PV178 HW02

V této úloze máte za úkol implementovat administrátorské konzolové rozhraní jednoduchého eshopu. Eshop bude obsahovat dvě entity - kategorie a produkty. Pro dané entity budete implementovat CRUD operace, všechny kromě update. Dané operace bude moci uživatel volat z konzolového uživatelského rozhraní, které naprogramujete. Volání CRUD operací bude zaznamenáno pomocí logování, které zachytí úspěšné i neúspěšné volání operace (viz Logování). Kromě logování bude aplikace zachycovat také analytická data, která budou zachycovat množství produktů v jednotlivých kategoriích (viz Analytická data). V zadání je krok za krokem vysvětleno, co se má v jaké části aplikace dít, od základních CRUD operací, přes logování a zpracování analytických dat až po prvotní naplnění souborů daty. Celkově se bude jednat o složitější úlohu zkoumající několik konceptů, mezi které ale nepatří práce se soubory, proto máte v kostře předpřipraveny třídy, které vám s tímto pomůžou.

Konzolové rozhraní + základní logika (BusinessContext)

V našem obchodě budeme evidovat dvě entity - kategorie a produkty. Každý produkt musí mít svou vlastní kategorii. Pro obě entity budeme implementovat základní operace pro vytváření, načítání a mazání entit, tedy operace CRUD kromě operace Update (viz Heroic Mode). Navíc u produktů můžeme chtít vypsat pouze ty, které patří do nějaké kategorie.

Třídy ProductDBContext a CategoryDBContext slouží k ukládání a načítání produktů a kategorií. Tyto třídy ukládají entity ve formátu JSON do souborů Product.json a Category.json. Mohou také pouze vrátit celý obsah souboru namapovaný na dané entity (Product, Category) nebo uložit kolekci entit do souboru s tím, že bude přepsán aktuální obsah. Tyto metody jsou již implementovány za vás, takže je nijak neměňte.

Ve třídě jsou definovány konstanty, které určují cestu ke složce s danými soubory. Jedná se o složku BussinessContext/DB/Storage, která je součástí řešení poskytnutého k domácímu

úkolu. Zadané cesty (proměnná _paths) ke složce a souborům upravte podle vašeho operačního systému.

Vaším úkolem bude implementovat uživatelské rozhraní pro správce eshopu, které bude podporovat následující operace:

`add-product <Name> <CategoryId> <Price>`

- ➔ Přidá produkt zadaný uživatelem do konzole.
- ➔ Entita Product bude obsahovat atributy Id, Name, CategoryId a Price.

`delete-product <ProductId>`

- ➔ Odebere produkt z našeho úložiště.

`list-products`

- ➔ Obsahuje seznam všech produktů z úložiště ve formátu:

```
Id | Name      | CategoryId | Price
-------------------------------------
1  | Product01 | 1          | 14
```

`add-category <Název>`

- ➔ Přidá kategorii zadanou uživatelem.
- ➔ Entita Kategorie bude obsahovat atributy Id a Name.

`delete-category <CategoryId>`

- ➔ Odstraní kategorii z našeho úložiště a všechny produkty patřící do této kategorie najednou (tzv. kaskádové odstranění).

`list-categories`

- ➔ Vypíše všechny kategorie z úložiště ve formátu:

```
Id | Name
------------------
1  | Category01
```

`get-products-by-category <CategoryId>`

- ➔ Vypíše všechny produkty dané kategorie z úložiště ve stejném formátu jako operace list-products.

Omezení

- ➔ Pokud přidávám produkt, musí kategorie již existovat, jinak úložiště vyhodí výjimku a produkt se neuloží.
- ➔ Id obou entit musí být jedinečné, jinak úložiště vyhodí výjimku. **Id generujete vy.**
- ➔ Odstraněním kategorie se odstraní všechny produkty v dané kategorii.
- ➔ K interakci se soubory, v nichž jsou data uložena, používejte pouze metody uvedené v kostře. Jedná se o metody v souborech *DBContext.
- ➔ Ke komunikaci s uživatelem použijte pouze jednu třídu. Console.WriteLine můžete volat z další (ale pouze jedné) třídy.
- ➔ Pro výše uvedené případy použití eshopu zkuste umístit logiku aplikace do samostatných tříd, tzv. "Services", pro kategorii a produkt (např. CategoryService, ProductService).
- ➔ Vrstva komunikující s uživatelem na konzoli se nemůže přímo dotazovat třídy CategoryDBContext/ProductDBContext, ale měla by k nim přistupovat pomocí servis uvedených v předchozím bodě.
- ➔ V případě potřeby si můžete vytvořit vlastní výjimky.

Logování (LoggerContext)

Každý z výše uvedených případů použití bude logován. Pro logování budeme používat textový soubor **LoggerContext/DB/Storage/Log.txt**. Ve složce **LoggerContext/DB** se nachází třída **LoggerDBContext**, která zapisuje logy do souboru logů, a to po jednom logu na řádek. Logy se zapisují v případě úspěšných i neúspěšných volání výše uvedených operací CRUD. Jinými slovy, log se zapisuje při všech uživatelských operacích. Log bude obsahovat následující informace:

- ➔ Timestamp ve formátu `dd/MM/rrrr HH:mm:ss`
- ➔ Typ logu: `Add` | `Delete` | `Get` → podle typu operace, kterou uživatel zadal. Mezi operace `Get` patří `list-*` a `get-products-by-category`.
- ➔ Typ entity: `Category` | `Product` → zda se jedná o log z volání na entitu kategorie nebo na entitu produkt.
- ➔ Úspěšnost operace: `Success` | `Failure` → určíme dle toho, zda byla během operace vyhozena nějaká výjimka.
- ➔ Referenci na samotnou entitu (snažte se používat zásady OOP). Všimněte si, že potřebujete pouze id entity, název entity a id kategorie (které obsahují obě entity).

K tomuto obsahu logu můžete připojit cokoli dalšího, co považujete za vhodné (např. chybovou zprávu v případě neúspěšné operace). V případě operace GET bude entita null, a proto bude log zkrácen o údaje o entitě (tj. Id entity; Název entity; CategoryId).

Formát logu

```
[Timestamp] Typ logu; Typ entity; Úspěch operace; Id entity;
Název entity; CategoryId
```

(na jednom řádku)

Příklady logů

```
[01/01/2023 18:14:41] Get; Product; Success

[01/01/2023 18:15:21] Add; Product; Failure; Invalid arguments for command AddProduct.

[01/01/2023 18:15:30] Add; Product; Success; 13; Lux; 1
```

Sběr analytických dat (AnalyticalDataContext)

Tato část úkolu se bude zabývat shromažďováním analytických dat při přidávání a odebírání entit. Analytická data budou zaznamenávat, kolik produktů je v jednotlivých kategoriích.

Analytická data jsou uložena v souboru **AnalyticalDataContext/DB/Storage/AnalyticalData.json**. Data jsou opět uložena ve formátu JSON. Pro práci s tímto souborem se používá třída **AnalyticalDBContext**, která poskytuje způsob, jak ze souboru číst entity a ukládat je do souboru. Metody opět mohou pouze vše číst a vše přepisovat.

Vaším úkolem bude zachytit tato analytická data spolu s logováním (viz odstavec Shrnutí logování a analytických dat). Pro odesílání dat do kontextu analytického zpracování použijte stejnou třídu/model, který jste zvolili pro logování. Po neúspěšné nebo GET operaci nebude v analytické části provedena žádná akce. Důležitá jsou zde pouze data o přidání a odstranění.

Analytická data se budou ukládat do JSON souboru ve tvaru:

Shrnutí logů a analytických údajů

Data pro účely logování v části aplikace pro logování a zpracování v části aplikace pro analýzu musí být do těchto částí aplikace odesílána současně. Jinými slovy, při odesílání dat pro logování musí být stejná data odeslána také do analytické části aplikace (analytická část aplikace již filtruje, která data potřebuje). Pro lepší představu můžeme předpokládat, že by byla daná data o přidání/odebrání/čtení entity odeslána do 10 dalších částí aplikace k různému zpracování. Pro tyto účely implementujeme zprávy pro logování a analýzu dat pomocí **událostí** (https://learn.microsoft.com/en-us/dotnet/standard/events/how-to-raise-and-consume-events?source=recommendations).

Předvyplnění dat (seeding)

Implementujte třídu, která se stará o předvyplnění dat. Stačí tři kategorie, každá po dvou produktech. Naplnění dat proběhne vždy při spuštění aplikace. Implementujte jej tak, aby se naplnila i analytická a logovací data.

Pár dobrých rad na závěr

- ➔ Upravte cesty k souborům ve třídách, které s nimi komunikují, podle svého operačního systému.
- ➔ V kostře máte připravené složky, které by vám mohly napovědět, jak aplikaci implementovat a rozdělit, nebo dokonce jaké konstrukce jazyka C# použít. Podle potřeby si můžete vytvořit i vlastní složky.
- ➔ Všechny třídy, které pracují se soubory, přepíší soubor při každé inicializaci, takže je stačí inicializovat pouze jednou při spuštění aplikace.
- ➔ Aplikace nemůže za žádných okolností spadnout. Vždy je třeba počítat s tím, že práce se souborem vyvolá výjimku, kterou je třeba zachytit a informovat uživatele. U výjimek, které definujete, dostane uživatel vámi zvolenou informaci - například "A category with the given Id does not exist", u ostatních výjimek dostane uživatel obecnou zprávu "An error occured: + text výjimky(pouze text/zpráva, ne celá výjimka)".
- ➔ Snažte se, aby byl Main co nejkratší. Patří tam pouze inicializace tříd nezbytných pro běh aplikace a řešení závislostí. Zbytek logiky a komunikaci s uživatelem implementujte v jiných třídách.

HEROIC MODE

(= dobrovolné rozšíření)

Implementujte do aplikace funkcionalitu Update, přidejte ji do logovací a analytické části aplikace.

Hodně štěstí při řešení úkolu!

Pokud máte nějaké dotazy, neváhejte se obrátit na Discord, vlákno hw02-discussion.