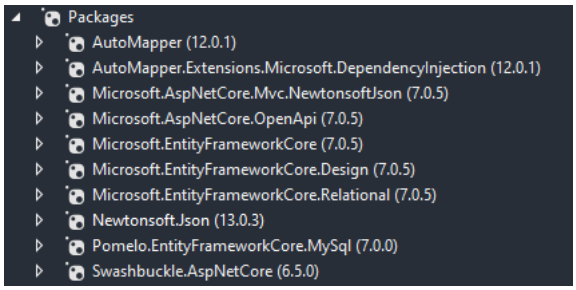
PeakPlanner

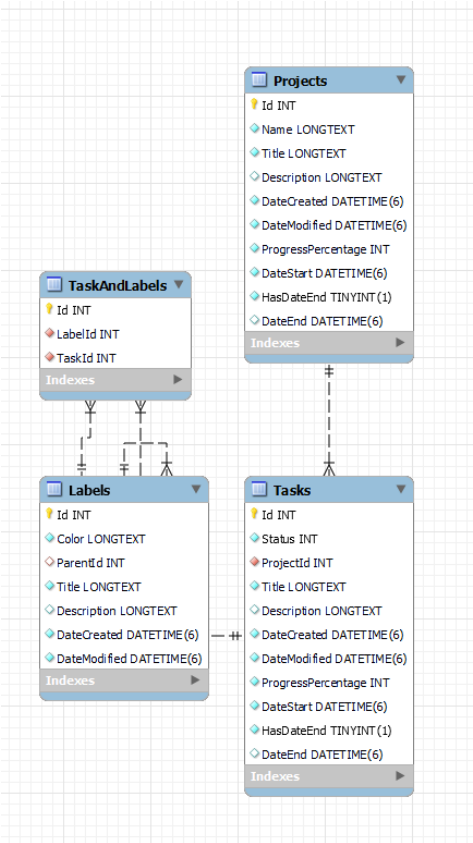
C# - NET Backend Assignment

Το συγκεκριμένο project αφορά το backend κομμάτι μιας διαχειριστικής εφαρμογής project που περιέχουν πολλαπλά Tasks. Η εφαρμογή έχει αναπτυχθεί σε C# 11 .NET Core 7 και είναι ένα ASP.NET Core Web API project.



Για την ανάπτυξη αυτής της εφαρμογής χρησιμοποιήθηκαν τα NuGet Packages δίπλα.

Στο διπλανό σχεδιάγραμμα φαίνεται η δομή της βάσης δεδομένων, οι πίνακες που δημιουργήθηκαν για την υλοποίηση του project, καθώς και τα properties και relationships μεταξύ των πινάκων.



Για την δημιουργία της βάσης χρησιμοποιήθηκε το Code First approach με το EF Core. Δηλαδή, πρώτα δημιουργήθηκαν τα απαραίτητα data models και μετά το EF Core δημιούργησε αυτόματα τους πίνακες με βάση αυτά τα models και τις σχέσεις που έχουμε ορίσει μεταξύ τους στο **PeakPlannerDBContext.cs** στην μέθοδο OnModelCreating(ModelBuilder modelBuilder).

Συγκεκριμένα έχουμε τις παρακάτω σχέσεις:

Projects:

One (Project) **With Many** (Tasks)

Labels:

One (Label) **With Many** (Labels)

One (Label) **With Many** (Task and Label pairs)

Tasks:

One (Task) **With Many** (Task and Label pairs)

Τα τελευταία δύο είναι στην ουσία ένα **Many** (Tasks) **To Many** (Labels) relationship.

Για την καλύτερη οργάνωση του κώδικα δημιουργήθηκαν και τα απαραίτητα **Interfaces**.

Program.cs

Παίρνει από το docker-compose.yml μέσα από το Environment τα στοιχεία για την δημιουργία του connection string για την σύνδεση στη βάση.

Δημιουργεί το connection με την βάση προσθέτοντας στο services το DbContext.

Για τον **AutoMapper**

Παίρνει όλα τα .cs αρχεία από το project που τελειώνει σε RequestModel, ResponseModel & Entity και τα προσθέτει σε αντίστοιχες λίστες.

Δημιουργεί τα αντίστοιχα mappings για Request σε Entity και Entity σε Response μέσω reflection.

```
cfg.CreateMap<TaskAndLabelEntity, EmbeddedLabelResponseModel>()
    .ForMember(x => x.Id, options => options.MapFrom(y => y.Label!.Id))
    .ForMember(x => x.Title, options => options.MapFrom(y => y.Label!.Title))
    .ForMember(x => x.Color, options => options.MapFrom(y => y.Label!.Color));
```

Ακόμη, δημιουργεί ένα map για τα TaskAndLabelEntity σε EmbeddedLabelResponseModel όπως φαίνεται στην είκονα.

Αν το origin έχει null σε property μην κάνεις map και πάρε την τιμή από το destination.

Για τον **Newtonsoft**

Όταν κάνει το serialization εφάρμοσε CamelCase.

Για τον **DbContext**

Τσεκάρει αν δημιουργήθηκε η βάση.

Routes.cs

Στο αρχείο αυτό υπάρχουν όλα τα απαραίτητα routes για τα calls στο API.

Entities

Για το progress των tasks και projects στο StandardEntity.cs που κάνουν και τα δύο models inherit έχει οριστεί ένα απλό logic που θέτει στο progress την τιμή 100 αν του τεθεί τιμή μεγαλύτερη του 100, ενώ για τιμή μικρότερη του 0 δεν υπάρχει πρόβλημα γιατί είναι ένας θετικός ακέραιος, δηλαδή είναι type of uint.

Για το status των tasks, επειδή είναι συγκεκριμένες οι τιμές που μπορεί να πάρει τέθηκε ως enum, όπως φαίνεται στην διπλανή εικόνα.

```
#region Private Members

/// <summary>
/// The member of the <see cref="ProgressPercentage"/> property
/// </summary>
private uint mProgressPercentage = 0;

#endregion

#region Public Properties

/// <summary>
/// The progress percentage
/// </summary>
/// <remarks>Default : 0</remarks>
/// <value>Min: 0 - Max: 100</value>
1 reference
public uint ProgressPercentage
{
    get => mProgressPercentage;
    set
    {
        mProgressPercentage = value;
        // If the value is greater than 100...
        if (mProgressPercentage > 100)
            mProgressPercentage = 100;
    }
}
```

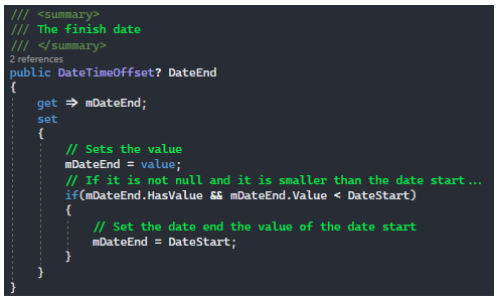
```
/// <summary>
/// Provides enumerations over the status of a <see cref="TaskEntity"/>
/// </summary>
3 references
public enum TaskStatus
{
    /// <summary>
    /// To do
    /// </summary>
    ToDo,

    /// <summary>
    /// In progress
    /// </summary>
    InProgress,

    /// <summary>
    /// In review
    /// </summary>
    InReview,

    /// <summary>
    /// Done
    /// </summary>
    Done
}
```

Για το finish date, επειδή μπορεί να μην έχει κάποια τιμή υπάρχει ένα flag το HasDateEnd που είναι true όταν υπάρχει κάποια τιμή και είναι valid. Σε αντίθετη περίπτωση ότι τιμή και να πάρει αγνοείται λόγω του flag. Η ύπαρξη του flag είναι απαραίτητη, διότι σε περίπτωση που θελήσει ο χρήστης να θέσει null καλώντας την update, απλά θα αγνοήσει ο AutoMapper το null και θα πάρει την τιμή του entity.



Γι' αυτό το λόγο, υπάρχει το flag που θα δηλώνει αν έχει νόημα να λάβει στα υπόψη την τιμή του date end ή όχι.

Repositories

Δημιουργήθηκαν για την δημιουργία και ενημέρωση των Task.

Μέσα τους τα request models αυτών έχουν λίστες με τα Ids των Labels και των Tasks που θέλουν να περιέχουν αντίστοιχα.

ControllerHelpers.cs

Μια στατική κλάση που περιέχει Generic μεθόδους για Get, Post, Put και Delete για να αποφευχθεί το code duplication, από τη στιγμή που το logic παραμένει ακριβώς ίδιο για τις μεθόδους αυτές σε απλά σενάρια δημιουργίας, ενημέρωσης, ανάκτησης και διαγραφής.

Controllers

Περιέχουν τα CRUD operations για τα Projects, Labels και Tasks.

Swagger

Τρέχοντας την εφαρμογή εμφανίζεται η σελίδα του swagger το documentation του API με τα αντίστοιχα endpoints και τα request και responses για τα labels, tasks και projects.

Σημειώσεις

Ένα παραπάνω κομμάτι που θα πρόσθετα στην υλοποίηση του συγκεκριμένου project θα ήταν η δημιουργία και σχεδιασμός ενός καλύτερου μηχανισμού για την διαχείριση των responses. Συγκεκριμένα θα έφτιαχνα έναν wrapper που θα περιείχε ό,τι έχει το response μαζί με τυχόν exceptions, error messages, a flag indicating whether the response was successful or not and the status code of the response. Θα μπορούσε να δημιουργεί και ένα ξεχωριστό project για unit testing με xUnit για την 100% επιβεβαίωση ότι το API δουλεύει σωστά.