T.C

BOLU ABANT İZZET BAYSAL ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

TASARIM DESENLERİ

Proje Ödevi

Eren ŞAŞKIN

171906056

İÇİNDEKİLER

1. GİRİŞ

1.1. Proje Özeti

Proje BAİBU Bilgisayar Mühendisiliği öğrenci bilgilerinin kayıtlarını tutmaktadır. Öğrencinin bilgilerinin oluşturulması, listelenmesi, güncellenmesi ve silinmesi işlemleri yapılmaktadır.

Senaryoya göre kullanıcıdan sadece öğrencinin *ad soyad* ve *kayıt yılı* bilgileri alınır. Bu bilgiler ile öğrenciye *okul numarası, email* ve *sınıf* bilgisini oluşturmakta ve bunları veritabanında saklamaktadır. Daha sonra yine senaryo gereği kayıt sisteminde yeniliğe gidilmiştir ve yeni sisteme göre *sınıf* bilgisinin yapısı değişmiştir (int iken string) ve ek olarak *parola* bilgisi istenmektedir.

1.2. Proje Türü

Asp.Net Web Uygulamasıdır. *.Net Core 5*, *Angular 11* ve NoSQL veritabanı olan *MongoDB* kullanılmıştır. Geliştirme ortamları Visual Studio 2019 ve Visual Studio Code. Programlama dili C# ve TypeScript.

1.3. Kullanılan Tasarım Desenleri

- Builder
- Adapter
- Repository
- Singleton

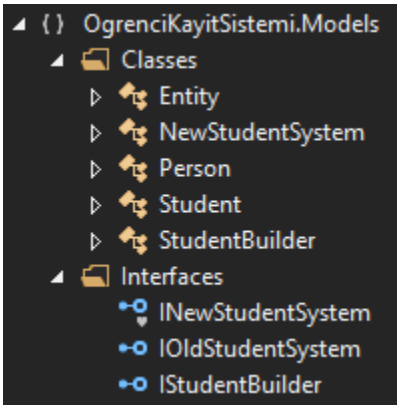
2. UYGULAMA TASARIMI

2.1. Proje Adımları

Öğrenci kayıt sistemi uygulaması oluştururken öncelikle model katmanı daha sonra veritabanı işlemi için kullanılacak servis katmanı sonra bu iki katmanın iletişimi için kontrolör katmanı oluşturuldu.

"Uygulamanın tamamı paylaşılacağı için raporda kodlara yer verilmedi. Yalnızca görevlerine değinildi".

2.1.1. Model Katmanı



Entity.cs: Repository sınıfına gönderilecek `Student.cs` ve `NewStudentSystem.cs` sınıflarının ortak özelliği olan id özelliklerini tutar. Bu iki sınıfta `Entity.cs` sınıfını implement eder.

NewStudentSystem.cs: Bu sınıf adapter modelini uygulayabilmek için senaryo gereği oluşturuldu. Eski sistem `Student.cs` sınıfının özelliklerinden farklı olarak öğrenci sınıfı `string` olarak düzeltildi ve ek olarak parola özelliği istendi.

Person.cs: Builder modelini uygulamak için senaryo gereği oluşturuldu. İçerisinde ad, soyad ve kayıt yılı bilgilerini bulundurur. Kullanıcı yeni kişi oluşturduğunda bu üç özelliği istenir. Bu özelliklerinden `Student.cs` sınıfında mail, sınıf ve numara oluşturulur.

Student.cs: `StudentBuilder.cs` sınıfında `Person.cs` den gelen veriler ile öğrenci oluşturur. Bu senaryoya göre eski öğrenci sistemidir. Sınıf verisi `int` değer alır ve parola bilgisi istenmez.

StudentBuilder.cs: Kurucu olarak `Person.cs` alır ve bunları `Student.cs` deki eş değerleriyle eşleştirir. Ayrıca mail, öğrenci numarası ve sınıfı metodlarını çalıştırarak öğrenci nesnesini döndüren `buildResult` metodu vardır.

INewStudentSystem.cs `NewStudenSystem.cs` sınıfının arayüzüdür. Adapter modelinin kullanılması için oluşturulmuştur.

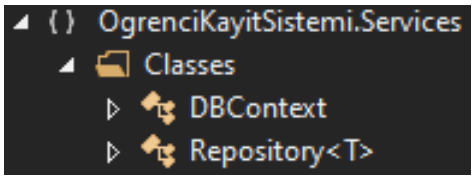
IOldStudentSystem.cs: `Student.cs` sınıfının arayüzüdür. Adapter modeli uygularken `NewStudentSystem.cs` içerisinde çağırılarak `SetOpr()` fonksiyonuyla değerleri yeni sisteme göre entegre edilir.

IStudentBuilder.cs `StudentBuilder.cs` sınıfının arayüzüdür. Builder modelin uygulanması için kullanılmıştır.

2.1.2. Servis Katmanı

Bu katmandakiler `Startup.cs` içerisinde AddSingleton metoduyla tanımlanmıştır.

```
services.AddSingleton<DBContext>();

services.AddSingleton(typeof(Repository<>));
```

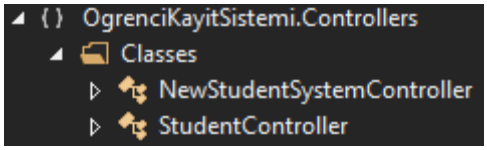


DBContext.cs: Veritabanı adresini ve veritabanı adını tutar. Bu sınıfa istek olduğunda IMongoDatabase tipinde veritabanını gönderir. Sadece `Repository.cs` sınıfında çağırılır.

Repository.cs: Kurucu olarak `DBContext.cs` den veritabanını alır. `Entity.cs` ile gelen nesnenin `Student.cs` mi yoksa `NewStudentSystem.cs` mi olduğu bilgisi alınır ve Kontrollörlerden gelen isteklerde gelen nesnenin tipine göre o tabloda veritabanı işlemlerini yapar API ye gerekli geri dönüşleri iletir.

2.1.3. Kontrolör Katmanı

Uygulamanın API bölümüdür. Kullanıcıdan gelen GET, POST, PUT, DELETE isteklerine yanıt verir. İstekler `Repository.cs` sınıfıyla veritabanında işlemler yapar.



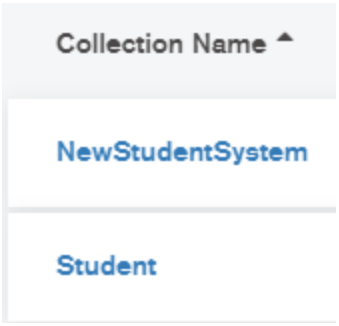
NewStudentSystemController: Yeni sistemin isteklerini yanıtlayan kontrolördür.

StudentController: Eski sistemin isteklerini yanıtlayan kontrolördür.

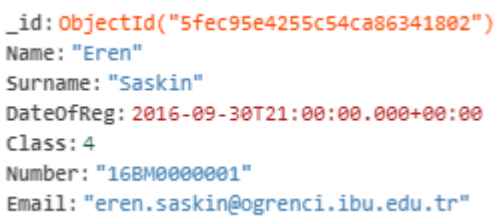
2.1.4. Veritabanı Oluşturma

MongoDB veritabanı NoSql bir veritabanıdır. Repository.cs ye DBContext.cs den verilen bağlantı adresi doğru ise gönderilen tipte koleksiyon (SQL veritabanlarına göre tablo) oluşturur ve gönderilen verileri ise belge şeklinde tutar.

Tablolar



Kayıt



2.1.5. Arayüz Oluşturma

Uygulama arayüzünde 2 sayfa bulunmaktadır. İşlemler sayfası (Anasayfa) ve Öğrenciler sayfası.

İşlemler sayfası: Yeni kişi oluşturmak için ad, soyad ve kayıt yılı isteyen bir form, kayıtların listelendiği ve güncelle, sil işlemlerinin yapılabildiği tablo bulundurur. Form gerekli doğrulama (validation) işlemlerini yapar. Angular 11 ile sayfa yenilenmeden listede güncelleme işlemleri yapılabilir.

Öğrenciler sayfası: Kişi oluşturulunca veritabanına kayıt olan öğrencileri listeler. Üstte eski sisteme göre kayıtlar listelenir altta ise yeni sisteme göre kayıtlar listelenir. Buradaki veriler işlemler sayfasında yapılan işlemlere göre değiştirilebilmektedir.

Eski sistemde öğrenci bilgileri

Numara	Ad - Soyad	Sınıf	Email	Kayıt Tarihi
19BM0000001	Safak Kayikci	2	safak.kayikci@ogrenci.ibu.edu.tr	01/10/2019
20BM0000002	Eren Saskin	1	eren.saskin@ogrenci.ibu.edu.tr	01/10/2020
17BM0000003	Ali Oz	4	ali.oz@ogrenci.ibu.edu.tr	01/10/2017
17BM0000004	Mehmet Yilmaz	4	mehmet.yilmaz@ogrenci.ibu.edu.tr	01/10/2017
18BM0000005	Veli Koc	3	veli.koc@ogrenci.ibu.edu.tr	01/10/2018

Yeni sistemde öğrenci bilgileri

Numara	Ad - Soyad	Sınıf	Email	Şifre	Kayıt Tarihi
19BM0000001	Safak Kayikci	2	safak.kayikci@ogrenci.ibu.edu.tr	SK19B	01/10/2019
20BM0000002	Eren Saskin	1	eren.saskin@ogrenci.ibu.edu.tr	ES20B	01/10/2020
17BM0000003	Ali Oz	4	ali.oz@ogrenci.ibu.edu.tr	AO17B	01/10/2017
17BM0000004	Mehmet Yilmaz	4	mehmet.yilmaz@ogrenci.ibu.edu.tr	MY17B	01/10/2017
18BM0000005	Veli Koc	3	veli.koc@ogrenci.ibu.edu.tr	VK18B	01/10/2018

2.2. Tasarım Desenlerinin Kullanım Yerleri

2.2.1. Singleton

C # 'daki Singleton tasarım modeli, yazılım tasarımı en yaygın tasarım modellerinden biridir. Singleton tasarım modelinde, bir sınıfın programda yalnızca bir örneği olmasını sağlar ve ona genel bir erişim noktası sağlar. Singleton, yalnızca tek bir örneğinin oluşturulmasına izin veren ve genellikle bu örneğe basit erişim sağlayan bir sınıftır.

.Net Core ile singleton yapıyı kurmak için sadece `startup.cs` sınıfında tek bir metodla oluşturmak münkün.

```
services.AddSingleton<DBContext>();
```

Bu metodu Startup.cs sınıfında configure metoduna ekleyerek proje çalıştığında DbContext sınıfının kolayca singleton olduğu tanımlanmış olur.

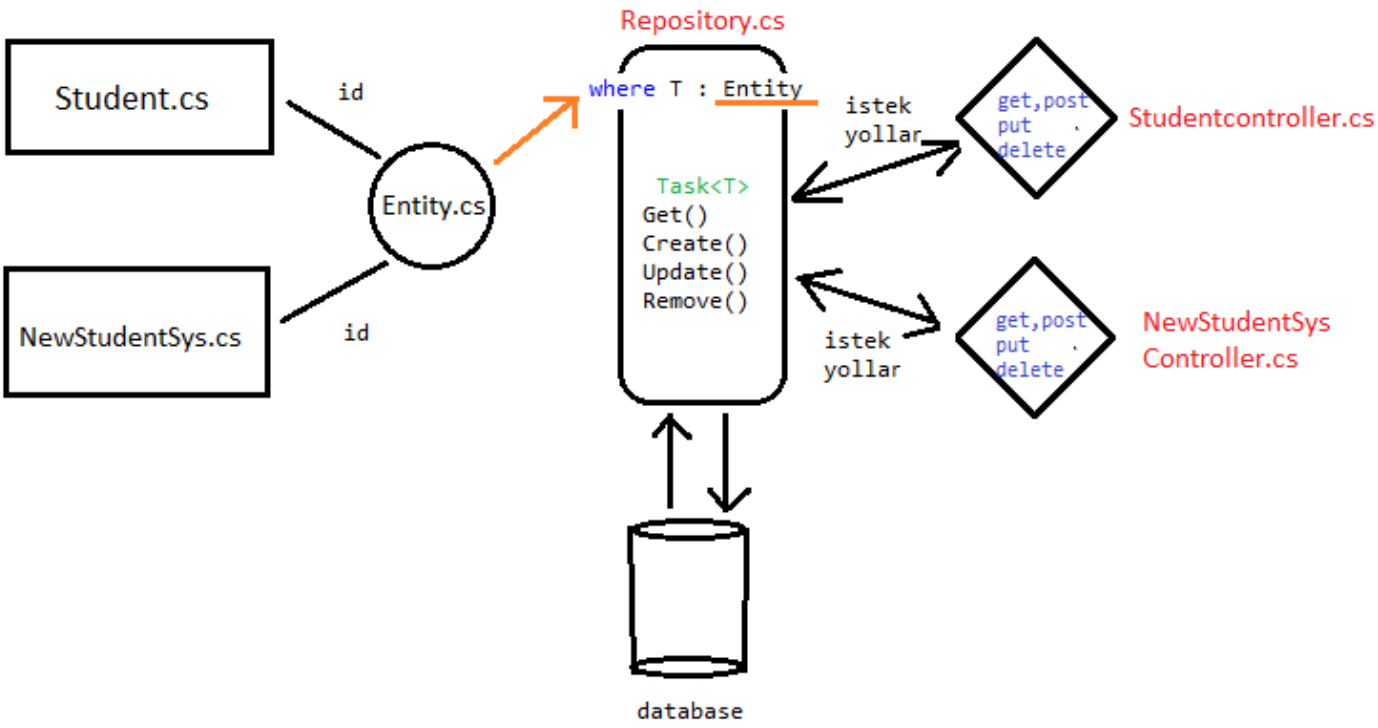
```
public class DBContext
    {
        protected readonly IMongoDatabase _database;
        public DBContext()
        {
            var client = new MongoClient("mongodb://localhost:27017");
            if(client != null)
            {
                _database = client.GetDatabase("StudentRegSystDB");
            }
        }

        public IMongoDatabase Database => _database;

    }
```

Projede veritabanı bağlantısını sağlamak için oluşturduğum DbContext.cs.

2.2.2. Repository

Her varlık türü için bir repository sınıfı oluşturmak, çok sayıda tekrarlanan kodla sonuçlanabilir. Repository, bu tekrarlamayı en aza indirmenin ve her tür veri için tek bir temel havuz çalışmasına sahip olmanın bir yoludur.

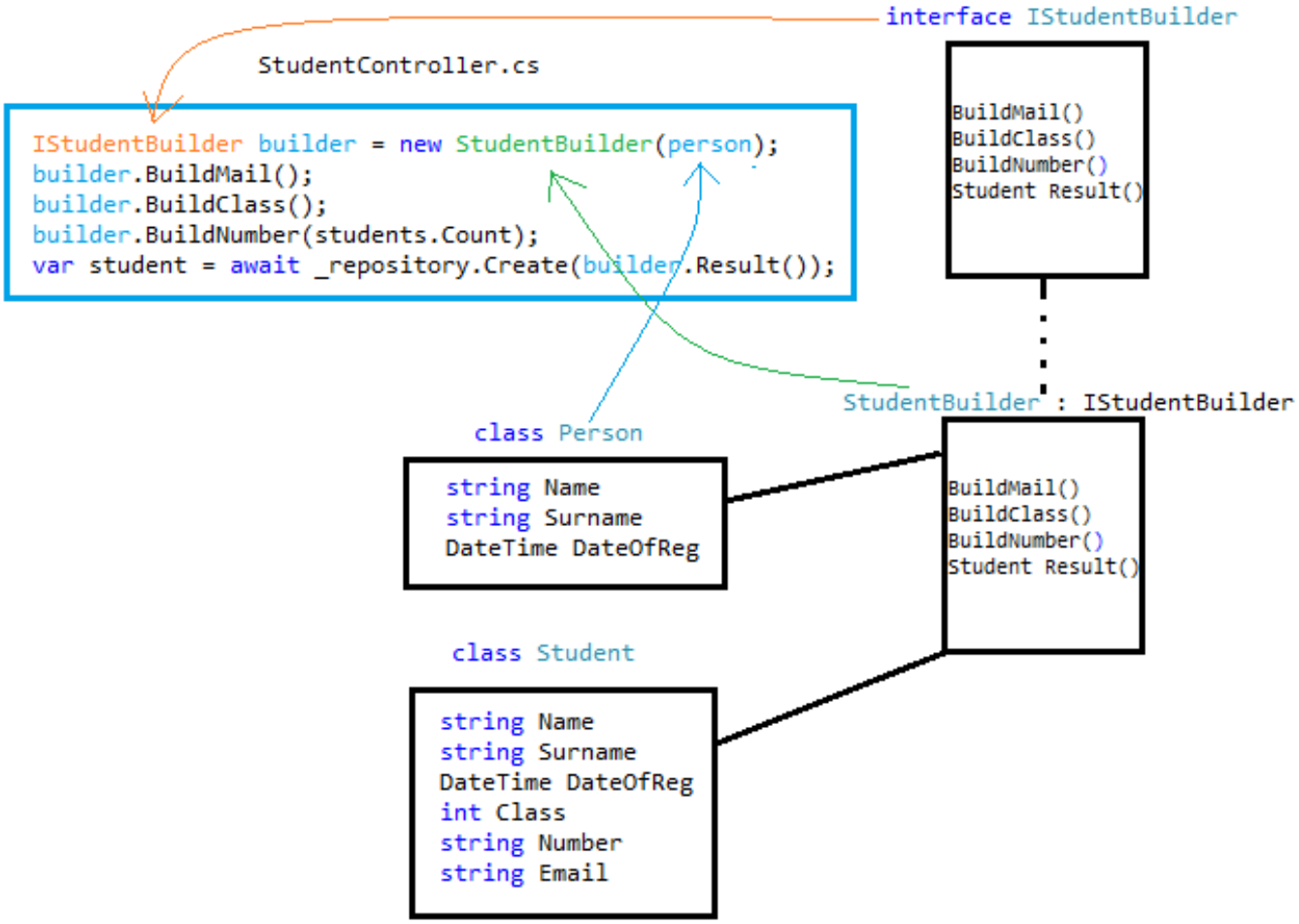
Servis Katmanında repository sınıfı bu görevi yerine getirmektedir. Veritabanı işlemlerinin (ekle, sil, listele, güncelle) yapıldığı bu sınıf, işlem yapılacak olan bir varlık alır ve yürüteceği işlemin o varlıkla ilgili olduğunu tanımlar. Bu projede Model Katmanında `Student.cs` ve `NewStudentSystem.cs` sınıfları birer varlık sınıfıdır. Eğer repository kullanılmasaydı her bir sınıf için veritabanı işlemlerini yürüteceğimiz servis sınıfı oluşturmak gerekecekti. Model Katmanındaki `Entity.cs` sınıfı ise varlıkların ortak özelliklerinin tutulduğu sınıftır. Kontrolörden bir istek olduğunda hangi varlığa ait olduğu bilgisi alınarak o varlığa göre veritabanına gerekli işlemleri yaptırır.



2.2.3. Builder

Builder, nesne yönelimli programlamada çeşitli nesne oluşturma sorunlarına esnek bir çözüm sağlamak için tasarlanmış bir tasarım modelidir. Builder tasarım modelinin amacı, karmaşık bir nesnenin yapımını temsilinden ayırmaktır.

Projede kullanım yerine baktığımızda; kullanıcı kişi oluşturduğunda (POST) `StudentController.cs` deki `Create` fonksiyonu çalışır. Bu fonksiyon `person.cs` alan StudentBuilder nesnesi. Bu nesne içerisinde de `person.cs` den alınan ad, soyad, kayıt yılı bilgileri `Student.cs` deki alanlarıyla eşleştirilir ve ek olarak mail, sınıf, numara fonksiyonlarıyla Student.cs deki diğer alanlar oluşturulur. Daha sonra da tekrar StudentController içerisinde, `Repository.cs` içerisindeki `Create` fonksiyonuna oluşturulan `Student.cs` nesnesi gönderilir. Böylece kayıt edilmiş olur.

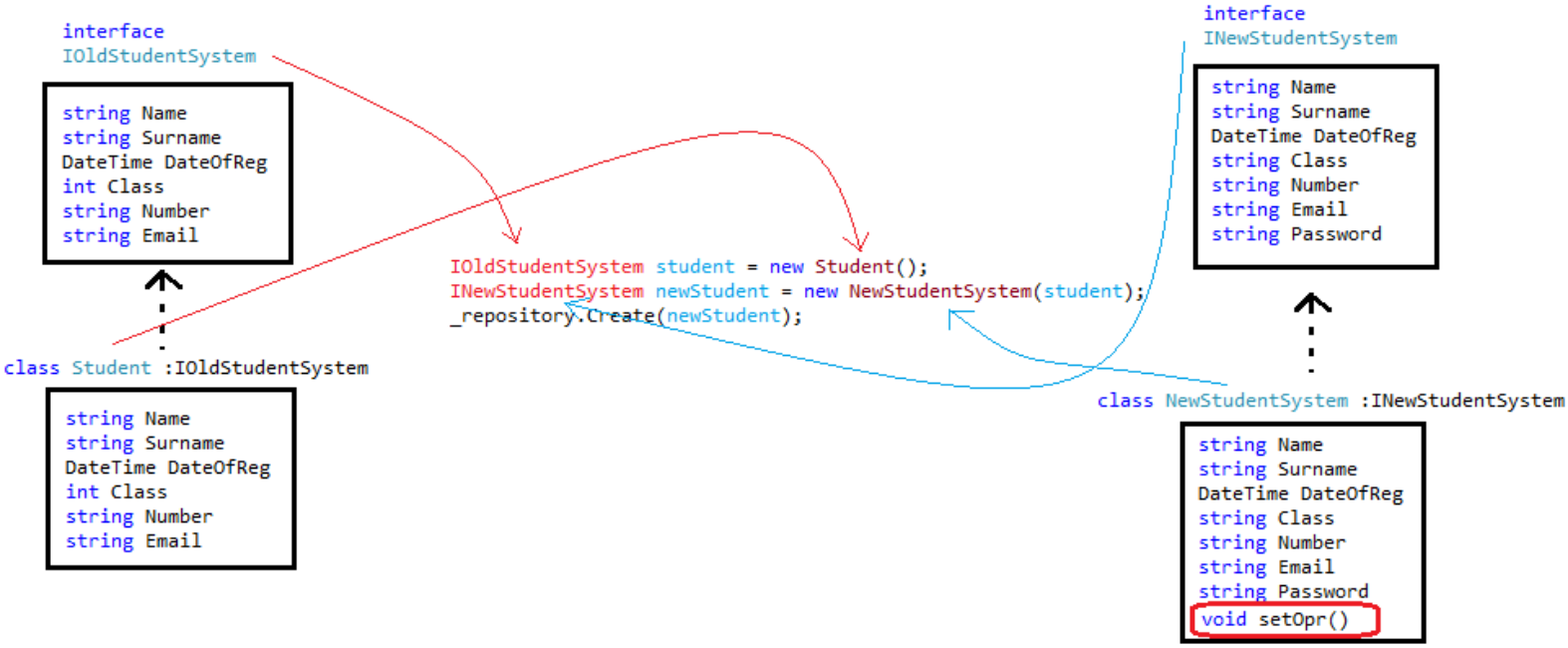


`Person.cs` sınıfından alınan ad, soyad bilgisiyle ad.soyad@ogrenci.ibu.edu.tr, Kayıt yılından alınan değerler ile de öğrenci numarası ve öğrenci sınıfı oluşturulur.

2.2.4. Adapter

Adaptör deseni, mevcut bir sınıfın arayüzünün başka bir arayüz olarak kullanılmasına izin veren bir modelidir. Genellikle varolan sınıfların kaynak kodlarını değiştirmeden başkalarıyla çalışmasını sağlamak için kullanılır.

Projede kullanım yerine baktığımızda; Kayıtların tutulduğu eski sistemde sınıf değeri sayısal bir değer almaktadır ve öğrenciye ait bir parola tanımlanmıştır. Yeni kayıt sistemine göre sınıf değeri bir `string` tipinde ve öğrenciye ait parola tanımlanması gerekmektedir. Eski sistemi yeni sisteme çevirebilmek için `NewStudentSystem.cs` sınıfında kurucu (constructor) olarak `IOldStudentSystem.cs` alır. `SetOpr()` fonksiyonuyla `IOldStudentSystem.cs` den gelen verilerde değişikleri yaparak atam işlemleri yapar.

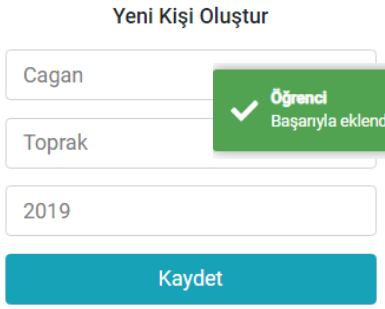



```
  void setOpr()
          {
              Id = this.oldStudent.Id;
              name = this.oldStudent.Name;
              surname = this.oldStudent.Surname;
              dateOfReg = this.oldStudent.DateOfReg;
              @class = (this.oldStudent.Class).ToString();
              number = this.oldStudent.Number;
              email = this.oldStudent.Email;
              password = this.oldStudent.Name.Substring(0, 1) +
this.oldStudent.Surname.Substring(0, 1) + this.oldStudent.Number.Substring(0,
3);
          }
```

`SetOpr()` fonksiyonu

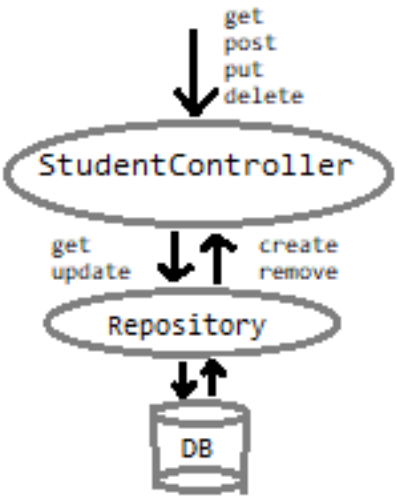
3. SONUÇLAR

Bir kayıtın uygulamada geçtiği yollara bakarsak;



1. İlk önce kullanıcı ad, soyad ve kayıt yılı girer.

2. Post metodu ile API ye istekte bulunur. StudentController, create fonksiyonunda `IStudentBuilder builder = new StudentBuilder(person);` oluşturur ve builder sınıfında öğrencinin diğer özellikleri(mail, sınıf, okul numarası) oluşturulur. Gönderilen kayıt yılından günümüzde kaçıncı sınıfta olduğu, ad-soyad bilgisinden mail ve kayıt yılından numara üretir. Bu sonuçları `repository.Create(builder.Result());` ile repository sınıfına gönderir.



3. Repository sınıfına giden builder sonuçları `collection.InsertOneAsync(item);` ile veritabanına kaydı oluşturur. Oluşturulan kişiyi geri döndürür.

4. Bu kez NewStudentSystemController bu Student nesnesini alır NewStudentSystem sınıfındaki `setOpr()` fonksiyonuyla yeni sisteme göre öğrenci oluşturur. Bu yeni sistem ise yeni olarak ad-soyad ve kayıt yılından alınan verilerle öğrenciye şifre oluşturur.

5. Arayüz tarafında `refreshGetAll()` metodu ise veritabanındaki kayıtlar için her iki Kontrollöre GET isteğinde bulunur böylece veritabanına kayıt edilen veriler tabloda listelenir. Güncelleme, silme işlemlerinde benzer adımlar izlenmektedir.

Eski kayıt sistemi tablosunda sonuç:

19BM0000006	Cagan Toprak	2	cagan.toprak@ogrenci.ibu.edu.tr	01/10/2019

Yeni kayıt sistemi tablosunda sonuç:

19BM0000006	Cagan Toprak	2	cagan.toprak@ogrenci.ibu.edu.tr	CT19B	01/10/2019