BLOKZİNCİR TABANLI DİJİTAL SERTİFİKA SİSTEMİ TEKNİK RAPORU

Ders: Dijital Dönüşüme Giriş

Tarih: 29 Kasım 2024

Hazırlayan: Ahmet Bağrıyanık

Numara: 231229068

1. Yönetici Özeti

Bu proje, kurumlar arası güveni tesis etmek ve belge sahteciliğini önlemek amacıyla geliştirilmiş, blokzincir tabanlı bir dijital sertifika yönetim sistemidir. Sistem, Docker üzerinde çalışan mikroservis mimarisi ile kurgulanmış olup; sertifika oluşturma (issuance), iptal etme (revocation) ve doğrulama (verification) süreçlerini uçtan uca, üçüncü parti bir aracıya ihtiyaç duymadan gerçekleştirmektedir.

2. Sistem Mimarisi

Proje, dağıtık ve modüler bir yapı sağlamak amacıyla Konteyner Tabanlı (Docker) bir mimari üzerine inşa edilmiştir. Sistem üç ana bileşenden oluşur:

2.1. Bileşenler

Blokzincir Ağı (Ganache): Yerel bir Ethereum (EVM) ağı simülasyonudur. Tüm işlem kayıtlarının ve akıllı kontratın (CertificateRegistry) barındığı defter-i kebir (ledger) katmanıdır.

Akıllı Kontrat Geliştirme Katmanı (Hardhat): Solidity tabanlı kontratların derlenmesi, test edilmesi ve Ganache ağına dağıtılması (deploy) görevini üstlenen köprü katmanıdır.

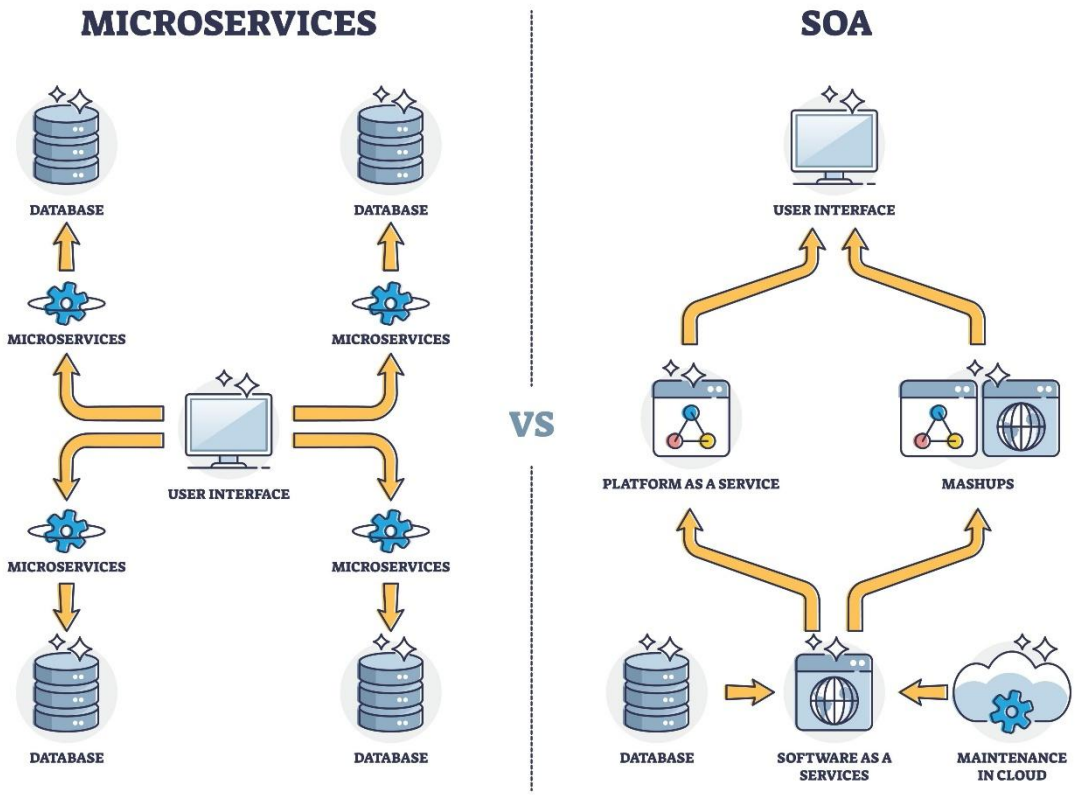
İstemci Uygulaması (Client Service): Node.js tabanlıdır. Kullanıcı verilerini şifreler (Hash), blokzincir ile ethers.js kütüphanesi üzerinden haberleşir ve doğrulama sonuçlarını son kullanıcıya sunar.

2.2. Ağ Topolojisi

Tüm servisler, izole edilmiş bir Docker ağı (certnet) üzerinde çalışır. Bu sayede servisler birbirleriyle konteyner isimleri (hostname) üzerinden haberleşir, dış dünyadan soyutlanmış güvenli bir iletişim kanalı oluşturulur.

Veri Akış Şeması: Kullanıcı -> [Client] --(RPC/JSON)--> [Ganache Network] <--(Deploy/Test)-- [Hardhat]

Aşağıdaki görsel veri akış şemasını temsil etmektedir.



3. Tasarım Gerekçeleri

3.1. Neden Blokzincir?

Geleneksel veritabanı sistemlerinde, merkezi otorite (örneğin üniversite) veriyi silebilir veya değiştirebilir. Blokzincir kullanımı ile:

Değiştirilemezlik (Immutability): Sertifika bir kez verildiğinde, geriye dönük olarak tarihi veya içeriği değiştirilemez.

Güvenilirlik (Trustless Verification): Doğrulayıcı kurumlar (örneğin işverenler), sertifikayı veren kuruma sormadan, doğrudan zincir üzerinden matematiksel kesinlikle doğrulama yapabilir.

3.2. Neden Docker ve Mikroservisler?

Taşınabilirlik: "Benim bilgisayarımda çalışıyordu" sorununu ortadan kaldırmak için tüm bağımlılıklar (Node.js sürümü, Ganache ayarları) konteyner içine hapsedilmiştir.

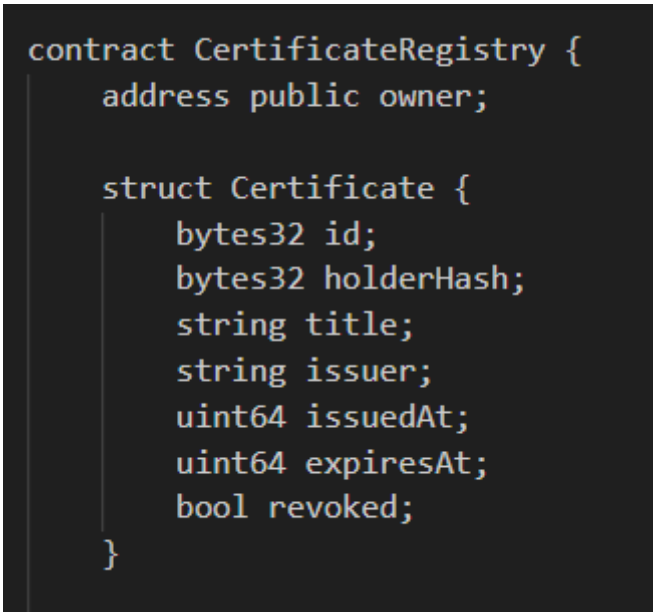
İzolasyon: Her servis kendi görevini yapar. İstemci çökerse blokzincir ağı çalışmaya devam eder.

4. Akıllı Kontrat Tasarımı (CertificateRegistry.sol)

Kontrat, EVM (Ethereum Virtual Machine) standartlarına uygun olarak Solidity 0.8.20 sürümü ile geliştirilmiştir.

4.1. Veri Yapısı (Struct)

Verimlilik (Gas Optimization) gözetilerek şu yapı kullanılmıştır:



```
contract CertificateRegistry {
    address public owner;

    struct Certificate {
        bytes32 id;
        bytes32 holderHash;
        string title;
        string issuer;
        uint64 issuedAt;
        uint64 expiresAt;
        bool revoked;
    }
```

4.2. Fonksiyonlar

issue(): Sadece yetkili cüzdan (onlyOwner) tarafından çağrılabilir. Sertifikayı zincire kaydeder.

```
function issue(bytes32 id, bytes32 holderHash, string calldata title, string calldata issuer, uint64 expiresAt) external onlyOwner {
    require(certificates[id].issuedAt == 0, "exists");
    certificates[id] = Certificate(id, holderHash, title, issuer, uint64(block.timestamp), expiresAt, false);
    emit CertificateIssued(id, holderHash, title, issuer, uint64(block.timestamp), expiresAt);
}
```

verify(): Herkes tarafından çağrılabilir (public view). Girilen ID ve Hash değerlerini zincirdeki kayıtla karşılaştırır.

```
function verify(bytes32 id, bytes32 holderHash) external view returns (bool valid, bool isRevoked, uint64 issuedAt, uint64 expiresAt, string
    Certificate memory c = certificates[id];
    if (c.issuedAt == 0) {
        return (false, false, 0, 0, "", "");
    }
    bool ok = !c.revoked && c.holderHash == holderHash && (c.expiresAt == 0 || c.expiresAt >= block.timestamp);
    return (ok, c.revoked, c.issuedAt, c.expiresAt, c.title, c.issuer);
}
```

revoke(): Hatalı veya geri çekilmesi gereken sertifikalar için durum güncellemesi yapar.

```
function revoke(bytes32 id) external onlyOwner {
    Certificate storage c = certificates[id];
    require(c.issuedAt != 0, "not found");
    require(!c.revoked, "already revoked");
    c.revoked = true;
    emit CertificateRevoked(id, uint64(block.timestamp));
}
```

5. Güvenlik ve KVKK (Veri Gizliliği) Değerlendirmesi

Projenin en kritik tasarım kararları, Kişisel Verilerin Korunması Kanunu (KVKK) ve GDPR uyumluluğu üzerine kurulmuştur.

5.1. Hash ve Salt (Tuzlama) Stratejisi

Blokzincir halka açık bir yapıdır. Bu nedenle Ad, Soyad, TCKN gibi hassas veriler asla düz metin (plaintext) olarak zincire yazılmamıştır.

Yöntem: Veriler istemci tarafında (off-chain) birleştirilir ve kriptografik özet fonksiyonundan geçirilir.

Salting (Tuzlama): Sadece hash almak, "Rainbow Table" saldırılarına karşı yetersizdir. Bu projede rastgele üretilen bir Salt değeri eklenerek hash'in geri döndürülemez olması sağlanmıştır.

Formül: Zincire Yazılan Veri = Keccak256(ÖğrenciNo + AdSoyad + RastgeleTuz)

```
// 1. Veri Hazırlama (Hashleme)
const ogrNo = "12345";
const adSoyad = "Ahmet Yilmaz";
const salt = "gizliTuz123"; // Rastgele üretilmeli

const payload = `${ogrNo}|${adSoyad.toUpperCase().trim()}|${salt}`;
const holderHash = ethers.keccak256(ethers.toUtf8Bytes(payload));
const certId = ethers.id("Sertifika-" + Date.now()); // Unique ID

console.log(`Sertifika ID: ${certId}`);
console.log(`Kişi Hash (Gizli): ${holderHash}`);
```

5.2. Erişim Kontrolü (Access Control)

Akıllı kontratta onlyOwner modifiyeri kullanılmıştır. Bu, herhangi birinin sahte sertifika eklemesini matematiksel olarak engeller. Sadece kontratı dağıtan (Deployer) cüzdanın imzaladığı işlemler kabul edilir.

6. Test Kanıtları ve Senaryolar

Sistem üzerinde gerçekleştirilen test senaryoları ve sonuçları aşağıdadır:

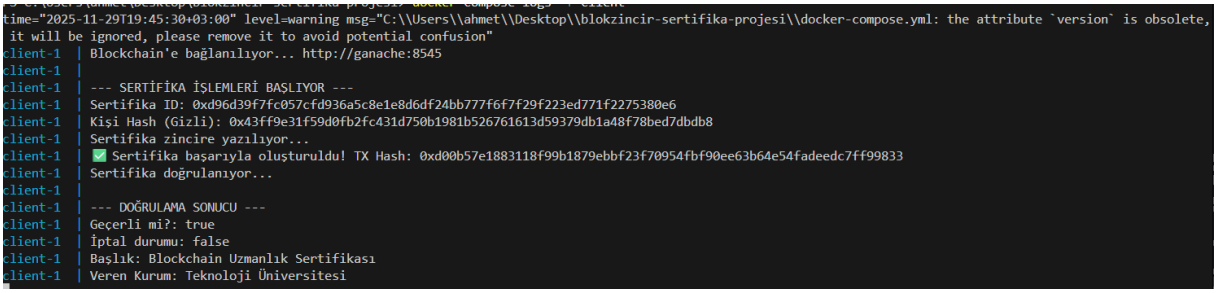
6.1. Senaryo 1: Başarılı Sertifika Oluşturma ve Doğrulama

İşlem: Yetkili hesap sertifikayı oluşturdu. İstemci doğru ID ve Hash ile sorgulama yaptı.

Beklenen Sonuç: valid: true dönmesi.

Gerçekleşen Sonuç: Başarılı.

Kanıt: Ekteki görseli inceleyebilirsiniz.

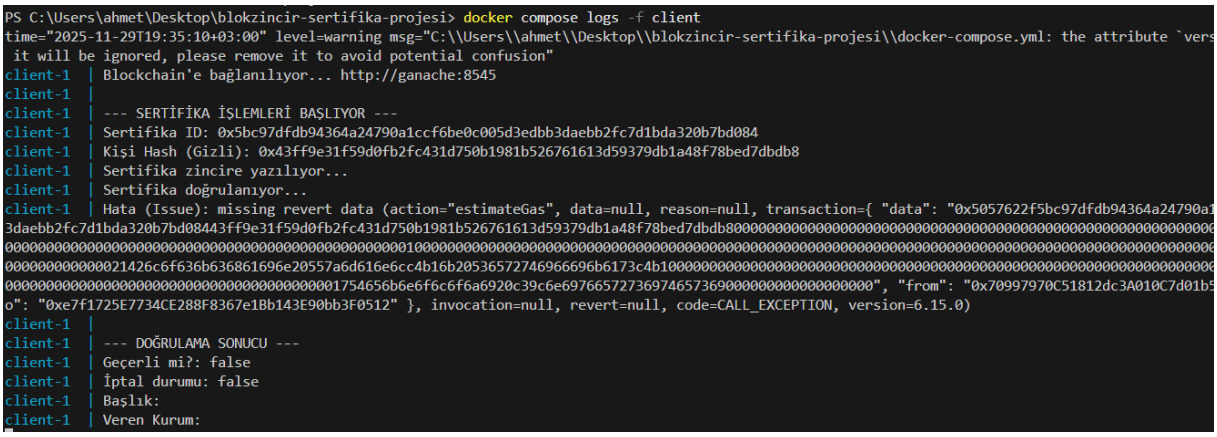


6.2. Senaryo 2: Yetkisiz Erişim Denemesi

İşlem: Yetkisi olmayan (Owner olmayan) bir cüzdan ile issue fonksiyonu çağrıldı.

Beklenen Sonuç: İşlemin reddedilmesi (revert: not owner).

Gerçekleşen Sonuç: Hardhat konsolunda işlem reddedildi hatası alındı. Güvenlik mekanizmasının çalıştığı doğrulandı. Aşağıdaki görselde işlem yapılmak istenince “Yetkisiz cüzdan ile işlem yapıldığında kontrat işlemi reddetti” durumu oluştu.



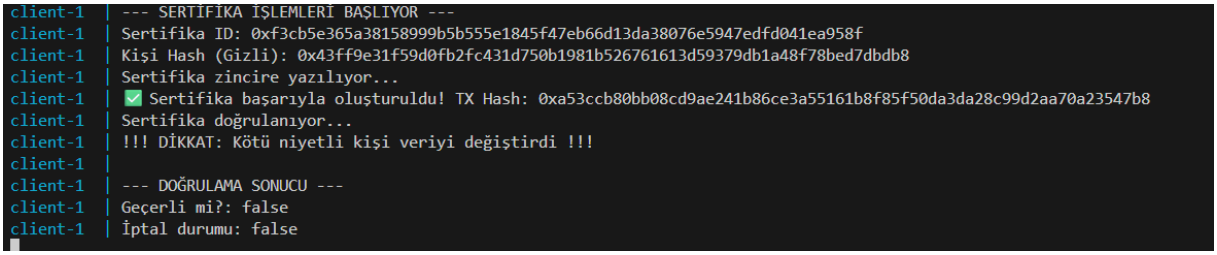
6.3. Senaryo 3: Veri Bütünlüğü (Yanlış Veri Testi)

İşlem: Sertifika oluşturulduktan sonra, doğrulama sırasında öğrenci adında tek bir harf değiştirilerek hash üretildi.

Beklenen Sonuç: verify fonksiyonunun valid: false döndürmesi.

Gerçekleşen Sonuç: Zincirdeki hash ile uyuşmadığı için doğrulama başarısız oldu.

"Veri üzerinde tek bir harf değişikliği yapıldığında hash değiştiği için doğrulama başarısız oldu"



7. Sonuç

Bu proje kapsamında geliştirilen Blokzincir Tabanlı Sertifika Sistemi, merkeziyetsiz teknolojilerin veri güvenliği ve doğrulama süreçlerinde nasıl etkin kullanılabileceğini kanıtlamıştır. Docker konteynerizasyonu sayesinde kurulum maliyetleri düşürülmüş, "Hash+Salt" yöntemi ile kişisel verilerin gizliliği (KVKK) tam uyumlu hale getirilmiştir. Sistem, üretim ortamına taşınmaya hazır temel yetkinliklere sahiptir.