ESCUELA DE SISTEMAS Y TECNOLOGÍAS

Transparencias de **ANALISTA DE SISTEMAS**
Edición 2023 - Materia: **Aplicaciones Móviles con Flutter**

TEMA: Layouts

Agenda

- Introducción
- ListView, ListTile y Card
- GridView y GridTile
- Stack y Positioned
- Table, TableRow y TableCell
- PageView
- Agregar Efectos
  - ClipRRect, ClipOval y ClipPath
  - Transform
- Agregar Scroll
  - SingleChildScrollView
  - CustomScrollView y Slivers

Introducción (1)

- Flutter permite crear interfaces de usuario atractivas y fluidas.
- Los widgets de layout en Flutter son componentes fundamentales para organizar y estructurar la interfaz de usuario de manera eficiente.
- Son componentes especializados que se utilizan para estructurar y posicionar otros widgets.
- Ayudan a controlar el flujo y la disposición de los elementos en la pantalla.
- Algunos de estos widgets ya los hemos visto anteriormente al estudiar los widgets básicos (Container, Column, Row, etc.), y otros serán vistos a continuación.

Introducción (2)

- También veremos algunos *widgets* para agregar efectos sobre otros, como redondear sus vértices o aplicarles transformaciones para cambiar su rotación, escala, traslación, etc.
- Finalmente aprenderemos a agregar *Scroll* o desplazamiento a los *widgets* que no cuentan con esta capacidad de manera predeterminada.

ListView (1)

- El *widget* **ListView** muestra una lista de elementos (que son otros *widgets*).
- Los elementos del *ListView* se pueden especificar de diferentes maneras.
- Si los elementos se conocen de antemano se pueden cargar a través del parámetro *List<Widget> children*.
- Para generar los elementos dinámicamente se pueden utilizar los constructures *ListView.builder(...)* y *ListView.separated(...)*.
- Estos constructores reciben un parámetro requerido *itemBuilder* al que se le debe pasar una función que se utilizará para generar los elementos.

ListView (2)

- Dicha función recibe el *BuildContext* y el índice del elemento como un entero, y devuelve el *widget* del elemento o *null*.
- Otro parámetro útil de ambos constructores es *itemCount* al que se le debe suministrar un entero con la cantidad de elementos a generar.
- El constructor *ListView.separated(...)* recibe además un parámetro requerido *separatorBuilder* similar al parámetro *itemBuilder*, pero que se utiliza para generar los *widgets* separadores de los elementos de la lista (la función que genera los separadores no puede devolver *null*).

ListTile (1)

- Un *widget* muy utilizado para generar los elementos de un *ListView* es el **ListTile**.
- Tiene parámetros para indicar un título y un subtítulo: *Widget? title* y *Widget? subtitle*.
- También permite mostrar un *widget* antes del título y después de éste: *Widget? leading* y *Widget? trailing*.
- Se puede cambiar el color de texto del título y del subtítulo mediante el parámetro *Color? textColor*.
- O cambiar el color de fondo tanto si el elemento está sin seleccionar o seleccionado: *Color? tileColor* y *Color? selectedTileColor*.
- El parámetro booleano *selected* establece si el elemento se encuentra seleccionado o no.

ListTile (2)

- El color de los textos y los iconos cuando el elemento está seleccionado se define con el parámetro *Color? selectedColor*.
- Para definir el comportamiento del elemento al tocarlo se utiliza el parámetro *onTap* que recibe una función sin parámetros y sin retorno.

Card

- Otro *widget* interesante para mostrar información dentro de una lista puede ser el *widget* **Card**.
- Muestra cualquier tipo de información que se coloque dentro del mismo, en forma de tarjeta.
- Su parámetro *child* permite especificar su contenido.
- Si se desea mostrar más de un elmento hijo se puede utilizar un *Column* o *Row* por ejemplo para contener otros elementos como *Text*, *Button* o incluso *ListTile*.
- Mediante el parámetro *double? elevation* se puede personalizar qué tan notorio es el efecto de sombra detrás de la tarjeta.

GridView (1)

- Para mostrar una grilla de elementos existe el *widget* **GridView**.
- Requiere un parámetro *SliverGridDelegate gridDelegate* que controla la disposición de los elementos hijos. Puede ser:
  - *SliverGridDelegateWithFixedCrossAxisCount*: Crea el *layout* con una cantidad fija de elementos en el eje secundario de la grilla. Requiere el parámetro *int crossAxisCount*.
  - *SliverGridDelegateWithMaxCrossAxisExtent*: Crea el *layout* con elementos que ocupan un tamaño máximo definido del eje secundario de la grilla. Requiere el parámetro *int maxCrossAxisExtent*.

GridView (2)

- *GridView* dispone de un constructor con nombre *GridView.count* que no requiere definir el parámetro *gridDelegate*, pero requiere definir un parámetro *int crossAxisCount*.
- Los constructores de los *SliverGridDelegate* y el constructor *GridView.count* también ofrecen los parámetros *mainAxisSpacing* y *crossAxisSpacing* para definir el espaciado entre los elementos.
- El parámetro children de *GridView* y *GridView.count* permite cargar los elementos que se mostrarán en la grilla, si éstos se conocen de antemano.
- El constructor *GridView.builder* se utiliza para generar los elementos dinámicamente.

GridView (3)

- Además del parámetro *gridDelegate*, también requiere un parámetro *itemBuilder* que recibe una función para generar los elementos.
- Dicha función recibe el *BuildContext* y el índice del elemento como un entero, y devuelve el *widget* del elemento o *null*.
- Otro parámetro útil de este constructor es *itemCount* al que se le debe suministrar un entero con la cantidad de elementos a generar.

GridTile

- El widget **GridTile** se utiliza para mostrar un elemento dentro de una grilla.
- Contiene un elemento hijo para mostrar su contenido (parámetro requerido *Widget child*).
- Su constructor recibe además los parámetros *Widget? header* y *Widget? footer* para mostrar opcionalmente un cabezal y un pie en cada elemento.

Stack (1)

- El *widget* **Stack** permite apilar varios *widgets* uno encima del otro, lo que facilita la composición de diseños complejos.
- Coloca sus hijos uno encima del otro en el orden de codificación.
- Los *widgets* pueden superponerse y posicionarse de manera relativa o absoluta.
- Los hijos se dimensionan automáticamente según su contenido, a menos que se especifique lo contrario.
- Es posible anidar varios *widgets Stack* para lograr diseños más complejos.

Stack (2)

- Los parámetros más importantes de *Stack* para controlar su apariencia y comportamiento son:
  - **alignment**: Determina cómo se alinean los hijos dentro del *Stack*.
  - **fit**: Determina cómo se ajustan los hijos dentro del espacio disponible.
  - **overflow**: Especifica cómo se comportan los hijos cuando se desbordan del *Stack*.
  - **textDirection**: define la dirección del texto dentro del *Stack*.

Positioned

- Por su parte el widget **Positioned** nos permite controlar la posición y el tamaño de un *widget* dentro de un *Stack*.
- Para utilizar el *widget Positioned*, se debe envolver el *widget* secundario con él, dentro del *Stack*.
- Luego se establecen los valores de las propiedades *top*, *bottom*, *left*, *right*, *width* y *height* según las necesidades de diseño.
- El *widget Positioned* nos permite crear diseños complejos y personalizados al posicionar y superponer múltiples *widgets* secundarios dentro del *Stack*.

Table, TableRow y TableCell (1)

- El widget **Table** se utiliza para mostrar datos tabulares en nuestras aplicaciones *Flutter*.
- Es un contenedor que organiza sus hijos en filas, y éstas a su vez organizan sus hijos en celdas o columnas (parámetro *children*).
- Se puede definir el número de filas y columnas de la tabla y ajustar el diseño según las necesidades.
- El *widget Table* proporciona flexibilidad en el diseño, permitiendo que las filas y columnas tengan diferentes tamaños y alineaciones.
- Las filas que forman parte de la tabla se definen mediante el *widget* **TableRow** y las celdas con el *widget* **TableCell**, a través del parámetro *children*.

Table, TableRow y TableCell (2)

- Se pueden personalizar los bordes de la tabla (parámetro *border*), el ancho predeterminado de las columnas (parámetro *defaultColumnWidth*), los anchos de ciertas columnas en particular (parámetro *columnWidths*), la alineación vertical predeterminada de las celdas (parámetro *defaultVerticalAlignment*).
- A las filas se les puede personalizar su borde, color de fondo, etc., mediante el parámetro *decoration*.
- A cada celda se le puede personalizar su alineación vertical mediante el parámetro *verticalAlignment*, y definir su contenido mediante el parámetro *child*.

PageView (1)

- El *widget* **PageView** proporciona una forma conveniente de desplazarse por una colección de *widgets* de forma horizontal o vertical (parámetro *scrollDirection*), similar a una vista de páginas en una aplicación.
- Brinda una interfaz de usuario intuitiva y atractiva para presentar contenido en varias páginas.
- Tiene amplias opciones de personalización para adaptarse a las necesidades de diseño de la aplicación.
- Es de fácil implementación y uso, lo que permite ahorrar tiempo y esfuerzo en el desarrollo de interfaces de usuario interactivas.

PageView (2)

- Permite deslizarse sin problemas entre las páginas utilizando gestos de deslizamiento.
- Proporciona automáticamente indicadores de paginación para mostrar la posición actual del usuario en la colección de *widgets*.
- Puede contener cualquier tipo de *widget* como contenido de página (parámetro *children*), lo que permite una gran variedad de diseños y personalizaciones.
- Permite controlar manualmente el desplazamiento a través de métodos como *nextPage()*, *previousPage()* o *jumpToPage()*.

ClipRRect (1)

- **ClipRRect** es un *widget* que permite recortar o redondear las esquinas de otros *widgets*.
- Proporciona una forma sencilla de aplicar efectos de borde redondeado a imágenes, contenedores, cajas de texto y más.
- El uso básico de *ClipRRect* implica envolver un *widget* dentro de él y especificar el radio de las esquinas que deseamos redondear.
- Por ejemplo, podemos envolver una imagen con *ClipRRect* y establecer un radio de 10 para redondear las esquinas, lo que dará como resultado una imagen con bordes suavemente redondeados.

ClipRRect (2)

- *ClipRRect* tiene varios parámetros que se pueden ajustar para personalizar su comportamiento:
  - **borderRadius**: Define el radio de las esquinas que queremos redondear.
  - **child**: El *widget* hijo que será recortado o redondeado por *ClipRRect*.
  - **clipBehavior**: Especifica cómo se debe recortar el *widget* hijo.

ClipOval

- Por otra parte el widget **ClipOval** permite recortar un *widget* hijo en forma de óvalo o círculo.
- *ClipOval* envuelve el *widget* hijo con una forma elíptica y recorta todo lo que se encuentra fuera de ésta, creando un efecto de recorte circular o elíptico.
- Es ideal para dar énfasis visual a elementos como imágenes de perfil o iconos circulares.
- Para lograr un recorte circular perfecto, el *widget* hijo debe tener las mismas dimensiones en ambos ejes (alto y ancho).
- Cuenta con los parámetros *child* y *clipBehavior* similares a los del *widget ClipRRect*.

ClipPath

- El *widget* **ClipPath** permite recortar un área específica de un *widget* hijo utilizando una forma personalizada definida por un *Path* (trazado).
- *ClipPath* brinda mayor flexibilidad que *ClipRRect* y *ClipOval* al permitir definir una forma de recorte más compleja.
- En su parámetro *clipper* se debe pasar un objeto de tipo *CustomClipper<Path>* para definir la forma del recorte mediante su método *getClip(Size)* que devuelve el *Path* personalizado.
- El *Path* se personaliza utilizando métodos como *moveTo(...)*, *lineTo(...)*, *close()*, etc.

Transform (1)

- El *widget* **Transform** se utiliza para aplicar transformaciones geométricas a otros *widgets*.
- Estas transformaciones incluyen rotación, escala, traslación y deformación:
  - **Reflejo**: Refleja el widget de manera horizontal y/o vertical.
  - **Rotación**: Rota el widget alrededor de un punto central.
  - **Escala**: Cambia el tamaño del widget en función de un factor de escala.
  - **Traslación**: Desplaza el widget horizontal o verticalmente.

Transform (2)

- El *widget Transform* cuenta con varios constructores (además del constructor sin nombre), para aplicar diferentes transformaciones sobre su elemento hijo:
  - *Transform.flip(flipX: ..., flipY: ..., child: ...)*
  - *Transform.rotate(angle: ..., child: ...)*
  - *Transform.scale(scale: ..., child: ...)*
  - *Transform.translate(offset: ..., child: ...)*
- También se puede aplicar en el parámetro *transform*, una matriz de transformación obteniéndola con *Matrix4.identity()* ajustándola luego con métodos como *rotateZ(...)*, *scale(...)*, *translate(...)*, etc.

SingleChildScrollView (1)

- **SingleChildScrollView** es un *widget* de desplazamiento que permite desplazar contenido que excede el tamaño de la pantalla.
- Proporciona una vista de desplazamiento (vertical u horizontal) para su contenido secundario definido como único hijo (en el parámetro *child*).
- ¿En qué situaciones se utiliza?
  - Cuando el contenido a mostrar excede el tamaño visible y no cabe en una sola pantalla.
  - Cuando se necesita implementar un desplazamiento personalizado en un diseño específico.

SingleChildScrollView (2)

- Entre sus parámetros más comunes encontramos:
  - **child**: Define el contenido que se desplazará.
  - **scrollDirection**: Especifica la dirección del desplazamiento (vertical u horizontal).
  - **physics**: Controla la física del desplazamiento, como el rebote y la fricción.
- La estructura del contenido dentro del *SingleChildScrollView* debe ser compatible con el desplazamiento (tener tamaño suficiente, no tener elementos posicionados de manera fija, etc.).
- Si se utiliza *SingleChildScrollView* dentro de otro *widget* de desplazamiento, como *ListView*, puede haber conflictos.

CustomScrollView y Slivers (1)

- **CustomScrollView** es un *widget* que proporciona una forma flexible de crear desplazamiento personalizado en función de las necesidades de la interfaz de usuario.
- Permite crear listas desplazables y desplazamientos bidireccionales (horizontales, verticales, o combinar ambos) utilizando una combinación de diferentes *slivers*.
- Los **slivers** son fragmentos de contenido desplazable que se pueden combinar dentro de *CustomScrollView*. Estos fragmentos de contenido pueden tener diferentes comportamientos de desplazamiento, disposición y apariencia.

CustomScrollView y Slivers (2)

- Algunos ejemplos de *slivers* son:
  - **SliverList**: Renderiza una lista de *widgets* de forma vertical u horizontal, según la dirección del desplazamiento.
  - **SliverGrid**: Renderiza una cuadrícula de *widgets*, colocándolos en filas y columnas.
  - **SliverAppBar**: Representa una barra de aplicación que se desplaza fuera de la vista a medida que se desplaza el contenido. Puede contener elementos como título, acciones y pestañas.
  - **SliverToBoxAdapter**: Envuelve un único *widget* que no es un *sliver* y lo convierte en un *sliver*.

CustomScrollView y Slivers (3)

- *CustomScrollView* utiliza una técnica de construcción perezosa, lo que significa que solo se renderizan los elementos visibles en la pantalla.
- Esto mejora el rendimiento y reduce el consumo de memoria, especialmente para listas grandes.
- *CustomScrollView* permite combinar diferentes *slivers* y personalizar completamente la experiencia de desplazamiento.
- Esto brinda un control total sobre la apariencia y el comportamiento de la interfaz de usuario desplazable.