Wymagania dla biblioteki do komunikacji z terminalem płatniczym w zakresie obsługi płatności

Spis treści

Definicje i akronimy

Nazwa	Objaśnienie
Kierowca	System informatyczny do obsługi procesu związanego z wydawaniem uprawnień do kierowania pojazdami i zamawiania personalizowanych praw jazdy.
Pojazd	System informatyczny do obsługi procesu rejestracji pojazdów i obsługi zarejestrowanych pojazdów i zamawiania personalizowanych dowodów.
WORD	System teleinformatyczny przeznaczony dla WORD i umożliwiający obsługę PKK, przeprowadzenie oraz obsługę egzaminu państwowego na prawo jazdy i pozwolenia z wykorzystaniem komputerowego urządzenia egzaminacyjnego i zapewniający realizację zadań ustawowych przez WORD.
PWPW	Polska Wytwórnia Papierów Wartościowych S.A
Proxy Terminal Connect	Biblioteka dostarczona przez dostawcę płatności w zakresie zapewnienia komunikacji pomiędzy komponentami po stronie SI Pojazd, SI Kierowca, SI WORD w celu wykonywania płatności z tych aplikacji z wykorzystaniem terminala dostarczonego przez dostawcę.
Dostawca	Operator płatności dostarczający terminale z biblioteką Proxy Terminal Connect
Aplikacje PWPW	Aplikacje w wydziałach komunikacji SI Pojazd, SI Kierowca, oraz w WORD SI Word.

1. Wprowadzenie

W ramach poniższego dokumentu zostały opisane wymagania do komponentu/biblioteki do komunikacji z terminalem, która zostanie dostarczona przez dostawcę razem z terminalami w zakresie obsługi płatności terminalowych. Dostarczona biblioteka będzie spełniała wymagania opisane w poniższym dokumencie. Zakładamy, że dostarczone terminale oraz biblioteka będzie umożliwiała wykonywanie płatności jednocześnie na kilka kont urzędów, w zależności od rodzaju opłaty.

2. Architektura logiczna

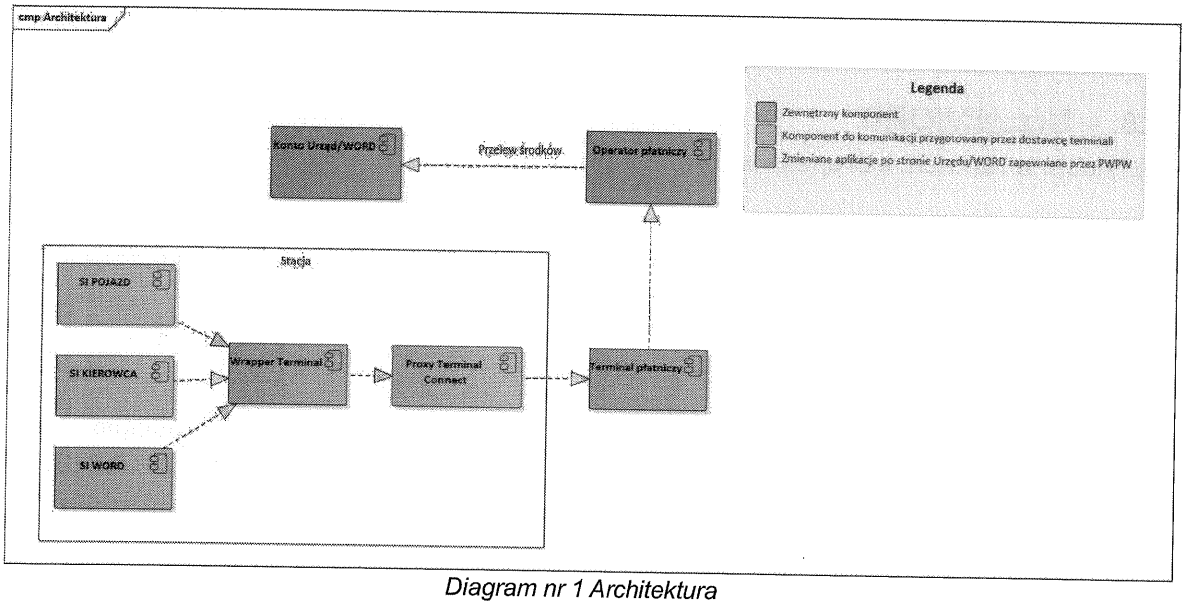

Diagram nr 1 Architektura

2.1. - SI POJAZD

System informatyczny w wydziale komunikacji umożliwiający rejestracje pojazdów i zamawianie dokumentów.

2.2. - SI KIEROWCA

System informatyczny w wydziale komunikacji umożliwiający obsługę wniosku o uzyskanie uprawnień oraz zamawianie i wydawanie dokumentów.

2.3. - SI WORD

System informatyczny w WORD umożliwiający obsługę rezerwacji i wykonywania egzaminów.

2.4. - Operator płatniczy

Operator dostarczający terminale i obsługujący płatności terminalowe.

2.5. - Wrapper Terminal

Komponent po stronie aplikacji Pojazd, Kierowca, WORD przeznaczony do komunikacji z biblioteką do terminala, opisaną w tym dokumencie jako Proxy Terminal Connect.

2.6. - Proxy Terminal Connect

Biblioteka przygotowana przez dostawcę terminali, która udostępnia metody opisane w ramach tego dokumentu.

2.7. - Terminal płatniczy

Urządzenie płatnicze umożliwiające realizacje płatności bezgotówkowych, które można podłączyć do sieci LAN oraz posiadający dodatkowy moduł GPRS (karta SIM) umożliwiający komunikację z bramką sieciową operatora płatniczego.

2.8. – Urząd/WORD

Instytucja w ramach, której wykonywane są płatności.

3. Założenia do biblioteki

3.1. Komunikacja

Standardowo terminal będzie podłączony do lokalnej sieci urzędu i osiągalny pod konkretnym adresem IP. Na bibliotece komunikacyjnej będzie spoczywać odpowiedzialność za weryfikację czy terminal jest osiągalny pod danym adresem oraz nawiązanie z nim połączenia. Ze względu na podłączenie terminala do sieci lokalnej zaleca się żeby terminal posiadał niezależny moduł GPRS (kartę SIM) do komunikacji z systemem operatora oraz w zakresie wykonywanych płatności. Dodatkowo dostawca zapewnia odpowiednie zabezpieczenia dla tego modułu GPRS.

3.2. Wymagania dotyczące implementacji interfejsu

1. **Platforma i środowisko uruchomieniowe**
   a. Biblioteka będzie używana w środowisku **.NET Framework 4.8**.
   b. Sama biblioteka może być napisana z wykorzystaniem platformy **.NET Framework 4.8** lub w języku **C/C++**. W przypadku implementacji w języku C/C++ należy zastosować mechanizm **P/Invoke (Platform Invocation Services)** do obsługi interfejsu.

2. **Architektura docelowa**
   a. Obsługiwane architektury to **x64** oraz **Any CPU** z wyłączonym wsparciem

dla trybu 32-bitowego (32-bit preferred = false).

3. **Statyczne linkowanie zależności**

    a. Wszystkie wymagane biblioteki zewnętrzne muszą być dołączone do głównej biblioteki w procesie budowania. Dzięki temu końcowy artefakt będzie samowystarczalny i niezależny od środowiska docelowego.

4. **Podpis cyfrowy**
    a. Biblioteka nie musi być podpisana cyfrowo.

5. **Sposób dostarczenia biblioteki**
    a. Biblioteka powinna być dostarczona w postaci odpowiednich plików, które można bezpośrednio użyć w projekcie.
    b. Dopuszcza się dostarczenie jej w formie instalatora, który rozpakowuje pliki do wybranego katalogu.

3.3. Interfejs i typy ogólne

W ramach poniższej części dokumentu został przedstawiony interfejs z odpowiednimi metodami biblioteki, które będą umożliwiały zrealizowanie płatności przez Systemy PWPW z wykorzystaniem Proxy Terminal Connect terminalowych płatności. Udostępnione metody:

- `ConnectAsync`
- `DisconnectAsync`
- `ProcessPaymentAsync`

Szczegółowo zostały przedstawione poniżej.
Zadaniem interfejsu komunikacyjnego/bazowego jest pokrycie funkcjonalności dotyczącej nawiązania połączenia z terminalem, przeprowadzania płatności i zarzadzania nią oraz rozłączenia z terminalem.

```
/// <summary>
/// Interfejs umożliwiający komunikację z terminalem płatniczym.
/// </summary> public interface
IPaymentTerminalService {
    /// <summary>
    /// Nawiązuje połączenie z terminalem płatniczym.
/// </summary>
    /// <param name="ipAddress">Adres IP terminala.</param>
    /// <param name="port">Port, na którym terminal nasłuchuje.</param>
```

```
    /// <returns>True, jeśli połączenie zostało nawiązane pomyślnie, false w
przeciwnym razie.</returns>
    Task<bool> ConnectAsync(string ipAddress, int
port);
    /// <summary>
    /// Zamyka połączenie z terminalem płatniczym.
    /// </summary>
    /// <returns>True, jeśli połączenie zostało prawidłowo zakończone.</returns>
    Task<bool> DisconnectAsync();

    /// <summary>
    /// Inicjuje płatność na terminalu. Obsługuje anulowanie płatności za
pomocą tokena anulowania.     /// </summary>
    /// <param name="paymentDetails">Lista szczegółów płatności dla różnych kont
Merchantów.</param>
    /// <param name="cancellationToken">Token anulowania.</param>
    /// <returns>Wynik płatności.</returns>
    Task<PaymentResult> ProcessPaymentAsync(List<PaymentDetail>
paymentDetails,
CancellationToken cancellationToken);
    /// <summary>
    /// Anuluje bieżącą płatność.
    /// </summary>
    /// <returns>True, jeśli anulowanie było pomyślnie, false w przeciwnym
razie.</returns>
    bool CancelPayment();

    /// <summary>
    /// Sprawdza, czy terminal jest gotowy do realizacji transakcji.
    /// </summary>
    /// <returns>True, jeśli terminal jest gotowy, false w
przeciwnym razie.</returns>     bool IsReady();
}
```

Interfejs *IPaymentTerminalService* zapewnia wszystkie niezbędne metody do zarządzania połączeniem z terminalem płatniczym oraz obsługi płatności. Dzięki tym metodom aplikacja może nawiązać połączenie, rozpocząć płatność, anulować ją w razie potrzeby oraz sprawdzać, czy terminal jest gotowy do obsługi kolejnych transakcji.

Oprócz typu prostego reprezentującego stan oraz interfejsu bazowego wskazane jest używanie klasy ogólnej *PaymentResult* do reprezentowania wyników transakcji płatniczej, zawierających szczegółowe informacje na temat stanu płatności po jej zakończeniu. Zawiera informacje takie jak identyfikator transakcji, status płatności, komunikat związany z transakcją oraz kwotę zapłaconą. Jest to kluczowy obiekt w systemie płatności, który umożliwia aplikacji zrozumienie czy transakcja przebiegła pomyślnie, czy wystąpiły jakiekolwiek problemy.

Jej reprezentacja znajduje się poniżej:

```
/// <summary>
/// Wynik transakcji płatniczej.
/// </summary> public
class PaymentResult
{
    /// <summary>
    /// Identyfikator transakcji.
    /// </summary>     public string
TransactionId { get; set; }

    /// <summary>
    /// Status płatności.     ///
</summary>     public PaymentState Status
{ get; set; }
    /// <summary>
    /// Wiadomość dotyczącą statusu płatności.
    /// </summary>     public string
Message { get; set; }
    /// <summary>
    /// Kwota transakcji.
/// </summary>
    public decimal Amount { get; set;
}
    /// <summary>
    /// Typ płatności (Card lub Blik).
    /// </summary>
    public PaymentType PaymentType { get; set; }
}
```

Interfejs zawiera również klasę *PaymentDetail* wykorzystywaną do opisania transakcji płatniczej. Zawiera kluczowe informacje o płatności, takie jak kwota, waluta oraz konto Merchanta, na które zostaną przelane środki. Dzięki tej klasie możliwe jest precyzyjne określenie danych potrzebnych do realizacji płatności, co zapewnia prawidłowy przebieg operacji płatniczej.

```
/// <summary>
/// Reprezentuje szczegóły jednej płatności.
/// </summary> public
class PaymentDetail
{
    /// <summary>
    /// Kwota transakcji.      ///
</summary>      public decimal Amount
{ get; set; }

    /// <summary>
    /// Waluta transakcji (np. "PLN").
    /// </summary>      public string
Currency { get; set; }
    /// <summary>
    /// Konto Merchanta, na które trafi płatność.
    /// </summary>      public string
MerchantAccount { get; set; }
}
```

PaymentType to enum, który reprezentuje metody płatności, które użytkownik może wybrać w terminalu płatniczym. Terminal po interakcji z użytkownikiem zwraca odpowiedni typ płatności w obiekcie *PaymentResult*, który wskazuje, czy transakcja została dokonana przy użyciu karty płatniczej czy za pomocą kodu BLIK.

```
/// <summary>
/// Typy płatności obsługiwane przez terminal.
/// </summary> public enum PaymentType
{
    /// <summary>      /// Płatność kartą.
    /// </summary>
    Card,

    /// <summary>
```

```
    /// Płatność kodem BLIK.
    /// </summary>
    Blik
}
```

W ramach interfejsu przewiduje się typ prosty *PaymentState* (dziedziczący po .NET'owym typie *int*), który reprezentuje ostateczny status płatności po jej przetworzeniu przez terminal płatniczy. Enum ten pozwala określić, czy transakcja została zakończona pomyślnie, anulowana, czy zakończyła się niepowodzeniem. Enum zawiera wartości takie jak:

Pending: Płatność czeka na rozpoczęcie przetwarzania.
Processing: Płatność jest w trakcie przetwarzania.
Completed: Płatność zakończona pomyślnie.
Cancelled: Płatność została anulowana przez użytkownika lub system. *Failed*: Płatność zakończona niepowodzeniem.

Przeznaczenie: *PaymentState* ma na celu śledzenie ostatecznego wyniku transakcji płatniczej, pomagając w informowaniu użytkownika o zakończeniu płatności oraz jej stanie, co pozwala na odpowiednie reakcje w systemie (np. potwierdzenie transakcji, obsługa błędów lub anulowanie płatności).

Typ ten prezentuje się następująco:

```
/// <summary>
/// Status płatności po jej przetworzeniu.
/// </summary> public enum PaymentState
{
    /// <summary>
    /// Płatność oczekuje na przetworzenie.
    /// </summary>
    Pending,

    /// <summary>
    /// Płatność jest w trakcie przetwarzania.
    /// </summary>
    Processing,

    /// <summary>
    /// Płatność zakończona pomyślnie.
    /// </summary>
    Completed,
```

```
    /// <summary>
    /// Płatność została anulowana.
    /// </summary>
    Cancelled,

    /// <summary>
    /// Płatność zakończona niepowodzeniem.
    /// </summary>
    Failed
}
```

3.4. Udostępnienie biblioteki interfejsowej

W celu usprawnienia przygotowania rozwiązania po stronie operatora dostarczającego terminale, możliwe jest na wniosek wybranego podmiotu przekazanie biblioteki, z którą będzie integrowała się biblioteka przygotowana przez operatora dostarczającego terminale. Następnie po przygotowaniu biblioteki obsługującej terminal przez operatora dostarczającego terminale, zostanie ona przekazania wraz z dokumentacją do PWPW w celu obsłużenia jej po stronie systemów PWPW.