# NOTLAR

* 2 tip Authentication yaklaşımı vardır.
* Bunlardan biri Cookie Based yaklaşım. Eskiden giriş yaptıktan sonra session’ da giriş bilgisi tutuluyordu. Ve o şekilde bir authorization altyapısı kurmaya çalışılıyordu. Şimdi günümüzde standartlar değişti. Bizim traditional olarak gördüğümüz şeyler Cookie Base Authenticationlar.
* Daha modern olan diğer yaklaşım ise Json Web Token, token bazlı bir giriş sisteminden bahsediliyor.
* Normal şartlarda Cookie’ nin mantığı bizim ilgili kullanıcımız giriyordu username ve passwordu ile biz onun sistemde olup olmadığını kontrol ediyorduk. Eğer bu kullanıcı sistemde varsa bir cookie oluşturup o cookie ile ilgili kullanıcının bilgisini tutuyorduk.
* JWT’ da ise biz bir endpoint oluşturduğumuzda mesela bir user bilgisini check eden bir endpointimiz var ve bu endpointe biz yine bir username ve password ile gidiyoruz. Gittiğimiz zaman bu sefer bize ilgili endpoint bir token, yani bir bilet veriyor. Ve bu bilet de aslında senin bilgilerin doğruysa al sen bu bileti ilgili sinema salonuna veya filmlere girebilirsin şeklinde bir sonuç veriyor. Yani burdaki doğrulama şemamız tamamıyla token bazlı oluyor. Artık o kişide o token varsa o tokenın bitiş süresine kadar yetkili olduğu kısımlarda tüketebiliyor. Burdaki mantık; ben gidiyorum bir endpointe, bir username ve bir password ile sonrasında da ilgili endpoint bana bir token üretiyor ve token çıktısını veriyor.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Bu token çıktısı da görselde görüldüğü üzere Encoded bölümüne benziyor.
* Bu token 3 parçadan oluşuyor.
* İlki yani kırmızı olan bölüm Header,
* İkincisi yani mor olan bölüm Payload,
* Üçüncüsü ve mavi olan bölüm ise Signature. (İmza kısmı).
* Token’ ın bağlı olduğu birtakım standartlar var buna da RFC 7519 deniyor.
* Görselin Decoded bölümünde Header kısmında bir algoritma ve tipten bahsetmiş. Tip token tipi, alg yani algoritma ise aslında bu tokenın şifrelendiği algoritma türü. Orda yazılan HS256, HMACSHA256’ nın kısaltmasıdır.
* HMACSHA256, bir şifreleme algoritmasıdır. Ve RFC standartlarına uygun ve tavsiye edilen bir şifrelemedir.
* Görselin Decoded bölümünde Payload kısmı userın, datası veya rol bilgilerinin tutulduğu kısımdır.
* Son bölüm olan signature kısmı ise imza kısmıdır.
* Json Web Token’ ı .NET CORE tarafında üretmek oldukça basit, bunun için bir nesne kullanıyoruz. Bu nesnenin adı JwtSecurityTokenHandler. Bu nesnenin içerisindeli WriteToken metoduna başka bir nesne aracılığıyla belirtmiş olduğumuz tokenı içerisine parametre olarak verip üretebiliyoruz.
* Bunun içerisinde bazı parametreler var. Bunlardan bazıları,
  + İssuer: İlgili tokenı oluşturan,
  + Audience: Bu tokenı kullanacak olanlar,
  + Expires: Tokenın geçerlilik süresi,
  + notBefore: Ne zamandan itibaren yani herhangi iki tarih arasında token geçerli olacak,
  + Claims: İlgili userın veya başka dataları tutabiliyoruz,
  + signinCredentials: İlgili tokenın imzasını tuttuğumuz kısım.
* Json Web Token’ ı kullanabilmemiz için Startup.cs sınıfına configure etmemiz gerekmektedir.
* 3 tip şifreleme şekli var. Simetrik, Asimetrik, Hash.
* Hash: Siz bir datayı bir kere şifrelediğiniz zaman o datayı artık kaybedersiniz yani tekrar çözmeniz mümkün değildir.
* Simetrik: Bir key vasıtasıyla (şifreleyen de şifreyi çözen de aynı keye sahip olmalı) bu şifrelemeyi çözebilirsiniz.
* Asimetrik: İşler biraz daha kompleks şifreyi çözen de şifreleyen de farklı keyler kullanıyor.
* CQRS, Amaçladığı şey command ile query’ i sorumluluklarına göre birbirinden ayırt etmektir.
* Meslea bir veri tabanınız var ve bir servisiniz var ve bu service’ e UI’ dan bir istek geldiğini düşünün. Eğer bu istek bir veriyi getirme isteğiyse bunun adı query, eğer bu istek bir veriyi ekleme, güncelleme veya silme isteğiyse bunun adı command olmaktadır.
* Düşünün ki benim bir requestim geliyor, arada bir sınıf var veya bir yapı var gelen requestim bu yapıya giriyor ve bu yapı diyor ki: “Bizim burada handlerlarımız var”. Bunu şöyle düşünebilirsiniz bunlardan birisi query, birisi command, birisi başka bir query handler, bu requeste bakarak karar veriyor ve diyor ki sen git şuradaki query handler’ ı çalıştır.Örneğin bize Update Student Command şeklinde bir command geliyor. Bu command geldikten sonra buradaki arabulucu şöyle bir karar veriyor; gelip diyor ki Update Student Command geldiyse sen git Update Student Command Handler’ ı çalıştır. İşte tam olarak buna arabulucu anlamındaki Mediator Pattern diyoruz.
* İşin gerçeği Mediator Pattern’ ın halihazırda yazılmış olan bir kütüphanesi var onu direkt .Net Core içerisine ekleyip kullanabiliriz. Ancak Mediator Pattern oradaki o kütüphanede kullanacağımız interfacelerle gerçekleşen bir şey, yani tek yolu o kütüphane gibi düşünmeyin. Bunu kolaylıkla da tasarlayabiliriz. O patternı doğrudan manuel olarak da uygulayabiliriz.