## Zadanie domowe z EF:

Realizując całe zadanie, można zdobyć 120%. Celem jest zaimplementowanie aplikacji o czystej architekturze (Domain, Application, Infrastructure, UI). Projekt pozwala na zarządzanie studentami, kursami i kadrą akademicką, z uwzględnieniem zaawansowanych mechanizmów numeracji, konfiguracji relacji i modelu danych.

## 1. Biblioteka `Domain` (Model Danych)

**(20 pkt.)** Zaimplementuj model danych reprezentujący byty uczelniane:

- **Osoby:**
  - **Student:** Imię, nazwisko, **`IndeksUczelniany`** (unikalny, np. "S1001"), rok studiów.
  - **Profesor:** Imię, nazwisko, **`IndeksUczelniany`** (unikalny, np. "P101"), tytuł naukowy.
- **Kurs:** Nazwa, kod kursu, punkty ECTS.
- **Wydział:** Nazwa.
- **Obiekt Wartości (Owned Entity):**
  - **Adres:** Klasa `Adres` (z właściwościami `Ulica`, `Miasto`, `KodPocztowy`). Encje `Student` i `Profesor` powinny zawierać tę encję jako **własność** (np. `AdresZamieszkania`).
- **Encja Sekwencji (Numeracja):**
  - **`LicznikIndeksow`:** Nowa encja odpowiedzialna za przechowywanie aktualnych liczników dla prefiksów. Powinna zawierać:
    - `Prefix` (string, Klucz główny, np. "S", "P").
    - `AktualnaWartosc` (int, np. 1001).
- **Relacje:**
  - **Zapisy (Enrollment):** Relacja **wiele-do-wielu z dodatkowymi danymi** (`Student` <-> `Kurs`). Encja łącząca `Enrollment` przechowuje **ocenę** oraz semestr.
  - **Wymagania wstępne (Prerequisites):** Relacja **wiele-do-wielu typu self-referencing** na encji `Kurs` (kurs może mieć wiele wymagań i być wymaganiem dla wielu innych).
  - **Gabinet (Office):** Relacja **jeden-do-jeden** (`Profesor` <-> `Gabinet`).

**(5 pkt.) Podencja (Dziedziczenie):** Zaimplementuj podencję **`StudentStudiowMagisterskich`** dziedziczącą po `Student`. Powinna ona posiadać dodatkowe informacje: `TematPracyDyplomowej` oraz relację do promotora (`Profesor`).

---

## 2. Biblioteka `Application` (Logika Biznesowa)

**(15 pkt.) Operacje CRUD:** Zaimplementuj operacje CRUD dla wszystkich głównych encji.

- **Logika Numeracji:** Operacje *tworzenia* `Studenta` i `Profesora` muszą implementować logikę pobierania i inkrementowania wartości z encji `LicznikIndeksow` (np. pobierz "S", weź `AktualnaWartosc` 1001, zwiększ na 1002 i

przypisz "S1001" nowemu studentowi). **Musi to być realizowane w ramach transakcji**, aby zapewnić spójność.
- **Zarządzanie Prefiksami:** Udostępnij operacje CRUD dla encji `LicznikIndeksow`, aby umożliwić **dodanie nowego prefiksu** (np. "D" dla doktorantów z wartością początkową 0).
- **Logika Usuwania (Cofanie Licznika):**
    - Operacja usuwania `Studenta` (lub `Profesora`) musi sprawdzać, czy usuwany obiekt ma najwyższy numer dla swojego prefiksu.
    - Jeśli tak (np. usuwamy "S1002", a licznik dla "S" wskazuje 1002), operacja musi **zdekrementować `AktualnaWartosc`** w `LicznikIndeksow` (z powrotem na 1001), aby uniknąć powstawania dziur na końcu sekwencji. Operacja ta również musi być częścią transakcji.

**(20 pkt.) Generator Danych (Bogus):** Zaktualizuj generator tak, aby podczas tworzenia fałszywych studentów i profesorów **korzystał z logiki biznesowej (Application)** do pozyskiwania kolejnych numerów `IndeksUczelniany`, zamiast generować losowe ciągi znaków.

---

## 3. Biblioteka `Infrastructure` (Dostęp do Danych)

**(20 pkt.) Kontekst i Migracje:** Zaimplementuj `UniversityDbContext`. Podczas konfiguracji modelu (OnModelCreating):

- **Skonfiguruj Encję Owned:** Użyj `modelBuilder.Entity<Student>().OwnsOne(s => s.AdresZamieszkania).`
- **Skonfiguruj Relacje:** Poprawnie skonfiguruj wszystkie relacje (O:O, O:M, M:M).
- **Niedomyślna Obsługa Kasowania:** Dla relacji `StudentStudiowMagisterskich` -> `Promotor` (Profesor) ustaw **`OnDelete(DeleteBehavior.SetNull).`** Chcemy, aby usunięcie profesora-promotora nie usuwało studenta, a jedynie zerowało u niego pole `PromotorId`.
- **Indeksowanie:** Dodaj **unikalny indeks** na kolumnie `IndeksUczelniany` w tabelach `Student` i `Profesor`, aby zapewnić unikalność i przyspieszyć wyszukiwanie.
- Wygeneruj migracje i logikę aktualizacji bazy danych.

---

## 4. Zapytania (LINQ / SQL)

**(20 pkt.) Złożone Zapytania:** Zaimplementuj następujące zapytania. Zapytania mogą być zrealizowane za pomocą LINQ (preferowane) lub czystego SQL (jeśli uznasz, że LINQ generuje nieoptymalny kod).

**Ważne:** Ocenie podlegać będzie nie tylko poprawność wyniku, ale również **wydajność zaimplementowanego zapytania**. Mile widziane jest aktywne unikanie typowych pułapek wydajnościowych EF.

Sugerowane techniki optymalizacyjne do rozważenia:

- **Projekcja Danych:** Zawsze używaj `.Select(new { ... })` lub `.Select(new WynikDTO { ... })`, aby pobierać z bazy *tylko* te kolumny, których potrzebujesz do wyświetlenia wyniku, zamiast ładować pełne encje.
- **Unikanie Problemu N+1:** Jeśli *musisz* załadować powiązane encje, używaj `Include()` i `ThenInclude()`, aby EF wygenerował odpowiedni `JOIN` (Eager Loading), zamiast polegać na leniwym ładowaniu (Lazy Loading) w pętli.
- **Zapytania "Read-Only":** Dla wszystkich poniższych zapytań (które tylko odczytują dane) stosuj `.AsNoTracking()`. Wyłącza to mechanizm śledzenia zmian, co znacząco przyspiesza materializację obiektów.
- **Ewaluacja Server-Side:** Upewnij się, że operacje takie jak `Where()`, `GroupBy()`, `Sum()`, `Average()` są wykonywane po stronie serwera SQL (zanim wywołasz `.ToList()` lub `.ToListAsync()`), a nie w pamięci aplikacji (Client-Side Evaluation).

---

**Wymagane zapytania:**

- **(5 pkt.)** Podaj profesora, który prowadzi kursy z największą łączną liczbą studentów (suma studentów ze wszystkich kursów prowadzonych przez danego profesora).
- **(5 pkt.)** Podaj średnią ocen (GPA) dla każdego kursu na danym wydziale (wydział wybierany przez użytkownika), wraz z liczbą studentów, którzy otrzymali ocenę.
- **(10 pkt.)** Znajdź studenta z "najtrudniejszym" planem zajęć. "Trudność" jest definiowana jako suma punktów ECTS wszystkich kursów, na które student jest aktualnie zapisany, ORAZ suma punktów ECTS wszystkich *bezpośrednich* wymagań wstępnych (prerekwizytów) dla tych kursów. Należy uważać na powtórzenia (jeśli dwa kursy mają ten sam prerekwizyt).

---

## 5. Interfejs Użytkownika (UI)

**(20 pkt.) Aplikacja (Konsola / Terminal.GUI / WPF):** Interfejs powinien umożliwiać:

- **(5 pkt.)** Uruchomienie generatora danych.
- **(12.5 pkt.)** Przeglądanie zawartości bazy (np. lista studentów z ich adresami i indeksami, lista liczników prefiksów).
- **(2.5 pkt.)** Usuwanie encji (testujące logikę cofania licznika).
- **(5 pkt.)** Uruchamianie i wyświetlanie wyników zaimplementowanych zapytań.