MINISTERIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

Unidad 3

Interfaces Gráficas de Usuario - Conceptos adicionales

Universidad Tecnológica de Pereira

Mision TIC 2022

MINISTERIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

Añadir assets e imágenes

Añadir assets e imágenes

Las aplicaciones de Flutter pueden incluir tanto código como assets (algunas veces llamados **recursos**).

Un asset es un archivo que se incluye e implementa con su aplicación, y es accesible en tiempo de ejecución.

Los tipos comunes de recursos incluyen datos estáticos (por ejemplo, archivos JSON), archivos de configuración, iconos e imágenes (JPEG, WebP, GIF, WebP/GIF animados, PNG, BMP y WBMP).

Flutter usa el archivo **pubspec.yaml**, ubicado en la raíz de su proyecto, para identificar los recursos requeridos por una aplicación.

Aquí hay un ejemplo:

```
flutter:
  assets:
    - assets/my_icon.png
    - assets/background.png
```

Para incluir todos los assets en un directorio, especifique el nombre del directorio con el carácter / al final:

```
flutter:
  assets:
    - assets/
```

Tenga en cuenta que solo se incluirán los archivos ubicados directamente en el directorio; para agregar archivos ubicados en subdirectorios, cree una entrada por directorio.

Cargando elementos de texto

Cada aplicación Flutter tiene un **rootBundle** (paquete raiz) para acceder fácilmente al paquete de recursos principal.

Sin embargo, se recomienda obtener el AssetBundle para el BuildContext actual utilizando **DefaultAssetBundle**.

Normalmente, utilizará **DefaultAssetBundle.of()** para cargar indirectamente un recurso, por ejemplo un archivo JSON, desde el **rootBundle** en tiempo de ejecución de la aplicación.

Es posible cargar recursos directamente utilizando la rootBundle estática global de package:flutter/services.dart.

```
import 'dart:async' show Future;

import 'package:flutter/services.dart' show rootBundle;


Future<String> loadAsset() async {

  return await rootBundle.loadString('assets/config.json');

}
```

Cargando imágenes

Para cargar una imagen, usa la clase **AssetImage** en el método **build** de un Widget.

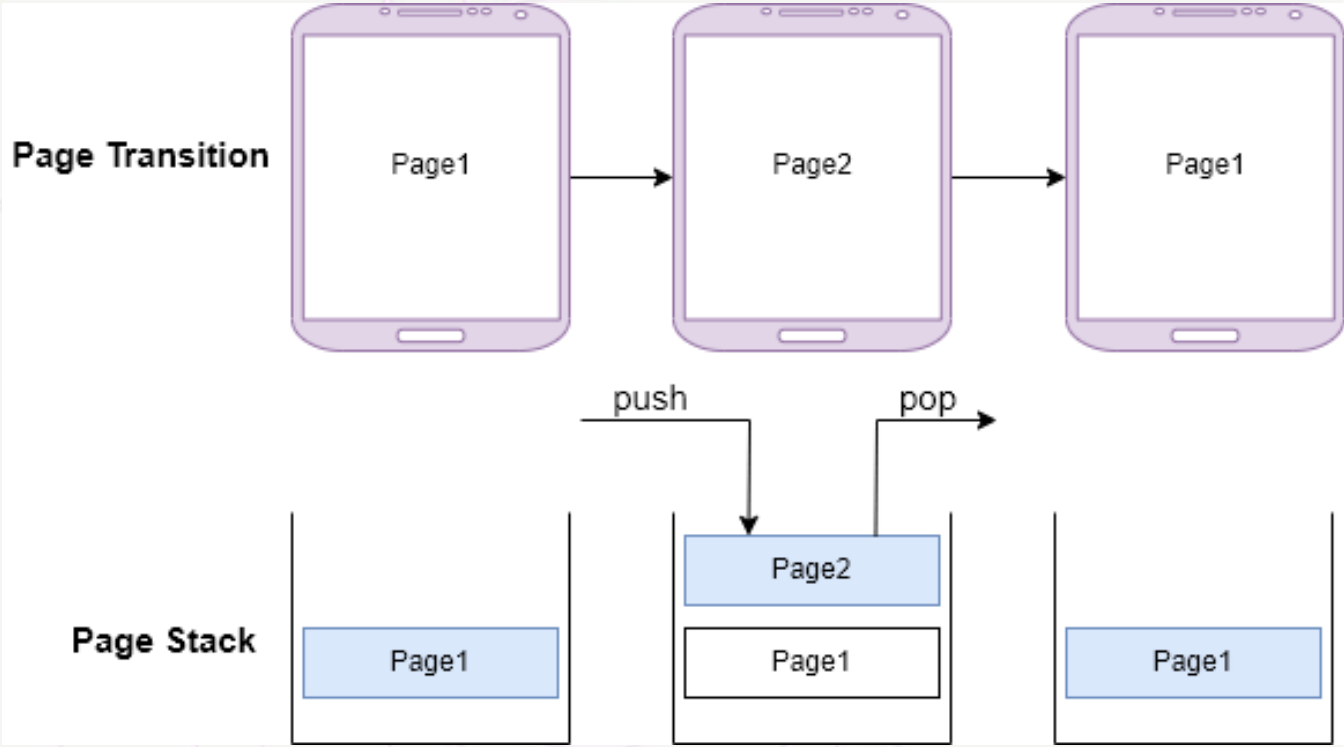
```
Widget build(BuildContext context) {
  // ...
  return DecoratedBox(
    decoration: BoxDecoration(
      image: DecorationImage(
        image:
AssetImage('graphics/background.png'),
        // ...
      ),
      // ...
    ),
  );
  // …
}
```

Universidad Tecnológica de Pereira

MisionTIC 2022

MINISTERIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

Navegación y enrutado



Pila de navegación

Usando la clase Navigator

El widget **Navigator** muestra pantallas como una pila utilizando las animaciones de transición correctas para la plataforma de destino.

Para navegar a una nueva pantalla, acceda al **Navegator** a través del **BuildContext** de la ruta y llame a métodos imperativos como `push()` o `pop()`.

Debido a que **Navigator** mantiene una pila de objetos **Route** (que representan la pila de historial), el método `push()` también toma un objeto **Route**.

El objeto **MaterialPageRoute** es una subclase de **Route** que especifica las animaciones de transición para Material Design.

```
onPressed: () {
  Navigator.of(context).push(
    MaterialPageRoute(
      builder: (context) => const SongScreen(song: song),
    ),
  );
},
child: Text(song.name),
```

Usando la clase Navigator (Ejemplo)

```
class FirstRoute extends StatelessWidget {
  const FirstRoute({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Route'),
      ),
      body: Center(
        child: ElevatedButton(
          child: const Text('Open route'),
          onPressed: () {
            Navigator.push(context,
              MaterialPageRoute(
                builder: (context) => const SecondRoute()));
          },
        ),
      ),
    );
  }
}
```

```
class SecondRoute extends StatelessWidget {
  const SecondRoute({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Second Route'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: const Text('Go back!'),
        ),
      ),
    );
  }
}
```

Enviando datos a una nueva ventana

Definir la clase de los datos a enviar

```
class Todo {
  final String title;
  final String description;

  const Todo(this.title, this.description);
}
```

Generar una lista de 20 **Todo**s

```
final todos = List.generate(
  20,
  (i) => Todo(
    'Todo $i',
    'A description for Todo $i',
  ),
);
```

Función **main()**

```
void main() {
  runApp(
    MaterialApp(
      title: 'Passing Data',
      home: TodosScreen(
        todos: List.generate(
          20,
          (i) => Todo(
            'Todo $i',
            'A description for Todo $i',
          ),
        ),
      ),
    ),
  );
}
```

Enviando datos a una nueva ventana

```
class TodosScreen extends StatelessWidget {
  const TodosScreen({super.key, required this.todos});

  final List<Todo> todos;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Todos'),
      ),
      body: ListView.builder(
        itemCount: todos.length,
        itemBuilder: (context, index) {
          return ListTile(title: Text(todos[index].title),
            onTap: () {
              Navigator.push(context,
                MaterialPageRoute(
                  builder: (context) => DetailScreen(todo: todos[index]),
                ),
              );
            });}));}}
```

```
class DetailScreen extends StatelessWidget {
  const DetailScreen({super.key, required
this.todo});

  final Todo todo;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(todo.title),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Text(todo.description),
      ),
    );
  }
}
```

Usando rutas nombradas

Las aplicaciones con requisitos de navegación simple y enlaces pueden usar **Navigator** para navegar y el parámetro **MaterialApp.routes** para enlaces.

Nota: No se recomienda usar rutas nombradas para la mayoría de las aplicaciones.

```
@override
  Widget build(BuildContext context) {
  return MaterialApp(
    routes: {
      '/': (context) => HomeScreen(),
      '/details': (context) => DetailScreen(),
    },
  );
}
. . .

onPressed: () {
  Navigator.pushNamed(context, '/details');
}
```

Usando rutas nombradas (Ejemplo)

```
class FirstScreen extends StatelessWidget {
  const FirstScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Screen'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pushNamed(context, '/second');
          },
          child: const Text('Launch screen'),
        ),
      ),
    );
  }
}
```

```
class SecondScreen extends StatelessWidget {
  const SecondScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Second Screen'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: const Text('Go back!'),
        ),
      ),
    );
  }
}
```

Universidad Tecnológica de Pereira

MisionTIC 2022

MINISTERIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

Animaciones

Animaciones

Cuando hablamos de **User Experience (UX)** las buenas animaciones son claves para hacer nuestra aplicación más agradable e intuitiva.

Las animaciones son una parte muy importante de la experiencia de usuario ya que son el punto de interacción entre el usuario y la máquina.

Las animaciones sirven para conectar el usuario con la interfaz de la aplicación, dar una sensación de fluidez, mostrar los cambios de estado y mucho más.

https://docs.flutter.dev/development/ui/animations/tutorial

Herramientas para realizar animaciones

Animation

Está compuesto de un valor genérico, usualmente *double*, y de un estado.

El estado indica si la animación está en ejecución, pausada, en reversa, detenida, en el inicio o final. Esta clase también permite que otros objetos detecten cambios en su valor o estado.

Un objeto de la clase **Animation** no trabaja el renderizado o el build de ningún widget; por lo que no sabe nada sobre lo que se muestra en la pantalla.

Curve

Define la velocidad de cambio de una animación en el tiempo.

Esto nos permite acelerar o desacelerar una animación y no solo ejecutarla a velocidad constante.

La aceleración en nuestras animaciones provee de fluidez y naturalidad a la misma.

Flutter nos provee una colección de curvas en la clase Curves.

Herramientas para realizar animaciones

Interval

Representa una Curve cuya función es trasladada entre los valores *begin* y *end*.

begin y *end* son puntos porcentuales de la duración, es decir, para begin: 0.4 y end: 0.6 estamos trasladando la curva entre el 40% y 60% de la duración.

Antes de este rango el *value* de la animación es inicio y después de este rango el del fin.

CurvedAnimation

Es útil cuando necesitamos trabajar una curva no lineal, sobre todo si vamos a definir curvas diferentes en el avance y retroceso de la animación.

Es de tipo **Animation<double>**, puedes pasarlas de forma intercambiable.

El objeto CurvedAnimation envuelve el objeto que está modificando.

Herramientas para realizar animaciones

AnimationController

Es un objeto especial de **Animation** el cual genera un nuevo valor cada vez que se está listo para un nuevo frame.

Hereda de Animation<double>, por lo que puede usarse donde se necesite un objeto Animation.

Adicionalmente, nos proporciona métodos extras para controlar la animación. El constructor recibe un argumento Vsync el cual previene animaciones fuera de pantalla para no realizar consumo de recursos innecesarios.

Tween

Su única función es definir un mapeo entre un rango de entrada y un rango de salida.

Este objeto no almacena ningún estado o valor. En cambio, provee el método evaluate que se encarga de aplicar el mapeado al *value* de la animación.

Provee el método animate. Este método nos devuelve un Animation que está controlada por el Animation dado pero que su valor está determinado por el Tween.