

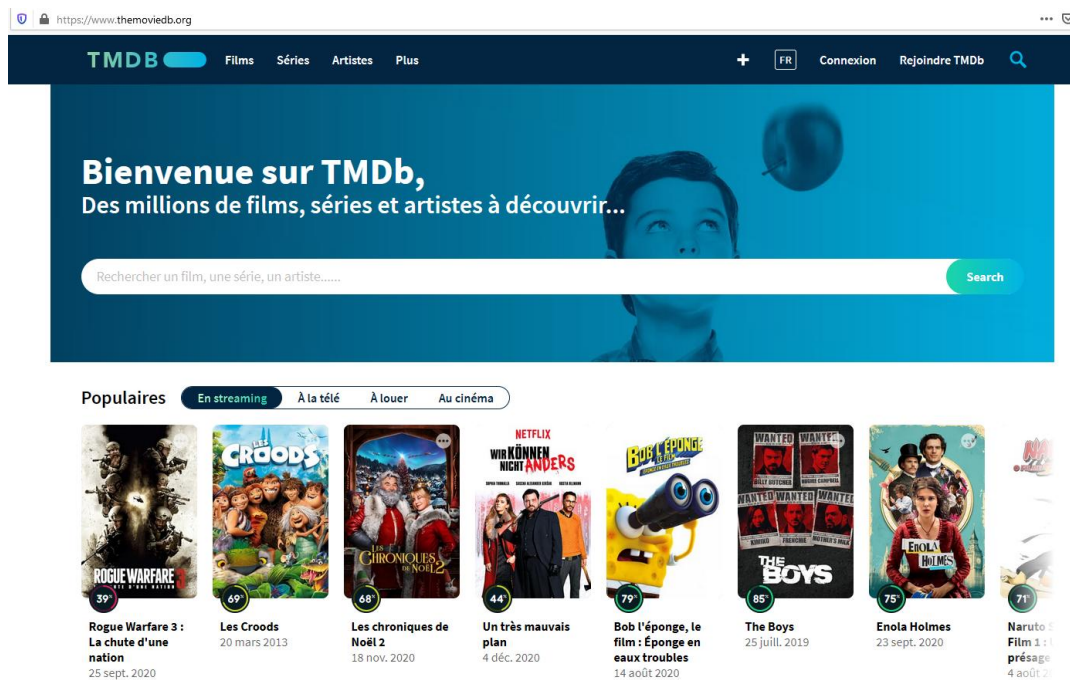
Le Besoin

"Mais si, tu sais, c'est ce film avec cet acteur et cette actrice qu'on aime bien. Je crois que ça se passe à New York. Attends je regarde sur Internet."

Il s'agit de proposer une application Android simple à utiliser, où il suffira de taper quelques mots-clés qui décrivent le film pour que le moteur de recherche retrouve le titre et un descriptif en question.

Cette application va s'appuyer sur la base de données communautaire de films « The Movie Database ». Le projet a été lancé en 2008 pour garder une trace d'une collection d'affiches de films. La base de données originale était basée sur les données du projet gratuit « Open Media Database » (omdb).

<https://www.themoviedb.org>



Le site web de TMDB propose un ensemble d'API, accessibles aux développeurs à condition de créer un compte utilisateur, ce qui permet d'obtenir une clé d'API. Les API constituent un service destiné aux développeurs qui souhaitent utiliser les images et / ou données de films, d'émissions de télévision ou d'acteurs dans leurs applications.

<https://www.themoviedb.org/documentation/api>

Questions fréquentes

- Notre histoire
- Restons en contact
- Logos et attribution
- Général
- Compte
- Site internet
- [Vue d'ensemble de l'API](#)
- Exemples d'API
- Sessions d'API
- Codes des statuts de l'API
- Bibliothèques de l'API

API Overview

Our API is available for everyone to use. A TMdb user account is required to request an API key. Professional users are approved on a per application basis.

As always, you must attribute TMdb as the source of your data. Please be sure to read more about this [here](#).

API Documentation

To view all the methods available, you should head over to developers.themoviedb.org. Everything outlined on this page is simply a high level overview to help you understand what is available.

What is TMDB's API?

The API service is for those of you interested in using our movie, TV show or actor images and/or data in your application. Our API is a system we provide for you and your team to programmatically fetch and use our data and/or images.

Why would I need an API?

The API provides a fast, consistent and reliable way to get third party data.

What is the difference between a commercial API and a developer API?

A commercial API is for commercial projects and a developer API is for developers. Your project is considered commercial if the primary purpose is to create revenue for the benefit of the owner.

Discover API Examples

Our discover methods can be powerful tools when you're looking for specific data. Below are some examples to showcase some of the powerful things you can do.

What movies are in theatres?

URL: `/discover/movie?primary_release_date.gte=2014-09-15&primary_release_date.lte=2014-10-22`

What are the most popular movies?

URL: `/discover/movie?sort_by=popularity.desc`

What are the highest rated movies rated R?

URL: `/discover/movie/?certification_country=US&certification=R&sort_by=vote_average.desc`

What are the most popular kids movies?


URL: `/discover/movie?certification_country=US&certification.lte=G&sort_by=popularity.desc`

What is are the best movies from 2010?

URL: `/discover/movie?primary_release_year=2010&sort_by=vote_average.desc`

L'ensemble de la documentation est disponible ici : <https://developers.themoviedb.org/3/getting-started/introduction>

https://developers.themoviedb.org/3/getting-started/introduction

**The Movie Database API**
https://api.themoviedb.org/3

OAS RAML Support

Select a different version
Filter sections...

GETTING STARTED

Introduction

Authentication

Daily File Exports

Languages

Images

Image Languages

Regions

External IDs

Popularity

Request Rate Limiting

JSON & JSONP

Append To Response

Search & Query For Details

ACCOUNT

AUTHENTICATION

CERTIFICATIONS

CHANGES

COLLECTIONS

COMPANIES

CONFIGURATION

CREDITS

DISCOVER

FIND

Getting Started

Introduction

Welcome to version 3 of The Movie Database (TMDb) API. Below you will find a current list of the available methods on our movie, tv, actor and image API. If you need help or support, please head over to our [API support forum](#).

To register for an API key, click the [API link](#) from within your account settings page. You can also view the screenshots below for help:

1. Click on your avatar or initials in the main navigation ([screenshot](#))
2. Click the "Settings" link ([screenshot](#))
3. Click the "API" link in the left sidebar ([screenshot](#))
4. Click "Create" or "click here" on the API page ([screenshot](#))

Please note that the API registration process is *not optimized* for mobile devices so you should access these pages on a desktop computer and browser.

Before being issued an API key you will have to agree to our terms of use. You can read that [here](#).

A few useful tips...

- The [configuration methods](#) are useful to get the static lists of data we use throughout the database. You can find things like the languages, countries, timezones and translations that we use. The configuration method also holds useful image information.
- Understanding the basics of our authentication is useful. You can read about this [here](#).
- We enforce rate limiting on the API. You can read about that [here](#).

JSON & JSONP

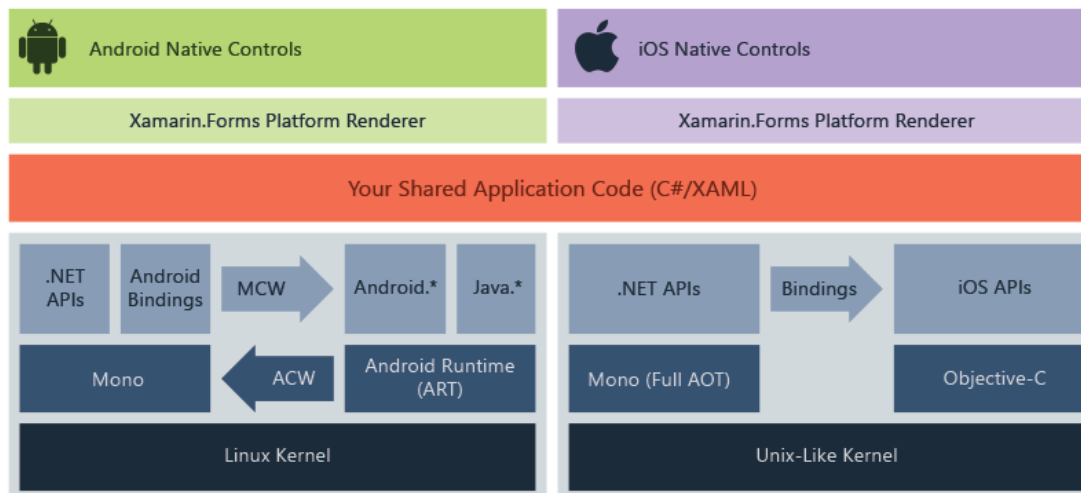
The only format we support is JSON. If you are using a JavaScript library and need to make requests from another public domain, you can use the `callback` paramater which will encapsulate the JSON response in a JavaScript function for you.

```
https://api.themoviedb.org/3/movie/550?api_key={api_key}&callback=test
```

L'application Android à développer

a) Environnement technologique

L'environnement de développement sera Xamarin.Forms, infrastructure de développement Open source. Xamarin.Forms permet aux développeurs de générer des applications Android, iOS et Windows Universel à partir d'un code unique : interfaces utilisateur en XAML avec « code-behind » en C# et prenant en charge la technologie de liaison de données « binding ».



L'accès aux API se font suivant l'architecture de Service Web, orientée ressource (REST) :

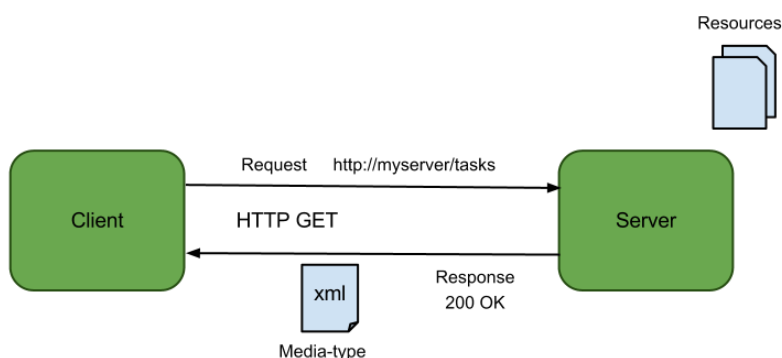
REST est un style d'architecture logicielle basé sur le constat que le protocole web HTTP fournit l'ensemble des méthodes (GET, POST, PUT, DELETE) permettant de manipuler des RESSOURCES présentes sur Internet :

- Des RESSOURCES DISTRIBUEES (les entités « METIER »), sont identifiées par leurs URI (« Uniform Resource Identifier ») et dotées de représentations concrètes (JSON, XML, texte).
- Avec REST, les ressources sont manipulées à distance via un ensemble prédéfini d'opérations (méthodes), principalement GET, PUT, DELETE et POST, que le programmeur utilise explicitement dans son code.

Par exemple, supposons que nous voulons réaliser un serveur REST pour gérer les livres d'une bibliothèque. Nous devons pouvoir ajouter (POST), modifier (PUT), Lire (GET) et Supprimer (DELETE) ces livres (la ressource à manipuler). Notre bibliothèque contient des ressources, en particulier des livres, qui pourront être manipulés à une URI formée par convention de la sorte : <http://bibliotheque/livre/>.

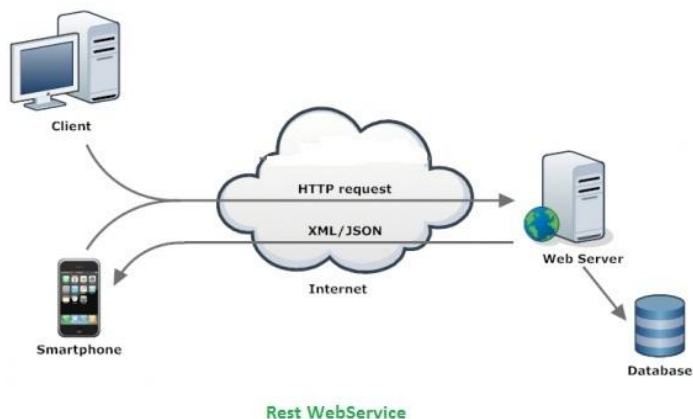
Nous pourrions effectuer plusieurs manipulations sur ces livres :

- Les LIRE : requête de type « GET » sur http://bibliotheque/livre/ID_DU_LIVRE_A_LIRE
- En ECRIRE : requête de type « POST » sur <http://bibliotheque/livre/>. Le corps du message « POST » représente le contenu du nouveau livre à créer. A la charge de la bibliothèque d'affecter un identifiant à notre nouveau livre.
- Les MODIFIER : requête de type « PUT » sur http://bibliotheque/livre/ID_DU_LIVRE. Le corps du message « PUT » représente le contenu modifié du livre d'identifiant ID_DU_LIVRE.
- Les SUPPRIMER : requête de type « DELETE » sur http://bibliotheque/livre/ID_DU_LIVRE.



Les formats d'échange

REST n'impose ni ne revendique un format d'échange entre client et serveur, mais de manière usuelle, on représente les données en XML ou en JSON (JavaScript Object Notation). JSON est un format de données textuelles. Un document JSON ne comprend que deux éléments structurels : des ensembles de paires nom / valeur et des listes ordonnées de valeurs.



```
{
  "fruits": [
    { "kiwis": 3,
      "mangues": 4,
      "pommes": null
    },
    { "panier": true }
  ],
  "legumes": {
    "patates": "amandine",
    "figues": "de barbarie",
    "poireaux": false
  }
};
```

b) Principe de fonctionnement attendu

Première Page

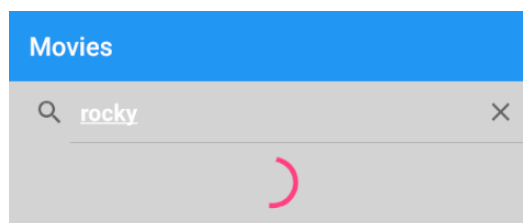
- L'utilisateur saisit des caractères correspondants à sa recherche. On peut utiliser une zone de texte « dédié », à savoir une `<SearchBar />`
- Aucune invocation au Service Web ne se fera tant que la saisie est vide ou trop petite (ex : 3 caractères minimum)



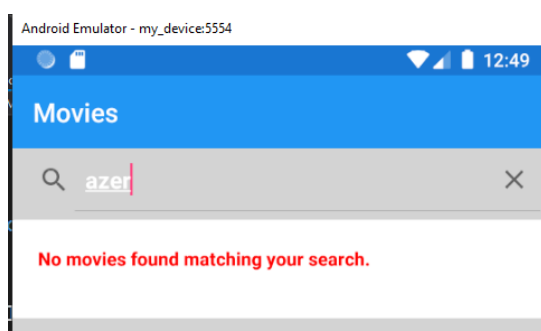
- L'affichage d'un contrôle visuel `<SearchBar />` pour indiquer que l'opération est en cours (invocation du Service Web) sera apprécié.



Exemple :

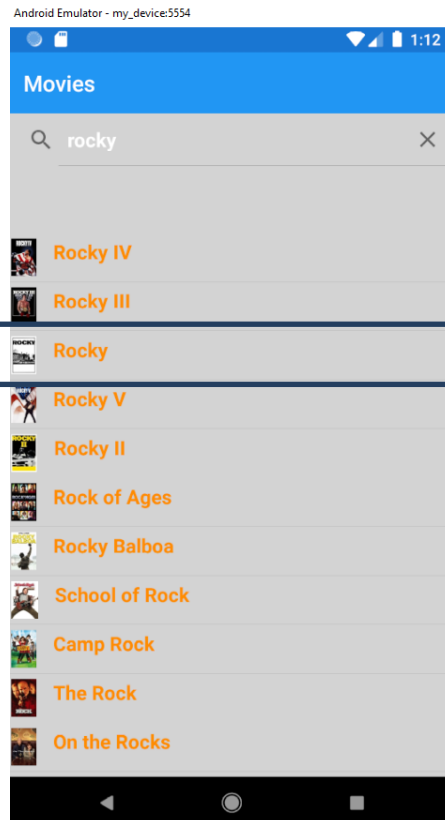


- Un message permettant d'informer l'utilisateur de l'absence de résultats correspondant à sa saisie : "No movies found matching your search." devra être également mise en œuvre. Ce message s'affichera dans un `<Label />` qui s'affichera dans une `<Frame />`



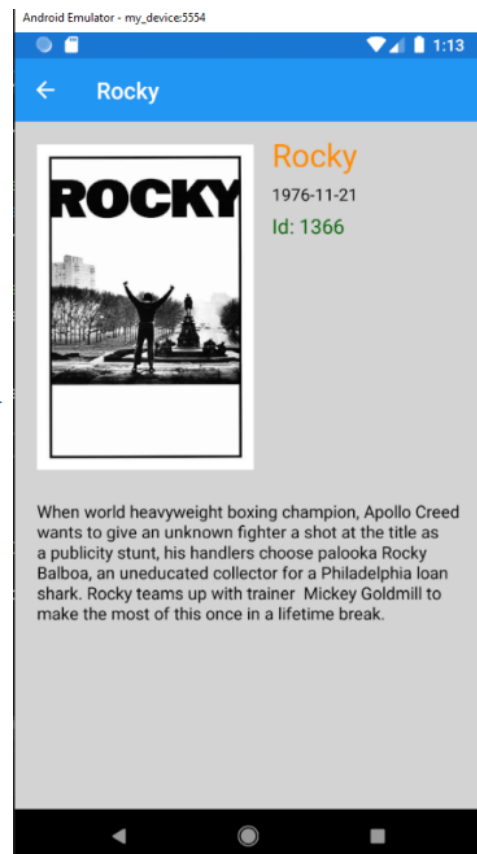
- L'application va alors afficher la liste des films correspondants dans la « listView » : titre et la photo. L'affichage se fera dans une `<ListView />` pour laquelle un « template » d'affichage devra être défini.

`<Image />` qui « binde » la propriété « poster_path »
`<Label />` qui « binde » la propriété « Title »



Deuxième Page

- L'utilisateur a ensuite la possibilité de cliquer sur un élément de la liste afin de consulter le détail du film sélectionné :
 - Photo de l'affiche en grand format
 - Titre du film
 - Date de parution (« ReleaseDate »)
 - Numéro (« Id »)
 - Résumé (« Overview »)



a) Classes correspondant à la ressource

```
// Architecture Service Web - Orienté ressource
//      * Classe RootObject (et classes imbriquées) correspondant à la ressource
//      * Désérialisation des données JSON
public class RootObject
{
    [JsonProperty("results")]
    public List<Movie> Results { get; set; }    // Propriété utilisée dans l'application

    [JsonProperty("page")]
    public int Page { get; set; }

    [JsonProperty("total_results")]
    public int TotalResults { get; set; }

    [JsonProperty("total_pages")]
    public int TotalPages { get; set; }
}

public class Movie
{
    [JsonProperty("id")]
    public int Id { get; set; }                // Utilisée

    [JsonProperty("title")]
    public string Title { get; set; }          // Utilisée

    [JsonProperty("popularity")]
    public double Popularity { get; set; }     // Utilisée

    [JsonProperty("poster_path")]
    public string PosterPath { get; set; }     // Utilisée

    [JsonProperty("overview")]
    public string Overview { get; set; }       // Utilisée

    [JsonProperty("release_date")]
    public string ReleaseDate { get; set; }    // Utilisée

    [JsonProperty("vote_count")]
    public int VoteCount { get; set; }

    [JsonProperty("video")]
    public bool Video { get; set; }

    [JsonProperty("vote_average")]
    public double VoteAverage { get; set; }

    [JsonProperty("original_language")]
    public string OriginalLanguage { get; set; }

    [JsonProperty("original_title")]
    public string OriginalTitle { get; set; }

    [JsonProperty("genre_ids")]
    public List<int> GenreIds { get; set; }

    [JsonProperty("backdrop_path")]
    public string BackdropPath { get; set; }

    [JsonProperty("adult")]
    public bool Adult { get; set; }
}
```

b) Utilisation de l'API

Exemple de requête : liste de films dont le titre contient la chaîne de recherche (« query »), ici « rock » :

https://api.themoviedb.org/3/search/movie?api_key=0f1d0b73d25595e9806aede52220a269&query=rock

```

▼ results:
  ▼ 0:
    adult: false
    backdrop_path: "/kFVeYpW0WPfYtINrh8JzRgoRzXw.jpg"
    ▼ genre_ids:
      0: 28
      1: 12
      2: 53
    id: 9802
    original_language: "en"
    original_title: "The Rock"
    ▼ overview:
      "FBI chemical warfare expert Stanley Goodspeed is sent on an urgent mission with a former British spy, John Patrick Mason, to stop Gen. Francis X. Hummel from launching chemical weapons on Alcatraz Island into San Francisco. Gen. Hummel demands $100 million in war reparations to be paid to the families of slain servicemen who died on covert operations. After their SEAL team is wiped out, Stanley and John deal with the soldiers on their own."
    popularity: 25.686
    poster_path: "/c6WuCykiY8GAsitJOTk1DEga1ML.jpg"
    release_date: "1996-06-07"
    title: "The Rock"
    video: false
    vote_average: 7.1
    vote_count: 3041

```

JSON	Données brutes	En-têtes
Enregistrer	Copier	Tout réduire
page:	1	
▶ results:	[...]	
total_pages:	138	
total_results:	2747	

Etc...

Si l'on veut obtenir un film avec une recherche exacte par son titre (exemple « rocky ») :

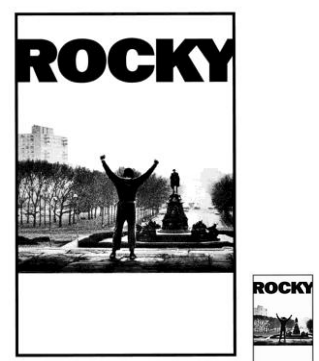
https://api.themoviedb.org/3/search/movie?api_key=0f1d0b73d25595e9806aede52220a269&query=rocky

Dans la liste de films, il suffit de récupérer le premier résultat :

```

▼ results:
  ▼ 0:
    adult: false
    backdrop_path: "/191kX9IGhVdLGq0dXT5HjXsRcez.jpg"
    ▼ genre_ids:
      0: 18
    id: 1366
    original_language: "en"
    original_title: "Rocky"
    ▼ overview:
      "When world heavyweight boxing champion, Apollo Creed wants to give an unknown fighter a shot at the title as a publicity stunt, his handlers choose palooka Rocky Balboa, an uneducated collector for a Philadelphia loan shark. Rocky teams up with trainer Mickey Goldmill to make the most of this once in a lifetime break."
    popularity: 43.656
    poster_path: "/i5xiwdSsrecBvO7mIfAJixeEDSg.jpg"
    release_date: "1976-11-21"
    title: "Rocky"
    video: false
    vote_average: 7.7
    vote_count: 5080

```



URL de chaque image

- Grand format : <https://image.tmdb.org/t/p/w500/i5xiwdSsrecBvO7mIfAJixeEDSg.jpg>
- Petit format : <https://image.tmdb.org/t/p/w92/i5xiwdSsrecBvO7mIfAJixeEDSg.jpg>

c) Classe TECHNIQUE de gestion du service web

Les URL à invoquer, avec la clé de l'API peuvent être placées dans des constantes :

```

private const string APIKEY = "0f1d0b73d25595e9806aede52220a269"; // Clé API
private readonly string MOVIE_URL = $"https://api.themoviedb.org/3/search/movie?api_key={APIKEY}";

```

Références à effectuer :

```

using System.Net.Http; using System.Threading.Tasks; using Newtonsoft.Json; using System.Net;

```


REMARQUE : un attribut statique « HttpClient » sera nécessaire pour gérer les échanges HTTP

Proposition de méthodes à écrire :

```
// Recherche TOUS les films dont le TITRE COMMENCE avec les caractères saisis par l'utilisateur
//      * Retourne un objet ressource RootObject
//      * Cette méthode permet de LISTER les films correspondants
//      * L'utilisateur pourra ensuite choisir un film pour voir le détail (cf. GetMovie())
public async Task<RootObject> FindMoviesByTitle(string search)

// Recherche du film dont le titre correspond strictement à la saisie utilisateur
//      * Cette méthode est appelée lorsque l'utilisateur CHOISI UN FILM dans la LISTE
//      * Retourne un objet ressource RootObject
public async Task<RootObject> GetMovie(string title)
```

REMARQUE : appels asynchrones. Le mot clé « await » devra être placé avant la demande http et la récupération de la réponse du serveur (=données sérialisées JSON).

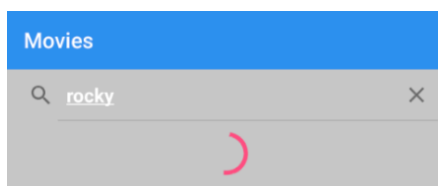
d) Page de contenu « MoviesPage » : recherche des films par mot clé

Structure XAML : une pile <StackLayout> englobant :

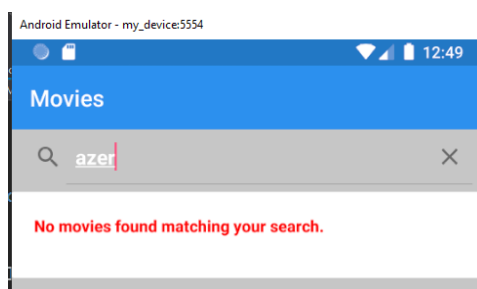
- Un contrôle qui fournit une zone de recherche : <SearchBar>. A chaque changement dans la zone de saisie (événement **TextChanged**), on devra exécuter la méthode (voir plus loin) :
`private async void SearchBar_TextChanged(object sender, TextChangedEventArgs e)`
- Contrôle visuel utilisé pour indiquer qu'une opération est en cours :
`<ActivityIndicator IsRunning="{Binding IsSearching}" />`

REMARQUE : « IsSearching » est une propriété « bindable » qui sera déclarée dans le code « .cs » associé. Sa valeur booléenne, sera positionnée à vrai avant l'invocation du service web (requête http pour obtenir la liste des films) n'est pas terminée, puis à faux après l'obtention de la liste des films.

```
private BindableProperty IsSearchingProperty = BindableProperty.Create("IsSearching",
                                                                    typeof(bool), typeof(MoviesPage), false);
public bool IsSearching
{
    get { return (bool)GetValue(IsSearchingProperty); }
    set { SetValue(IsSearchingProperty, value); }
}
```



Un cadre <Frame> englobant un contrôle d'affichage <Label> avec le texte : « No movies found matching your search. » Evidemment, le cadre aura au départ sa propriété « IsVisible » à faux, et ce sera dans le code C# qu'elle sera positionnée à vrai dans le cas où la recherche n'aboutit à aucun résultat.



Enfin, un contrôle <ListView> permettant d'afficher la liste des films retrouvés. Chaque élément pourra être affiché suivant le « template » suivant (une image réduite concernant l'affiche du film, puis le titre du film) :

```
<ViewCell>
    <StackLayout Orientation="Horizontal" Padding="5">
        <Image Source="{Binding PosterPath, StringFormat='https://image.tmdb.org/t/p/w92{0}}'" />
        <Label Text=" " />
        <Label Text="{Binding Title}" TextColor="DarkOrange" FontAttributes="Bold" FontSize="Medium"/>
    </StackLayout>
</ViewCell>
```



Code C# associé :

Il s'agit ici d'invoquer le service web à travers une requête http afin d'obtenir puis « binder » la liste des films obtenue. Puis à chaque sélection d'un élément de la liste, naviguer sur la deuxième page de contenu (= détail du film).

```
// A chaque nouveau caractère saisi
private async void SearchBar_TextChanged(object sender, TextChangedEventArgs e)

// Sélection d'un film de la liste
private async void MoviesListView_ItemSelected(object sender, SelectedItemChangedEventArgs e)
```

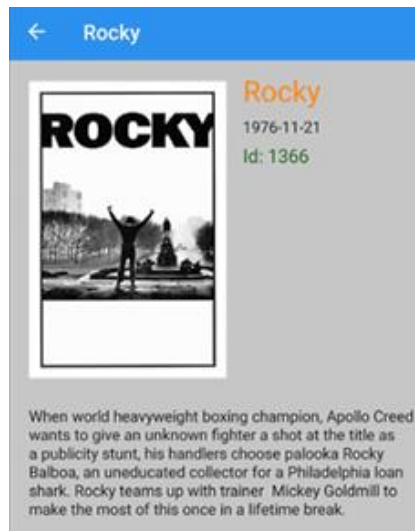
e) Page de contenu « MoviesDetailPage »: détail du film choisi

Structure XAML : page de contenu dont l'attribut Title sera le titre du film sélectionné, puis utilisant une pile <StackLayout> englobant :

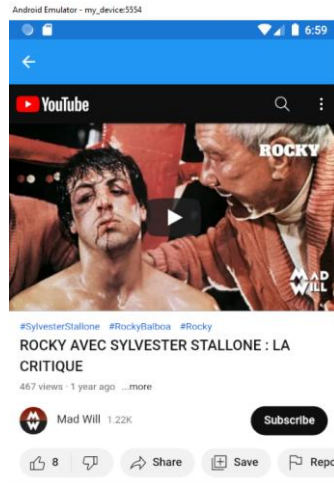
- Un contrôle <Image> affichant l'affiche du film, et donc effectuant le binding :
Source= "{Binding PosterPath, StringFormat='https://image.tmdb.org/t/p/w500{0}}'" />
- Des contrôles <Label> effectuant le binding respectif des propriétés : Title, ReleaseDate, Id et Overview

Code C# associé :

Il s'agira simplement d'invoquer le service web pour récupérer la ressource (le film) correspondant au titre sélectionné précédemment. Puis, on effectuera un binding de ses propriétés.



EVOLUTION - Affichage des vidéos YouTube pour le film sélectionné



Une balise XAML « ImageButton » supplémentaire permet d'afficher un « bouton PLAY » (fichier « .png » fourni) uniquement s'il existe au moins une vidéo pour le film sélectionné.

Un clic sur ce bouton permet d'accéder à une nouvelle page (« MoviesYouTube ») dans laquelle on pourra visualiser la première vidéo trouvée dans YouTube

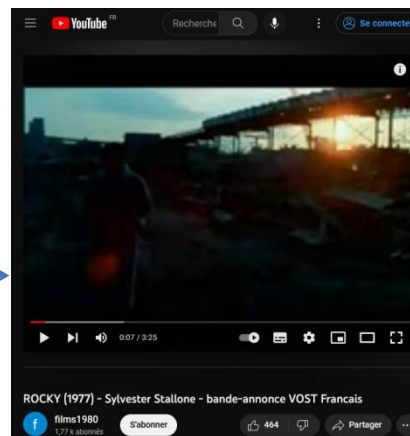
Une nouvelle page devra être ajoutée à l'application. Cette page va traiter l'URL YouTube correspondante à travers un contrôle XAML « WebView » :

```
<WebView x:Name="videoYoutube"
HeightRequest="1000"
WidthRequest="1000"
VerticalOptions="FillAndExpand"
HorizontalOptions="FillAndExpand"/>
```

Les URL à invoquer

https://api.themoviedb.org/3/movie/1366/videos?api_key=0f1d0b73d25595e9806aede52220a269&language=fr

```
id: 1366
▼ results:
  0:
    iso_639_1: "fr"
    iso_3166_1: "FR"
    name: "ROCKY : LA CRITIQUE CINÉ EN 3 MINUTES"
    key: "v78idDv3gfM"
    site: "YouTube"
    size: 1080
    type: "Featurette"
    official: false
    published_at: "2021-03-22T09:42:23.000Z"
    id: "60814bc8a1d3320029002520"
  1:
    iso_639_1: "fr"
    iso_3166_1: "FR"
    name: "ROCKY (1977) - Sylvester Stallone - bande-annonce VOST Français"
    key: "rnS20usFqYs"
    published_at: "2011-03-26T00:45:33.000Z"
    site: "YouTube"
    size: 360
    type: "Trailer"
    official: false
    id: "5a93e06ac3a3682529031a5c"
```



<https://www.youtube.com/watch?v=rnS20usFqYs>

Eléments techniques

« MoviesDetailPage.xaml » Il s'agira d'ajouter un bouton affichant l'image (.png)

```
<ImageButton x:Name="play" Source="play.png" BackgroundColor="Transparent" WidthRequest="50"
HeightRequest="50" Grid.Row="0" Scale="0.50" Clicked="ImageButton_Clicked"/>
```

« MoviesYouTube.cs » Son constructeur reçoit la CLE (=cf. clé JSON « key » de l'une des vidéos du film sélectionné) afin de formater l'URL et l'affecter à la propriété « Source » du composant « WebView ».

Classes ressources

Deux nouvelles classes seront à ajouter afin de désérialiser les données reçues à travers l'invocation de la nouvelle ressource :

https://api.themoviedb.org/3/movie/1366/videos?api_key=0f1d0b73d25595e9806aede52220a269&language=fr

public class RootMovieVideo

```
{
    [JsonProperty("results")]
    public List<MovieVideo> Results { get; set; }
}
```

public class MovieVideo

```
{
    [JsonProperty("key")]
    public string key { get; set; }
}
```