**Examen PMDM 2º DAM**
Ordinario
febrero 2025

I.E.S. Doctor Balmis
GENERALITAT VALENCIANA
Conselleria de Educación, Cultura y Deporte
Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

| Nombre | |
|---|---|

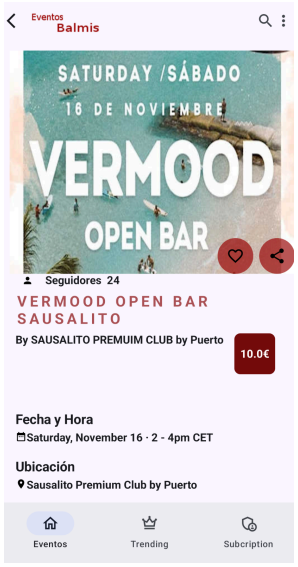
**Instrucciones para la realización de la prueba:**

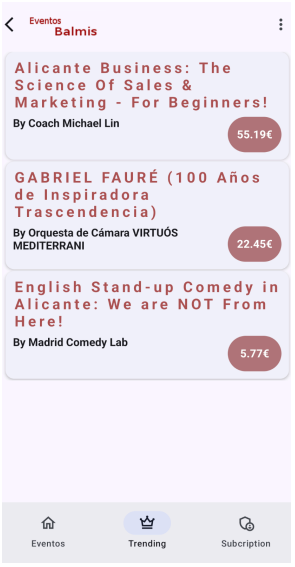
- Duración de la prueba **3 h 30 min** desde el comienzo de la misma.
- Lee atentamente el enunciado y no formules tus preguntas en voz alta, levanta la mano y espera a que el profesor te diga que tienes que hacer.
- Si te atascas en un apartado déjalo y pasa al siguiente. Cada apartado tiene su puntuación, independientemente del funcionamiento del programa final y el orden propuesto en el enunciado.
- **No se puede usar ningún tipo de dispositivo USB, ni código de ejercicios o ejemplos realizados en clase. Solo los pdf de apuntes que se te proporcionarán para la ocasión**.
- En caso de pillar a algún alumno copiando o usando material no autorizado se le retirará el examen y su calificación en la evaluación será un suspenso.

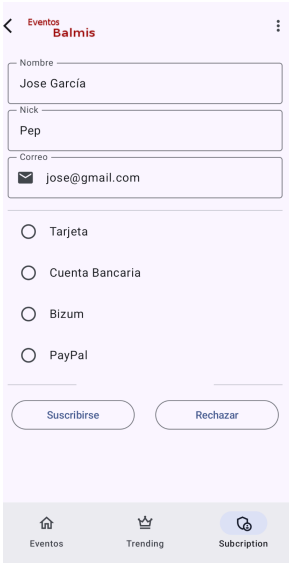
## Aplicación

Vamos a modificar la app del examen de la primera evaluación para que sea más real, recuerda que era una aplicación para visualizar los eventos que iban a ocurrir en la ciudad y poder inscribirse en ellos. Ahora la app constará de:

Una pantalla principal `EventosScreen.kt` que dependiendo de la navegación podrá mostrar distintas pantallas:




Como se ve en la imágenes de ejemplo, contendrá un `TopAppBar` con un botón de navegación hacia atrás, el logo, un icono de búsqueda y un menú desplegable `DropdownMenu`

Esta pantalla también contendrá una `NavigationBar` con tres opciones:

- `Eventos` : Que mostrará una lista de navegación lateral `LazyRow` con todos los eventos que actualmente están en cartelera y que además es la pantalla inicial.
- `Trending` : Que mostrará una lista con los eventos que tienen más seguidores (en este caso solo tres por cuestiones prácticas).
- `Subscription` : Que mostrará una pantalla que permitirá al usuario suscribirse a la página, lo que hará que el precio de los eventos sea menor.

> **Criterios de calificación generales (Penalizaciones)**:
>
> (❌ **-0,5 pts**) Si el proyecto entregado tiene errores de compilación que impiden su ejecución.
> (❌ **-0,25 pts**) Por cada fuente, clase, función, variable, etc. en nuestra implementación cuyo nombre no tenga nada que ver con la aplicación que estamos implementando de **StockFarmacia** resultado del copy-paste. Como por ejemplo: tienda, agenda, recetas, etc. Incluso si el funcionamiento es correcto.
> (❌ **-0,25 pts**) Por cada clase, función o definición innecesaria para la resolución del problema y no descrita en las especificaciones, salvo las que se proporcionan inicialmente.

### Parte 1 (2 pts)

Crea un BD con Room para almacenar todos los datos de nuestro modelo denominada "***eventos.db***". La BD debe contener **una única tabla** denominada ***eventos*** donde la clave primaria será el campo ***id*** de tipo ***Int*** que tienes en el modelo. Los campos de las tabla deben ir en `snake_casing` y los puedes deducir del modelo.

> 📌 **Nota**: Además puedes ver que hay otro paquete en Mock que se llama **suscripcion**, todo lo referente a suscripción se tratará directamente como datos mocks, no tendremos que crear la tabla en la base de datos.

Las operaciones en `EventoDao` serán todas asíncronas y debes definir las siguientes:

- Operaciones básicas `insert`, `update` y `count`
- `get()`: Devuelve todos los eventos de la base de datos.
- `get(Int)`: Devuelve un evento de la base de datos a partir de su id.

Prepara los '*providers*' que consideres necesarios de las instancias de Room para la inyección de dependencias con **Hilt**. Teniendo en cuenta que se creará una **instancia única** de cada uno de ellos.

Define el `EventoRespository` para usar los métodos definidos. Las operaciones en `EventoRepository` serán las mismas que en `EventoDao` más:

- `getSuscriptores()`: Devuelve la lista con todos los subscriptores que se obtendrá directamente del `SuscripcionDaoMock`.

> 📌 **Nota**: Dispones de `EventoRepositoryConverters.kt` con algunas definiciones en el paquete `.data`.

Por último, realiza la primera carga de datos de la base de datos de Room a partir de los datos `EventoDaoMock`. **Solo si la BD está vacía**. Para ello, realizarás la carga de datos en la BD al iniciar la aplicación, utilizando el método `onCreate` de la clase `Aplicacion` e inyectando los repositorios con Hilt.

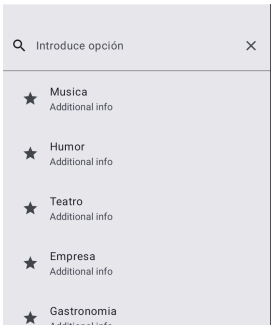
> **Criterios de calificación:**
>
> (✅ **1,25 pts**) Definir persistencia con Room, clases y métodos descritos en la especificación.
> (✅ **0,25 pts**) Preparar '*providers*' necesarios de las instancias de Room para la inyección de dependencias con Hilt.
> (✅ **0,5 pts**) Realizar primera carga de datos de la base de datos de Room a partir de los datos `EventoDaoMock`. **Solo si la BD está vacía**.

## Parte 2 (2 pts)

En el código base podrás ver que se pasa el fichero `.ui.features.presentacion_eventos.EventoScreen.kt` con el método `EventosScreen` con el siguiente interfaz:

```
@Composable
fun EventosScreen(
    query: String,
    suggestion: List<String>,
    onQueryChange: (String) -> Unit,
    suscrito: Boolean,
    onSuscrito: (Boolean) -> Unit,
    navController: NavHostController = rememberNavController(),
)
```

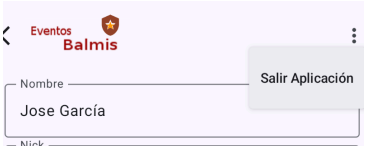
Dispones de la clases `EventoUiState` y `EventoUiConverte` que te ayudarán a implementar la clase **EventoViewModel** que se encargará de la gestión de las filtraciones con los parámetros query, suggestion y onQueryChange, para ello se pasa el componente `BaraDeBusqueda` en el paquete composable.




> **Importante**
>
> Para implementar la barra de filtro se tendrá que controlar si se ha pulsado el icono de la lupa en la `TopAppBar`, en el caso de que se encuentre pulsado se ocultará la `TopAppBar` y se mostrará la BarraDeBúsqueda en el content del Scaffold (cuando se cierre la BarraDeBúsqueda se mostrará la `TopAppBar`). Suggestion contendrá la lista del tipo de espectáculos, que puede ser como la siguiente:
> `var suggestion: List<String> = listOf("Musica", "Humor", "Teatro", "Empresa", "Gastronomia")` que serán fijos.

Deberás de completar `EventosScreen` para que emita un `Scaffold` que contendrá:

1. Una `BarraSuperior` con un Logo que se pasa ya implementado (en el que se mostrará una pequeña imagen en el caso de que el usuario esté suscrito a la página), un icono de navegación hacia atrás, el icono de búsqueda y el de menú desplegable que también se te pasan implementados. El **menú desplegable** solo tendrá la opción de salir de la app y **es lo único que tendrás que añadir a la implementación ya dada**.




2. Como contenido principal, tendrá un `NavHost` que permitirá navegar por las tres pantalla, la que muestra todos los eventos, la que muestra solamente los tres eventos con más seguidores y la que muestra el formulario de inscripción.

3. Una `BottomAppBar` con un NavigationBar que nos permita la navegación principal.

Por último, debes definir utilizando inyección de dependencias con Hilt la clase `EventoViewModel` para que contenga la lógica de negocio de la pantalla y que se encargue de gestionar los eventos que se producen en la misma comunicándose con el repositorio.

> **Criterios de calificación:**
>
> (✅ **1 pts**) Definición del `Scaffold` con la estructura especificada que muestre la `TopAppBar` y la `NavigationBar`.
> (✅ **0,5 pts**) Filtrado de los eventos usando el componente `BarraDeBusqueda` pasado como recurso.
> (✅ **0,5 pts**) Menú desplegable que se muestre y permita salir de la App.

### Parte 3 (1,5 pts)

Completa la definición de las pantallas:

- `ListadoEventoScreen.kt` que se te proporciona, con un `LazyRow` y descomenta el ViewModel de los recursos para que una vez se haya implementado correctamente el EventoRepository con Hilt funcione de forma que se muestren todos los eventos al hacer scroll lateral y funcione correctamente la pulsación sobre el corazón. Si se ha implementado el Icono de búsqueda de la `TopAppBar`, se podrá visualizar un filtrado por tipo de evento.
- `TrendigSreen` completa esta función para que se visualice la lista de los tres elementos con más seguidores, además permitirá navegar a la pantalla `InformacionTrendingScreen` del elemento pulsado.

> **Criterios de calificación:**
>
> (✅ **0,5 pts**)Implementación de `ListadoEventoScreen` con el funcionamiento correcto después de **descomentar el código del ViewModel**.
> (✅ **0,5 pts**) Funcionalidad que permita mostrar los tres elementos com más seguidores, para ello se creará la clase `TrendingScreenViewModel`, pantalla.
> (✅ **0,5 pts**) Navegación y visualización de la información completa del evento que se ha pulsado.

### Parte 4 (2,5 pts)

Completa la definición de la pantalla:

- `SuscripcionScreen.kt` con el siguiente interfaz:

```
fun SuscripcionScreen(
    suscripcionUiState: SuscripcionUiState,
    validacionSuscripcionUiState: ValidacionSuscripcionUiState,
    onSuscrito: (Boolean) -> Unit,
    onSuscripcionEvent: (SuscripcionEvent) -> Unit
)
```

Y que deberás construir desde el inicio, pero de la que se te proporciona el código del componente `RadioButtonPago` que está en composables. Se deberá validar el correo con la entrada de datos, para ello se proporciona como recurso las clases de validación la interfaz sellada de los eventos y la clase `SuscripcionUiState`. Tendrás que tener en cuenta las siguientes especificaciones:

- Si no se ha seleccionado forma de pago, no dejará realizar la suscripción (no pasará nada al pulsar el botón).
- Al suscribirse/desuscribirse haremos 2 cosas:
    i. Llamar al callback `onSuscripcionEvent` con el evento correspondiente que modificará la propiedad adecuada en `suscripcionUiState`.

ii. Se mostrará el icono de suscripción al lado del logo de la AppBar para ello deberás llamar al callback `onSuscrito` que llegará desde `EventoScreen` y que gestionará su `ViewModel`.


Al desuscribirnos **se ocultará el icono**.

- El correo se deberá validar de forma correcta.
- Al estar suscrito, se mostrarán superpuestos los precios en `ListadoEventosScreen` con un **20%** de descuento.


> **Criterios de calificación:**
>
> (✅ **1 pts**) Definición de la clase `SuscripcionScreen` de forma correcta.
> (✅ **1 pts**) Definición del `ViewModel` para el control de las entradas de datos y los eventos correspondientes.
> (✅ **0,5 pts**) funcionamiento correcto del icono y del descuento del **20%** al estar suscrito.

## Parte 5 (2 pts)

Define en el paquete `.ui.navegation` la navegación entre:

- Las tres pantallas de navegación que se controlan a través de los eventos de la barra de navegación inferior.
- La navegación desde un item de Trending a `InformacionTrendingScreen`
- Navegar hacía atrás cuando se pulsa el icono dispuesto para ello en la AppBar **que ya está implementado**.

> **Criterios de calificación:**
>
> (✅ **0,5 pts**) Sigue los convenios de nombres y ficheros explicados en clase.
> (✅ **1,5 pts**) Correcto funcionamiento de la navegación entre pantallas según las especificaciones.