



2025



“PROYECTO DE SEGUNDO PARCIAL”

CARRERA:

Ingeniería Informática

MATERIA:

Desarrollo de Aplicaciones Móviles

NOMBRE DE LOS ESTUDIANTES:

Juan José Hernández Olmos BS19240189

Juan Antonio Pérez Cabrera CS20241007

Pablo Dayan Trejo Solis RS21110534

DOCENTE:

Jair Emmanuel Ramírez Flores

Purísima del Rincón

7/11/2025





2025

Contenido

Lista de Cotejo	3
Problemática	4
Desarrollo paso a paso.....	5
Resultados	31
Conclusión.....	36





2025

Lista de Cotejo

Requisito	Descripción	Valor
Autenticación de Usuario	<ul style="list-style-type: none">Registro de nuevos usuarios funcional (10 puntos)Inicio de sesión de usuarios existentes funcional (10 puntos)	20
CRUD Completo de Películas	<ul style="list-style-type: none">Formulario funcional para agregar nuevas películas con los campos correctos (8 puntos)Funcionalidad para editar una película existente correctamente implementada (7 puntos)Capacidad para eliminar películas del catálogo de forma efectiva (7 puntos)Pantalla de visualización de detalles de una película completa (título, sinopsis, imagen, calificación, reseñas) (8 puntos)	30
Filtrado y Búsqueda de Películas	<ul style="list-style-type: none">Implementación de búsqueda funcional por título o calificación (7 puntos)Opciones de ordenamiento por calificación o alfabéticamente (8 puntos)	15
Persistencia de Datos	<ul style="list-style-type: none">Datos de las películas persisten después de cerrar y abrir la aplicación (local o AsyncStorage) (15 puntos)	15
Navegación Completa	<ul style="list-style-type: none">Uso adecuado de Navegación para gestionar las pantallas (5 puntos)Navegación funcional entre las distintas pantallas, con estructura clara y lógica (5 puntos)	10
Interfaz de Usuario	<ul style="list-style-type: none">Diseño de la UI es moderno, responsive y funciona correctamente en Android (5 puntos)Uso de librerías de extra para algún apartado de diseño (5 puntos)	10

Juan Jose Hernandez Olmos *JJH*
Juan Antonio Pérez Cabrera *Antonio Pérez*
Pablo Dayan Trejo SOLIS PABLO DAYAN





2025

Problemática

Desarrollar una aplicación móvil en Kotlin que permita gestionar un catálogo de películas, donde los usuarios puedan registrarse, iniciar sesión, agregar, editar, eliminar y visualizar películas, además de filtrarlas por criterios como título o calificación. Esta solución busca ofrecer una experiencia funcional, segura y visualmente atractiva, aplicando conceptos fundamentales del desarrollo Android, arquitectura MVVM, y el uso de componentes como Room, Navigation y ViewModel.

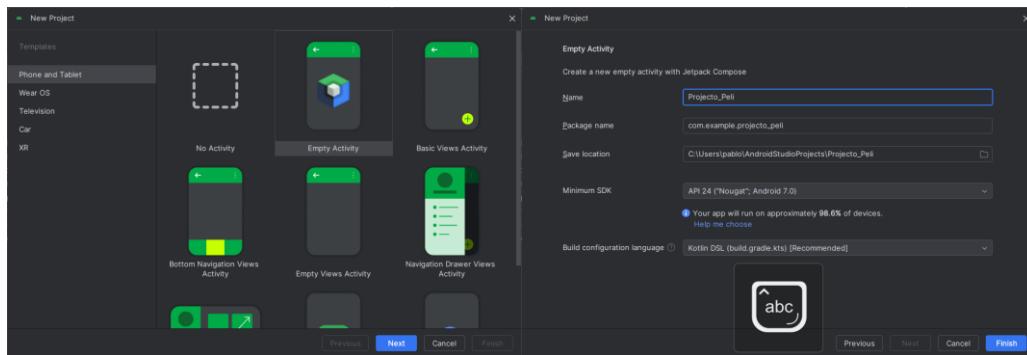




2025

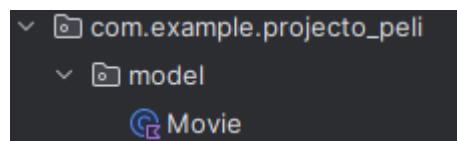
Desarrollo paso a paso

Primero creamos un nuevo proyecto de tipo Empty activity que se llamará Projecto_peli

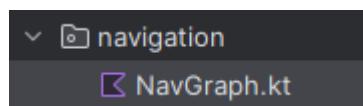


Ya que se creo el nuevo proyecto vamos a crear diferentes package y class file

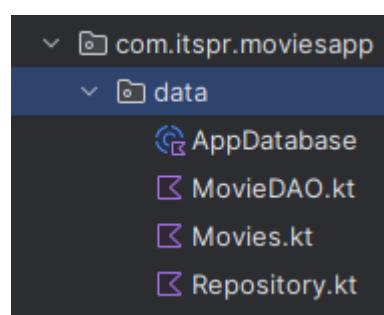
Primer package va a ser model y la class va a ser movie



Segundo package navigation y su class file va a ser NavGraph.kt



La otra va ser una carpeta de data que contiene la lógica de la app



en el UI va estar el package screens y va tener diferentes class file

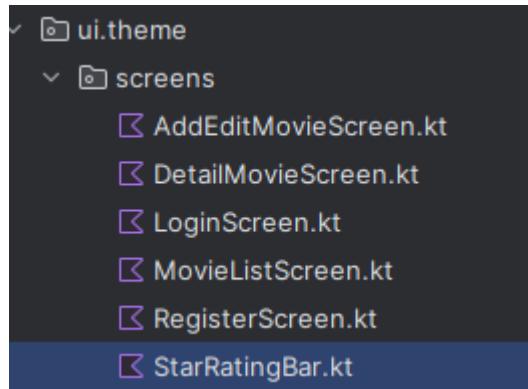
- addEditmoviescreen.kt
- detailmoviescreen.kt
- loginscreen.kt
- movielistscreen.kt



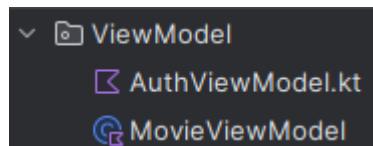


2025

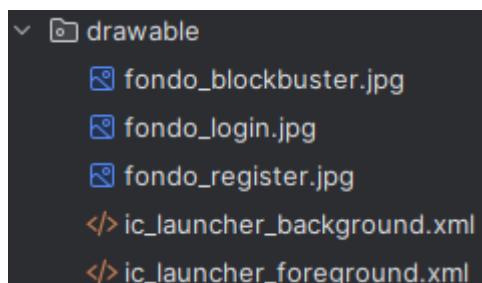
- registerscreen.kt
- starRatingbar.kt



Se crea otro package que es viewModel donde va a tener la class movieviewmodel



En el drawable tenemos los fondos que vamos a ocupar para la aplicación



Ya que tenemos eso ahora nos vamos al buildgradle.kts es para agregar las dependencias para room.





2025

MovieDao (interface)

Creamos una interface llamada movieDao la cual contendrá las consultas SQL que se usaran para manipular la base de datos

```
package com.itspr.moviesapp.data

import androidx.room.*
import kotlinx.coroutines.flow.Flow

@Dao
interface MovieDao {
    //funcion para busqueda por titulo
    @Query("SELECT * FROM movies WHERE title LIKE '%' || :query || '%'")
    fun searchByTitle(query: String): Flow<List<Movie>>

    //funcion para ordenar desc por calificacion
    @Query("SELECT * FROM movies ORDER BY rating DESC")
    fun getAllByRatingDesc(): Flow<List<Movie>>

    //funcion para ordenar asc por calificacion
    @Query("SELECT * FROM movies ORDER BY rating ASC")
    fun getAllByRatingAsc(): Flow<List<Movie>>

    //funcion para vista normal, ver todas las peliculas
    @Query("SELECT * FROM movies ORDER BY title ASC")
    fun getAll(): Flow<List<Movie>>

    //insercion de pelicula nueva
    @Insert
    suspend fun insert(movie: Movie)

    //actualizar al editar una pelicula
    @Update
    suspend fun update(movie: Movie)

    //eliminar una pelicula
    @Delete
    suspend fun delete(movie: Movie)
}
```

Movies.k (data class)

Creamos una data class donde declaramos la entidad y los atributos de nuestra tabla:

```
package com.itspr.moviesapp.data

import androidx.room.Entity
import androidx.room.PrimaryKey

//entity indica que es entidad de room
@Entity(tableName = "movies")//tabla movies
```





2025

```
data class Movie( //declarar los atributos de la entidad
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val title: String,
    val synopsis: String,
    val rating: Float,
    val review: String,
    val imageUri: String
)
```

AppDatabase.kt (kotlin class)

Creamos un abstract class donde definimos la base de datos con room que usaremos, delcaramos una tabla (entidad) llamda movies

```
//indica tabla de tipo Movie
@Database(entities = [Movie::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    abstract fun movieDao(): MovieDao //muestra el movieDAO para las
operaciones CRUD
}
```

Repository.kt (kotlin class)

Finalmente creamos una clase llamada Repository en la cual delegamos las funciones de SQL como inserción, actualización y eliminación

```
package com.itspr.moviesapp.data

import kotlinx.coroutines.flow.Flow

class MovieRepository(private val movieDao: MovieDao) {
    val allMovies: Flow<List<Movie>> = movieDao.getAll()
        //mostrar la lista completa, accede a la función getAll de movieDao

    //funciones de inserción, edición y eliminación de movies, acceden a
    las funciones del movieDAO
    suspend fun insert(movie: Movie) = movieDao.insert(movie)
    suspend fun update(movie: Movie) = movieDao.update(movie)
    suspend fun delete(movie: Movie) = movieDao.delete(movie)
}
```





2025

Conectar la aplicación Android con los servicios de Room en el gradle.kts

```
// Room
implementation(libs.androidx.room.runtime)
implementation(libs.androidx.room.ktx)
kapt(libs.androidx.room.compiler)
```

LoginScreen.kt

Se crea el login con el parámetro navController para navegar a la pantalla principal cuando se validen las credenciales. Se crean las variables de usuario y contraseña y se crea un mapa de credenciales locales.

```
@Composable
fun LoginScreen(navController: NavController) {
    //variables para ingresar datos
    var user by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    val context = LocalContext.current

    //crendiciales locales
    val credenciales = mapOf(
        "Antonio" to "123",
        "qq" to "11",
        "Juan" to "456",
        "Pablo" to "789"
    )
```

Se coloca una imagen de fondo en la pantalla de login

```
Box(modifier = Modifier) {
    Image("//imagen para fondo del login
        painter = painterResource(id =
com.itspr.moviesapp.R.drawable.cine2_bg), //ubicacion
        contentDescription = "fondo_login",
        modifier = Modifier.fillMaxSize(),
        contentScale = ContentScale.Crop,
        alpha = 0.15f //nitidez del fondo
    )
```

Se coloca un título para el login y una imagen como tipo logotipo

```
Column(
    modifier = Modifier
        .padding(16.dp)
        .fillMaxSize(),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Top
) { //título de la pantalla
    Spacer(modifier = Modifier.height(64.dp))
    Text(text = "BIENVENIDO AL", fontSize = 25.sp, color =
Color(0xFF000000))
```

Bvd. del Valle #2301, Col. Guardarrayas; Purísima del Rincón, Guanajuato, C.P.36413

Tel. (476) 744 71 00 e-mail: dirección@tecpurisima.edu.mx

www.tecpurisima.edu.mx





2025

```
Text(text = "CATALOGO DE PELICULAS", fontSize = 25.sp, color =
Color(0xFF000000))
Spacer(modifier = Modifier.height(16.dp))

//imagen o logo/
Image(
    painter = painterResource(id = R.drawable.bb_lg), //ubicacion
    contentDescription = "Foto",
    modifier = Modifier //dimensiones de la foto
        .size(150.dp)
        .padding(end = 10.dp)
)
Spacer(modifier = Modifier.height(56.dp))
```

Creamos los textos de ampo donde se ingresaran las credenciales

```
Text(text = "Inicia sesión", fontSize = 20.sp, color = Color(0xFF000000))
Spacer(modifier = Modifier.height(16.dp))

//campo de texto para que ingrese usuario
TextField(
    value = user,
    onValueChange = { user = it },
    label = { Text("Ingresa tu usuario") }
)
Spacer(modifier = Modifier.height(38.dp))

//campo de texto para que ingrese contraseña
TextField(
    value = password,
    onValueChange = { password = it },
    label = { Text("Ingresa tu contraseña") },
    visualTransformation = PasswordVisualTransformation()

)
```

Creamos el botón que valida las credenciales y da acceso a la pantalla principal
En caso de que sean validas, en caso contrario arroja un mensaje de error.

```
Spacer(modifier = Modifier.height(30.dp))
Button(onClick = { //valida si coinciden las credenciales
    val validacion = credenciales[user] == password
    if (validacion) { //si valida manda a la pabtalla principal
        navController.navigate("movie_list")
    } else { //mensaje si no valida
        Toast.makeText(context, "Usuario o contraseña incorrecto",
        Toast.LENGTH_SHORT).show()
    }
}) {
    Text(text = "Iniciar")
}
```





2025

Esto garantiza encapsulamiento: nadie fuera del repositorio puede modificar los datos directamente. En Compose, cuando un StateFlow cambia, la interfaz se recomponen automáticamente.

```
private val _movies = MutableStateFlow<List<Movie>>(emptyList())
val allMovies: StateFlow<List<Movie>> get() = _movies
```

el contador ID Se usa para asignar un ID único a cada película nueva (simulando un autoincremento de base de datos).

```
private var nextId = 1
```

para agregar una película crea una nueva película:

- Copia el objeto recibido y le asigna un ID único.
- Agrega esa película a la lista (_movies.value + newMovie). Usa suspend porque, en el futuro, podría integrar Firebase o Room sin cambiar la estructura del código.

```
suspend fun insert(movie: Movie) {
    val newMovie = movie.copy(id = nextId++)
    _movies.value = _movies.value + newMovie
}
```

Para actualizar una película recorre la lista y reemplaza la película que tenga el mismo id. Esto actualiza solo los datos de esa película sin alterar las demás.

```
suspend fun update(movie: Movie) {
    _movies.value = _movies.value.map { if (it.id == movie.id) movie else
it }
}
```

Eliminar una película filtra la lista, quitando la película con id coincide con la que se desea eliminar.

```
suspend fun delete(movie: Movie) {
    _movies.value = _movies.value.filter { it.id != movie.id }
}
```

Define el paquete donde se encuentra el view model, que almacena toda la lógica de acceso a datos.

```
package com.example.projecto_peli.ViewModel
```

En las importaciones la import android.util.Patterns Se usa para validar el formato del correo electrónico.

import androidx.lifecycle.ViewModel Permite declarar la clase como un ViewModel de Android:





2025

import androidx.lifecycle.viewModelScope Agrega el alcance de corutinas (coroutine scope) dentro del ViewModel se puede ejecutar tareas asíncronas sin bloquear la interfaz:

import kotlinx.coroutines.flow.MutableStateFlow Permite crear un flujo de datos reactivo y mutable que guarda el estado actual de la interfaz.

import kotlinx.coroutines.flow.StateFlow Es la versión inmutable del flujo anterior.

import kotlinx.coroutines.flow.asStateFlow Convierte un MutableStateFlow en un StateFlow solo de lectura.

import kotlinx.coroutines.launch Permite ejecutar funciones asíncronas dentro de viewModelScope.

```
import android.util.Patterns
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.projecto_peli.repository.AuthRepository
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
```

MovieViewModel

Ahora nos vamos al apartado movieviewmodel.kt

se importan los módulos necesarios:

- androidx.lifecycle.ViewModel base del patrón MVVM.
- androidx.compose.runtime.* para variables reactivas que actualizan la UI automáticamente.
- viewModelScope para ejecutar tareas asíncronicas sin bloquear la interfaz.
- kotlinx.coroutines.flow.* para manejar flujos reactivos (listas de películas, filtros, etc.).
- Movie y MovieRepository tus clases del modelo y repositorio.

```
package com.example.projecto_peli.ViewModel

import android.net.Uri
import androidx.compose.runtime.*
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.projecto_peli.model.Movie
```





2025

```
import com.example.projecto_peli.repository.MovieRepository
import kotlinx.coroutines.flow.SharingStarted
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.combine
import kotlinx.coroutines.flow.stateIn
import kotlinx.coroutines.launch
```

la clase principal movieviewmodel se encarga de toda la lógica de películas:

Controla los formularios de creación y edición.

Se comunica con el MovieRepository (que simula la base de datos).

Expone flujos (StateFlow y MutableState) que la UI observa.

```
class MovieViewModel(private val repository: MovieRepository) :  
    ViewModel() {
```

Las variables de estado de formulario representan los campos del formulario actual (cuando el usuario crea o edita una película).

Si el usuario selecciona una película, los valores se llenan automáticamente.

Si está agregando una nueva, se limpian con clearForm().

```
var currentMovie = mutableStateOf<Movie?>(null)
var title = mutableStateOf("")
var synopsis = mutableStateOf("")
var rating = mutableStateOf(3.0f)
var review = mutableStateOf("")
var imageUri = mutableStateOf<Uri?>(null)
    private set
```

En el manejo de la imagen actualiza el URI de la imagen seleccionada desde la galería o cámara.

```
fun updateImageUri(uri: Uri?) {
    imageUri.value = uri
}
```

Para cargar o limpiar la imagen setMovie(movie): Si recibe una película existente, llena el formulario con sus datos. Si recibe null, limpia el formulario.

clearForm(): Limpia todos los campos y deja listo el formulario para agregar una nueva película.

```
fun setMovie(movie: Movie?) {
fun clearForm() {
```





2025

Guardar o actualizar una película Esta función Crea una nueva instancia de Movie con los valores del formulario. Si se está editando, usa copy() para mantener el mismo id.

Si no hay una película actual (currentMovie == null) inserta una nueva.

Si ya hay una (currentMovie != null) la actualiza. Luego limpia el formulario. Todo se ejecuta dentro de una corutina (viewModelScope.launch), lo que permite mantener la interfaz fluida.

```
fun setMovie(movie: Movie?) {
    currentMovie.value = movie
    if (movie != null) {
        title.value = movie.title
        synopsis.value = movie.synopsis
        rating.value = movie.rating
        review.value = movie.review
        imageUri.value = if (movie.imageUri.isNotEmpty())
            Uri.parse(movie.imageUri) else null
    } else {
        clearForm()
    }
}

fun clearForm() {
    title.value = ""
    synopsis.value = ""
    rating.value = 3.0f
    review.value = ""
    imageUri.value = null
    currentMovie.value = null
}

fun saveMovie() {
    viewModelScope.launch {
        val movie = currentMovie.value?.copy(
            title = title.value,
            synopsis = synopsis.value,
            rating = rating.value,
            review = review.value,
            imageUri = imageUri.value?.toString() ?: ""
        ) ?: Movie(
            title = title.value,
            synopsis = synopsis.value,
            rating = rating.value,
            review = review.value,
            imageUri = imageUri.value?.toString() ?: ""
        )

        if (currentMovie.value == null) repository.insert(movie)
        else repository.update(movie)
    }
}
```

Blvd. del Valle #2301, Col. Guardarrayas; Purísima del Rincón, Guanajuato, C.P.36413

Tel. (476) 744 71 00 e-mail: dirección@tecpurisima.edu.mx

www.tecpurisima.edu.mx





2025

```
        clearForm()
    }
}
```

Eliminar película Llama al repositorio para eliminar la película seleccionada.

```
fun deleteMovie(movie: Movie) {
    viewModelScope.launch { repository.delete(movie) }
}
```

La búsqueda y ordenamiento Variables internas de control: Sirven para:

Buscar películas por título (_searchQuery).

Cambiar el orden por calificación (_sortByRating).

Alternar entre ascendente/descendente (_sortDescending).

```
private val _searchQuery = mutableStateOf("")
val searchQuery: State<String> get() = _searchQuery

private val _sortByRating = mutableStateOf(false)
val sortByRating: State<Boolean> get() = _sortByRating

private val _sortDescending = mutableStateOf(true)
val sortDescending: State<Boolean> get() = _sortDescending
```

Lista reactiva de películas

Combina cuatro flujos (combine()):

- Todas las películas del repositorio.
- El texto de búsqueda.
- Las opciones de ordenamiento.
- Filtra y ordena las películas según esos valores.

Expone el resultado como un StateFlow que la UI puede observar en tiempo real.

```
val movies: StateFlow<List<Movie>> = combine(
    repository.allMovies,
    snapshotFlow { _searchQuery.value },
    snapshotFlow { _sortByRating.value },
    snapshotFlow { _sortDescending.value }
) { allMovies, query, byRating, descending ->
    var filtered = allMovies
    if (query.isNotBlank()) filtered = filtered.filter {
        it.title.contains(query, ignoreCase = true) }
    filtered = when {
        byRating -> if (descending) filtered.sortedByDescending { it.rating } else filtered.sortedBy { it.rating }
    }
}
```





2025

```
        else -> filtered.sortedBy { it.title }
    }
    filtered
}.stateIn(
    scope = viewModelScope,
    started = SharingStarted.WhileSubscribed(5000),
    initialValue = emptyList()
```

Las funciones de actualización Permiten a la interfaz (por ejemplo, botones o campos de texto) modificar los filtros y el orden de manera reactiva.

```
fun updateSearchQuery(query: String) { _searchQuery.value = query }
fun toggleSortByRating() { _sortByRating.value = !_sortByRating.value }
fun toggleSortDirection() { _sortDescending.value
= !_sortDescending.value }
```

AddEditMovieScreen.kt

Ahora nos vamos al package screens el primero AddEditMovieScreen.kt en las importaciones Se importan los componentes de Compose para layouts, scroll, botones, iconos y texto.

Se importa NavController para manejar la navegación entre pantallas.

AsyncImage de Coil sirve para mostrar imágenes cargadas desde una URI o URL.

Se importa tu MovieViewModel que contiene la lógica de la película (datos, guardar, etc.).

```
package com.example.projecto_peli.ui.theme.screens

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons(Icons)
import androidx.compose.material.icons.filled.ArrowBack
import androidx.compose.material.icons.filled.Image
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import coil.compose.AsyncImage
import com.example.projecto_peli.ViewModel.MovieViewModel
```





2025

La función composable `@Composable` indica que esta función dibuja UI declarativa. `@OptIn(ExperimentalMaterial3Api::class)` permite usar componentes experimentales de Material3.

Parámetros:

- `navController`: permite regresar o navegar a otras pantallas.
- `movieId`: si es null, la película es nueva; si tiene valor, es edición.
- `onPickImage`: función que se ejecuta al presionar “Seleccionar Imagen”.
- `viewModel`: contiene los datos de la película y la lógica de negocio.

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AddEditMovieScreen(
    navController: NavController,
    movieId: String?,
    onPickImage: () -> Unit,
    viewModel: MovieViewModel
```

La lectura de datos del ViewModel Se obtienen los valores actuales de la película desde el ViewModel.

by y `mutableStateOf` permiten que Compose observe los cambios y actualice la UI automáticamente.

El rating e `imageUri` se envuelven en `remember` para mantener el estado local mientras la pantalla esté activa.

```
{
    val title by viewModel.title
    val synopsis by viewModel.synopsis
    val review by viewModel.review
    val rating by remember { mutableStateOf(viewModel.rating.value) }
    val imageUri by remember { mutableStateOf(viewModel.imageUri.value) }
```

Scaffold es un contenedor base que facilita un layout con TopBar, BottomBar, etc.

CenterAlignedTopAppBar es la barra superior con título y botón de regreso.

`navController.popBackStack()` regresa a la pantalla anterior.

```
Scaffold(
    topBar = {
        CenterAlignedTopAppBar(
            title = { Text(if (viewModel.currentMovie.value == null)
"Nueva Película" else "Editar Película") },
            navigationIcon = {
```

Bvd. del Valle #2301, Col. Guardarrayas; Purísima del Rincón, Guanajuato, C.P.36413

Tel. (476) 744 71 00 e-mail: dirección@tecpurisima.edu.mx

www.tecpurisima.edu.mx





2025

```
    IconButton(onClick = { navController.popBackStack() }) {
        Icon(Icons.Filled.ArrowBack, contentDescription =
    "Volver")
    }
}
}
```

Layout principal (Column scrollable) Usamos Column para apilar elementos verticalmente.

El verticalScroll permite hacer scroll si hay muchos elementos.

El padding agrega espacio alrededor de los componentes.

```
Column(modifier =
Modifier.padding(padding).padding(16.dp).fillMaxSize().verticalScroll(rem
emberScrollState()))
```

Los campos de texto OutlinedTextField crea cajas de texto con borde.

onValueChange actualiza el valor en el ViewModel cuando el usuario escribe.

minLines asegura que la caja sea más grande para textos largos.

```
OutlinedTextField(value = title, onValueChange = { viewModel.title.value =
it }, label = { Text("Título") }, modifier = Modifier.fillMaxWidth())
Spacer(Modifier.height(8.dp))
OutlinedTextField(value = synopsis, onValueChange = {
viewModel.synopsis.value = it }, label = { Text("Sinopsis") }, minLines =
3, modifier = Modifier.fillMaxWidth())
Spacer(Modifier.height(8.dp))
Spacer(Modifier.height(8.dp))
OutlinedTextField(value = review, onValueChange = {
viewModel.review.value = it }, label = { Text("Reseña") }, minLines = 3,
modifier = Modifier.fillMaxWidth())
Spacer(Modifier.height(16.dp))
```

Calificación con estrellas Se muestra el valor actual de la calificación.

StarRatingBar es un componente personalizado (no incluido en el snippet) que permite elegir la calificación con estrellas.

onRatingChanged actualiza la calificación en el ViewModel.

```
Text("Calificación: ${viewModel.rating.value.toInt()}")
StarRatingBar(rating = viewModel.rating.value, onRatingChanged = {
viewModel.rating.value = it})
```





2025

Botón para seleccionar imagen Llama a la función onPickImage que debería abrir la galería o cámara. Muestra un ícono de imagen y el texto “Seleccionar Imagen”.

```
Button(onClick = onPickImage, modifier = Modifier.fillMaxWidth()) {  
    Icon(Icons.Filled.Image, "Imagen")  
    Spacer(modifier.width(8.dp))  
    Text("Seleccionar Imagen")
```

Mostrar la imagen seleccionada Si imageUrl no es null, se muestra la imagen usando Coil. AsyncImage carga la imagen de forma asíncrona desde URI o URL

```
imageUri?.let {  
    AsyncImage(model = it, contentDescription = null, modifier =  
    Modifier.fillMaxWidth().height(200.dp).padding(8.dp))
```

Botón guardar Llama al método saveMovie() del ViewModel para guardar la película en la base de datos o repositorio. Regresa a la pantalla anterior después de guardar.

```
Spacer(modifier.height(16.dp))  
Button(onClick = {  
    viewModel.saveMovie()  
    navController.popBackStack()  
, modifier = Modifier.fillMaxWidth()) { Text("Guardar") }
```

DetailsMovieScreen.kt

Ahora nos vamos a la siguiente que es DetailMovieScreen.kt

En las importaciones Importa componentes de Compose, AsyncImage de Coil para mostrar imágenes, y MovieViewModel para acceder a los datos.

Icons.Default.Movie se usa como placeholder y icono de error si la imagen no carga.

```
package com.example.projecto_peli.ui.theme.screens  
  
import androidx.compose.foundation.layout.*  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.filled.Movie  
import androidx.compose.material3.*  
import androidx.compose.runtime.*  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.graphics.vector.rememberVectorPainter  
import androidx.compose.ui.unit.dp  
import androidx.navigation.NavController  
import coil.compose.AsyncImage  
import com.example.projecto_peli.ViewModel.MovieViewModel
```





2025

Declaración de la función composable navController: permite volver a la pantalla anterior.

- movield: id de la película que queremos mostrar; puede ser null.
- viewModel: contiene la lista de películas y lógica de negocio.

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun DetailMovieScreen(
    navController: NavController,
    movieId: String?,
    viewModel: MovieViewModel
```

Obtención de la película seleccionada movies se obtiene como un StateFlow del ViewModel y se convierte en un estado observable con collectAsState().

movield?.toIntOrNull() convierte el id a entero si es posible.

movies.find { it.id == id } busca la película con ese id en la lista.

```
val movies by viewModel.movies.collectAsState()
val movie = movieId?.toIntOrNull()?.let { id ->
    movies.find { it.id == id }
```

Comprobación de existencia Si movie es null, se muestra un mensaje: "Película no encontrada". Si existe, se renderizan los detalles de la película.

```
? : Box(
    modifier = Modifier.fillMaxSize(),
    contentAlignment = androidx.compose.ui.Alignment.Center
) {
    Text("Película no encontrada", style =
MaterialTheme.typography.bodyLarge)
```

Scaffold y TopAppBar Scaffold organiza la pantalla.

TopAppBar muestra el título de la película y un botón para volver a la pantalla anterior.

```
movie?.let { selectedMovie ->
    Scaffold(
        topBar = {
            CenterAlignedTopAppBar(
                title = { Text(selectedMovie.title) },
                navigationIcon = {
                    IconButton(onClick = { navController.popBackStack() })
                }
    ) {
        Icon(
```





2025

```
Icons.Default.Movie,  
contentDescription = "Volver"  
)
```

Layout principal (Column) Usa Column para apilar los elementos verticalmente.
padding agrega espacio alrededor de los elementos.

```
{ padding ->  
    Column(  
        modifier = Modifier  
            .padding(padding)  
            .padding(16.dp)
```

Imagen de la película AsyncImage carga la imagen desde la URI.

Si no hay imagen (imageUri vacío) o falla la carga, se usa el ícono de película como placeholder.

fillMaxWidth() hace que ocupe todo el ancho, height(300.dp) define la altura.

```
AsyncImage(  
    model = if (selectedMovie.imageUri.isNotEmpty())  
selectedMovie.imageUri else null,  
    contentDescription = selectedMovie.title,  
    modifier = Modifier  
        .fillMaxWidth()  
        .height(300.dp),  
    placeholder = rememberVectorPainter(Icons.Default.Movie),  
    error = rememberVectorPainter(Icons.Default.Movie)
```

Detalles de la película Muestra:

- Sinopsis de la película.
- Calificación con estrellas.
- Reseña del usuario o crítica.
- Spacer se usa para separar visualmente los elementos.

```
Spacer(Modifier.height(16.dp))  
  
Text("Sinopsis:", style = MaterialTheme.typography.titleMedium)  
Text(selectedMovie.synopsis)  
Spacer(Modifier.height(8.dp))  
  
Text("Calificación: ${selectedMovie.rating} ★")  
Spacer(Modifier.height(8.dp))  
  
Text("Reseña:", style = MaterialTheme.typography.titleMedium)  
Text(selectedMovie.review)
```





2025

Si no encuentra o no hay película, muestra un mensaje:

```
        }
    } ?: Text("Película no encontrada")
}
```

MovieListScreen.kt

Ahora nos vamos a movieListScreen.kt

En las importaciones se importan componentes Compose, iconos, layouts, y AsyncImage para mostrar imágenes.

NavController se usa para navegar entre pantallas.

MovieViewModel maneja la lista de películas, búsqueda y ordenamiento

```
package com.example.proyecto_peli.ui.theme.screens

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons(Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import coil.compose.AsyncImage
import com.example.proyecto_peli.ViewModel.MovieViewModel
import com.example.proyecto_peli.model.Movie
```

Declaración de la función componible

Parámetros:

navController: navegación entre pantallas.

viewModel: contiene datos, métodos de edición/eliminación y estado de búsqueda/orden.

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MovieListScreen(navController: NavController, viewModel:
MovieViewModel) {
```





2025

Se observan estados del ViewModel

- movies: lista de películas filtrada/ordenada.
- searchQuery: texto ingresado en la barra de búsqueda.
- sortByRating: indica si se ordena por calificación.
- sortDescending: indica si el orden es descendente.

```
val movies by viewModel.movies.collectAsState()
val searchQuery by viewModel.searchQuery
val sortByRating by viewModel.sortByRating
val sortDescending by viewModel.sortDescending
```

Scaffold y TopBar Text muestra el título de la app.

OutlinedTextField permite buscar películas, llamando a updateSearchQuery del ViewModel para actualizar la lista.

```
Scaffold(
    topBar = {
        Column {
            Spacer(Modifier.height(30.dp))
            Text("Cineteca Digital", textAlign = TextAlign.Center,
modifier = Modifier.padding(20.dp).fillMaxWidth())
            OutlinedTextField(
                value = searchQuery,
                label = { Text("Buscar") },
                onValueChange = viewModel::updateSearchQuery,
                modifier = Modifier.fillMaxWidth().padding(horizontal =
8.dp, vertical = 4.dp)
        )
    }
)
```

Filtros y ordenamiento Dos FilterChip permiten cambiar el criterio de orden: calificación o título.

IconButton cambia la dirección del orden: ascendente/descendente.

FloatingActionButton Botón flotante para agregar nueva película.

```
Row(
    modifier = Modifier.fillMaxWidth().padding(horizontal = 8.dp,
vertical = 4.dp),
    horizontalArrangement = Arrangement.SpaceBetween
) {
    FilterChip(
        selected = sortByRating,
        onClick = { viewModel.toggleSortByRating() },
        label = { Text("Calificación") },
        leadingIcon = {
            Icon(Icons.Filled.Star, null, tint = if (sortByRating)
MaterialTheme.colorScheme.primary else LocalContentColor.current)
        }
)
```

Bvd. del Valle #2301, Col. Guardarrayas; Purísima del Rincón, Guanajuato, C.P.36413

Tel. (476) 744 71 00 e-mail: dirección@tecpurisima.edu.mx

www.tecpurisima.edu.mx





2025

```
)  
    FilterChip(  
        selected = !sortByRating,  
        onClick = { viewModel.toggleSortByRating() },  
        label = { Text("Título") }  
    )  
    IconButton(onClick = { viewModel.toggleSortDirection() }) {  
        Icon(imageVector = if (sortDescending) Icons.Filled.ArrowDownward  
else Icons.Filled.ArrowUpward, contentDescription = "Orden")  
    }  
}
```

Llama a clearForm() para limpiar los campos y navega a AddEditMovieScreen.

```
floatingActionButton = {  
    FloatingActionButton(onClick = {  
        viewModel.clearForm()  
        navController.navigate("add_edit")  
    }) { Icon(Icons.Filled.Add, "Agregar") }  
}
```

Contenido principal: lista o mensaje vacío Si no hay películas, muestra un mensaje central.

Si hay películas, usa LazyColumn para mostrarlas en lista desplazable.

Cada ítem permite ver detalle, editar o eliminar.

```
if (movies.isEmpty()) {  
    Box(Modifier.fillMaxSize().padding(padding), contentAlignment =  
Alignment.Center) {  
        Text(if (searchQuery.isEmpty()) "No hay películas" else "No se  
encontraron resultados", style = MaterialTheme.typography.bodyLarge)  
    }  
} else {  
    LazyColumn(modifier = Modifier.padding(padding)) {  
        items(movies) { movie ->  
            MovieListItem(  
                movie = movie,  
                onClick = { navController.navigate("detail/${movie.id}") }  
,  
                onEdit = {  
                    viewModel.setMovie(movie)  
                    navController.navigate("add_edit")  
                },  
                onDelete = { viewModel.deleteMovie(movie) }  
            )  
        }  
    }  
}
```

Si la lista contiene películas guardadas se itera sobre película de la lista y muestra la película y muestra los botones para editar y eliminar los cuales mediante el viewmodel eliminan o abren la función de editar según corresponda. Si se da clic en una película se abre la pantalla de detalles de la película con su id.





2025

```
else { //si la lista contiene películas registradas
    LazyColumn(modifier = Modifier.padding(padding)) {
        items(movies) { movie -> //itera sobre cada movie de movies
            MovieListItem( //muestra lista de películas
                movie = movie,
                //al dar clic en la película manda a la pantalla de
                detalles de la película
                onClick = { navController.navigate("detail/${movie.id}") }
            ),
            onEdit = {
                viewModel.setMovie(movie) //carga al viewmodel
                navController.navigate("add_edit") //manda a la
                pantalla para editar la información
            },
            onDelete = { viewModel.deleteMovie(movie) } //al dar clic
            elimina la película
        )
    }
}
```

StarRating.kt

en StarRating.kt

en las importaciones Incluye clases de diseño y disposición

Row Contenedor horizontal para colocar las estrellas.

Arrangement.spacedBy(4.dp) Espacio de 4 dp entre cada estrella.

Modifier Permite aplicar tamaño, padding, alineación, etc.

dp Unidad de medida para tamaño y espaciado.

Icons Biblioteca de iconos de Material.

Icons.Filled.Star Icono de estrella llena (usada para representar la calificación).

Icons.Outlined.StarBorder Icono de estrella vacía (para las estrellas no seleccionadas).

Icon Composable que muestra un ícono vectorial (estrella en este caso).

IconButton Botón que contiene un ícono y es clickeable, usado para seleccionar la calificación.

@Composable Indica que StarRatingBar es un composable de Jetpack Compose, es decir, se dibuja en la UI y puede reaccionar a cambios de estado.

Modifier Aplica tamaño, padding, disposición, etc. a cada estrella.

dp Unidad de medida para definir tamaño de íconos y espaciado (ej: Modifier.size(40.dp)).





2025

```
package com.example.projecto_peli.ui.theme.screens

import androidx.compose.foundation.layout.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Star
import androidx.compose.material.icons.outlined.StarBorder
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
```

Declaración del Composable rating: Float Valor actual de la calificación (ej. 3.5).

onRatingChanged: (Float) Unit Callback que se ejecuta al seleccionar una estrella.

modifier: Modifier Permite aplicar padding, tamaño o alineación desde afuera.

maxStars: Int Número máximo de estrellas (por defecto 5).

```
@Composable
fun StarRatingBar(
    rating: Float,
    onRatingChanged: (Float) -> Unit,
    modifier: Modifier = Modifier,
    maxStars: Int = 5
```

Contenedor Row Organiza las estrellas horizontalmente.

Arrangement.spacedBy(4.dp) Espacio de 4 dp entre cada estrella.

```
Row(modifier = modifier, horizontalArrangement =
Arrangement.spacedBy(4.dp)) {
```

Loop para dibujar las estrellas

Loop for:

Recorre desde 1 hasta maxStars (por defecto 5).

Cada iteración representa una estrella.

IconButton:

Cada estrella es un botón que el usuario puede pulsar.

onClick = { onRatingChanged(star.toFloat()) } Cambia la calificación a la estrella correspondiente.

Modifier.size(40.dp) Tamaño de cada botón/estrella.





2025

Icon:

imageVector: Si la estrella está dentro del rating (star <= rating) estrella llena (Icons.Filled.Star).

Si está fuera del rating estrella vacía (Icons.Outlined.StarBorder).

contentDescription Texto accesible para lectores de pantalla: "1 estrellas", "2 estrellas", etc.

tint Color de la estrella: Amarillo dorado Color(0xFFFFD700) para estrellas llenas.

Gris Color.Gray para estrellas vacías.

```
for (star in 1..maxStars) {
    IconButton(onClick = { onRatingChanged(star.toFloat()) }, modifier =
Modifier.size(40.dp)) {
        Icon(
            imageVector = if (star <= rating) Icons.Filled.Star else
Icons.Outlined.StarBorder,
            contentDescription = "$star estrellas",
            tint = if (star <= rating) Color(0xFFFFD700) else Color.Gray
    )
}
```

NavGraph.kt

ahora en el package de navigation en la class file NavGraph.kt

en las importaciones

@Composable Indica que la función es un composable de Jetpack Compose, es decir, forma parte de la UI y puede reaccionar a cambios de estado.

NavHost Contenedor que define la estructura de navegación de la app, con rutas y destinos.

composable Define una pantalla destino dentro del NavHost. Cada composable es una ruta que puede ser navegada.

rememberNavController Crea un controlador de navegación que mantiene el estado de la pila de pantallas y permite moverse entre ellas.

MovieViewModel Contiene estado y lógica de la app para las películas (lista de películas, rating, agregar/editar/eliminar).

LoginScreen Pantalla de inicio de sesión.

RegisterScreen Pantalla de registro de usuarios.

MovieListScreen Pantalla que lista todas las películas.





2025

DetailMovieScreen Pantalla de detalle de una película seleccionada.

AddEditMovieScreen Pantalla para agregar o editar una película.

```
@Composable
fun MovieApp(viewModel: MovieViewModel, onPickImage: () -> Unit) {
    val navController = rememberNavController()

    NavHost(navController = navController, startDestination = "login") {
        composable("login") { LoginScreen(navController) }
        composable("movie_list") { MovieListScreen(viewModel,
navController) }
        composable("add_edit") { AddEditMovieScreen(viewModel,
navController, onPickImage) }
        composable("detail/{movieId}") { backStackEntry ->
            val movieId =
backStackEntry.arguments?.getString("movieId")?.toIntOrNull()
            DetailMovieScreen(viewModel, movieId, navController)
        }
        composable("register") { Register(navController) }
    }
}
```

MainActivity

MainActivity hereda de ComponentActivity, que es la base para apps con Jetpack Compose.

Permite usar setContent { ... } para mostrar composables en lugar de layouts XML tradicionales.

```
class MainActivity : ComponentActivity() {
```

Las propiedades de la clase repository Encapsula fuentes de datos (local o remoto) para películas.viewModel Maneja estado y lógica de la app (lista de películas, agregar, editar, eliminar, rating, imágenes).

lateinit significa que estas variables se inicializarán después de la creación del objeto, en onCreate.

```
private lateinit var viewModel: MovieViewModel
private lateinit var repository: MovieRepository
```





2025

El selector de imágenes

registerForActivityResult(ActivityResultContracts.GetContent()) { ... } Crea un launcher para seleccionar contenido del dispositivo (imágenes, archivos, etc).

ActivityResultContracts.GetContent() abre un selector de archivos.

uri?.let { ... } Si el usuario selecciona una imagen (uri no es null), se procesa.

takePersistableUriPermission Permite mantener acceso a la imagen seleccionada incluso si la app se reinicia.

Intent.FLAG_GRANT_READ_URI_PERMISSION Solo permiso de lectura.

viewModel.updateImageUri(it) Actualiza la imagen de la película en el ViewModel, lo que hace que la UI se redibuje automáticamente.

```
private val pickImageLauncher =  
    registerForActivityResult(ActivityResultContracts.GetContent()) {  
        uri: Uri? ->  
            uri?.let {  
                try {  
                    contentResolver.takePersistableUriPermission(  
                        it,  
                        Intent.FLAG_GRANT_READ_URI_PERMISSION  
                    )  
                } catch (e: SecurityException) {  
                    // Ignorar si ya tiene permiso  
                }  
                viewModel.updateImageUri(it)  
            }  
    }
```

Método onCreate Se inicializa el repositorio y el ViewModel.

Esto permite que todas las pantallas accedan a la misma instancia de MovieViewModel, manteniendo el estado.

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)
```

SetContent con Jetpack Compose setContent { ... } Reemplaza el setContentView tradicional de XML y define la UI Compose.

Projecto_PeliTheme { ... } Aplica tema personalizado (colores, tipografía, formas) a toda la app.

MovieApp(viewModel, onPickImage) Inicia el NavHost con todas las pantallas (Login, Register, MovieList, AddEditMovie, DetailMovie).





2025

onPickImage Lanza el selector de imágenes cuando el usuario quiere elegir una imagen para una película.

Le indicamos cual sera la pantalla que se tiene que dibujar, por lo que pasamos la instancia del viewmodel

```
val repository = AppModule.provideRepository(this)
viewModel = MovieViewModel(repository)

setContent {
    MoviesAppTheme {
        MovieApp(
            viewModel = viewModel,
            onPickImage = { pickImageLauncher.launch("image/*") }
        )
    }
}
```





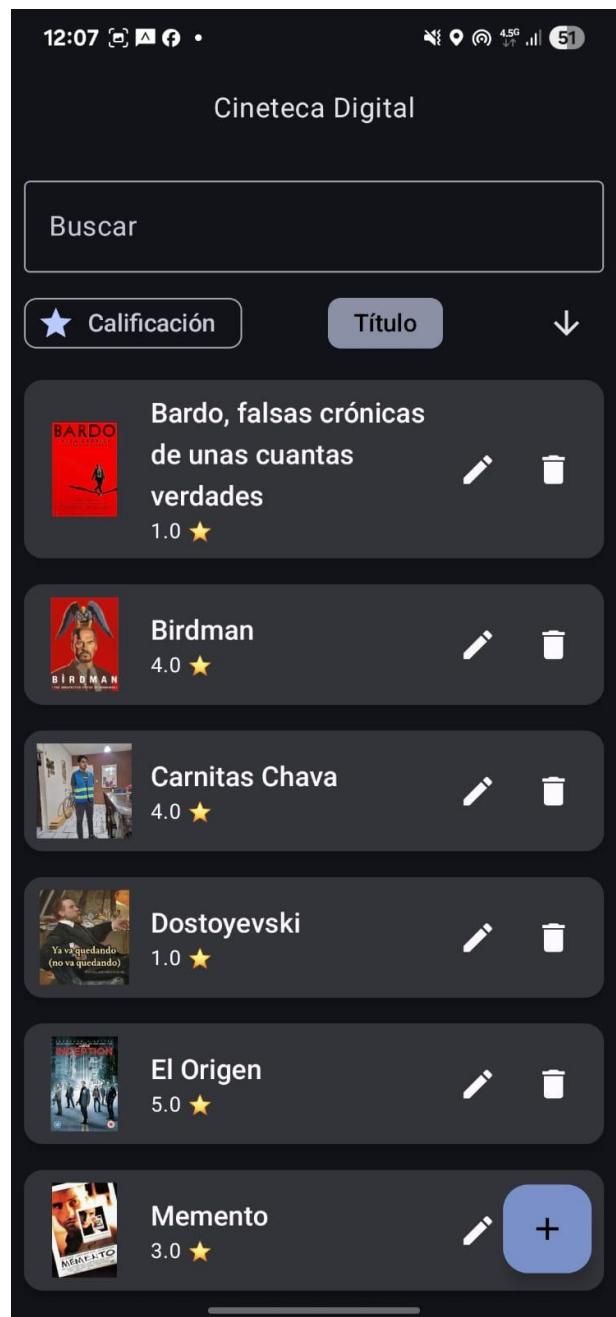
2025

Resultados

Interfaz de Login



Página principal





2025

Busqueda por título

12:08

Cineteca Digital

Buscar: bi

Calificación

Título

Birdman

4.0 ★

1 2 3 4 5 6 7 8 9 0

q w e r t y u i o p

a s d f g h j k l ñ

z x c v b n m

Español(América Latina)

Blvd. del Valle #2301, Col. Guardarrayas; Purísima del Rincón, Guanajuato, C.P.36413

Tel. (476) 744 71 00 e-mail: direccion@tecpurisima.edu.mx

www.tecpurisima.edu.mx

Filtro de ordenamiento descendente por calificación

12:09

Cineteca Digital

Buscar

Calificación

Título

El Origen

5.0 ★

Birdman

4.0 ★

The Revenant

4.0 ★

Whiplash

4.0 ★

Zodiac

4.0 ★

Memento

3.0 ★

+ Agregar

Blvd. del Valle #2301, Col. Guardarrayas; Purísima del Rincón, Guanajuato, C.P.36413

Tel. (476) 744 71 00 e-mail: direccion@tecpurisima.edu.mx

www.tecpurisima.edu.mx





2025

Filtro de ordenamiento por calificación ascendiente

12:09 🔍 ⚡ f • 4.5G 51

Cineteca Digital

Buscar

Calificación Título ↓

Título	Calificación
Bardo, falsas crónicas de unas cuantas verdades	1.0 ★
Birdman	4.0 ★
Dostoyevski	1.0 ★
El Origen	5.0 ★
Memento	3.0 ★
The Revenant	4.0 ★

Campos para agregar una nueva película

12:09 🔍 ⚡ f • 4.5G 51

Nueva Película

Título

Sinopsis

Calificación: 3

Calificación: 3/5

★ ★ ★ ☆ ☆

Reseña

Seleccionar Imagen

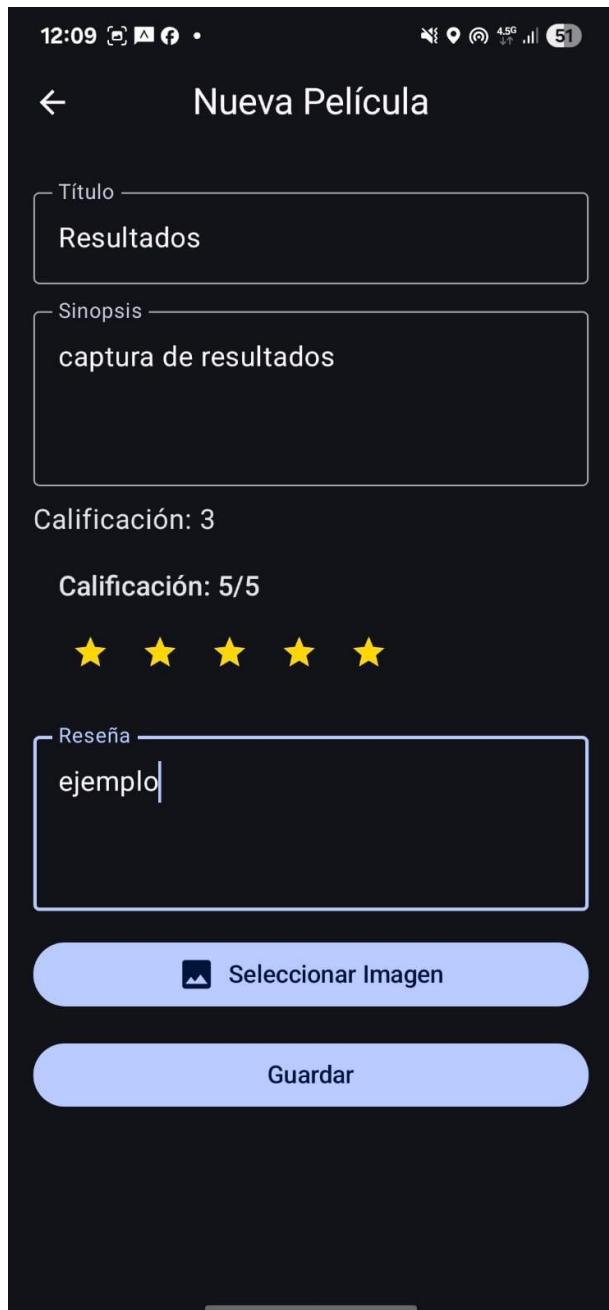
Guardar



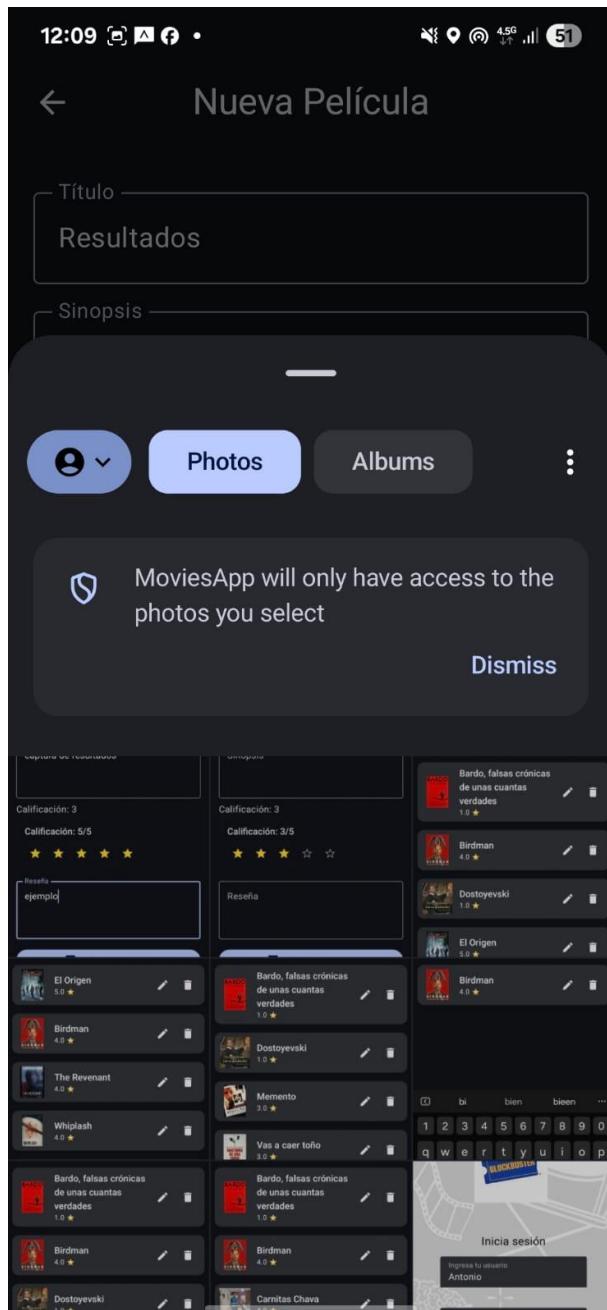


2025

Llenando campos



Seleccionado imagen desde La galería





2025

Vista de detalles de una película

12:10 4G 51

Memento

GUY PEARCE CARRIE-ANNE MOSS JOE PANTOLIANO
MEMENTO

Sinopsis:
Un hombre que busca resolver un crimen pero su amnesia se lo dificulta

Calificación: 3.0 ★

Reseña:
Buena película, muy diferente y con un buen giro al final





2025

Conclusión

El desarrollo de la aplicación de catálogo de películas permitió integrar herramientas modernas de Android, logrando una solución completa con autenticación, CRUD, persistencia de datos y navegación fluida. Se fortalecieron habilidades en Kotlin, ViewModel, Room y NavGraph, garantizando una experiencia de usuario consistente.

El proyecto demuestra que con Android Jetpack y buenas prácticas de programación es posible crear aplicaciones funcionales, seguras y escalables, contribuyendo al aprendizaje y desarrollo de competencias en el ámbito móvil.

Además, se incorporaron buenas prácticas como el encapsulamiento de lógica en repositorios, la validación de credenciales, y el manejo de imágenes mediante Coil, lo que demuestra una comprensión sólida de los componentes modernos de Android. La implementación de filtros y ordenamientos dinámicos en tiempo real también refleja el dominio de flujos reactivos y la capacidad de construir interfaces adaptables a las necesidades del usuario.

Este proyecto no solo consolidó habilidades técnicas, sino que también fortaleció el trabajo colaborativo, la planificación estructurada y la capacidad de resolver problemas de manera efectiva. En conjunto, la aplicación desarrollada es una muestra del potencial que tienen los estudiantes para crear soluciones funcionales y escalables en el entorno móvil, preparándolos para enfrentar desafíos reales en el desarrollo de software.

