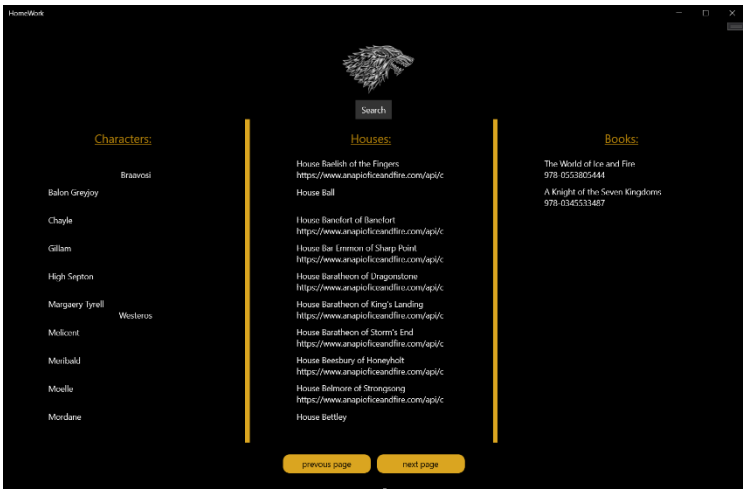
Game of Thrones

Kliensoldali technológiák házi feladat – Készítette: Kalmár Gábor (XCCGBS)

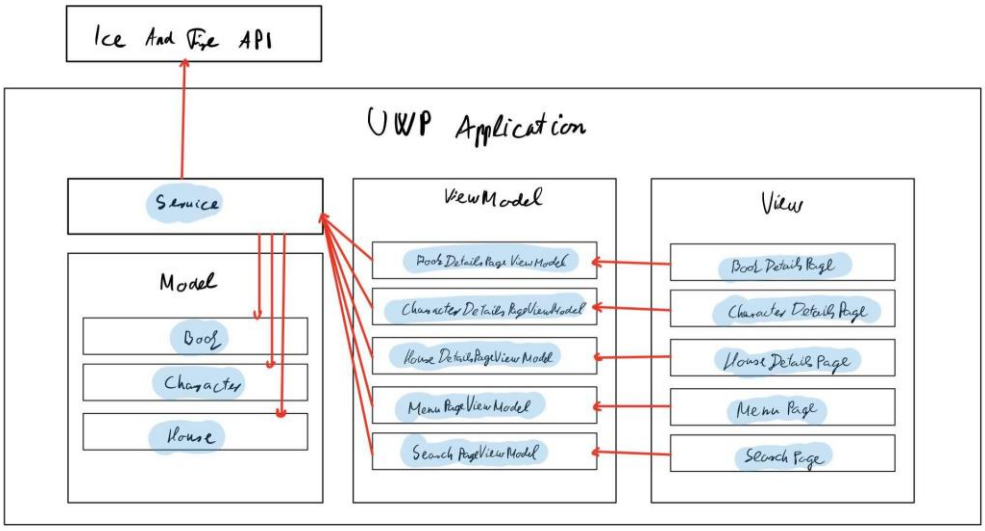
Összefoglaló



UWP technológiát használva készítettem el a Trónok Harca nevű alkalmazást, mely a könyvsorozat kiadásait és szereplőit mutatja be. Az alkalmazás az Ice And Fire API-t használta adatforrásként (https://anapioficeandfire.com/). A program segítségével a felhasználónak lehetősége nyílik interaktív módon bejárni a sorozatban megjelenő entitásokat (House, Character, Book) melyek a kezdőoldalon három oszlopban jelennek meg. A kezdőoldal egy lapozható felület, ahol alul beállítható az oldalanként megjelenő elemek száma. Az egyes elemekre kattintva a felhasználó megtekintheti az adott entitást részletező oldalt. A részletező felületen tovább lehet navigálni az adott entitással kapcsolatban lévő többi entitásokhoz a megjeleníteni kívánt elemre kattintva. A felhasználónak lehetősége van a programban szűrni, mely funkció a főoldalon lévő Search gombra kattintva érhető el. A keresésnél megadható, hogy milyen típusú elemre szűrünk (minden, karakter, család, könyv). A főoldalra a minden oldal alján megtalálható Home gombbal lehet visszanavigálni.

Architektúra

Az alkalmazást UWP technológiával valósítottam meg és az Ice and Fire publikus API-ját használtam adatforrásként. A programot a MVVM architektúra alapján terveztem meg. A rendszer felépítése és az elemek főbb kapcsolódási pontjai a következők:



Model osztályok

- **Book**: Ez az osztály tartalmazza egy könyv adatait.
- **Character**: Ez az osztály tartalmazza egy karakter adatait.
- **House**: Ez az osztály tartalmazza egy család adatait.

ViewModel osztályok

- **BookDetailsPageViewModel**: Az osztály tartalmaz egy Book property-t ami adatkötéssel lesz megjelenítve, valamint tartalmazza a más oldalakra való navigációt megvalósító függvényeket.
- **CharacterDetailsPageViewModel**: Az osztály tartalmaz egy Character property-t ami adatkötéssel lesz megjelenítve, valamint tartalmazza a más oldalakra való navigációt megvalósító függvényeket.
- **HouseDetailsPageViewModel**: Az osztály tartalmaz egy House property-t ami adatkötéssel lesz megjelenítve, valamint tartalmazza a más oldalakra való navigációt megvalósító függvényeket.
- **MenuPageViewModel**: Az osztály tartalmazza a `List<Book>, List<Character>, List<House>` property-ket, melyek adatkötéssel vannak megjelenítve. Az ablakon kiválasztható, hány elem jelenjen meg oldalanként (10,20,50), a korábban felsorolt listák maximum ennyi elemet tartalmaznak az adott objektumokból. Az osztály tartalmazza a más oldalakra való navigációt megvalósító függvényeket.
- **SearchPageViewModel**: Az osztály tartalmazza a `List<Book> books, List<Character> characters, List<House> houses` property-ket, melyek adatkötéssel vannak megjelenítve. A felsorolt listák tartalmazzák a keresési feltételeknek megfelelő elemeket. Az osztály ezeken kívül tartalmazza az összes trónok harca entitást, amiket az osztály létrehozásakor tölt be. Az osztály tartalmazza a keresést (search) valamint a keresés típusát (type) is, amik alapján szűr az összes entitás között. Végül az osztály tartalmazza a más oldalakra való navigációt megvalósító függvényeket.

View komponensek

- **BookDetailsPage**: Ez a komponens felel a könyv részletező ablak megjelenésért. Adatkötéssel megjeleníti a **BookDetailsPageViewModel** osztályban található könyv adatait. Az XAML felület mögött lévő kód felel az oldal megnyitásakor kapott paraméter átadásáért a viewmodel-nek, valamint a felületen való interakciók kezeléséért.
- **CharacterDetailsPage**: Ez a komponens felel a karakter részletező ablak megjelenésért. Adatkötéssel megjeleníti a **CharacterDetailsPageViewModel** osztályban található karakter adatait. Az XAML felület mögött lévő kód felel az oldal megnyitásakor kapott paraméter átadásáért a viewmodel-nek, valamint a felületen való interakciók kezeléséért.
- **HouseDetailsPage**: Ez a komponens felel az családrészletező ablak megjelenésért. Adatkötéssel megjeleníti a **HouseDetailsPageViewModel** osztályban található család adatait. Az XAML felület mögött lévő kód felel az oldal megnyitásakor kapott paraméter átadásáért a viewmodel-nek, valamint a felületen való interakciók kezeléséért.
- **MenuPage**: Ez a komponens felel az kezdőablak megjelenésért. Adatkötéssel megjeleníti a **MenuPageViewModel** osztályban található karakter, család, könyv listák 2 fontosabb adatát (Név, ISBN/Culture/CurrentLord). Az XAML felület mögött lévő kód felel a felületen való interakciók kezeléséért.

- **SearchPage**: Ez a komponens felel az keresési ablak megjelenésért. Adatkötéssel megjeleníti a **SearchPageViewModel** osztályban található szűrt listák fontosabb adatait a MenuPage-hez hasonlóan. Az XAML felület mögött lévő kód felel a felületen való interakciók kezeléséért.

Szolgáltatások

- **Service**: Az osztály tartalmazza az API-val való hálózati kommunikációt megvalósító függvényeket

Program működése, fontosabb komponensek

Az alkalmazás használja a platformspecifikus aszinkron mintákat és nyelvi elemeket, így a hosszabb időt igénylő funkció nem akasztja meg a felhasználói felületet, ezáltal az minden esetben reszponzív marad.

MenuPage

Az alkalmazást megnyitva ez az oldal tárul a felhasználók szeme elé. Az oldal felépülése során betölti az első 10-10 elemet az egyes entitástípusokból, majd ezeket jeleníti meg. Az oldal alján található gombokkal lehet előre és vissza navigálni a lapok között. Legalul állítható a laponként betöltődő elemek száma. Az első oldalnál eltűnik a vissza gomb, utolsó oldalnál az előre gomb. Az ablak tetején található search gombra kattintva megjelenik egy keresési rész, ami a cancel hatására eltűnik. Amennyiben kerestünk, akkor a SearchPage oldal nyílik meg, ahol megtekinthető a keresési eredmény.

DetailsPage

A részletező felületeken mindig egy adott entitás adatát jeleníti meg a program. Az egyes DetailsPage-ek közötti navigáció rendkívül egyszerű, a megjelenített entitás megfelelő relációjára kattintva megnyitható az adott DetailsPage.

SearchPage

A főoldalon a keresésre kattintva nyílik meg a SearchPage. Az oldal felépülése során betölti az összes elemet az egyes entitástípusokból, majd ezek egy szűrt halmazát jeleníti meg a keresési feltételeknek megfelelően. A betöltés miatt egy kis időt kell várni az első keresési eredményekre, de az oldalon maradva a további keresési eredmények már azonnal megjelennek. A keresési eredmények a főoldalhoz hasonlóan jelennek meg, 3 oszlopba rendezve. Az eredményként kapott elemekre rákattintva az adott entitás DetailsPage-e nyílik meg.

Ice And Fire API

Az 1. verziójú ICE And Fire API egy nyílt API ami azt jelenti, hogy nincs szükség hitelesítésre az adatok lekérdezéséhez. Mivel nincs szükség hitelesítésre, csak GET-kérések támogat. Az API rengeteg adatot szolgáltat Westeros világáról. Annak megakadályozására, hogy szerverek túlterhelődjenek, az API automatikusan oldalszámozza a válaszokat. A következő részben olvasható, hogyan lehet oldalszámozási paraméterekkel kérést indítani. A ?Page paraméterrel megadható, hogy melyik oldalt szeretnénk elérni. Amennyiben nincs megadva ?Page paraméter, akkor az első oldalt adja meg automatikusan. Az oldal mérete a ?PageSize paraméterrel adható meg. Amennyiben nincs megadva ?PageSize paraméter, akkor az alapértelmezett 10-es méret lesz használva. A ?PageSize paraméter maximális értéke 50.
Az API rosszindulatú használatának megakadályozása érdekében korlátozva van az adott IP-cím által az API-hoz intézett kérelmek száma. Ez a korlát napi 20000 kérést jelent. A határ túllépése esetén a következő 24 órában minden kérésre 403-as hibát ad az API.

```
private readonly Uri serverUrl = new Uri("https://www.anapioficeandfire.com");
private async Task<T> GetAsync<T>(Uri uri)
{
    using (var client = new HttpClient())
    {
        var response = await client.GetAsync(uri);
        var json = await response.Content.ReadAsStringAsync();
        T result = JsonConvert.DeserializeObject<T>(json);
        return result;
    }
}
```

Az API-val való kommunikációt a képen látható kód valósítja meg. A http kérést a HttpClient osztály segítségével indítottam.

Keresés

Az alábbiakban részletes leírás található a keresés funkció működéséről.

1. Főoldalon a keresés gombra kattintva megjelenik a kereső mező.
2. A Search Type-nál kiválasztható, hogy melyik kategóriára szeretnénk szűrni, a Search mezőbe pedig beírható a keresési kulcsszó.
3. A Search gombra kattintva meghívódik a viewmodel (MenuPageViewModel) NavigateToSearch() metódus, ami megnyitja a SearchPage oldalt és tovább adja a paraméterként kapott string[] – öt, ami a keresési feltétel.
4. A SearchPage-en Load() metódus betölti az adatokat, ami annyit jelent, hogy meghívja a Service osztályból példányosított service objektum GetAllCharactersAsync(), GetAllBooksAsync(), GetAllHousesAsync() metódusokat amik a megfelelő objektumok listájával térnek vissza.
5. Az egyik ilyen metódust mutatom be részletesebben.

```
public async Task<List<Book>> GetAllBooksAsync()
{
    int page = 1;
    List<Book> newBook = await GetBooksAsync(page, 50);
    List<Book> allbooks = new List<Book>();
    while (newBook.Count == 50)
    {
        for (int i = 0; i < newBook.Count; i++)
        {
            allbooks.Add(newBook[i]);
        }
        page++;
        newBook = await GetBooksAsync(page, 50);
    }

    for(int i = 0; i < newBook.Count; i++)
    {
        allbooks.Add(newBook[i]);
    }

    return allbooks;
}
```

6. A könyveket az API által megengedett 50-es limitszámban töltöm be, és adom hozzá az allbooks listához, egészen addig, amíg kevesebb elemet nem ad vissza a http kérés mint 50.
7. A meghívott GetBookAsync() függvény a könyvek listájával tér vissza. a kód az alábbi képen látható. Egy generikus metódus (GetAsync) segítségével adja vissza a listát.

```
//aszinkron metódus ami pagesize méretű, page oldalú <List<Book> ad vissza
public async Task<List<Book>> GetBooksAsync(int page, int pagesize)
{
    return await GetAsync<List<Book>>(new Uri(serverUrl, "api/books?page=" + page.ToString()+ "&pageSize=" + pagesize.ToString()));
}
```

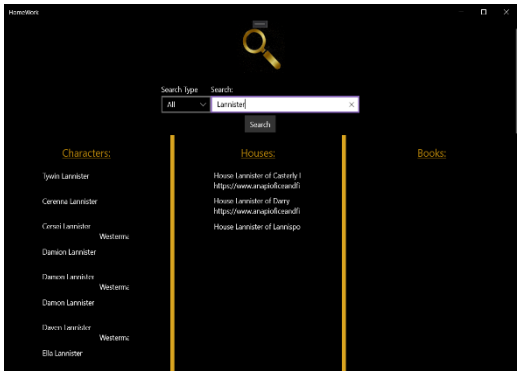
A Generikus metódus kódja a következő ábrán látható:

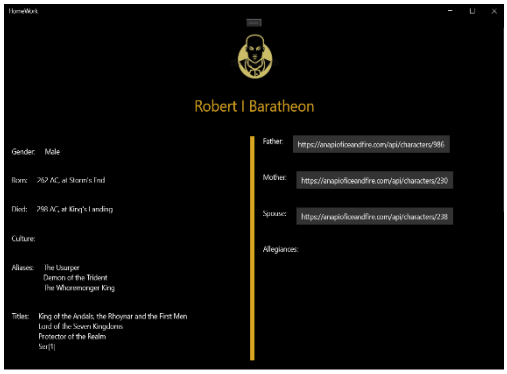
```
private async Task<T> GetAsync<T>(Uri uri)
{
    using (var client = new HttpClient())
    {
        var response = await client.GetAsync(uri);
        var json = await response.Content.ReadAsStringAsync();
        T result = JsonConvert.DeserializeObject<T>(json);
        return result;
    }
}
```

8. Az adatok betöltését követően elindul a szűrés, amit a Filter() függvény valósít meg.
9. A szűrt eredmény megjelenik az oldalon. Az egyes listaelemek rájuk kattintva megnyithatóak
10. További keresésre is van lehetőség az oldalon

Konklúzió és képernyőképek

A programban az egyes függvények működése a forrásban való részletes kommentezésnek köszönhetően könnyen megérthetőek.





Források

API részletesebb dokumentáció: https://anapioficeandfire.com/