1 ) Case 1 ‘in kontrolu için CodeCaseController’i kullanabilirsiniz.

Bu controllerda 2 farklı metod mevcuttur. Bunlar CreateCode ve CheckCode

**CreateCode** : Mevcutta verilen karakterler ile 8 haneli şifre oluşturur.

Request Type : Get,

Path : …/api/CodeCase/CreateCode

Response : List<string>

8 haneli unique kod üretim aşamalari :

Öncelikle mevcutta verilen karakterler string array ‘e çevirimi yapılıyor. Bunu dökümanda **codeCharacterArray** ismi ile kullanmaya devam edicem.

Sonra 4 farklı random sayı türetiyoruz.

CharacterNumber1: codeCharacterArray’in içindeki kaçıncı indexi alacağımızı gösteriyor ve bu rakam ile codeCharacterArray in içinden o indexte bulunan karakteri aliyoruz. Bu karakter kodun 8. Hanesini ekleniyor.

CharacterNumber2: codeCharacterArray’in içindeki kaçıncı indexi alacağımızı gösteriyor ve bu rakam ile codeCharacterArray in içinden o indexte bulunan karakteri aliyoruz. Bu karakter kodun 2. Hanesini ekleniyor.

isUpperItteration1 ve isUpperItteration2 : bu Boolean değerler random seçilen sayılar codeCharacterArray ‘ın count’inin yarısından büyükse true değilse false oluyor.

ArrayNumber1 : 8 haneli unique kodun kaçıncı haneye geleceğini gösteriyor. Bunda random sayısı 3 ve 6 arası üretiliyor. Burdan gelen random sayıyı codeCharacterArray de index olarak baksak herzaman 4 karakter kullanılırdı. Bundan dolayı bu sayısı codeCharacterArray de ileri yada geri kulanmak için kullanıyoruz. Burda ileri yada geri gitmeyi kontrol etmek için isUpperItteration1 boolean değerinden yararlanıyoruz. CharacterNumber1 in karşılık geldiği codeCharacterArray den ileri yada geri arraynumber1 değeri kadar gidiyoruz. Bulduğumuz karakter kodun 1. Hanesini oluşturuyor.

**(CharacterNumber1 \* ArrayNumber1) % codeCharacterArray.Count** hesabından çıkan değer codeCharacterArray indexi oluyor ve o indexteki değer alınıp, kodda arraynumber1 hanesine yazılıyor.

ArrayNumber2 : 8 haneli unique kodun kaçıncı haneye geleceğini gösteriyor. Bunda random sayısı 3 ve 6 arası üretiliyor ancak arrayNumber 1 den farklı olması lazım çünkü aynı haneye veri atamayız. Burdan gelen random sayıyı codeCharacterArray de index olarak baksak herzaman 3 karakter kullanılırdı. Bundan dolayı bu sayısı codeCharacterArray de ileri yada geri kulanmak için kullanıyoruz. Burda ileri yada geri gitmeyi kontrol etmek için isUpperItteration2 boolean değerinden yararlanıyoruz. CharacterNumber2 in karşılık geldiği codeCharacterArray den ileri yada geri arraynumber2 değeri kadar gidiyoruz. Bulduğumuz karakter kodun 7. Hanesini oluşturuyor.

**(CharacterNumber2 \* ArrayNumber2) % codeCharacterArray.Count** hesabından çıkan değer codeCharacterArray indexi oluyor ve o indexteki değer alınıp, kodda arraynumber2 hanesine yazılıyor.

Geri eksik 2 hane kalıyor onlarda sırası ile bu hesaplar yapılarak giriliyor:

**(CharacterNumber2 \* ArrayNumber1) % codeCharacterArray.Count** hesabından çıkan değer codeCharacterArray indexi oluyor ve o indexteki değer alınıp, boş kalan ilk haneye yazılıyor.

**(CharacterNumber1 \* ArrayNumber2) % codeCharacterArray.Count** hesabından çıkan değer codeCharacterArray indexi oluyor ve o indexteki değer alınıp, boş kalan son haneye yazılıyor.

**CheckCode** : Kullanıcının yüklediği kodun doğruluğunu kontrol etmek.

Request Type : Post

Path : …/api/CodeCase/ CheckCode

Body : string

Response : returnCodeEnum

public enum ReturnCodeEnum

{

Success = 0,

WrongCode = 1,

EmptyCode = 2,

WrongSizeCode = 3

}

8 haneli unique kod kontrol aşamalari :

8 haneli kodun 8. hanesi CharacterNumber1 ve 2. hanesi CharacterNumber2 yi oluşturuyor.

isUpperItteration1 ve isUpperItteration2 : bu Boolean değerler random seçilen sayılar codeCharacterArray ‘ın count’inin yarısından büyükse true değilse false oluyor.

Boolean değeri kontrol ederek

8. hanenin codeCharacterArray deki indexinden 1. hanenin codeCharacterArray deki indexini toplayıp yada çıkartarak. arrayNumber1 bulunuyor.

2. hanenin codeCharacterArray deki indexinden 7. hanenin codeCharacterArray deki indexini toplayıp yada çıkartarak. arrayNumber2 bulunuyor.

arrayNumber1 kodda kaçıncı haneye denk geliyor ise onu

**(CharacterNumber1 \* ArrayNumber1) % codeCharacterArray.Count** hesabından çıkan değer codeCharacterArray indexi oluyor ve o indexteki değer alınıp koddaki arraynumber1 hanesindeki karakter ile aynımı diye kontrol ediliyor.

arrayNumber2 kodda kaçıncı haneye denk geliyor ise onu

**(CharacterNumber2 \* ArrayNumber2) % codeCharacterArray.Count** hesabından çıkan değer codeCharacterArray indexi oluyor ve o indexteki değer alınıp koddaki arraynumber2 hanesindeki karakter ile aynımı diye kontrol ediliyor.

Diğer haneler sırası ile

**(CharacterNumber2 \* ArrayNumber1) % codeCharacterArray.Count** hesabından çıkan değer codeCharacterArray indexi oluyor ve o indexteki değer alınıp, koddaki hanesindeki karakter ile aynımı diye kontrol ediliyor.

**(CharacterNumber1 \* ArrayNumber2) % codeCharacterArray.Count** hesabından çıkan değer codeCharacterArray indexi oluyor ve o indexteki değer alınıp, koddaki hanesindeki karakter ile aynımı diye kontrol ediliyor.

2) Case 2 ‘in kontrolu için SalesSlipCaseController’ı kullanabilirsiniz.

**CheckCode** : Kullanıcının yüklediği kodun doğruluğunu kontrol etmek.

Request Type : Post

Path : …/api/SalesSlipCase/ SalesSlip

Body : responseJson

Response : List<SlipSummary>

public class SlipSummary

{

public int Line { get; set; }

public string Text { get; set; }

}

X ve y eksenleri üzerine çalışmalarımı yaptım aynı satırda olan değerlerin y ekseninde değişim 8-10 değerleri arasında oluyor

Bunun için bir gap değeri verdim, gapvalue = 10;

Bu gap değeri ile hangileri aynı satırda olduğunu anladım ve hepsine idler verdim.

Aynı idde olan verileri grupladım ve x eksenindeki verileri sayesinde olması gereken texti çıkardım.

Y eksenindeki veriler sayesindede kaçıncı satırda olmaları gerektiğini çıkardım ve ona göre sıraladım.