Indice

Introducción

Esta aplicación móvil permite a los usuarios explorar un menú interactivo de comidas y ofertas, además de filtrar por tipos de comida a mostrar.

Descripción del Proyecto

El objetivo del proyecto es desarrollar una aplicación que facilite la gestión y visualización de un menú interactivo de comidas. Los usuarios pueden explorar ofertas, filtrar por tipo de comida y gestionar su carrito de compras.
Entre sus características principales están:
- Visualización de comidas por tipo.
- Auto-scroll desactivable por el usuario para destacar ofertas de forma constante.
- Filtro dinámico para tipos de comida.

Requisitos

- Android Studio 2022.2.1 o superior
- SDK de Android 33 o superior
- JDK 11+

Configuración del Entorno

1. Clona el repositorio del proyecto desde: https://github.com/Alexfh94/TrabajoFinalUnidad3
2. Abre el proyecto en Android Studio.
3. Configura el SDK de Android y verifica que esté actualizado.
4. Asegúrate de que las dependencias de Gradle estén sincronizadas.

Estructura del Proyecto

```
/TrabajoFinalUnidad3
|
├── app/                    # Carpeta raíz del proyecto Android
|   ├── manifests/          # Archivos de manifiesto de la aplicación
|   |   └── AndroidManifest.xml  # Configuración global de la aplicación
|   ├── java/               # Código fuente de la aplicación
|   |   ├── com.example.trabajofinalunidad3/
|   |   |   ├── Comida.java        # Clase que representa los objetos "Comida"
|   |   |   ├── ComidaAdapter.java # Adaptador para gestionar la lista de
                                     comidas
|   |   |   ├── MainActivity.java  # Actividad principal que gestiona la
                                     interfaz de usuario
|   |   |   └── OfertaAdapter.java # Adaptador para gestionar las ofertas
                                     relacionadas
|   |   ├── (androidTest)/       # Pruebas de instrumentación (interacción con
                                   la UI)
|   |   └── (test)/              # Pruebas unitarias para lógica del negocio
|   ├── res/                # Recursos estáticos de la aplicación
|   |   ├── drawable/           # Imágenes y recursos gráficos
|   |   ├── layout/             # Archivos XML para el diseño de la interfaz
|   |   ├── mipmap/             # Iconos de la aplicación para distintas
                                  densidades
|   |   └── values/             # Strings, estilos y otros valores
                                  reutilizables
```

```
│   │       └── strings.xml       # Cadena de texto utilizadas en la app
│   └── build.gradle              # Archivo de configuración para la compilación
│
├── build/                        # Archivos generados automáticamente al
                                    compilar
├── gradle/                       # Configuración de Gradle para
                                    la automatización del proyecto
└── README.md                     # Descripción general del proyecto
                                    (documentación básica)
```

MainActivity.java

- **onCreate(Bundle savedInstanceState)**: Método principal que inicializa la actividad. Configura la vista, genera los datos de la lista de comidas y configura los elementos de la interfaz como botones, RecyclerViews y Spinner.

- **generarArrayDatos()**: Genera y devuelve una lista de objetos Comida con información precargada (nombre, imagen, descripción, precio, oferta y tipo).
- **setupImageButtons()**: Configura botones de imagen para mostrar un mensaje *"Próximamente"* mediante un Toast cuando se presionan.
- **setupRecyclerViews(ArrayList<Comida> comidasArrayList)**: Configura dos RecyclerViews:
    - Uno para mostrar todas las comidas.
    - Otro para mostrar solo las comidas con ofertas. También inicializa los adaptadores y divide las comidas en dos listas: comidaArrayList y ofertaArrayList.
- **setupSpinner(ArrayList<Comida> comidasArrayList, ArrayList<Comida> comidaArrayList, ComidaAdapter comidaAdapter)**: Configura un Spinner para filtrar las comidas por tipo. Al seleccionar una opción, llama a filtroComidas para actualizar la lista mostrada.

- **filtroComidas(ArrayList<Comida> comidasArrayList, ArrayList<Comida> comidaArrayList, String selectedItem)**: Filtra las comidas según el tipo seleccionado en el Spinner. Actualiza la lista comidaArrayList.
- **setupAutoScroll(RecyclerView rvOfertas, ArrayList<Comida> ofertaArrayList, OfertaAdapter ofertaAdapter)**: Configura el autoscroll del RecyclerView de ofertas. Usa un Runnable para añadir elementos y desplazarse automáticamente. También responde al estado del interruptor (Switch) que activa/desactiva esta función.

  (FUENTE: ChatGPT y consultas con el profesor para su implementación)
- **getRandomOferta()**: Genera y devuelve un número aleatorio representando una oferta (0 si no hay oferta). Simplemente para que la aplicación cambie con cada ejecución para comprobar como se vería al aplicar cambios. 50% de probabilidad de tener oferta o no, para que tanto el array de ofertas como el de comidas tengan contenido.
- **onClickCarrito(View view, int position)**: Maneja el evento al presionar el botón "Añadir al carrito" en un elemento del RecyclerView de comidas. Muestra un mensaje indicando que se añadió al carrito.
- **onClickCardComida(View view, int position)**: Maneja el evento al presionar cualquier lugar fuera del botón en un elemento del RecyclerView de comidas. Muestra el nombre y el precio de la comida.
- **onClickCardOferta(View view, int position)**: Maneja el evento al hacer clic en un elemento del RecyclerView de ofertas. (El método está incompleto en el fragmento).
- Métodos de la interfaz **ComidaAdapter.OnClickComida** y **OfertaAdapter.OnClickOferta**: Estos permiten gestionar los eventos relacionados con los clics en los elementos y botones de las tarjetas en los adaptadores.

Comida.java

1. **Constructor**
   - Comida(String titulo, int imagen, String descripcion, double precio, int imgOferta, TipoComida tipo)
     Permite crear una instancia de Comida con todos sus atributos.
2. **Getters y Setters**
   - getTitulo() / setTitulo(String titulo): Obtiene o establece el título de la comida.
   - getImagen() / setImagen(int imagen): Obtiene o establece el recurso de imagen asociado a la comida.
   - getDescripcion() / setDescripcion(String descripcion): Obtiene o establece la descripción de la comida.

- getPrecio() / setPrecio(double precio):
    - **Getter**: Calcula y devuelve el precio considerando la oferta asignada (imgOferta). Redondea el resultado a 2 decimales.
    - **Setter**: Establece el precio base de la comida.
- getImgOferta() / setImgOferta(int imgOferta): Obtiene o establece el identificador de oferta asociado.
- getTipo() / setTipo(TipoComida tipo): Obtiene o establece el tipo de comida (CARNE, PESCADO, VEGETALES, OTROS).

3. **toString()**
    - Devuelve una representación en formato String de los atributos de la clase para facilitar la depuración.

`ComidaAdapter.java`

1. **Constructor**
    - ComidaAdapter(ArrayList<Comida> coleccion, OnClickComida listener, Context context)
      Inicializa el adaptador con la lista de comidas (coleccion), un listener para clics y el contexto.
2. **Métodos del Adaptador**
    - onCreateViewHolder(ViewGroup parent, int viewType)
      Infla la vista de una tarjeta (layout ficha_comida) y devuelve un ViewHolder para manejarla.
    - onBindViewHolder(ComidaViewHolder holder, int position)
      Enlaza los datos de una instancia de Comida con la vista correspondiente en la posición indicada.
    - getItemCount()
      Devuelve el tamaño de la lista de comidas.
3. **Clase Interna** ComidaViewHolder
    - Representa la vista individual de cada elemento. Contiene:
        - ImageView imageView: Para mostrar la imagen de la comida.
        - TextView tv_titulo, tv_descripcion, tv_precio: Para mostrar el título, descripción y precio.
        - Button btnAccion: Para un botón de acción relacionado con la comida.
4. **Interfaz** OnClickComida
    - Define métodos para manejar clics en:
        - Una tarjeta (onClickCardComida(View view, int position)).

- El botón de acción de una tarjeta (onClickCarrito(View view, int position)).

OfertaAdapter.java

1. **Constructor**
    - OfertaAdapter(ArrayList<Comida> coleccion, OnClickOferta listener, Context context)
      Similar a ComidaAdapter, inicializa el adaptador para gestionar comidas en oferta.
2. **Métodos del Adaptador**
    - onCreateViewHolder(ViewGroup parent, int viewType)
      Infla la vista de una tarjeta (layout ficha_oferta) y devuelve un ViewHolder para manejarla.
    - onBindViewHolder(OfertaViewHolder holder, int position)
      Enlaza los datos de una instancia de Comida con la vista correspondiente en la posición indicada.
      También establece la imagen de la oferta (imgOferta) en función de los valores.
    - getItemCount()
      Devuelve el tamaño de la lista de comidas.
3. **Clase Interna** OfertaViewHolder
    - Representa la vista individual de cada oferta. Contiene:
        - ImageView imgComida: Para mostrar la imagen de la comida.
        - TextView tv_titulo, tv_precio: Para mostrar el título y precio.
        - ImageView imgOferta: Para mostrar la imagen asociada a la oferta.
4. **Interfaz** OnClickOferta
    - Define un método para manejar clics en una tarjeta de oferta:
        - onClickCardOferta(View view, int position).

Instalación

1. Con Android Studio abierto, selecciona 'Run' > 'Run app'.
2. Conecta un dispositivo Android o utiliza un emulador.
3. Espera a que la aplicación se compile e instale automáticamente.

Uso

1. Abre la aplicación.
2. Explora las comidas disponibles.
3. Filtra las comidas por tipo utilizando el Spinner.
4. Añade comidas al carrito mediante los botones de las tarjetas.
5. Revisa las ofertas en el carrusel automático, y añade al carrito haciendo clic sobre ellas.

Pruebas

Se realizaron las siguientes pruebas para garantizar el correcto funcionamiento de la aplicación:

1. Prueba de Visualización de Comidas

- **Objetivo:** Verificar que todas las comidas se muestran correctamente en el RecyclerView principal.
- **Procedimiento:**
    1. Abrir la aplicación.
    2. Asegurarse de que las imágenes, títulos, descripciones y precios aparecen correctamente.
    3. Confirmar que los precios se ajustan dinámicamente si la comida tiene una oferta activa.
- **Resultado esperado:** Todas las comidas deben ser visibles y los datos deben coincidir con los proporcionados en el archivo de datos.

2. Prueba de Filtro de Comidas (Spinner)

- **Objetivo:** Validar que el Spinner filtra correctamente las comidas según el tipo seleccionado.
- **Procedimiento:**
    1. Seleccionar un tipo de comida desde el Spinner (e.g., "Carne").
    2. Observar el RecyclerView principal y verificar que solo aparecen comidas del tipo seleccionado.
    3. Cambiar entre diferentes opciones del Spinner, incluyendo "Todos", y confirmar que las comidas se actualizan correctamente.
- **Resultado esperado:** El RecyclerView debe mostrar únicamente las comidas que corresponden al tipo seleccionado.

3. Prueba de Auto-scroll en Ofertas

- **Objetivo:** Verificar que el auto-scroll en el RecyclerView de ofertas funciona sin interrupciones.
- **Procedimiento:**
    1. Observar el RecyclerView de ofertas al iniciar la aplicación.
    2. Confirmar que las ofertas se desplazan automáticamente cada 3 segundos.
    3. Activar y desactivar el switch de auto-scroll para verificar que el comportamiento cambia dinámicamente.
- **Resultado esperado:** El auto-scroll debe desplazarse automáticamente cuando el switch está activado y detenerse al desactivarlo.

4. Prueba de Botones de Imagen (Toast)

- **Objetivo:** Verificar que los botones de imagen (perfil, recientes, configuración, carrito) muestran el mensaje "Coming soon".
- **Procedimiento:**
    1. Pulsar cada botón de imagen en la interfaz principal.
    2. Confirmar que aparece el mensaje "Coming soon" como un Toast en cada caso.
- **Resultado esperado:** Cada botón debe mostrar consistentemente el mensaje "Coming soon".

5. Prueba de Funcionalidad "Añadir al carrito"

- **Objetivo:** Verificar que las comidas pueden añadirse correctamente al carrito desde el RecyclerView principal y el de ofertas.
- **Procedimiento:**
    1. Pulsar el botón "Añadir al carrito" en varias comidas del RecyclerView principal y de ofertas.
    2. Observar que aparece un Toast confirmando la acción.
    3. Verificar que el nombre de la comida en el Toast coincide con la comida seleccionada.
- **Resultado esperado:** El mensaje del Toast debe confirmar la adición correcta al carrito, incluyendo el nombre de la comida seleccionada.

6. Prueba de Gestión de Excepciones

- **Objetivo:** Garantizar que la aplicación maneja errores comunes sin cerrarse inesperadamente.
- **Procedimiento:**
    1. Intentar realizar acciones inesperadas, como desactivar el auto-scroll antes de cargar las ofertas.
    2. Manipular el Spinner sin seleccionar una opción válida.
    3. Cargar imágenes o datos corruptos y verificar que se manejan con mensajes de error claros.
- **Resultado esperado:** La aplicación debe manejar todos los casos sin cerrarse y mostrar mensajes de error cuando sea necesario.

7. Prueba de Rendimiento con Grandes Listas de Datos

- **Objetivo:** Validar el rendimiento de los RecyclerViews al añadir una gran cantidad de elementos, asegurando fluidez y estabilidad.
- **Procedimiento:**
    1. Generar dinámicamente una lista de comidas con cientos de elementos.
    2. Verificar que el RecyclerView principal funciona correctamente al desplazarse por toda la lista.
    3. Probar el RecyclerView de ofertas con un número masivo de elementos para simular un scroll "infinito".
- **Resultado esperado:** Ambos RecyclerViews deben funcionar sin ralentizaciones, y el RecyclerView de ofertas debe permitir un scroll continuo e ininterrumpido.

Flujo de Datos

El flujo de datos en la aplicación sigue un modelo estructurado que permite la interacción entre la interfaz gráfica, las listas de datos y los eventos del usuario:

1. **Generación de Datos**

   Los datos de las comidas y las ofertas se generan dinámicamente en el método generarArrayDatos de la clase MainActivity. Cada instancia de Comida contiene atributos como título, descripción, imagen, precio y tipo.

2. **Configuración de Adaptadores**

   Las listas de datos generadas se asignan a dos adaptadores:

   - **ComidaAdapter**: Gestiona los elementos mostrados en el RecyclerView principal.
   - **OfertaAdapter**: Maneja los elementos del RecyclerView de ofertas.

3. **Interacción con el Usuario**

   - **Filtros**: Los datos mostrados en el RecyclerView principal se actualizan dinámicamente al seleccionar un tipo de comida en el Spinner. Esto se gestiona con el método filtroComidas.
   - **Añadir al Carrito**: Al pulsar en el botón de una tarjeta, el evento se maneja mediante los métodos onClickCarrito o onClickCardOferta.

4. **Auto-Scroll**

   El RecyclerView de ofertas utiliza un Runnable para simular un scroll continuo. Este flujo es activado/desactivado mediante un interruptor (Switch).

5. **Actualización en Tiempo Real**

   Las vistas se actualizan dinámicamente con los adaptadores al modificar las listas de datos, garantizando que los cambios se reflejen en pantalla.

Futuras Mejoras

1. **Persistencia de Datos**

    - Implementar almacenamiento local para guardar las selecciones del usuario, como elementos añadidos al carrito, preferencias de filtrado o estado del auto-scroll.

2. **Mejoras en las ofertas**

    - Añadir la posibilidad de ajustar las ofertas disponibles desde el panel de ajustes para un usuario administrador.

3. **Mejoras en los recyclerView**

    - Cambiar la funcionalidad actual de mostrar toast en cada tarjeta por descripciones avanzadas o por la funcion de añadir al carrito real al pulsar el boton.

4. **Carrito Avanzado**

    - Implementar una nueva actividad donde los usuarios puedan gestionar el contenido del carrito con opciones para editar, eliminar y confirmar compras.

5. **Conexión a Bases de Datos Remotas**

    - Integrar un backend para obtener datos de comidas y ofertas dinámicamente, mejorando la personalización y escalabilidad del proyecto.

6. **Implementar funciones reales a los botones del menu**

    - Integrar funciones reales para los botones de perfil, recientes y ajustes.

Contacto

- Autor: Alejandro De La Fuente Heras
- Email: alexfh94@gmail.com
- GitHub: https://github.com/Alexfh94/TrabajoFinalUnidad3