# Cart Managment(Çiçek Sepeti Projesi)

## 1. Ne Kullandım ?

- ✓ C#
- ✓ .Net Core Api
- ✓ Redis
- ✓ Docker
- ✓ Unit Test (XUnit)
- ✓ Swagger
- ✓ Postman
- ✓ Dependency Injection
- ✓ FluentValidation
- ✓ Automapper
- ✓ .Net Core IOC Container
- ✓ Dockerfile
- ✓ Docker-compose.yml
- ✓ Mock Library
- ✓ Katmanlı Mimari
- ✓ Visual Studio Code
- ✓ Dotnet cli

## 2. Nasıl Kullandım?

- Proje Katmanlarını Oluşturma

I. Projelerimi dotnet cli ile oluşturdum ve Visual Studio Code ile geliştirmelerimi yaptım
II. Bütün projelerde ortak kullanılabilecek katman olan Core Katmanını oluşturdum.
III. Entitlerimi eklediğim Entities Katmanını oluşturdum.
IV. Veritabanı ile iletişimi sağlayacak olan DataAccess Katmanını oluşturdum.
V. Mantıksal işlemlerinin yapılacağı katman olan Business Katmanını oluşturdum.
VI. Son olarak Api Katmanı için .Net Core Web Api projesi oluşturdum.

- Projeyi Test Etme ve Biraz Kodlama

  Bütün katmanları oluşturduktan sonra postman kullanarak web api yi test ettim. Api'nin düzgün bir şekilde çalıştığına emin olduktan sonra sıra kodlamaya geldi. Bunun için projemde kullanabileceğim entitilerimi Entities Katmanında oluşturdum. Daha sonra cache için ve ilerde farklı cache mekanizmalarını kullanabileceğimizi düşünerek Core Katmanına generic bir interface (ICacheManager adında) oluşturdum.

- Redis Entegrasyonu

  Generic interface hazır olduğuna göre artık redis veritabanını entegre edebilirim. Bunun için docker üzerinde redisi ayağa kaldırdım. Stackoverflow yazılımcılarının geliştirmiş olduğu StackExchange.Redis kütüphanesinin yardımı ile Redisi kütüphaneme dahil ettim. ICacheManager interfacesini implemente eden ve redis üzerinde işlem yapabileceğim RedisCache somut sınıfını oluşturdum.

- Business Katmanının Kodlanması

  Rediste projeme dahil olunca artık sepet işlemlerini yapabilirim. Bunun için business katmanında CartService adında bir sınıf içerisinde daha önce yazmış olduğum ICacheManager sınıfını constructordan enjecte ettim ve dependency injection için default gelen .Net Core IOC containerı kullandım. Artık sepete ürün eklerken yapılacak kontrol için ortam hazır durumda.

- Automapper ve FluentValidation kütüphanelerini Dahil Etme

  UI'dan gelen (DTO diyorum) modelleri istediğim modele dönüştürmek (Enitity) için Automapper kütüphanesini dahil ettim. Daha sonra bu modellerin validasyonu için FluentValidation kütüphanesini kullandım. Böylece sepete ürün eklerken model validasyonunu ve modeli istediğimiz entitiye dönüştürmüş olduk.

- Api Kodlama

  Dışarıya açılacak kapımız olan .Net Core Api ' ya ICartService interfaceni ekledim. Bu interface ile soyutladıktan sonra api end pointlere gerekli metotları çağırarak sepete ekleme işlemlerini baştan sona yapabilir duruma geldim. Api hata yakalama için bir tane middleware yazdım. Bu middleware ile back-end de oluşan hataları istediğim gibi manipüle etme imkanım oldu.

- Projeyi Dockerize etme

  Uçtan uca projeyi çalışabilir hale getirdikten sonra projeyi dockera yüklemek için Api projeme Dockerfile dosyasını ekledim ve projemi dockera yüklemiş oldum. Daha sonra projeyi ve redisi tek komut ile çalıştırmak için docker-compose.yml dosyasına gerekli konfigürasyonları yaparak tek komut ile projenin çalışır duruma gelmesini sağladım.

- Unit Test Yazma

  Projenin business kısımları için gerekli gördüğüm Unit testlerimi yazdım. Unit testleri yazarken XUnit kütüphanesini tercih ettim. Burda bağımlılakları yönetmek ve sahte datalar için "Moq" kütüphanesinden ve dependency injectionun nimetlerinden faydalandım.

- Swagger Entegrasyonu

  Yazdığım api end-pointlerini dökümante etmek ve testlerinin kolayca yapılmasını sağlamak için projeme swaggerı dahil ettim. Böylece tek bir url ile bütün api end pointlerini tek seferde görüp test etme imkanım oldu.

- Versiyon Yönetimi

  Projeyi git altyapısını kullanan githuba yükledim.Projeye https://github.com/ahmetunge/dotnet-core-redis-cart-managment adresinden ulaşabilirsiniz.

# 3. Uygulamayı Nasıl Çalıştırırsınız?

## 1) Docker ile projeyi çalıştırmak

- Uygulamayı docker üzerinde çalıştırmak isterseniz öncelikle bilgisayarda docker kurulu olması gerekiyor.
- Docker çalışıyorsa proje içerisine gelip “cmd” ekranında docker-compose up komutunu çalıştırmanız yeterli olacaktır.
- Bu uygulamayı 1907 portundan, redisi 6379 portundan ayağa kaldıracaktır.
- RunApi(http://localhost:1907/swagger);

## 2) Docker olamadan projeyi çalıştırmak

- Eğer docker olamadan projeyi çalıştırmak isterseniz. Redisi lokal ortama kurmalı ve appsettings.Development.json içerisinde portu doğru bir şekilde vermelisiniz.
- Src içerisindeki Api dosya yoluna gelip dotnet run çalıştırmanız yeterli olacaktır.
- Bu yapılanlar projeyi 5000 portunda çalıştıracaktır.
- RunApi(http://localhost:5000/swagger)
- **NOT: Bu yazdıklarımı bildiğinize eminim projeyi hızlı çalıştırabilmek için yukarıdakileri not düştüm. Lütfen beni yanlış anlamayın.**

```
public string RunApi(string url ) {
    return  “Uygulama çalıştıktan sonra url ile api ve end-pointlerine ulaşabilir ve testlerini yapabilirsiniz. Alternatif olarak Postman üzerinden de testler yapılabilir. Bunun için src içerisinde paylaşmış olduğum postman dosyalarını import etmeniz yeterli olacaktır.”;

}
```

# 4. Faydalandığım Kaynaklar ve Kişiler

- https://stackoverflow.com/
- https://www.nuget.org/
- https://www.udemy.com/
- https://github.com/
- https://automapper.org/
- https://fluentvalidation.net/
- https://www.docker.com/
- https://redis.io/
- Neil Cummings
- Maximilian Schwarzmüller
- Geçmişte yaptığım projeler (Ahmet Ünge)
- Engin Demiroğ