# VIPER Egzersiz Uygulaması

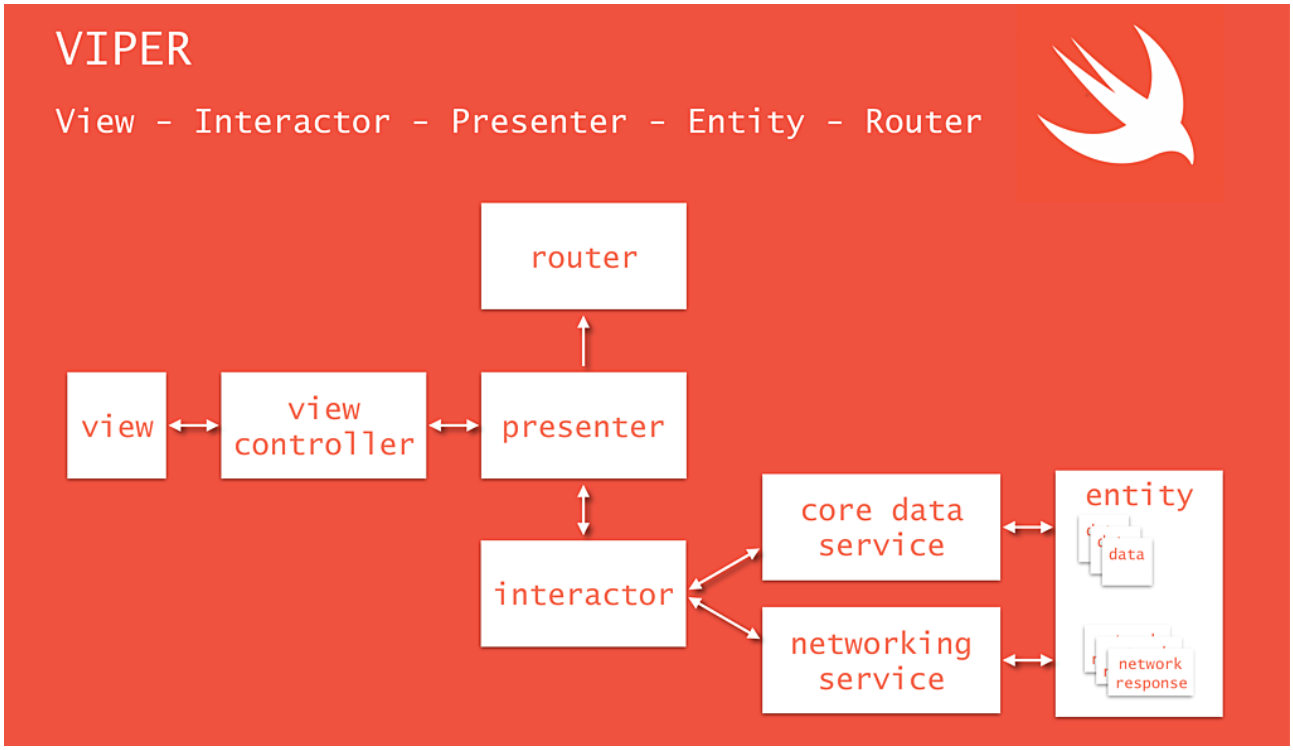
## Uygulama Hakkında

Yapılan egzersiz uygulaması, 2 parametreden oluşan bir veri modeli üzerinden kullanıcıya currency-price bilgileri gösteren bir listeleme uygulamasıdır. Uygulamada gösterilen veriler güncel olmamakla birlikte bir URL üzerinde bulunan JSON formatlı bir veri setinden alınmaktadır.

Uygulama içerisinde; İnternetten veirlerin alındığı bir Interactor sınıfı, Kullanıcıya içeriklerin sunulduğu bir View sınıfı, JSON datanın modellendiği bir Entity sınıfı, modelden alınan verilerin işlendiği MVVM tasarım örüntüsündekine benzer bir Presenter sınıfı ve tüm bu sınıfların birbirleri arasında iletişim kurmasını sağlatan Router sınıfı yer almaktadır.

VIPER Design Pattern'de ana odak noktası tüm ekranların veya kullanıcı arayüzlerinin storyboard kullanılmadan programatic UI odaklı bir şekilde parçalar ayrılarak kodlanmasıdır. VIPER tasarım örüntüsü, geliştiricilere parçalanmış ve SOLID prensiplerini daha belirgin bir şekilde uygulama imkanı sunar. Ayrıca VIPER, iOS geliştirme süreçleri için uygulanabilirliği en geçerli olan tasarım örüntüsüdür.

VIPER tasarım örüntüsü genel olarak 5 ana unsurdan oluşur ve bu unsurların birbirleri arasındaki iletişim aşağıdaki görselde olduğu gibidir.




## VIPER Tasarım Örüntüsü

VIPER tasarım örüntüsü, iOS geliştirme süreçlerinde yeni fakat oldukça sık kullanılan bir tasarım örüntüsüdür. Hepimizin sıklıkla kullandığı ve bildiği sahibinden.com sitesine ait iOS mobil uygulaması VIPER pattern kullanılarak oluşturulmuştur.

VIPER tasarım örüntüsünün kendine ait bazı zorlukları olsa bile kompleks yapılı

uygulamalarda okunabilirlik, test edilebilirlik, geliştirilebilirlik ve SOLID prensipleri için oldukça verimli bir iş çıkartmayı sağlar. Tüm iş birimleri birbirlerinden ayrıldığı için kodun anlaşılabilir olması oldukça kolaydır. Şimdi birlikte VIPER için kullanılan 5 ana unsuru detaylıca inceleyelim.

## View

View, her uygulamada olmazsa olmaz olan sınıf/bileşen olarak değerlendirebiliriz. Neticede bir mobil uygulama için kullanıcı arayüzü olmak zorunda. VIPER modelinde View tasarlarken storyboard kullanmadığımız için View'ın programatik olarak yapılandırılması gerekiyor. Bu süreç kompleks uygulamalarda zaman alabiliyor ve farklı yaklaşımlar gerektirebiliyor. En zor kısmı ise component sayısı arttıkça hata yapma olasılığının artması. İşte bu riski minimize edebilmek için VIPER modelini kullanan projelerde her bir ekran için ayrı bir klasörleme yaparak ilerlemek gerekiyor. Bu durum delegate sayısını ve klasör sayısını arttırıyor fakat doğru isimlendirme ve dosyalama yöntemleriyle oldukça düzenli bir uygulama geliştirmek mümkün.

View sınıfını kodlarken bir önceki konu başlığında yer alan akış diagramını referans alarak ilerlediğimizde sürecin oldukça kolaylaştığını görebiliyoruz. Örnek olarak yaptığımız uygulama içerisinde tableView'ın bulunduğu ana ekrandan seçilen bir hücre ile detay ekranına gitmek istediğimizde, detay ekranımızın ayrı bir dosya içerisinde tekrar VIPER modelinin uygulanması eforu biraz arttırıyor fakat yeni eklentiler yapmak istediğimizde eski kodlarımıza dokunmadan oldukça kolay eklemelerle uygulamamızı büyütebiliriz.

Unutmadan ekleyelim, View sınıf tıpkı MVVM modelinde olduğu gibi sadece View nesnelerini yönetmekle yükümlü, verilerin ne olduğu, nereden geldiği nasıl geldiği gibi konularla hiç bir işi yok. View'ın tek görevi Presenter'ı referans alarak içinde bulunan komponentleri doldurmak. Bu nedenle View'ın sürekli olarak Presenter ile iletişimde kalması gerekiyor. Bunun için View ve Presenter arasında protocol ve delegate üzerinden iletişim sağlıyoruz.

## Interactor

Interactor sınıfının temel görevi ilgili servisleri kullanarak dış ortamdan getirilecek olan verilerin alınmasını sağlamak. Yani kısaca Interactor internetten veya CoreData gibi lokal servislerden alınacak olan verinin alınmasından sorumlu.

Interactor bu veri alma işlerini yaparken Presenter ile iletişimde olmak zorunda ki Interactor'ün aldığı veriler Presenter üzerinde gerekli model kullanılarak işlenebilsin ve istediğimiz verileri istediğimiz şekilde kullanıcımıza gösterebilelim.

## Presenter

Presenter sınıfını tpkı MVVM modelinde olduğu gibi bir ViewModel sınıfı olarak düşünebiliriz. Presenter, Interactor tarafından aktarılan datayı, Entity sınıfı içinde tanımlı olan model ile cast etmek ve bu elde edilen anlamlandırılmış veriyi (örneğin JSON datası için decode edilmiş JSON verisini) Router aracılığı ile View'ın kendini güncellemesini sağlamak ile yükümlü.

Bu tanımlama biraz karmaşık geldiyse örnek uygulamamız üzerinden inceleyelim; uygulamamızda bulunan Interactor verdiğimiz GitHub linkinde bulunan 2241 adet veri modelinden oluşan bir JSON veri kümesini URLSession kullanarak indirmeyi deniyor. İndirme işleminin başarılı olması durumunda JSON verisini Entity içerisinde tanımlanmış olan Model struct'ına referans alan bir dizi ile cast ediyor ve ardından Presenter sınıfımız içerisinde tanımlı olan interactorDidDownloadData protokol fonksiyonunu tetikliyor. Bu protokol fonksiyonu ise View protokolüne ait olan update fonksiyonunu tetikliyor ve View'ımız güncellenmiş oluyor.

## Entity

Yine tüm tasarım örüntülerinden tanıdığımız ama ismi farklı bir sınıf. Entity, VIPER modelinde kimse ile iletişim içerisinde değildir. Sadece bir referans tanımlar. Entity, tıpkı MVC ve MVVM modellerinde olan Model olarak bilinen olmazsa olmaz parçalarımızdan birisidir ve tıpkı diğer örüntülerde olduğu gibi JSON datasını tanımlamak için kullanılır. Interactor üzerinden JSON verisinin decode edilmesi için önemli bir bileşendir.

## Router

İşin en albenili kısmı Router… Router'ın görevi yukarıda tanımladığımız View, Interactor, Presenter, Entity gibi ana bileşenlerin hepsinin birbirleri arasında iletişim kurmasını sağlayan ve bu "orkestrayı" yöneten şef olarak düşünebiliriz. Router'ın temel görevi bu sınıfları tanımlarken kullandığımız protokoller içerisinde belirtilen Any ibarelerini tanımlamak, anlamlandırmak, kimin içinde kimin ne olduğunu ifade etmek gibi görevlerdir. AppDelegate'in bir alt birimi gibi düşünmek ne kadar doğru bilmiyorum ama görev tanımları birbirlerine benziyor.

Kimin içinde kimin ne olduğunu ifade etmek cümlesini örnek uygulamamızdan referansla tanımlayacak olursak startExecution fonksiyonumuzu ele aldığımızda göreceğiz ki view'ın içerisinde tanımladığımız presenter'ın Presenter sınıfı olduğunu tanımlıyor. Yani bütün sınıflarımızı birbirlerine bağlayarak tanımlandırıyoruz.

# Kapanış

VIPER tasarım örüntüsü, örnek uygulamamız gibi küçük çaplı uygulamalarda işi biraz zorlaştırabiliyor. Ayrıca, büyük projelerde kod kalitesi anlamında bir çok avantajı olmasına karşın takımın tümünün VIPER modelini bilmesi, yaptıkları işleri bu modele oturtarak kullanmak zorunda olması ve çok fazla klasörleme yapılması bol miktarda delegate kullanılması nedeniyle bazen doğru kişilerin bir araya getirilmesini zorlaştırmaktadır.

Ek olarak, VIPER modelinde storyboard kullanımı olmadığı için bu model ile bir proje geliştirmeden önce xCode IDE'si tarafından projelerimizde otomatik eklenen SceneDelegate, AppDelegate gibi dosyaların ne işe yaradığı, özelliklerinin neler olduğu, bu dosyaları nasıl manipüle edeceğimizi de iyi biliyor olmamız gerekiyor. Elbette ki SceneDelegate ve AppDelegate dosyalarını manipüle edebilmek için bir iOS uygulamasının yaşam döngüsünü iyice kavramış olmakta şart.