TECNOLÓGICO NACIONAL DE MEXICO

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

Ingeniería en Sistemas Computacionales.

Tarea 10_ Unidad 4 Navegacion y Arquitectura de la App

Asignatura: Programación Nativa para Moviles

Docente: Jorge Peralta Escobar

Hora: 14:00 – 15:00 pm

Integrantes:

Del Angel Del Angel Erika Yaneth #20071926

Villaseñor Grimaldo Alejandro #19071548

Semestre: Enero – Junio 2025.

DEJANDO HUELLA

GITHUB -Workspace/Programacion_movil at master · Alejandro19071548/Workspace

RUTA DE APRENDIZAJE 1 – COMPONENTES DE LA ARQUITECTURA

VIDEO 1 – RUTA DE APRENDIZAJE 1

La Unidad 4 se centra en el desarrollo de interfaces de usuario más complejas y dinámicas, aplicando los principios de Material Design y aprovechando las capacidades avanzadas de Jetpack Compose.

Temas clave:

- Navegación entre pantallas.
- Gestión del estado de la aplicación.
- Personalización de componentes UI.
- Buenas prácticas para mantener una arquitectura limpia y escalable.

El instructor presenta estos objetivos como fundamentales para construir apps modernas, estructuradas y fáciles de mantener.

VIDEO 2 – RUTA DE APRENDIZAJE 1

En esta parte del curso se presenta el flujo de datos unidireccional (UDF), un patrón en el que los datos fluyen desde capas inferiores hacia la interfaz de usuario, mientras que los eventos del usuario se propagan hacia abajo para actualizar el estado.
Este enfoque mejora la gestión del estado, mantiene el código organizado y reduce errores.

Conceptos clave:

- Uso de un estado de UI inmutable que actúa como una "foto" del estado actual.
- Recomendación de emplear ViewModel para manejar la lógica de presentación.
- Exposición del estado mediante tipos observables como StateFlow o LiveData, lo que permite a la UI reaccionar de forma eficiente a los cambios.

También se recalca la importancia de la separación de responsabilidades:

- La lógica de negocio debe estar en capas como dominio o datos.
- La lógica de UI (como navegación o mensajes visuales) debe residir en la capa de presentación.

CUESTIONARIO RUTA 1

1. ¿Qué método se llama primero cuando la app ya no tiene enfoque?

- (●) onPause() ¡Correcto!
- ( ) onStart()
- ( ) onCreate()
- ( ) onStop()

2. Después de ___, la app deja de estar visible en la pantalla.

- ( ) onPause()
- ( ) onStart()
- ( ) onCreate()
- (●) onStop() ¡Correcto!

3. Usa ___ para escribir un mensaje de depuración. Este método tiene dos argumentos: la etiqueta de registro y el mensaje de registro.

- ( ) `Log.i()`
- (●) `Log.d()` ¡Correcto!

4. Para guardar un valor que necesita sobrevivir a un cambio de configuración, se deben declarar sus variables con ___.

- ( ) `MutableState{}`
- (●) `rememberSaveable{}` ¡Correcto!
- ( ) `remember{}`
- ( ) Elevación de estado

5. El principio de separación de problemas indica que la app debe dividirse en clases, cada una con responsabilidades independientes.

- (●) Verdadero ¡Correcto!
- ( ) Falso

6. La IU es lo que ve el usuario y el estado de la IU es lo que la app dice que debería ver.

- (●) Verdadero ¡Correcto!
- ( ) Falso

7. Según la arquitectura de app recomendada, cada app debe tener al menos las siguientes dos capas:

- ○ Capa de dominio y de datos
- ◉ Capa de la IU y de datos ¡Correcto!
- ○ Capa del repositorio y de la IU
- ○ Capa del dominio y de la IU

8. `StateFlow` es un flujo observable contenedor de datos que emite actualizaciones de estado actuales y nuevas.

- ◉ Verdadero ¡Correcto!
- ○ Falso

9. ¿Cuál de las siguientes configuraciones se debe agregar al archivo `build.gradle` a fin de incorporar dependencias para el código fuente de la prueba de unidades?

- ○ `implementation`
- ◉ `testImplementation` ¡Correcto!

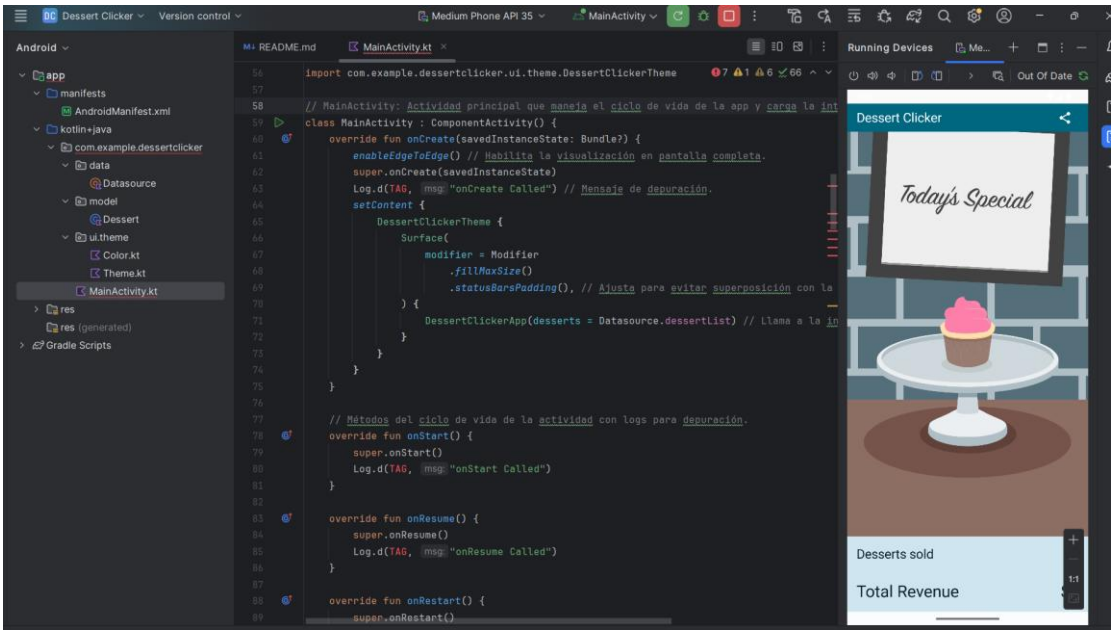
10. Las pruebas de unidades se ejecutan en un emulador o dispositivo Android.

- ○ Verdadero
- ◉ Falso ¡Correcto!

Results

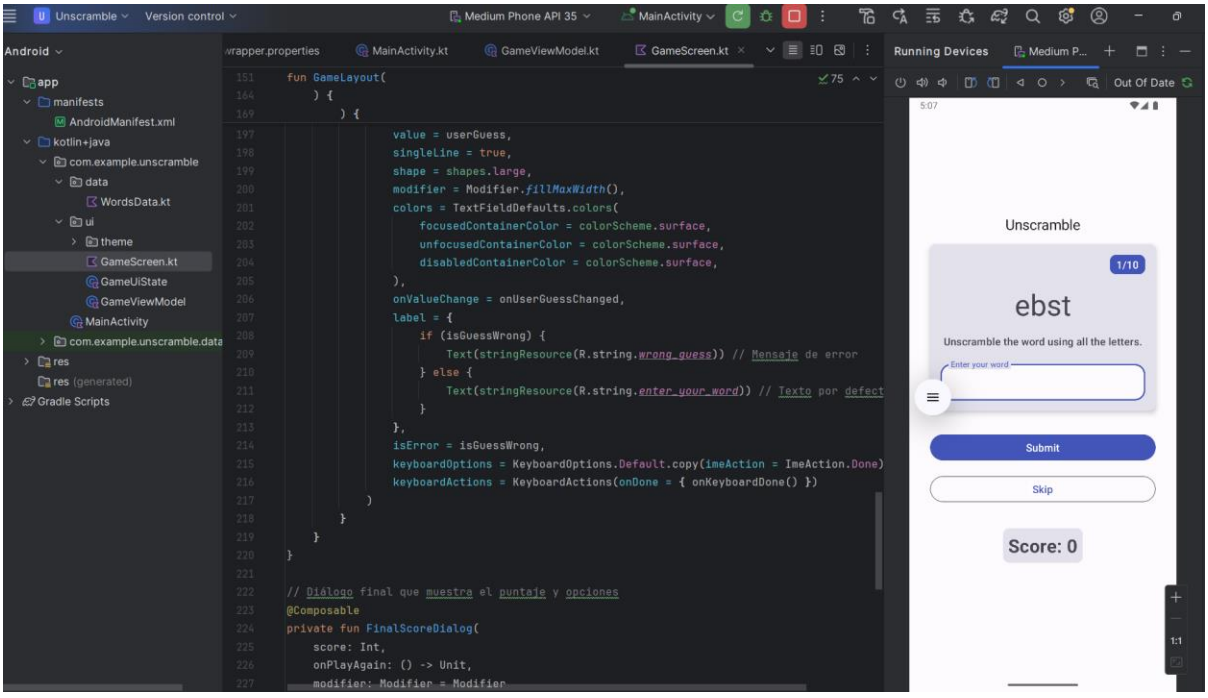
Tu puntuación es 10 de 10. ¡Felicitaciones! Aprobaste el cuestionario.

DESSER CLICKER



Esta aplicación llamada "Dessert Clicker" simula la venta de postres. Cada vez que el usuario toca la imagen de un postre, se contabiliza una venta y se incrementa el ingreso total. A medida que se venden más postres, se muestran postres más caros automáticamente. La app también permite compartir la cantidad de postres vendidos e ingresos generados. Está diseñada siguiendo los principios de Jetpack Compose y el ciclo de vida de una actividad en Android.

UNSCRAMBLE GAME



Este archivo define la pantalla principal del juego "Unscramble" en Jetpack Compose, que incluye:

- Un ViewModel que maneja la lógica del juego.
- Un campo de texto para ingresar la palabra adivinada.
- Botones para enviar o saltar la palabra.
- Una tarjeta con el puntaje actual.
- Un diálogo que aparece cuando el juego termina con el puntaje y opciones para salir o volver a jugar.

Todo está construido de forma reactiva y componible, siguiendo buenas prácticas de diseño moderno con Jetpack Compose.

RUTA DE APRENDIZAJE 2 – NAVEGACION EN JETPACK COMPOSE

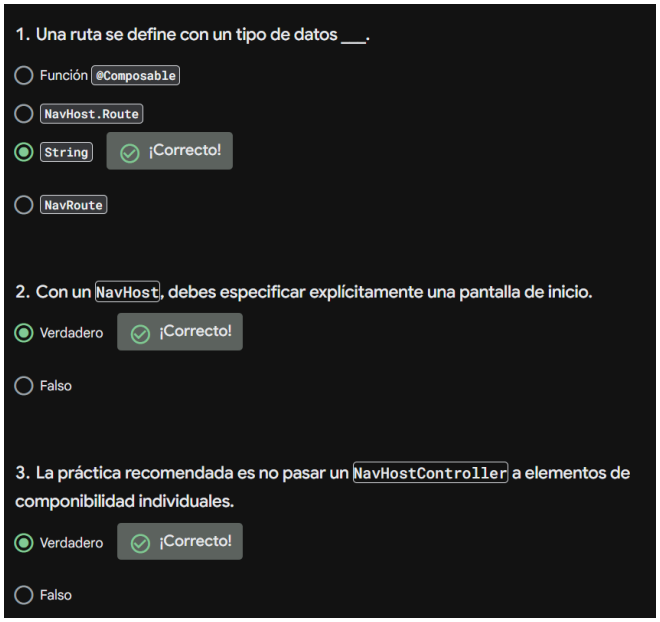
VIDEO 1 RUTA DE APRENDIZAJE 2

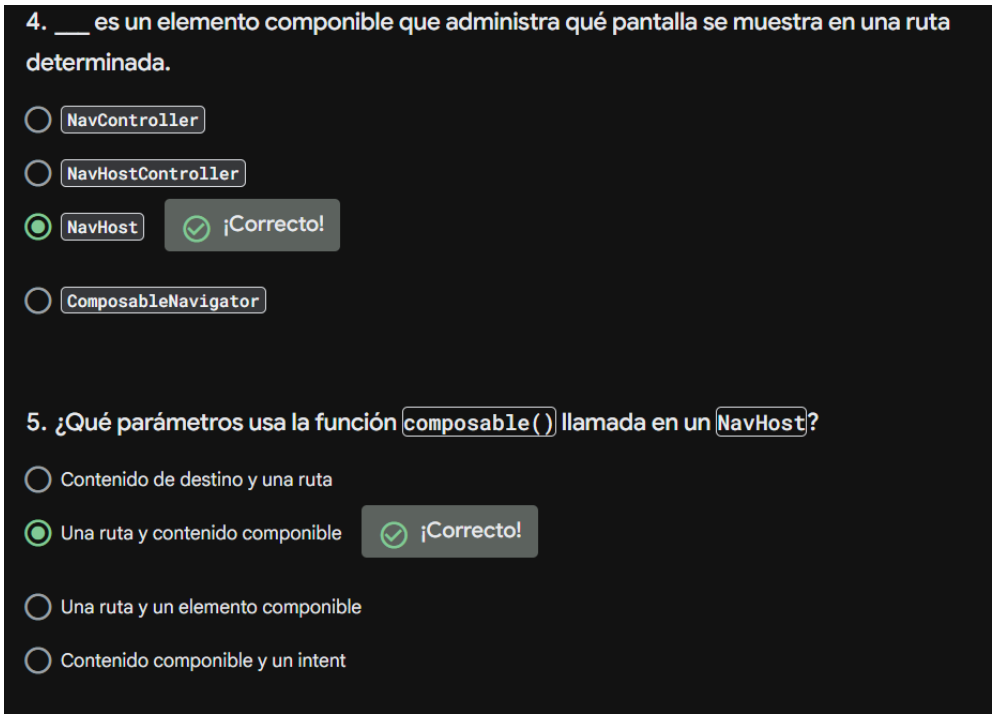
En el video se presenta Cupcake, una app de ejemplo donde el usuario puede hacer pedidos de cupcakes eligiendo cantidad, sabor y fecha de recogida. Esta aplicación se utiliza como caso práctico para demostrar cómo implementar navegación entre pantallas en Android usando Jetpack Compose.

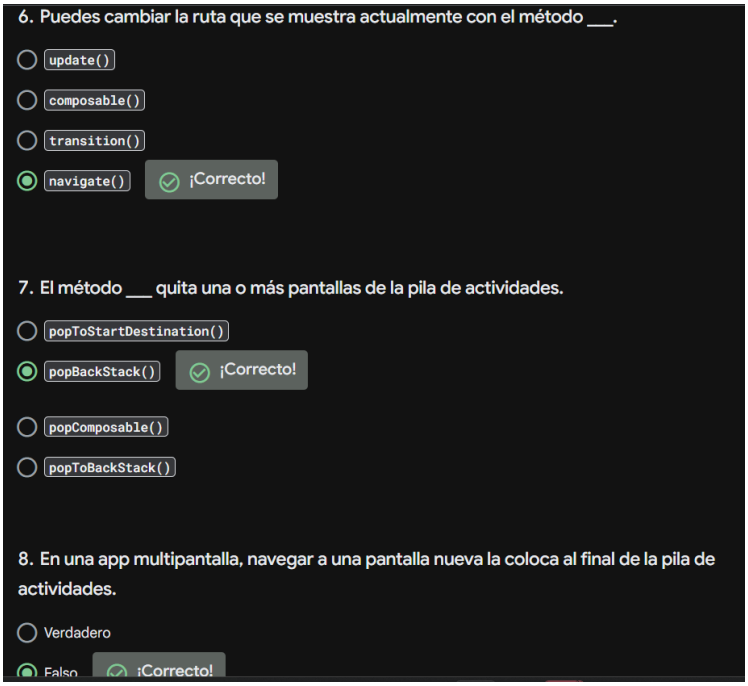
Puntos clave:

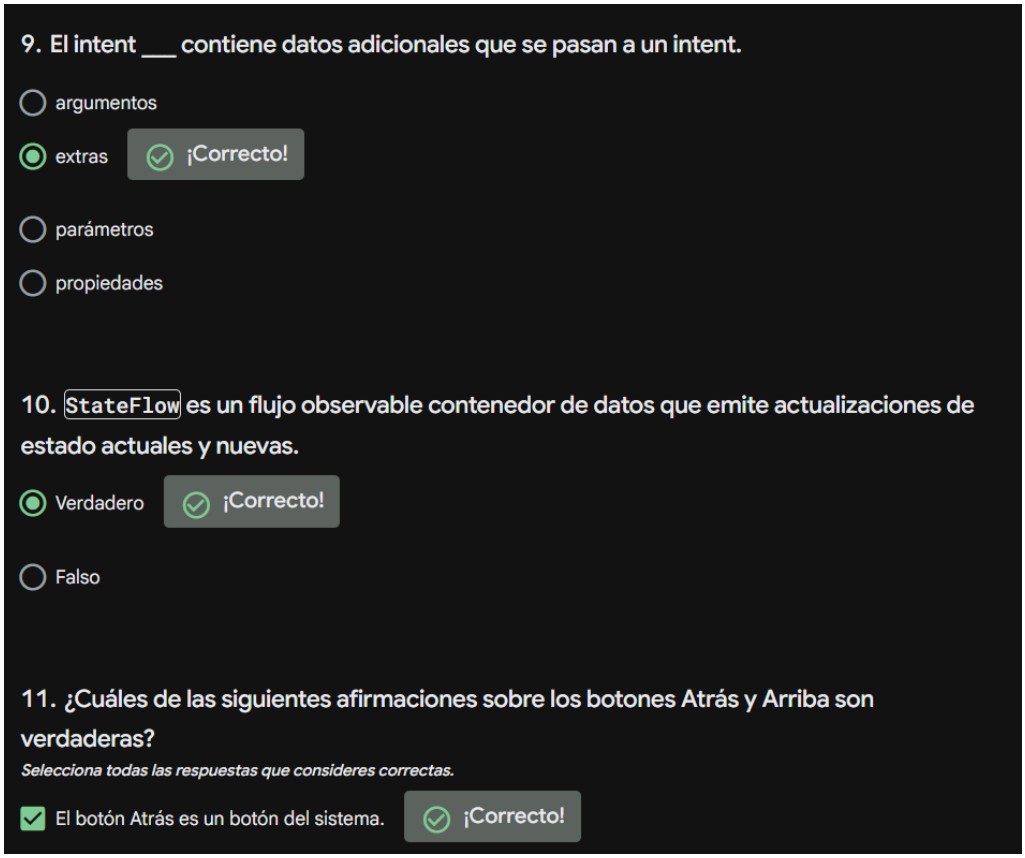
- Se explica cómo construir una arquitectura de navegación clara utilizando los componentes NavHost y NavController.
- Se muestra cómo transferir datos entre pantallas de forma eficiente.
- Se utiliza un ViewModel compartido para mantener el estado del pedido a lo largo de la navegación, asegurando consistencia en los datos.
- Se enfatiza la importancia de realizar pruebas de UI para verificar que tanto la navegación como el flujo funcional se comporten correctamente.

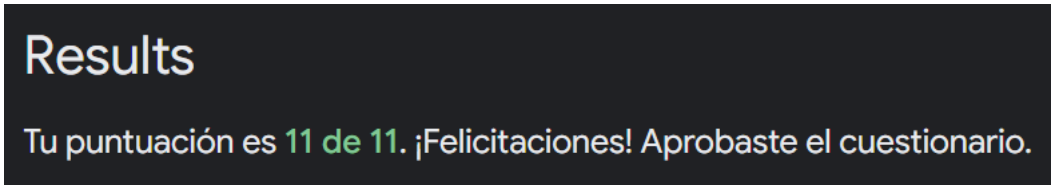
CUESTIONARIO DE RUTA 2


1. Una ruta se define con un tipo de datos ___.
Función @Composable
NavHost.Route
String
¡Correcto!
NavRoute
2. Con un NavHost, debes especificar explícitamente una pantalla de inicio.
Verdadero
¡Correcto!
Falso
3. La práctica recomendada es no pasar un NavHostController a elementos de componibilidad individuales.
Verdadero
¡Correcto!
Falso

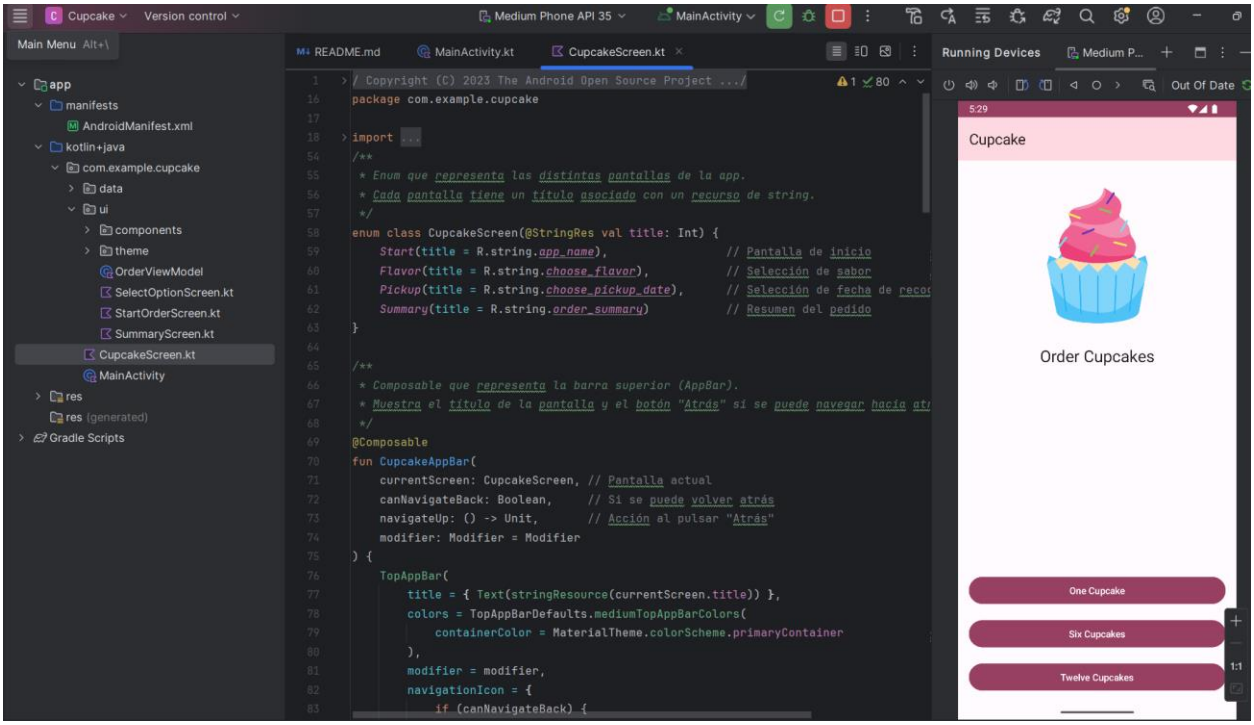

4. ___ es un elemento componible que administra qué pantalla se muestra en una ruta determinada.
NavController
NavHostController
NavHost
¡Correcto!
ComposableNavigator
5. ¿Qué parámetros usa la función composable() llamada en un NavHost?
Contenido de destino y una ruta
Una ruta y contenido componible
¡Correcto!
Una ruta y un elemento componible
Contenido componible y un intent


6. Puedes cambiar la ruta que se muestra actualmente con el método ___.
update()
composable()
transition()
navigate()
¡Correcto!
7. El método ___ quita una o más pantallas de la pila de actividades.
popToStartDestination()
popBackStack()
¡Correcto!
popComposable()
popToBackStack()
8. En una app multipantalla, navegar a una pantalla nueva la coloca al final de la pila de actividades.
Verdadero
Falso
¡Correcto!


9. El intent ___ contiene datos adicionales que se pasan a un intent.
argumentos
extras
¡Correcto!
parámetros
propiedades
10. StateFlow es un flujo observable contenedor de datos que emite actualizaciones de estado actuales y nuevas.
Verdadero
¡Correcto!
Falso
11. ¿Cuáles de las siguientes afirmaciones sobre los botones Atrás y Arriba son verdaderas?
Selecciona todas las respuestas que consideres correctas.
El botón Atrás es un botón del sistema.
¡Correcto!



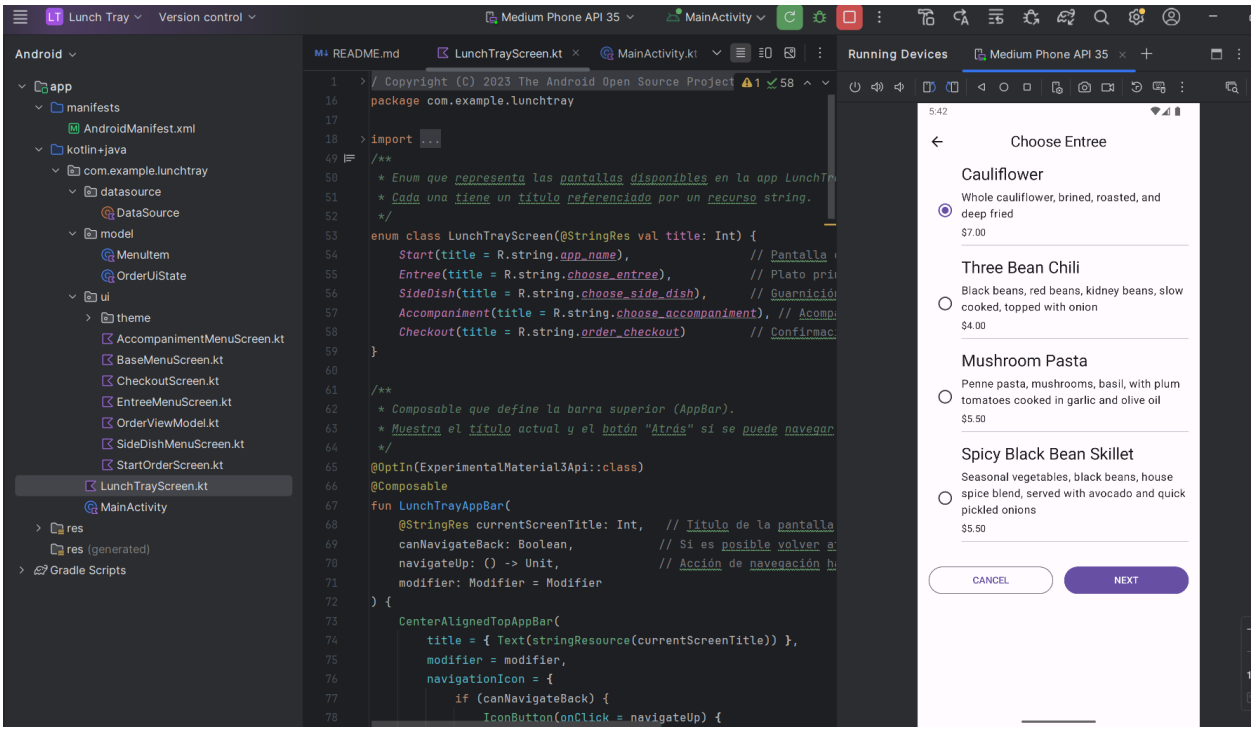
CUPCAKE



Este archivo implementa la navegación principal de la app Cupcake usando Jetpack Compose y Navigation. Se compone de:

- 4 pantallas principales (Start, Flavor, Pickup, Summary) representadas por un enum.
- Un AppBar que muestra el título correspondiente y un botón de regreso si aplica.
- Un ViewModel que gestiona el estado del pedido del usuario.
- Un NavHost que permite moverse entre pantallas, y pasar datos entre ellas usando el ViewModel.
- Funciones para cancelar un pedido (reseteando el estado) o compartir el resumen del pedido mediante un intent.

LUNCHTRAY



Esta app llamada LunchTray es una app de selección de menú paso a paso, y funciona de la siguiente forma:

- Define una navegación estructurada en 5 pantallas (Inicio, Plato principal, Guarnición, Acompañamiento y Confirmación).
- Usa un enum class para representar las pantallas con sus títulos.
- Un AppBar centrado que muestra el nombre de la pantalla y un botón de navegación hacia atrás cuando sea necesario.
- Un ViewModel controla el estado de la orden, como lo que selecciona el usuario.
- Usa NavHost para definir las rutas y cómo se navega de una pantalla a otra.
- En cada paso, el usuario puede:
    - Elegir una opción (plato principal, guarnición, etc.)
    - Cancelar el pedido (reseteando el estado y volviendo al inicio)
    - Avanzar a la siguiente pantalla.

Esto sigue el patrón de wizard o flujo guiado, útil para apps de pedidos, formularios o selección paso a paso.

RUTA DE APRENDIZAJE 3 – ADAPTATE A DIFERENTES TAMAÑO DE PANTALLA

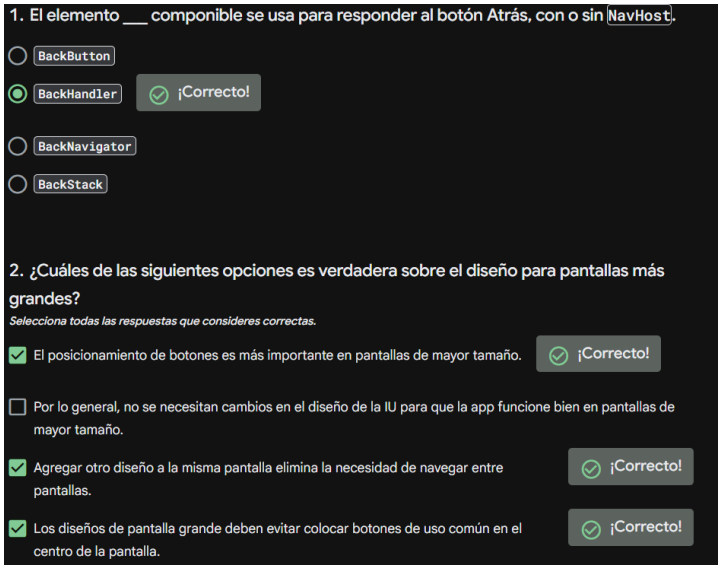
VIDEO 1 RUTA DE APRENDIZAJE 3

El video aborda la importancia de crear interfaces adaptativas en Android, capaces de ajustarse a diferentes tipos de dispositivos como teléfonos, tablets y dispositivos plegables. El objetivo es proporcionar una experiencia de usuario óptima y coherente, sin importar el tamaño o configuración de pantalla.

Puntos clave:

- Clases de tamaño de ventana: Se introducen las clases compacto, mediano y expandido para categorizar el espacio disponible en la interfaz. Esto permite adaptar el diseño de forma inteligente según el dispositivo.
- Componentes adaptativos de Compose: Se presentan componentes como NavigationSuiteScaffold, ListDetailPaneScaffold y SupportingPaneScaffold del paquete androidx.compose.material3.adaptive, que ayudan a construir interfaces dinámicas que responden a cambios de tamaño y postura del dispositivo.
- Mejores prácticas: Se recomienda probar las interfaces en distintas configuraciones de pantalla, usar bibliotecas especializadas y seguir las herramientas oficiales de Jetpack Compose para asegurar compatibilidad y buena usabilidad en todos los dispositivos.

CUESTIONARIO DE RUTA 3



3. La medida ___ es una medida específica de ancho o alto en la que debe cambiar el diseño de una app.

- (●) clase de ventana — Incorrecto.
- ( ) punto de diseño
- ( ) bucket de tamaño
- ( ) punto de interrupción

4. Por lo general, la clase de tamaño de ventana de ancho compacto se refiere a dispositivos más pequeños, como los teléfonos en modo vertical.

- (●) Verdadero — ¡Correcto!
- ( ) Falso

5. La API de ___ simplifica la implementación de diseños adaptables.

- ( ) `SizeClass`
- ( ) `WindowSizeState`
- ( ) `SizeBucket`
- (●) `WindowSizeClass` — ¡Correcto!

6. Un riel de navegación suele ser apropiado para diseños de ancho ___.

- ( ) compacto
- ( ) estándar
- (●) media — ¡Correcto!
- ( ) expandido

7. Cuando creas apps con diseños adaptables, debes usar una sola vista previa para cada pantalla.

- ( ) Verdadero
- (●) Falso — ¡Correcto!

8. El diseño de lista y detalles requiere la navegación hacia atrás en pantallas compactas, pero no en pantallas en las que se muestran al mismo tiempo las pantallas de lista y de detalles.

- (●) Verdadero — ¡Correcto!
- ( ) Falso

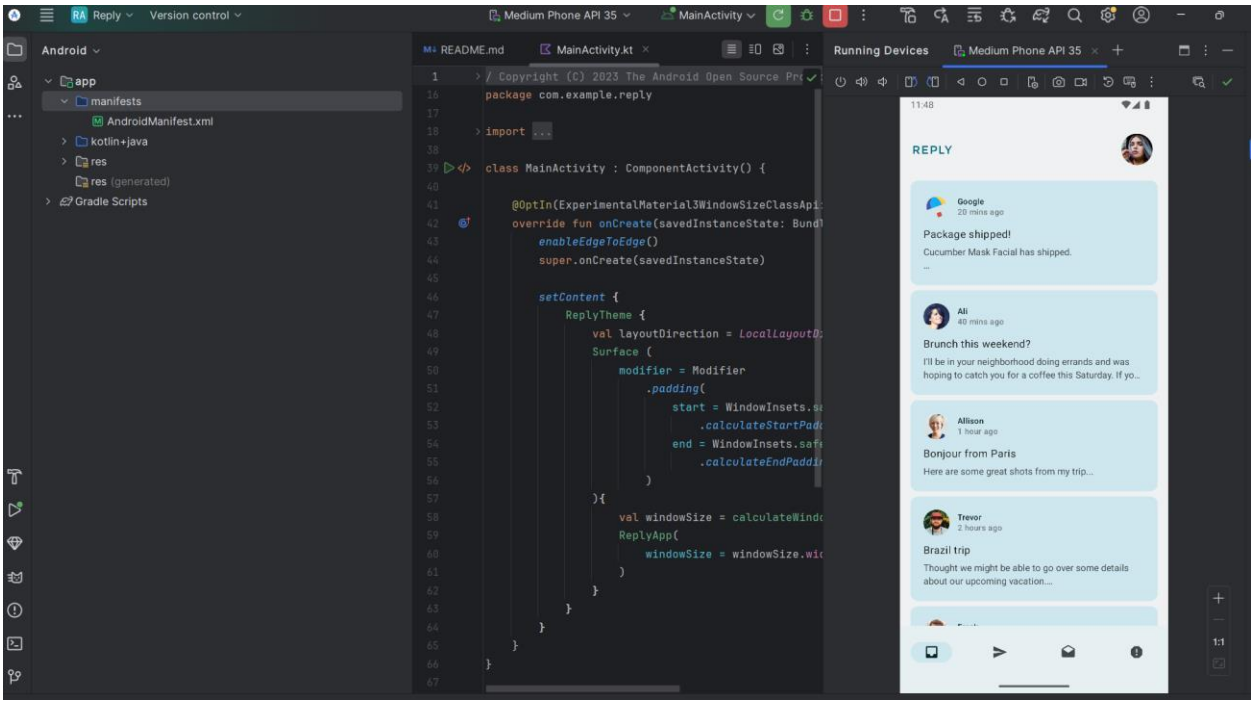
9. Supongamos que tienes una app de contactos que muestra una lista de contactos y detalles para cada uno de ellos. ¿Cuáles son las formas adecuadas de adaptar la IU a diferentes tamaños de pantalla?

- (●) Usa el diseño de lista y detalles para mostrar uno o dos paneles en paralelo, según el ancho disponible de la pantalla. — ¡Correcto!
- ( ) Los elementos de la lista deben ocupar todo el ancho de la pantalla, independientemente de qué tan angosta o ancha sea.
- ( ) El botón Arriba siempre debe aparecer dentro de la app y, si se hace clic en él, se debería salir de la app.
- ( ) Cuando se rota el dispositivo, el elemento seleccionado en la lista (y los detalles correspondientes que se muestran) se debe restablecer en el primer elemento de la lista.
- ( ) Es necesario usar el componente de Jetpack Navigation para que la IU sea responsiva a diferentes tamaños de pantalla.

10. Las pruebas se pueden configurar para ejecutar solo funciones de prueba con anotaciones personalizadas a través de la configuración de ___.

- ( ) módulo
- ( ) paquete
- ( ) clase de instrumentación
- (●) argumentos de instrumentación — ¡Correcto!

REPLYAPP



Esta función ReplyApp es el punto de entrada de una app de correo electrónico (como Gmail), hecha con Jetpack Compose. Se adapta dinámicamente al tamaño de pantalla usando el parámetro windowSize para elegir entre tres tipos de navegación:

- BottomNavigation (pequeñas),
- NavigationRail (medianas),
- PermanentNavigationDrawer (grandes).

Usa un ViewModel para gestionar el estado de la UI, y pasa datos y acciones a ReplyHomeScreen, que es la interfaz principal. La navegación y la lógica de interacción están bien separadas, siguiendo las buenas prácticas de arquitectura moderna en Android.