REST API GÜVENLİĞİ

Yunus Santur

RESTful Web Servisleri ve Güvenlik Neden Önemlidir?

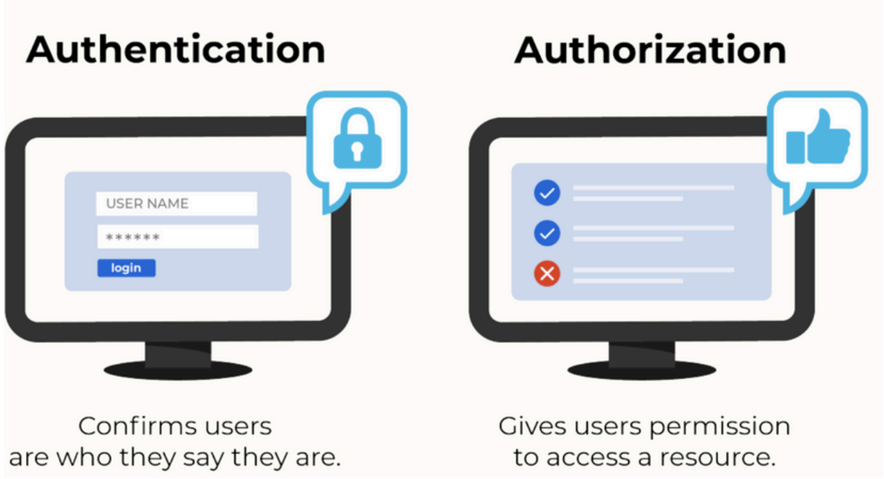
Veri Güvenliği ve Gizliliği: Web servisleri, kullanıcı ve sistem verilerini taşır. Güvenlik önlemleri olmadan veriler yetkisiz kişilerin eline geçebilir, bu da veri sızıntısı ve kişisel bilgilerin ifşasına yol açar.

Yetkilendirme ve Kimlik Doğrulama: RESTful servisler, kullanıcıların hangi kaynaklara erişebileceğini belirlemelidir. JWT, OAuth2 gibi mekanizmalarla kimlik doğrulama yapılmazsa, saldırganlar sisteme yetkisiz erişim sağlayabilir.

Servislerin Sürekliliği ve Performansının Korunması: Güvenlik açıkları, servisleri hedef alan DoS/DDoS saldırılarına yol açabilir. Güvenli tasarım, servislerin kesintisiz çalışmasını sağlar.

REST’te Authentication & Authorization

Günümüzde web uygulamaları, mobil uygulamalar ve IoT cihazları, veri alışverişi ve işlem yönetimi için sıkça RESTful Web Servisleri kullanıyor. Bu servisler, farklı sistemler arasında standart bir iletişim sağlar, ancak veri ve sistem güvenliğini sağlamak kritik bir öneme sahiptir. Çünkü kötü niyetli kişiler, güvenlik açıklarından faydalanarak hassas bilgilere erişebilir veya sistem üzerinde yetkisiz işlemler gerçekleştirebilir. Bu nedenle RESTful servislerde kimlik doğrulama (**Authentication**) ve yetkilendirme (**Authorization**) kavramlarını doğru şekilde uygulamak, hem kullanıcı verilerini korumak hem de sistemin güvenliğini sağlamak için gereklidir.



Kimlik Doğrulama (Authentication)

Kimlik Doğrulama (Authentication), bir kullanıcının gerçekten iddia ettiği kişi olup olmadığını kontrol etme sürecidir. Kısaca, sistem “Bu kişi gerçekten o mu?” sorusuna yanıt arar.

Örnekler:

- Bir web sitesine giriş yaparken kullanıcı adı ve şifre girersiniz. Sistem bu bilgileri veritabanıyla karşılaştırır: Doğruysa erişim izni verir, yanlışsa engeller.
- Cep telefonunuzu parmak izi veya yüz tanıma ile açarsınız. Cihaz, biyometrik verilerinizi önceden kayıtlı olanlarla eşleştirir ve yalnızca sahibine kilidi açar.

Yetkilendirme (Authorization)

Yetkilendirme (Authorization), kimliğiniz doğrulandıktan sonra “Bu kişi hangi işlemleri yapabilir veya hangi verilere erişebilir?” sorusuna yanıt verir. Kısaca, sistem kullanıcıya hangi hakları vereceğine karar verir.

Örnekler:

- Bir sosyal medya sitesinde kullanıcı olarak kendi profilinizi düzenleyebilirsiniz, ancak başka birinin profilinemüdahale edemezsiniz. Yönetici (admin) rolündekiler ise tüm kullanıcı verilerini görüntüleyip değiştirebilir.
- Ofis binasında giriş kartınız yalnızca kendi katınıza erişim izni veriyorsa, güvenlik sistemi kartınızın hangi alanlara yetkili olduğunu kontrol eder.

Authentication vs Authorization

Özellik	Authentication (Kimlik Doğrulama)	Authorization (Yetkilendirme)
1. Temel Amaç	Kullanıcının gerçekten **kim olduğunu doğrular.**	Kimliği doğrulanan kullanıcının **ne yapabileceğini veya hangi kaynaklara erişebileceğini belirler.**
2. Ana Soru	"**Bu kişi gerçekten o mu?**"	"**Bu kişi ne yapabilir veya hangi verilere erişebilir?**"
3. Zamanlama	Sisteme **giriş (login) sırasında**gerçekleşir.	Girişten sonra, **erişim kontrolü sırasında**uygulanır.
4. Örnekler	Kullanıcı adı ve şifre, parmak izi, yüz tanıma, OTP (Tek Kullanımlık Şifre).	Sadece kendi dosyalarını görebilme, admin yetkisi ile tüm verilere erişim, okuma/yazma/silme izinleri.
5. Sonuç	**Kim olduğunuzu doğrular.**	Kim olduğunuz belirlendikten sonra **haklarınızı yönetir.**

Kimlik Doğrulama (Authentication) Yöntemleri

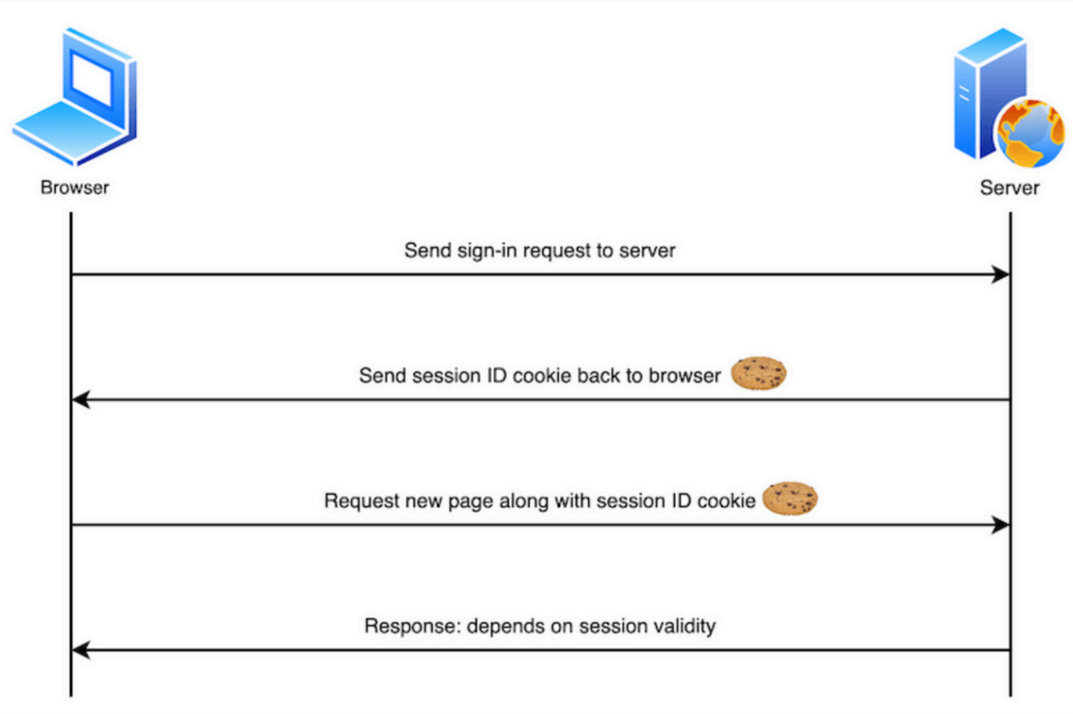
Oturum Tabanlı Kimlik Doğrulama

Oturum tabanlı kimlik doğrulama, web uygulamalarında kullanıcının giriş yaptıktan sonra kimliğinin sunucu tarafından hatırlanmasını sağlayan bir yöntemdir. Burada iki temel kavram vardır: **Session (Oturum) ve Cookie (Çerez).**

Session ve Cookie Nedir?

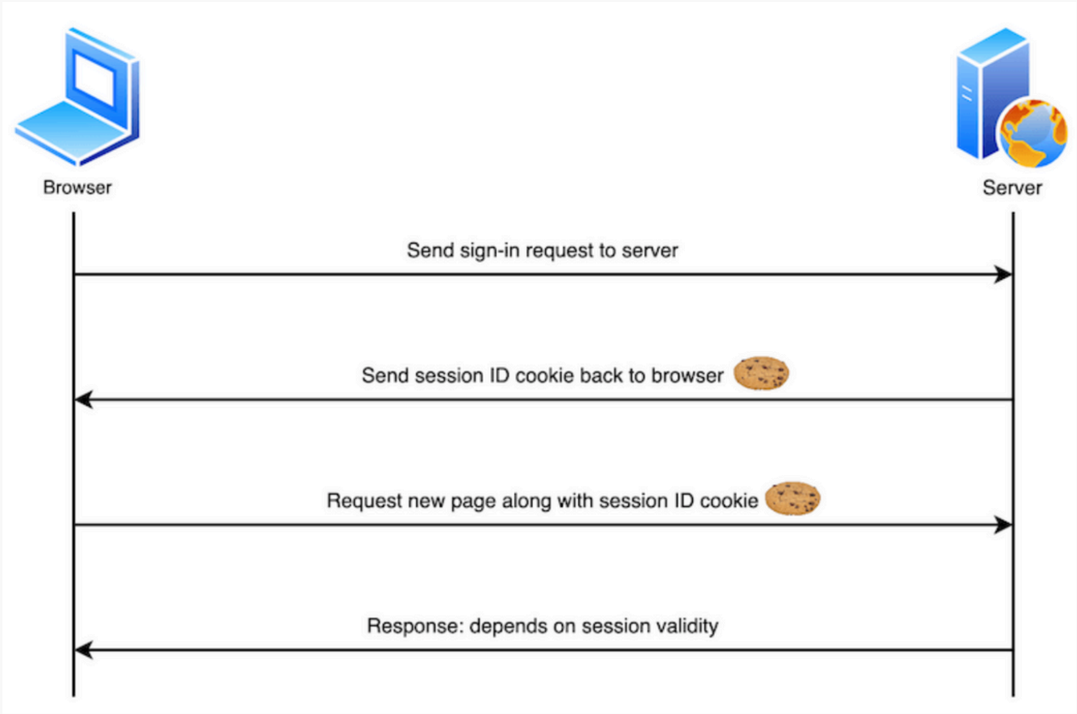
- Session (Oturum): Kullanıcının giriş bilgilerini ve oturum durumunu sunucu tarafında saklayan bir veri yapısıdır. Örneğin kullanıcı giriş yaptığında sunucu, kullanıcıya özel bir oturum ID (Session ID) oluşturur ve bunu saklar. Bu sayede kullanıcı site içinde gezindiğinde tekrar tekrar kimlik doğrulaması yapmak gerekmez.

- Cookie (Çerez): Bu, kullanıcının tarayıcısında saklanan küçük veri parçalarıdır. Session ID genellikle bir cookie aracılığıyla kullanıcının tarayıcısına gönderilir. Böylece kullanıcı her sayfa yenilemesinde veya yeni bir istekte bulunduğunda tarayıcı, session ID'yi sunucuya gönderir ve sunucu kullanıcının kim olduğunu doğrular.



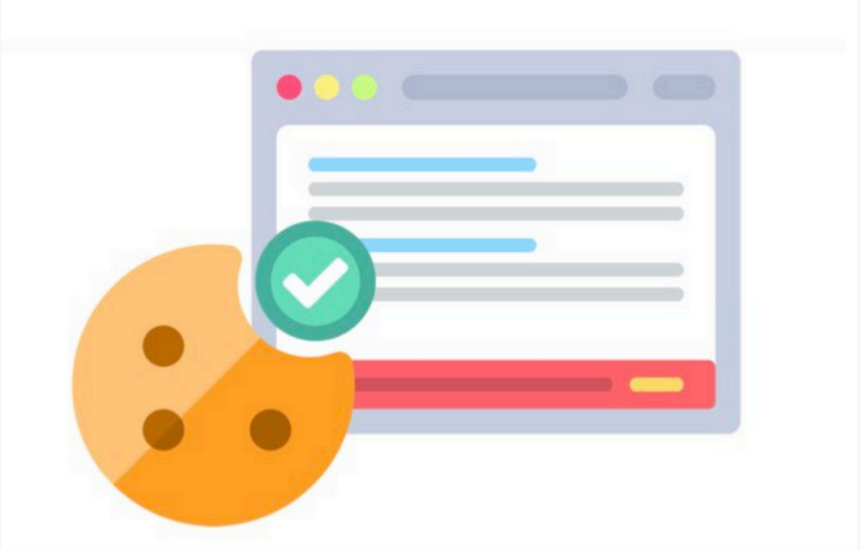
Session + Cookie : Çalışma Süreci

1. Kullanıcı web sitesine giriş yapmak için kullanıcı adı ve şifreyi girer.
2. Sunucu bilgileri doğrular ve bir Session ID oluşturur.
3. Bu Session ID, cookie olarak kullanıcı tarayıcısına gönderilir.
4. Kullanıcı site içinde gezinirken tarayıcı, bu cookie'yi sunucuya gönderir.
5. Sunucu, gelen Session ID'ye bakarak kullanıcının kimliğini doğrular ve yetkilerini uygular.



Oturum Tabanlı Kimlik Doğrulama: Örnek

- Kullanıcı bir alışveriş sitesine giriş yapar.
- Giriş yaptıktan sonra siteye eklediği ürünler, sepet durumu gibi bilgiler session üzerinde tutulur.
- Kullanıcı başka bir sayfaya geçtiğinde tekrar giriş yapmak zorunda kalmaz; tarayıcıdaki cookie sayesinde session sunucuya iletilir ve kimliği doğrulanır.



Session Tabanlı Kimlik Doğrulamanın Avantajları ve Dezavantajları

Avantajları:

1. Kullanıcı site içinde gezdikçe tekrar tekrar giriş yapmak zorunda kalmaz; oturum aktif kaldığı sürece kesintisiz deneyim sağlar.
2. Kimlik bilgileri doğrudan istemci tarafında değil, sunucu tarafında saklandığı için güvenlik seviyesi yüksektir.
3. Session yapısı, kullanıcıya özel bilgiler (örneğin sepet, dil tercihi, tema ayarı vb.) tutmak için uygundur.
4. Kullanıcı çıkış yaptığında veya tarayıcı kapatıldığında session kolayca sonlandırılabilir, bu da kontrolü artırır.
5. Hassas veriler için HTTPS ve güvenli cookie ayarlarıyla birlikte kullanıldığında yüksek düzeyde koruma sağlar.

Dezavantajları:

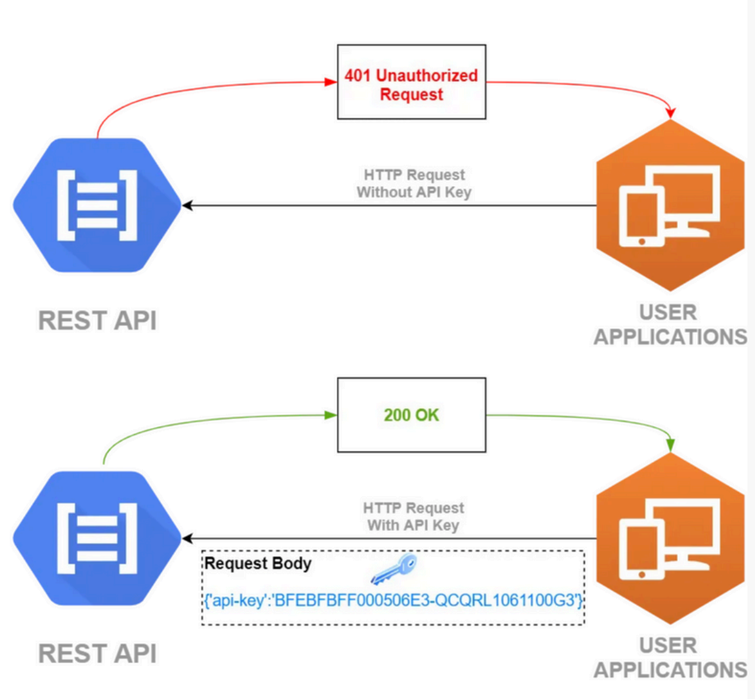
1. Tüm oturum bilgileri sunucuda tutulduğu için, kullanıcı sayısı arttıkça sunucu belleği üzerinde yük oluşturabilir.
2. Cookie çalınırsa, kötü niyetli kişiler kullanıcının oturumunu ele geçirebilir (Session Hijacking).
3. Birden fazla sunucu (örneğin load balancer yapısı) kullanıldığında session yönetimi karmaşıklaşabilir.
4. Cookie'lerin yanlış yapılandırılması (örneğin HttpOnly veya Secure bayraklarının eksik olması) güvenlik açıklarına yol açabilir.
5. Tarayıcılar cookie'leri silebilir veya engelleyebilir; bu da oturumun beklenmedik şekilde sonlanmasına neden olabilir.

Token Tabanlı Kimlik Doğrulama — Nedir?

Token tabanlı kimlik doğrulama, kullanıcı giriş yaptıktan sonra sunucunun kullanıcıya bir “token” yani dijital kimlik anahtarı vermesiyle çalışır. Bu token, sonraki isteklerde kullanıcıyı tanımak için kullanılır.

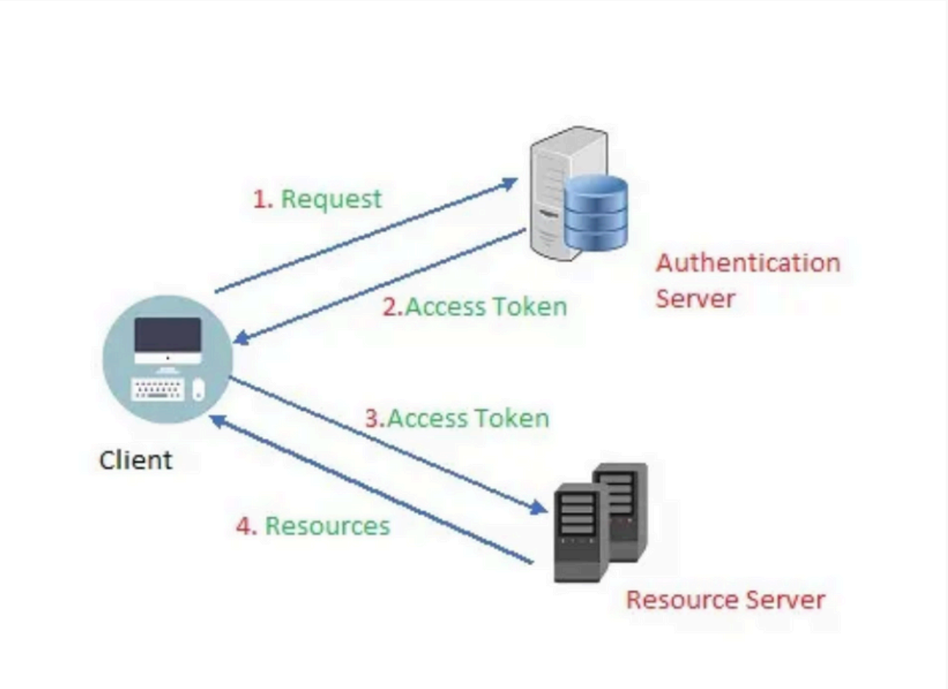
Geleneksel (session + cookie) yöntemlerde kimlik bilgisi sunucuda saklanırken, token tabanlı yapıda bu bilgi istemci tarafında tutulur ve her istekte sunucuya gönderilir. Böylece sistem “stateless” olur, yani sunucu kullanıcı bilgilerini hafızasında tutmak zorunda kalmaz.

Bu yöntem, özellikle mobil uygulamalar ve tek sayfa uygulamaları (SPA) için daha esnek ve ölçeklenebilir bir yapı sunar. Ancak token’ın güvenli saklanması ve HTTPS üzerinden iletilmesi büyük önem taşır.



Token Tabanlı Kimlik Doğrulama — Genel Akış

- Kullanıcı kimlik bilgileriyle (kullanıcı adı/şifre) oturum açar.
- Sunucu kimliği doğrular ve bir erişim tokenı (access token) (ve genellikle bir yenileme tokenı (refresh token)) döner.
- İstemci erişim tokenını sonraki API çağrılarında Authorization: Bearer <token> ile gönderir.
- Sunucu tokenı doğrular (imza/şifreleme veya lookup) ve isteği işler.
- Erişim tokenin süresi bitince istemci refresh token ile yeni erişim tokeni alır veya tekrar giriş ister.



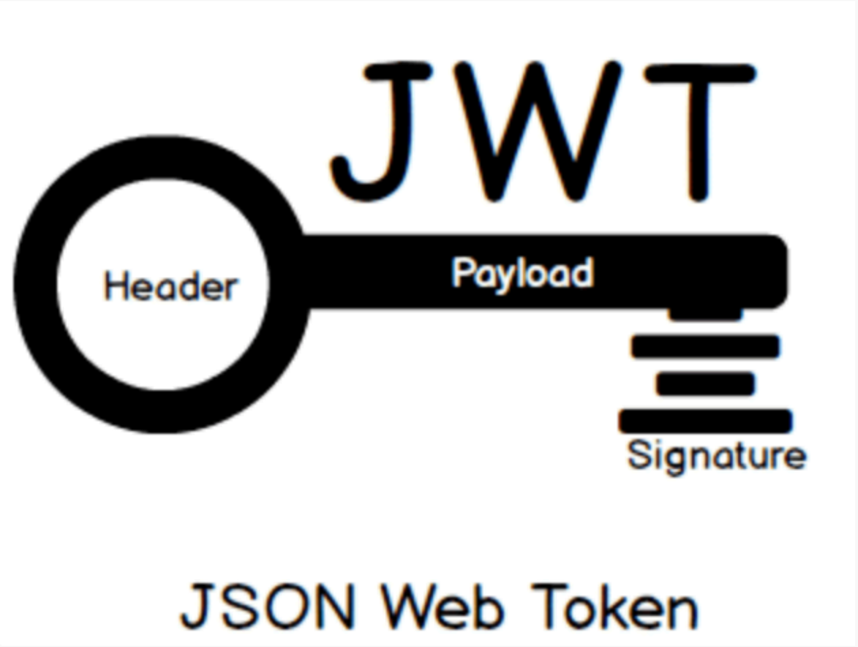
Token Tabanlı vs Cookie–Session Tabanlı Kimlik Doğrulama

Karşılaştırma Noktası	Token Tabanlı (JWT / Opaque)	Cookie – Session Tabanlı
1. Veri Saklama Yeri	Kimlik bilgileri **token içinde istemci tarafında** saklanır.	Kullanıcı bilgileri **sunucu tarafında session olarak** saklanır.
2. Durumsallık (State)	**Stateless:** Sunucu token saklamaz; her istek token üzerinden doğrulanır.	**Stateful:** Sunucu her kullanıcı için session bilgisini tutar.
3. Ölçeklenebilirlik	Mikroservis ve dağıtık sistemlerde **ölçeklendirmesi kolaydır** (sunucu yükü az).	Kullanıcı sayısı arttıkça session belleği yükü artar; **ölçeklendirme zorlaşır**.
4. Kullanım Alanı	Mobil uygulamalar, SPA'lar, API tabanlı sistemler için **idealdir**.	Klasik web uygulamaları (ör. PHP, Django, ASP.NET) için **yaygındır**.
5. Güvenlik & Revokasyon	Token çalınırsa süresi dolana kadar geçerliliğini korur; iptal edilmesi (revokasyonu) genellikle zordur.	Session iptali kolaydır; sunucudan silindiğinde **hemen geçersiz olur**.

JWT (JSON Web Token)

Nedir:
JWT, kullanıcı doğrulandıktan sonra ona verilen dijital bir “kimlik kartı” gibidir. Bu kartın içinde kullanıcının kim olduğu ve hangi yetkilere sahip olduğu bilgileri yer alır. Bu bilgiler şifrelenmez, sadece imzalanır; böylece token değiştirilmeye çalışılsa bile sunucu bunu fark eder.

Neden Kullanılır:
JWT, özellikle mobil uygulamalarda ve API tabanlı sistemlerde çok tercih edilir çünkü sunucuda oturum tutmaya gerek yoktur. Her istekle birlikte istemci (örneğin tarayıcı veya mobil uygulama) bu token’ı gönderir, sunucu da token’ın geçerli olup olmadığına bakar. Bu yüzden sistem daha hızlı, daha az kaynak tüketen ve ölçeklenebilir olur.



JWT (JSON Web Token) Çalışma Mantığı

1. Kullanıcı giriş yapar: Kullanıcı, örneğin kullanıcı adı ve şifre ile sisteme giriş yapar.
2. Sunucu JWT üretir: Sunucu, kullanıcının kimliğini doğruladıktan sonra bu kullanıcıya özel bir JWT (token) oluşturur ve istemciye gönderir.
3. Token istemcide saklanır: İstemci (örneğin tarayıcı veya mobil uygulama), aldığı bu token'ı localStorage veya güvenli bir cookie içinde saklar.
4. Her istekte token gönderilir: Kullanıcı sistemde işlem yaptıkça, istemci bu token'ı her isteğe ekleyip sunucuya gönderir.
5. Sunucu token'ı doğrular: Sunucu gelen token'ın geçerli olup olmadığını kontrol eder, kimliği doğrular ve buna göre isteği işler.

örnek :

Bir e-ticaret sitesine giriş yaptığında sistem sana bir "geçici kimlik kartı" (JWT) verir. Sen bu kartla sepetine ürün ekleyebilir, ödeme yapabilir veya hesabını görüntüleyebilirsin — her seferinde yeniden giriş yapmana gerek kalmaz.

JWT Token Üretimi ve Çözümlemesi – Python Örneği

```python
import jwt #pip install PyJWT
import datetime
import json

SECRET_KEY = "gizli_anahtar_123"

payload = {
    "user_id": 42,
    "username": "huseyin",
    "role": "admin",
    "exp": datetime.datetime.now(datetime.UTC) + datetime.timedelta(minutes=10)
}

token = jwt.encode(payload, SECRET_KEY, algorithm="HS256")
print("Oluşturulan JWT Token:\n", token)

try:
    decoded = jwt.decode(token, SECRET_KEY, algorithms=["HS256"])
    json_output = json.dumps(decoded, indent=4, ensure_ascii=False)
    print("\nToken Geçerli ")
    print("Çözümlenen JSON veri:")
    print(json_output)

except jwt.ExpiredSignatureError:
    print("\nToken süresi dolmuş")
except jwt.InvalidTokenError:
    print("\nGeçersiz token")
```

- payload: Token içinde taşınacak verileri (örneğin kullanıcı kimliği, rolü, geçerlilik süresi) içerir.
- SECRET_KEY: Token'ı imzalamak için kullanılan gizli anahtar (güvenli bir yerde tutulmalı).
- exp: Token'ın geçerlilik süresini belirler (burada 10 dakika).
- jwt.encode() → Token oluşturur.
- jwt.decode() → Token'ı çözer ve geçerli olup olmadığını kontrol eder.

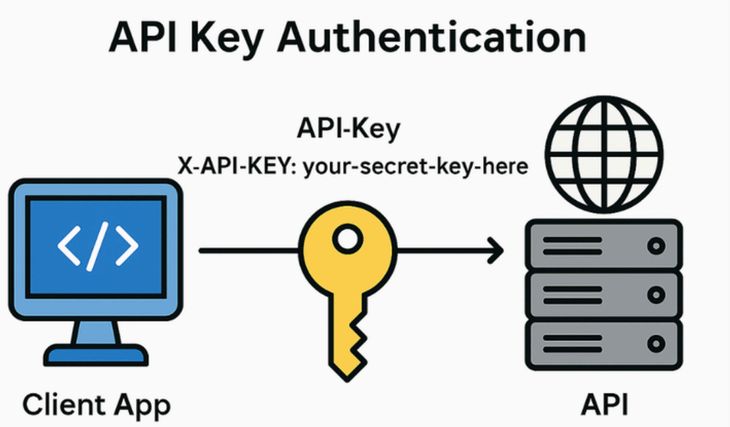
```
Oluşturulan JWT Token:
 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjo0MiwidXNlcm5hbWUiOiJodXNl

Token Geçerli ✅
Çözümlenen JSON veri:
{
    "user_id": 42,
    "username": "huseyin",
    "role": "admin",
    "exp": 1761854797
}
```

JWT Token Avantajları ve Dezavantajları/Riskleri

Kategori	Avantajlar	Riskler / Dikkat Edilmesi Gerekenler
Ölçeklenebilirlik & Performans	**Stateless Çalışma:** Sunucuda oturum bilgisi saklamaya gerek yok; her istek token ile doğrulanır **ölçeklenebilir sistemler için ideal.**	**Revokasyon Zorluğu:** Stateless olduğu için token'ı sunucudan iptal etmek zor; **blacklist/allowlist gerekir.**
Doğrulama Hızı	**Hızlı Doğrulama:** Token imzası ile doğrudan kullanıcı kimliği doğrulanabilir **her istekte DB sorgusu gerekmez.**	**Yanlış Konfigürasyon:** Algoritma seçimi, secret key yönetimi ve HTTPS zorunluluğu ihmal edilirse **güvenlik ciddi şekilde tehlikeye girer.**
Mimari Uyum	**Mikroservis Entegrasyonu Kolay:**Token aynı anda farklı servisler tarafından kullanılabilir **SSO veya API-first uygulamalar için uygun.**	**Token Çalınması:** Token bir şekilde ele geçirilirse süresi boyunca geçerli olur **kısa ömürlü token + refresh önerilir.**
Yetkilendirme	**Esnek Yetkilendirme:** Token payload'ına rol ve yetki bilgisi eklenerek route bazlı kontrol yapılabilir.	**Payload Gizli Değil:** Base64 encode edilmiş bilgiler **okunabilir hassas veri saklamaktan kaçınılmalı.**
Platform Desteği	**Mobil ve SPA Dostu:** Web, mobil veya tek sayfa uygulamalarda rahatça taşınabilir ve HTTP header veya cookie ile iletilebilir.	**Yanlış Saklama Riski:** Token'ı localStorage veya JS üzerinden taşımak **XSS riskini artırır HttpOnly cookie tercih edilmeli.**

API Key Nedir?

API Key, bir uygulamanın veya kullanıcının bir API'ye erişim hakkını tanımlayan benzersiz bir kimlik doğrulama anahtarıdır. Genellikle uzun ve rastgele bir karakter dizisinden oluşur ve her API sağlayıcısı kendi sisteminde bu anahtarı oluşturur. Kullanıcı veya uygulama, API'ye yaptığı her istekte bu anahtarı gönderir; API sunucusu anahtarı doğruladıktan sonra isteği kabul eder. API Key, özellikle servisler arası iletişimde, uygulamaların kimliklerini doğrulamak ve yetkilendirmek için kullanılır.



API Key'in Kullanım Amacı ve Avantajı

API Key, temel olarak bir API'ye erişimi kontrol etmek ve kimlik doğrulama sağlamak için kullanılır. Örneğin, bir harita servisi, sadece geçerli bir API Key ile yapılan istekleri işler; geçersiz veya eksik bir anahtar varsa istek reddedilir. API Key'ler genellikle basit ve hızlı bir doğrulama yöntemi sunar, kullanıcı adı ve şifre gibi ek bilgi gerektirmez. Ancak API Key'ler genellikle kısa süreli yetki kontrolü veya gelişmiş kullanıcı rolü kontrolü sağlamaz; daha güvenli uygulamalarda JWT veya OAuth gibi ek mekanizmalarla desteklenir.

OAuth2’de Token Nedir

OAuth2’de kullanıcı, bir uygulamaya (client) doğrudan kullanıcı adı ve şifresini vermez. Bunun yerine kullanıcı, yetkilendirme sunucusu üzerinden uygulamaya bir token verir. Bu token, uygulamanın kullanıcının adına kaynak sunucusuna (örn. Google Drive, Spotify, vs.) erişmesine izin verir.

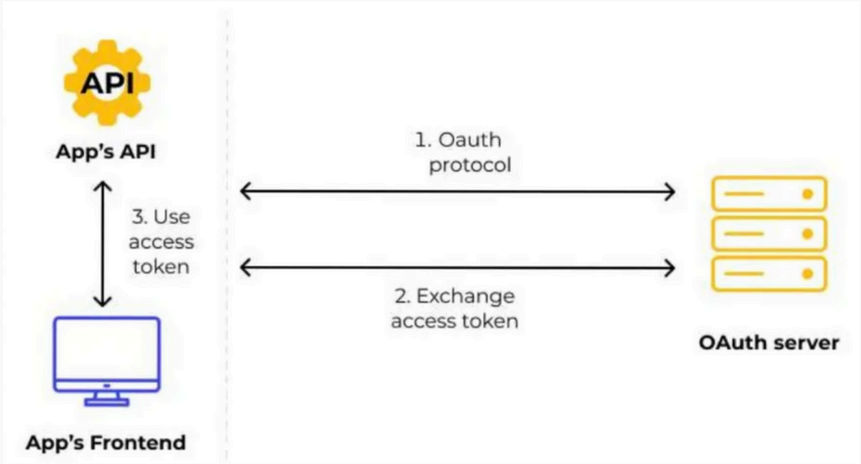
Yani token: “Bu uygulama benim adıma bu işlemi yapabilir” diyen dijital bir izin belgesidi

örnek :

Bir uygulama, örneğin “Tweetlerini analiz eden bir uygulama”, senin Twitter hesabına erişmek ister.

Uygulama şifreni asla istemez. Bunun yerine seni Twitter’ın yetkilendirme sayfasına yönlendirir.

Sen izin verdiğinde, uygulama şifreni görmeden sadece senin onayladığın işlemleri yapabilir; örneğin tweetlerini okuyup analiz edebilir. Böylece uygulama, hesabında yalnızca senin verdiğin izinler kadar işlem yapar ve şifren hâlâ gizli kalır.

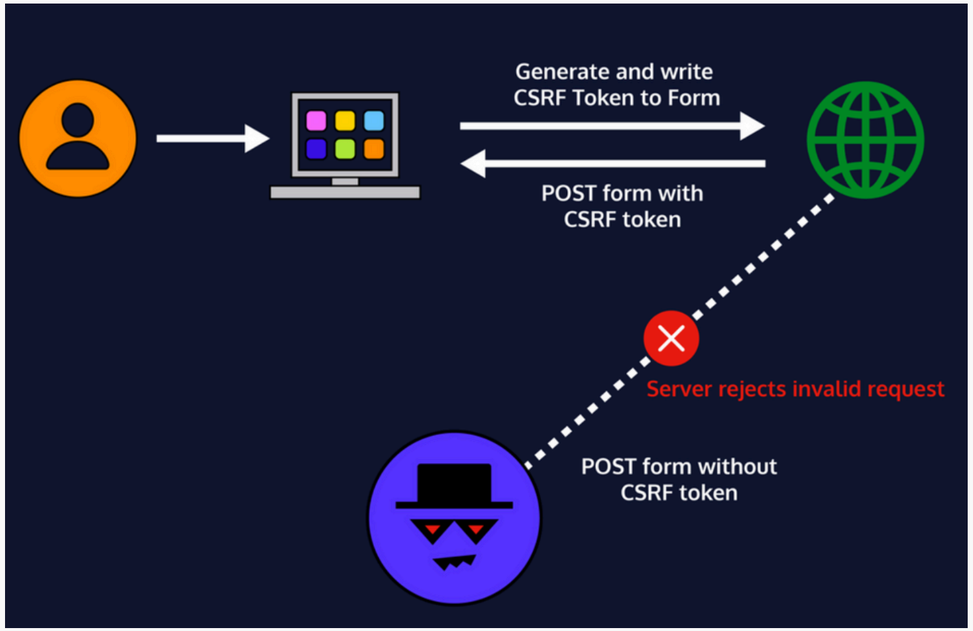


OAuth2 Yetkilendirme Sürecinde Token Kullanımının Sağladığı Avantajlar

Avantaj Türü	Açıklama	Örnek / Not
Güvenlik	Kullanıcı şifresi paylaşılmaz, yalnızca **Erişim Belirteci (Token)** üzerinden yetkilendirme yapılır.	Uygulama, şifrenizi görmeden sadece izin verilen işlemleri yapabilir.
Süreli ve Sınırlı Erişim	Access Token (Erişim Belirteci) genellikle kısa ömürlüdür; yetki süresi dolunca geçersiz olur.	Kötü niyetli bir token ele geçirilse bile, kısa süreyle erişebilir ve etkisi sınırlı kalır.
Yetki Sınırlaması (Scope)	Token ile uygulamaya sadece **belirli izinler**(kapsamlar) verilebilir.	Sadece tweet'lerinizi okumaya izin verip, **DM'lerinize (Özel Mesajlarınıza) erişimi engelleyebilirsiniz**.
Taşınabilirlik	Token, **web, mobil veya IoT** gibi farklı uygulama türlerinde kullanılabilir.	Aynı token, mobil uygulama ve web uygulaması tarafından API çağrılarında geçerli olabilir.
Refresh Mekanizması	Uzun süreli erişim gerekiyorsa, **Refresh Token** ile kullanıcı tekrar giriş yapmadan yeni Access Token alınabilir.	Kullanıcı tekrar giriş yapmaya zorlanmadan uygulama erişimini sürdürebilir.
Kullanıcı Deneyimi	Kullanıcı şifreyi tekrar tekrar girmek zorunda kalmaz. Tek Tıkla Giriş (SSO) deneyimi sağlar.	Spotify veya Twitter uygulamaları arka planda token ile erişim sağlar, her açtığınızda şifre sormaz.
Kolay İptal	İstenildiği zaman token iptal edilebilir veya süresi kısaltılabilir.	Hesap güvenliği tehdit edildiğinde **yalnızca token iptal edilir**, şifre değiştirmeye gerek kalmaz.

CSRF Token Nedir?

CSRF (Cross-Site Request Forgery), Türkçesiyle Siteler Arası İstek Sahteciliği, bir saldırı türüdür.
Bu saldırıda kötü niyetli bir web sitesi veya uygulama, kullanıcının tarayıcısında zaten giriş yapılmış (authenticated) bir siteye, fark ettirmeden yetkisiz istekler gönderir.



Örnek: Kullanıcı banka sitesine giriş yapmış ve tarayıcısında oturumu açık. Aynı kullanıcı, kötü niyetli bir siteyi ziyaret ettiğinde bu site, kullanıcının adına banka hesabından para transferi yapacak bir istek gönderebilir. Tarayıcı otomatik olarak çerezleri gönderdiği için banka sunucusu bunu meşru bir istek gibi kabul eder.

İşte CSRF Token, bu tür saldırılara karşı alınan bir önlem mekanizmasıdır.

CSRF Token’ın Özellikleri ve İşleyişi

1. Token Oluşturma: Sunucu, kullanıcı oturum açtığında her kullanıcı için rastgele ve benzersiz bir token üretir.
2. Token’ın Taşınması: Bu token, genellikle HTML formunun içine gizli bir alan (<input type="hidden">) veya HTTP header aracılığıyla istemciye gönderilir.
3. İstek Doğrulama: Kullanıcı formu gönderdiğinde veya kritik bir işlem yaptığında (ör. ödeme, şifre değişikliği) token sunucuya iletilir. Sunucu, gelen token’ı kendi sakladığı token ile karşılaştırır. Eğer tokenlar eşleşiyorsa, istek meşru kabul edilir.
4. Saldırı Önleme: Kötü niyetli bir site, kullanıcının token’ını bilemez. Dolayısıyla kendi gönderdiği isteklerde token eksik veya yanlış olur ve sunucu isteği reddeder.
5. Tek Kullanımlık ve Süreli Kullanım: Bazı uygulamalarda token tek seferlik veya süreli olabilir, yani bir kez kullanıldıktan sonra geçersiz hâle gelir. Bu, güvenliği daha da artırır.

CSRF Token Neden Kullanılır?

- **Sadece gerçekte oturum açmış kullanıcının isteği olduğunu garantilemek** için. CSRF token, sunucunun "bu isteği gerçekten kullanıcı mı gönderdi yoksa üçüncü bir site mi taklit ediyor?" sorusuna net bir evet/hayır cevabı verir.
- **Kullanıcının isteği ile kötü niyetli isteği ayırır**. Tarayıcı otomatik olarak çerezleri gönderir; CSRF token ise yalnızca sayfayı/forumu gerçekten açan ve o sayfada token'ı gören kullanıcı tarafından gönderilebilir — kötü siteler bu token'a erişemeyeceği için saldırı başarısız olur.
- Yetkisiz işlemleri (ör. para transferi, şifre değişikliği) engellemek için. Kritik eylemler token doğrulaması olmadan işleme alınmaz; böylece kullanıcı bilgisi dışında yapılan işlemler reddedilir.
- Gerçek kullanıcıyla sahte isteğin arasındaki kesin ayırıcıdır. Token, istemcide (form veya header) bulunur; sunucuda saklanan beklenen token ile eşleşmezse istek iptal edilir — yani istek yapanın "o kullanıcı" olduğuna dair kanıt olmaz.
- Ek güvenlik sağlar (tek kullanımlık/süreli token ile). Token kısa ömürlü veya tek seferlik olursa, ele geçirilse bile kullanımı sınırlanır; böylece oturum boyunca sadece belirli, doğrulanmış istekler kabul edilir.

CSRF token, "tarayıcı çerezini otomatik göndermek isteyen herkes değil; bu isteği gerçekten kullanıcı mı başlattı?" sorusunun sunucuda kesin şekilde cevaplanmasını sağlar — yani yalnızca oturum açmış kullanıcının istekleri kabul edilir.

REST API Güvenliğinde Tokenlerin Rolü

Özetle :
REST API güvenliğinde tokenler, hem kimlik doğrulama ve yetkilendirme hem de isteğin güvenliğini sağlamak için kritik öneme sahiptir. OAuth 2.0 Access Tokenler, kullanıcı parolasını tekrar göndermeye gerek kalmadan API kaynaklarına sınırlı süreli ve kapsamlı erişim sağlar; böylece yetkisiz erişim engellenir ve mikroservis mimarilerinde stateless ölçeklenebilirlik mümkün olur.

CSRF tokenleri ise özellikle tarayıcı tabanlı saldırılara karşı koruma sağlar. Rastgele üretilen bu tokenler, her hassas istekte sunucuya iletilerek, yalnızca uygulamanın kendi arayüzünden gelen geçerli isteklerin kabul edilmesini garanti eder ve kullanıcının bilgisi dışında zararlı işlemlerin yapılmasını önler.

Buna ek olarak JWT (JSON Web Token) mekanizması, token içeriğinin dijital olarak imzalanmasını sağlayarak hem doğruluk hem de bütünlük kontrolü sunar. JWT'ler, taşınabilir ve self-contained oldukları için kullanıcı bilgilerini ve yetki alanlarını güvenli şekilde taşır; böylece hem kimlik doğrulama hem de yetki kontrolü tek bir yapı üzerinden sağlanabilir, ve tokenler çalınsa bile süresi dolana kadar yetkisiz kullanım riski sınırlandırılmış olur.