ESCUELA DE SISTEMAS Y TECNOLOGÍAS

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

TEMA: Rutas y Navegación

Agenda

- Introducción
- Definir Rutas
- Navegación
- Pasaje de Argumentos
- PopupMenuButton y PopupMenuItem
- Drawer
- TabBar, DefaultTabBarController, Tab y TabBarView
- BottomNavigationBar y BottomNavigationBarItem

Introducción (1)

- Las aplicaciones *Flutter* pueden tener varias pantallas o páginas, de tal manera que sus funcionalidades se puedan organizar en grupos ordenados y concisos.
- El usuario puede entonces navegar entre las diferentes páginas para utilizar diferentes funcionalidades de la aplicación.
- Las páginas se identifican mediante rutas que pueden ser previamente definidas (o no), vinculadas con el *widget* de la página correspondiente.
- Las rutas son manejadas por el *widget Navigator* (ya disponible en el árbol de *widgets* al instanciar el *WidgetsApp* (o *MaterialApp*, etc.).

Introducción (2)

- El *widget Navigator* es un *stateful widget* que utiliza una pila (colección *LIFO, Last In First Out*) de rutas para recordar las páginas por las que el usuario va navegando, siendo la página actual la que se encuentra en el punto más alto de dicha pila.

Definir Rutas (1)

- Las rutas correspondientes a las diferentes páginas de la aplicación se definen en el parámetro **routes** al construir el *widget* de la aplicación.
- Este parámetro recibe un mapa cuya clave es un *String* con el nombre de la ruta, y su valor es una función que recibe por parámetro el *BuildContext* y debe retornar el *widget* correspondiente a la página.
- De esta manera, al referenciar el nombre de la ruta, se podrá navegar a la página definida mediante el *widget* especificado.
- El *widget* de la aplicación también tiene un parámetro **initialRoute** en el que se establece el nombre de la ruta que debe mostrarse al iniciar la aplicación.

Definir Rutas (2)

- Típicamente se utiliza el nombre de ruta '/' para identificar la ruta inicial de la aplicación.
- Si el mapa de rutas contiene una ruta llamada '/' no debe utilizarse el parámetro *home* (aún cuando no la tuviere, si se trabaja con rutas se aconseja utilizar siempre el parámetro *initialRoute* para definir la página inicial).
- El parámetro **onGenerateRoute** del *widget* de la aplicación recibe una función que a partir de la información de un objeto *RouteSettings*, construye una ruta (típicamente de tipo *MaterialPageRoute*).
- Dicho método se ejecuta si se intenta navegar a una ruta que no está definida en el mapa *routes*.

Definir Rutas (3)

- Esto permite, por ejemplo, generar una ruta para mostrar una página en caso de intentar navegar a una ruta inexistente.
- A la instancia de **MaterialPageRoute** se le pasa (entre otros parámetros opcionales) una función requerida que recibe el *BuildContext* y retorna el *widget* correspondiente a la página a la que se navegará.
- Es una práctica común que los valores suministrados a los parámetros vistos anteriormente (*routes*, *initialRoute*, *onGenerateRoute*, etc.) provengan de atributos y métodos estáticos definidos en una clase específica para tal fin, como una forma de organizar el código correspondiente al manejo de rutas.

Navegación (1)

- Para navegar hacia una ruta se utiliza el *widget* **Navigator**.
- Para obtener este widget se puede utilizar el método estático *of(BuildContext)* de la clase *Navigator*.
- Algunos de los métodos que ofrece *Navigator*:
  - ***push(Route<T>)***: Navega hacia la ruta pasada por parámetro (típicamente un objeto *MaterialPageRoute*) y agrega la ruta a la pila de navegación.
  - ***pushNamed(String)***: Idem al anterior, pero utiliza el nombre de una ruta previamente definida.

Navegación (2)

- Algunos de los métodos que ofrece *Navigator* (cont.):
  - ***pushReplacement(Route<T>)***: Navega hacia la ruta pasada por parámetro (típicamente un objeto *MaterialPageRoute*), reemplazando la ruta actual en la pila de navegación por dicha ruta.
  - ***pushReplacementNamed(String)***: Idem al anterior, pero utiliza el nombre de una ruta previamente definida.
  - ***pop()***: Quita la ruta actual de la pila de navegación y navega hacia la ruta anterior.

Pasaje de Argumentos (1)

- Al navegar hacia otra ruta, es posible pasar un parámetro **arguments** de tipo *Object?* con información útil para la página de destino.
- Este parámetro se puede pasar directamente en los métodos *pushNamed(...)* y *pushReplacementNamed(...)*.
- Los argumentos también se pueden pasar al instanciar un objeto *MaterialPageRoute* mediante el parametro *settings* que recibe un objeto *RouteSettings?*, cuyo constructor a su vez recibe los parámetros con nombre *String? name* con el nombre de la ruta, y *Object? arguments* con los argumentos.

Pasaje de Argumentos (2)

- Para recibir los argumentos en la página de destino se utiliza un objeto *Route* o *ModalRoute* obtenido del contexto, y su propiedad *settings*, cuyo valor es un objeto de tipo *RouteSettings* que dispone de la propiedad *arguments* (de tipo *Object?*) con los argumentos enviados:

  Route.of(context).settings.arguments;

  ModalRoute.of(context).settings.arguments;

PopupMenuButton y PopupMenuItem (1)

- El *widget* **PopupMenuButton** es un componente de interfaz de usuario que despliega un menú emergente cuando se toca o hace clic en él.
- Este menú emergente proporciona una lista de opciones para que los usuarios interactúen y realicen acciones adicionales dentro de nuestra aplicación.
- Ofrece una amplia gama de opciones de personalización. Podemos adaptar su apariencia y comportamiento para que se ajuste a los requisitos de la aplicación, incluyendo la apariencia del botón y del menú emergente.

PopupMenuButton y PopupMenuItem (2)

- El menú emergente puede contener una lista de opciones, como ítems de menú (*widget* **PopupMenuItem**), acciones o cualquier otro elemento interactivo. Estas opciones pueden ser textos, iconos u otros *widgets* personalizados.
- El menú emergente puede mostrarse cuando se toca o hace clic en el *widget PopupMenuButton*, brindando una forma intuitiva de proporcionar opciones adicionales sin ocupar espacio en la interfaz principal.

PopupMenuButton y PopupMenuItem (3)

- Los parámetros más utilizados de *PopupMenuButton* son los siguientes:
  - **itemBuilder**: Es requerido y define la función que construye los elementos del menú emergente (*PopupMenuItem*). Recibe un contexto (*BuildContext*) y debe devolver una lista de objetos *PopupMenuEntry*.
  - **onSelected**: Función que se invoca cuando un elemento del menú se selecciona. Recibe como argumento el valor (*value*) del elemento seleccionado.
  - **icon**: Especifica el ícono que se mostrará en el botón de *PopupMenuButton*.

PopupMenuButton y PopupMenuItem (4)

- Los parámetros más utilizados de *PopupMenuItem* son los siguientes:
  - **value**: Es requerido y representa el valor asociado al elemento del menú. Este valor se pasa a la función *onSelected* cuando el elemento se selecciona. Puede ser un valor de cualquier tipo.
  - **child**: Es el *widget* que se muestra como contenido del elemento del menú.
  - **enabled**: Especifica si el elemento del menú está habilitado o deshabilitado. Si se establece en *false*, el elemento del menú no responderá a eventos de selección y se mostrará con un estilo deshabilitado.

Drawer (1)

- El *widget* **Drawer** es un contenedor deslizable que se utiliza para mostrar opciones de navegación y contenido adicional.
- Se encuentra ubicado en el lateral de la pantalla y se puede abrir y cerrar mediante gestos o mediante acciones específicas, como un botón de menú.
- Proporciona una forma intuitiva y eficiente de acceder a diferentes secciones o funcionalidades de la aplicación.
- Se puede adaptar a la identidad visual de la aplicación y personalizar su apariencia.
- Permite mostrar diferentes opciones de navegación, como enlaces a secciones, ajustes, etc.

Drawer (2)

- Los usuarios pueden abrir y cerrar el *Drawer* de manera sencilla mediante gestos o acciones específicas.
- Proporciona espacio adicional para mostrar contenido que no es esencial pero puede resultar útil o informativo.
- Mejora la navegación y la accesibilidad de la aplicación.
- Optimiza el espacio en pantalla al ocultar opciones no esenciales hasta que sean necesarias.
- Permite una organización jerárquica de la información y funcionalidades, facilitando la usabilidad de la aplicación.

Drawer (3)

- Para implementarlo se debe utilizar un *Scaffold* como estructura principal de la página.
- Para incorporar un *Drawer* al *Scaffold* se utiliza el parámetro **drawer** (existe también **endDrawer**), al que se le pasa un *widget* de tipo *Drawer*.
- Se puede abrir y cerrar deslizando desde el borde.
- Si se utiliza una *AppBar* en el *Scaffold*, ésta también incorporará un botón para abrir el *Drawer*.
- Para abrirlo programáticamente se utiliza el método *openDrawer()* del *Scaffold* así:

```
Scaffold.of(context).openDrawer();
Scaffold.of(context).openEndDrawer();
```

Drawer (4)

- Para cerrarlo programáticamente se utiliza el método *pop()* del *widget Navigator* así:

  Navigator.of(context).pop();

- En el parámetro **child** del *Drawer* se puede especificar cualquier *widget*, aunque típicamente se utiliza un *ListView*, y dentro de éste *widgets* de tipo *ListTile* para definir las opciones de navegación.
- También se puede utilizar dentro del *ListView*, un *widget* de tipo *DrawerHeader* para definir el cabezal del *Drawer*.

TabBar, DefaultTabController, Tab y TabBarView (1)

- Exploraremos los widgets *TabBar*, *DefaultTabController*, *Tab* y *TabBarView* y su papel en la creación de interfaces de usuario con pestañas.
- El *widget* **TabBar** proporciona una barra de pestañas en la parte superior de la pantalla, lo que permite al usuario navegar fácilmente entre diferentes secciones de contenido.
- **DefaultTabController** es un controlador de pestañas que se utiliza con *TabBar*. Maneja la interacción entre las pestañas y permite al usuario cambiar de una pestaña a otra mediante gestos o acciones de selección.

TabBar, DefaultTabController, Tab y TabBarView (2)

- El *widget* **Tab** define una pestaña individual dentro de un *TabBar*. Cada pestaña puede tener un título, un ícono, e incluso contenido personalizado.
- **TabBarView** es un *widget* que muestra el contenido correspondiente a la pestaña seleccionada en un *TabBar*. Cada pestaña tiene su propio contenido, que puede ser una vista simple o incluso un árbol de *widgets* más complejo.

TabBar, DefaultTabController, Tab y TabBarView (3)

- Para crear una interfaz de usuario con pestañas se deben seguir los siguientes pasos:
  - Crear un *DefaultTabController* en un punto del árbol que la haga disponible para los elementos hijos *TabBar*, *Tab* y *TabBarView*. Establecer su parámetro *length* con la cantidad de pestañas a mostrar.
  - Crear un *TabBar* (típicamente pasándolo al parámetro *bottom* de un *AppBar* dentro de un *Scaffold*).
  - En la propiedad *tabs* del *TabBar* definir una lista de *widgets Tab* (tantos como se haya indicado en el *DefaultTabController*).

TabBar, DefaultTabController, Tab y TabBarView (4)

- Para crear una interfaz de usuario con pestañas se deben seguir los siguientes pasos (cont.):
  - Crear el contenido de cada pestaña dentro de un *widget TabBarView* (típicamente pasándolo en el parámetro *body* de un *Scaffold*). En el parámetro *children* se definen los *widgets* correspondientes a cada pestaña (respetando la cantidad y orden).

BottomNavigationBar y BottomNavigationBarItem (1)

- El *widget* **BottomNavigationBar** consta de una barra de navegación en la parte inferior de la pantalla que permite al usuario cambiar entre diferentes vistas o secciones de una aplicación.
- Está compuesto por una lista de ítems (*widget* **BottomNavigationBarItem**), cada uno de los cuales representa una sección o vista de la aplicación. Se pueden agregar etiquetas opcionales a cada ítem para proporcionar información adicional al usuario.
- El *BottomNavigationBar* destaca el ítem seleccionado, lo que facilita al usuario identificar en qué sección de la aplicación se encuentra actualmente.

BottomNavigationBar y BottomNavigationBarItem (2)

- Al hacer clic en un ítem del *BottomNavigationBar*, se puede cambiar de una vista a otra de forma suave y sin problemas. Esto permite una navegación fluida dentro de la aplicación.
- Se puede personalizar para adaptarse al diseño y estilo de la aplicación. Se pueden ajustar colores, iconos, etiquetas y más para lograr una apariencia coherente.
- Los parámetros más utilizados de *BottomNavigationBar* son:
  - **currentIndex**: Establece el índice del item seleccionado actualmente, que se mostrará resaltado.

BottomNavigationBar y BottomNavigationBarItem (3)

- Los parámetros más utilizados de *BottomNavigationBar* son (cont.):
  - **items**: Lista de items *BottomNavigationBarItem* que componen la barra. A cada item se le puede definir un icono (parámetro *icon*), un texto (parámetro *label*), etc.
  - **onTap**: Función que se dispara al tocar un item de la barra. Recibe por parámetro el índice del item que se tocó.