Zadania powinny zostać umieszczone w jednym repozytorium. Rozwiązania zadań mogą być umieszczane w oddzielnych solucjach (*.sln), projektach (*.*proj) lub plikach tekstowych, jeżeli zadanie nie wymaga kodu. Rozwiązania powinny być umieszczone w publiczym repozytorium - github, azure repos (devops).

Zadanie 1

Mając dany model wejściowy – *DemoSource.Person* - oraz wyjściowy – *DemoTarget.PersonWithEmail* - dokończ implementację metody *DemoImplementation.Flatten*.
Metoda ta powinna zwrócić *DemoTarget.PersonWithEmail* dla każdej pary *DemoSource.Person* – *DemoSource.EmailAddress* Zwróć uwagę na relację pomiędzy modelami *DemoSource.Person* oraz *DemoSource.EmailAddress*.
W jakim przypadku może zostać użyte takie mapowanie i jakie to ze sobą niesie konsekwencje? Napisz odpowiedź w komentarzu do metody *DemoImplementation.Flatten* w kodzie.

Listing 1: Model wejściowy

```
namespace DemoSource
{
    public class Person
    {
        public string Id { get; set; }
        public string Name { get; set; }
        public IEnumerable<EmailAddress> Emails { get; set; }
    }

    public class EmailAddress
    {
        public string Email { get; set; }
        public string EmailType { get; set; }
    }
}
```

Listing 2: Model docelowy

```
namespace DemoTarget
{
    public class PersonWithEmail
    {
        // contains only characters from english alphabet
        // (a-z, both uppercase and lowercase) and numbers 0-9
        public string SanitizedNameWithId { get; set; }

        // to be formatteed based on email and email type
        public string FormattedEmail { get; set; }
    }
}
```

Listing 3: Implementacja

```
namespace DemoImplementation
{
    public static IEnumerable<DemoTarget.PersonWithEmail>
        Flatten(IEnumerable<DemoSource.Person> people)
    {
        throw new NotImplementedException();
    }
}
```

Zadanie 2

Mając załadowane do pamięci: kolekcję group, kolekcję kont oraz kolekcję adresów email, przypisz osobę do każdego konta posługując się adresem email. Jedna osoba może mieć wiele adresów email i tym samym wiele kont.

- Pomiń walidację danych wejściowych,
- Przyjmij, że adres email jest unikalny w systemie,
- Przyjmij, że osoba może należeć tylko do jednej grupy
- Załóż, że w systemie jest
  - kilkadziesiąt group
  - średnio 1000 (tysiąc) osób na grupę
  - każda osoba ma średnio 4 adresy email

Dokończ implementację metody $ToDo.MatchPersonToAccount$. Wydajność metody powinna być dostosowana do ilości danych w systemie.

Listing 4: Model wejściowy

```
namespace DemoSource
{
    public class Person
    {
        public string Id { get; set; }
        public string Name { get; set; }
        public IEnumerable<EmailAddress> Emails { get; set; }
    }

    public class EmailAddress
    {
        public string Email { get; set; }
        public string EmailType { get; set; }
    }

    public class Account
    {
        public string Id { get; set; }
        public EmailAddress EmailAddress { get; set; }
    }

    public class Group
    {
        public string Id { get; set; }
        public string Label { get; set; }
        public IEnumerable<Person> People { get; set; }
    }
}
```

Listing 5: Implementacja

```
internal class ToDo
{
    public IEnumerable<(DemoSource.Account, DemoSource.Person)>
        MatchPersonToAccount(
            IEnumerable<DemoSource.Group> groups,
            IEnumerable<DemoSource.Account> accounts,
            IEnumerable<string> emails)
    {
        throw new NotImplementedException();
    }
}
```

Zadanie 3

Zmodyfikuj zmienną *predicate*, aby nie ryzykować iterowania po wszystkich elementach zagnieżdżonej kolekcji

Listing 6: Listy

```
1 public static IEnumerable<IEnumerable<string>>
2     OnlyBigCollections(List<IEnumerable<string>> toFilter)
3 {
4     Func<IEnumerable<string>, bool> predicate =
5         list => list.Count() > 5;
6
7     return toFilter.Where(predicate);
8 }
```