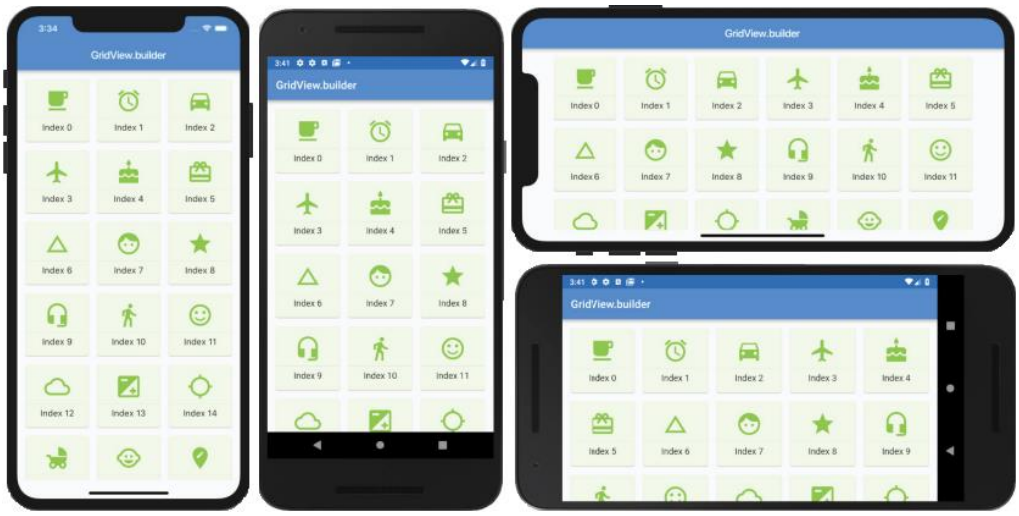
13. Para la propiedad secundaria InkWell, pasa una columna con estos elementos secundarios: widgets Icon, Divider y Text. Ten en cuenta que se llama al itemBuilder para cada elemento de la fila. Para cada fila, se crea un widget y se agrega al árbol de widgets.

    El onTap imprimirá el índice del elemento Grid tocado.

14. Agrega en la parte superior del archivo la declaración de importación para la clase grid_icons.dart.

```
import 'package:flutter/material.dart';
import 'package:gridview/classes/grid_icons.dart';

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

  @override
  Widget build(BuildContext context) {
    List<IconData> _iconList = GridIcons().getIconList();

    return GridView.builder(
      itemCount: 20,
      padding: EdgeInsets.all(8.0),
      gridDelegate: SliverGridDelegateWithMaxCrossAxisExtent(maxCrossAxisExtent:
150.0),
      itemBuilder: (BuildContext context, int index) {
        print('_buildGridViewBuilder $index');

        return Card(
          color: Colors.lightGreen.shade50,
          margin: EdgeInsets.all(8.0),
          child: InkWell(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: <Widget>[
                Icon(
                  _iconList[index],
                  size: 48.0,
                  color: Colors.lightGreen,
                ),
                Divider(),
                Text(
                'Index $index',
                textAlign: TextAlign.center,
                style: TextStyle(
                  fontSize: 16.0,
                ),
              )
            ],
          ),
          onTap: () {
            print('Row $index');
          },
        ),
      );
    },
  );
}
```

El constructor GridView.builder toma un itemCount y usa el itemBuilder para construir un widget para cada registro secundario. Cada widget secundario se agrega al árbol de widgets con los valores adecuados. A medida que se agrega cada widget secundario, puedes personalizar las filas de acuerdo con las especificaciones de la aplicación. En este ejemplo, usaste una Card para darle a cada elemento de Grid Row un aspecto agradable y usaste un InkWell para usar la animación de toque de Material Design y la propiedad onTap. El elemento secundario InkWell es una columna y sus elementos secundarios muestran un icono y un texto.



Usando la pila

El widget Stack se usa comúnmente para superponer, colocar y alinear widgets para crear una apariencia personalizada. Un buen ejemplo es un carrito de compras con la cantidad de artículos que se deben comprar en la parte superior derecha. La lista de hijos Stack del widget está posicionada o no posicionada. Cuando utilizas un widget posicionado, cada widget secundario se coloca en la ubicación adecuada.

El widget Stack cambia de tamaño para adaptarse a todos los elementos secundarios no colocados. Los elementos secundarios no colocados se colocan en la propiedad de alineación (superior izquierda o superior derecha, según el entorno de izquierda a derecha o de derecha a izquierda). Cada widget secundario de Stack se dibuja en orden de abajo hacia arriba, como apilar trozos de papel uno encima del otro. Esto significa que el primer widget dibujado está en la parte inferior de la pila, y luego el siguiente widget se dibuja sobre el widget anterior y así sucesivamente. Cada widget secundario se coloca uno encima del otro en el orden de la lista de elementos secundarios Stack. La clase RenderStack maneja el diseño de la pila.

Para alinear a cada hijo en la Pila, usa el widget Positioned. Al usar las propiedades superior, inferior, izquierda y derecha, alineas cada widget secundario dentro de la pila. Las propiedades de alto y ancho del widget Positioned también se pueden configurar (figura 9.6).

También aprenderás a implementar la clase FractionalTranslation para colocar un widget fraccionalmente fuera del widget principal. Establece la propiedad de traducción con la clase Offset (dx, dy) (valor de tipo doble para los ejes x e y) que se escala al tamaño del hijo, lo que resulta en mover y posicionar el widget. Por ejemplo, para mostrar un icono favorito movido un tercio del camino hacia la esquina superior derecha del widget principal, establece la propiedad de translation con el valor de Offset (0.3, -0.3).

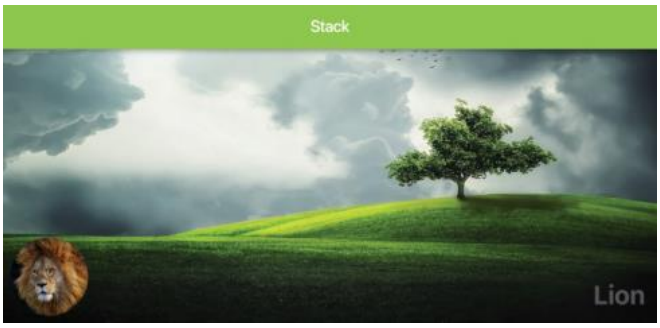


Figura 9.6: Diseño de pila que muestra los widgets de imagen y texto apilados sobre la imagen de fondo.

El siguiente ejemplo (figura 9.7) muestra un widget Stack con una imagen de fondo y, al usar la clase FractionalTranslation, establece la propiedad de translation en el valor Offset (0.3, -0.3), colocando el ícono de estrella un tercio a la derecha del eje x y un tercio negativo (mover el icono hacia arriba) en el eje y.

```
Stack(
  children: <Widget>[
    Image(image: AssetImage('assets/images/dawn.jpg')),
    Positioned(
      top: 0.0,
      right: 0.0,
      child: FractionalTranslation(
        translation: Offset(0.3, -0.3),
        child: CircleAvatar(
          child: Icon(Icons.star),
        ),
      ),
    ),
    Positioned(/* Eagle Image */),
    Positioned(/* Bald Eagle */),
  ],
),
```

Figura 9.7: Clase FractionalTranslation que muestra el icono favorito movido hacia la esquina superior derecha.

Creación de la aplicación Stack

En este ejemplo, la lista secundaria de widgets del widget Stack presenta una imagen de fondo y dos widgets posicionados con widgets CircleAvatar y Text. Para mostrar un diseño alternativo, usa el mismo diseño de pila anterior y agrega un widget posicionado con la propiedad secundaria como una clase FractionalTranslation para mostrar un CircleAvatar anclado en la esquina superior derecha a mitad del camino fuera de la pila.
Se usa un ListView para crear la lista de muestra, y cada fila muestra un widget de pila alternativo.

1. Crea un nuevo proyecto de Flutter y asígnale el nombre stack. Nuevamente, sigue las instrucciones del tema 4. Para este proyecto, debes crear las páginas, los widgets y las carpetas de assets/images. Crea la clase de inicio (Home Class) como un widget sin estado (StatelessWidget), ya que sus datos no requieren cambios.

2. Abre el archivo pubspec.yaml para agregar recursos. En la sección de activos, agrega la carpeta assets/images/.

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

3. Haz clic en el botón Save y, dependiendo del editor que estés utilizando, automáticamente ejecutarás los paquetes de flutter get y, una vez finalizado, mostrará un mensaje de Proceso terminado con el código de salida 0. Si no ejecuta automáticamente el comando para ti, abre la ventana de Terminal (ubicada en la parte inferior de tu editor) y escribe flutter packages get.

4. Agrega los recursos de la carpeta y las imágenes de la subcarpeta en la raíz del proyecto y luego copia los archivos dawn.jpg, eagle.jpg, lion.jpg y tree.jpg en la carpeta de imágenes.

5. Abre el archivo home.dart y agrega al cuerpo un SafeArea con ListView.builder() como hijo.

```
body: SafeArea(
   child: ListView.builder(),
),
```

6. Agrega en la parte superior del archivo la declaración de importación para las clases de widgets stack.dart y stack_favorite.dart que crearás a continuación.

```
import 'package:flutter/material.dart';
import 'package:stack/widgets/stack.dart';
import 'package:stack/widgets/stack_favorite.dart';
```

7. Agrega a ListView.builder el argumento itemCount con un valor establecido en 7. Para este ejemplo, especifica que se enumeren siete filas de datos. Para el callback de itemBuilder, pasa el BuildContext y el índice del widget como un valor int.
   Con cada diseño de Stack, alterna entre ellos verificando si el valor del índice es par o impar y luego llama a las clases de widget StackWidget() y StackFavoriteWidget(), respectivamente.
   Queremos mostrarte que puedes personalizar los widgets que presentas al usuario. Supongamos que tienes una aplicación que se distribuye como software gratuito y cada diez registros muestra un anuncio o un consejo incrustado en la lista. Esta técnica no es tan intrusiva como una ventana emergente mientras el usuario ve registros.

```
body: SafeArea(
  child: ListView.builder(
    itemCount: 7,
    itemBuilder: (BuildContext context, int index) {
      if (index.isEven) {
        return const StackWidget();
      } else {
        return const StackFavoriteWidget();
      }
    },
  ),
),
```

8. 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 stack.dart y haz clic en el botón OK para guardar.

9. Importa la biblioteca material.dart, agrega una nueva línea y luego comienza a escribir st; se abre la ayuda de autocompletado, donde puedes seleccionar la abreviatura stless (StatelessWidget) y darle un nombre de StackWidget.

10. Modifica la clase de widget StackWidget para devolver una pila. La lista secundaria de widgets de Stack consta de una imagen con el AssetImage tree.jpg.

11. Agrega un widget posicionado con propiedades inferiores e izquierdas con un valor de 10.0. El hijo es un CircleAvatar con un radio de 48.0 y tiene la propiedad backgroundImage establecida en AssetImage lion.jpg.

12. Agrega otro widget posicionado con propiedades inferior y derecha con un valor de 16.0. El hijo es un widget Text con un valor de cadena de Lion y tiene una propiedad de estilo con una clase TextStyle con un fontSize establecido en 32.0 píxeles, color establecido en white30 y fontWeight establecido en negrita.

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

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

  @override
  Widget build(BuildContext context) {
    return Stack(
      children: <Widget>[
        Image(
          image: AssetImage('assets/images/tree.jpg'),
        ),
        Positioned(
          bottom: 10.0,
          left: 10.0,
          child: CircleAvatar(
            radius: 48.0,
            backgroundImage: AssetImage('assets/images/lion.jpg'),
          ),
        ),
        Positioned(
          bottom: 16.0,
          right: 16.0,
          child: Text(
            'Lion',
            style: TextStyle(
              fontSize: 32.0,
              color: Colors.white30,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
      ],
    );
  }
}
```

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

14. Importa la biblioteca material.dart, agrega una nueva línea y luego comienza a escribir st; Se abre la ayuda de autocompletado, así que selecciona la abreviatura stless (StatelessWidget) y asígnale un nombre de StackFavoriteWidget.

15. Modifica la clase de widget StackFavoriteWidget para devolver un Container. Usa un contenedor para establecer el color en black87 y para el hijo usa un relleno (Padding) con EdgeInsets.all(16.0). Esto creará un efecto de marco oscuro alrededor de la pila.

16. Para la lista secundaria de widgets de Pila, agrega un conjunto de imágenes a AssetImage dawn.jpg.

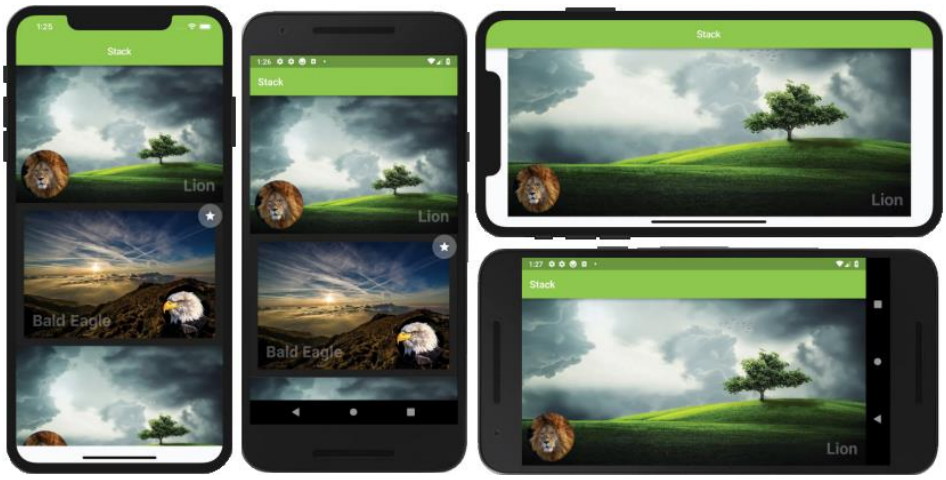
17. Agrega un widget posicionado con propiedades inferior y derecha con los valores 0.0. El hijo es una clase FractionalTranslation con una propiedad de translation de Offset (0.3, -0.3). El hijo es un CircleAvatar y, al usar el Offset, lo muestra anclado a la esquina superior derecha a la mitad del exterior de la Pila.

18. Agrega un widget posicionado con propiedades en la parte inferior y derecha con un valor de 10.0. El hijo es un CircleAvatar con un radio de 48.0 y backgroundImage con AssetImage eagle.jpg.

19. Agrega otro widget posicionado con propiedades en la parte inferior y derecha que tengan valores de 16.0. El hijo es un widget de texto con un valor de cadena de Bald Eagle y tiene una propiedad de estilo con un TextStyle con fontSize establecido en 32.0 píxeles, color establecido en white30 y fontWeight establecido en negrita.

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

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

  @override
  Widget build(BuildContext context) {
    return Container(
      color: Colors.black87,
      child: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Stack(
          children: <Widget>[
            Image(
              image: AssetImage('assets/images/dawn.jpg'),
            ),
            Positioned(
              top: 0.0,
              right: 0.0,
              child: FractionalTranslation(
                translation: Offset(0.3, -0.3),
                child: CircleAvatar(
                  radius: 24.0,
                  backgroundColor: Colors.white30,
                  child: Icon(
                    Icons.star,
                    size: 24.0,
                    color: Colors.white,
                  ),
                ),
              ),
            ),
            Positioned(
              bottom: 10.0,
```

```
                right: 10.0,
                child: CircleAvatar(
                  radius: 48.0,
                  backgroundImage: AssetImage('assets/images/eagle.jpg'),
                ),
              ),
              Positioned(
                bottom: 16.0,
                left: 16.0,
                child: Text(
                  'Bald Eagle',
                  style: TextStyle(
                    fontSize: 32.0,
                    color: Colors.white30,
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ),
            ],
          ),
        ),
      );
  }
}
```



La pila toma una lista secundaria de widgets y se ajusta a sí misma para acomodar todos los widgets no colocados. Cuando utilizas widgets no colocados en la pila, se colocan automáticamente en la configuración de alineación (superior izquierda o superior derecha según el entorno). Cada widget secundario de Stack se dibuja en orden de abajo hacia arriba, lo que significa que cada widget secundario se coloca uno encima del otro.

El uso del widget Posicionado permite alinear cada widget secundario utilizando las propiedades superior, inferior, izquierda y derecha. Aprendiste a colocar el icono de favorito en la esquina superior derecha del widget principal mediante el uso de la propiedad de translation de la clase FractionalTranslation con el valor Offset (0.3, -03).

Personalización de CustomScrollView con Slivers

El widget CustomScrollView crea efectos de desplazamiento personalizados utilizando una lista de astillas (slivers). Las slivers son una pequeña porción de algo más grande. Por ejemplo, las slivers se colocan dentro de un puerto de visualización como el widget CustomScrollView. En las secciones anteriores, aprendiste cómo implementar los widgets ListView y GridView por separado. Pero, ¿y si necesitaras presentarlos juntos en la misma lista?

La respuesta es que puedes usar CustomScrollView con la lista de widgets de propiedades de slivers configurada en los widgets SliverSafeArea, SliverAppBar, SliverList y SliverGrid (slivers). El orden en el que los colocas en la propiedad de slivers CustomScrollView es el orden en el que se representan. La tabla 9.1 muestra las slivers de uso común y el código de muestra.

Tabla 9.1: Slivers.

Sliver	Descripción	Código
SliverSafeArea	Agrega relleno para evitar la muesca del dispositivo que generalmente se encuentra en la parte superior de la pantalla.	SliverSafeArea(sliver: SliverGrid(),)

SliverAppBar	Agrega una barra de aplicaciones.	SliverAppBar(
		  expandedHeight: 250.0,
		  flexibleSpace: FlexibleSpaceBar(
		  title: Text('Parallax'),
		  ),
		)

SliverList	Crea una lista de widgets desplazable lineal.	SliverList(
		  delegate: SliverChildListDelegate(
		  List.generate(3, (int index) {
		  return ListTile();
		  }),
		  ),
		)

SliverGrid	Muestra mosaicos de widgets desplazables en formato de cuadrícula.	SliverGrid(
		  delegate: SliverChildBuilderDelegate(
		  (BuildContext context, int index) {
		  return Card();
		  },
		  childCount: _rowsCount,
		  ),
		  gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 3),
		)

SliverList y SliverGrid usan delegados para construir la lista de hijos de manera explícita o perezosa.

Una lista explícita construye primero todos los elementos y luego los muestra en la pantalla. Una lista construida de forma perezosa solo crea los elementos visibles en la pantalla y cuando el usuario se desplaza, se crean (de forma perezosa) los siguientes elementos visibles, lo que da como resultado un mejor rendimiento. SliverList tiene una propiedad de delegado y SliverGrid tiene un delegado y una propiedad gridDelegate.

La propiedad de delegado SliverList y SliverGrid puede usar SliverChildListDelegate para construir una lista explícita o usar SliverChildBuilderDelegate para construir la lista de manera perezosa. SliverGrid tiene una propiedad gridDelegate adicional para especificar el tamaño y la posición de los mosaicos de la rejilla. Especifica la propiedad gridDelegate con la clase SliverGridDelegateWithFixedCrossAxisCount para crear un diseño de rejilla con un número fijo de mosaicos para el eje transversal; por ejemplo, muestra tres mosaicos de ancho. Especifica la propiedad gridDelegate con la clase SliverGridDelegateWithMaxCrossAxisExtent para crear un diseño de rejilla con mosaicos que tienen una extensión máxima de eje transversal, el ancho máximo de cada mosaico; por ejemplo, 150.0 píxeles de ancho máximo para cada mosaico.

La Tabla 9.2 muestra los delegados SliverList y SliverGrid para ayudarte a crear listas.

Tabla 9.2: Delegados Sliver.

Sliver	Descripción	Código

SliverList	SliverChildListDelegate crea una lista de un número conocido de filas (explícito). SliverChildBuilderDelegate construye perezosamente una lista de un número desconocido de filas.	SliverList(
		delegate:
		SliverChildListDelegate(<Widget>[
		ListTile(title: Text('One')),
		ListTile(title: Text('Two')),
		ListTile(title: Text('Three')),
		]),
		)
		o
		SliverList(
		delegate: SliverChildListDelegate(
		List.generate(30, (int index) {
		return ListTile();
		}),
		),
		)
		o
		SliverList(
		delegate: SliverChildBuilderDelegate((BuildContext
		context,
		int index) {
		return ListTile();
		},
		childCount: _rowsCount,
		),
		)

SliverGrid	SliverChildListDelegate crea una lista explícita.
	SliverChildBuilderDelegate construye perezosamente una lista de un número desconocido de mosaicos.
	La propiedad gridDelegate controla la posición y el tamaño de los widgets secundarios.
		SliverGrid(
		delegate: SliverChildListDelegate-(<Widget>[
		Card(),
		Card(),
		Card(),
		]),
		gridDelegate:
		SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 3),
		)
		o
		SliverGrid(
		delegate: SliverChildListDelegate(
		List.generate(30, (int index) {
		return Card();
		}),
		),
		gridDelegate:
		SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 3),
		)
		o
		SliverChildBuilderDelegate(
		(BuildContext context,
		int index) {
		return Card();
		},
		childCount: _rowsCount,
		),
		gridDelegate:
		SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 3),
		)

El widget SliverAppBar puede tener un efecto de paralaje (figura 9.8) mediante el uso de las propiedades extendedHeight y flexibleSpace. El efecto de paralaje desplaza una imagen de fondo más lentamente que el contenido en primer plano.

Si necesitas mostrar CustomScrollView inicialmente desplazado en una posición en particular, usa un controlador y establece la propiedad ScrollController.initialScrollOffset.

Por ejemplo, establecerías initialScrollOffset inicializando el controller = ScrollController (initialScrollOffset: 10.0).

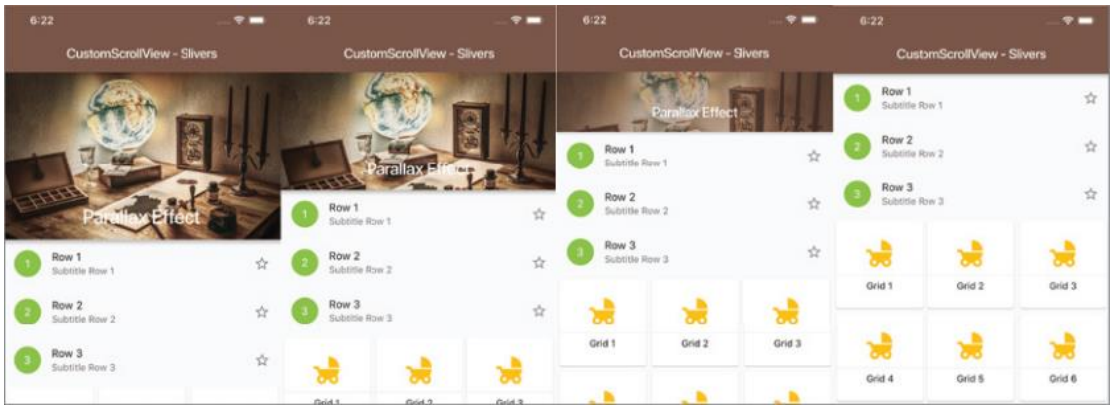


Figura 9.8: SliverAppBar efecto de paralaje de desplazamiento.

Creación de la aplicación CustomScrollView Slivers

En este ejemplo, la lista secundaria de widgets CustomScrollView contiene SliverAppBar, SliverList, SliverSafeArea y SliverGrid. El widget SliverAppBar usa flexibleSpace con una imagen de fondo que tiene un efecto de paralaje mientras se desplaza. SliverList genera tres elementos con el constructor List.generate. Para tener en cuenta la muesca del dispositivo, usa un SliverSafeArea para envolver el SliverGrid y genera 12 (el valor de muestra puede ser más o menos) elementos.

1. Crea un nuevo proyecto Flutter y asígnale el nombre customscrollview_slivers; puedes seguir las instrucciones de la práctica 4. Para este proyecto, solo necesitas crear las carpetas de páginas y assets/images. Crea la Home Class como un StatelessWidget, ya que los datos no requieren cambios.

2. Abre el archivo pubspec.yaml para agregar recursos. En la sección de assets, agrega la carpeta assets/images /.

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

3. Haz clic en el botón Save y, según el editor que estés utilizando, ejecutarás automáticamente los paquetes de flutter get. Una vez terminado, mostrará un mensaje de Proceso terminado con el código de salida 0. Si no ejecuta automáticamente el comando por ti, abre la ventana Terminal (ubicada en la parte inferior de tu editor) y escribe flutter packages get.

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

5. Abre el archivo home.dart y agrega al cuerpo un CustomScrollView(). Para este proyecto, establece la propiedad de elevación AppBar en 0.0 porque habilita la sombra SliverAppBar en su lugar.

```
return Scaffold(
   appBar: AppBar(
      title: Text('CustomScrollView - Slivers'),
      elevation: 0.0,
   ),
   body: CustomScrollView(
      slivers: <Widget>[
      ],
   ),
);
```

6. Agregue en la parte superior del archivo la declaración de importación para las clases de widgets sliver_app_bar.dart, sliver_list.dart y sliver_grid.dart que creará a continuación.

```
import 'package:flutter/material.dart';
import 'package:customscrollview_slivers/widgets/sliver_app_bar.dart';
import 'package:customscrollview_slivers/widgets/sliver_list.dart';
import 'package:customscrollview_slivers/widgets/sliver_grid.dart';
```

7. Agrega llamadas a las clases de widgets SliverAppBarWidget(), SliverListWidget() y SliverGridWidget() a la propiedad de slivers CustomScrollView(). Asegúrate de que las llamadas a las clases de widgets utilicen la palabra clave const para aprovechar el almacenamiento en caché para aumentar el rendimiento.

```
return Scaffold(
   appBar: AppBar(
     title: Text('CustomScrollView - Slivers'),
     elevation: 0.0,
   ),
   body: CustomScrollView(
     slivers: <Widget>[
        const SliverAppBarWidget(),
        const SliverListWidget(),
        const SliverGridWidget(),
     ],
   ),
);
```

8. 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 sliver_app_bar.dart y haz clic en el botón OK para guardar.

9. Importa la biblioteca material.dart, agrega una nueva línea y luego comienza a escribir st; se abre la ayuda de autocompletado, así que selecciona la abreviatura stless (StatelessWidget) y asígnale el nombre SliverAppBarWidget.

10. Modifica la clase de widget SliverAppBarWidget para devolver un SliverAppBar.

11. Para mostrar una sombra en la parte inferior de la barra, establece la propiedad forceElevated en true.

12. Para crear un efecto de paralaje mientras se desplaza, establecea extendedHeight en 250.0 píxeles y flexibleSpace en FlexibleSpaceBar.

13. Para la propiedad de fondo FlexibleSpaceBar, usa el widget Image con el archivo desk.jpg y configura el ajuste en BoxFit.cover.

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

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

   @override
   Widget build(BuildContext context) {
     return SliverAppBar(
       backgroundColor: Colors.brown,
       forceElevated: true,
       expandedHeight: 250.0,
       flexibleSpace: FlexibleSpaceBar(
         title: Text(
           'Parallax Effect',
         ),
         background: Image(
           image: AssetImage('assets/images/desk.jpg'),
           fit: BoxFit.cover,
         ),
       ),
     );
   }
 }
```

14. 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 sliver_list.dart y haz clic en el botón OK para guardar.

15. Importa la biblioteca material.dart, agrega una nueva línea y luego comienza a escribir st; se abre la ayuda de autocompletado, así que selecciona la abreviatura stless (StatelessWidget) y asígnale el nombre SliverListWidget.

16. Modifica la clase de widget SliverListWidget para devolver una SliverList. Para la propiedad delegada SliverList, pasa SliverChildListDelegate.

17. Utiliza el constructor List.generate para crear tu lista de datos de muestra. El constructor toma dos argumentos: la longitud de la lista y el índice. Devuelve un ListTile con un CircleAvatar inicial con el elemento secundario como un widget de texto con interpolación de cadenas configurada con ${index + 1}.

18. Además, establece el título ListTile, el subtítulo y las propiedades finales.

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

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

  @override
  Widget build(BuildContext context) {
    return SliverList(
      delegate: SliverChildListDelegate(
        List.generate(3, (int index) {
          return ListTile(
            leading: CircleAvatar(
              child: Text("${index + 1}"),
              backgroundColor: Colors.lightGreen,
              foregroundColor: Colors.white,
            ),
            title: Text('Row ${index + 1}'),
            subtitle: Text('Subtitle Row ${index + 1}'),
            trailing: Icon(Icons.star_border),
          );
        }),
      ),
    );
  }
}
```

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

20. Importa la biblioteca material.dart, agrega una nueva línea y luego comienza a escribir st; se abre la ayuda de autocompletado, así que selecciona la abreviatura stless (StatelessWidget) y asígnale el nombre SliverGridWidget.

21. Modifica la clase de widget SliverGridWidget para devolver un SliverSafeArea. Dado que SliverGrid no maneja la muesca del dispositivo automáticamente, lo envuelve en un SliverSafeArea. La propiedad del delegado SliverGrid es un SliverChildBuilderDelegate que toma el índice BuildContext e int.

22. Desde SliverChildBuilderDelegate, devuelve una Card con el hijo como Columna. La lista secundaria Column del widget tiene widgets de ícono, divisor y texto.

23. Para la propiedad childCount, pasa 12 que representa cuántos elementos crea el constructor. La propiedad gridDelegate se establece en SliverGridDelegateWithFixedCrossAxisCount crossAxisCount: 3) mostrando tres mosaicos.

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

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

  @override
  Widget build(BuildContext context) {
    return SliverSafeArea(
      sliver: SliverGrid(
        delegate: SliverChildBuilderDelegate(
              (BuildContext context, int index) {
            return Card(
              child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: <Widget>[
                  Icon(Icons.child_friendly, size: 48.0, color: Colors.amber,),
                  Divider(),
                  Text('Grid ${index + 1}'),
                ],
              ),
            );
          },
          childCount: 12,
        ),
        gridDelegate:
        SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 3),
      ),
    );
  }
}
```

Para crear un efecto de desplazamiento personalizado, CustomScrollView utiliza una lista de slivers. Usaste SliverAppBar para crear un efecto de desplazamiento de paralaje mediante el uso de FlexibleSpaceBar. También creaste una SliverList y estableciste la propiedad delegate en la clase SliverChildListDelegate. Para manejar la muesca del dispositivo, SliverGrid está

envuelto en un widget SliverSafeArea. La propiedad de delegado SliverGrid usa SliverChildBuilderDelegate, que toma un BuildContext y un índice int.

Resumen

En esta práctica, aprendiste a usar la Card para agrupar información con el contenedor con esquinas redondeadas y una sombra. Usaste ListView para crear una lista de widgets desplazables y para alinear datos agrupados con ListTile, y usaste GridView para mostrar datos en mosaicos, usando la Card para agrupar los datos. Incrustaste una pila en un ListView para mostrar una imagen como fondo y apilaste diferentes widgets con el widget posicionado para superponerlos y colocarlos en las ubicaciones adecuadas utilizando las propiedades superior, inferior, izquierda y derecha.

En la próxima práctica, aprenderás a crear diseños personalizados mediante SingleChildScrollView, SafeArea, Padding, Column, Row, Image, Divider, Text, Icon, SizedBox, Wrap, Chip y CircleAvatar. Aprenderás a tener una vista de alto nivel y una vista detallada para separar y anidar widgets para crear una interfaz de usuario personalizada.