



**KARADENİZ TEKNİK ÜNİVERSİTESİ BİLGİSAYAR BİLGİSAYAR
GÜVENLİĞİ TEMELLERİ DERSİ DÖNEM ÖDEVİ RAPORU**

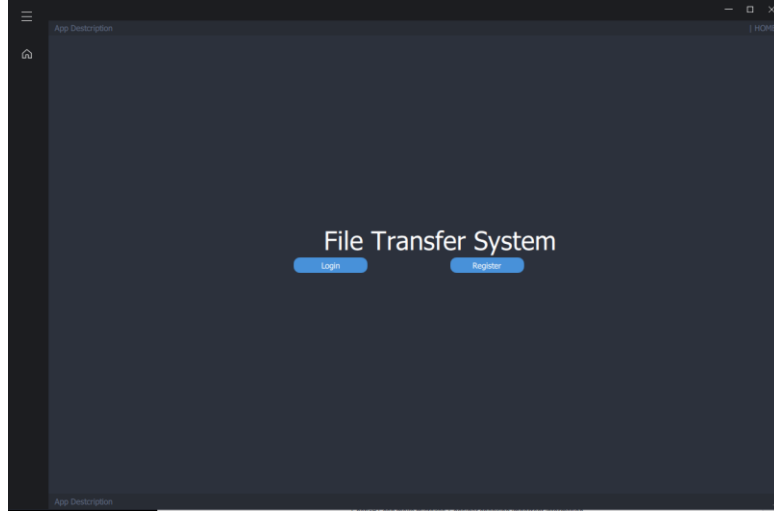
Ufuk BULUT (394811)

Osman Can AKSOY(394797)

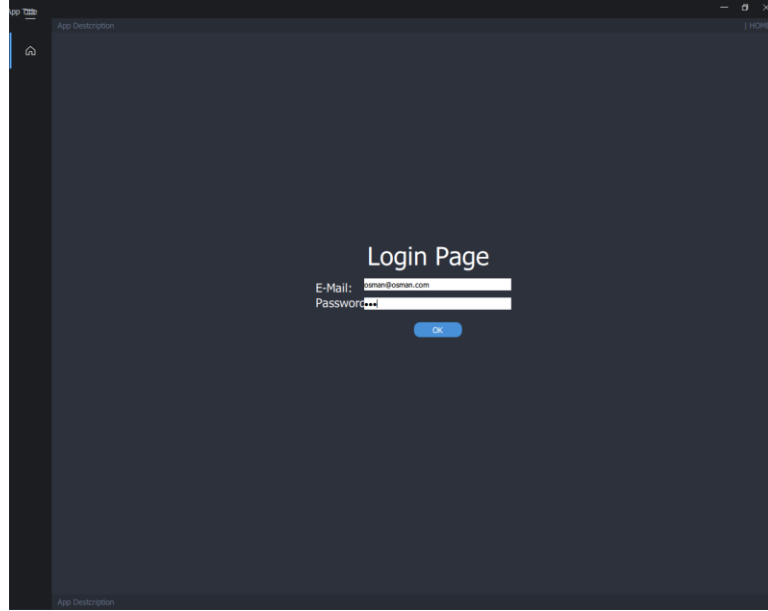
GİRİŞ

Bu projede, bilgisayar güvenliği temelleri kapsamında "Güvenli Bir Dosya Transferi Aracı Geliştirilmesi" amacıyla bir yazılım oluşturulmuştur. Proje, dosya transferi sırasında kullanıcı bilgilerinin güvenli bir şekilde saklanması, iletim esnasında güvenli bağlantı kurulması, dosyaların şifrlenmesi ve deşifrlenmesi, ayrıca uygulama etkinliklerinin loglanması ve hata izleme mekanizmalarının entegrasyonunu içermektedir. Bu dört temel modül, kullanıcı kaydı ve oturum yönetimi, güvenli bağlantı kurma, dosya şifreleme ve loglama/hata izleme olarak belirlenmiştir.

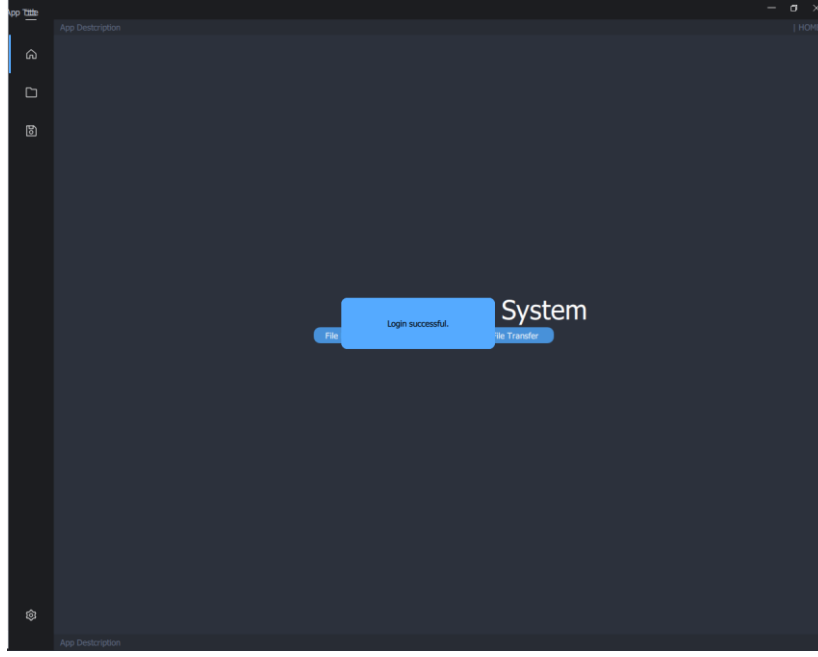
1. Kullanıcı Kaydı ve Oturum Yönetimi Modülü:



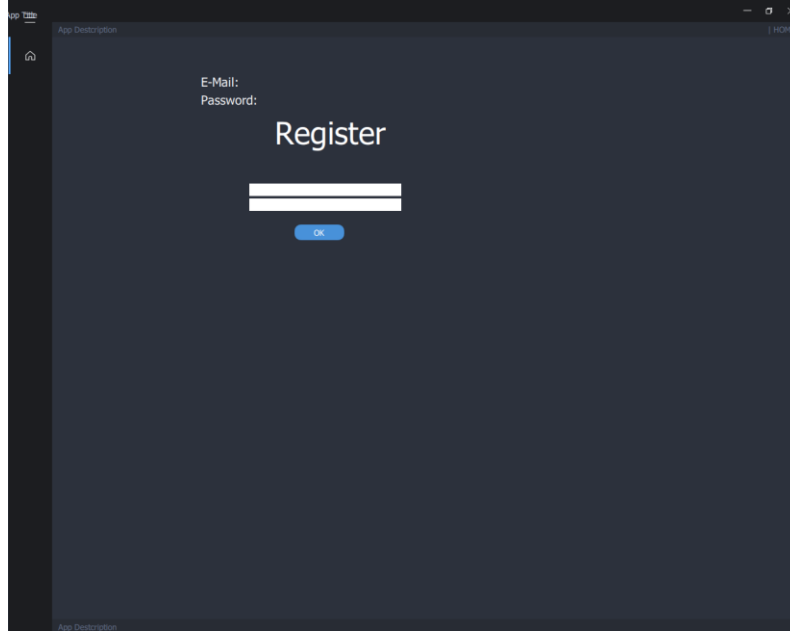
Şekil 1 Uygulama Açılış Ekranı



Şekil 2 Uygulama Giriş Ekranı



Şekil 3 Başarılı Giriş Sonrasındaki Uygulama İçindeki Ekran Görüntüsü



Şekil 4 Kayıt Olma Ekranı

Bağlantı Kurma ve Veritabanı Ayarları:

Sınıf, QObject temel sınıfından türetilmiştir ve QSqlDatabase kullanılarak SQLite veritabanına bağlanmaktadır.

Veritabanı bağlantısı başarılı bir şekilde kurulduğunda bilgilendirme mesajları yazdırılmıştır.

```
Database::Database(QObject *parent) : QObject(parent)
{
    mydb = QSqlDatabase::addDatabase("SQLITE");
    mydb.setDatabaseName("database.db");
    if(!mydb.open()) {
        qCritical() << "Could not connect to database.";
    }
}
```

```

        cout << "Could not connect to database." << endl;
    }
    else {
        qInfo() << "Connected to the database.";
        cout << "Connected to the database." << endl;
    }

    // QSqlQuery query(mydb);
    // query.exec("create table users(id integer primary key, email text,
    // password text)");
}

```

Kullanıcı Girişi İçin Fonksiyon:

get_login_values fonksiyonu, kullanıcının giriş bilgilerini alır, şifreyi SHA256 hash'leyip veritabanında kontrol eder ve girişin başarılı olup olmadığını bildiren bir sinyal gönderir.

```

void Database::get_login_values(QString email, QString password)
{
    QSqlQuery query(mydb);
    query.prepare("select * from users where email = :email and password =
:password");
    query.bindValue(":email", email);
    query.bindValue(":password",
QCryptographicHash::hash(password.toLocal8Bit(),
QCryptographicHash::Sha256).toHex());
    query.exec();

    if(query.next()) {
        qInfo() << "Login to the system is successful.";
        cout << "Login to the system is successful." << endl;
        emit pushQueryStatus(true);
        emit pushLoginStatus(true);
    }
    else {
        qCritical() << "Login to the system failed.";
        cout << "Login to the system failed." << endl;
        emit pushQueryStatus(false);
    }
}

```

Oturum Kapatma İçin Fonksiyon:

get_logout_status fonksiyonu, kullanıcının oturumu kapattığına dair bir bilgi mesajı gönderir.

```

void Database::get_logout_status()
{
    qInfo() << "The system has been logged out.";
    cout << "The system has been logged out." << endl;
    emit pushLogoutStatus(true);
}

```

Kullanıcı Kayıt İçin Fonksiyon:

get_register_values fonksiyonu, kullanıcının kayıt bilgilerini alır, şifreyi SHA256 hash'leyip kullanıcı bilgilerini veritabanına ekler ve kayıt işleminin başarılı olup olmadığını bildiren bir sinyal gönderir.

```

void Database::get_register_values(QString email, QString password)
{
    QSqlQuery query(mydb);
    query.prepare("insert into users (email, password) values(:email,
:password)");
    query.bindValue(":email", email);
    query.bindValue(":password",
QCryptographicHash::hash(password.toLocal8Bit(),
QCryptographicHash::Sha256).toHex());

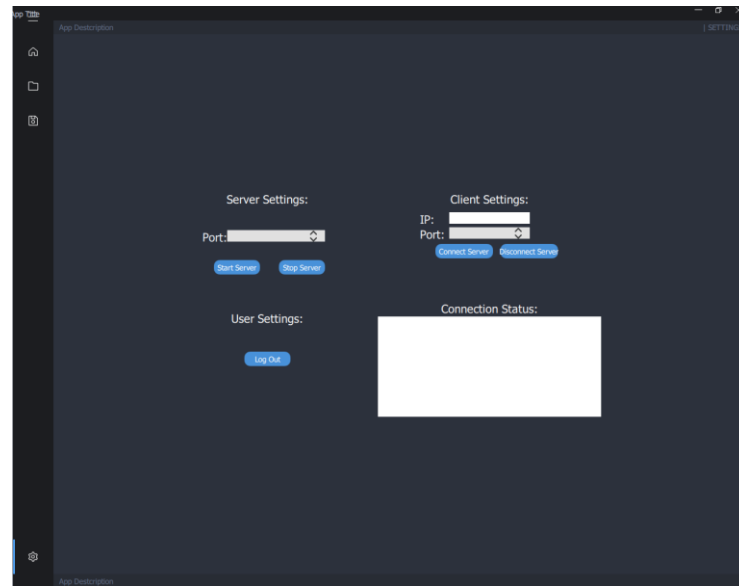
    if(query.exec()) {
        qInfo() << "Registration to the system was successful.";
        cout << "Registration to the system was successful." << endl;
        emit pushQueryStatus(true);
    }
    else {
        qCritical() << "Registration failed.";
        cout << "Registration failed." << endl;
        emit pushQueryStatus(false);
    }
}

```

Bu modül, kullanıcı yönetimi işlevselliği için temel bir altyapı sağlamaktadır. Projenin geri kalanındaki modüllerle entegrasyonu sağlanarak, dosya transferi uygulamasının güvenli bir şekilde çalışması hedeflenmektedir.

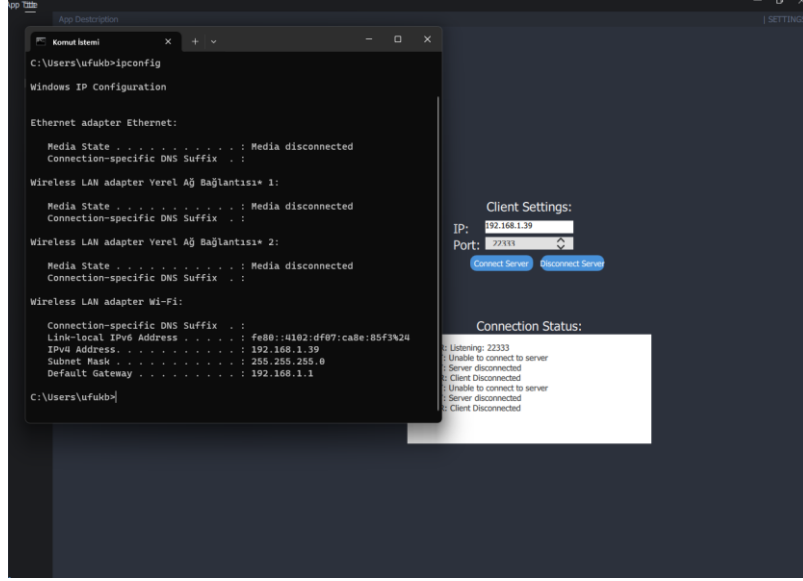
2. Güvenli Bağlantı Kurma Modülü:

Bu bölümde, projenin "Güvenli Bağlantı Kurma" modülü ile ilgili yazılmış C++ sınıflarının bir örneğini görmekteyiz. Bu modül, güvenli bir bağlantı kurma işlevselliğini sağlar.



Şekil 5 Modül 2'nin Genel Ekran Görüntüsü

Client Tarafı:



Şekil 6 Client Tarafından IP Adresi ile Oluşturulan SSL Sertifikası ile Bağlantı Sağlama

Client sınıfının constructor'ında, QSslSocket nesnesi oluşturulur ve bu nesnenin hazır okuma (readyRead), bağlantı kesilmesi (disconnected) ve SSL hata (sslErrors) sinyalleri ile ilgili slotlara bağlantılar yapılır.

CA sertifikası eklenerek sunucunun sertifikasının güvenilir olduğu doğrulanır.

```
Client::Client(QObject *parent) : QObject(parent)
{
    connect(&server, &QSslSocket::readyRead, this, &Client::rx);
    connect(&server, &QSslSocket::disconnected, this,
    &Client::serverDisconnect);
    connect(&server, SIGNAL(sslErrors(QList<QSslError>)), this,
    SLOT(sslErrors(QList<QSslError>)));
}
```

connectToServer fonksiyonu, belirtilen IP ve port'a güvenli bir bağlantı kurar.

waitForEncrypted fonksiyonu ile bağlantının şifrelenmesi beklenir. Bağlantı başarılıysa bilgi mesajları gönderilir, başarısızsa hata mesajları gönderilir.

```
void Client::connectToServer(QString ip, QString port)
{
    server.connectToHostEncrypted(ip, port.toUInt());
    if (server.waitForEncrypted(5000)) {
        qDebug() << "Authentication Succeeded";
        cout << "CLIENT: Authentication Succeeded" << endl;
        emit pushClientStatus("CLIENT: Authentication Succeeded");
    }
    else {
        qDebug() << "Unable to connect to server";
        cout << "CLIENT: Unable to connect to server" << endl;
        emit pushClientStatus("CLIENT: Unable to connect to server");
    }
}
```

sslErrors fonksiyonu, SSL hata durumları ile ilgilenir. Hatalar çıktıya ve loglamaya yazılır.

```
void Client::sslErrors(const QList<QSslError> &errors)
{
    foreach (const QSslError &error, errors) {
        qCritical() << error.errorString();
        cout << "CLIENT: " << error.errorString().toStdString() << endl;
    }
}
```

onPushSendFile, onPushSendHashFile ve onPushSendPublicKey fonksiyonları belirtilen dosya yollarındaki dosyaları okuyarak sunucuya gönderir. Örneğin, onPushSendFile fonksiyonu, belirtilen dosyayı okuyarak sunucuya gönderir.

```
void Client::onPushSendFile()
{
    QFile file("cipher_text.txt");
    file.open(QIODevice::ReadOnly);
    QByteArray data = file.readAll();
    server.write(data);
    server.waitForBytesWritten(1000);

    file.close();
    qInfo() << "File sent!";
    cout << "CLIENT: File sent!" << endl;
    emit pushSentFileStatus();
}
```

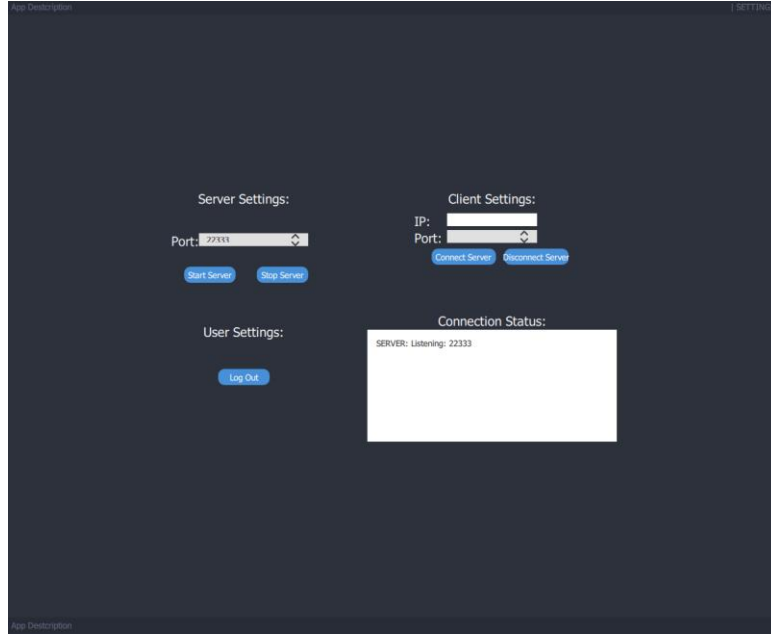
rx fonksiyonu, sunucudan gelen veriyi okuyarak dosyaya kaydeder ve bu durumu çıktıya yazar.

```
void Client::rx(void)
{
    file_counter++;
    QString message = server.readAll();
    qDebug() << message;

    ofstream myfile;
    myfile.open ("recieved" + std::to_string(file_counter) + ".txt");
    myfile << message.toStdString();
    myfile.close();
    qDebug() << "CLIENT: File received!";
    cout << "CLIENT: " << message.toStdString() << endl;
    emit pushInComingFileStatus();
}
```

Bu fonksiyonlar bir araya geldiğinde, Client sınıfı belirli bir sunucu ile güvenli bir şekilde iletişim kurabilen ve dosya transferi yapabilen bir istemci tarafı sağlar.

Server Tarafı:



Şekil 7 Server Dinleme Anındaki Ekran Çıktısı

Bu SslServer sınıfı, güvenli bir TCP sunucusu oluşturan ve bir istemci ile iletişim kurabilen bir QTcpServer alt sınıfını temsil eder. Constructor içinde, sunucunun RSA şifreleme anahtarını (key_ufulk_ev.key) ve sertifikasını (key_ufulk_ev.pem) dosyalardan okuyarak sınıfın üye değişkenlerine atar. Ayrıca, yeni bir bağlantı geldiğinde link fonksiyonuna bağlantı yapar.

```
void SslServer::setLocalKey(QString localKey)
{
    QFile keyFile(localKey);
    keyFile.open(QIODevice::ReadOnly);
    key = QSslKey(keyFile.readAll(), QSsl::Rsa);
    keyFile.close();
    qDebug("Server Key has been set.");
}

void SslServer::setLocalCertificate(QString localCertificate)
{
    QFile certFile(localCertificate);
    certFile.open(QIODevice::ReadOnly);
    cert = QSslCertificate(certFile.readAll());
    certFile.close();
    qDebug("Server Certificate has been set.");
}
```

incomingConnection Fonksiyonu yeni bir bağlantı geldiğinde çağrılır ve gelen bağlantı için yeni bir QSslSocket oluşturur. Bu socket'ı, sunucu sertifikası, özel anahtar ve kullanılacak protokol ile yapılandırır. startServerEncryption fonksiyonu ile şifreleme başlatılır ve bağlantı kabul edilir.

```
void SslServer::incomingConnection(qintptr socketDescriptor)
{
    QSslSocket *sslSocket = new QSslSocket(this);

    connect(sslSocket, SIGNAL(sslErrors(QList<QSslError>)), this,
        SLOT(sslErrors(QList<QSslError>)));
    sslSocket->setSocketDescriptor(socketDescriptor);
}
```



```

sslSocket->setLocalCertificate(cert);
sslSocket->setPrivateKey(key);
sslSocket->setProtocol(QSsl::TlsV1SslV3);

sslSocket->startServerEncryption();

addPendingConnection(sslSocket);
}

```

link Fonksiyonu yeni bir istemci bağlantısı olduğunda çağrılır ve bu bağlantı için rx fonksiyonuna ve bağlantının kesilmesi durumunda disconnected fonksiyonuna bağlantı yapar.

```

void SslServer::link()
{
    QTcpSocket *clientSocket;

    clientSocket = nextPendingConnection();
    emit pushServerStatus("SERVER: Client connected.");
    connect(clientSocket, &QTcpSocket::readyRead, this, &SslServer::rx);
    connect(clientSocket, &QTcpSocket::disconnected, this,
&SslServer::disconnected);
}

```

Rx Fonksiyonu client tarafından gönderilen veriyi okur ve bu veriyi bir dosyaya kaydeder. Ardından, istemciye bir mesaj gönderir ve bir dosyayı gönderir.

```

void SslServer::rx()
{
    file_counter++;
    QTcpSocket* clientSocket = qobject_cast<QTcpSocket*>(sender());
    QString message = clientSocket->readAll();
    qDebug() << message;

    ofstream myfile;
    myfile.open ("recieved" + std::to_string(file_counter) + ".txt");
    myfile << message.toStdString();
    myfile.close();
    emit pushInComingFileStatus();
    qDebug() << "File received!";

    QFile file("my_public_key.txt");
    file.open(QIODevice::ReadOnly);
    QByteArray data = file.readAll();
    clientSocket->write(data);
    clientSocket->waitForBytesWritten(1000);
    file.close();
    qDebug() << "SERVER: Public Key Sent.";
    cout << "SERVER: Public Key Sent." << endl;
    emit pushServerStatus("SERVER: Client connected.");
}

```

onPushStartServer ve onPushStopServer Fonksiyonları sunucuyu başlatmak ve durdurmak için kullanılır.

```

void SslServer::onPushStartServer(QString port)
{
    if (!listen(QHostAddress(QHostAddress::Any), port.toUInt())) {
        qDebug() << "SERVER: Unable to start the TCP server";
        emit pushServerStatus("SERVER: Unable to start the TCP server");
    }
}

```

```

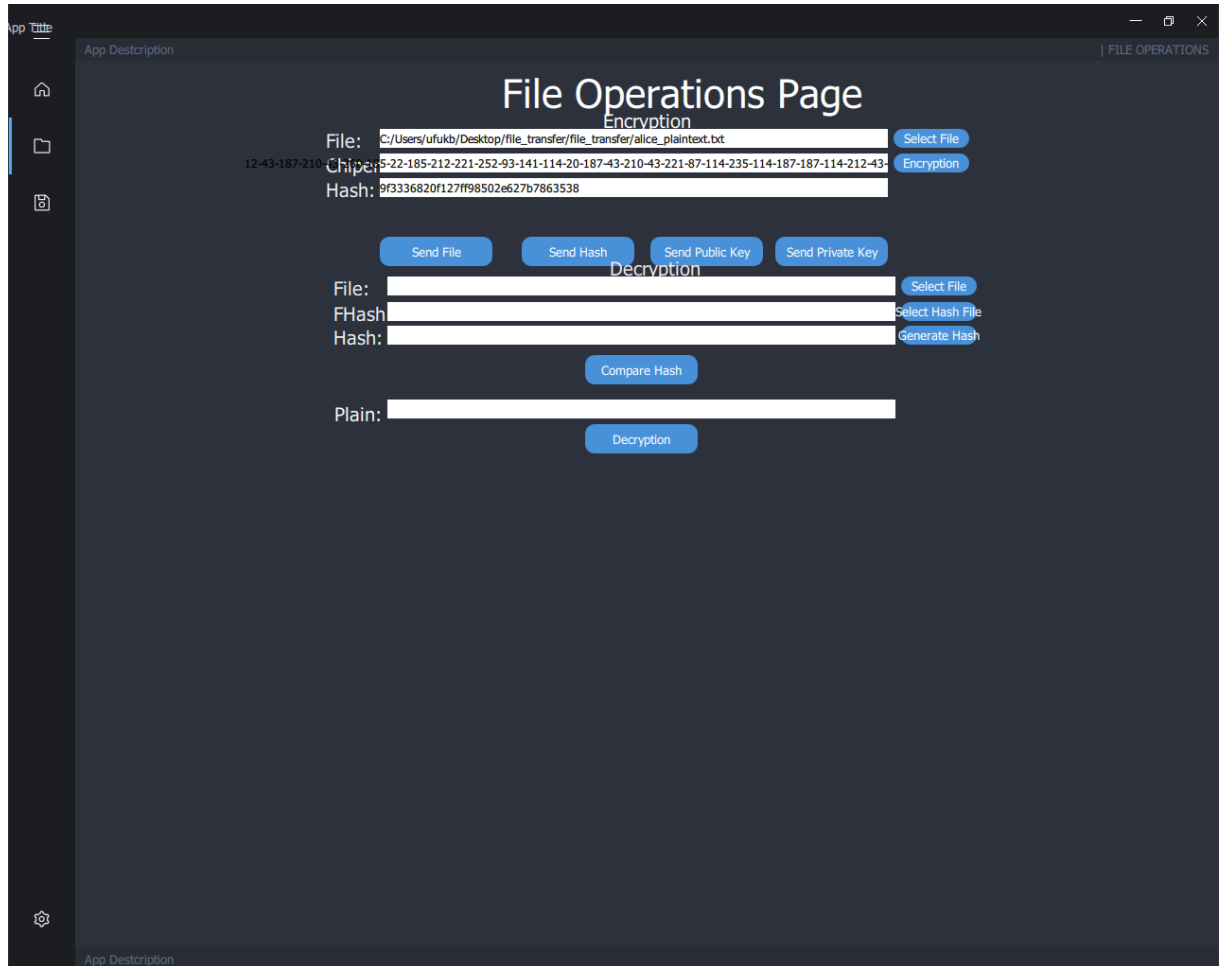
    }
    else {
        qDebug() << "SERVER: Listening: " + port;
        emit pushServerStatus("SERVER: Listening: " + port);
    }
}

void SslServer::onPushStopServer()
{
    if(isListening()) {
        close();
        qDebug() << "SERVER: Server is closing";
        emit pushServerStatus("SERVER: Server is closing");
    }
}
}

```

Bu sınıf, belirli bir portu dinleyerek ve güvenli bir şekilde bağlantılar kurarak istemcilerle iletişim kurabilir. Ayrıca, istemciden gelen veriyi alabilir ve istemciye dosya gönderebilir.

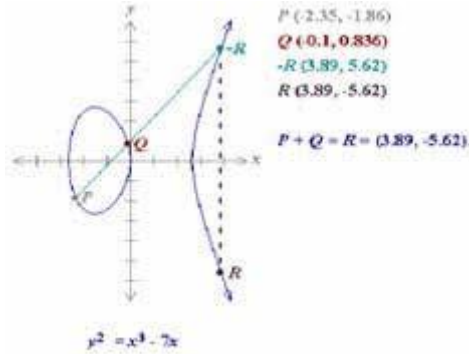
3. Dosya Şifreleme Modülü:



Şekil 8 Modül 3 Ekran Tasarımı

Dosya şifreleme modülü herhangi bir kütüphane kullanmadan c++ ile yazılmıştır. Asimetrik şifrelemelerden birisi olan Eliptik Eğri Şifreleme kullanılmıştır. FiniteFieldElement.hpp,

EllipticCurve.hpp ve Ecc.cpp, Ecc.h dosyalarından oluşmaktadır Eliptik Eğri şifrelemesi alt tarafta bu kodlar detaylı bir şekilde açıklanacaktır.



EllipticCurve sınıfı: Bu sınıf, belirli bir sonlu alan üzerindeki bir eliptik eğriyi temsil eder. Point adlı iç içe geçmiş bir sınıf kullanarak eğri üzerindeki noktaların işlemlerini gerçekleştirir.

Point sınıfı: Bu sınıf, bir eliptik eğri üzerindeki bir noktayı temsil eder. Noktaların toplama, çıkarma ve skalar çarpım gibi işlemleri gerçekleştirebilmek için çeşitli üye fonksiyonlar içerir.

EllipticCurve Sınıfı (EllipticCurve.hpp)

Genel Üyeler:

typedef FiniteFieldElement<P> ffe_t: Bu, P düzenindeki bir sonlu alan elemanını temsil etmek için bir tür tanımlar.

class Point: Bu iç içe geçmiş sınıf, eliptik eğri üzerindeki bir noktayı temsil eder. Nokta aritmetiği için çeşitli üye fonksiyonlar içerir ve ayrıca aşırı yüklenmiş operatörlerle nokta işlemlerini sağlar.

static Point ONE: Bu, grubun birim elemanını temsil eden bir static örnektir.

Genel Üye Fonksiyonlar:

CalculatePoints(): Eliptik eğri üzerindeki tüm noktaları hesaplar ve bir tabloya kaydeder.

operator[]: İndis numarasına göre eğri üzerindeki bir noktayı (grup elemanını) döndürür.

Size(): Eğrideki toplam nokta sayısını döndürür.

Degree(): Eliptik eğrinin matematiksel derecesini döndürür.

a(): Eğri denklemindeki a parametresini döndürür.

b(): Eğri denklemindeki b parametresini döndürür.

PrintTable(): Eğri üzerindeki tüm noktaları yazdırır.

Özel Üyeler:

m_table_: Eliptik eğri üzerindeki noktaların bir tablosunu tutar.

a_ ve b_: Eğri denklemindeki parametreleri saklar.

table_filled_ : Tablonun doldurulup doldurulmadığını belirtir.

FiniteFieldElement Sınıfı (FiniteFieldElement.hpp)

Bu sınıf, bir Galois alanındaki F_p 'nin bir elemanını temsil eder. Bu, eliptik eğrilerde kullanılan sonlu bir karakteristikli alan türünü ifade eder.

Sınıf, temel aritmetik işlemleri (toplama, çıkarma, çarpma, bölme) destekler ve aşırı yüklenmiş operatörlerle bu işlemleri gerçekleştirebilir.

Ayrıca, genişletilmiş Euclid algoritması gibi yardımcı işlevleri içerir. Bu, eliptik eğrilerde modüler ters alma işlemlerinde kullanılır.

```
int a = 10;
int b = 13;
//mpz_class
Ethereum_P("11579208923731619542357098500868790785326998466564056403945758400791312963
9935");
int P = 26363;
typedef EllipticCurve<21473> ec_t;
ec_t myEllipticCurve(a, b);

// Generate base point G
ec_t::Point G = myEllipticCurve[0];
while (G.y() == 0 || G.x() == 0 || G.Order() < 2) {
    int n = utils::irand(1, myEllipticCurve.Size());
    G = myEllipticCurve[n];
}

// Generate Alice's keys
int a_private = generatePrivateKey(myEllipticCurve);
EllipticCurve<21473>::Point Pa = generatePublicKey(a_private, myEllipticCurve);
writeKeysToFile("alice_keys.txt", a_private, Pa);
std::cout << "Alice Public " << Pa << std::endl;
std::cout << "Alice Private " << a_private << std::endl;

// Generate Bob's keys
int b_private = generatePrivateKey(myEllipticCurve);
EllipticCurve<21473>::Point Pb = generatePublicKey(b_private, myEllipticCurve);
writeKeysToFile("bob_keys.txt", b_private, Pb);
std::cout << "Bob Public " << Pb << std::endl;
std::cout << "Bob Private " << b_private << std::endl;

// Alice encrypts the message
std::string plaintext = readFile("alice_plaintext.txt");

EllipticCurve<21473>::Point Pk = a_private * Pa;
std::string ciphertext = encryptMessage(plaintext, Pk);

// Output ciphertext
writeFile("ciphertext.txt", "Ciphertext: " + ciphertext);
std::cout << "Encrypted message from Alice: " << ciphertext << std::endl;
//////////

//ÇÖZME
```

```

// Bob decrypts the message
Pk = a_private * Pa;
std::string decrypted_text = decryptMessage(ciphertext, Pk);

// Output decrypted text
writeFile("decrypted_message.txt", "Decrypted message by Bob: " + decrypted_text);
std::cout << "Decrypted message by Bob: " << decrypted_text << std::endl;

// Veritabanından alınan koordinatlar
int x ;
int y ;
x = 24101; y = 15145;
//Cryptography::EllipticCurve<263>::Point publicKeyFromDB =
convertDBCoordinatesToPoint(x_coord_from_db, y_coord_from_db, myEllipticCurve);
//std::cout << "Public Key from Database: " << publicKeyFromDB << std::endl;

//Cryptography::EllipticCurve<26363>::Point publicKeyFromDB;
//publicKeyFromDB= myEllipticCurve.convertDBCoordinatesToPoint(x, y);
// Sonucu kullanın

```

Yukarıda Elliptic Curve Algoritmasının deneme kodu yer almaktadır. Bu kod sırasıyla aşağıdaki işlemleri gerçekleştirmektedir:

Elliptic Curve Parametreleri Tanımlama:

a ve b sabitleri ile bir eliptik eğri tanımlanır.

P bir asal sayı ile belirlenen bir sonlu alanın büyüklüğünü temsil eder.

Elliptic Curve Nesnesi Oluşturma:

Tanımlanan parametrelerle bir EllipticCurve nesnesi oluşturulur.

Base Point Oluşturma:

Rastgele bir nokta seçilerek bu nokta G olarak atanır.

Bu noktanın x ve y koordinatları sıfır olmamalı ve noktanın grubundaki eleman sayısı 2'den büyük olmalıdır.

Alice ve Bob'un Anahtarlarını Oluşturma:

generatePrivateKey fonksiyonu kullanılarak rastgele özel anahtarlar oluşturulur.

generatePublicKey fonksiyonu ile her bir özel anahtara karşılık gelen genel anahtarlar hesaplanır.

Oluşturulan anahtarlar dosyalara yazılır.

Mesaj Şifreleme (Client):

readFile fonksiyonu ile "alice_plaintext.txt" dosyasındaki metin okunur.

encryptMessage fonksiyonu kullanılarak metin şifrelenir.

Şifreli metin "ciphertext.txt" dosyasına yazılır.

Mesaj Şifresini Çözme (Server):

decryptMessage fonksiyonu kullanılarak şifreli metin çözülür.

Çözülen metin "decrypted_message.txt" dosyasına yazılır.

Veritabanındaki Verinin Okunup Point Class Üyesi Haline Getirme:

x ve y koordinatları belirlenmiş bir nokta publicKeyFromDB ile oluşturulmaya çalışılır.

Bu adımda veritabanından alınan x ve y koordinatları kullanılarak bir nokta oluşturulmaya çalışılmış gibi görünüyor.

Sonuçları Yazdırma:

Oluşturulan anahtarlar ve şifreleme/çözme işlemlerinin sonuçları konsola yazdırılır.

hash birbirine uyduğunda ve eliptik eğri üzerinde kontrol edilen anahtarlar birbirine uyumlu ise şifre çözülür. Txt içerisindeki metin Plain olarak alt kısma yerleştirilir.

Private Key'in Mail ile Bildirilmesi:

Deşifreleme yapacak karşı kullanıcıya Private Key'in gönderilmesi için Qt Framework'ünde yazılmış olan open-source Smtplib-for-Qt adlı kütüphane kullanılmıştır. İlgili işleme dair kod bloğu aşağıdaki gibidir.

```
void ECC::onPushSendPrivateKey(QString email)
{

    EmailAddress sender("osmanaksoy53@gmail.com", "Osman Can AKSOY");
    message.setSender(sender);

    EmailAddress to(email, "");
    message.addRecipient(to);

    message.setSubject("Private key for project");

    text.setText(QString::number(m_privateKey));

    message.addPart(&text);

    Smtplib smtp("smtp.gmail.com", 465, Smtplib::SslConnection);

    smtp.connectToHost();
    if (!smtp.waitForReadyConnected()) {
        qDebug() << "Failed to connect to host!";
    }

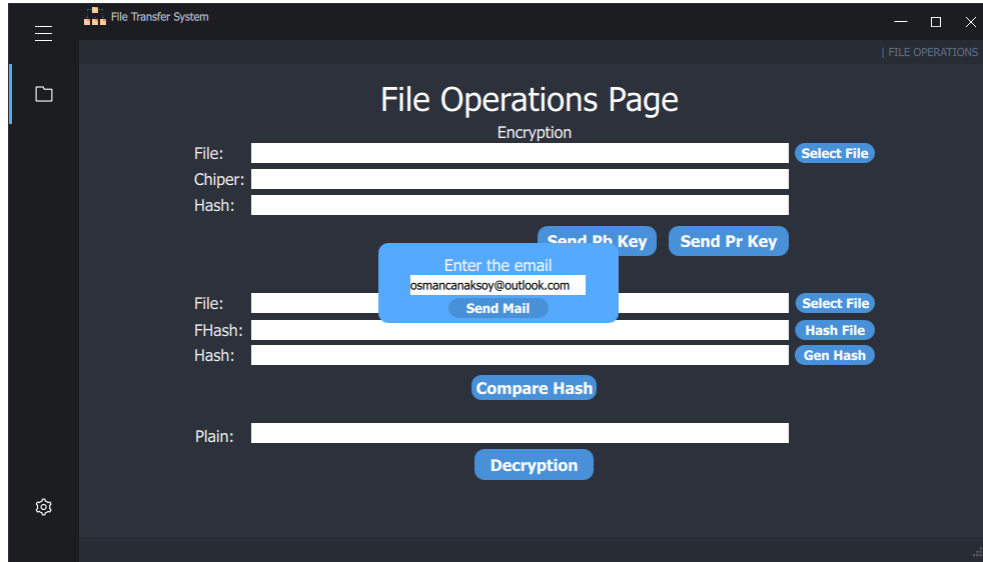
    smtp.login("osmanaksoy53@gmail.com", "upba dmid tngv dopl");
    if (!smtp.waitForAuthenticated()) {
        qDebug() << "Failed to login!";
    }

    smtp.sendMail(message);
    if (!smtp.waitForMailSent()) {
        qDebug() << "Failed to send mail!";
    }

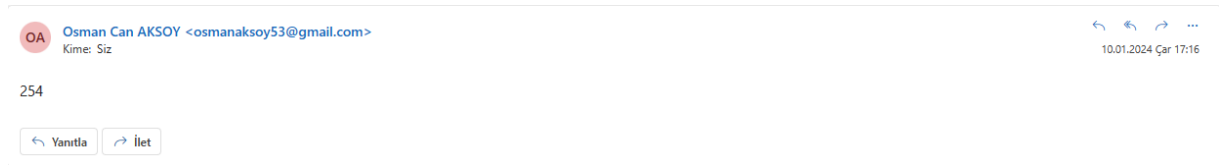
    smtp.quit();
}
```

1. **EmailAddress** sınıfından bir gönderen e-posta adresi oluşturuluyor ve mesaj bu gönderenle ilişkilendiriliyor.
2. E-posta alıcısı için bir **EmailAddress** oluşturuluyor.
3. Konu belirleniyor, burada "Private key for project" adlı mail konusu belirleniyor.
4. E-posta içeriğine Private Key değeri ekleniyor.

5. SMTP istemcisi (**SmtpClient**) oluşturuluyor ve Gmail sunucusuna bağlanmak için gerekli ayarlar yapılıyor.
6. Bağlantı oluşturuluyor ve gerekli kimlik doğrulama adımları gerçekleştiriliyor.
7. Oluşturulan e-posta mesajı SMTP istemcisine gönderiliyor.
8. E-posta gönderimi başarılıysa SMTP oturumu sonlandırılıyor.



Şekil 11 Private Key Değerinin Mail ile Gönderilmesi Ekran Görüntüsü



Şekil 12 Alınan Private Key Değerine Ait Mail Görüntüsü

4. Loglama ve Hata izleme Modülü:

Logger Sınıfı ve Static Değişkenler:

Logger sınıfı, loglama işlevselliğini sağlamak üzere tasarlanmıştır.

logFile: Log dosyasını temsil eden bir QFile nesnesi. Bu, log bilgilerinin yazılacağı dosyayı temsil eder.

isInit: Logger'ın başlatılıp başlatılmadığını kontrol eden bir bayrak. Birden çok başlatma girişimini önlemek için kullanılır.

contextNames: Log türlerine karşılık gelen metin isimlerini içeren bir QHash.

init Metodu: Logger'ı başlatmak için kullanılır.

isInit kontrol edilir ve Logger daha önce başlatılmışsa işlem yapılmaz.

Log dosyası (MyLog.log) oluşturulur ve dosya, metin modunda açılır (QIODevice::Text).

qInstallMessageHandler ile messageOutput fonksiyonu, Qt log mesajlarını ele almak üzere atanır.

Dosyanın içeriği temizlenir ve isInit bayrağı true olarak ayarlanır.

clean Metodu:

Logger'ı temizlemek için kullanılır.

logFile kontrol edilir ve varsa kapatılır, ardından nesne silinir.

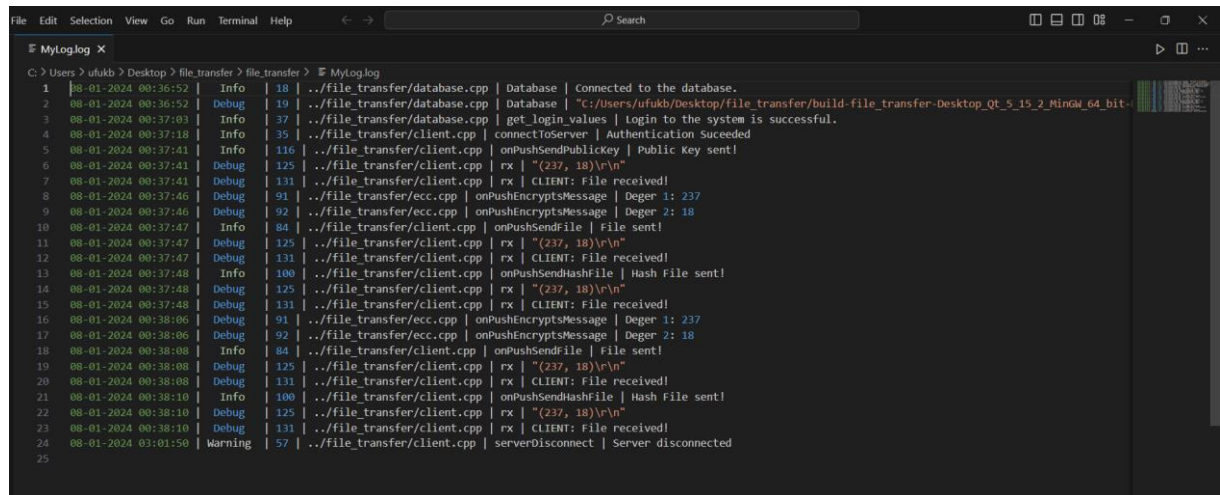
messageOutput Fonksiyonu:

Qt'nin log mesajlarını ele alacak olan özel bir mesaj çıkış işlevini temsil eder.

Her log girdisi, belirli bir formatta birleştirilir (tarih, tür, dosya, satır, fonksiyon, mesaj).

Bu bilgiler, log dosyasına yazılır (write), ardından dosya hemen temizlenir (flush).

Bu kod, uygulamanın çalışma zamanında oluşan logları bir dosyaya yazan bir loglama mekanizmasını implemente eder. init fonksiyonu bir kere çağrıldıktan sonra, messageOutput fonksiyonu Qt loglarını ele alır ve bu logları belirtilen dosyaya ekler. Log dosyası, MyLog.log adında bir dosya olarak aynı dizinde oluşturulur ve mevcut içeriği temizlenir. Log türleri, dosyada farklı sütunlarda gösterilir.



```
1 08-01-2024 00:36:52 | Info | 18 | ../file_transfer/database.cpp | Database | Connected to the database.
2 08-01-2024 00:36:52 | Debug | 19 | ../file_transfer/database.cpp | Database | "C:/Users/ufukb/Desktop/file_transfer/build-file_transfer-Desktop_Qt_5_15_2_MinGW_64_bit-
3 08-01-2024 00:37:03 | Info | 37 | ../file_transfer/database.cpp | get_login_values | Login to the system is successful.
4 08-01-2024 00:37:18 | Info | 35 | ../file_transfer/client.cpp | connectToServer | Authentication Succeeded
5 08-01-2024 00:37:41 | Info | 116 | ../file_transfer/client.cpp | onPushSendPublicKey | Public Key sent!
6 08-01-2024 00:37:41 | Debug | 125 | ../file_transfer/client.cpp | rx | "(237, 18)\r\n"
7 08-01-2024 00:37:41 | Debug | 131 | ../file_transfer/client.cpp | rx | CLIENT: File received!
8 08-01-2024 00:37:46 | Debug | 91 | ../file_transfer/ecc.cpp | onPushEncryptsMessage | Deger 1: 237
9 08-01-2024 00:37:46 | Debug | 92 | ../file_transfer/ecc.cpp | onPushEncryptsMessage | Deger 2: 18
10 08-01-2024 00:37:47 | Info | 84 | ../file_transfer/client.cpp | onPushSendFile | File sent!
11 08-01-2024 00:37:47 | Debug | 125 | ../file_transfer/client.cpp | rx | "(237, 18)\r\n"
12 08-01-2024 00:37:47 | Debug | 131 | ../file_transfer/client.cpp | rx | CLIENT: File received!
13 08-01-2024 00:37:48 | Info | 100 | ../file_transfer/client.cpp | onPushSendHashFile | Hash File sent!
14 08-01-2024 00:37:48 | Debug | 125 | ../file_transfer/client.cpp | rx | "(237, 18)\r\n"
15 08-01-2024 00:37:48 | Debug | 131 | ../file_transfer/client.cpp | rx | CLIENT: File received!
16 08-01-2024 00:38:06 | Debug | 91 | ../file_transfer/ecc.cpp | onPushEncryptsMessage | Deger 1: 237
17 08-01-2024 00:38:06 | Debug | 92 | ../file_transfer/ecc.cpp | onPushEncryptsMessage | Deger 2: 18
18 08-01-2024 00:38:08 | Info | 84 | ../file_transfer/client.cpp | onPushSendFile | File sent!
19 08-01-2024 00:38:08 | Debug | 125 | ../file_transfer/client.cpp | rx | "(237, 18)\r\n"
20 08-01-2024 00:38:08 | Debug | 131 | ../file_transfer/client.cpp | rx | CLIENT: File received!
21 08-01-2024 00:38:10 | Info | 100 | ../file_transfer/client.cpp | onPushSendHashFile | Hash File sent!
22 08-01-2024 00:38:10 | Debug | 125 | ../file_transfer/client.cpp | rx | "(237, 18)\r\n"
23 08-01-2024 00:38:10 | Debug | 131 | ../file_transfer/client.cpp | rx | CLIENT: File received!
24 08-01-2024 03:01:50 | Warning | 57 | ../file_transfer/client.cpp | serverDisconnect | Server disconnected
25
```

Yukarıda deneme aşamalarında alınmış olan Log çıktılarının bir görüntüsü yer almaktadır.