# MovieApp - Film Világ

Kliensoldali technológiák házi feladat

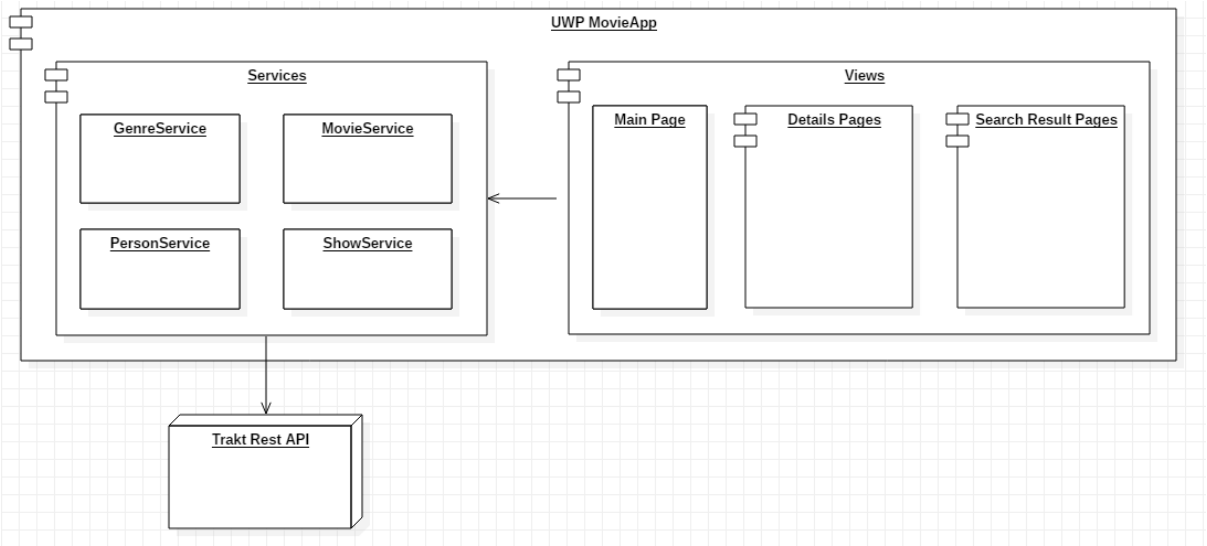
Készítette: Harmaci Marcell (V7BH6J)

## Összefoglaló

A MovieApp alkalmazás általános segítséget ad az esti filmvadászatban. Lehetővé teszi kulcsszavak alapján filmek, sorozatok és színészek keresését. A találati listában egy elemre kattintva pedig részletes információkat kaphatunk az adott elemről. Ezen kívül filmek esetén lehetőségünk van tovább böngészni annak szereplői között, míg sorozatok esetében megtekinthetjük milyen évadokkal és epizódokkal rendelkezik. A főoldalon a népszerű filmek közt nyílik lehetőségünk műfaj szerint nézelődni.

## Architektúra

Az alkalmazást UWP segítségével, MVVM minta használatával valósítottam meg és a Trakt publikus REST API-ját használtam backendként.
A rendszer felépítése a következő:




Az ábrát a StarUML programmal készítettem

# Komponensek

## Nézetek

- **MainPage:** Az alkalmazás főoldala
    - Keresősávban lehet keresni filmekre, sorozatokra és színészekre
    - Népszerű filmek listázása lapozással és műfaj szerinti szűréssel
- **Details Pages:** Részletes információk megjelenítése
    - MovieDetailsPage
        - Film részletei
        - Szereposztás
    - ShowDetailsPage
        - Sorozat részletei
        - Évadok és epizódok listázása
    - PersonDetailsPage - Színészek részletei
- **SearchResult Pages:** Keresési találatok listázása
    - SearchMoviesPage - Filmek listázása
    - SearchShowsPage - Sorozatok listázása
    - SearchPeoplePage - Színészek listázása

## Szolgáltatások

- **TraktService:** Általános, paraméterezhető Trakt API kérések megvalósítása
  Absztrakt osztály, ebből származik le az összes többi szolgáltatás.
- **GenreService:** Műfajokkal kapcsolatos kérések
- **MovieService:** Filmekkel kapcsolatos kérések
- **ShowService:** Sorozatokkal kapcsolatos kérések
- **PersonService:** Emberekkel kapcsolatos kérések

# Trakt API kommunikáció

A Trakt által biztosított REST API 2-es verzióját (https://trakt.docs.apiary.io) használom.
Mivel csak GET kéréseket és nem biztonságkritikus műveleteket végzek az API-on, ezért authorizációhoz elegendő volt a kérések fejlécében beállítani a kliens azonosítóm, amit string erőforrásként tároltam.

```
// Add header to the GET request
HttpRequestHeaders headers = httpClient.DefaultRequestHeaders;
headers.Add("ContentType", "application/json");
headers.Add("trakt-api-version", "2");
headers.Add("trakt-api-key", traktApiKey);
```

A REST kérések végrehajtásához a ***System.Net.Http.HttpClient*** osztályt használom és a kapott választ a ***Newtonsoft.Json.JsonConvert*** osztályával deszerializálom.

```
// Run GET method
var response = await httpClient.GetAsync(uri);
response.EnsureSuccessStatusCode();
json = await response.Content.ReadAsStringAsync();

// Convert json result to return type
T result = JsonConvert.DeserializeObject<T>(json);
```

# Filmek keresése

Ha kiválasztottuk a keresés típusát és beírtuk a keresési kulcsszót, akkor a *Search* gombra kattintva az alkalmazás kilistázza a keresési találatokat.

Gombnyomás után először a ViewModel-ben ellenőrzöm, hogy a nézet megfelelő állapotban van-e, majd ha igen megjelenik egy új ***SearchMoviesPage***, aminek átadom a keresési kulcsszavakat.

```
public void Search()
{
    if (ValidateString(SearchCategory) && ValidateString(SearchTerm))
    {
        switch (SearchCategory)
        {
            case "Movies":
                {
                    NavigationService.Navigate(typeof(SearchMoviesPage), SearchTerm);
                    break;
                }
```

A ***SearchMoviesPage*** indításakor egyből elküldöm a kérést, hogy megkapjuk a keresési eredményeket.

```
public override async Task OnNavigatedToAsync(
    object parameter,
    NavigationMode mode,
    IDictionary<string, object> state
)
{
    SearchTerm = (string)parameter;

    // Get search results
    var service = new MovieService();
    var result = await service.GetMovieSearchResultsAsync(searchTerm);
```

A kérést a szerverre a *search/movie?query=<keresési kulcsszó>* relatív címre küldöm. A válasz típusa ***MovieSearchResult*** lista. Ezt az osztályt a válasz json deszerializálása hoztam létre. Több információt tárol mint maga a film ezért, abból még készítek egy ***Movie*** listát, hogy a ViewModel-nek csak a nézethez tartozó modellen kelljen dolgoznia és ezt adom vissza.

```
public async Task<List<Movie>> GetMovieSearchResultsAsync(string searchTerm)
{
    string relativeUri = $"search/movie?query={searchTerm}*";
    Uri uri = new Uri(serverUrl, relativeUri);

    // Get search results from Trakt API
    var searchResults = await GetAsync<List<MovieSearchResult>>(uri);

    // Convert search results to a Movie list
    var movies = new List<Movie>();
    if (searchResults != null && searchResults.Count != 0)
    {
        foreach (MovieSearchResult result in searchResults)
        {
            movies.Add(result.Movie);
        }
    }

    return movies;
}
```

Ezt a listát a ViewModel-ben hozzáadom egy ObservableCollection-höz.

```
public ObservableCollection<Movie> Movies { get; set; } =
    new ObservableCollection<Movie>();
```

```
var result = await service.GetMovieSearchResultsAsync(searchTerm);

foreach(Movie movie in result)
{
    // Display new result
    Movies.Add(movie);
    // Update no result text visibility
    RaisePropertyChanged(() => NoResultTextVisibility);
}
```

A ***Movies*** listát pedig adatkötéssel a nézet listájához rendelem adatforrásként.

```
<ListView ItemsSource="{Binding Movies}"
          IsItemClickEnabled="True"
          ItemClick="MovieListItem_OnItemClick">
```