T.C.

ALTINBAŞ ÜNİVERSİTESİ

Fen Bilimleri Enstitüsü / Bilişim Teknolojileri Bölümü

MOBİL JET ÖDEME SİSTEMİ YAZILIM ALTYAPI OLUŞTURULMASI

Tolga KISAOĞULLARI

Yüksek Lisans Tezi

Danışman

Prof. Dr. Oğuz Bayat

İstanbul, 2019

MOBİL JET ÖDEME SİSTEMİ YAZILIM ALTYAPI OLUŞTURULMASI

TOLGA KISAOĞULLARI

Fen Bilimleri Enstitüsü

Yüksek Lisans Tezi

ALTINBAŞ ÜNİVERSİTESİ

2019

Bu tezi okuduğumuzu; kapsam ve nitelik bakımından Yüksek Lisans tezi olarak yeterli bulduğumuzu beyan ederiz.

Prof. Dr. Oğuz BAYAT
Danışman

Sınav komitesi:

Prof. Dr. Oğuz BAYAT	Mühendislik ve Doğa Bilimleri Fakültesi, Altınbaş Üniversitesi	_____
Prof. Dr. Osman Nuri UÇAN	Mühendislik ve Doğa Bilimleri Fakültesi, Altınbaş Üniversitesi	_____
Prof. Dr. Hasan Hüseyin BALIK	Hava Harp Okulu, Milli Savunma Üniversitesi	_____

Bu tezin Yüksek Lisans tezi olarak bütün şartları sağladığını beyan ederim.

Dr. Öğr. Üyesi Oğuz ATA
Anabilim Dalı Başkanı

Fen Bilimleri Enstitüsü kabul tarihi:

____/____/____

Prof. Dr. Oğuz BAYAT
Enstitü Müdürü

Bu tezdeki tüm bilgilerin akademik kurallara ve etik davranışlara uygun olarak edinildiğini ve sunulduğunu beyan ederim. Ayrıca, bu kuralların ve davranışların gerektirdiği şekilde, bu çalışmada, orijinal olmayan tüm materyalleri ve sonuçları tamamen alıntı yaptığımı ve referans gösterdiğimi de beyan ederim.

Tolga KISAOĞULLARI

İTHAF

Bu tez çalışmasında değerli fikir ve tecrübelerini bizden eksik etmeyen sayın danışman hocamız Prof. Dr. Oğuz BAYAT'a, varlıklarıyla ve fikirleriyle bana destek olan aileme ve sevgili eşime, meslektaşlarıma, iş arkadaşlarıma, firmama ve tezimizi birlikte hazırladığımız değerli dostum Tevfik ELMAS'a en içten sevgi ve saygı duygularımı belirtmeyi bir borç bilir, şükranlarımı sunarım.

ÖN SÖZ

Hazırlamış olduğumuz bu tez çalışmamızda hedefimiz insanların alışveriş tecrübelerini arttırmak, daha hızlı ve kolay bir alışveriş deneyimi sunmaktır. Bunun yanında sistemde üye iş yeri diye tabir ettiğimiz mağaza ve market gibi bu sistemi kullanacak kurumların da yararına olacağını düşündüğümüz; hangi müşterisinin ne tarz ürünler aldığı, ne kadarlık harcama yaptığı ya da X isimli müşterisinin ödemesini yarım bırakması gibi durumları takip edebildiği ve daha birçok avantajı da yakalayabileceği bir ödeme sistemi oluşturduk. Yaygın kullanımda birçok artı katacağına inandığımız bu sistemde hedefimiz sadece alışveriş deneyimini arttırmanın yanında ülkemize de değer katan bir hizmet olmasını hedeflemekteyiz.

ÖZET

MOBİL JET ÖDEME SİSTEMİ YAZILIM ALTYAPI OLUŞTURULMASI

Tolga KISAOĞULLARI

Yüksek Lisans, Bilişim Teknolojileri, Altınbaş Üniversitesi

Danışman: Prof. Dr. Oğuz BAYAT

Tarih: Haziran, 2019

Sayfa Sayısı: 32

Ödeme sistemleri her geçen gün çeşitlilik kazanarak yaygınlaşmaktadır. Her ödeme sisteminin temel amacı müşteriye daha iyi bir alışveriş deneyimi kazandırmaktadır. Bu sistemde de amacımız sadece müşteriye daha hızlı, güvenli ve kolay bir alışveriş deneyimi sağlamak değil ayrıca bu sistemi kullanacak olan mağaza / marketlerin de faydalanabileceği satış öncesi ve sonrası faydalı ve kullanışlı özellikler sağlamaktadır.

Anahtar Kelimeler: Hızlı ödeme, Mobil ödeme, Mobil jet kasa

ABSTRACT

MOBILE JET CASHBOX SOFTWARE INFRASTRUCTURE

Tolga KISAOĞULLARI

M.Sc., Information Technologiest, Altınbaş University

Danışman: Prof. Dr. Oğuz BAYAT

Tarih: Haziran, 2019

Page: 32

As payment systems become widespread worlwide, It cause serious changes in recent years. what lies behind those changes is to provide faster, easier and more secure payment experience to their customers. Our aim with this study is to make a positive difference in terms of speed, easiness and security, where it provides big advantages for both customers and targeted stores that use the system

Keywords: Fast Payment, Mobile Payment, Mobile Jet Cashbox

İÇİNDEKİLER

ŞEKİL LİSTESİ

1. GİRİŞ

Ödeme yöntemlerine baktığımızda zaman içerisinde birçok değişikliğe uğradığını görmekteyiz. Bu değişiklikler bir malın karşılığında başka bir malın verilmesiyle başlayıp sonrasında temelinde para olan birçok ödeme yöntemi ile devam etmiştir. Günümüzde de yaygın olarak kredi kartı, senet, ticket vs. gibi ödeme yöntemleri nakit ödeme ile birlikte kullanılmaktadır.

Ödeme yöntemlerinin zaman içerisinde çeşitlilik kazanmasının birçok sebebi vardır. Ödemelerin daha hızlı, daha güvenli, daha sorunsuz, daha kolay olması gibi arayışlar bu yöntemlerin doğuş noktası olmaktadır. Yeni yöntemler geliştikçe beraberinde de başka sorunlar getirmektedirler. Hızın getirdiği güvenlik sorunları, güvenliğin getirdiği yavaşlık, kolaylığın getirdiği iş geliştirme kompleksitesi ve iş maliyeti sorunlarını beraberinde getirmektedir.

2. MOBİL ÖDEME SİSTEMİ

Mobil ödeme sistemi; en basit haliyle müşterilerin kayıt olabildikleri ve kayıt oldukları bu sisteme para aktararak ödeme yapabildikleri ve mağaza/market benzeri kurumların da ödeme alabilecekleri online bir platformdur. Bu sistemde nakit ödeme ve kredi kartı ile ödemede oluşan ve bu sistemlerin kullanımından dolayı oluşan çevresel sorunların çözülmesi amaçlanmaktadır. Buradaki çevresel sorunlar kasa yapılarıdır. Günümüzde kullanılan en yaygın kasa tipleri standart bantlı kasalar ve jet kasa diye tabir edilen müşterinin kendi başına ürün okutup ödemesini yaptığı kasalardır.

Bu kasalarda ödeme işlemleri ya kredi kartı ile ya da nakit ödeme ile gerçekleşmektedir. Mevcut sistemlerde birçok sorun vardır ve bu sorunlardan hem müşteriler hem de mağaza / marketler etkilenmektedirler.

Şekil 2.1: Jet veya hızlı kasa diye tabir edilen kasa modeli [1]

Şekil 2.2: Bantlı kasa modeli [2]

2.1 MEVCUT ÖDEME ARAÇLARINDA YAŞANAN SORUNLAR

Mevcut ödeme sistemlerinin avantajları olduğu gibi birçok sorunu da vardır. Bu sorunların bazıları kasa modellerinden kaynaklandığı gibi bazıları da nakit ve kredi kartı gibi kullanılan ödeme araçlarından kaynaklanmaktadır. Bu sorunları başlıca şöyle sıralayabiliriz:

2.1.1 Para Üstü İçin Beklemek

Mevcut sistemde normal kasalar ve hızlı ödeme kasaları bulunmaktadır. Bu her iki sistemde de para üstünü alma sorunu yaşanmaktadır. Normal kasalarda para üstünün sayılmasından dolayı hem işlemi yapan müşteri hem de sıra bekleyen diğer müşteriler etkileniyor. Bazen kasada yeterli miktar çıkmayabiliyor, bu sefer başka kasalardan tedarik etme işlemleri başlıyor veya daha ufak

çaplı bir yerde alışveriş yapılıyorsa ve yeterli para üstü çıkmama durumu oluşuyorsa müşteriye parasının bozdurup getirilmesini söyleyerek müşteri gönderilebiliyor**.**

Şekil 2.3: Sık sık yaşanan kasa yoğunluğu [3]

Jet kasalarda ise benzer durumların yanında ek olarak cihazda nakit para bittiğinde ya da yeterli para bulunmadığında müşterinin bir çalışandan yardım istemesi gerekmekte ki o çalışan kasaya para koyup müşteri kaldığı yerden ödemeye devam edebilsin. Tabi burada her çalışan bu para yükleme işlemini gerçekleştiremiyor. Mağazada bu işi yapan belli başlı görevliler bulunduğu için doğru çalışanı bulabilme gibi bir sorun da oluşuyor.

2.1.2 İade İşlemlerinin Zorluğu

İade işlemleri alışverişlerde en çok sorun yaşanan ve zahmetli süreçlerden birisi olarak karşımıza çıkmaktadır. Mevcut sistemlerde iade işlemi için önce mağazaya veya markete gidip müşteri hizmetlerine gitmek gerekiyor. Burada sıra varsa sıra bekleniyor, daha sonra kredi kartı ile ödeme yapıldıysa karta iade, nakit ödeme yapıldıysa nakit ödeme yapılıyor. Fakat iade işleminin en başında ürünün fişi elinizde bulunması gerekiyor. Aksi takdirde çoğu yerde iade işlemi gerçekleştirmek mümkün değil ya da çok fazla uğraşmak gerekiyor.

2.1.3 Çok Fazla Yer Kaplamaları

Normal bir kasa ile ürün bandı büyük boyutlara sahip, aynı şekilde jet kasalarda nispeten normal kasalara göre daha ufak olsalar da oldukça büyük cihazlar. Bu cihazların en büyük problemleri de büyük olmalarıyla beraber kolay yer değiştiremez olmaları.

2.1.4 Çok Fazla Maliyetli Olmaları

Standart kasalar ve jet kasalar maliyet açısından pahalı cihazlardır. Sadece satın alma maliyeti değil aynı zamanda bakım onarım maliyetleri de bulunmaktadır.

2.1.5 Sorun Yaşandığında Çözümün Uzun Sürmesi

Satın alma işlemi sırasında sorun yaşandığında, yaşanan sorunun çözümü genelde kolay olmuyor. Standart kasada işlem yapılırken unutulan ya da fazladan girilen bir ürünün işleme dahil edilmesi-çıkarılması uzun zaman alıyor. Fazla girilen bir üründe ise kasiyerin yetkili birisini çağırması gerekiyor ki iptal işlemi başlasın ve ürün iptal edilsin. Bu aşamada görevliye ulaşmak ve onun gelip sorunu çözmesi tamamen o kişinin ulaşılabilir olmasına bağlı. Ödeme işlemine geçilmiş ve okutulması unutulan ürün veya ürünler var ise bu sefer mevcut işlemi bitirip yeniden okuma yaparak alışveriş ikinci defa başlatılıyor. Yani mevcut ödeme üzerine ekleme yapıp devam edilemiyor. İşin bir de jet kasa kısmı var ki burada ise sorunlar daha fazla. Ürün okutma işlemine başlandığı zaman eğer ürünleri okutma sırasında biraz bekleme yaparsanız (bu bekleme cihazın ürünü okumamasından kaynaklı oluyor genellikle) yeni ürün ekleyemiyorsunuz, bu durumla karşılaşıldığında ya ödeme işlemini bitirip tekrardan diğer ürünler için işlem başlatılması gerekiyor ya da görevli personeli onun müdahale etmesi bekleniyor. Bu müdahale işlemi görevli personelin sahip olması gereken bir kâğıt yardımı ile oluyor, bu kâğıdı jet kasaya okutup, daha sonra belli sıfırlama ayarları yapıp işlemi tekrardan gerçekleştiriyor. Böyle bir sorunla karşılaşıldığında doğru görevliyi bulmanın dışında eğer o kişi kâğıdı kaybetme ya da bulamama ihtimalinden doğan bir problem de var.

2.1.6 Kullanımlarının Karmaşık Olması

Jet kasalarda ödeme işlemi esnasında bir sürü seçenek karşımıza çıkmaktadır. Ürünü okutma sırasında ürünün nereden okutulacağı karıştırılabiliyor, bunun dışında ödeme kısmına

geçildiğinde kredi kartı, nakit ve banka kartı gibi birden fazla ödeme seçeneği çıkıyor, müşteri burada da karar vermede zorlanabiliyor. Kart ile ödeme yapılacaksa kartı nasıl okutacağı, nakit ile ödemede de paranın nereden cihaza koyulacağı karmaşıklığı oluşuyor.

Şekil 2.4: Bir sürü ödeme yöntemi sunan bir jet kasa [4]

2.1.7 Kart ile Ödemede Yaşanan Sorunlar

Standart kasalarda kart ile ödeme esnasında yaşanabilecek ya da yaşanan sorunların başında; kartın deforme olmasından dolayı kartın okunamaz hale gelmesi, tekrar tekrar deneme yapılmasından dolayı da birçok müşterinin mağdur edilmesi ve uzayan kasa sırası sayılabilir. Jet kasalarda da aynı sorun oluşmakta bunun yanında müşteri kartı cihaza nasıl okutmasını bilmeme durumunda da yine işlemin uzaması söz konusu.

2.1.8 Nakit Ödemede Yaşanan Sorunlar

Standart kasalarda çok ucuz bir ürün için büyük banknotlar ile ödeme yapılmak istendiğinde kasiyerin kasasındaki bozuk paraların yetersiz olabiliyor ya da işlem yapmak istemeyebiliyor. Çünkü kasasındaki bozuk paraları azalıyor. Yırtık paralar ile alışveriş yapılmak istendiğinde sorunlar yaşanıyor her yer bu paraları kabul etmiyor. Jet kasalarda ise ürünleri okutup ödeme işlemine geçildiğinde nakit ödeme sırasında para üstü kalmadığında görevli bir personel bulunması gerekiyor ve kasaya para yüklemesi gerekiyor. Bazen de nakit parada herhangi bir deformasyon olmamasına rağmen kasanın parayı kabul etmeme durumu da oluşuyor.

2.1.9 İnsan Faktörünün Olması

Standart kasalarda çalışan kasiyerler yetersiz, tecrübesiz ya da yaşadığı herhangi bir sorundan dolayı dalgın olabiliyor. Bu gibi problemlerden dolayı oluşabilecek sorunlara örnek verecek olursak; sahte para ile alışveriş yapılmak istendiğinde kasiyer bunu fark etmeyebilir ve bundan dolayı mağaza ya da market zarara uğrayabilir. Ya da müşteriye eksik para üstü verilip müşteri mağdur edilebilir, fazla para üstü verilip mağaza ya da market zarara uğrayabilir. İnsan faktörü olduğu için birçok sebepten birçok hata oluşabilir.

2.2 MOBİL ÖDEME SİSTEMİ KULLANICI TİPLERİ

Sistemde iki tip kullanıcı bulunmaktadır. Bunlardan ilki ödeme alan anlaşmalı iş yerleri. Bu iş yerleri hesapları anlaşma yapıldıktan sonra sistem admini tarafından açılmakta. İş yeri hesabı açıldıktan sonra kendilerine ait yönetim panelinde, iş yerine ait olan ödeme cihazlarını görebilmekteler ve bu cihazların hangi mağazalarında aktif olduklarını, hangi cihazdan kaç tane ya da ne kadarlık alışveriş yapıldığı, kaç tane işlemin beklemede olduğu ve kaç tane işlemin de iptal olduğu görülebilmektedir. Aynı zamanda iş yerleri ödeme yapan ya da işlemi iptal eden, yarıda bırakan müşterileri görüp bunlara özel kampanyaların çıkılmasını da talep edebilir.

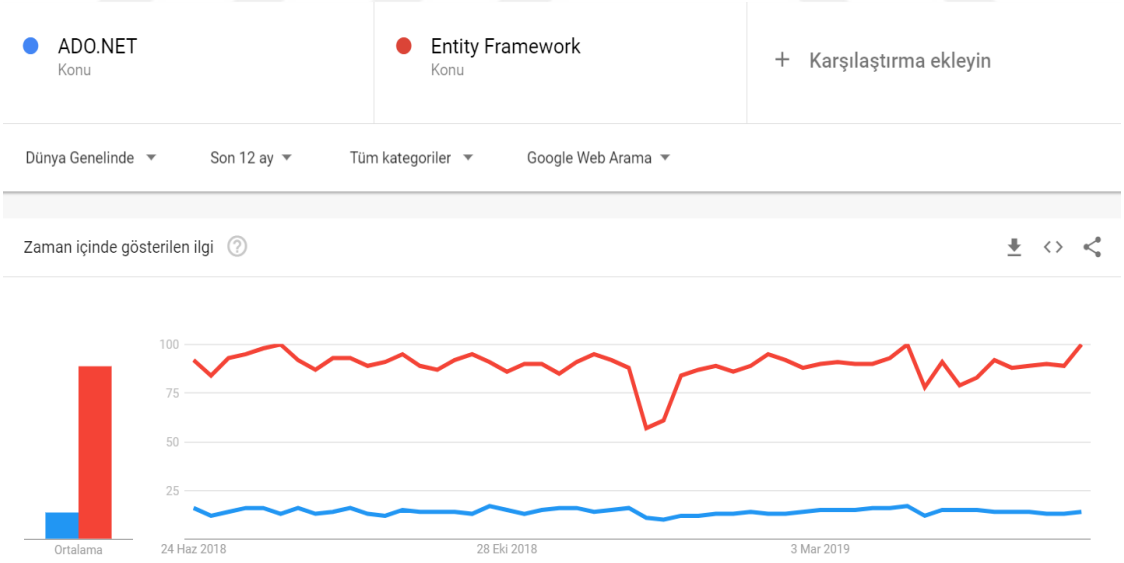
Diğer kullanıcı tipi ise ödeme yapan müşteri hesabı. Kullanıcılar istedikleri an sisteme kayıt olabilirler. Atm veya banka hesapları üzerinden sisteme para yükleyip üye iş yerlerine ait mağaza veya marketlerden alışveriş yapabilirler ve bu alışveriş geçmişlerini sistemden görebilir. Hangi kategoride ne kadarlık ve kaç tane işlem yapmış olduğunu listeleyebilir.

3. ALT YAPI VE VERİ TABANI İLİŞKİSİ

Mobil ödeme sisteminin alt yapısı restul api modülleri üzerinde oturmaktadır. Bu yapıda Get ve Post metodları kullanılmıştır. Rest api tercih edilmesinin sebeplerinin başında iletişimin daha hafif ve hızlı olmasının yanında alt yapıda kullanılan metodların hem mobil uygulamada hem de web sitesinde kullanılması amacı gelmektedir. Böylece yazılan kodun tek bir yerden yönetilmesi ve hızlı bir şekilde geliştirilmesi kolay hale gelmiştir.

Yazılım dili olarak C# 7.0, framework olarak Asp.Net 4.6.1 ve veri tabanı olarak MsSql kullanılmıştır.

Projede api ile veri tabanı arasında ORM (Object Relational Mapping) olarak isimlendirilen Microsoft tarafından geliştirilen EntityFramework 6.2.0 versiyonu kullanılmaktadır. EntityFramework Ado.Net alt yapısını kullanmaktadır. Ado.Net ise Asp.Net üzerinde SQL sorgularını çağırabildiğimiz bir kütüphanedir.



Şekil 3.1: Google aramalarına göre trend grafiği [5]

Ado.Net doğası gereği EF ye göre performans açısından çok daha hızlı bir kütüphanedir fakat çok daha hızlı olmasına rağmen kullanışlılık, geliştirme yapma, kod bakımı ve daha az kod yazılması açısından bakıldığında da EF ciddi anlamda ön plana çıkmaktadır.

```
using (var context = new CompanyContext())
{
    var emp = context.Employee;
}
```

Şekil 3.2: Örnek bir entity framework sorgusu [6]

```
SqlCommand command = new SqlCommand("SELECT * FROM Employee;", connection);

connection.Open();
SqlDataReader reader = command.ExecuteReader();
if (reader.HasRows)
{
  while (reader.Read())
  {
    // do some coding
  }
  else
  {
    //  do some coding
  }
  reader.Close();
}
```

Şekil 3.3: Örnek bir Ado.Net sorgusu [6]

Örnek sorgulara bakıldığında fark rahatlıkla görülmektedir. Ado.Net açısından bakıldığında yazılan SQL sorgusunun kompleksliği arttıkça kodu okuması ve yazması da o kadar artmaktadır. Ödeme hizmeti veren projelerde performans çok önemli bir parametre olmasına rağmen, hız her şey demek değildir, bu tarz projelerde kod kalitesi, olabilecek kodsal hataların azaltılması ve kod bakımı çok daha önemlidir. Çünkü işin en temelinde para ile ilgili işlemler yapıldığı için hızdan daha çok bug diye tabir ettiğimiz kodsal hatanın olabildiğince az olması temel hedeflerden biridir.

Performans için grafiklere baktığımızda ise;

Entity Framework Results

RUN	PLAYER BY ID	PLAYERS FOR TEAM	TEAMS FOR SPORT
1	1.64ms	4.57ms	127.75ms
2	0.56ms	3.47ms	112.5ms
3	0.17ms	3.27ms	119.12ms
4	1.01ms	3.27ms	106.75ms
5	1.15ms	3.47ms	107.25ms
6	1.14ms	3.27ms	117.25ms
7	0.67ms	3.27ms	107.25ms
8	0.55ms	3.27ms	110.62ms
9	0.37ms	4.4ms	109.62ms
10	0.44ms	3.43ms	116.25ms
Average	**0.77ms**	**3.57ms**	**113.45ms**

Şekil 3.4: Entity Framework ile yapılan 10 adet sorgu sonucu [7]

ADO.NET Results

RUN	PLAYER BY ID	PLAYERS FOR TEAM	TEAMS FOR SPORT
1	0.01ms	1.03ms	10.25ms
2	0ms	1ms	11ms
3	0.1ms	1.03ms	9.5ms
4	0ms	1ms	9.62ms
5	0ms	1.07ms	7.62ms
6	0.02ms	1ms	7.75ms
7	0ms	1ms	7.62ms
8	0ms	1ms	8.12ms
9	0ms	1ms	8ms
10	0ms	1.17ms	8.88ms
Average	**0.013ms**	**1.03ms**	**8.84ms**

Şekil 3.5: Ado.Net ile yapılan 10 adet sorgu sonucu [7]

Performans açısından baktığımızda id'sine göre veri tabanından çekilen kullanıcıların 10 adet sorgunun ortalama değerleri arasında neredeyse 60 kat fark bulunmaktadır. Ama yukarda belirttiğimiz gibi performans her zaman en önemli kriter değildir.

Projede EF kullanmamızın bir diğer sebeplerinden birisi de üç tip veri tabanı oluşturma yaklaşımından biri olan CodeFirst yaklaşımı kullanılmaktadır. Diğer iki yaklaşım DatabaseFirst ve ModelFirst'tür. Bu üç yaklaşımdan ModelFirst diğer ikisine göre çok daha az

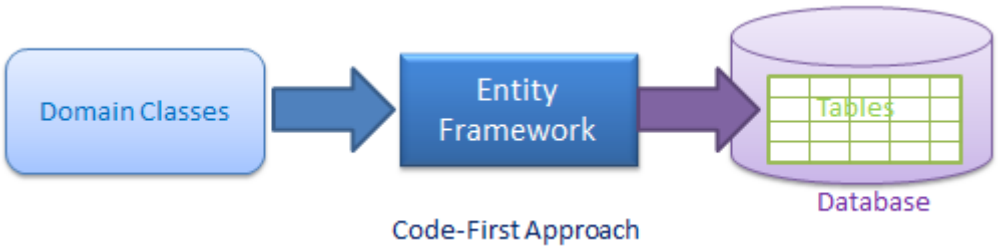
kullanılmaktadır bundan dolayı kıyaslama yapma kısmında CodeFirst ve DatabaseFirst arasında bir seçim yapmamız gerekti.

3.1 CODE FIRST VS DATABASE FIRST

Database First yaklaşımına bakacak olursak, bu yaklaşımda ilk önce veri tabanında tablolar oluşturulur ve daha sonra bu tablolar Entity Framework diagramına tanımlatılır. Bu yaklaşım Code First'e göre daha eski kalmaktadır. Genellikle veri tabanı daha önceden oluşturulmuş sistemlerde bu yaklaşım uygulanmaktadır ve veri tabanında yapılan bir geliştirme, yeni kolan ya da tablo ekleyip silme işlemi gerçekleştirilmek istendiğinde bu işlemler manuel olarak veri tabanından yapılmaktadır ve daha sonra EF diagramı kod tarafında güncellenmektedir. Dolayısıyla çok daha fazla iş yükü getirmektedir. [8]

Şekil 3.6: Database First yaklaşımı diagramı [9]

Code First yaklaşımına baktığımızda, bütün veri tabanı işlemleri yani yeni tablo ekleme, çıkarma, güncelleme gibi işler kod kısmında yapılmaktadır. Ayrıca veri tabanına gitmeye gerek yoktur. Dolayısıyla artı kısımlarına baktığımızda yazılımcının veri tabanına erişip müdahale etmesini engellemiş oluyoruz. Bütün operasyonu tek bir yerden (kod kısmı) kontrol etmiş oluyoruz.



Şekil 3.7: Code First yaklaşımı diagramı [9]

Code First yaklaşımında veri tabanı için gerekli olan tablolar C# tarafında class dediğimiz nesneler ile temsil edilmektedir. Bu veri tabanı nesneleri eklendiğinde Entity Framework tarafında normal nesnelerden farklı olduğunu ve bu nesnelerin veri tabanı için olduğunu belirtmemiz için DbSet property'leri ile ilişkilendirerek Entity Framework'ün DataContext yani veri tabanı nesnesine eklememiz gerekmektedir. [10]

```
public class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public DateTime? DateOfBirth { get; set; }
    public byte[]  Photo { get; set; }
    public decimal Height { get; set; }
    public float Weight { get; set; }

    public Grade Grade { get; set; }
}
```

Şekil 3.8: Örnek veri tabanı modeli [11]

```
public class Grade
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }

    public ICollection<Student> Students { get; set; }
}
```

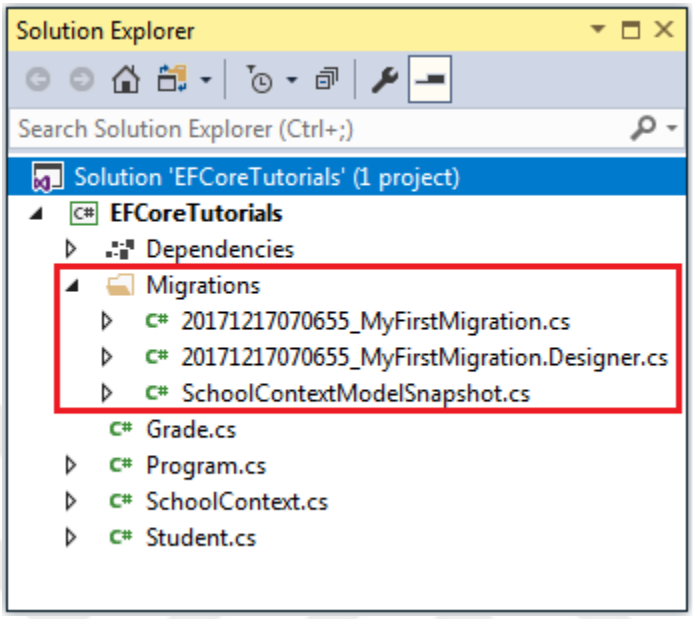
Şekil 3.9: Örnek veri tabanı modeli [11]

```
namespace EF6Console
{
    public class SchoolContext: DbContext
    {
        public SchoolContext(): base()
        {

        }

        public DbSet<Student> Students { get; set; }
        public DbSet<Grade> Grades { get; set; }
    }
}
```

Şekil 3.10: Veri tabanı ile ilişkilendirme kısmı [11]

Bu işlemleri yaptığımızda ilk olarak kullandığımız IDE'nin "Command Line" kısmından "Migration" yani veri tabanına erişimimizi aktif etmemiz gerekmektedir. Bunun için Command Line konsoluna "Enable-Migration" sorgusunu yazıp daha sonra oluşturduğumuz nesneler için migration oluşturmamız gerekmektedir. Ör: Add-Migration [Migration Adı].

Migration oluşturma işlemi başarılı bir şekilde gerçekleştiğinde projemizde Migrations adında bir klasör oluşur ve her bir migration işlemi buraya kaydedilir. Dolayısıyla Code First'in bir diğer artısı da geriye yönelik olarak neler yaptığımız kontrol edebilmemizdir.



Şekil 3.11: Migration örneği [12]

3.2 PROJEDEKİ VERİ TABANI MODELLERİ

3.2.1 User (Kullanıcı Sınıfı)

Kullanıcı verilerilerini barındıran modeldir. Asp.Net Identity kütüphanesinde bulunan IdentityUser modelinden miras alınarak türetilmiştir. Bundaki amaç Asp.Net Identity sisteminin kullanıcı alt yapısını kullanabilmek ve giriş, kayıt olma, şifre değiştirme gibi yapılarını kullanabilmektir.

```
public class User : IdentityUser
    {
        public User()
        {
            Transactions = new List<Transaction>();
            Balances = new List<UserBalace>();
            TransactionRequests = new List<TransactionRequest>();
            Payments = new List<Payment>();
            BankAccounts = new List<UserBankAccount>();
        }

        /// <summary>
        /// First name of the user
        /// </summary>
        [MaxLength(20)]
        public string FirstName { get; set; }

        /// <summary>
        /// Last name of the user
        /// </summary>
        [MaxLength(30)]
        public string LastName { get; set; }

        public decimal FeePercentage { get; set; }

        public string MerchantId { get; set; }

        public Merchant Merchant { get; set; }

        public List<UserBalace> Balances { get; set; }

        public ICollection<Transaction> Transactions { get; set; }

        public ICollection<UserBankAccount> BankAccounts { get; set; }

        public ICollection<TransactionRequest> TransactionRequests { get; set; }

        public ICollection<Payment> Payments { get; set; }
    }
```

Şekil 3.12: Kullanıcı tablosu modeli

3.2.2 BaseEntity<T> (Temel Model)

Bu model genel bütün modellerin temelinde kullanılır ve her model bundan kalıtım alarak türetilir. Buradaki T bu nesnenin genel bir tip alacağını belirtir ve BaseEntity nesnesinden türetilen yeni nesnenin Id özelliğinin tipi bu kısımda belirlenir. Bu nesnenin içerisinde her modelin ihtiyacı olan özellikler vardır bunlar, veri tabanında tablo için gerekli olan Id kolonu ve oluşturulma (CreatedAt) tarihidir.

Abstract bir model olmasından dolayı new anahtar kelimesiyle yeni bir nesne oluşturulamayacağı anlamını taşımaktadır. Dolayısı ile bu nesne sadece başka nesnelere miras olabilir kendi başına kullanılamaz.

```
public abstract class BaseEntity<T>
    {
        public BaseEntity()
        {
            CreatedAt = DateTime.UtcNow;
        }

        [Key]
        public T Id { get; set; }

        [Index]
        public DateTime CreatedAt { get; set; }
    }
```

Şekil 3.13: Bütün sınıfların miras aldığı en temeldeki sınıf

3.2.3 UserBalance (Kullanıcı Bakiyesi)

BaseEntity<int> dan kalıtım alarak türetilmiş bir modeldir. Buradaki int tipi daha önce belirttiğimiz gibi Id özelliğinin tipini belirtmektedir.

Bu nesne kullanıcının bakiyesini temsil etmektedir. Şu an sadece tek bir para birimine göre bakiye oluşturulsa da sistemsel olarak çoklu para birimi desteklenmektedir. Bu özellik Currency enum parametresi ile gerçekleşmektedir. Currency enum modelinde şimdilik TRY (Türk Lirası), USD (Dolar) ve EUR (Euro) para birimleri bulunmaktadır. Sisteme kolaylıkla yeni bir para birimi eklenebilmektedir.

```
public class UserBalace : BaseEntity<int>
    {
        public UserBalace()
        {
            Currency = Currency.TRY;
            TotalBalance = 0;
            LockedBalance = 0;
        }

        public Currency Currency { get; set; }

        public decimal TotalBalance { get; set; }

        public decimal LockedBalance { get; set; }

        public decimal AvailableBalance => this.TotalBalance + this.LockedBalance;

        [Timestamp]
        public byte[] RowVersion { get; set; }

        [MaxLength(100)]
        public string UserId { get; set; }

        public User User { get; set; }
    }
```

Şekil 3.14: Kullanıcı bakiyesi

Şekil 3.15: Para birimleri

3.2.4 Merchant (Üye İşyeri)

Bu model anlaşmalı olunan ödeme kabul eden iş yerlerini temsil etmektedir. BaseModel<string> nesnesinden kalıtım alınarak türetilmiştir.

```
public class Merchant : BaseEntity<string>
    {
        public Merchant()
        {
            Id = Guid.NewGuid().ToString();
            IsActive = true;
            Balances = new List<MerchantBalance>();
            PaymentDevices = new List<PaymentDevice>();
            MerchantUsers = new List<User>();
        }

        [MaxLength(100)]
        public string Name { get; set; }

        public bool IsActive { get; set; }

        public decimal FeePercentage { get; set; }

        public ICollection<MerchantBalance> Balances { get; set; }

        public ICollection<PaymentDevice> PaymentDevices { get; set; }

        public ICollection<User> MerchantUsers { get; set; }
```

Şekil 3.16: Üye iş yeri

3.2.5 MerchantBalance (Üye İş yeri Bakiyesi)

Üye iş yerinin bakiyesini temsil etmektedir. Aynı kullanıcıda olduğu gibi Currency enum modelini kullanarak birden fazla para birimini desteklemektedir. BaseModel<int> nesnesinden kalıtım alınarak türetilmiştir.

```
public class MerchantBalance : BaseEntity<int>
    {
        public MerchantBalance()
        {
            Currency = Currency.TRY;
            TotalBalance = 0;
            LockedBalance = 0;
        }

        public Currency Currency { get; set; }

        public decimal TotalBalance { get; set; }

        public decimal LockedBalance { get; set; }

        public decimal AvailableBalance => this.TotalBalance + this.LockedBalance;

        [Timestamp]
        public byte[] RowVersion { get; set; }

        [MaxLength(100)]
        public string MerchantId { get; set; }

        public Merchant Merchant { get; set; }
    }
```

Şekil 3.17: Üye iş yeri bakiyesi

3.2.6 Transaction (İşlem)

Bu model bütün işlem hareketlerinin ortak noktalarını barındıran abstract modeldir. Kendi başına new anahtarı ile kullanılamamaktadır. Başka modellere miras kalıtımı ile uygulanır. BaseEntity<string> modelinden türetilmiştir.

```
public abstract class Transaction : BaseEntity<string>
    {
        [MaxLength(100)]
        public string UserId { get; set; }

        public User User { get; set; }

        public decimal Amount { get; set; }

        public decimal Fee { get; set; }

        [MaxLength(250)]
        public string Description { get; set; }
    }
```

Şekil 3.18: İşlem hareketi modeli

3.2.7 Payment (Ödeme)

Bu model ödeme işlemini temsil etmektedir. Transaction nesnesinden türetilmiş bir modeldir. Kullanıcının alışveriş bilgilerini tutmaktadır. İçerisinde bulunan CartId yani Sepet bilgisi ile kullanıcının alışveriş detaylarına ulaşılabilmektedir.

```
public class Payment : Transaction
    {
        public Payment()
        {
            Id = Guid.NewGuid().ToString();
        }

        [MaxLength(100)]
        public string CartId { get; set; }

        public Cart Cart { get; set; }

        public Merchant Merchant { get; set; }
    }
```

Şekil 3.19: Ödeme sınıfı

3.2.8 TransactionRequest (İşlem İsteği)

Bu model işlem isteği oluşturmak için kullanılan abstract bir modeldir. BaseEntity<string> modelinden kalıtım alınarak türetilmiştir. Amacı ödeme isteği, para yatırma ve çekme isteklerini karşılamaktır. Her isteğin 3 aşaması bulunmaktadır. Bunlar: “Bekliyor”, “İptal Edildi” ve “İşlendi”.

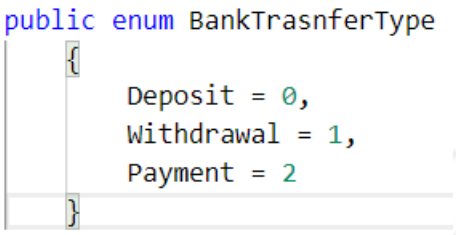
```
public abstract class TransactionRequest : BaseEntity<string>
    {
        public TransactionRequest()
        {
            Id = Guid.NewGuid().ToString();
            Status = TransactionRequestStatus.Pending;
        }

        public string UserId { get; set; }

        public User User { get; set; }

        public decimal Amount { get; set; }

        public decimal Fee { get; set; }

        public TransactionRequestStatus Status { get; set; }

        [Index]
        public BankTrasnferType Type { get; set; }
    }
```

Şekil 3.20: İşlem isteği sınıfı

```
public enum TransactionRequestStatus
{
    Pending = 0,
    Cancelled = 1,
    Processed = 2
}
```

Şekil 3.21: İşlem isteği durumu

İşlem tipleri olarak da 3 tip bulunmaktadır: “Para Yatırma”, “Para Çekme” ve “Ödeme”.



Şekil 3.22: Banka transfer tipleri

TransactionRequest modeli oluşturulurken durumu “Pending” yani bekliyor olarak oluşturulur.

3.2.9 Cart (Sepet)

Bu model TransactionRequest modelinden miras alınarak türetilmiştir ve ödeme işlemi isteğini karşılamaktadır. Mağazalardaki ödeme cihazları tarafından oluşturulan sepetler bu model tarafından sağlanmaktadır ve her oluşturulan sepet aslında bekleyen bir ödeme işlemi isteğidir

```
public class Cart : TransactionRequest
    {
        [MaxLength(100)]
        public string DeviceId { get; set; }

        [JsonIgnore]
        public PaymentDevice Device { get; set; }

        public string MerchantId { get; set; }

        public Merchant Merchant { get; set; }

        public ICollection<CartItem> CartItems { get; set; }
    }
```

Şekil 3.23: Sepet sınıfı

3.2.10 CartItem (Sepetteki Ürün)

Bu model sepetteki her bir ürünü temsil etmektedir ve BaseEntity<string> modelinden kalıtım alınarak türetilmiştir. Her ürünün markasını, açıklamasını ve fiyatını tutmaktadır. Sepetin detay kısmı burasıdır.

```
public class CartItem : BaseEntity<string>
    {
        public CartItem()
        {
            Id = Guid.NewGuid().ToString();
        }

        public decimal Amount { get; set; }

        [MaxLength(50)]
        public string BrandName { get; set; }

        [MaxLength(150)]
        public string Description { get; set; }

        public bool IsRefunded { get; set; }
    }
```

Şekil 3.24: Sepet için nesne sınıfı

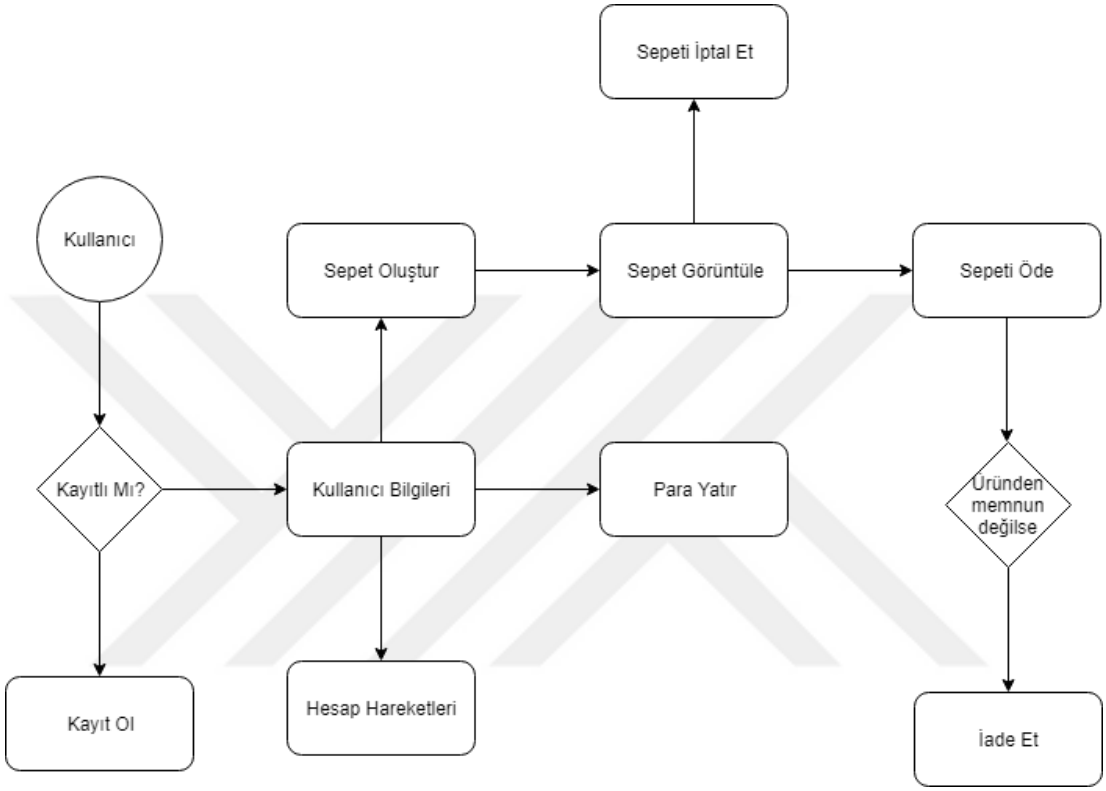
3.2.11 PaymentDevice (Mobil Ödeme Cihazı)

Bu model mağazalarda bulunan jet kasa alternatifi olan ödeme cihazlarını temsil etmektedir ve BaseModel<string> modeli tarafından miras alınarak üretilmiştir. Her cihaz tek bir üye işyerine ait olabilmektedir dolayısıyla kullanıcı tarafından oluşturulan sepetin hangi üye işyeri ve hangi üye işyerinin mağazasından yapıldığı bilgilerini taşımaktadır.

```
public class PaymentDevice : BaseEntity<string>
    {
        public PaymentDevice()
        {
            Id = Guid.NewGuid().ToString();
        }

        [MaxLength(100)]
        public string MerchantId { get; set; }

        public Merchant Merchant { get; set; }
    }
```

Şekil 3.25: Ödeme cihazı sınıfı

4. MOBİL ÖDEME SİSTEMİ AŞAMALARI



Şekil 4.1: Sistem diagramı

4.1 KAYIT OLMA (REGISTER)

Bu metod kullanılarak kullanıcılar sisteme kayıt olabilmektedirler. Metod HttpPost tipinde istekleri karşılayan ve parametre olarak "Ad", "Soyad", "E-posta" ve "Şifre" almaktadır. Kayıt olan kullanıcıların şifreleri Sha-256 olarak hashlenip kaydedilmektedir.

```
POST /auth/register HTTP/1.1
Host: [Host Adresi]
Content-Type: application/json
cache-control: no-cache
Postman-Token: 3ea2e0b9-3142-42e5-b8e0-b24392ddda9a

{
    "firstName":"Ad",
    "lastName":"Soyad",
    "email":"eposta@mail.com",
    "password":"112442"
}
```

Şekil 4.2: Postman sorgu örneği

4.2 GİRİŞ YAPMA (LOGIN)

Bu metod üye olmuş kullanıcılar tarafından sisteme giriş yapmak için kullanılır. HttpPost tipinde olan bu metod parametre olarak “E-posta” ve “Şifre” parametrelerini alır.

```
POST /auth/login HTTP/1.1
Host: [Host Adresi]
Content-Type: application/json
cache-control: no-cache
Postman-Token: 71727d3c-e318-439a-9d1d-fdbeeb7e93c5

{
    "email":"eposta@mail.com",
    "password":"112442"
}
```

Şekil 4.3: Postman sorgu örneği

4.3 PARA YÜKLEME

Giriş yapan kullanıcılar “Para Yükleme” sayfası altından şirketin banka hesaplarına havale veya EFT işlemi ile para gönderdikten sonra, bu sayfa altında para yükleme isteği oluşturmaktadırlar. Burada para gönderdikleri şirket banka adı, miktar ve para yatırmada girdikleri açıklama kısmını yazarak para yatırma isteğini oluştururlar. HttpPost tipinde bir metodudur.

```
POST /banktransfer/deposit HTTP/1.1
Host: 172.20.10.2:81
Content-Type: application/json
Authorization: Bearer O_hf9uY3jNIiEpvmnmz3F9_dM_Q4QfK2h6HWCu4cMF8dUbWfEfEEycQvpT30KlY10ajHFRVC
    9csTiYY4Fxb71EKeIdkhzoXZ3y1QN
    -Oxwr2luK6JKkIdM645xXCX2vIrxXA_FhCuZVappdV8vkewt140kvLJiAQQrgFM8hCPCQMyTjiqKPj0a9Mc9HFHf57
    I6tpUWQ4WqqDCqU0j3oyC-_QrH57hApVZgT7t74MZ05W3TcmVVB3cm4G6AZlbS0pFATuylk_c8MKA3
    -BbN095D6B_M45GmmgaOJwFjlMds9HiaJFlqwQHDMAqyTUox3IME8JwPW9r7hT1tncNh6iJwdoWXj8
cache-control: no-cache
Postman-Token: aeee7d48-191e-45b8-95f6-3fedf4482e81

{
    "companyBankAccountId" : 1,
    "amount": 25
}
```

Şekil 4.4: Postman sorgu örneği

4.4 İŞLEM GEÇMIŞI

Bu sayfa altında kullanıcılar geçmişe yönelik harcamalarını, para yatırma ve çekme işlemlerini detaylarıyla birlikte görebilmektedirler. HttpGet tipinde bir metoddur.

```
GET /user/transactions?pageNumber=1&amp; pageItemCount=10&amp; transactionType=0 HTTP/1.1
Host: [Host Adresi]
Authorization: Bearer O_hf9uY3jNIiEpvmnmz3F9_dM_Q4QfK2h6HWCu4cMF8dUbWfEfEEycQvpT30K1Y10ajHFRVC9
    csTiYY4Fxb71EKeIdkhzoXZ3y1QN
    -Oxwr2luK6JKkIdM645xXCX2vIrxXA_FhCuZVappdV8vkewt140kvLJiAQQrgFM8hCPCQMyTjiqKPj0a9Mc9HFHf57I
    6tpUWQ4WqqDCqU0j3oyC-_QrH57hApVZgT7t74MZ05W3TcmVVB3cm4G6AZlbS0pFATuylk_c8MKA3
    -BbN095D6B_M45GmmgaOJwFjlMds9HiaJFlqwQHDMAqyTUox3IME8JwPW9r7hT1tncNh6iJwdoWXj8
cache-control: no-cache
Postman-Token: 2826cfbe-9054-49d0-ba46-c61ff3c3826e
```

Şekil 4.5: Postman sorgu örneği

4.5 SEPET OLUŞTURMA

Bu kısım kullanıcının ulaşabildiği bir kısım değildir. Sadece mağaza/marketlerde bulunan ödeme cihazları tarafından oluşturulabilmektedir. Kullanıcı satın almak istediği ürünleri cihaza okuttuktan sonra kullanıcıdan cep telefonundan oluşturduğu QR kodu okutması beklenmektedir. Bu kodu okuttuktan sonra ödeme cihazı aslında sadece ödeme işleminde geçerli olan o kullanıcıya ait bir token elde eder. Bu elde ettiği token ile sunucuya sepet oluşturma isteği gönderir. Bu metod HttpPost metoddur. İşlemin başarılı olması durumunda “Pending” tipinde bir ödeme isteği (Cart) oluşturur.

```
POST /payment/cart HTTP/1.1
Host: 172.20.10.2:81
Content-Type: application/json
Authorization: Bearer O_hf9uY3jNIiEpvmnmz3F9_dM_Q4QfK2h6HWCu4cMF8dUbWfEfEEycQvpT30KlY10ajHFRVC
    9csTiYY4Fxb71EKeIdkhzoXZ3y1QN
    -Oxwr2luK6JKkIdM645xXCX2vIrxXA_FhCuZVappdV8vkewt140kvLJiAQQrgFM8hCPCQMyTjiqKPj0a9Mc9HFHf57
    I6tpUWQ4WqqDCqU0j3oyC-_QrH57hApVZgT7t74MZ05W3TcmVVB3cm4G6AZlbS0pFATuylk_c8MKA3
    -BbN095D6B_M45GmmgaOJwFjlMds9HiaJFlqwQHDMAqyTUox3IME8JwPW9r7hT1tncNh6iJwdoWXj8
cache-control: no-cache
Postman-Token: b4a806fa-7de8-4de1-a924-33ceb0c9ac25

{
    "deviceId":"64115d77-372e-4aa0-8e4d-8eaae30394a9",
    "cartItems":
    [
        {"amount":1.50,"brandName":"Ulker","description":"Tatli Biskuvi"},
        {"amount":0.50,"brandName":"Ulker","description":"Tuzu Biskuvi"}
    ]
}
```

Şekil 4.6: Postman sorgu örneği

4.6 SEPET GÖRÜNTÜLEME

Sepeti başarılı bir şekilde oluşturulan kullanıcı cep telefonu uygulaması üzerinden bu sepeti görebilmektedir. HttpGet metoddur.

```
GET /payment/cart?id=562576b8-ed95-4000-8fa3-8330fe42fe2f HTTP/1.1
Host: 172.20.10.2:81
Authorization: Bearer O_hf9uY3jNIiEpvmnmz3F9_dM_Q4QfK2h6HWCu4cMF8dUbWfEfEEycQvpT30KlY10ajHFRVC9
    csTiYY4Fxb71EKeIdkhzoXZ3y1QN
    -Oxwr2luK6JKkIdM645xXCX2vIrxXA_FhCuZVappdV8vkewt140kvLJiAQQrgFM8hCPCQMyTjiqKPj0a9Mc9HFHf57I
    6tpUWQ4WqqDCqU0j3oyC-_QrH57hApVZgT7t74MZ05W3TcmVVB3cm4G6AZlbS0pFATuylk_c8MKA3
    -BbN095D6B_M45GmmgaOJwFjlMds9HiaJFlqwQHDMAqyTUox3IME8JwPW9r7hT1tncNh6iJwdoWXj8
cache-control: no-cache
Postman-Token: 0d5679c3-9c34-462f-8d14-0af32c0d6231
```

Şekil 4.7: Postman Örneği

4.7 SEPET ÖDEME/İPTAL

Bu aşamada oluşturulan sepet kullanıcının mobil uygulamasında gözükmektedir. Kullanıcı eğer bakiyesi varsa ödeme işlemini gerçekleştirebilir ya da işlemi iptal edebilir. Eğer kullanıcı ödeme isteği işlemini iptal ederse, “Pending” tipinde olan sepet(Cart) “Cancelled” olarak güncellenir. Eğer kullanıcı işlemi onaylarsa bu sefer “Pending” olan sepet “Processed” olarak yani işlendi olarak güncellenir. İptal edilen veya ödenen bir işlem tekrardan işlenemez ya da “Pending” durumuna güncellenemez. Bu metodta HttpPost tipinde bir metoddur. Ayrıca bu metoda istek

yapılabilmesi için ödeme tipinde bir access token alınmış olması gerekmektedir, kullanıcının giriş işlemi yaparak aldığı token ile bu işlem gerçekleştirilememektedir. Bunun sebebi ise herhangi bir şekilde kullanıcı access token'ını çaldırması durumunda geçerlilik süresi çok daha kısa olan (3dk) ve sadece oluşturulmuş sepeti ödemeyi sağlayan token bilgisili çaldırmış olacak, bununla da çalan kişi herhangi bir işlem yapamayacak.

```
GET /payment/auth HTTP/1.1
Host: 172.20.10.2:81
Authorization: Bearer O_hf9uY3jNIiEpvmnmz3F9_dM_Q4QfK2h6HWCu4cMF8dUbWfEfEEycQvpT30K1Y10ajHFRVC9
    csTiYY4Fxb71EKeIdkhzoXZ3y1QN
    -Oxwr2luK6JKkIdM645xXCX2vIrxXA_FhCuZVappdV8vkewt140kvLJiAQQrgFM8hCPCQMyTjiqKPj0a9Mc9HFHf57I
    6tpUWQ4WqqDCqU0j3oyC-_QrH57hApVZgT7t74MZ05W3TcmVVB3cm4G6AZlbS0pFATuylk_c8MKA3
    -BbN095D6B_M45GmmgaOJwFjlMds9HiaJFlqwQHDMAqyTUox3IME8JwPW9r7hT1tncNh6iJwdoWXj8
cache-control: no-cache
Postman-Token: 166d0e58-5129-40ca-9b3e-8718b1337a0a
```

Şekil 4.8: Postman sorgu örneği (Ödeme token bilgisi alma ve ödeme yapma)

```
POST /payment/cart/pay HTTP/1.1
Host: 172.20.10.2:81
Content-Type: application/json
Authorization: Bearer mIxd-iI-TJIl0V9VIauxzHDG3kme
    -DMfnuyH2zrR8HG2q_k6Um5_Y2TwAg3piCn1RPsSLw_s2x8V8ezJfvyCFGH_cXSIXBb5YG90TgawyiOPocc6s67GN6
    c6Xl2NmkikOPhUMdtl_36q6kWA9EFLfBuZkGlJYV0VowixWt7WjwUCc0Jb0Zd9pZeW9BsNoGmvXVC5MjNdykb5r1vt
    ocLy7eZ9KqUFaNoSzGEGOidHF0GxATUz0ttX1BM6O5mPkPAomBmgdj
    -c7Mz_KrxFjPW8AKxlL4Ss0hKzJ5vMccbLANhOTnEfGHyt5ZHaD41Wz2vmDpBJQQ
cache-control: no-cache
Postman-Token: 2935d6a5-d23f-4384-8272-54313ff290fa

{
    "cartId": "562576b8-ed95-4000-8fa3-8330fe42fe2f"
}
```

Şekil 4.9: Bu aşamadaki token sadece ödeme işlemini gerçekleştirmek ya da iptal etmek için geçerli olan token

4.8 İADE KISMI

Aldığı ürünlerin tamamını veya bir kısmını iade etmek isteyen bir kullanıcı, ürünleri aldığı mağazaya gidip iade işlemi için görevli personeli bulması gerekmektedir. Personel müşteriden mobil uygulaması üzerinden işlem geçmişi kısmından hangi alışverişi üzerinde iade işlemi yapmak istediğini sorar ve bu işlemi uygulama üzerinden açıp işlemin QR kodunu açmasını ister.

Daha sonra personel açılan QR kodu üye işyeri için yazılmış olan mobil uygulamasını açarak okutur ve kullanıcının alışveriş detaylarını görüp burada hangi ürünlerin iade edileceğini müşteriye sorar. Müşterinin belirttiği ürünleri seçerek "İade et" butonuna basarak müşteriye para iadesi sağlanmış olur. Aynı zamanda mağaza/marketten alınan komisyonda firmaya iade edilir. HttpPost tipinde bir metodur.

```
POST /payment/refund HTTP/1.1
Host: 172.20.10.2:81
Content-Type: application/json
Authorization: Bearer O_hf9uY3jNIiEpvmnmz3F9_dM_Q4QfK2h6HWCu4cMF8dUbWfEfEEycQvpT30K1Y10ajHFRVC
    9csTiYY4Fxb71EKeIdkhzoXZ3y1QN
    -Oxwr2luK6JKkIdM645xXCX2vIrxXA_FhCuZVappdV8vkewt140kvLJiAQQrgFM8hCPCQMyTjiqKPj0a9Mc9HFHf57
    I6tpUWQ4WqqDCqU0j3oyC-_QrH57hApVZgT7t74MZ05W3TcmVVB3cm4G6AZlbS0pFATuylk_c8MKA3
    -BbN095D6B_M45GmmgaOJwFjlMds9HiaJFlqwQHDMAqyTUox3IME8JwPW9r7hT1tncNh6iJwdoWXj8
cache-control: no-cache
Postman-Token: 38a681fd-b247-47d9-9ace-531ef8812e39

{
    "cartId" : "12342",
    "cartItemsId": ["11","12","15"]
}
```

Şekil 4.10: Postman sorgu örneği

5. GÜVENLİK

Sistemin güvenliği JWT token kullanılarak sağlanmaktadır. Açılımı “Json Web Tokens” olan bu yöntem. sunucu ile istemci arasında veri alışverişi ve doğrulamayı sağlar. Genellikle başlık (header), yük (payload) ve imza (signature) kısımlarından oluşmaktadır. Başlık kısmında oluşturulacak olan imzanın tipi belirtilir. [13]

```
header = '{"alg":"HS256","typ":"JWT"}'
```

Şekil 5.1: JWT token başlığı [13]

Burada imza için kullanılan imzanın HMAC-SHA256 ile şifrelendiğini ve JWT token tipini belirtir. Yük kısmında ise saklanmak istenen verilerin bulunduğu kısımdır. Claim kısmıda denmektedir. Burada “E-posta”, “Kullanıcı “id’si”, “Kullanıcı Adı”, “Şifre” veya “Kullanıcı rolleri” gibi bilgiler tutulur. Buradaki asıl amaç sunucu tarafında doğrulama yapılması ve doğrulanmış isteklerin ihtiyacı olan metod parametrelerini içerebilir.

```
payload = '{"userRole":"admin","userId":13134422, "expire": "6/16/2019 2:38:51 PM" }'
```

Şekil 5.2: Payload içeriği [13]

Expire yani token’ın geçerlilik tarihi girilmesi şiddetle önerilmektedir çünkü her bir kullanıcı giriş yaptıktan sonra token alır ve bu token’ların geçerlilik süresi olmazsa aldıkları bu token’lar ile sürekli giriş yapabilirler. Bu yüzden geçerlilik tarihi girilmesi gereklidir.

İmza kısmında ise oluşturulmuş olan başlık ve payload kısımları ayrı ayrı base64 ile şifrelenir ve aralarında nokta(.) konularak HMAC-SHA256 ile şifrelenir. Şifreleme gizli bir anahtar ile yapılır, bu anahtar bizim kullandığımız sistemde machine key diye geçen makine anahtarı kullanılmaktadır. Bu yüzden A sunucusunda şifrelenmiş olan bir token sadece A sunucusunda çözümlenebilir. Makine anahtarı projemizin Web.config dosyası altında ayarlanabilmektedir.

key = 'machineKey'

unsignedToken = encodeBase64Url(header) + '.' + encodeBase64Url(payload)

signature = HMAC-SHA256(key, unsignedToken)

Şekil 5.3: JWT token yapısı [13]

yJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsb2dnZWRJbkFzIjoiYWRtaW4iLCJpYXQiOjE0MjI3Nzk2Mzh9.gzSraSYS8EXBxLN_oWnFSRgCzcmJmMjLiuyu5CSpyHI

Şekil 5.4: JWT token örneği [13]

```
<system.web>
<compilation debug="true" targetFramework="4.6.1" />
<httpRuntime targetFramework="4.6.1" />
<machineKey
validationKey="6C9DD960A14AC0DE722049FEDED1ECF71C97AAAB0F5CA63D466B946DE547413B1BF86EEC817D9E19F3D0C326E7158F44CF2997193E2970919D60D98EE2598127"
decryptionKey="9F7C724D8E75A7D522408397F50EB250BFF41CB955C817C6FF676DD44E06FFDE" validation="SHA1" decryption="AES" />
</system.web>
```

Şekil 5.5: Örnek bir Web.config makine anahtarı [13]

Makine anahtarı zorunlu bir alan değildir, Web.config dosyasında belirtilmezse asp.net otomatik olarak makinenin kendi anahtarını kullanmaktadır fakat birden fazla sunucu ile çalışıyorsak bunu sabit şekilde her bir sunucunun ortak kullanması için bu ayar yapılmalıdır. Aksi takdirde yukarıda belirtildiği gibi A sunucunda oluşturulan bir token B sunucusunda çözümlenemez. Dolayısıyla kullanıcı sisteme giriş yapmış olduğu halde 401 hata kodlu bir hata alır bunun anlamı "Giriş yetkiniz yok. Sisteme giriş yapmanız gerekmektedir."

Girişte oluşturulan token'ın kullanımı ise istemci(web-mobil) tarafından yapılan isteğin(request) başlık(header) kısmına "Authorization" başlığı ile eklenmelidir.

Authorization: Bearer yJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC...

Şekil 5.6: HTTP isteğinde token kullanılışı

Bu yöntemin avantajları olduğu gibi dezavantajları da bulunmaktadır.

Avantajları:

- İçerisinde veriler tutabildiği için sürekli veri tabanına sorgu atmak gerekmez.
- Çerezler kullanılmadan kimlik doğrulama yapılabilmektedir.
- Doğrulama için veri tabanına istek yapılmadığı için performans açısından hızlıdır.

Dezavantajları:

- Yük (payload) kısmına ne kadar çok veri koyulursa oluşturulan token boyutu o kadar büyümektedir. Dolayısı ile token sistemi yanlış kullanılırsa ve içerisine gereğinden fazla veri atılırsa boyutunun büyüklüğünden dolayı performans düşüşü oluşabilir.
- Token durumsuz bir yapı olduğu için kullanıcı yeni bir giriş işlemi yapıp yeni token aldığında eski token geçerlilik süresi bitene kadar kullanılabilir olacaktır.
- İmza kısmında kullanılan gizli anahtarın güçlü ve bulunabilir olması zor değilse token dışarıdan saldırgan (hacker) tarafından brute force ile çözümlenebilir.

6. SONUÇ

Mobil jet ödeme sistemi ile günümüzde kullanılan ödeme kanalları karşılaştırdığımızda ciddi anlamda avantaja sahip olduğu gözükmektedir. Bu hem kullanılan alt yapı ile hem de cihaz tasarımı ile gerçek olmaktadır. Sistem web api ile haberleştiği için hem güncellemeler kolay bir şekilde yapılmaktadır hem de yenilikçi özelliklere açık hale gelmiştir. Bunlardan birisi iade işlemi için fiş saklama zorunluluğunun ortadan kalması gibi sadece uygulama üzerinden işlem geçmişine bakıp hangi alışveriş iade edilmek isteniyorsa mağaza görevlisi işlemin QR kodunu okutup anında iade sağlayabilmektedir. Bunun yanı sıra online platform olup anlık müşteri bazlı indirim ve kampanyalar düzenlenebilme, cihazları daha portatif hale getirip daha farklı ürünler ortaya koyabilme gibi çok fazla yenilikçi ürün ve özellikler ortaya konabilir. Sonuç olarak çok daha az maliyetli, daha güvenli, daha hızlı, daha kolay ve çok fazla yeniliği barındırabilecek bir sistem olarak mevcut sistemlere alternatif olarak sunulmuştur

KAYNAKÇA

[1] http://perakendegazetesi.com/2015/01/21/adesede-hizli-kasa-donemi-basladi/, 12 Mayıs 2019

[2] http://ankadrs.com/tr/kasa-onu-gondolu/, 12 Mayıs 2019

[3] https://www.kocaeligazetesi.com.tr/haber/1429741/bilin-bakalim-bu-ne-kuyrugu#, 5 Mayıs 2019

[4] https://www.youtube.com/watch?v=kszesgtvHK4, 5 Mayıs 2019

[5] https://trends.google.com.tr/trends/explore?q=ADO.NET,Entity%20Framework, 13 Mayıs 2019

[6] https://www.quora.com/Which-is-better-ado-net-or-entity-framework, 20 Mayıs 2019

[7] https://exceptionnotfound.net/dapper-vs-entity-framework-vs-ado-net-performance-benchmarking/ ,1 Haziran 2019

[8] https://entityframework.net/code-first-vs-database-first, 6 Haziran 2019

[9] https://www.entityframeworktutorial.net/choosing-development-approach-with-entity-framework.aspx, 14 Mayıs 2019

[10] Code-First Development with Entity Framework, Sergey Barskiy, Sayfa 135

[11] https://www.entityframeworktutorial.net/code-first/simple-code-first-example.aspx, 13 Mayıs 2019

[12] https://www.entityframeworktutorial.net/efcore/entity-framework-core-migration.aspx, 13 Mayıs 2019

[13] Pro ASP.NET Web API Security, Badrinarayanan Lakshimiraghavan, Sayfa 98