

Programación Multimedia y Dispositivos Móviles

UP3.2: Tabs y Drawers

Autor: Antonio Calabuig Puigvert y Sebastián Villa Ponce

Centro: IES Salvador Gadea

Departamento: Informática

Ciclo: Desarrollo de Aplicaciones Multiplataforma

Fecha: 2025-11-09



UP3.2: Tabs, Drawers y Icons

Elementos de navegación

- Programación Multimedia y Dispositivos Móviles
 - UP3.2: Tabs y Drawers
 - UP3.2: Tabs, Drawers y Icons
 - 1. Introducción
 - 2. Tabs
 - 2.1. Asociar a cada Tab el Stack correspondiente
 - 3. Personalizar los Tabs
 - 4. Drawer - Menú lateral
 - 5. Drawer - Personalización
 - 6. DrawerContent
 - 7. Conectar Drawer con Tabs y con Stack
 - 8. Optimización de nombres de directorios
 - 9. Mostrar Drawer de forma controlada
 - 10. Cambiar título basado en los argumentos
 - 11. Conclusiones
 - 12. Bibliografía
-

1. Introducción

En esta parte del temario, vamos a reforzar la navegación de la actividad con un menú lateral (Drawer) y tabs que nos permitirán crear un menú inferior basado en Tabs.

También veremos como incluir Iconos y sobretodo personalizar tanto el drawer como las tabs que incluyamos. El objetivo principal es reforzar los conocimientos sobre navegación y la estructura de ficheros. Cuando incluimos (), [] y cuando no y cuáles son los efecto sobre la visualización de los mismos en las rutas.

Para el desarrollo de esta sección el punto de partida será la aplicación desarrollada en el punto anterior. Si no la tenéis completa, está enlazada en el entorno virtual de aprendizaje (Aules).

2. Tabs

En esta sección de este tema vamos a abordar en primer lugar los **Tabs**, para los que tenéis aquí en enlace de consulta sobre la documentación oficial. En los ejemplos que trabajamos en clase siempre vemos opciones pero no todas y es conveniente siempre consultar la documentación para poder ver más opciones de configuración, funcionamiento y adaptabilidad a los requisitos de los distintos tipos de aplicaciones que podamos desarrollar.

En la estructura de ficheros, los **tabs** se almacenan dentro del directorio **app** entre () para indicar que no forman parte de la ruta.

Internamente crearemos un **_layout.tsx** para los **tabs** que será lo que definirá la estructura en la que se van a mostrar. Es decir, la vista del **BottomAppNavigation** que es como se llamaría en el sistema Android.

Es importante remarcar, que la configuración de los tabs puede ser muy personalizable incluso si dentro de una pantalla concreta queremos tener otros tabs diferentes a los que se ven en cualquier otra pantalla.

Vamos a crear dentro de **app** la carpeta **tabs**, sin paréntesis, para que forme parte de la ruta. En esta carpeta **tabs**, creamos el **_layout.tsx** de las tabs que vamos a implementar.

```

import React from 'react'
import { Text, View } from 'react-native'

const TabsLayout = () => {
  return (
    <View>
      <Text>TabsLayout</Text>
    </View>
  )
}

export default TabsLayout

```

Dentro de la carpeta `tabs` crearemos dos carpetas más `home` y `favorites` y en cada una de ellas un fichero `index.tsx` con la inicialización de cada vista como se muestra a continuación:

```

// home/index.tsx
import React from 'react'
import { Text, View } from 'react-native'

const HomeScreen = () => {
  return (
    <View>
      <Text>HomeScreen</Text>
    </View>
  )
}

export default HomeScreen

// favorites/index.tsx
import React from 'react'
import { Text, View } from 'react-native'

const FavoritesScreen = () => {
  return (
    <View>
      <Text>FavoritesScreen</Text>
    </View>
  )
}

export default FavoritesScreen

```

Ahora desde la documentación oficial de React Native en el apartado de tabs, copiamos la configuración de las tabs y las adaptamos a nuestra aplicación. Con esto modificaremos el `tabs/_layout.tsx`.

```

import { FontAwesome } from '@expo/vector-icons'
import { Tabs } from 'expo-router'
import React from 'react'

const TabsLayout = () => {
  return (
    <Tabs screenOptions={{ tabBarActiveTintColor: 'purple' }}>
      <Tabs.Screen
        name="home/index"
        options={{
          title: 'Home',
          tabBarIcon: ({ color }) => <FontAwesome size={28} name="home"
color={color} />,
        }}
      />
      <Tabs.Screen
        name="favorites/index"
        options={{
          title: 'Favorites',
          tabBarIcon: ({ color }) => <FontAwesome size={28}
name="heartbeat" color={color} />,
        }}
      />
    </Tabs>
  )
}

export default TabsLayout

```

Importamos `Tabs` y `FontAwesome` de `import { FontAwesome } from '@expo/vector-icons'` para poder cargar la funcionalidad de los iconos.

Realizamos la configuración necesaria para adaptarlo a nuestras vistas creadas dentro de cada subcarpeta de `tabs`.

Alternativamente a los iconos de `FontAwesome` que son populares también podemos usar `ionicons` que no tiene una amalgama de iconos tan grande como la de `FontAwesome` pero también nos pueden servir para el desarrollo de aplicaciones.

Los `ionicons` también vienen por defecto en el framework de React Native y vamos a hacer uso de estos, solo tenemos que cambiar el import y el uso en el `_layout.tsx` que nos ataña.

```
import { Ionicons } from '@expo/vector-icons'  
...  
tabBarIcon: ({ color }) => <Ionicons size={28} name="home-outline" color={color}>  
>,  
...  
tabBarIcon: ({ color }) => <Ionicons size={28} name="star-outline" color={color}>  
>,  
...
```

Con la propiedad `screenOptions` en su atributo `tabBarActiveTintColor` de `<Tabs></Tabs>` podemos cambiar el color que va a tener nuestro ícono cuando su vista esté activa. El active tint, es una propiedad que ya viene implementada de facto y que por norma general nos quita trabajo de implementación.

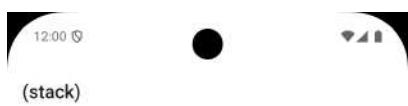
Otro atributo destacable es el de esconder la etiqueta de cada ícono del TabBar `tabBarShowLabel`, de forma que podemos dejar que los íconos hablen por sí solos. Inicializaremos el valor a `false`.

Probadlo en el simulador.

2.1. Asociar a cada Tab el Stack correspondiente

En primer lugar, si lo que queremos hacer es que el stack se muestre dentro de cada Tab correspondiente, lo primero será modificar el **filesystem** de nuestra aplicación.

1. Movemos la carpeta (**stack**) dentro de **tabs**.



Home



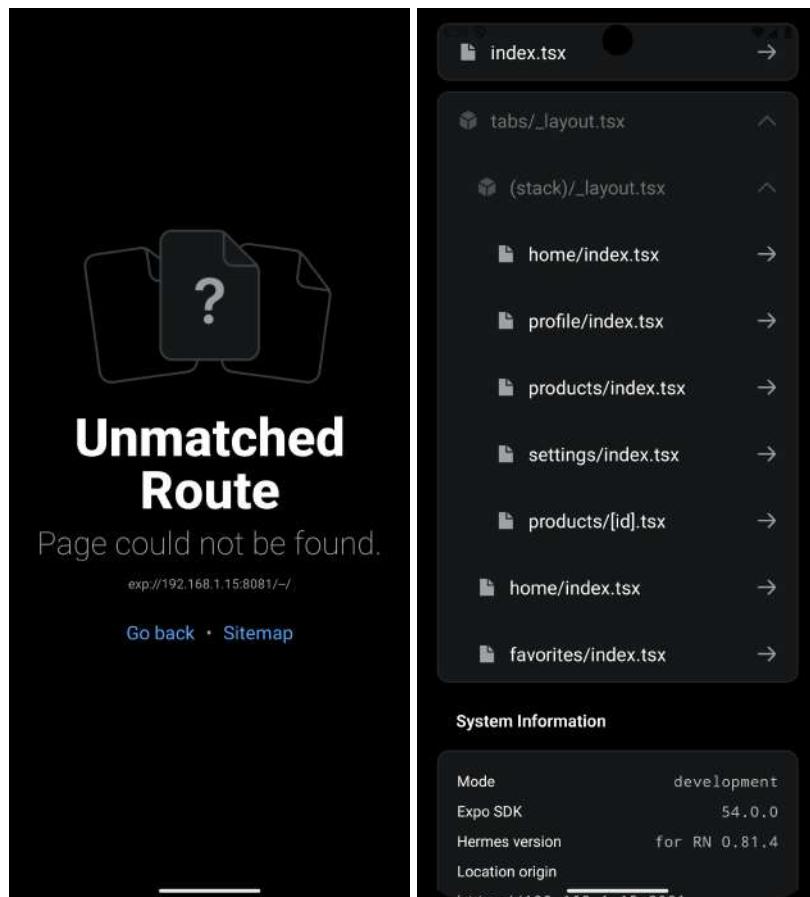
2. Adaptamos para que nuestro Tab coja el stack introducido.

```
<Tabs.Screen
  name="(stack)"
  options={{
    title: 'Stack',
    tabBarIcon: ({ color }) => <Ionicons size={28} name="file-tray-
  stacked-outline" color={color} />,
  }}
/>
```

Si os fijáis veréis que hay un doble header, el del stack y el de la página de inicio. Pues bien ocultemos el header del stack ubicado en [app/tabs/\(stack\)/_layout.tsx](#).

```
<Stack screenOptions={{
  headerShown: false,
  headerShadowVisible: false,
  contentStyle: {
    backgroundColor: 'white'
  }
}>
```

Si ahora intentamos navegar a productos veréis que da un error, eso es porque intentamos navegar a una ruta que no existe. En el debugger propio de Expo podemos observar las rutas disponibles en el sistema:



Para solucionar este problema, tendremos que ir al `app/tabs/(stack)/home/index.tsx` y aplicar los cambios necesarios para arreglar las rutas y que puedan alcanzar cada una de las pantallas asociadas a los botones de la `HomeScreen`.

```
...
<CustomButton className='mb-2' color='primary'
  onPress={() => router.push('/tabs/products')}
  Products
/>
...
```

Aplicaremos también los mismos cambios en la página de `app/tabs/(stack)/products/index.tsx`.

```
...
<Link href={`/tabs/(stack)/products/${item.id}`} className='text-primary'>
  Ver detalles
</Link>
...
```

Si nos fijamos, al ocultar el header y dejar el del Stack principal ahora no aparece la flecha de vuelta atrás. Tenemos dos tipos de Header ahora mismo. Uno oculto y el otro visible. Dejaremos ambos visibles.

```
...
`headerShown: false,` en el StackLayout (`app/tabs/(stack)/layout.tsx`)
...
```

Podríamos crear un componente personalizado que controle la navegación de la aplicación con todos los condicionantes de cuando puede volver atrás o no. Pero en nuestro caso, la implementación actual ya lo controla y vamos a decidir con cuál de los dos headers nos quedamos y si tenemos que incluir alguna personalización sobre lo existente.

3. Personalizar los Tabs

Vamos a trabajar con el `_layout.tsx` de la carpeta `tabs`.

[Expo Router native tabs](#)

Option	Description	Example
tabBarActiveTintColor	The color of the active tab's icon and label.	tabBarActiveTintColor: '#007AFF'
tabBarInactiveTintColor	The color of inactive tabs' icons and labels.	tabBarInactiveTintColor: '#A9A9A9'
tabBarBadge	A badge to display on a tab bar item.	tabBarBadge: '🔥'
tabBarBadgeStyle	Custom styles for the badge.	tabBarBadgeStyle: { backgroundColor: 'red', color: 'white' }
tabBarIcon	A function that returns a React element for the icon.	tabBarIcon: ({ focused, color, size }) => <Ionicons name={focused ? 'home' : 'home-outline'} size={size} color={color} />
tabBarLabelStyle	Styles for the tab bar labels.	tabBarLabelStyle: { fontSize: 12, fontWeight: 'bold' }
tabBarBackground	A custom background component for the tab bar.	tabBarBackground: () => <View style={{ backgroundColor: '#EOE0EO', flex: 1 }} />
tabBarIndicatorStyle (Android)	Styles for the tab indicator (Android).	tabBarIndicatorStyle: { backgroundColor: 'orange' }
tabBarActiveBackgroundColor	The background color for the active tab (Android & iOS).	tabBarActiveBackgroundColor: '#f0f0f0'
icon	A component or image source to use as the tab's icon.	icon: ({ focused }) => <Image source={focused ? require('./active-icon.png') : require('./inactive-icon.png')} />
iconColor	The color of the icon on Android and iOS.	iconColor: 'blue'

```
<Tabs screenOptions={{
  tabBarActiveTintColor: 'indigo',
  headerShown: false,
  // tabBarShowLabel: false,
  // tabBarStyle: {
  //   backgroundColor: 'black',
  // },
  // tabBarInactiveBackgroundColor: 'red',
}}>
```

4. Drawer - Menú lateral

Vamos a proceder en primer lugar con la instalación de [navigation drawer](#), algo muy común en aplicaciones móviles. Esto no viene por defecto en la default template de React Native y hay que instalarlo.

Para la instalación tenemos que tener en cuenta el SDK de expo de nuestra aplicación, que esto podemos verlo en el fichero [package.json](#).

```
$ npx expo install @react-navigation/drawer react-native-reanimated react-native-worklets
```

Tened en cuenta que el [Drawer](#) es algo que se encuentra a nivel global de la aplicación con el icono de "hamburguesa" en la esquina superior izquierda o derecha que al ser pulsado mostrará el menú lateral. Para una prueba y ver como se muestra vamos a hacer una implementación en nuestro RootLayout ([app/_layout.tsx](#)).

```
import { Drawer } from 'expo-router/drawer';
...
return (
  <Drawer>
    <Slot />
  </Drawer>
)
...
...
```

Visualizamos en el simulador para ver como se muestra. Borramos el drawer de este layout y vamos a crear el nuestro propio

Ahora vamos a realizar la implementación correcta en la que crearemos un layout específico para nuestro Drawer.

Crearemos una carpeta llamada `app/drawer` con un fichero `_layout.tsx` y crearemos un `rnfe` llamado `DrawerLayout`.

En este Layout, basados en la documentación de Drawer, vamos a implementar el ejemplo que viene adaptado a nuestra aplicación.

Creamos dentro de la carpeta `drawer` los siguientes ficheros:

1. `schedule/index.tsx`

```
import React from 'react'
import { Text, View } from 'react-native'

const ScheduleScreen = () => {
  return (
    <View>
      <Text>ScheduleScreen</Text>
    </View>
  )
}

export default ScheduleScreen
```

2. `user/index.tsx`

```
import React from 'react'
import { Text, View } from 'react-native'

const UserScreen = () => {
  return (
    <View>
      <Text>UserScreen</Text>
    </View>
  )
}

export default UserScreen
```

3. Implementación del Drawer basado en la [documentación](#). Copiamos el ejemplo extendido y lo adaptamos a las carpetas creadas dentro de (`/app/drawer`)

```

import { Drawer } from 'expo-router/drawer'
import React from 'react'

const DrawerLayout = () => {
  return (
    <Drawer>
      <Drawer.Screen
        name="user/index"
        options={{
          drawerLabel: 'User',
          title: 'User',
        }}
      />
      <Drawer.Screen
        name="schedule/index"
        options={{
          drawerLabel: 'Schedule',
          title: 'Schedule',
        }}
      />
    </Drawer>
  )
}

export default DrawerLayout

```

4. Cambio en la estructura de directorios. Es momento de reestructurar todos los directorios y finalmente incluiremos todo dentro de la carpeta `drawer`. Es decir, movemos la carpeta `tabs` dentro de `drawer`, junto con las carpetas que ya contiene `drawer`.

Veamos como queda en el simulador y comprobemos que se carga la información correctamente. En caso de que no cargue de forma correcta, cerrad los procesos del emulador y los relanzamos de nuevo con `npm run start -c` para borrar cachés.

5. Drawer - Personalización

Vamos a realizar ahora unos cambios visuales tanto para el Drawer Container (**Drawer**) como para los Drawer Items (**Drawer.Screen**).

```
<Drawer screenOptions={{
    // color con transparencia para ver lo que hay detrás del drawer
    overlayColor: 'rgba(0,0,0,0.4)',
    // color del elemento seleccionado en el drawer
    drawerActiveTintColor: 'indigo',
    // elimina la sombra debajo del header
    headerShadowVisible: false,
    sceneStyle: {
        // elimina la linea que separa header del resto de la escena
        backgroundColor: 'white'
    }
}}>
    ...
</Drawer>
```

Especificación de los screenOptions más usados en el componente **Drawer**.

Option	Description
drawerStyle	Estilo para el panel del menú del drawer (por ejemplo ancho, color de fondo)
drawerType	Tipo de drawer (por ejemplo 'front', 'back', 'slide', 'permanent')
drawerPosition	'left' o 'right'
swipeEnabled	Booleano para permitir abrir/cerrar mediante gesto de "swipe"
swipeEdgeWidth, swipeMinDistance	Ajustes del gesto de apertura
drawerActiveTintColor, drawerInactiveTintColor	Colores para íconos/etiquetas activos/inactivos del menú
drawerActiveBackgroundColor, drawerInactiveBackgroundColor	Fondo para elementos activos/inactivos
lazy	Indica si las pantallas del drawer deberían renderizarse sólo cuando se accede la primera vez
headerShown	Aunque es común en Stack Navigator, también se puede usar para esconder/mostrar el header en cada pantalla del drawer

Los elementos internos del Drawer `Drawer.Screen` también se pueden personalizar. La personalización se hace a través del atributo `options`.

En el modelo por defecto, ya vienen dos atributos de `options` inicializados como el `drawerLabel` y el `title`. Además de estos dos, podemos personalizarlos con muchas opciones como puede ser un ícono. Y el equinó si es a través de una fuente, poder personalizar también su tamaño (`size`) y color (`color`).

En el ejemplo propuesto a continuación siguiendo la estructura de nuestra aplicación y la fuente de iconos introducida de `Ionicons`, hemos procedido a crear un componente para incluir el ícono como elemento ilustrativo de las opciones del Drawer.

Como nuestro `drawerIcon` va a ser un componente, éste se inicializa como una función `() => ()`.

```
...
<Drawer.Screen
  name="user/index"
  options={{
    drawerLabel: 'User',
    title: 'User',
    drawerIcon: () => (
      <Ionicons name='person-add-outline' />
    )
  }}
/>
...
```

Ahora ilustramos el ejemplo anterior con desestructuración de las propiedades del componente `Ionicons` para adaptar el ícono y el tamaño al de la fuente de los textos.

```
...
<Drawer.Screen
  name="user/index"
  options={{
    drawerLabel: 'User',
    title: 'User',
    drawerIcon: ({ color, size }) => (
      <Ionicons name='person-add-outline' color={color} size={size} />
    )
  }}
/>
...
```

Adáptadlo a los dos `Drawer.Screen` que tenemos en el Drawer.

Como en el caso anterior del atributo `screenOptions` de `Drawer`, os dejo también una tabla de las `options` de `Drawer.Screen`.

Option	Description
drawerStyle	Estilo para el panel del menú del drawer (por ejemplo ancho, color de fondo)
drawerType	Tipo de drawer (por ejemplo 'front', 'back', 'slide', 'permanent')
drawerPosition	'left' o 'right'
swipeEnabled	Booleano para permitir abrir/cerrar mediante gesto de "swipe"
swipeEdgeWidth, swipeMinDistance	Ajustes del gesto de apertura
drawerActiveTintColor, drawerInactiveTintColor	Colores para íconos/etiquetas activos/inactivos del menú
drawerActiveBackgroundColor, drawerInactiveBackgroundColor	Fondo para elementos activos/inactivos
lazy	Indica si las pantallas del drawer deberían renderizarse sólo cuando se accede la primera vez
headerShown	Aunque es común en Stack Navigator, también se puede usar para esconder/mostrar el header en cada pantalla del drawer

Sirva también la combinación de teclas **ctrl + space**, se pueden desplegar las opciones de personalización de los elementos drawer y las screenoptions y podéis ir jugando con las diferentes opciones disponibles.

6. DrawerContent

En esta sección vamos a trabajar con un poco más de profundidad el contenido y la personalización del Drawer.

Una de las propiedades más importantes de Drawer es **drawerContent**. Esta propiedad viene a "sustituir" el contenido interno del **Drawer**, es decir el **Drawer.Screen**.

Esta opción permite poder utilizar componentes personalizados para las opciones del componente. Por ejemplo si inicializamos esta propiedad y metemos un Text:

```
<Drawer
  drawerContent={() => <Text>Opción</Text>}
  screenOptions={{
    ...
  }}
```

Se observará en el simulador que en el Drawer solo se renderiza ese componente ignorando los **Drawer.Screen** existentes.

Vamos ahora a crear en nuestra carpeta (`components/shared/`) y creamos un nuevo fichero llamado `CustomDrawer.tsx` y lo inicializamos mediante `rnfe`.

```
import React from 'react'
import { Text, View } from 'react-native'

const CustomDrawer = () => {
  return (
    <View>
      <Text>CustomDrawer</Text>
    </View>
  )
}

export default CustomDrawer
```

Ahora en la configuración del Drawer (`app/drawer/_layout.tsx`) donde habíamos creado el Text, realizamos la llamada al componente de la siguiente forma.

```
<Drawer
  drawerContent={CustomDrawer}
  screenOptions={{
  ...
}}
```

Esto habilitará que todo el contenido de nuestro Drawer, se renderice como opciones del Drawer.

Vamos ahora a desarrollar las distintas opciones que presentará nuestro Drawer desde el propio componente `CustomDrawer.tsx`.

En primer lugar cambiaremos el `View` por `DrawerContentScrollView` para que permita hacer scroll para el caso de menú largo.

```
const CustomDrawer = () => {
  return (
    <DrawerContentScrollView>
    ...
    </DrawerContentScrollView>
  )
}
```

Podéis probarlo duplicando varias veces el componente `Text` dentro del `DrawerContentScrollView` para comprobar que realiza el scroll.

Para colocar los elementos dentro del contenido del Drawer, vamos a crear un view y las clases para distribuir los elementos en él. Vamos a crear un primer elemento para reflejar un cuando con imagen corporativa, por ejemplo:

```
<DrawerContentScrollView>
  <View className='flex justify-center items-center mx-3 p-10 mb-10 h-[150px]
rounded-xl bg-primary'>
    <View className='flex justify-center items-center bg-white rounded-full h-
24 w-24'>
      <Text className='text-primary font-work-black text-3xl'>FH</Text>
    </View>
  </View>
</DrawerContentScrollView>
```



Siguiendo ahora con la implementación el Drawer, vamos a crear las opciones que va a tener. Pero antes de empezar, tenemos que tener claro que nuestro `DrawerLayout` (`app/drawer/_layout.tsx`) en la especificación de la propiedad `drawerContent` es donde llamamos a nuestro `CustomDrawer` y que implícitamente estamos pasando las `props` de `DrawerContentComponentProps`.

```
const Drawer = (props) => {
  return (
    <DrawerLayout
      drawerContent={props.drawerContent}
      screenOptions={props.screenOptions}
    >
      {props.children}
    </DrawerLayout>
  );
}

Drawer.propTypes = {
  drawerContent: PropTypes.func,
  screenOptions: PropTypes.object,
};
```

Por esto, en nuestro `CustomDrawer`, vamos a indicar que las `props` son del tipo `DrawerContentComponentProps`.

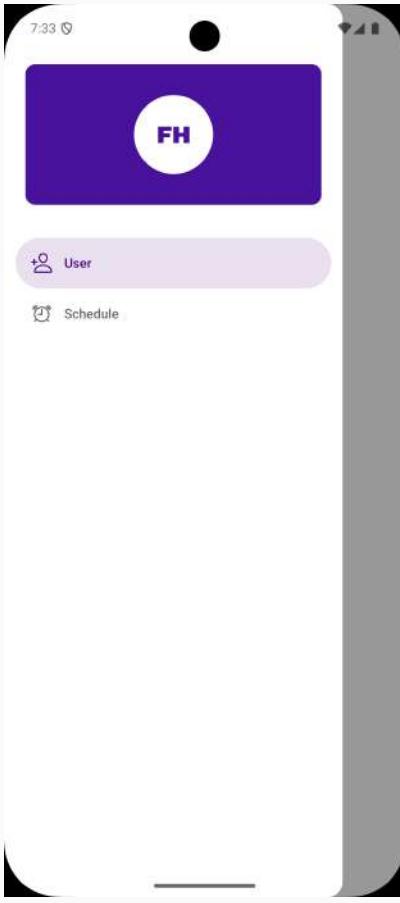
```
const CustomDrawer = (props: DrawerContentComponentProps) => {
  return (
    <DrawerContentScrollView {...props}>
    ...
  )
}
```

Esto nos va a permitir empezar a trabajar con `<DrawerItemList>` con las props esparcidas que tenemos importar los items que ya tenemos definidos en nuestro `DrawerLayout`. Con lo que nuestro `CustomDrawer` quedaría así:

```
<DrawerContentScrollView {...props}>
  <View className='flex justify-center items-center mx-3 p-10 mb-10 h-[150px]
rounded-xl bg-primary'>
  <View className='flex justify-center items-center bg-white rounded-full h-
24 w-24'>
    <Text className='text-primary font-work-black text-3xl'>FH</Text>
  </View>
</View>

 {/* DrawerItems */}
 <DrawerItemList {...props} />
</DrawerContentScrollView>
```

Relanzad la aplicación y comprobad que todo se muestra como en la imagen.



Con esto ya implementado y finalizado, ¿qué puedo hacer yo si lo que quiero es que mi home, la que se mostraba a través del Stack sea la primera que se muestre? Puesto es lo que vamos a ver en la siguiente sección.

En todo caso dejo una copia de mi proyecto para que podías recuperar si algo falla. (*Aules: navigation-app-drawer.zip*).

7. Conectar Drawer con Tabs y con Stack

Esta sección no es complicada pero puede llevar a confusiones, así que por si hay algún problema, lo mejor es hacerse un commit del proyecto en este punto antes de empezar o realizar una copia del proyecto.

Inicialmente nuestra carpeta `app/drawer` tiene las dos carpetas `schedule` y `user`. Y fuera nuestra carpeta `tabs` ya con estructura configurada. Para asociar todo y que pueda ser gestionado a través del `drawer`, **vamos a mover la carpeta `tabs` dentro de `drawer`**.

Al hacer esto y refrescar la aplicación en el emulador, veremos que dentro del `Drawer` nos aparecen los `tabs` y es porque usa el filesystem para construir las opciones del menú.

Ahora vamos al DrawerLayout para montar las opciones a nuestro gusto e inicialmente solo crearemos una nueva opción que incluya el link a los tabs.

```
<Drawer.Screen
  name="tabs"
  options={{
    drawerLabel: 'Tabs + Stack',
    title: 'Tabs + Stack',
    drawerIcon: ({ color, size }) => (
      <Ionicons name='albums-outline' color={color} size={size} />
    )
  }}
/>
```

Si refrescamos la aplicación, veremos que ya muestra la opción y carga el Stack con Tabs.

Ahora cuando intentamos navegar por los botones los enlaces están rotos. Esto se debe al cambio de filesystem, cuando hemos movido la carpeta. De nuevo, en el error, podemos visitar el Sitemap actual para poder arreglar los enlaces rotos.

En el fichero ([app/tabs/\(stack\)/home/index.tsx](#)) donde está la configuración de los botones actualizaremos las rutas de los botones y links.

```
<CustomButton className='mb-2' color='primary'
  onPress={() => router.push('/drawer/tabs/products')}
>
  Products
</CustomButton>
```

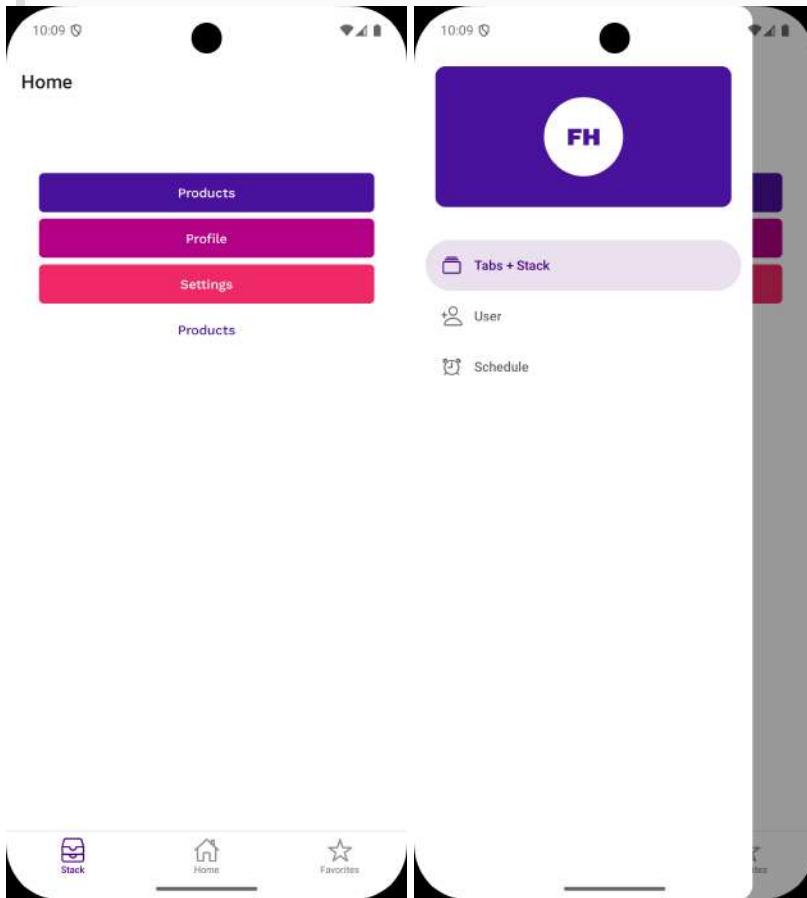
En la vista de products ([app/tabs/\(stack\)/products/index.tsx](#)) también tenemos un enlace a la vista concreta del producto y esta también tenemos que actualizarla.

```
...
<Link href={`/drawer/tabs/products/${item.id}`} className='text-primary'>
  ...
</Link>
```

Se puede observar ahora que en toda la navegabilidad, tenemos dos cabeceras. En nuestro caso solo queremos mantener las cabeceras de las distintas vistas y ocultar la cabecera del **Drawer**, por lo que en las **screenOptions** de **DrawerLayout** ([app/drawer/_layout.tsx](#)), vamos a ocultar la cabecera del **Drawer**.

```
<Drawer
    drawerContent={CustomDrawer}
    screenOptions={{{
        headerShown: false,
        overlayColor: 'rgba(0,0,0,0.4)',
        drawerActiveTintColor: 'indigo',
        headerShadowVisible: false,
        sceneStyle: {
            backgroundColor: 'white'
        }
    }}}>
```

No se observa el menú hamburguesa pero con un simple deslizamiento de izquierda a derecha aparecerá nuestro Drawer.



8. Optimización de nombres de directorios

Vamos a simplificar un poco los nombres de los directorios para dejar una estructura más clara e intuitiva. Lo primero que vamos a hacer es descartar aquellos nombres de directorio que no queremos que formen parte de la ruta en la url. En nuestro caso ya tenemos (**stack**), pero esto vamos a extenderlo a (**tabs**) y (**drawer**).

De nuevo este cambio va a provocar algunas roturas de enlaces, pero si nuestro IDE, es sabio, que lo es (VSCode/WebStorm) nos indicará qué ficheros tienen enlaces rotos.

1. (app/index.tsx)

```
return <Redirect href="/home" />
```

2. (app/(drawer)/_layout.tsx)

```
<Drawer.Screen
  name="(tabs)"
  options={{
    drawerLabel: 'Tabs + Stack',
    title: 'Tabs + Stack',
    drawerIcon: ({ color, size }) => (
      <Ionicons name='albums-outline' color={color} size={size} />
    )
  }}
/>
```

Hemos puesto entre paréntesis tabs => (**tabs**). Dependiendo de la versión de expo-router, puede que esto ya no de error porque automáticamente lo interprete.

3. (app/(drawer)/(tabs)/(stack)/home/index.tsx)

```
<CustomButton className='mb-2' color='primary'
  onPress={() => router.push('/products')}>
  Products
</CustomButton>
```

Actualizamos todos los botones/links de esta pantalla. Recordad que los enlaces se pueden simplificar excluyendo todos aquellos path que van entre paréntesis, ya que no se van a mostrar y de forma automática a nivel de filesystem, expo-router tambien lo entiende así.

4. (app/(drawer)/(tabs)/(stack)/products/index.tsx)

```
<View className='flex flex-row justify-between mt-2'>
  <Text className='font-work-black'>{item.price}</Text>
  <Link href={`/products/${item.id}`} className='text-primary'>
    Ver detalles
  </Link>
</View>
```

Actualizamos la ruta que accede al elemento único de producto, en la que simplemente podemos dejarlo con el enlace `/products/{id}`. Obviando nuevamente todos los path que en el filesystem están entre paréntesis.

Para finalizar, solo quedaría que las pantallas que no muestran header, que muestren de forma adecuada su header con el nombre de la pantalla o sección que estamos mostrando. Esta implementación la realizaremos en las dos siguientes secciones.

9. Mostrar Drawer de forma controlada

En la primera parte de esta sección vamos a ver de qué forma podríamos abrir el Drawer de forma controlada. Es decir, crear una acción o un botón que nos permita abrir el Drawer desde una posición que no es la habitual.

La idea de contemplar esta implementación es porque a veces en una aplicación puede existir más de un Drawer, entonces vamos a ver qué formas tenemos de abrir y cerrar drawers de forma controlada.

En la pantalla de HomeScreen (app/(drawer)/(tabs)/(stack)/home/index.tsx) crearemos una función `onToggleDrawer` para manejar la apertura o cierre del Drawer y una variable `navigation` del tipo `useNavigation` de `expo-router` que nos dará acceso al Drawer.

A través de `navigation` podemos acceder a un montón de propiedades para manejar las acciones de adelante y atrás en una pantalla o página, reiniciar la navegación, opciones de la página como el título, abrir el drawer y una infinidad más de opciones.

```
const onToogleDrawer = () => {
  navigation.
    Identifier addListener (method) addListener<EventName>(type: EventN...
  return (
    <SafeAreaVi ...
      <View c ...
        <Cu ...
          <Cu ...
            <Cu ...
              <Cu ...
                <Cu ...
                  <Cu ...
                    <Cu ...
                      <Cu ...
                        <Cu ...
                          <Cu ...
                            <Cu ...
                              <Cu ...
                                <Cu ...
                                  <Cu ...
                                    <Cu ...
                                      <Cu ...
                                        <Cu ...
                                          <Cu ...
                                            <Cu ...
                                              <Cu ...
                                                <Cu ...
                                                  <Cu ...
                                                    <Cu ...
                                                      <Cu ...
                                                        <Cu ...
                                                          <Cu ...
                                                            <Cu ...
                                                              <Cu ...
                                                                <Cu ...
                                                                  <Cu ...
                                                                    <Cu ...
                                                                      <Cu ...
                                                                        <Cu ...
                                                                          <Cu ...
                                                                            <Cu ...
                                                                              <Cu ...
                                                                                <Cu ...
                                                                                  <Cu ...
                                                                                    <Cu ...
                                                                                      <Cu ...
                                                                                      <Cu ...
                                                                                      <Cu ...
                                                                                      <Cu ...
                                                                                      <Cu ...
                                                                                      <Cu ...
                                                                                      <Cu ...
                                                                                      <Cu ...
                                                                                      <Cu ...
                                                                                      <Cu ...
                                                                                      <Cu ...
                                                                                      <Cu ...
................................................................
```

Centrándonos en la propiedad para abrir y cerrar el drawer, quedaría tal que así:

```
const navigation = useNavigation();

const onToogleDrawer = () => {
  navigation.dispatch(DrawerActions.toggleDrawer)
}
```

En esta misma vista, vamos a crear un nuevo CustomButton a continuación de los que ya hay y realizar la llamada a la función que acabamos de crear.

```
<CustomButton onPress={onToogleDrawer}>Open menu</CustomButton>
```

Lanzad los cambios sobre el emulador y comprobad la funcionalidad implementada.

Seguimos con las mejoras y vamos ahora a la opción del usuario del menú y es que cuando estamos en ella no tenemos forma de regresar. Si recordáis en esta parte habíamos en el ([app/drawer/_layout.tsx](#)) habíamos dejado de mostrar el header, pero de forma global a todo lo que está dentro de ([\(drawer\)](#)).

Tenemos una forma de personalizar en que Screen asociada al Drawer poder mostrar el header o no. Es decir, en la propiedad [options](#) de [Drawer.Screen](#) podemos marcar el [headerShown: false](#). Así podemos decidir donde mostrarlo y donde no.

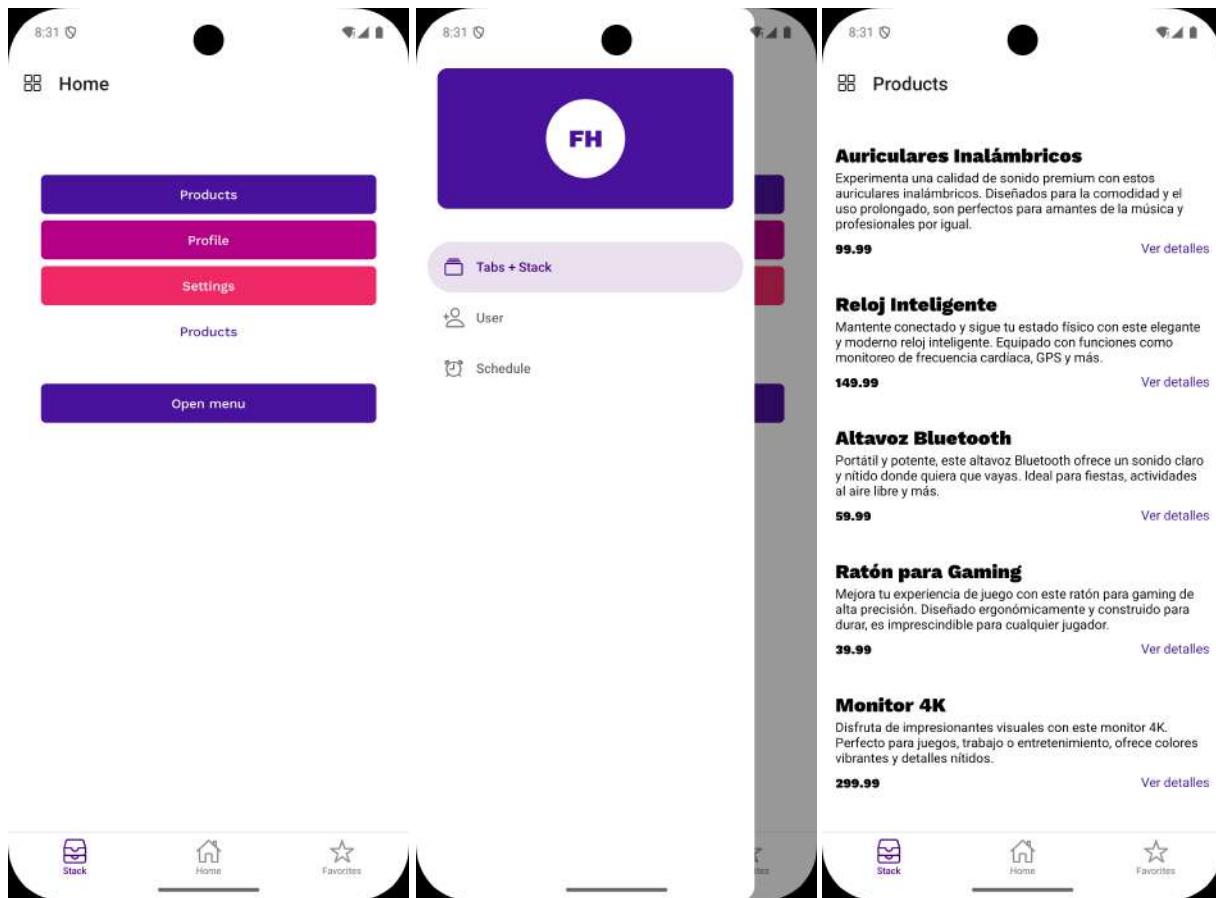
```
<Drawer
    drawerContent={CustomDrawer}
    screenOptions={{
        overlayColor: 'rgba(0,0,0,0.4)',
        drawerActiveTintColor: 'indigo',
        headerShadowVisible: false,
        sceneStyle: {
            backgroundColor: 'white'
        }
    }}>
<Drawer.Screen
    name="(tabs)"
    options={{
        headerShown: false,
        drawerLabel: 'Tabs + Stack',
        title: 'Tabs + Stack',
        drawerIcon: ({ color, size }) => (
            <Ionicons name='albums-outline' color={color} size={size} />
        )
    }}
/>
<Drawer.Screen
    name="user/index"
    options={{
        drawerLabel: 'User',
        title: 'User',
        drawerIcon: ({ color, size }) => (
            <Ionicons name='person-add-outline' color={color} size={size} />
        )
    }}
/>
<Drawer.Screen
    name="schedule/index"
    options={{
        drawerLabel: 'Schedule',
        title: 'Schedule',
        drawerIcon: ({ color, size }) => (
            <Ionicons name='alarm-outline' color={color} size={size} />
        )
    }}
/>
</Drawer>
```

Ahora vamos al layout de `(app/(drawer)/(tabs)/(stack)/_layout.tsx)`. En este el menú de hamburguesa no se muestra para ninguna de las páginas del Stack. En este layout tenemos comentada la línea de `headerShown: false`, por lo que no se mostrará el menu. Este header está comentado, por que lo que queremos es evitar el Header del Drawer y el Header del Screen, pero Stack, tiene una propiedad que nos permite insertar un componente en el Header de los componentes que residen dentro de él. Vamos a trabajar con la propiedad `headerLeft` que puede albergar una función flecha y mostrar un componente. En nuestro caso un componente IIcono que al clickar mostrará el Drawer. Como en el caso anterior creamos una variable `navigation` y una función que llamaremos `onHeaderLeftClick`.

```
<Stack screenOptions={{
    //headerShown: false,
    headerShadowVisible: false,
    contentStyle: {
        backgroundColor: 'white'
    },
    headerLeft: ({ tintColor, canGoBack }) => (
        <Ionicons
            name='grid-outline'
            // name='menu-outline'
            className='mr-5'
            size={20}
            onPress={onHeaderLeftClick}
        />
    ),
}}>
```

Lo comprobamos en el simulador.

¿Porqué desestructuramos `canGoBack`? Pues la intención es que al navegar entre las distintas páginas del Stack, poder volver atrás, porque ahora en todas ellas siempre muestra el icono del menú y es la única forma de navegar que tendríamos, volver siempre a la Home del Stack.



Para poder cambiar el icono que se muestra lo haremos teniendo en cuenta el booleano `canGoBack` y condicionaremos el icono que se muestra.

```
name={canGoBack ? 'arrow-back-outline' : 'grid-outline'}
```

Ahora solo queda controlar, que la acción al pulsar el icono de vuelta atrás, vuelva atrás y no despliegue de nuevo el Drawer. Lo primero es pasar como parámetro `canGoBack` a la función `onHeaderLeftClick` y si es true, pues volver atrás sino desplegar el Drawer. Para volver atrás, al tener un `Stack` con diferentes enrutados, necesitaremos acceder al `router` que es una constante de `expo-router` que tiene acceso al historial de navegación y permite volver atrás en la pila de transiciones entre pantallas.

```

const onHeaderLeftClick = (canGoBack: boolean) => {
  if (canGoBack) {
    router.back()
    return;
  }

  navigation.dispatch(DrawerActions.toggleDrawer)
}

...
<Ionicons
  name={canGoBack ? 'arrow-back-outline' : 'grid-outline'}
  className='mr-5'
  size={20}
  onPress={() => onHeaderLeftClick(canGoBack)}
/>

```

Probad en el emulador que todo funciona correctamente y que al acceder a una pantalla del Stack el botón de menú cambia por el botón de vuelta atrás y que la acción que realiza es volver a la pantalla anterior y no abrir el Drawer hasta llegar al top de la navegación, es decir hasta llegar al a HomeScreen, no debería cambiar el icono de vuelta atrás por el del menú.

Para finalizar, como ya sabemos manejar en qué pantallas queremos mostrar el header y en cuales no, en el fichero de `(app/(drawer)/(tabs)/_layout.tsx)`. Quitamos el `headerShown: false` global en `<Tabs>` y ocultaremos el header en el `<Tabs.Screen>` del `(stack)` y así se mostrará para el resto de `<Tabs.Screen>`.

10. Cambiar título basado en los argumentos

A veces es importante, poder personalizar los nombres de las pantallas que aparecen en el header. Por ejemplo, en nuestra aplicación cuando accedemos a la pantalla de descripción del producto vemos que muestra `products/[id]` y sería más descriptivo que al menos se muestre el título del producto. Esto es parametrizable y vamos a ver cómo podemos cambiarlo.

En el fichero `(app/(drawer)/(tabs)/(stack)/products/[id].tsx)` ya tenemos acceso al objeto producto que se representa en la pantalla, por lo tanto podemos acceder a su título.

Con acceso al `navigation` y con un `useEffect` que variará dependiendo del producto que se muestra en la pantalla, podemos cambiar el display del título en el header.

```
const navigation = useNavigation();
...
useEffect(() => {
  navigation.setOptions({
    title: product?.title ?? 'Producto'
  })
}, [product])
```



Auriculares Inalámbricos

Experimenta una calidad de sonido premium con estos auriculares inalámbricos. Diseñados para la comodidad y el uso prolongado, son perfectos para amantes de la música y profesionales por igual.

99.99

Auriculares Inalámbricos

Experimenta una calidad de sonido premium con estos auriculares inalámbricos. Diseñados para la comodidad y el uso prolongado, son perfectos para amantes de la música y profesionales por igual.

99.99



11. Conclusiones

Resumiendo todas las partes vistas en este tema, alcanzamos todos los ejemplos más comunes en cuanto a navegación. Desde el punto de vista de una aplicación tenemos la vista de tabs, una ejemplo de menú que se muestra en determinadas pantallas. Según la pantalla podemos habilitar navegación o deshabilitarla e incluso cambiar los iconos en función de si queremos navegar o abrir el menú.

También y muy importante, la parametrización y condicionado tanto para abrir Drawer como para cambiar títulos del header de cada una de las pantallas.

Se ha intentado alcanzar la mayoría de los ejemplos que se pueden dar en el día a día del desarrollo de aplicaciones. Cualquier variante que surga a patir de aquí, aplicando un poco de lógica y con el punto de partida de esta aplicación de navegación, todo es posible implementarse.

12. Bibliografía

JavaScript tabs. (2025, 31 octubre). Expo Documentation. <https://docs.expo.dev/router/advanced/tabs/>

Expo Vector Icons. (2025, 8 agosto). Expo Documentation. <https://docs.expo.dev/guides/icons/>

*Ionic.** (s. f.). Ionicons: The premium icon pack for Ionic Framework. <https://ionic.io/ionicons>

Router native tabs. (s. f.). Expo Documentation. <https://docs.expo.dev/versions/latest/sdk/router-native-tabs/>

Drawer. (2025, 5 noviembre). Expo Documentation. <https://docs.expo.dev/router/advanced/drawer/>