DOCUMENTATION Application Movies

Cette application mobile utilise un Web Service, déjà existant, pour lister les films.

Un fichier “**MoviesService.cs**” contient une classe :

Les attributs contiennent une clé d’API et deux URL utiles pour contacter le Web Service.

```
public static readonly int MIN_SEARCH_LENGTH = 3;
private const string APIKEY = "0f1d0b73d25595e9806aede52220a269";
private readonly string MOVIE_URL = $"https://api.themoviedb.org/3/search/movie?api_key={APIKEY}";
private readonly string MOVIE_VIDEO_URL = $"https://api.themoviedb.org/3/movie/";

private const string IMAGE_URL = "https://image.tmdb.org/t/p/w500/";

private static HttpClient _client = new HttpClient();
```

Trois méthodes s’y trouvent :

```
public async Task<RootObject> FindMoviesByTitle(string search)
{
    return search.Length >= MIN_SEARCH_LENGTH ? JsonConvert.DeserializeObject<RootObject>(
        await _client.GetStringAsync($"{this.MOVIE_URL}&query={search}")
        ) : new RootObject();
}
```

```
public async Task<RootObject> GetMovie(string title)
{
    return title.Length >= MIN_SEARCH_LENGTH ? JsonConvert.DeserializeObject<RootObject>(
        await _client.GetStringAsync($"{this.MOVIE_URL}&query={title}%20II")
        ) : new RootObject();
}
```

Ces deux méthodes renvoient un objet “RootObject” en task car la fonction est asynchrone. On contacte le WS avec l’url et on lui passe le paramètre voulu, il répondra avec un l’objet demandé en JSON, il faut donc le désérialiser.

Ici la fonction renvoie un objet contenant un ID utilisable pour les liens Youtube :

```
public async Task<RootMovieVideo> GetMovieVideoData(string id)
{
    return JsonConvert.DeserializeObject<RootMovieVideo>(
        await _client.GetStringAsync($"{MOVIE_VIDEO_URL}{id}/videos?api_key={APIKEY}&language=fr"));
}
```

On utilise ici, en ne prenant que le premier résultat et son ID

```
private async void LienYTB(int id)
{
    videoYoutube.Source = "https://www.youtube.com/watch?v="
        +
        ((RootMovieVideo)await this.WebS.GetMovieVideoData(id.ToString())).Results[0].key;
}
```

Ce WS est attribut de chaque page :

```
private MovieService WebS;
public MoviesDetailPage(Movie unFilm)
{
    InitializeComponent();

    // WebServices
    this.WebS = new MovieService();

    this.BindingContext = unFilm;
}
```

Et est instancié dans les constructeurs de celle-ci.

Enfin pour binder les informations d’une page XAML :

```
<StackLayout>
    <Label Text= "{Binding Title}"
        FontSize= "28"
        TextColor="DarkOrange"
        HorizontalOptions="Center" />
    <Label Text= "{Binding ReleaseDate}"
        FontSize= "15"
        TextColor= "Black" />
    <Label Text= "{Binding Id, StringFormat='Id: {0}'}"
        TextColor="DarkGreen"
        FontSize="Medium"
        />
</StackLayout>
```

Les noms des bindings correspondent au nom des attributs de la classe qui sera binder.

On peut aussi atteindre le XAML pour le modifier :

```
private async void SearchBar_TextChanged(object o, TextChangedEventArgs e)
{

    // WebService + Binding résultat
    this.moviesListView.ItemsSource = ((RootObject)await this.WebS.FindMoviesByTitle(((SearchBar)o).Text)).Results;
    try
    {
        if (((List<Movie>)this.moviesListView.ItemsSource).Count > 0)
            this.notFound.IsVisible = false;
        else
            this.notFound.IsVisible = true;
    }
    catch(Exception ex)
    { }
}
```

Le projet démarre avec une page de navigation, celle-ci permet à l'application de pouvoir créer d’autre page :

```
public App()
{
    InitializeComponent();

    // MainPage = new MainPage();
    MainPage = new NavigationPage(new MoviesPage());
}
```

En effet si l'on avait créé seulement "MoviePage" cette dernière n'aurait pas pu créer d'autres page, hormis des page enfant.

```
private async void MoviesListView_ItemSelected(object o, SelectedItemChangedEventArgs e)
{
    //await Navigation.PushModalAsync(new MoviesDetailPage((Movie)this.moviesListView.SelectedItem));
    await Navigation.PushAsync(new MoviesDetailPage((Movie)this.moviesListView.SelectedItem));
}
```

Ici on crée une nouvelle page et on l'affiche.

Classes Métiers :

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

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

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

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

```
// cf. dernière partie du sujet
public class RootMovieVideo
{
    [JsonProperty("results")]
    public List<MovieVideo> Results { get; set; }
}

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

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

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

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

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

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

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

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

```
    [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; }
}
```