Упражнение: Работа с чужд код

Проектът: Ще ви бъде предоставен проект - .NET Core Web приложение, което служи за управление на заявки за отпуск.

1. Цел

Целта на упражнението е да направите първите си стъпки в работата с чужд код. Ще се запознаете с дадения ви проект и ще направите промени с цел подобряване качеството на кода.

2. Запознаване с проекта

Започнете, като се запознаете с проекта:

- Идентифицирайте слоевете и изследвайте структурата им.
- Изследвайте структурата на всеки слой
- Проучете как слоевете си комуникират

Стартиране на проекта

Добър начин за запознаване с проект е като първо го стартираме и видим какво изпитва потребителя, използвайки софтуера. За да направите това първо трябва да сте сигурни, че имате инсталиран допълнителен софтуер/сървър, на който приложението разчита. В случая ще ни е нужен MSSQL Server (как да се стартира проекта често е описано в readme файл, а ако е нужен login, за да се достъпи приложението най-вероятно ще ви бъде предостъпен временен акаунт или ще се наложи да вкарате сами потребител в базата).

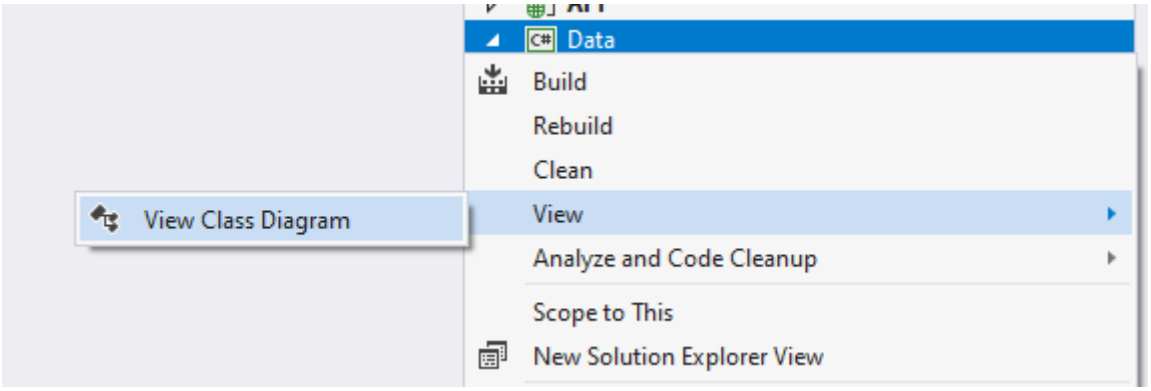
След като стартирате приложението, проучете какви изгледи предоставя то и какво изпитва потребителя при работа с всеки един от тях. Минете през функционалността на приложението, която може да бъде достъпена през потребителския интерфейс, запознайте се цялостно със системата.

Забележка: Потребители, с които можете да влезете в приложението можете да откриете, след като се запознаете добре със структурата на проекта.

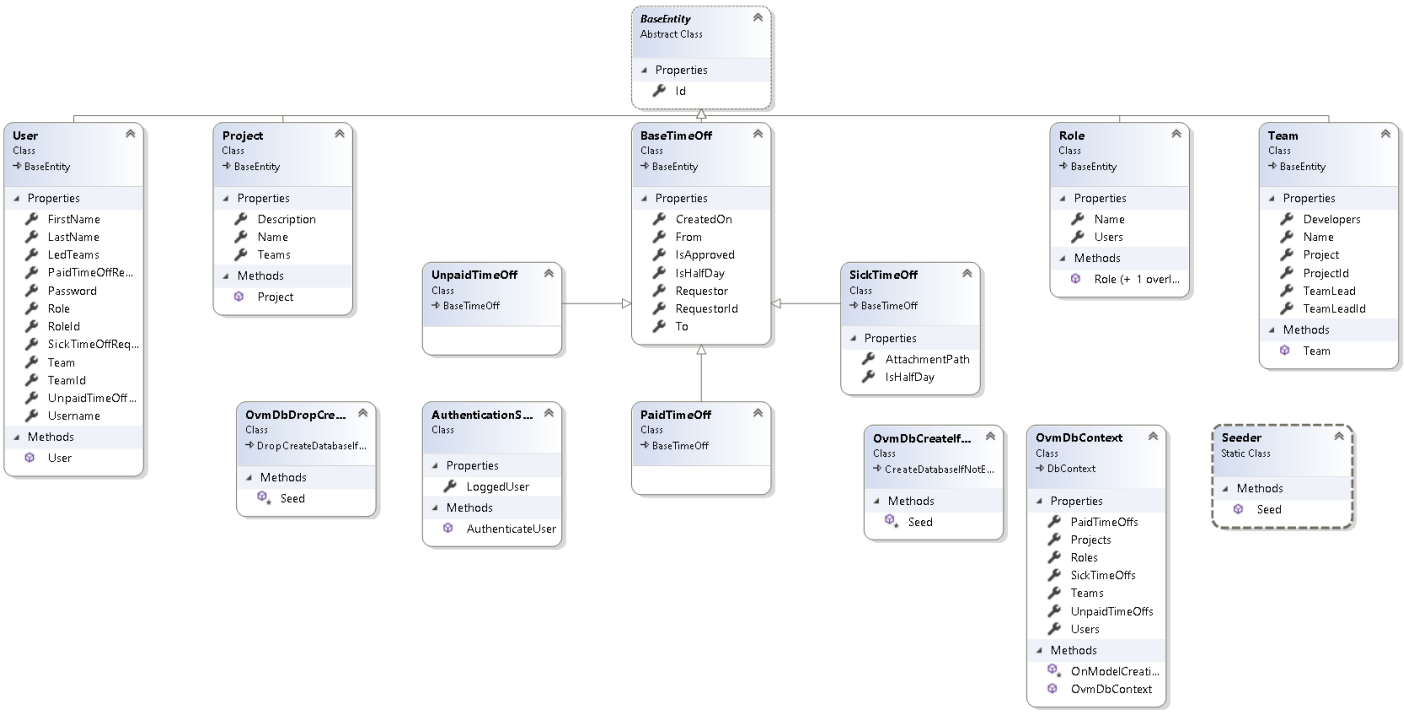
Картографиране на проекта

Използвайте инструментите, които Visual Studio предлага, за да изградите клас-диаграма на текущия проект. Това ще ви помогне по-добре и по-бързо да се запознаете със предоставената ви система.

За целта първо ще създадем клас-диаграма за проекта, отговарящ за работата с данни – Data, след което ще създадем такава и за уеб проекта в нашия solution.



Съответно за нас ще бъде създадена диаграма, която изглежда по подобен начин:



По аналогичен начин, създайте и клас-диаграма за уеб приложението. След тази стъпка сме готови със запознаването с проекта.

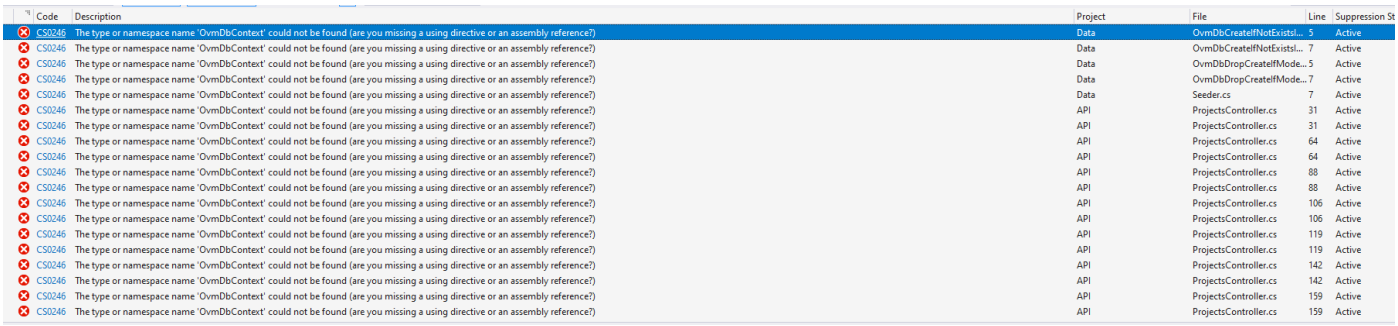
3. Рефакториране

След като се запознахме с проекта е време за работа с цел подобряването му. Ако сте вникнали добре в системата ще забележите, че заявките към базата данни се изпращат директно чрез контролерите, използвайки DbContext класа. Това е много неустойчив похват за работа с данни. Проучете как може да се подобри качеството на кода в тази посока.

Забележка: Прочетете повече за Repository Pattern като цяло, както и за имплементацията с Entity Framework

Процесът по рефакториране

Добър подход е да се започне с октриването на всички места в кода, където се използват данни от базата. Един начин за откриването на тези места е като се закоментира DbContext класа и се build-не наново проектът. Visual Studio ще изведе Error List, от който можем да добием представа къде се използват заявки към нашата база.



Code	Description	Project	File	Line	Suppression St
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	Data	OvmDbCreateIfNotExistsI...	5	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	Data	OvmDbCreateIfNotExistsI...	7	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	Data	OvmDbDropCreateIfMode...	5	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	Data	OvmDbDropCreateIfMode...	7	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	Data	Seeder.cs	7	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	31	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	31	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	64	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	64	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	88	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	88	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	106	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	106	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	119	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	119	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	142	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	142	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	159	Active
CS0246	The type or namespace name 'OvmDbContext' could not be found (are you missing a using directive or an assembly reference?)	API	ProjectsController.cs	159	Active

След като имаме представа къде ще променяме кода си е време да напишем нашите репозиторита. Най-подходящото място за тях ще е в нова папка Repository, намираща се в проекта Data. Можем да започнем с имплементацията на репозиторитата, като трябва да има такова за всяка таблица от нашата база. По презумция всяко репозитори трябва да съдържа методи за добавяне, редактиране, изтриване, извличане на един обект, извличане на всички обекти, но в различните приложения се налага създаването и на други по-специфични методи. Какви методи ще напишете във вашите репозиторита?

Забележка: За целта на упражнението напишете репозитори за нашите потребители и тествайте дали системата работи с вашото репозитори. За да се упражните повече може да напишете репозитори и за другите таблици в базата самостоятелно.

След като имате написано първото си репозитори е време то да бъде тествано. Разкоментирайте DbContext класа, за да продължи да работи приложението, променете даден контролер, за да използва вече новосъздаденото репозитори и тествайте програмата (с unit тестове или предизвиквайки контролера да бъде извикан през потребителския интерфейс).

Аналогично повтаряйте процеса за всяко репозитори, докато вече не се използват директни заявки към базата.

Накрая отново не забравяйте да тествате дали всичко работи след промените и ако работи значи сте готови с рефакторирането на нашия проект.

Пример за репозитори (отвори само при нужда)

Ето примерен вариант как може да изглежда вашето репозитори за потребители:

```
public class UserRepository
    {
        private readonly OvmDbContext _dbContext;

        public UserRepository(OvmDbContext dbContext)
        {
            this._dbContext = dbContext;
        }
```

```
        public IQueryable<User> GetAll()
        {
            return this._dbContext.Users.AsQueryable();
        }

        public IQueryable<User> GetAll(Expression<Func<User, bool>> predicate)
        {
            return this._dbContext.Users.Where(predicate).AsQueryable();
        }

        public User GetOne(int id)
        {
            return this._dbContext.Users.Find(id);
        }

        public User GetOne(Expression<Func<User, bool>> predicate)
        {
            return this._dbContext.Users.FirstOrDefault(predicate);
        }

        public void Add(User entity)
        {
            this._dbContext.Users.Add(entity);
            this._dbContext.SaveChanges();
        }

        public void Update(User entity)
        {
            this._dbContext.Entry(entity).State = EntityState.Modified;
            this._dbContext.SaveChanges();
        }

        public void Remove(User entity)
        {
            this._dbContext.Users.Remove(entity);
            this._dbContext.SaveChanges();
        }

        public int Count()
        {
            return this._dbContext.Users.Count();
        }

        public int Count(Expression<Func<User, bool>> predicate)
        {
            return this._dbContext.Users.Count(predicate);
        }
    }
```

Пример как се използва в контролер:

```
private readonly UserRepository _userRepository;

        public UsersController()
        {
            this._userRepository = new UserRepository(new OvmDbContext());
        }

        [HttpGet]
        public ActionResult Index(UsersIndexVM model)
        {
```

```
            if (AuthenticationManager.LoggedUser == null)
                return RedirectToAction("Login", "Home");

            model.Pager = model.Pager ?? new PagerVM();
            model.Pager.Page = model.Pager.Page <= 0 ? 1 : model.Pager.Page;
            model.Pager.ItemsPerPage = model.Pager.ItemsPerPage <= 0 ? 10 :
model.Pager.ItemsPerPage;

            model.Filter = model.Filter ?? new UsersFilterVM();

            bool emptyUsername = string.IsNullOrWhiteSpace(model.Filter.Username);
            bool emptyFirstName = string.IsNullOrWhiteSpace(model.Filter.FirstName);
            bool emptyLastName = string.IsNullOrWhiteSpace(model.Filter.LastName);
            bool emptyRoleName = string.IsNullOrWhiteSpace(model.Filter.RoleName);

            IQueryable<User> query = this._userRepository.GetAll(u =>
                (emptyUsername || u.Username.Contains(model.Filter.Username)) &&
                (emptyFirstName || u.FirstName.Contains(model.Filter.FirstName)) &&
                (emptyLastName || u.LastName.Contains(model.Filter.LastName)) &&
                (emptyRoleName || u.Role.Name.Contains(model.Filter.RoleName)));

            model.Pager.PagesCount = (int)Math.Ceiling(query.Count() /
(double)model.Pager.ItemsPerPage);

            query = query.OrderBy(u => u.Id).Skip((model.Pager.Page - 1) *
model.Pager.ItemsPerPage).Take(model.Pager.ItemsPerPage);

            model.Items = query.Select(u => new UsersVM
            {
                Id = u.Id,
                Username = u.Username,
                FirstName = u.FirstName,
                LastName = u.LastName,
                RoleName = u.Role.Name
            }).ToList();

            return View(model);
        }
```

4. Какво постигнахме

След рефакторирането нашият код е много по-подреден и лесен за четене. Освен това, въвеждането на бъдещи промени също би било по-лесно. Например, ако в бъдеще искаме данните да не идват от MSSQL Server, а от друг сървър или файлова система, всичко, което трябва да направим е да въведем промени само на едно единствено място – репозиторатата, които въведохме.