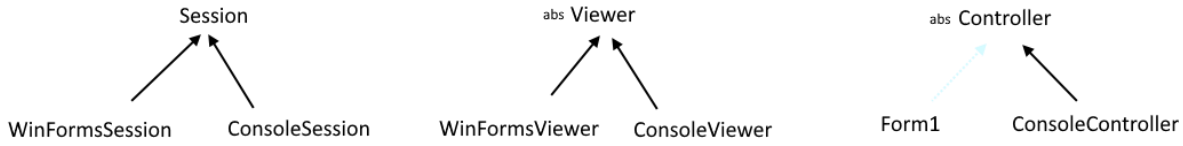
Programátorská dokumentace

Solution obsahuje 3 projekty. Jeden hlavní – *RecommendationSystemInterface* – který slouží jako knihovna pro ostatní dva (*WinFormsRecSys* a *ConsoleRecSys*), které se ji pouze snaží implementovat a rozšířit.

V programu jsem se snažil implementovat MVC pattern.

V každém projektu jsou hlavní 3 třídy rozšiřující / definující **Session** (M), **Viewer**, **Controller**. Napříč projekty to vypadá takto:



RecommendationSystemInterface

Obsahuje 6 hlavních souborů (na každý specifická třída a její rozšíření) a složku s 8 rozhraními a třídami, které od nich dědí.

6 hlavních souborů (mimo složku Interfaces)**:**

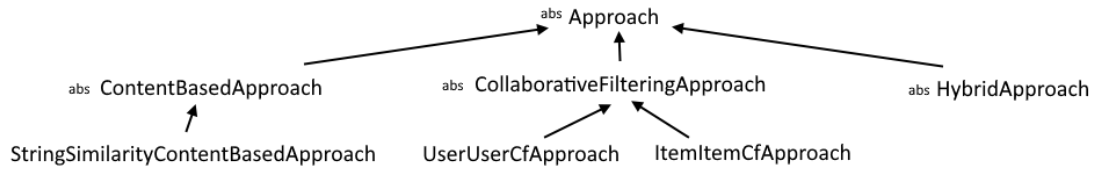
Hlavní tři třídy jsou **Session**, **Viewer** a **Controller**.

Session – Obsahuje a spravuje data programu, hlavní funkce – např. funkce *GetRecommendations()*, která rozběhne samotné doporučování. Dostává pokyny od Controlleru. Ukazuje si na Viewer, který dostane v konstruktoru.

Viewer – Odpovědný za vizuální výstup aplikace. Zde definován jako abstraktní třída. Obsahuje abstraktní funkce ViewFile, ViewString a definuje pomocnou funkci ReadFirstKLines, která vrátí string, který obsahuje prvních *k* řádků požadovaného souboru.

Controller – Odpovědný za úkolování Session, někdy také Viewer.

Hybatel programu je abstraktní třída **Approach**, která definuje způsob doporučování. Struktura:



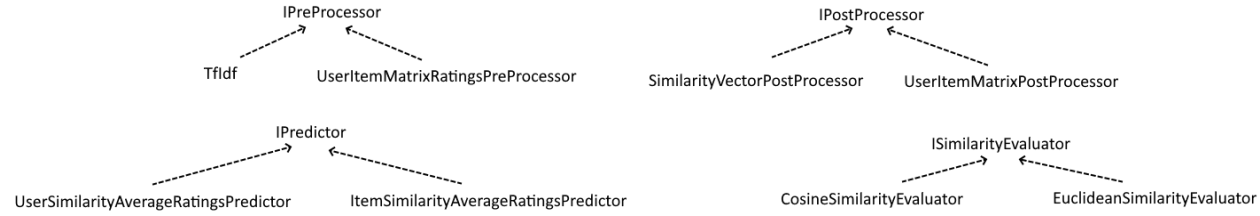
Obsahuje:

```
public IDisposableLineReader RecordReader { get; set; }
public IPreProcessor PreProcessor { get; set; }
public ISimilarityEvaluator Evaluator { get; set; }
public IPostProcessor PostProcessor { get; set; }

public abstract string Recommend();
```

Instance třídy, která dědí od *IDisposableLineReader* je hlavním nástrojem pro načítání dat v hrubé podobě. Na ni navazuje *PreProcessor*, který hrubá data zpracovává (většinou do jagged array). Po načtení dat do podoby, do které chceme, nastupuje (mimo jiných objektů definovaných v třídách dědících od Approach) objekt *Evaluator*, který vyhodnotí podobnost mezi daty zpracovaných do vektorové reprezentace. To vše přebírá *PostProcessor*, který by měl končit hlavní funkci *Recommend()*, ten vypíše získaná data do výsledného souboru.

Zde je přehled, jaké třídy tyto rozhraní implementují:



IPredictor je další rozhraní, které je definované převážně CollaborativeFilteringApproach třídami. Slouží k predikci chybějících hodnot v user-item matici.

Abstraktní třída **User** je pouze obrys toho, jak by měl uživatel vypadat. Obsahuje *IUserVectorizer*, který slouží k převedení dosavadní reprezentace uživatele, ať už byla jakákoliv, na vektorovou. Struktura:

SisUser je třídou, která reprezentuje uživatele jako objekt s *Wish-list*em a seznamem oblíbených předmětů (*Favorites*). Implementuje SisUserVectorizer.

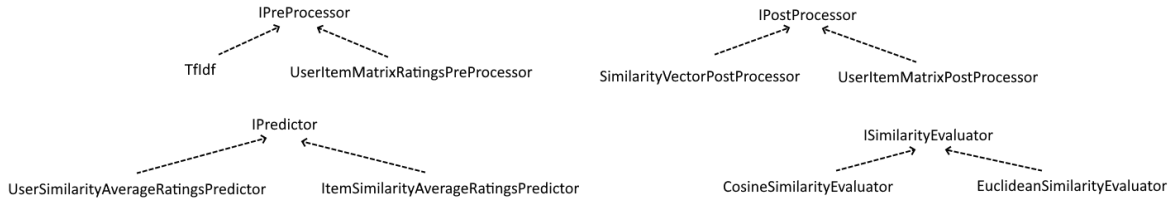
MovieDbsUser reprezentuje uživatele jako někoho se seznamem hodnocení. Tedy *Dictionary<int, int> userItemRatings*, kde key je index předmětu v dané matici a hodnota je jeho hodnocení od 1 do 5... 0, pokud hodnocení chybí.

LoggerException je třída rozšiřující třídu Exception s jednoduchým rozšířením, aby se mohla vynořovat se specifickou zprávou:

```
public LoggerException(string message) : base(message) { }
```

Složka Interfaces (s 8 rozhraními a třídami, které definují jejich funkce):

Začněme rozhraními, které jsou definovány u Approach. Znovu pro přehled:



TfIdf je ve zkratce převádění na vektorovou reprezentaci pomocí term frequency–inverse document frequency. Více zde TF-IDF. Aby se nezabíralo místo v paměti, jsou funkce definované tak, že si pomáhají dočasnými soubory. TfIdf převede každý záznam (řádek) vstupního dokumentu na vektor s hodnotami TF-IDF každého slova v tomto záznamu.

UserItemMatrixRatingsPreProcessor čte specifické číselné údaje záznamů v daném dokumentu, udržuje si indexy a výsledné údaje k nim vázané. Po přečtení si zpracované záznamy uloží do matice.

SimilarityVectorPostProcessor dostane už „vyplněný vektor“, který nejprve CountSortem seřadí a pak převede do výsledného souboru ve formátu ‚[index-ve-vektoru] [podobnost-k-uživateli]‘.

UserItemMatrixPostProcessor dostane vyplněnou matici, kterou také CountSortem seřadí a ve stejném formátu převede do výsledného souboru.

UserSimilarityAverageRatingsPredictor – doplní chybějící (nulové) hodnoty v matici pomocí podobností mezi uživateli. Podle vzorce:

$$r(user, item) = \frac{\sum_{k=0}^{n} sim(user,\ user_k) * r(user_k, item)}{\sum_{k=0}^{n} |sim(user,\ user_k)|}$$

Kde *r* je výsledná matice predikovaných hodnot, *sim* je matice podobnosti.

ItemSimilarityAverageRatingsPredictor – stejně jako v předchozím případě pouze využije vztahu mezi předměty, ne uživateli:

$$r(user, item) = \frac{\sum_{k=0}^{n} sim(item,\ item_k) * r(user, item_k)}{\sum_{k=0}^{n} |sim(item,\ item_k)|}$$

CosineSimilarityEvaluator dostane dva vektory a vrátí $\cos\theta$, kde θ je úhel mezi vektory. Pokud se v průběhu dělí nulou nebo nejsou vektory stejně dlouhé. Vrací se -1. Čím si jsou podobnější, tím blíže k 1.

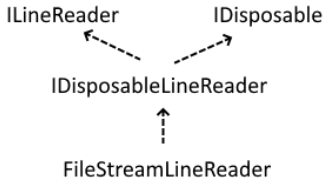
EuclideanSimilarityEvaluator mezi vektory spočítá vzdálenost, pokud je menší než jedna, vrátí 1, pokud ne, vrátí převrácenou hodnotu vzdálenosti. Vrací -1 při potížích.

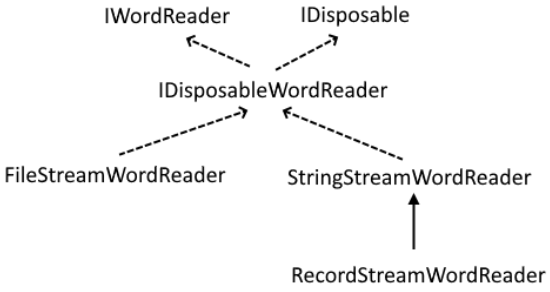
Zpět k uživateli, konkrétně k *IUserVecorizeru:*

SisUserVectorizer dostane na vstupu uživatele a načtenou matici, uživatelský vektor vyrobí tak, že každý předmět ve *Wish-list*u a *Favorites* najde v matici (tzn. její řádek) a přičte ho k vyrobenému uživatelskému vektoru (který byl na začátku nulový). Na konci vektor vydělí počtem předmětů v dotyčných seznamech. Tzn. udělá průměr všech oblíbených a chtěných předmětů. Očekáváme, že tímto způsobem dostaneme uživatelský vektor ve vektorovém prostoru mezi nebo v oblasti s předměty, které by si uživatel do jednoho ze seznamů přidal.

MovieDbsVectorizer pouze vytvoří vektor vhodné délky a doplní na indexy předmětů v seznamu uživatelovo hodnocení.

Třídy v IReader.cs souboru. Pro přehled:





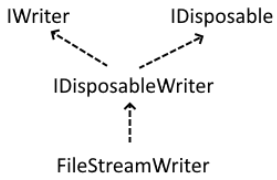
FileStreamLineReader – třída, která obsahuje StreamReader, který čte řádek po řádku. StreamReader je z venku Disposovatelný.

FileStreamWordReader – implementuje StreamReader, ale vrací přečtená slova, která jsou definována jako alespoň jeden znak ohraničený separátory (které jsou dodány v konstruktoru) nebo začátkem či koncem souboru.

StringStreamWordReader – implementuje StringReader.

RecordStreamWordReader (factory pattern) – v konstruktoru dostane ILineReader a separátory. Přečte řádek, ten předá StringReaderu a tím se vytvoří WordReader místo LineReaderu.

V IWriter.cs:



FileStreamWriter – jednoduchý StreamWriter, který, oproti normálnímu, definuje LoggerException při spadnutí. Je Disposable.

WinFormsRecSys

WinFormsSession dědí od Session a přidává funkcionality nutné pro integraci knihovny do Windows Forms prostředí. Definuje také, jako jediný projekt funkce pro integraci Convertoru.

WinFormsViewer dědí od Viewer. Ve svém konstruktoru bere TextBox, který slouží jako univerzálnější okno pro textový výstup. Má definované další dvě pomocné funkce, které jsou spíše pokusem o oddělení ovladače (Form1) od Vieweru.

Form1 je brán jako WinFormsController - bohužel od Controller nedědí. Definuje pouze funkce, které reagují na uživatelův vstup (kliknutí tlačítka apod.). Většina funkcí pošle svůj vstup do WinFormsSession, kde je zpracován a výsledek následně, pokud nutno, poslán do WinFormsViewer. Při volání konstruktoru Form1 jsou dynamicky vyplněny combo boxy dostupnými třídami.

WinFormsSisUser, **WinFormsMovieDbsUser** jsou třídy dědící od svých jmenovitých předků a zároveň od *IWinFormsUserUtil*, která, mimo jiné, dává schopnost převádět z formátu, ve kterým je uživatel ve Windows Forms definován. Další je například ta schopnost, že skladuje a vydává demo verze uživatelů.

ConsoleRecSys

ConsoleSession – znovu, slouží jako mediátor a zde hlavně jako pomocník při tvorbě příkladového uživatele. Definuje demo příklady. Definuje svoji funkci *Recommend()*, která nejdříve vytvoří instanci User třídy a pak teprve volá *GetRecommendations()* Session předka.

ConsoleViewer – zde jsou pouze, oproti předkovi, dodefinovány pomocné funkce pro hezčí zobrazování chybových zpráv a indexovaných seznamů.

ConsoleController – bere vstup z příkazové řádky, dokud mu nepřijde null nebo prázdný řádek. Vstup pošle do funkce Crunch, která má fixní seznam cases, kde se vstup zpracuje. Ojediněle, kvůli tomu, že v průběhu musí brát vstup, jsou funkce s dialogy (pro tvorbu Approach, User a selekci Demo příkladů) obsaženy zde.

ConsoleSisUser, **ConsoleMovieDbsUser** to samé, pouze s tím, že *IConsoleUserUtil* má další schopnosti charakteristické pro interakce v příkazovém řádku – například postupné přidávání jednotlivých předmětů, vymazání dosavadního seznamu.