

Usługi i aplikacje Internetu rzeczy (PBL5)

Aplikacje dla systemu Android

Mykyta Vovk oraz Aleksander Pruszkowski

Instytut Telekomunikacji Politechniki Warszawskiej

Tworzenie klienta MQTTS (APP007)

MQTT Security

- **Nawiązanie bezpiecznego połączenia z brokerem MQTT za pomocą certyfikatów SSL:**
 - Klasa Certificates posiada statyczne zmienne String. Wartościami tych zmiennych są zawartości plików z certyfikatami oraz kluczem prywatnym (plik Certificates.java)

```
public class Certificates {...}
```

- Zawartość pliku z certyfikatem "ca.crt" została przypisana do zmiennej String

```
public final static String ca_Crt =
```

Słowo kluczowe **final** oznacza niezmiennosc deklarowanego elementu.

```
"-----BEGIN CERTIFICATE-----\n" +
```




```
"MIIeAzCCAuugAwIBAgIU\n" + ... +
```

```
"-----END CERTIFICATE-----\n";
```

Znak specjalny.
Znak nowej linii.

- Tak samo zostały przepisane zawartości plików „tester.crt” oraz „tester.key”

```
ca.crt
1 -----BEGIN CERTIFICATE-----
2 MIIEAzCCAuugAwIBAgIUBy1h1CGvdj4NhBXkZ/uLUZNLAWwDQYJKoZIhvcNAQEL
3 BQAwZAxhZCZAJBgNVBAYTAkdCMRcwFQYDVQQLIDAVbml0ZWQsS2luZ2RvbTEOMAwG
4 A1UEBwwFRG9yYnkvEjAqBgNVBAoMCU1vc3FlaXR0bzELMAkGA1UECwwQOExFjAU
5 BgNVBAMMDW1vc3FlaXR0by5vcmcxH2AdBgkqhkiG9w0BCQEWElJvZ2VzQG90Y2hv
6 by5vcmcwHhcNMjAwNjA5MTEwNjM5WmcNMzAwNjA3MTEwNjM5WjCBkDELMAkGA1UE
7 BhMCR0IxZmFzAVBgNVBAGMD1VuaXRlZCBLaW5nZG9tMQ4wDAYDVQQHDAVEZXXJieTES
8 MBAGA1UECgwJTW9zcXVpdHRvMQswCQYDVQQLDAJDQTEWMBQGA1UEAwwNbW9zcXVp
9 dHRvLm9yZzEfMB0GCSqGSIb3DQEJARYQcm9nZXJAYXRjaG9vLm9yZzCCASIwDQYJ
10 KoZIhvcNAQEBBQADggEPADCCAQoCggEBAME0HKmIzftOwkKLT3THHe+ObdizamPg
11 UZmD64Tf3zJdNeYGYn4CEXbyP6fy3tWc8S2boW6dzrH8SdFf9uo320GJA9B7U1FW
12 Te3xda/Lm3JffaHjkWw7jBwcauQZjpGINHapHRLpicZsquAthOgxW9SgDgY1GzEA
13 s06pkEFiMw+qDfLo/sxFKB6vQlFekMeCymjLCbNwPJyqyhFmPWwio/PDMruBTzPH
14 3cioBnrJWKKc3OjXdLGFJOfj7pP0j/dr2LH72eSvv3PQQF190CZPFhrCUCRHSSxo
15 E6yJG0dnz7f6PveLIB574kQORwt8ePn0ydrTClictikED3nHYHMOUCAwEAAANT
16 MFEEJG9VUDQ0RDVFBWU6zBUEBfCVRDwE5Y2LHk4NOVMBQGA1UEJTBkY2VzQG90Y2hv
```

 ca.crt
 tester.crt
 tester.key

MQTT Security

■ Interakcja z użytkownikiem – obsługa CheckBox MQTTS

- W celu wyboru typu połączenia z brokerem (tcp:// czy ssl://) został dodany element CheckBox do DashboardFragment oraz została dodana zmienna przechowująca aktualny typ połączenia

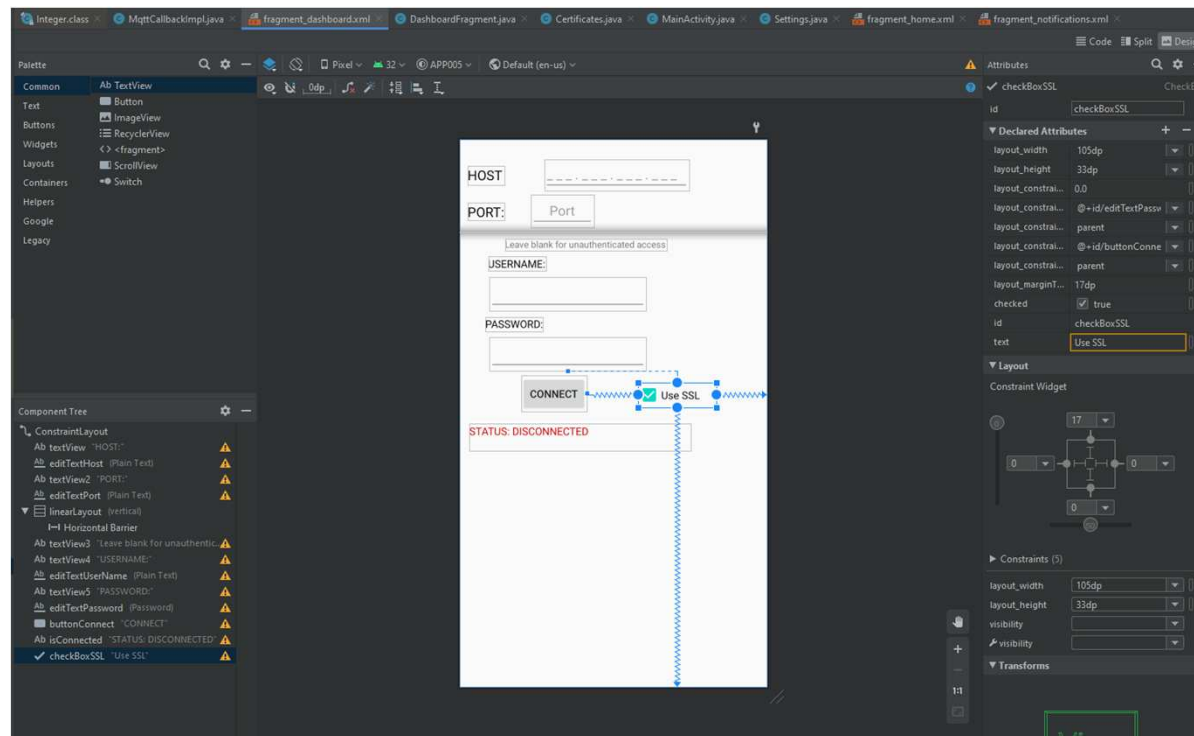
```
public static String conType = "ssl://";
```

- Deklaracja CheckBox

```
private CheckBox checkBoxSSL;
```

- Mapowanie elementu graficznego z obiektem java

```
checkBoxSSL = root.findViewById(R.id.checkBoxSSL);
```



■ Interakcja z użytkownikiem – obsługa CheckBox MQTTS

- Po naciśnięciu na klawisz Connect aplikacja sprawdza, jakiego protokołu ma użyć

```
SSLConnectionFactory socketFactory = null;
```

```
if (checkBoxSSL.isChecked()) {
```

```
    MqttCallbackImpl.conType="ssl://";
```

```
    try {
```

```
        socketFactory = MqttCallbackImpl.getSocketFactory("");
```

```
    } catch (Exception e) {
```

```
        e.printStackTrace();
```

```
    }
```

```
        mqttConnectOptions.setSocketFactory(socketFactory);
```

```
    } else
```

```
        MqttCallbackImpl.conType = "tcp://";
```

```
    ...
```

Jeżeli CheckBox jest zaznaczony, to tworzy się obiekt `SSLConnectionFactory` za pomocą metody `getSocketFactory()`

Dodanie dodatkowych opcji w przypadku wykorzystania SSL.
Ustawianie `SocketFactory`

■ Interakcja z użytkownikiem – obsługa CheckBox MQTTS

- Wywołanie metody *connect()*, używając typ połączenia

```
if (mqttCallback.connect(MqttCallbackImpl.conType + hostTxt, portNum, mqttConnectOptions) {  
    connectButton.setText("DISCONNECT");  
}
```

Ten parametr przechowuje obiekt Socket Factory



■ Interakcja z użytkownikiem – interakcja z SocketFactory

- W celu umożliwienia nawiązywania połączenia za pomocą SSL należy zaimportować poniższe biblioteki

```
dependencies {
```

```
    implementation 'org.bouncycastle:bcprov-jdk15on:1.70'
```

```
    implementation 'org.bouncycastle:bcpkix-jdk15on:1.70'
```

```
    ...
```

```
}
```

■ Interakcja z użytkownikiem – interakcja z SocketFactory

- Dodawanie funkcji *getSocketFactory()* do klasy *MqttCallbackImpl*, która zwraca obiekt typu *SSLSocketFactory*. *SSLSocketFactory* jest generowany na podstawie posiadanych certyfikatów, które wykorzystują się w celach autentyfikacji z serwerem

```
public static SSLSocketFactory getSocketFactory( final String password) throws Exception {
```

```
    Security.addProvider(new BouncyCastleProvider());
```

← Dodawanie nowego providera *

```
    X509Certificate caCert = null;
```

← Tworzenie obiektu CA certyfikatu

```
    InputStream stream = new ByteArrayInputStream(Certificates.ca_Crt.getBytes(StandardCharsets.UTF_8));
```

```
    BufferedInputStream bis = new BufferedInputStream(stream);
```

```
    CertificateFactory cf = CertificateFactory.getInstance("X.509");
```

```
    while (bis.available() > 0) {
```

```
        caCert = (X509Certificate) cf.generateCertificate(bis);
```

```
    }
```

```
    ...
```

```
}
```

* Patrz: <https://www.baeldung.com/java-bouncy-castle#cryptographic-operations>

■ Interakcja z użytkownikiem – interakcja z SocketFactory

■ Tworzenie obiektu klienckiego certyfikatu

```
public static SSLSocketFactory getSocketFactory( final String password) throws Exception {  
    ...  
    stream = new ByteArrayInputStream(Certificates.testers_Crt.getBytes(StandardCharsets.UTF_8));  
    bis = new BufferedInputStream(stream);  
    X509Certificate cert = null;  
    while (bis.available() > 0) {  
        cert = (X509Certificate) cf.generateCertificate(bis);  
    }  
    ...  
}
```

■ Interakcja z użytkownikiem – interakcja z SocketFactory

■ Tworzenie klucza prywatnego

```
public static SSLSocketFactory getSocketFactory( final String password) throws Exception {  
  
    ...  
  
    PEMParser pemParser = new PEMParser(new StringReader(Certificates.testKey));  
    Object object = pemParser.readObject();  
    PEMDecryptorProvider decProv = new JcePEMDecryptorProviderBuilder().build(password.toCharArray());  
    JcaPEMKeyConverter converter = new JcaPEMKeyConverter();    KeyPair key;  
    if (object instanceof PEMEncryptedKeyPair) {  
        key = converter.getKeyPair(((PEMEncryptedKeyPair) object).decryptKeyPair(decProv));  
    } else {  
        key = converter.getKeyPair((PEMKeyPair) object);  
    }  
    pemParser.close();  
    bis.close();  
    stream.close();  
    ...  
}
```

■ Interakcja z użytkownikiem – interakcja z SocketFacktory

```
public static SSLSocketFactory getSocketFactory( final String password) throws Exception {
```

```
...
```

```
KeyStore caKs = KeyStore.getInstance(KeyStore.getDefaultType());  
caKs.load(null, null);  
caKs.setCertificateEntry("ca-certificate", caCert);  
TrustManagerFactory tmf = TrustManagerFactory.getInstance("X509");  
tmf.init(caKs);
```

← Wykorzystanie CA certyfikatu
do autentyfikacji z serwerem

```
KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());  
ks.load(null, null);  
ks.setCertificateEntry("certificate", cert);  
ks.setKeyEntry("private-key", key.getPrivate(), password.toCharArray(),  
new java.security.cert.Certificate[] { cert });  
KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory .getDefaultAlgorithm());  
kmf.init(ks, password.toCharArray());
```

```
...
```

```
}
```

← Klucz kliencki oraz certyfikat są wysłane do serwera,
żeby on mógł autentyfikować klienta

■ Interakcja z użytkownikiem – interakcja z SocketFactory

- Inicjalizacja kontekstu za pomocą TrustManagerFactory oraz KeyManagerFactory i zwracanie obiektu typu SSLSocketFactory

```
public static SSLSocketFactory getSocketFactory( final String password) throws Exception {  
    ...  
    SSLContext context = SSLContext.getInstance("TLSv1.2");  
    context.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);  
    return context.getSocketFactory();  
}
```

Zadanie:

Utworzyć aplikację (opartą o kod APP006) dodając możliwość łączenia się z brokerem za pomocą protokołu TLS przez port 8884 z serwerem test.mosquitto.org

Dziękuję za uwagę