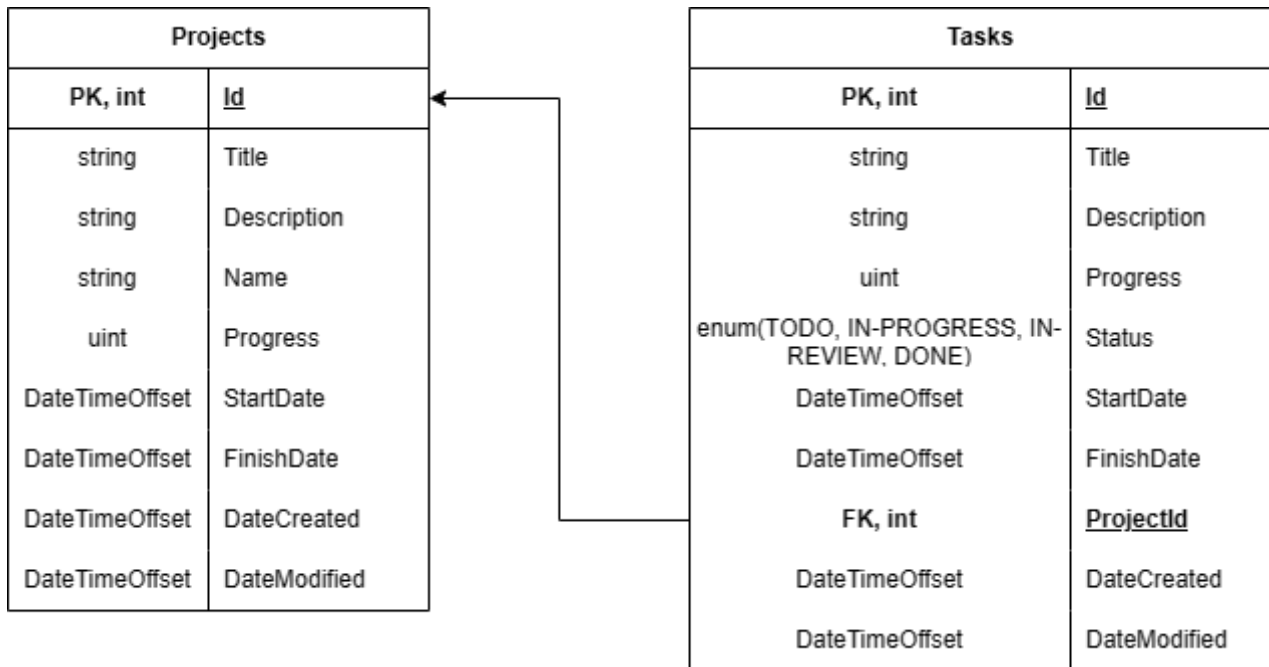


BOARDCONNECT DOCUMENTATION

Repository URL: [TPrappas/BoardConnect \(github.com\)](https://github.com/TPrappas/BoardConnect)

Το πρώτο βήμα για την υλοποίηση του Assignment είναι η κατανόηση της εκφώνησης. Αποφάσισα να χρησιμοποιήσω το diagrams.net έτσι ώστε να δημιουργήσω το παρακάτω διάγραμμα με τους πίνακες που χρειάζονται












Στην συνέχεια, δημιούργησα ένα ASP.NET Core Web API project στο Visual Studio 2022. Για το assignment αποφάσισα να ακολουθήσω το Code First Approach και να χρησιμοποιήσω την MySQL σαν βάση. Αρχικά μέσω του MySQL Workbench δημιούργησα μια βάση με όνομα boardconnect και στην συνέχεια στο appsettings.json πρόσθεσα τα credentials της βάσης

```
"ConnectionStrings": {
  "MySQLConnection":
    "server=localhost;port=3306;database=boardconnect;user=root;password=12345678;"
}
```

Το παραπάνω MySqlConnection αφορά το πρώτο στάδιο χωρίς το docker. Καθώς πρώτα τέσταρα το API με την χρήση του Postman.

Στην συνέχεια έκανα εγκατάσταση τα παρακάτω NuGet Packages.

	AutoMapper by Jimmy Bogard A convention-based object-object mapper.	12.0.1
	Microsoft.AspNetCore.Mvc.NewtonsoftJson by Microsoft ASP.NET Core MVC features that use Newtonsoft.Json. Includes input and output formatters for JSON and JSON PATCH.	7.0.5
	Microsoft.AspNetCore.OpenApi by Microsoft Provides APIs for annotating route handler endpoints in ASP.NET Core with OpenAPI annotations.	7.0.4 7.0.5
	Microsoft.EntityFrameworkCore by Microsoft Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.	7.0.5
	Microsoft.EntityFrameworkCore.Design by Microsoft Shared design-time components for Entity Framework Core tools.	7.0.5
	Microsoft.EntityFrameworkCore.Tools by Microsoft Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.	7.0.5
	MySQL.EntityFrameworkCore by Oracle MySQL Server database provider for Entity Framework Core.	7.0.2
	Newtonsoft.Json by James Newton-King Json.NET is a popular high-performance JSON framework for .NET	13.0.3
	Swashbuckle.AspNetCore by Swashbuckle.AspNetCore Swagger tools for documenting APIs built on ASP.NET Core	6.5.0

Ξεκίνησα αρχικά από τα Models τα οποία θα χρησιμοποιήσω. Αρχικά δημιούργησα τον φάκελο DataModels που περιέχει τα Entities και τα Enums. Στα Enums περιλαμβάνεται το αρχείο με τα enums που αφορούν το Status του Task καθώς έχει fixed τιμές. Στα Entities αρχικά δημιούργησα την abstract class BaseEntity το οποίο περιλαμβάνει το int Id, DateTimeOffset DateCreated, DateTimeOffset DateUpdated, τα οποία περιλαμβάνονται σε κάθε Entity. Ο λόγος που χρησιμοποίησα το DateTimeOffset είναι επειδή αντιπροσωπεύει ένα σημείο στο χρόνο και ταυτόχρονα συμπεριλαμβάνει και την πληροφορία του time zone. Συγκεκριμένα χρησιμοποιεί το local date time της περιοχής που βρίσκεται ο server. Στην συνέχεια δημιούργησα τα υπόλοιπα Entities τα οποία κληρονομούν από το BaseEntity. Ύστερα δημιούργησα τον φάκελο APIModels που περιλαμβάνει τα RequestModels και ResponseModels. Με παρόμοιο τρόπο δούλεψα στα RequestModels δημιούργησα πρώτα το BaseRequestModel από το οποίο κληρονομούν τα υπόλοιπα RequestModel. Τέλος δημιούργησα τα ResponseModels, ξεκίνησα από το abstract BaseResponseModel το οποίο περιέχει μόνο το Id, στην συνέχεια το abstract DateResponseModel που περιέχει τα DateCreated και DateUpdated και κληρονομεί από το BaseResponseModel. Από αυτά τα δύο μπορούν κληρονομήσουν τα υπόλοιπα ResponseModels ανάλογα με το τι θα θέλουμε να επιστρέφεται. Τέλος έχουν προστεθεί και κάποια EmbeddedResponseModel τα οποία περιέχουν τις βασικές πληροφορίες των ResponseModel που έχουν δημιουργηθεί.

Μετά δημιούργησα το dbContext για να καθορίσω την δομή των πινάκων και την σχέση τους. Στην Startup έκανα set up το database connection string καθώς και τον AutoMapper. Δημιούργησα επιπλέον το FrameworkConstructionExtensions.cs το οποίο βοηθάει στον «διαχωρισμό» των Suffixes και των Prefixes. Επιπλέον τροποποίησα το Project.cs έτσι ώστε να χρησιμοποιεί την Startup για να δημιουργεί τον Host.

Στην συνέχεια χρησιμοποίησα το Package Manager Console αρχικά με την εντολή 'Add-Migration Initial' δημιούργησα ένα αρχικό migration και στην συνέχεια με την εντολή 'Update-Database' δημιουργήθηκαν αυτόματα οι πίνακες και η σχέση μεταξύ τους.

Επίσης δημιούργησα το ControllersHelper.cs το οποίο περιλαμβάνει τις γενικές CRUD μεθόδους που χρησιμοποιούνται από τους Controllers. Επιπλέον με το routes.cs καθορίζονται τα routes του API. Ακολούθως, δημιούργησα τα ProjectController.cs και TaskController.cs για να διαχειρίζονται τα CRUD operations.

Ύστερα προσέθεσα τον φάκελο APIArgs ο οποίος περιλαμβάνει τα arguments τα οποία χρησιμοποιούνται σαν query parameters. Έτσι ενημέρωσα τις GetAllAsync μεθόδους για να ενσωματωθούν τα arguments.

Σε αυτό το σημείο έγινε test του API με το Postman, και αφού επιβεβαίωσα ότι όλα λειτουργούν κανονικά προχώρησα με το Docker.

Τέλος επεξεργάστηκα το αρχείο docker-compose.yml, στην συνέχεια δημιούργησα ένα καινούργιο connection στο MySQL Workbench με τα στοιχεία που έδωσα στο αρχείο yml δηλαδή με port 8003. Επιπλέον επεξεργάστηκα το appsettings.json έτσι ώστε να δώσω το κατάλληλο connection string:

```
"ConnectionStrings": {  
  "MySQLConnection":  
    "server=boardconnectdb;port=3306;database=boardconnect;user=root;password=12345678;"  
}
```

Ύστερα δημιούργησα την βάση boardconnect σε αυτό το connection και στην συνέχεια με την χρήση του Postman και στο port 8080 που όρισα, δοκίμασα να κάνω API calls. Τα οποία επιστρέφουν τα αναμενόμενα αποτελέσματα.

Για το συγκεκριμένο project ένα από τα πράγματα που θα πρόσθετα επιπλέον θα ήταν οι φάκελοι CreateRequestModels και UpdateRequestModels. Ουσιαστικά με αυτά τα Models θα μπορούμε να δώσουμε για κάθε entity συγκεκριμένα Properties για όταν θέλουμε να κάνουμε Create ή Update αντίστοιχα. Επιπλέον θα μπορούσα να προσθέσω Repositories για μεθόδους οι οποίες χρειάζονται πιο «βαρύ» logic. Τέλος θα δημιουργούσα ένα Object το οποίο κληρονομεί από το Ifailable και μέσα σε αυτό αποθηκεύουμε πληροφορίες όπως το error message, αν ήταν successful, το status code καθώς και το result. Στην συνέχεια με operators θα μετέτρεπα το object σε object result και θα πρόσθετα τα Repositories ως services στο Program για να έχω πρόσβαση σε αυτά και στους Controllers.

Παρακάτω φαίνονται τα API Endpoints:

BoardConnectAPI <small>1.0 OAS3</small> <small>/swagger/v1/swagger.json</small>	
Project ^	
POST	/api/projects ✓
GET	/api/projects ✓
GET	/api/projects/{projectId} ✓
PUT	/api/projects/{projectId} ✓
DELETE	/api/projects/{projectId} ✓
Task ^	
POST	/api/tasks ✓
GET	/api/tasks ✓
GET	/api/tasks/{taskId} ✓
PUT	/api/tasks/{taskId} ✓
DELETE	/api/tasks/{taskId} ✓