Poker Online

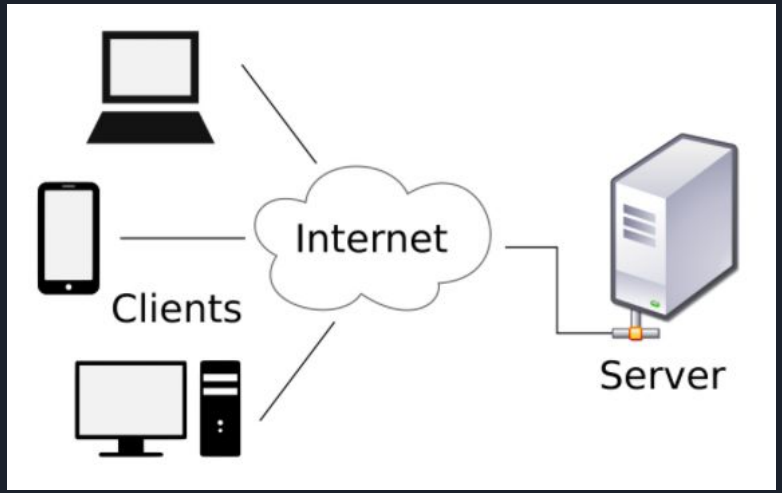
Wykonawcy:

Christian Białek - cbialek@student.agh.edu.pl

Mateusz Zdyrski - zdyrski@student.agh.edu.pl

Plan prezentacji

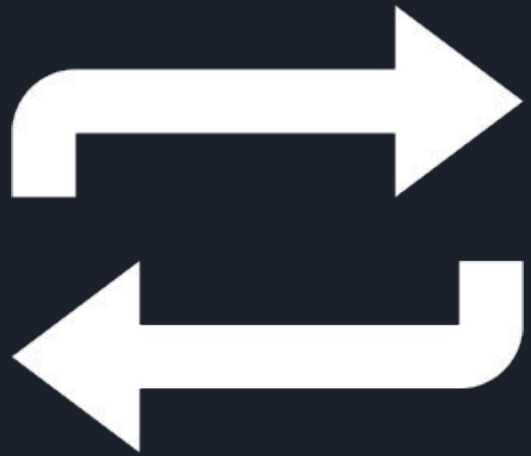
1. Demonstracja naszego projektu
2. Programowanie socketowe
3. Serializacja
4. Bezpieczeństwo
5. JavaFX
6. Wielowątkowość - JavaFX



1. Demonstracja projektu

Nasz projekt korzysta z architektury klient-serwer, a tłem całego projektu jest w pełni zaimplementowana gra karciana Poker.

- serwer nasłuchuje na porcie 5000 czekając na połączenie
- klient łączy się z serwerem --> start nowego wątku
- po zalogowaniu się 3 klientów, zaczyna się gra
- następuje ciągła wymiana informacji między klientem a serwerem

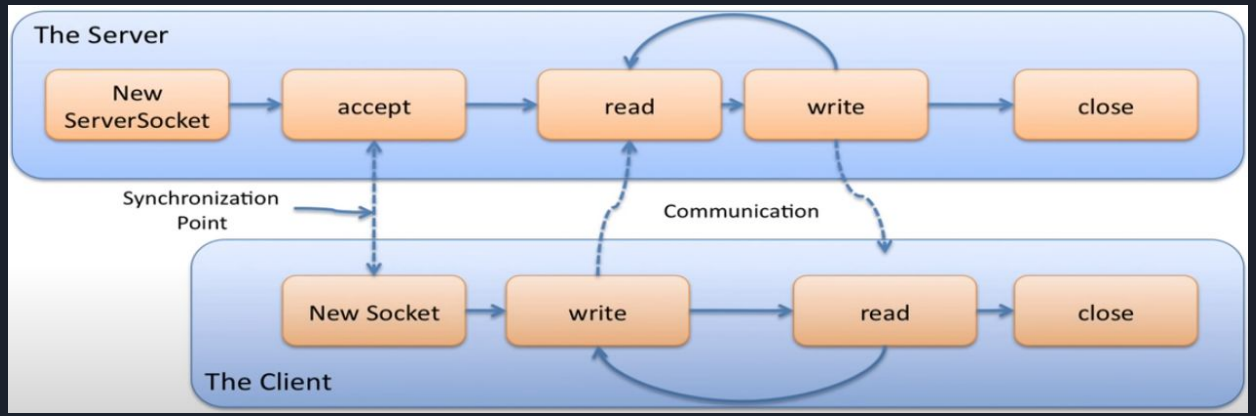


2. Programowanie socketowe

Zacznijmy od tego czym w ogóle jest ten socket?

Socket jest to pewien mechanizm umożliwiający otwarcie kanału komunikacji pomiędzy hostami. Każde gniazdo posiada adres IP oraz numer portu i może przesyłać m.in pakiety:

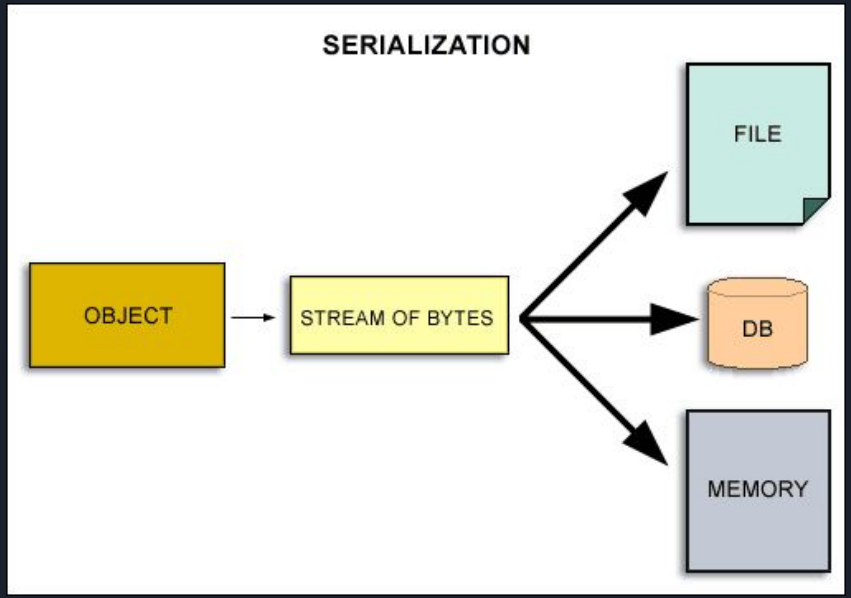
- datagramy (pakiety UDP)
- strumienie (pakiety TCP)



3. Interfejs java.io.Serializable

Serializacja służy do konwersji stanu obiektu w strumień(sekwencję) bajtów, które mogą być zapisane w bazy danych lub przesłane przez internet i następnie wiernie odtworzone.

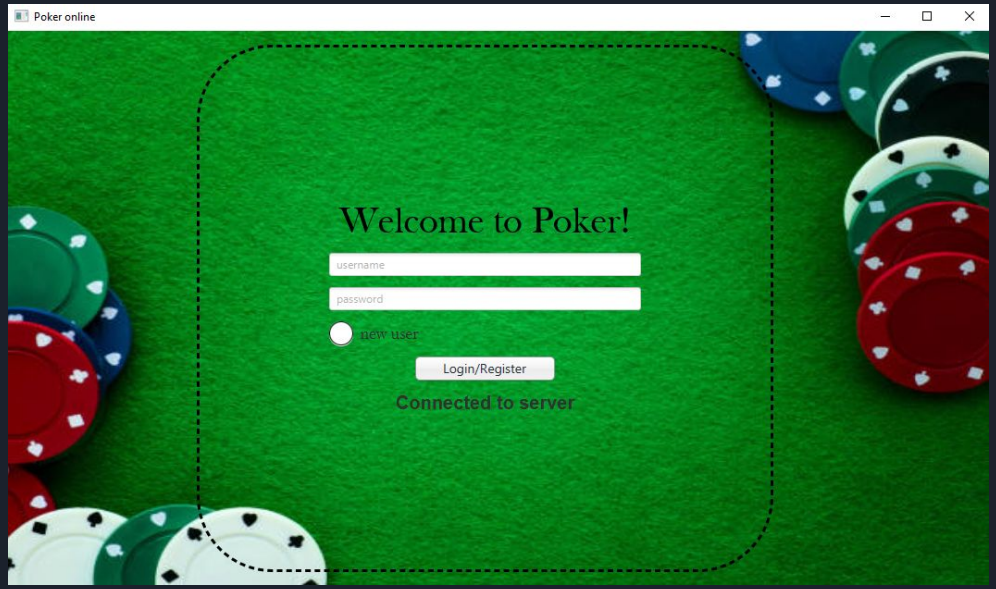
Odpowiednie klasy wyznaczone do serializacji muszą zaimplementować tzw. marker interface.




```
public class Message implements Serializable
```

4. Bezpieczeństwo

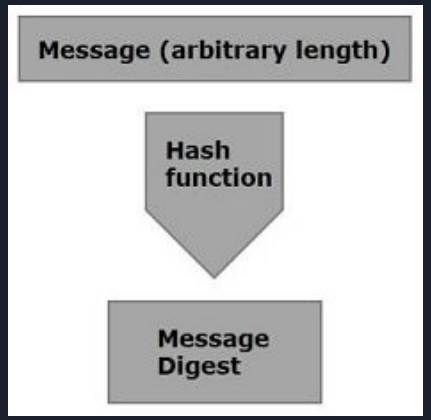
Czyli jak bezpiecznie przechowywać hasła użytkowników w bazie danych?



Algorytmy skrótu SHA w Javie

SHA-256 – ang. Secure Hash Algorithm 256-bit (256-bitowy bezpieczny algorytm haszujący). Kryptograficzna funkcja skrótu, czyli algorytm używany do ochrony danych.

MessageDigest class -> provides applications the functionality of a message digest algorithm, such as SHA-1 or SHA-256.



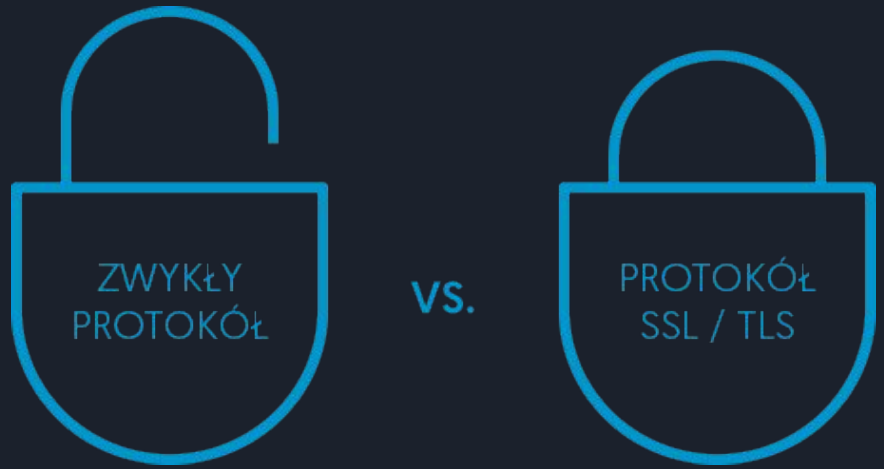

```java
private String generateHash(String password, String algorithm) {
    String hashValue = "";

    try {
        MessageDigest messageDigest = MessageDigest.getInstance(algorithm);
        messageDigest.reset();
        byte[] digest = messageDigest.digest(password.getBytes());
        hashValue = bytesToStringHex(digest);
    } catch (NoSuchAlgorithmException e) {
        System.out.println("error:hash: " + e.getStackTrace() + " " + e.getCause());
    }
    return hashValue;
}
```

Certyfikat SSL/TLS

SSL - Secure Sockets Layer, TLS - Transport Layer Security

- Bezpieczne przesyłanie wiadomości
- Zasada działania



5. Czym jest JavaFX

Java FX jest domyślną biblioteką definiowania graficznego interfejsu użytkownika. Zastąpiła ona starsze biblioteki takie jak Swing i AWT.

Istnieją dwa podstawowe sposoby tworzenia GUI wykorzystującego JavaFX:

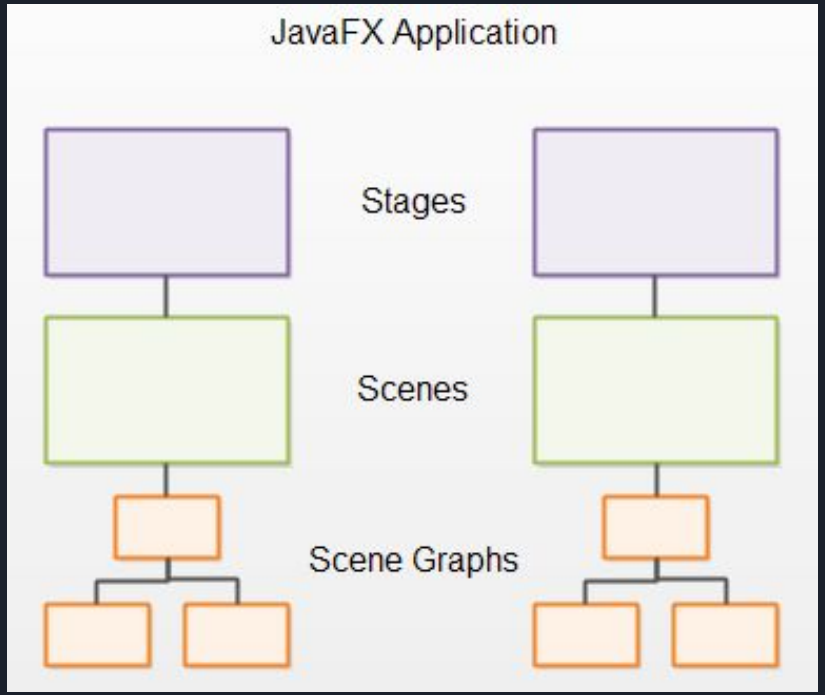
- z użyciem „czystej” Javy (jak w przypadku naszego projektu),
- z użyciem plików FXML (udostępnia do tego użyteczne narzędzie o nazwie Scene Builder).

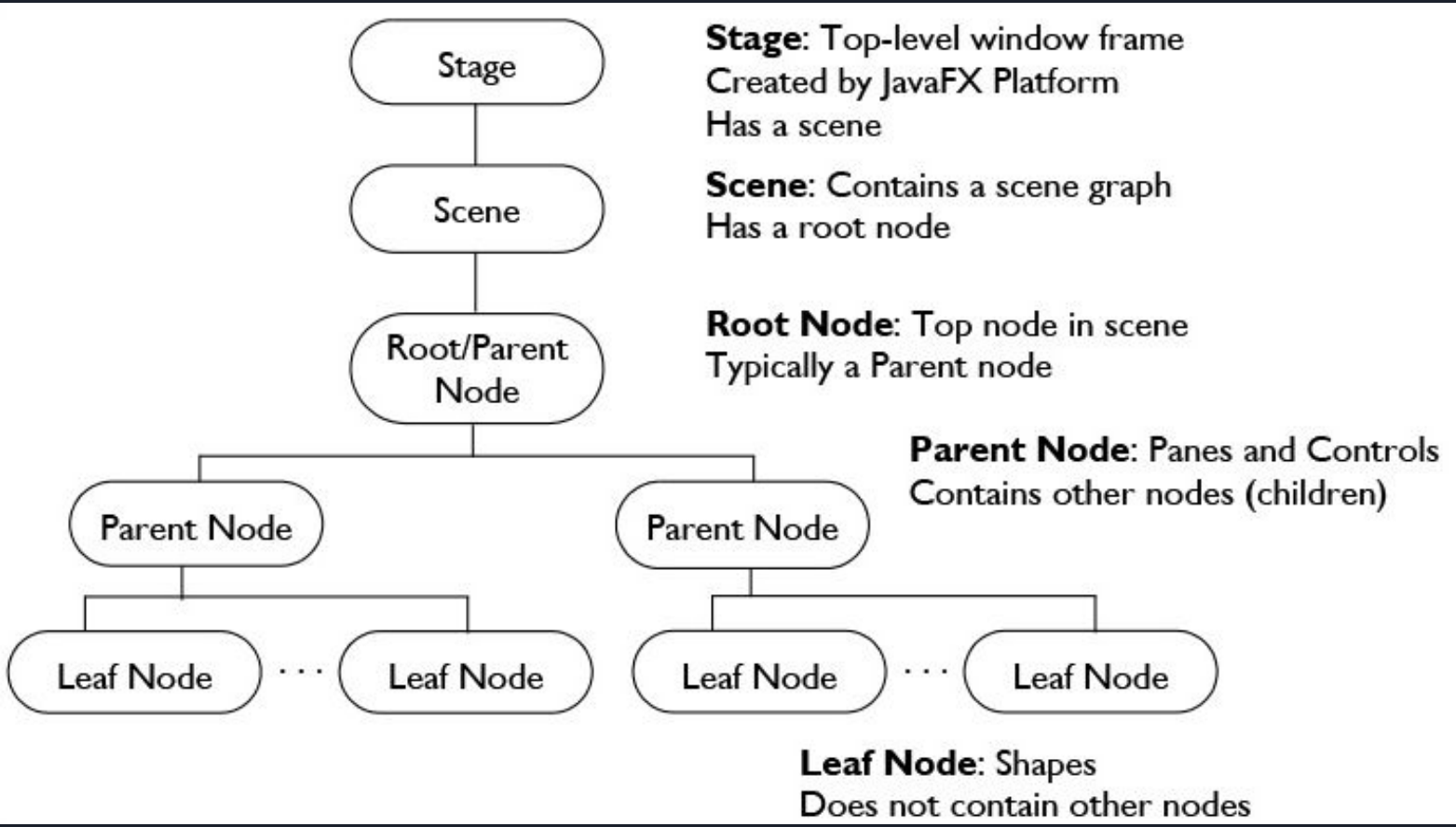

```
.button {
    -fx-background-color:
            rgba(0,0,0,0.08),
            linear-gradient(#9a9a9a, #909090),
            linear-gradient(white 0%, #f3f3f3 50%, #ececec 51%, #f2f2f2 100%);
    -fx-background-insets: 0 0 -1 0,0,1;
    -fx-background-radius: 5,5,4;
    -fx-padding: 3 30 3 30;
    -fx-text-fill: #242d35;
    -fx-font-size: 14px;
}

.button:hover {
    -fx-background-color: #555555;
    -fx-text-fill: white;
}
```

JavaFX - charakter hierarchiczny

- Podstawowym elementem aplikacji graficznej w JavaFX jest stage, która jest „kontenerem” dla wszystkich elementów znajdujących się w oknie aplikacji
- Stage zawiera scene, która z kolei zawiera całe drzewo składające się z węzłów.
- Obiekt root jest przykładem panelu – obiektu służącego do rozmieszczania innych obiektów w aplikacji.



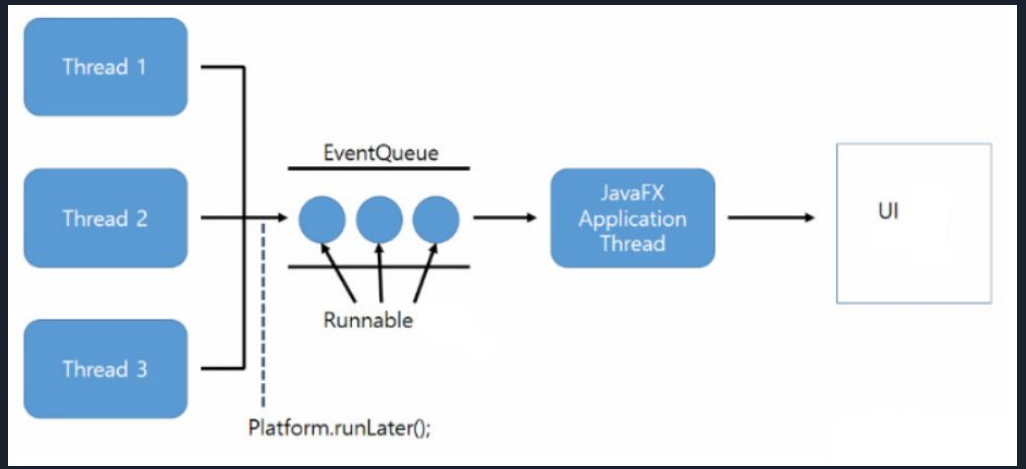

Stage
Scene
Root/Parent Node
Parent Node
Parent Node
Leaf Node
Leaf Node
Leaf Node
Leaf Node
Stage: Top-level window frame
Created by JavaFX Platform
Has a scene
Scene: Contains a scene graph
Has a root node
Root Node: Top node in scene
Typically a Parent node
Parent Node: Panes and Controls
Contains other nodes (children)
Leaf Node: Shapes
Does not contain other nodes

6. Wielowątkowość w JavaFX - Platform.runLater()

JavaFX używa modelu jednowątkowego(single-threaded rendering design).

- tylko jeden wątek ma niejako dostęp do wyświetlanej sceny
- JavaFX application thread --> możliwość zmiany JavaFX Scene Graph.

JavaFX posiada klasę Platform zawierającą metodę runLater() - **rozwiązanie naszego problemu**



Platform.runLater() - rozwiązanie naszego problemu

```
public void displayHand(String[] handCards){
    Platform.runLater(() -> {
        for(int i = 0; i < handCards.length; i++){
            Node handChange = myHand.getChildren().get(i);
            if (handChange instanceof ImageView){
                ((ImageView) handChange).setImage(new Image("images/"+handCards[i]+".png"));
            }
        }
        GridPane.setHalignment(myHand, HPos.CENTER);
    });
}
```

```
public void actualizePoolMoney(int i){
    Platform.runLater(() -> {
        String money = String.valueOf(i);
        pool.setText("Pool: " + money + " $");
    });
}
```

```
public void resetCards(){
    Platform.runLater(() -> {
        for(int i = 0; i < 5; i++){
            Node cardReset = crHand.getChildren().get(i);
            if (cardReset instanceof ImageView){
                ((ImageView) cardReset).setImage(null);
            }
        }
        for(int i = 0; i < 2; i++){
            Node cardReset = myHand.getChildren().get(i);
            if (cardReset instanceof ImageView){
                ((ImageView) cardReset).setImage(null);
            }
        }
    });
}
```

Zadanie 1 - Server-Client

W metodzie main() klienta i serwera użyć odpowiednich gotowych metod tak, aby:

- nawiązać połączenie
- klient wysłał wiadomość message i odczytał wiadomość od serwera
- serwer odebrał wiadomość i wysłał swoją

Zadanie 2 - przesyłanie obiektów.

Podobnie jak w zadaniu 1, w metodzie main() klienta i serwera użyć odpowiednich gotowych metod tak, aby:

- nawiązać połączenie
- klient wysłał wiadomość message i odczytał wiadomość od serwera
- serwer odebrał wiadomość i wysłał swoją

tylko tym razem zamiast String'a przesyłamy obiekt klasy Message2.

Zadanie 3 - wielowątkowość

Server3 dla każdego połączonego klienta startuje wątek ClientHandler do obsługi tego klienta i dodaje go do arrayListy clientHandlers. ClientHandler ciągle nasłuchuje na wiadomość od swojego klienta i jeśli jakąś dostanie rozsyła ją do pozostałych klientów.

W klasie ClientHandler uzupełnić metodę sendToAll() tak, aby każdy element arrayListy clientHandlers wysłał podany jako argument obiekt message. Zwróć uwagę, że element, który odebrał wiadomość nie powinien tego robić.

Bibliografia

- https://javastart.pl/b/java/dlaczego-javafx-jest-lepsza-od-swinga/
- https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm
- http://tutorials.jenkov.com/javafx/concurrency.html
- https://www.speedcheck.org/pl/wiki/socket/
- https://sekurak.pl/tworzenie-narzedzi-sieciowych-w-pythonie-z-uzyciem-socketow/
- http://architektura-oprogramowania.blogspot.com/p/architektura-klient-serwer.html
- https://vavatech.pl/technologie/architektura/Klient-Server
- https://www.informit.com/articles/article.aspx?p=2251210&seqNum=2
- http://tutorials.jenkov.com/javafx/overview.html
- https://docs.oracle.com/javase/9/docs/api/java/security/MessageDigest.html
- https://slideplayer.pl/slide/440309/
- https://certyfikatyssl.pl/news/protokol-ssl-a-tls.html
- https://www.ssl.certum.pl/cert/certy_informacje_co_to_jest_certyfikat_ssl/

Dziękujemy za uwagę :)