ENLACE GITHUB

1. Edición de MediaList

Para este aplicación hemos hecho uso del framework Jetpack Compose para la creación de interfaces de usuario en Android con Kotlin. Hemos hecho uso de la plantilla del curso de JetPackCompose de Antonio Leiva, modificándola para nuestra necesidades usándo su videotutorial.

El primer elemento que hemos modificado es la “MediaList”, este elemento está diseñado para representar una lista de elementos multimedia en la pantalla principal de la aplicación. A continuación, se comparan dos versiones del código que realizan funciones similares, pero con algunas diferencias notables.

Versión modificada

```
package com.antonioleiva.mymovies.ui.screens.main

// (imports omitidos por brevedad)

@Composable
fun MediaList(
    onClick: (MediaItem) -> Unit,
    modifier: Modifier = Modifier
) {
    val mediaList: List<MediaItem> = getMedia()

    LazyColumn(
        contentPadding = PaddingValues(dimensionResource(R.dimen.padding_xsmall)),
        modifier = modifier
    ) {
        items(mediaList) { mediaItem ->
            MediaListItem(
                mediaItem = mediaItem,
                onClick = { onClick(mediaItem) },
                modifier = Modifier.padding(dimensionResource(R.dimen.padding_xsmall))
            )
        }
    }
}

// (código de MediaListItem y Title omitidos por brevedad)
```

```
package com.antonioleiva.mymovies.ui.screens.main

// (imports omitidos por brevedad)

@ExperimentalFoundationApi
@Composable
fun MediaList(
   onClick: (MediaItem) -> Unit,
   modifier: Modifier = Modifier
) {
   LazyVerticalGrid(
      cells = GridCells.Adaptive(dimensionResource(R.dimen.cell_min_width)),
      contentPadding = PaddingValues(dimensionResource(R.dimen.padding_xsmall)),
      modifier = modifier
   ) {
      items(getMedia()) {
         MediaListItem(
            mediaItem = it,
            onClick = { onClick(it) },
            modifier = Modifier.padding(dimensionResource(R.dimen.padding_xsmall))
         )
      }
   }
}

// (código de MediaListItem y Title omitidos por brevedad)
```

Versión Original

La principal diferencia radica en cómo se organiza visualmente la lista. En la primera versión, se utiliza LazyColumn para representar la lista de manera vertical. Cada elemento de la lista es representado por el componente MediaListItem, que a su vez contiene un Card con una miniatura y un título.

En la segunda versión, se utiliza LazyVerticalGrid para representar la lista en una cuadrícula vertical, lo que permite mostrar varios elementos en una fila.

Ambas versiones comparten la estructura básica, como la función MediaListItem que representa un elemento individual en la lista con una miniatura y un título. La elección entre las dos versiones dependerá de las preferencias de diseño y los requisitos específicos del desarrollador. La segunda versión proporciona una disposición en cuadrícula, lo que puede ser útil en ciertos contextos.

```kotlin
@Composable
fun MediaList(
    onClick: (MediaItem) -> Unit,
    modifier: Modifier = Modifier
) {
    val mediaList: List<MediaItem> = getMedia()

    LazyColumn(
        contentPadding = PaddingValues(dimensionResource(2dp)),
        modifier = modifier
    ) { this: LazyListScope
        items(mediaList) { this: LazyItemScope  mediaItem ->
            MediaListItem(
                mediaItem = mediaItem,
                onClick = { onClick(mediaItem) },
                modifier = Modifier.padding(dimensionResource(2dp))
            )
        }
    }
}

@Composable
fun MediaListItem(
    mediaItem: MediaItem,
    onClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Card(
        modifier = modifier.clickable { onClick() }
    ) {
        Column { this: ColumnScope
            Thumb(mediaItem)
            Title(mediaItem)
        }
    }
}
@Composable
private fun Title(mediaItem: MediaItem) {
    Box(
        contentAlignment = Alignment.Center,
        modifier = Modifier
            .fillMaxWidth()
            .background(MaterialTheme.colors.secondary)
            .padding(dimensionResource(16dp))
    ) { this: BoxScope
        Text(
            text = mediaItem.title,
            style = MaterialTheme.typography.h6
        )
    }
}

@Preview
@Composable
fun MediaListItemPreview() {
    MyMoviesApp {
        val mediaItem = MediaItem( id: 1,  title: "Item 1",  thumb: "", MediaItem.Type.VIDEO)
        MediaListItem(mediaItem = mediaItem, onClick = {})
    }
}
```

2. Edicion de DetailScreen

Este código pertenece a la implementación de una pantalla de detalle, “DetailScreen”. Hemos modificado el aspecto para añadir encima del cuadro de imagen el titulo, el autor y una pequeña descripcion, la cual nos ha llevado a modificaciones en el objeto “MediaItem” que explicaremos en el siguiente punto. A continuación, se presenta una combinación de ambas versiones del código, junto con comentarios

```
package com.antonioleiva.mymovies.ui.screens.detail

// (imports omitidos por brevedad)

@Composable
fun DetailScreen(mediaId: Int, onUpClick: () -> Unit) {
  val mediaItem = remember { getMedia().first { it.id == mediaId } }

  Scaffold(
    topBar = {
      TopAppBar(
        title = { Text(text = mediaItem.title) },
        navigationIcon = { ArrowBackIcon(onUpClick) }
      )
    },
    content = {
      LazyColumn(
        modifier = Modifier
          .fillMaxSize()
          .padding(16.dp)
      ) {
        // Contenido omitido por brevedad
      }
    }
  )
}
```

que resaltan los cambios realizados:

```
package com.antonioleiva.mymovies.ui.screens.detail

// (imports omitidos por brevedad)

@Composable
fun DetailScreen(mediaId: Int, onUpClick: () -> Unit) {
  val mediaItem = remember { getMedia().first { it.id == mediaId } }

  Scaffold(
    topBar = {
      TopAppBar(
        title = { Text(text = mediaItem.title) },
        navigationIcon = { ArrowBackIcon(onUpClick) }
      )
    }
  ) {
    Thumb(mediaItem = mediaItem)
  }
}
```

Versión editada:

Versión original:

En la segunda versión, se ha eliminado el uso de LazyColumn y su contenido relacionado. En su lugar, el contenido principal de la pantalla es simplemente la miniatura (Thumb) del elemento multimedia.

Esto implica que la segunda versión simplifica la pantalla y muestra solo la imagen asociada al elemento multimedia sin información adicional como título, autor, descripción, etc.

El contenido principal (argumento content de Scaffold) se define directamente como la miniatura del elemento multimedia (Thumb). Esto es diferente de la primera versión, que utiliza un LazyColumn para mostrar información detallada del elemento multimedia.

La primera versión utiliza un LazyColumn para organizar y mostrar varios elementos de información en una estructura vertical. La segunda versión simplifica la estructura y muestra solo la miniatura del elemento multimedia, lo que puede ser adecuado para casos donde se prefiere una vista de detalle más minimalista.

```
@Composable
fun DetailScreen(mediaId: Int, onUpClick: () -> Unit) {
    val mediaItem = remember { getMedia().first { it.id == mediaId } }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text(text = mediaItem.title) },
                navigationIcon = { ArrowBackIcon(onUpClick) }
            )
        },
        content = { it: PaddingValues
            LazyColumn(
                modifier = Modifier
                    .fillMaxSize()
                    .padding(16.dp)
            ) { this: LazyListScope
                item { this: LazyItemScope
                    // Título del video
                    Text(
                        text = mediaItem.title,
                        style = Typography.hó,
                        modifier = Modifier.padding(bottom = 8.dp)
                    )
                }

                item { this: LazyItemScope
                    // Autor más pequeño
                    Text(
                        text = "Autor: ${mediaItem.author}",
                        style = Typography.caption,
                        modifier = Modifier.padding(bottom = 8.dp)
                    )
                }

                item { this: LazyItemScope
                    // Espacio para la descripción del video
                    Text(
                        text = mediaItem.description ?: "",
                        style = Typography.body1,
                        modifier = Modifier.padding(bottom = 16.dp)
                    )
                }

                item { this: LazyItemScope
                    // Imagen del video
                    Thumb(mediaItem = mediaItem)
                }
            }
        }
    )
}
```

3. Edición de MediaItem

Hemos definido una clase de datos llamada “MediaItem” que representa elementos multimedia. Ambas versiones comparten esta clase, pero difieren en la información proporcionada y la estructura de la función getMedia. A continuación, presentamos

```
package com.antonioleiva.mymovies.model

// (imports omitidos por brevedad)

data class MediaItem(
    val id: Int,
    val title: String,
    val thumb: String,
    val type: Type,
    val author: String? = null,
    val description: String? = null
) {
    enum class Type { PHOTO, VIDEO }
}

fun getMedia(): List<MediaItem> {
    return listOf(
        // Lista de objetos MediaItem
        // Cada objeto representa un elemento multimedia con información específica
        // Los objetos contienen información como título, miniatura, tipo, autor y descrip-
ción
        // Los elementos multimedia incluyen videos y fotos relacionadas con la Fórmula 1
        // Se proporciona información de ejemplo para varios elementos multimedia
    )
}
```

una versión combinada de ambas respuestas con comentarios detallados:

Versión modificada:

Versión original:

```
package com.antonioleiva.mymovies.model

// (imports omitidos por brevedad)

data class MediaItem(
    val id: Int,
    val title: String,
    val thumb: String,
    val type: Type
) {
    enum class Type { PHOTO, VIDEO }
}

fun getMedia() = (1..10).map {
    MediaItem(
        id = it,
        title = "Title $it",
        thumb = "https://loremflickr.com/400/400/cat?lock=$it",
        type = if (it % 3 == 0) Type.VIDEO else Type.PHOTO
    )
}
```

Autor y Descripción:

En el primer código, la clase MediaItem incluye propiedades adicionales como author y description que pueden contener información sobre el autor y la descripción del elemento multimedia.

En el segundo código, estas propiedades (author y description) no están presentes en la definición de MediaItem, simplificando la estructura y proporcionando solo las propiedades básicas.

Datos de Ejemplo:

En el primer código, la función getMedia proporciona una lista de objetos MediaItem con información de ejemplo específica para cada elemento multimedia, incluyendo títulos, miniaturas, tipos, autores y descripciones.

En el segundo código, la función getMedia utiliza un rango de 1 a 10 para crear automáticamente objetos MediaItem con títulos genéricos ("Title $it"), miniaturas de imágenes aleatorias y tipos (video o foto) basados en la condición if (it % 3 == 0).

```kotlin
package com.antonioleiva.mymovies.model

import com.antonioleiva.mymovies.model.MediaItem.Type

data class MediaItem(
    val id: Int,
    val title: String,
    val thumb: String,
    val type: Type,
    val author: String? = null,
    val description: String? = null
) {
    enum class Type { PHOTO, VIDEO }
}

fun getMedia(): List<MediaItem> {
    return listOf(
        MediaItem(
            id = 1,
            title = "Carrera Nocturna",
            thumb = "https://s1.sportstatics.com/relevo/www/multimedia/202311/21/media/cortadas/verstappen-RaG4DCHFRVicdUkHIFfjc8N-1200x648@Relevo.
            type = Type.VIDEO,
            author = "Pepe",
            description = "Emocionante carrera nocturna bajo las luces del circuito de Las Vegas"
        ),
        MediaItem(
            id = 2,
            title = "Detrás de Escena",
            thumb = "https://phantom-marca.unidadeditorial.es/efa333df293aee1e9f66f65d39218afb/crop/0x0/2044x1363/resize/828/f/jpg/assets/multimedi
            type = Type.VIDEO,
            author = "Jose",
            description = "Acceso exclusivo detrás de escena en el paddock de la F1 con los mejores pilotos"
        ),
        MediaItem(
            id = 3,
            title = "Duelo de Titanes",
            thumb = "https://cdn-images.motor.es/image/m/800w.webp/fotos-noticias/2021/08/duelo-hamilton-alonso-mas-alla-limite-siempre-queja-20218
            type = Type.VIDEO,
            author = "Pedro",
```

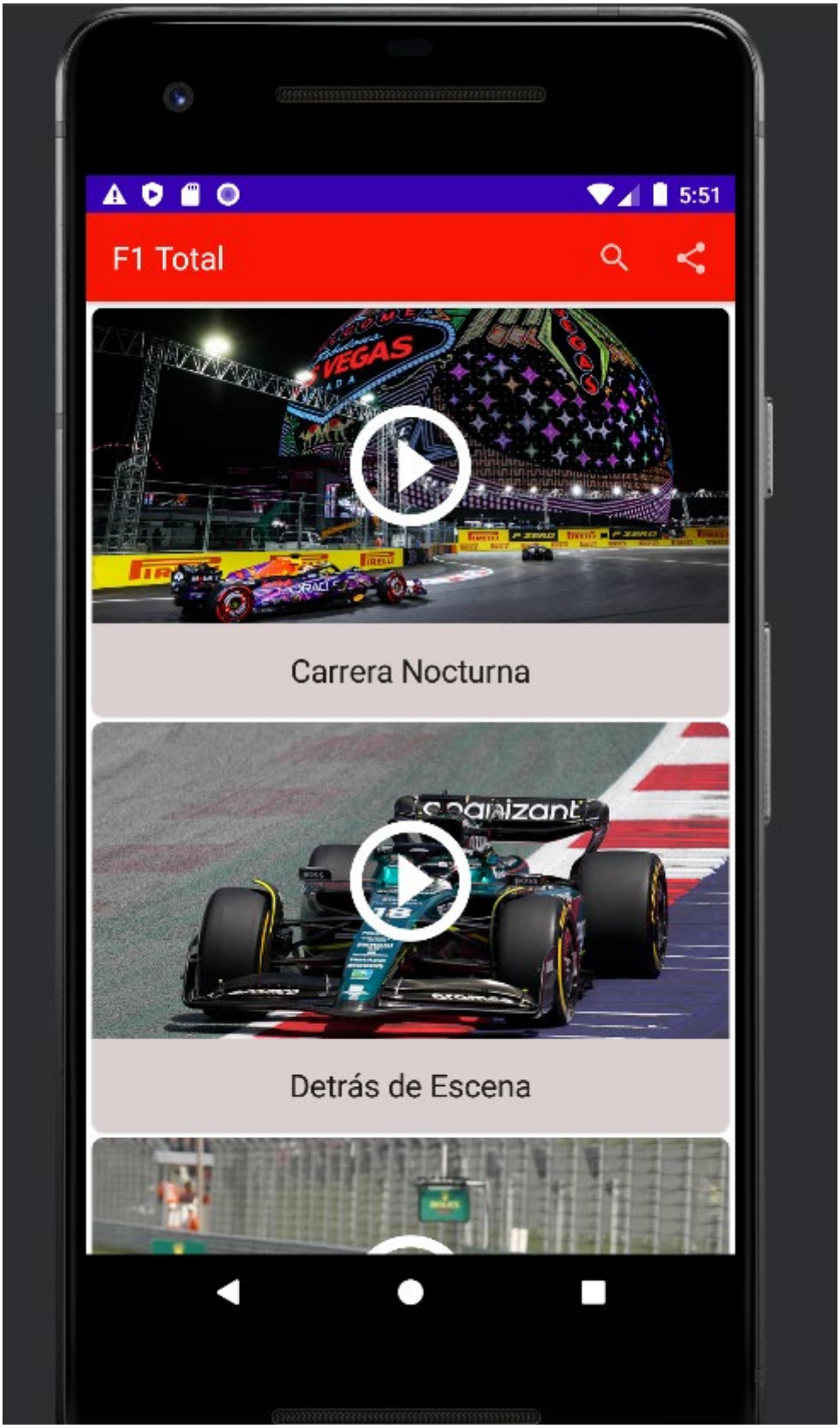
4. Modificaciones menores

Además de cambios en layout e información que cogen las cards, hemos modificado los colores de la aplicación y el titulo de la cabecera de la misma desde strings.xml. Son detalles menores, pero que tambien tuvieron que modificarse para darle identidad propia.

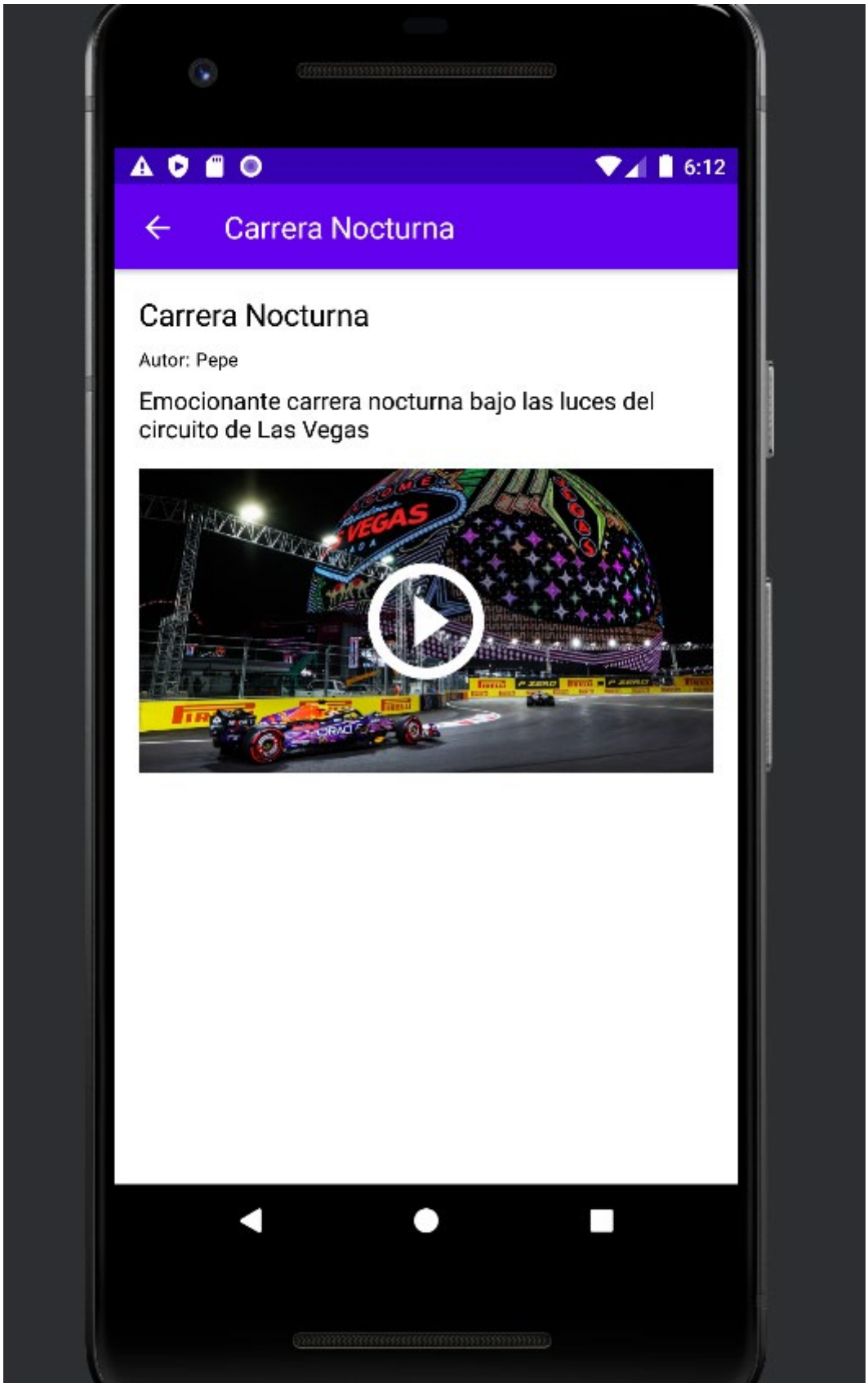
```xml
<resources>
    <string name="app_name">F1 Total</string>
</resources>
```

```kotlin
package com.antonioleiva.mymovies.ui.theme

import androidx.compose.ui.graphics.Color

val Purple200 = Color( color: 0xFF721610)
val Purple500 = Color( color: 0xFFF71504)
val Purple700 = Color( color: 0xFF13DFDF)
val Teal200 = Color( color: 0xFFDBD0D0)
```

5. Capturas de resultados finales




6:15
F1 Total
Velocidad en Monza
cognizant
aramco
Celebración en el Podio
HONDA
RedBull
ORACLE
Mobil 1
33%
33%
2022


6:12
Carrera Nocturna
Carrera Nocturna
Autor: Pepe
Emocionante carrera nocturna bajo las luces del circuito de Las Vegas
VEGAS