Universidad Claeh
Centro CTC

Analista Programador
Taller de Tecnologías

“.NET FLIX”

Leonardo Suárez
Francisco Machado
Angel Machado

2024

Resumen

El presente documento tiene como finalidad exponer el desarrollo de una aplicación multiplataforma para la cadena de cines privados de streaming. La propuesta para este proyecto incluye la implementación de una aplicación desarrollada en .NET MAUI que permite gestionar las sucursales de la cadena, acceder a una extensa lista de películas, y gestionar perfiles de usuarios de manera eficiente.

El objetivo principal es proporcionar una plataforma integral que ofrezca a los usuarios una experiencia única al interactuar con las películas, permitiéndoles buscar, filtrar, y marcar películas como favoritas. Además, los usuarios podrán gestionar sucursales en un mapa interactivo, donde se muestran las ubicaciones de las mismas y la ruta más corta desde su ubicación actual. Adicionalmente, la aplicación permitirá la creación y gestión de perfiles de usuario, ofreciendo diferentes métodos de autenticación según la plataforma utilizada, como la autenticación mediante huella dactilar en dispositivos Android.

Para la implementación de este proyecto se han utilizado tecnologías modernas y robustas que garantizan un rendimiento óptimo y una experiencia de usuario fluida. La aplicación está desarrollada en .NET MAUI, un framework que permite la creación de aplicaciones multiplataforma para Android, iOS y Windows, utilizando una única base de código. Para la gestión de datos, se ha utilizado SQLite, una base de datos ligera que facilita el almacenamiento local de datos de manera eficiente. Además, el sistema integra el consumo de la API de The Movie Database (TMDb) para obtener información actualizada sobre películas, actores y otros contenidos relacionados, lo que enriquece la experiencia del usuario al acceder a contenido multimedia actualizado y de calidad.

Índice

1. Introducción

En la presente documentación se abordarán diversos conceptos y tecnologías fundamentales en el desarrollo de aplicaciones multiplataforma modernas, destacándose la Programación Orientada a Objetos (POO), el framework .NET MAUI, y la implementación de bases de datos utilizando SQLite. Además, se explorará el consumo de APIs externas para enriquecer la funcionalidad del sistema y ofrecer una experiencia de usuario avanzada.

La Programación Orientada a Objetos (POO) es un paradigma de programación que permite estructurar el código en entidades denominadas clases y objetos. Las clases actúan como plantillas que definen atributos y métodos comunes, mientras que los objetos son instancias de estas clases con propiedades específicas. Este enfoque modular y estructurado facilita la reutilización y el mantenimiento del código, promoviendo una mayor eficiencia y organización en el desarrollo de software.

.NET MAUI (Multi-platform App UI) es un framework de desarrollo que permite la creación de aplicaciones nativas para múltiples plataformas (Android, iOS, Windows y macOS) utilizando una única base de código. Sigue el patrón de diseño Modelo-Vista-ViewModel (MVVM), el cual separa la lógica de la aplicación, la interfaz de usuario y la interacción con los datos, permitiendo un desarrollo más limpio y escalable. Este patrón mejora la capacidad de mantenimiento y extensión de la aplicación, facilitando la implementación de nuevas funcionalidades y la adaptación a diferentes plataformas.

SQLite es una base de datos ligera y altamente eficiente que se integra perfectamente con aplicaciones móviles y de escritorio. En este proyecto, se utiliza SQLite para la gestión de datos locales, permitiendo a la aplicación almacenar y recuperar información de manera rápida y fiable, incluso en dispositivos con recursos limitados. Las operaciones CRUD (Create, Read, Update, Delete) se gestionan de manera eficiente, asegurando la persistencia de los datos críticos para el funcionamiento del sistema.

El consumo de APIs (Application Programming Interfaces) es una práctica esencial en el desarrollo de aplicaciones modernas. Las APIs permiten la integración de servicios y datos externos, enriqueciendo la funcionalidad del sistema. En este proyecto, se hace un uso extensivo de la API de The Movie Database (TMDb) para obtener información actualizada sobre películas, actores y otros contenidos multimedia, mejorando la experiencia del usuario al ofrecerle acceso a un vasto catálogo de contenido audiovisual.

Este proyecto integra diversas tecnologías y conceptos clave en el desarrollo de software, combinando POO, .NET MAUI, bases de datos con SQLite y el consumo de APIs, para construir una aplicación multiplataforma robusta, escalable y altamente funcional.

1.1. Clases y Objetos

Las clases y los objetos constituyen los pilares fundamentales de la Programación Orientada a Objetos (POO), y mantienen una estrecha interrelación. Los objetos son instancias creadas a partir de clases, y estas últimas funcionan como modelos o moldes para la creación de objetos. Comúnmente, a los objetos se les denomina instancia de plantilla. Ahora, piensa en personas concretas: tú, tu madre, el presidente de tu país, figuras históricas como Alan Turing o Nelson Mandela, entre otros ejemplos. Estos individuos particulares son ejemplos de objetos. Cada uno de ellos se ajusta a los atributos definidos por la plantilla de la clase "persona", como el nombre, la edad, la estatura, y así sucesivamente.[1]

1.2. .NET MAUI

.NET Multi-platform App UI (.NET MAUI) es un marco multiplataforma para crear aplicaciones móviles y de escritorio nativas con C# y XAML.
Con .NET MAUI, puede desarrollar aplicaciones que se pueden ejecutar en Android, iOS, macOS y Windows desde una sola base de código compartida.

.NET MAUI es una evolución de código abierto de Xamarin.Forms, extendida desde escenarios móviles a escritorio, con controles de interfaz de usuario recopilados para mejorar el rendimiento y la extensibilidad. Si ha usado anteriormente Xamarin.Forms para compilar interfaces de usuario multiplataforma, observará muchas similitudes con .NET MAUI. Sin embargo, también hay algunas diferencias. Mediante .NET MAUI, puede crear aplicaciones multiplataforma con un solo proyecto, pero puede agregar recursos y código fuente específicos de la plataforma si es necesario. Uno de los objetivos clave de .NET MAUI es permitirle implementar la mayor parte de la lógica de la aplicación y el diseño de la interfaz de usuario en una única base de código.[2]

1.3. SQLite

SQLite es una biblioteca en proceso que implementa un motor de base de datos SQL transaccional autónomo , sin servidor y sin configuración . El código de SQLite es de dominio público y, por lo tanto, su uso es gratuito para cualquier propósito , comercial o privado. SQLite es la base de datos más implementada en el mundo, con más aplicaciones de las que podemos contar, incluidos varios proyectos de alto perfil.

SQLite es un motor de base de datos SQL integrado. A diferencia de la mayoría de las otras bases de datos SQL, SQLite no tiene un proceso de servidor independiente. SQLite lee y escribe directamente en archivos de disco ordinarios. Una base de datos SQL completa con múltiples tablas, índices, activadores y vistas está contenida en un solo archivo de disco. El formato de archivo de base de datos es multiplataforma: puede copiar libremente una base de datos entre sistemas de 32 bits y 64 bits o entre arquitecturas big-endian y little-endian . Estas características hacen de SQLite una opción popular como formato de archivo de aplicación . [3]

1.4. API

Las API son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos. Por ejemplo, el sistema de software del instituto de meteorología contiene datos meteorológicos diarios. La aplicación meteorológica de su teléfono “habla” con este sistema a través de las API y le muestra las actualizaciones meteorológicas diarias en su teléfono.

API significa “interfaz de programación de aplicaciones”. En el contexto de las API, la palabra aplicación se refiere a cualquier software con una función distinta. La interfaz puede considerarse como un contrato de servicio entre dos aplicaciones. Este contrato define cómo se comunican entre sí mediante solicitudes y respuestas. La documentación de su

API contiene información sobre cómo los desarrolladores deben estructurar esas solicitudes y respuestas.[4]

1.4.1. API REST

REST significa transferencia de estado representacional. REST define un conjunto de funciones como GET, PUT, DELETE, etc. que los clientes pueden utilizar para acceder a los datos del servidor. Los clientes y los servidores intercambian datos mediante HTTP.

La principal característica de la API de REST es que no tiene estado. La ausencia de estado significa que los servidores no guardan los datos del cliente entre las solicitudes. Las solicitudes de los clientes al servidor son similares a las URL que se escriben en el navegador para visitar un sitio web. La respuesta del servidor son datos simples, sin la típica representación gráfica de una página web.[5]

1.4.2. API de TMDB

El servicio API está destinado a aquellos interesados en utilizar nuestras imágenes y/o datos de películas, programas de televisión o actores en su aplicación. Nuestra API es un sistema que le proporcionamos a usted y a su equipo para que obtengan y utilicen de manera programática nuestros datos y/o imágenes.[6]

2. Metodología de estudio

El desarrollo de este sistema se fundamenta en una arquitectura robusta y modular utilizando .NET MAUI, lo que asegura una clara separación de responsabilidades y facilita tanto el mantenimiento como la escalabilidad del proyecto. En el núcleo de esta estructura se encuentra la implementación del patrón MVVM (Modelo-Vista-ViewModel), que organiza el proyecto en tres componentes principales: el Modelo, que gestiona la lógica de los datos y la interacción con la base de datos; la Vista, que maneja la presentación de la interfaz de usuario; y el ViewModel, que actúa como intermediario entre el Modelo y la Vista, gestionando los datos y la lógica de presentación sin involucrar directamente la interfaz de usuario.

El proyecto se basa fundamentalmente en el consumo de una API de películas, que es la fuente principal de información para las funcionalidades clave del sistema. Esta API permite obtener y gestionar datos sobre diversas películas, integrándolos en la aplicación de manera eficiente y segura.

La autorización de los usuarios en el sistema no se gestiona a través de roles, sino a nivel individual. Los usuarios deben registrarse para acceder al sistema, y el proceso de autenticación varía según la plataforma. En dispositivos Android, los usuarios pueden autenticarse mediante huella dactilar, mientras que en Windows, la autenticación se realiza mediante el uso de email y contraseña. Esto asegura que solo usuarios registrados puedan acceder a las funcionalidades del sistema.

El sistema incluye dos clases principales: Sucursales y Usuarios. Los usuarios pueden agregar y eliminar sucursales, las cuales se representan mediante pins en un mapa interactivo. El manejo de mapas es una característica crucial del sistema, permitiendo a los usuarios visualizar todas las sucursales en un mapa, agregar nuevas ubicaciones y eliminar sucursales existentes. Además, el sistema permite calcular y mostrar la ruta desde la ubicación actual del usuario hasta un pin seleccionado en el mapa, lo que mejora significativamente la experiencia de usuario y la funcionalidad de la aplicación.

La base de datos del proyecto está diseñada utilizando SQLite, una solución ligera y eficiente que garantiza la integridad y seguridad de la información tanto en dispositivos móviles como de escritorio. La interacción con la base de datos se realiza de manera eficiente, facilitando la manipulación de datos a través de consultas y operaciones que aseguran un rendimiento óptimo y una experiencia de usuario fluida.

La integración con APIs externas, como la API de geocodificación de Google, permite convertir direcciones en coordenadas geográficas que se utilizan para mostrar las ubicaciones de las sucursales en el mapa. Esta integración se maneja de manera eficiente, asegurando que los datos externos sean incorporados de forma segura y consistente dentro del sistema.

La metodología de desarrollo del sistema se basa en el uso de tecnologías modernas y prácticas de desarrollo de software que aseguran un producto final robusto, escalable y fácil de mantener, cumpliendo con los requisitos específicos del proyecto y proporcionando una experiencia de usuario óptima tanto en plataformas móviles como de escritorio.

2.1. Services

2.2. TmdbService

```
namespace ObligatorioTT.Services
{
    11 referencias
    public class TmdbService
    {
        private const string ApiKey = "7aa67c19f9c8135507e8aaf245874e41";
        public const string TmdbHttpClientName = "TmdbClient";
        private readonly IHttpClientFactory _httpClientFactory;

        0 referencias
        public TmdbService(IHttpClientFactory httpClientFactory)
        {
            _httpClientFactory = httpClientFactory;
        }

        6 referencias
        private HttpClient HttpClient => _httpClientFactory.CreateClient(TmdbHttpClientName);

        1 referencia
        public async Task<IEnumerable<Genre>> GetGenresAsync()
        {
            var genresWrapper = await HttpClient.GetFromJsonAsync<GenreWrapper>($"{TmdbUrls.MoviesGenres}&api_key={ApiKey}");
            return genresWrapper.Genres;
        }
        1 referencia
        public async Task<IEnumerable<Media>> GetTrendingAsync() =>
            await GetMediasAsync(TmdbUrls.Trending);

        1 referencia
        public async Task<IEnumerable<Media>> GetTopRatedAsync() =>
            await GetMediasAsync(TmdbUrls.TopRated);
        1 referencia
        public async Task<IEnumerable<Media>> GetPopularAsync() =>
            await GetMediasAsync(TmdbUrls.Popular);
        1 referencia
        public async Task<IEnumerable<Media>> GetUpcomingAsync() =>
            await GetMediasAsync(TmdbUrls.Upcoming);

        1 referencia
        public async Task<IEnumerable<Media>> GetActionAsync() =>
            await GetMediasAsync(TmdbUrls.Action);

        1 referencia
        public async Task<IEnumerable<Media>> GetSimilarAsync(int id, string type = "movie") =>
            await GetMediasAsync(
                $"{TmdbUrls.GetSimilar(id, type)}&api_key={ApiKey}");
```

Figura 2.1.1. TmdbService.

```
        1 referencia
        public async Task<IEnumerable<Video>?> GetTrailersAsync(int id, string type = "movie")
        {
            var videosWrapper = await HttpClient.GetFromJsonAsync<VideosWrapper>
                ($"{TmdbUrls.GetTrailers(id, type)}&api_key={ApiKey}");

            if (videosWrapper?.results.Length > 0)
            {
                var trailerTeasers = videosWrapper.results.Where(VideosWrapper.FilterTrailerTeasers);
                return trailerTeasers;
            }
            return null;
        }
        6 referencias
        private async Task<IEnumerable<Media>> GetMediasAsync(string url)
        {
            var trendingMoviesCollections = await HttpClient.GetFromJsonAsync<Movie>($"{url}&api_key={ApiKey}");
            return trendingMoviesCollections.results.Select(r => r.ToMediaObject());
        }

        1 referencia
        public async Task<MovieDetail> GetMovieDetailAsync(int movieId)
        {
            return await HttpClient.GetFromJsonAsync<MovieDetail>($"{TmdbUrls.GetMovieDetails(movieId)}&api_key={ApiKey}");
        }
        1 referencia
        public async Task<Credits> GetMovieCreditsAsync(int movieId)
        {
            string url = $"https://api.themoviedb.org/3/movie/{movieId}/credits?api_key={ApiKey}";

            var response = await HttpClient.GetAsync(url);

            if (response.IsSuccessStatusCode)
            {
                return await response.Content.ReadFromJsonAsync<Credits>();
            }
            else
            {
                // Manejo del error si la solicitud no fue exitosa
                return null;
            }
        }

        1 referencia
        public async Task<IEnumerable<Media>> SearchMoviesAsync(string query)
        {
            var url = $"https://api.themoviedb.org/3/search/movie?query={query}&api_key={ApiKey}";
            var searchResult = await HttpClient.GetFromJsonAsync<Movie>(url);
            return searchResult.results.Select(r => r.ToMediaObject());
        }
```

Figura 2.1.2. TmdbService.

```
9 referencias
public static class TmdbUrls
{
    public const string Trending = "3/trending/all/week?language=es-sp";
    public const string TopRated = "3/movie/top_rated?language=es-sp";
    public const string Popular = "3/movie/popular?language=es-sp";
    public const string Upcoming = "3/movie/upcoming?language=es-sp";
    public const string Action = "3/discover/movie?language=es-sp&with_genres=28";
    public const string MoviesGenres = "3/genre/movie/list?language=es-sp";
    1 referencia
    public static string GetTrailers(int movieId, string type = "movie") => $"3/{type ?? "movie"}/{movieId}/videos?language=es-sp";
    1 referencia
    public static string GetMovieDetails(int movieId, string type = "movie") => $"3/{type ?? "movie"}/{movieId}?language=es-sp";
    1 referencia
    public static string GetSimilar(int movieId, string type = "movie") => $"3/{type ?? "movie"}/{movieId}/similar?language=es-sp";

}


2 referencias
public class Movie
{
    0 referencias
    public int page { get; set; }
    2 referencias
    public Result[] results { get; set; }
    0 referencias
    public int total_pages { get; set; }
    0 referencias
    public int total_results { get; set; }

}
```

Figura 2.1.3. TmdbService.

```
1 referencia
public class Result
{
    1 referencia
    public string backdrop_path { get; set; }
    0 referencias
    public int[] genre_ids { get; set; }
    1 referencia
    public int id { get; set; }
    1 referencia
    public string original_title { get; set; }
    1 referencia
    public string original_name { get; set; }
    1 referencia
    public string overview { get; set; }
    1 referencia
    public string poster_path { get; set; }
    1 referencia
    public string release_date { get; set; }
    1 referencia
    public string title { get; set; }
    1 referencia
    public string name { get; set; }
    0 referencias
    public bool video { get; set; }
    1 referencia
    public string media_type { get; set; } // "movie" or "tv"
    3 referencias
    public string ThumbnailPath => poster_path ?? backdrop_path;
    1 referencia
    public string Thumbnail => $"https://image.tmdb.org/t/p/w600_and_h900_bestv2/{ThumbnailPath}";
    1 referencia
    public string ThumbnailSmall => $"https://image.tmdb.org/t/p/w220_and_h330_face/{ThumbnailPath}";
    1 referencia
    public string ThumbnailUrl => $"https://image.tmdb.org/t/p/original/{ThumbnailPath}";
    1 referencia
    public string DisplayTitle => title ?? name ?? original_title ?? original_name;

    2 referencias
    public Media ToMediaObject() =>
        new()
        {
            Id = id,
            DisplayTitle = DisplayTitle,
            MediaType = media_type,
            Overview = overview,
            ReleaseDate = release_date,
            Thumbnail = Thumbnail,
            ThumbnailSmall = ThumbnailSmall,
            ThumbnailUrl = ThumbnailUrl
        };
}
```

Figura 2.1.4. TmdbService.

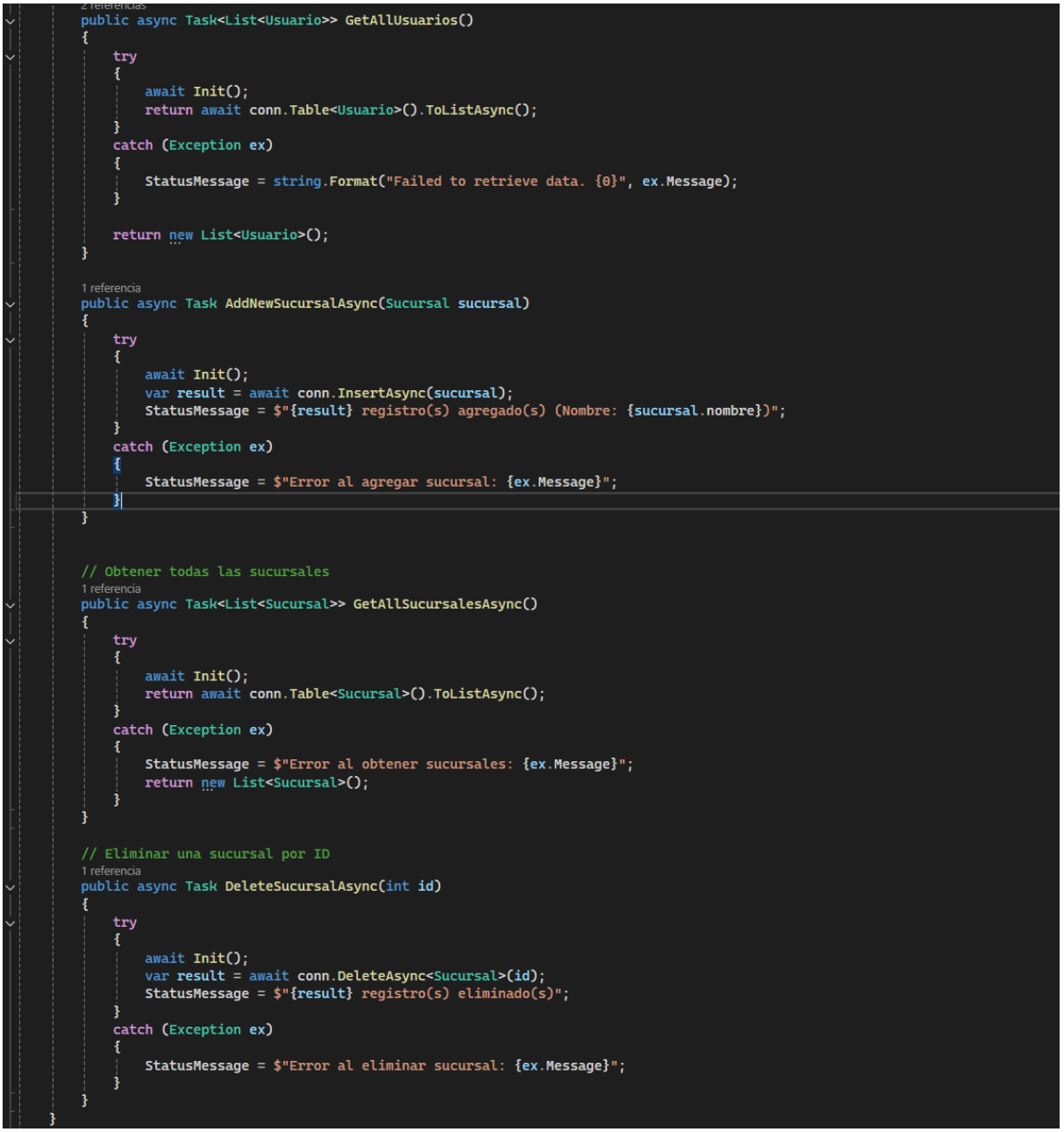
```
public class MovieDetail
{
    1 referencia
    public bool adult { get; set; }
    0 referencias
    public string backdrop_path { get; set; }
    0 referencias
    public object belongs_to_collection { get; set; }
    0 referencias
    public int budget { get; set; }
    3 referencias
    public Genre[] genres { get; set; }
    0 referencias
    public string homepage { get; set; }
    0 referencias
    public int id { get; set; }
    0 referencias
    public string imdb_id { get; set; }
    0 referencias
    public string original_language { get; set; }
    0 referencias
    public string original_title { get; set; }
    0 referencias
    public string overview { get; set; }
    0 referencias
    public float popularity { get; set; }
    0 referencias
    public string poster_path { get; set; }
    0 referencias
    public Production_Companies[] production_companies { get; set; }
    0 referencias
    public Production_Countries[] production_countries { get; set; }
    0 referencias
    public string release_date { get; set; }
    0 referencias
    public int revenue { get; set; }
    0 referencias
    public int runtime { get; set; }
    0 referencias
    public Spoken_Languages[] spoken_languages { get; set; }
    0 referencias
    public string status { get; set; }
    0 referencias
    public string tagline { get; set; }
    0 referencias
    public string title { get; set; }
    0 referencias
    public bool video { get; set; }
    0 referencias
    public float vote_average { get; set; }
    0 referencias
    public int vote_count { get; set; }
}
```

Figura 2.1.5. TmdbService.

2.3. Repository

```
public class Repository
{
    string rutaDB;
    8 referencias
    public string StatusMessage { get; set; }

    private SQLiteAsyncConnection conn;

    5 referencias
    private async Task Init()
    {
        if (conn is not null)
            return;
        conn = new SQLiteAsyncConnection(rutaDB);
        await conn.CreateTableAsync<Usuario>();
        await conn.CreateTableAsync<Sucursal>();
    }

    0 referencias
    public Repository(string _rutaDB)
    {
        rutaDB = _rutaDB;
    }

    1 referencia
    public async Task AddNewUsuarioAsync(Usuario usuario)
    {
        try
        {
            await Init();
            var result = await conn.InsertAsync(usuario);
            StatusMessage = $"{result} registro(s) agregado(s) (Nombre: {usuario.nombre})";
        }
        catch (Exception ex)
        {
            StatusMessage = $"Error al agregar usuario: {ex.Message}";
        }
    }
```

Figura 2.3.1. Repository.



```
2 referencias
public async Task<List<Usuario>> GetAllUsuarios()
{
    try
    {
        await Init();
        return await conn.Table<Usuario>().ToListAsync();
    }
    catch (Exception ex)
    {
        StatusMessage = string.Format("Failed to retrieve data. {0}", ex.Message);
    }

    return new List<Usuario>();
}

1 referencia
public async Task AddNewSucursalAsync(Sucursal sucursal)
{
    try
    {
        await Init();
        var result = await conn.InsertAsync(sucursal);
        StatusMessage = $"{result} registro(s) agregado(s) (Nombre: {sucursal.nombre})";
    }
    catch (Exception ex)
    {
        StatusMessage = $"Error al agregar sucursal: {ex.Message}";
    }
}


// Obtener todas las sucursales
1 referencia
public async Task<List<Sucursal>> GetAllSucursalesAsync()
{
    try
    {
        await Init();
        return await conn.Table<Sucursal>().ToListAsync();
    }
    catch (Exception ex)
    {
        StatusMessage = $"Error al obtener sucursales: {ex.Message}";
        return new List<Sucursal>();
    }
}

// Eliminar una sucursal por ID
1 referencia
public async Task DeleteSucursalAsync(int id)
{
    try
    {
        await Init();
        var result = await conn.DeleteAsync<Sucursal>(id);
        StatusMessage = $"{result} registro(s) eliminado(s)";
    }
    catch (Exception ex)
    {
        StatusMessage = $"Error al eliminar sucursal: {ex.Message}";
    }
}
}
```

Figura 2.3.2. Repository.

2.4. DetailsViewModel

```
namespace ObligatorioTT.ViewModels
{
    [QueryProperty(nameof(Media), nameof(Media))]
    11 referencias
    public partial class DetailsViewModel : ObservableObject
    {
        private readonly TmdbService _tmdbService;
        0 referencias
        public DetailsViewModel(TmdbService tmdbService)
        {
            _tmdbService = tmdbService;
        }

        [ObservableProperty]
        private Media _media;

        [ObservableProperty]
        private string _mainTrailerUrl;

        [ObservableProperty]
        private Media _selectedMedia;

        1 referencia
        public ObservableCollection<Video> VideosCol { get; set; } = new();
        1 referencia
        public ObservableCollection<Media> similarMedia { get; set; } = new();

        [ObservableProperty]
        private int _similarItemWidth = 125;

        [ObservableProperty]
        private bool _isBusy;

        [ObservableProperty]
        private string _castNames;

        [ObservableProperty]
        private string _crewNames;

        [ObservableProperty]
        private string _videoSource;

        [ObservableProperty]
        private bool _isPlaying;

        // Propiedad para MovieDetail
        [ObservableProperty]
        private MovieDetail _movieDetail;

        0 referencias
        public string AdultContent => $"Contenido de Adultos: {(MovieDetail?.adult == true ? "Sí" : "No")}";
        0 referencias
        public string GenreNames => MovieDetail?.genres != null && MovieDetail.genres.Any()
                                    ? string.Join(", ", MovieDetail.genres.Select(g => g.Name))
                                    "Sin géneros";
    }
```

Figura 2.4.1. DetailsViewModel.

2.5. MainPage.xaml

```
<Grid Margin="5" Padding="5">
    <ScrollView BackgroundColor="black">
        <VerticalStackLayout>
            <Grid HeightRequest="550">
                <Image Aspect="Center"
                       Margin="50"
                       HeightRequest="365"
                       Style="{StaticResource ImageStyle}"
                       VerticalOptions="Start"
                       Opacity="0.8">
                    <Image.Source>
                        <UriImageSource Uri="{Binding TrendingMovie.Thumbnail}"/>
                    </Image.Source>
                    <Image.GestureRecognizers>
                        <TapGestureRecognizer Command="{Binding SelectMediaCommand}"
                                              CommandParameter="{Binding TrendingMovie}"/>
                    </Image.GestureRecognizers>
                </Image>

                <HorizontalStackLayout VerticalOptions="End"
                                       HorizontalOptions="Center"
                                       Spacing="70">
                    <VerticalStackLayout>
                        <Grid HorizontalOptions="Center">
                            <Image Margin="2"
                              Style="{StaticResource IconStyle}"
                              HorizontalOptions="Center"
                              VerticalOptions="Center"
                              Source="info.png">
                            </Image>
                        </Grid>
                        <Label Text="INFO"
                          TextColor="White"
                          FontAttributes="Bold"
                          HorizontalOptions="Center"
                          VerticalOptions="Center"
                          FontSize="20"></Label>
                        <VerticalStackLayout.GestureRecognizers>
                            <TapGestureRecognizer Command="{Binding SelectMediaCommand}"
                                                  CommandParameter="{Binding TrendingMovie}"/>
                        </VerticalStackLayout.GestureRecognizers>
                    </VerticalStackLayout>

                </HorizontalStackLayout>
```

Setter.Value

Figura 2.5.1. MainPage.xaml.

```
            </Grid>
            <controls:MovieRow Heading="Mejores valoradas"
                               Movies="{Binding TopRated}"
                               IsLarge="False"
                               MediaSelected="MovieRow_MediaSelected"/>
            <controls:MovieRow Heading="Populares"
                               Movies="{Binding Popular}"
                               IsLarge="True"
                               MediaSelected="MovieRow_MediaSelected"/>
            <controls:MovieRow Heading="Trendencias"
                               Movies="{Binding Trending}"
                               IsLarge="False"
                               MediaSelected="MovieRow_MediaSelected"/>
            <controls:MovieRow Heading="Próximamente"
                               Movies="{Binding Upcoming}"
                               IsLarge="False"
                               MediaSelected="MovieRow_MediaSelected"/>
        </VerticalStackLayout>
    </ScrollView>
    <Grid HeightRequest="35"
          VerticalOptions="Start">
        <FlexLayout JustifyContent="End">


            <ImageButton
              Source="user"
              Aspect="AspectFill"
              HeightRequest="35"
              WidthRequest="35"
              Clicked="UserButton_Clicked"
              Margin="10">


            </ImageButton>
        </FlexLayout>
    </Grid>
    <controls:MovieInfoBox  Media="{Binding SelectedMedia}"
                            VerticalOptions="End"
                            IsVisible="{Binding ShowMovieInfoBox}"
                            Closed="MovieInfoBox_Closed"
                            />

</Grid>
```

Figura 2.5.2. MainPage.xaml.

```
public partial class MainPage : ContentPage
{
    private readonly HomeViewModel _homeViewModel;
    private readonly SearchViewModel _searchViewModel;
    0 referencias
    public MainPage(HomeViewModel homeViewModel)
    {
        InitializeComponent();
        _homeViewModel = homeViewModel;
        BindingContext = _homeViewModel;
    }
    0 referencias
    private void MovieRow_MediaSelected(object sender, Controls.MediaSelectEventsArgs e)
    {
        _homeViewModel.SelectMediaCommand.Execute(e.Media);
    }
    0 referencias
    private void MovieInfoBox_Closed(object sender, EventArgs e)
    {
        _homeViewModel.SelectMediaCommand.Execute(null);
    }
    0 referencias
    protected async override void OnAppearing()
    {
        base.OnAppearing();
        await _homeViewModel.InitializeAsync();
    }
    0 referencias
    private async Task AnimateLabel(Label label, bool isSelected)
    {
        if (isSelected)
        {
            await label.ScaleTo(1.5, 250);

        }
        else
        {
            await label.ScaleTo(1, 250);
        }
    }
    0 referencias
    private async void UserButton_Clicked(object sender, EventArgs e)
    {
        await Navigation.PushAsync(new Profile());

    }
}
```

Figura 2.5.3. MainPage.cs.

2.6. Sucursales.xaml

```xaml
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:maps="clr-namespace:Microsoft.Maui.Controls.Maps;assembly=Microsoft.Maui.Controls.Maps"
             xmlns:sensors="clr-namespace:Microsoft.Maui.Devices.Sensors;assembly=Microsoft.Maui.Essentials"
             BackgroundColor="Black"
             x:Class="ObligatorioTT.Pages.Sucursales"
             Title="Sucursales">
    <Grid Padding="10" RowDefinitions="Auto,Auto,Auto,Auto,*">
        <maps:Map Grid.Row="0"
                  x:Name="SucursalesMap"
                  MapType="Street"
                  IsShowingUser="True"
                  IsZoomEnabled="True"
                  IsScrollEnabled="True"
                  VerticalOptions="FillAndExpand"
                  HorizontalOptions="FillAndExpand"
                  HeightRequest="500">
        </maps:Map>
        <Label Grid.Row="1"
               Text="Lista de Sucursales"
               FontSize="Large"
               HorizontalOptions="Center"
               Margin="0,10,0,0"/>
        <Grid Grid.Row="2" ColumnDefinitions="*,*">
            <Button Grid.Column="0"
                    Text="Agregar Nueva Sucursal"
                    Command="{Binding NavigateToCrearSucursalCommand}"
                    HorizontalOptions="Center"
                    Margin="0,10,10,0"/>
            <Button Grid.Column="1"
                    Text="Mostrar/Ocultar Sucursales"
                    Command="{Binding ToggleSucursalesVisibilityCommand}"
                    HorizontalOptions="Center"
                    Margin="10,10,0,0"/>
        </Grid>
        <CollectionView Grid.Row="4"
                        x:Name="SucursalesCollectionView"
                        ItemsSource="{Binding Sucursales}"
                        IsVisible="{Binding IsSucursalesVisible}">
            <CollectionView.ItemTemplate>
                <DataTemplate>
                    <Grid ColumnDefinitions="*,Auto" Padding="10">
                        <StackLayout Grid.Column="0" Spacing="0">
                            <Label Text="{Binding nombre}"
                                   FontSize="Medium" />
                            <Label Text="{Binding direccion, StringFormat='Dirección: {0}'}"
                                   FontSize="Small"
                                   TextColor="Gray" />
                            <Label Text="{Binding telefono, StringFormat='Teléfono: {0}'}"
                                   FontSize="Small"
                                   TextColor="Gray" />
                        </StackLayout>
                        <Button Grid.Column="1"
                                Text="Eliminar"
                                Command="{Binding Source={RelativeSource AncestorType={x:Type ContentPage}}, Path=BindingContext.DeleteSucursalCommand}"
                                CommandParameter="{Binding .}"
                                HorizontalOptions="End"
                                VerticalOptions="Center"
                                Margin="10,0,0,0"/>
                    </Grid>
                </DataTemplate>
            </CollectionView.ItemTemplate>
        </CollectionView>
    </Grid>
</ContentPage>
```

Figura 2.6.1. Sucursales.xaml.

```
public partial class Sucursales : ContentPage
{
    private readonly SucursalesViewModel _viewModel;
    private readonly DirectionsService _directionsService;
    0 referencias
    public Sucursales(SucursalesViewModel viewModel)
    {
        InitializeComponent();
        _viewModel = viewModel;
        BindingContext = _viewModel;
        _directionsService = new DirectionsService("AIzaSyB6mykcb3pshOlbdIaUpYtOmLK-fdkXsW0");
        SuscribirEventos();
    }
    1 referencia
    private void SuscribirEventos()
    {
        MessagingCenter.Subscribe<SucursalesViewModel, Location>(this, "SucursalAgregada", (sender, posicion) =>
        {
            var nuevoPin = new Pin
            {
                Label = "Sucursal",
                Location = new Location(posicion.Latitude, posicion.Longitude),
                Address = "Dirección ingresada"
            };
            SucursalesMap.Pins.Add(nuevoPin);
            nuevoPin.MarkerClicked += (s, args) => OnPinClicked(nuevoPin);
        });
        MessagingCenter.Subscribe<SucursalesViewModel, Location>(this, "SucursalEliminada", (sender, posicion) =>
        {
            var pinToRemove = SucursalesMap.Pins.FirstOrDefault(pin =>
                pin.Location.Latitude == posicion.Latitude && pin.Location.Longitude == posicion.Longitude);

            if (pinToRemove != null)
            {
                SucursalesMap.Pins.Remove(pinToRemove);
                SucursalesMap.MapElements.Clear();
            }
        });
        SucursalesMap.MapClicked += (s, e) =>
        {
            SucursalesMap.MapElements.Clear();
        };
    }
    1 referencia
    private async void OnPinClicked(Pin pin)
    {
        var ubicacionDispositivo = await Geolocation.GetLastKnownLocationAsync();

        if (ubicacionDispositivo != null && pin.Location != null)
        {
            SucursalesMap.MapElements.Clear();
            var polyline = await _directionsService.GetRouteAsync(ubicacionDispositivo, pin.Location);
            SucursalesMap.MapElements.Add(polyline);
        }
    }
    0 referencias
    protected override async void OnAppearing()
    {
        base.OnAppearing();
        await _viewModel.LoadSucursalesAsync();
        var geolocalizacion = new GeolocationRequest(GeolocationAccuracy.High, TimeSpan.FromSeconds(1));
        var ubicacion = await Geolocation.GetLocationAsync(geolocalizacion);
        SucursalesMap.MoveToRegion(MapSpan.FromCenterAndRadius(ubicacion, Distance.FromKilometers(2)));
    }
}
```

Figura 2.6.2. Sucursales.cs.

3. Resultados y discusión

En este proyecto se desarrolló una aplicación móvil orientada a la búsqueda y gestión de películas, implementada en .NET MAUI. La aplicación permite a los usuarios explorar películas populares, buscar títulos específicos, ver detalles de cada película, reproducir trailers y gestionar una lista de películas favoritas. La estructura de la aplicación incluye varias páginas principales, cada una con funcionalidades específicas que se detallan a continuación.

La aplicación fue diseñada con una interfaz intuitiva, utilizando un enfoque modular para facilitar la mantenibilidad y escalabilidad del código. Cada funcionalidad clave está respaldada por un ViewModel que maneja la lógica de la interfaz de usuario, siguiendo el patrón MVVM (Model-View-ViewModel). La persistencia de datos se maneja a través de servicios que interactúan con la API de The Movie Database (TMDb), asegurando que la información mostrada esté siempre actualizada.

El desarrollo del proyecto incluyó la implementación de varias características como la búsqueda de películas, la visualización de detalles y trailers, y la capacidad de agregar películas a una lista de favoritos. Cada una de estas funcionalidades se presenta en su respectiva pantalla dentro de la aplicación, lo que proporciona una experiencia de usuario coherente y centrada en la interacción.

3.1. Login y Creación de usuario

La pantalla de Login y Creación de Usuario es la puerta de entrada principal a la aplicación, diseñada para proporcionar a los usuarios una experiencia de acceso personalizada y segura. Este componente es crucial tanto para los nuevos usuarios como para aquellos que ya han registrado sus credenciales, permitiendo el acceso a todas las funcionalidades del sistema.

La interfaz de usuario en esta pantalla presenta un diseño moderno y limpio, con campos de entrada claramente etiquetados para el correo electrónico y la contraseña. El botón de inicio de sesión se destaca visualmente, lo que facilita su identificación inmediata. Debajo del formulario de inicio de sesión, se encuentra un enlace para la creación de un nuevo usuario, así como un botón que permite el acceso con huella dactilar.

En cuanto a la funcionalidad de login, el proceso es sencillo. El usuario debe introducir su correo electrónico y la contraseña que ha registrado previamente. Al hacer clic en el botón "Iniciar Sesión", la aplicación se encarga de validar las credenciales contra la base de datos. Si las credenciales son correctas, el usuario es redirigido a la pantalla principal de la aplicación. En caso contrario, se muestra un mensaje de error que indica la necesidad de verificar los datos ingresados.

Para los usuarios que necesitan crear una nueva cuenta, la aplicación ofrece una opción de "Crear Usuario". Al seleccionar esta opción, el usuario es llevado a un formulario donde se

le solicita que proporcione información básica, como su nombre completo, correo electrónico y una contraseña. El sistema se encarga de validar que el correo electrónico no esté previamente registrado y que la contraseña cumpla con los criterios de seguridad establecidos, como la longitud mínima y la complejidad. Una vez completado el proceso de registro, el usuario es redirigido automáticamente a la pantalla de inicio de sesión, donde puede acceder a la aplicación utilizando las credenciales recién creadas.

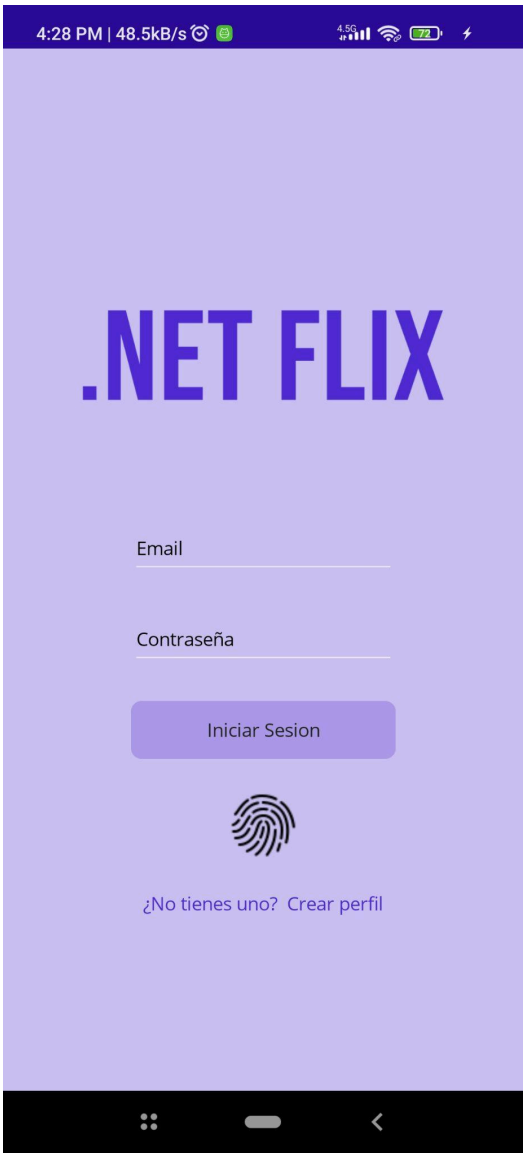


Figura 3.1.1 Login.

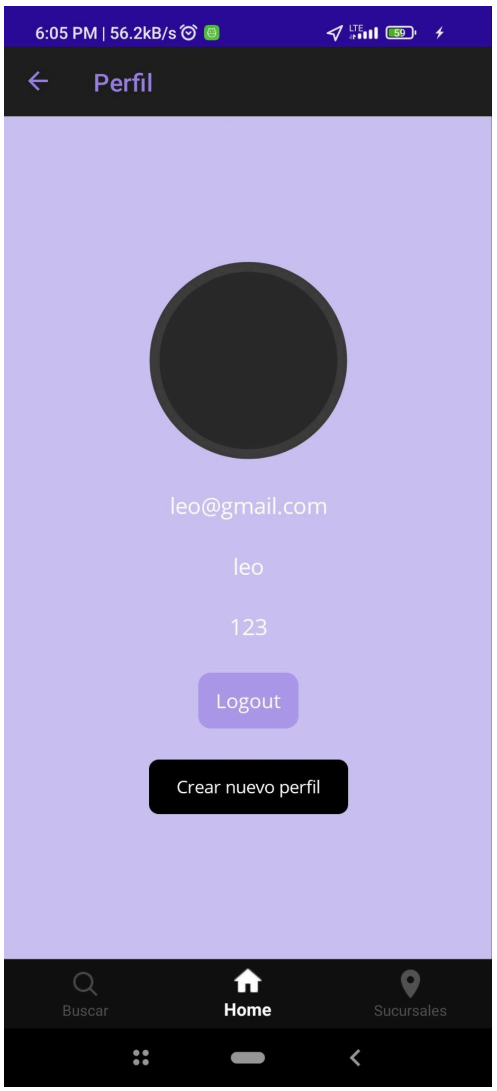


Figura 3.1.2 Crear Perfil.

3.2. Main y Detalles

En la página principal de la aplicación, se destaca una película sugerida cuya selección alterna aleatoriamente, ofreciendo al usuario una experiencia dinámica y atractiva. Justo debajo de la imagen de la película, se presenta un botón que permite acceder a los detalles completos de la película mostrada. Más abajo, se muestra una grilla que contiene listas de películas categorizadas según su valoración, popularidad, tendencias actuales y próximos estrenos, ofreciendo al usuario una navegación intuitiva y personalizada a través del contenido más destacado de la aplicación.

Al seleccionar una película, se despliega una pantalla detallada que permite al usuario ver el tráiler principal de la película junto con información clave como la fecha de estreno, duración, presupuesto, ganancias, y si contiene contenido para adultos. Además, se proporciona una sinopsis completa de la trama, lo que permite al usuario conocer más sobre la película antes de decidir verla. También se muestra el elenco completo, destacando a los actores principales. La interfaz ofrece la opción de acceder a más videos y tráilers relacionados si están disponibles, así como a títulos similares al seleccionado, brindando al usuario una experiencia de exploración más rica y personalizada.

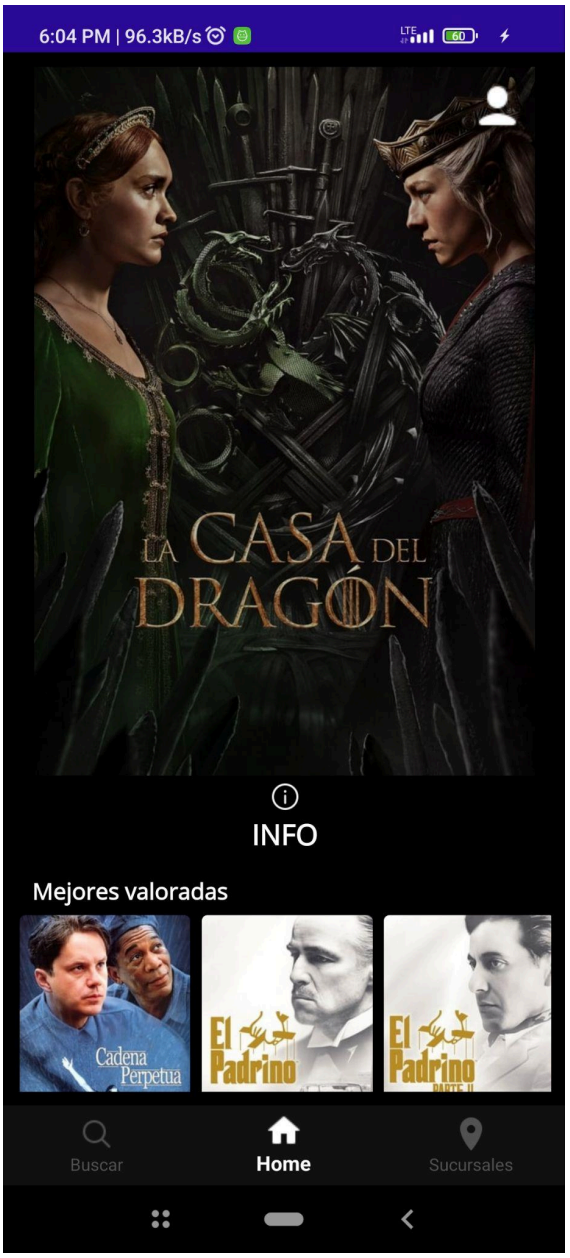


Figura 3.2.1 Main.

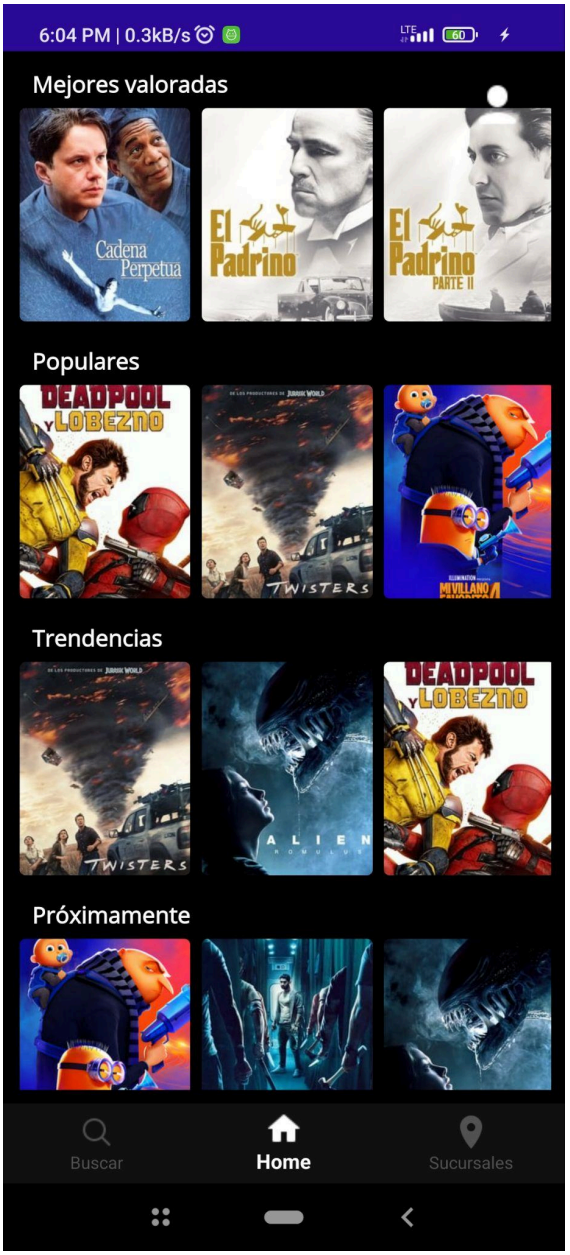


Figura 3.2.2 Main - Se despliegan carruseles con las llamadas de la api para las listas creadas nativamente por la misma. Most Rated, Populars, Upcoming y Trendings.

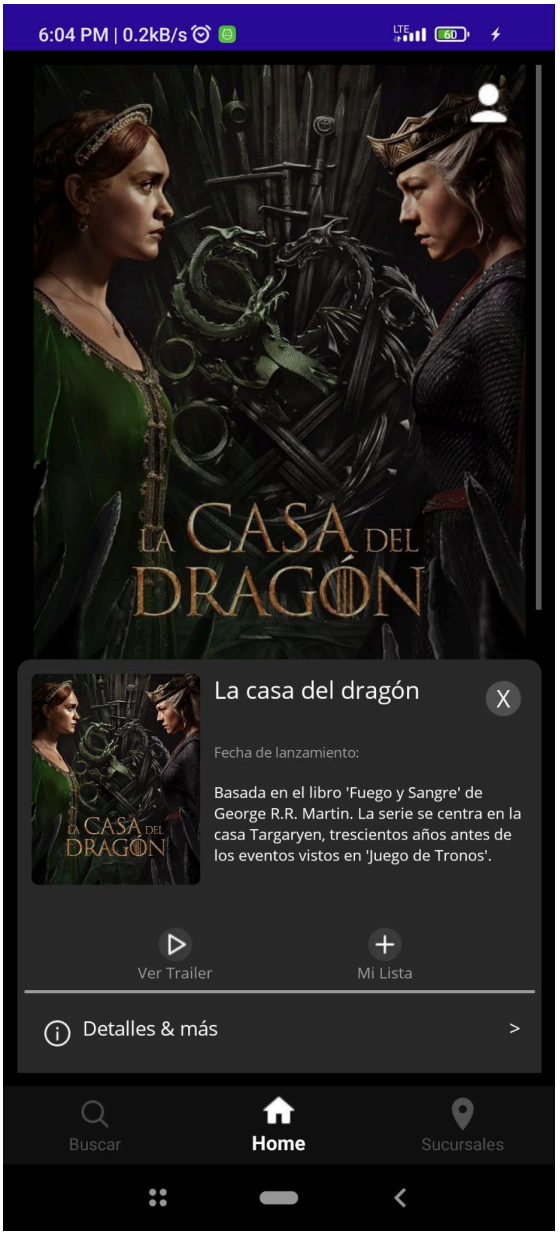


Figura 3.2.3 Main y Detalles - Se muestra la sinópsis de la película seleccionada y la posibilidad de acceder a trailers y detalles con los botones disponibles.

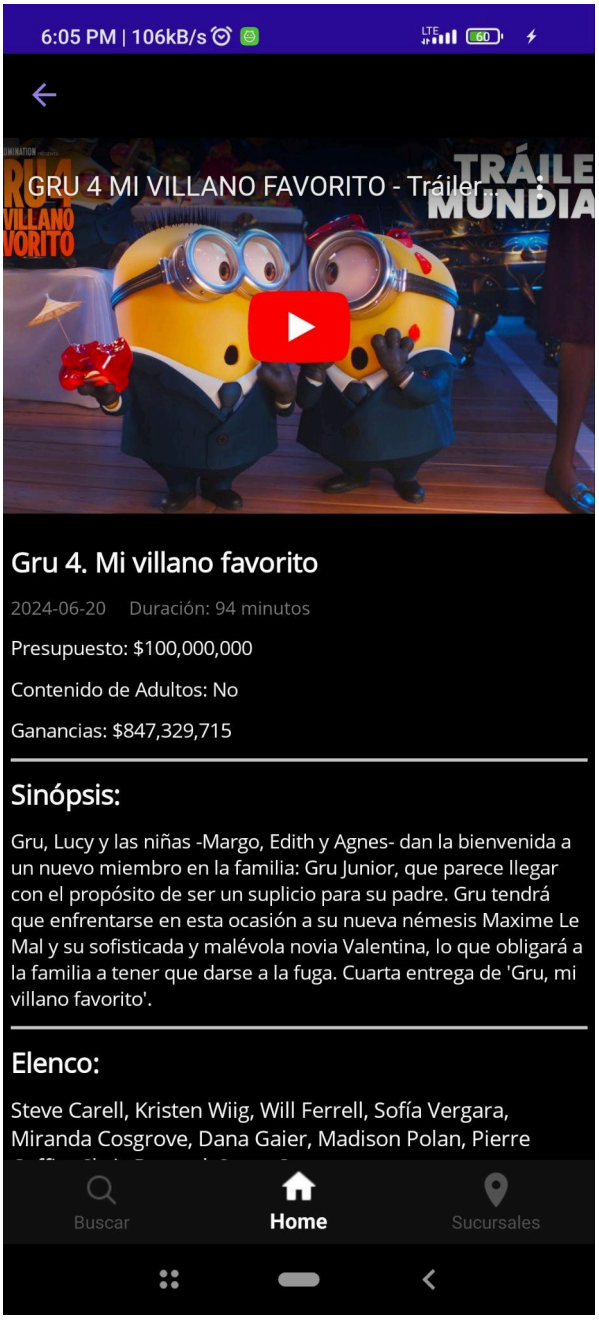


Figura 3.2.4 Detalles - Se muestra el trailer y datos generales.

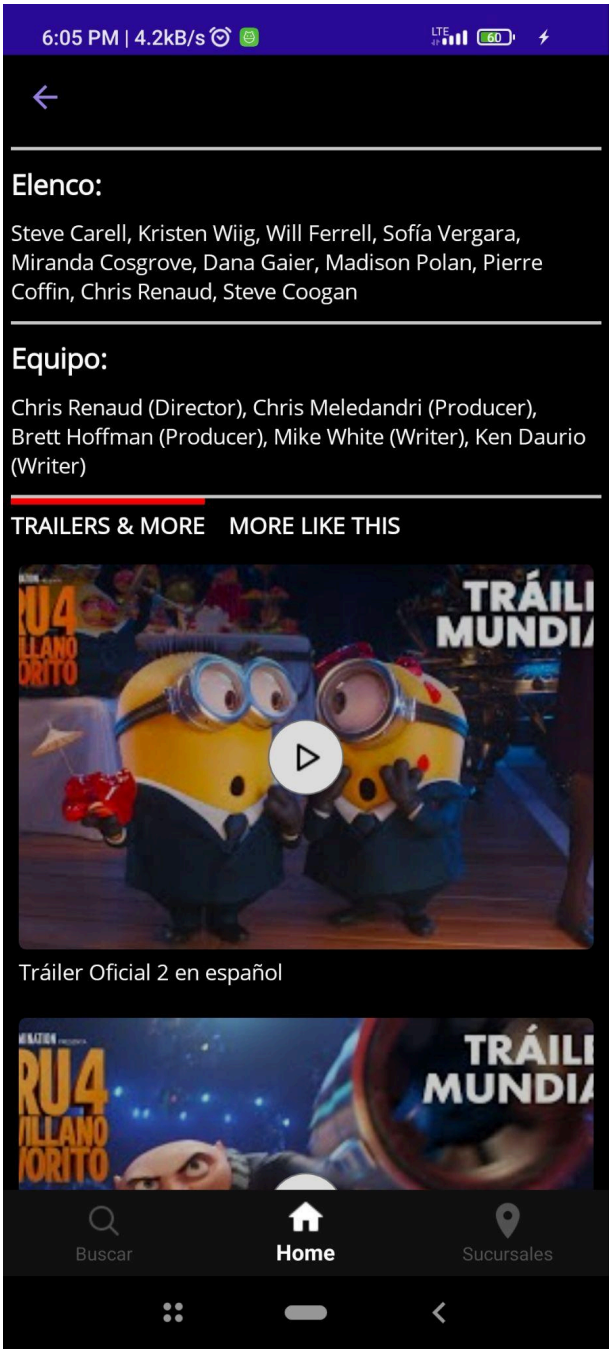


Figura 3.2.5 Detalles.

3.3. Buscar

La página de búsqueda ofrece al usuario la capacidad de encontrar rápidamente películas específicas utilizando una barra de búsqueda intuitiva. Al ingresar el título de una película en la barra de búsqueda, el sistema realiza una búsqueda en tiempo real dentro de la base de datos, mostrando los resultados pertinentes a medida que el usuario escribe. Los resultados de la búsqueda se presentan en una cuadrícula visualmente atractiva, con las carátulas de las películas correspondientes al título buscado. Esta función no solo facilita la localización de películas específicas, sino que también permite al usuario descubrir nuevas películas relacionadas o versiones diferentes del título buscado, optimizando la experiencia de navegación dentro de la aplicación. Además, el diseño de la interfaz asegura que la

búsqueda sea rápida y eficiente, con un enfoque en la usabilidad y la experiencia del usuario.

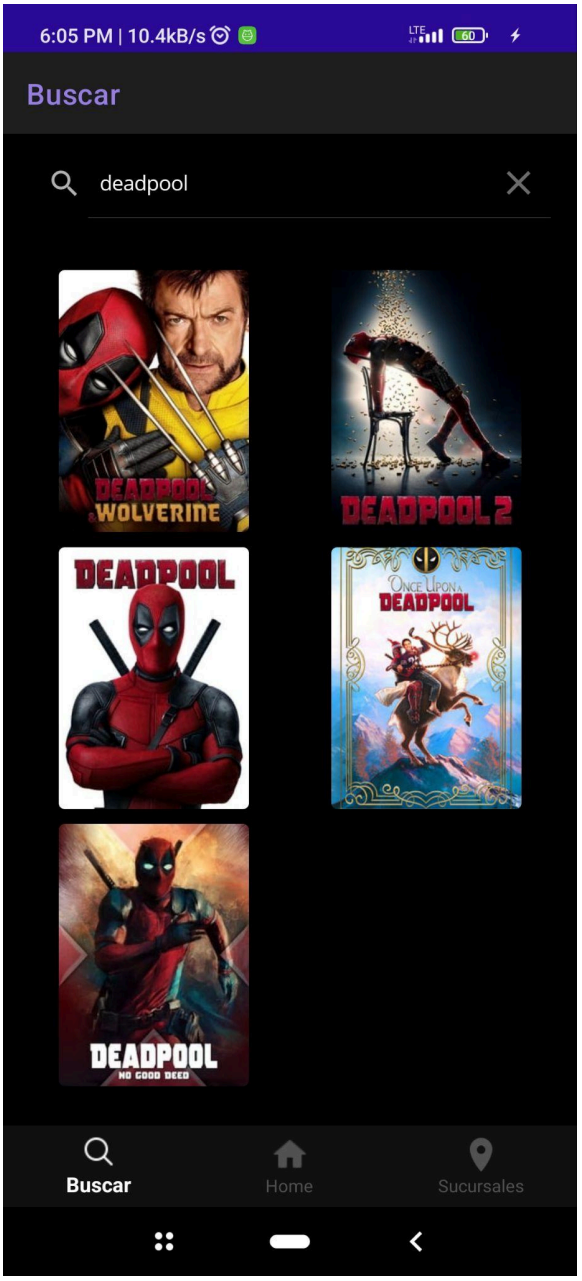


Figura 3.3.1 Buscar.

3.4. Sucursales y Mapa

La página de Sucursales en la aplicación ofrece una vista completa y funcional de todas las ubicaciones de las sucursales. Al ingresar, los usuarios son recibidos por un mapa interactivo que muestra las sucursales mediante pins, lo que facilita la identificación y la interacción con cada ubicación. Además del mapa, la página incluye dos botones clave: uno para agregar nuevas sucursales, que al ser seleccionado permite ingresar los detalles de la sucursal y añadirla al mapa, y otro para mostrar u ocultar la lista de sucursales, permitiendo al usuario centrarse en el mapa o revisar las sucursales de forma detallada.

La lista de sucursales, que se encuentra debajo del mapa y los botones, proporciona información básica como nombre, dirección y teléfono de cada sucursal, con la opción de eliminar cualquier sucursal, lo que también elimina automáticamente el pin correspondiente del mapa. Una característica avanzada de la página es la posibilidad de calcular y mostrar la ruta más corta desde la ubicación actual del usuario hasta una sucursal seleccionada, ofreciendo así una guía visual directamente en el mapa.

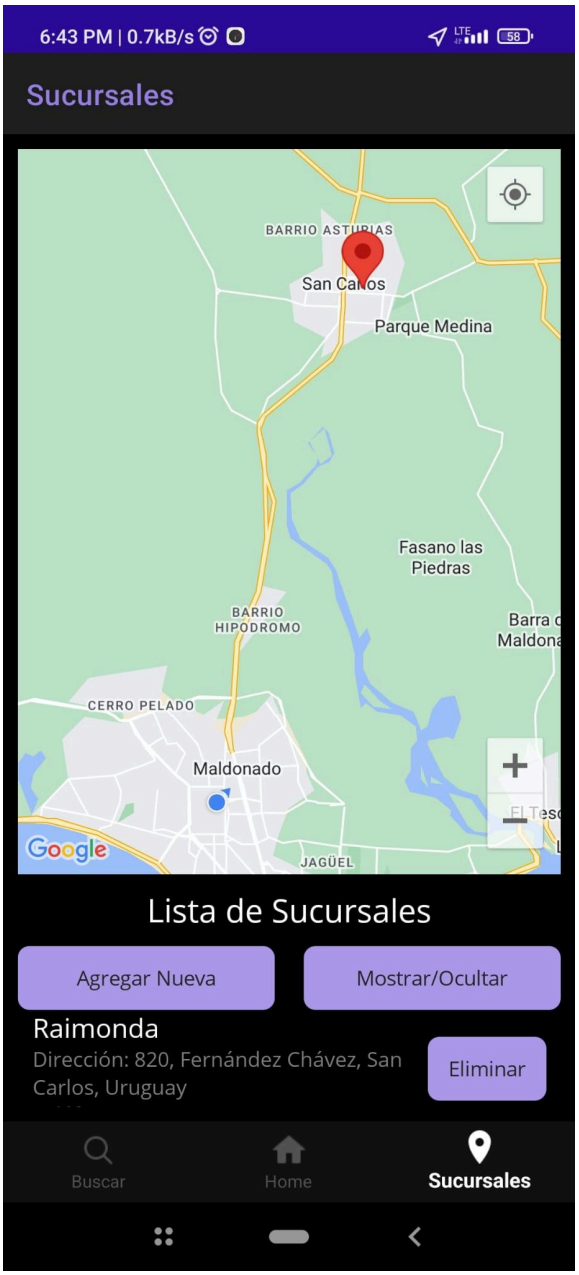


Figura 3.4.1 Sucursales y Mapa.

4. Conclusiones

El desarrollo de este proyecto nos brindó una valiosa oportunidad para aprender y aplicar nuevas tecnologías y metodologías de trabajo, tales como .NET MAUI, MVVM y el consumo de APIs, especialmente en el contexto del manejo de mapas y la integración de datos externos. La oportunidad de centrar el proyecto en el consumo de la API de películas

nos permitió estructurar el sistema de manera eficiente y enfocar nuestras funcionalidades clave desde una etapa temprana. Esto nos permitió implementar un diseño coherente y una lógica de negocio robusta en la aplicación, facilitando una experiencia de usuario óptima tanto en plataformas móviles como de escritorio.

A lo largo del proyecto, implementamos diversas funcionalidades esenciales, como la gestión de sucursales y usuarios, la autenticación mediante huella dactilar en Android y por correo electrónico y contraseña en Windows, así como la visualización de rutas en mapas interactivos. La capacidad de agregar y eliminar sucursales, visualizar su ubicación en un mapa y calcular la ruta desde la ubicación del usuario hasta un pin seleccionado, demostró ser un componente clave de la solución. Sin embargo, una mejora futura que identificamos es la posibilidad de optimizar la interfaz de usuario para mejorar aún más la usabilidad en diferentes dispositivos y contextos.

Consideramos que la aplicación desarrollada cumple con todos los requisitos planteados y ofrece una experiencia de usuario satisfactoria. La integración sólida de la API de películas y el manejo eficiente de mapas aseguran que el sistema sea funcional y atractivo para los usuarios finales. Además, la implementación de un sistema de autenticación seguro, junto con la posibilidad de gestionar sucursales de manera intuitiva, hace que esta aplicación sea una solución viable para su uso en un entorno real.

Este proyecto nos permitió crecer y ampliar nuestros conocimientos en el desarrollo de aplicaciones móviles y de escritorio complejas. La implementación de tecnologías modernas y la planificación meticulosa desde el inicio nos permitieron crear una solución integral que no solo cumple con los objetivos iniciales, sino que también ofrece oportunidades para futuras mejoras y expansiones. Este aprendizaje nos prepara mejor para enfrentar desafíos similares en el futuro y continuar desarrollando nuestras habilidades en el ámbito de la programación y la gestión de proyectos de software.

5. Referencias

[1]. POO, Clases y Objetos
[2]. .NET MAUI
[3]. SQLite
[4]. API
[5]. API REST
[6]. API DE TMDB