

Uniwersytet w Siedlcach
Wydział Nauk Ścisłych i Przyrodniczych

Kierunek Informatyka
Programowanie w Sieciach Komputerowych

**Aplikacja w Architekturze Mikroserwisów
z szyfrowaniem**

Opracował:

Karol Przewuski

Prowadzący:

dr hab. Prof. Ucz. Stanisław
Ambroszkiewicz

Siedlce, 2025

Spis treści

1. Założenia projektowe	2
2. Schemat blokowy	6
3. Opis poszczególnych aplikacji.....	7
Client	7
Server (API Gateway)	9
Login.....	10
Register	11
Tablica	13
Chat.....	13
Transfer plików wejście	15
Transfer plików wyjście	16
SecurityUtils(Narzędzia bezpieczeństwa)	16
4. Monitorowanie i Analiza Ruchu Sieciowego za pomocą WireShark w Projekcie	18
5. Kody Źródłowe	21
SecurityUtils	21
Client	25
Server	34
Login.....	39
Register	40
Tablica	41
Chat.....	42
FileTransferIn	43
FileTransferOut	44

1. Założenia projektowe

Głównym celem projektu jest stworzenie zintegrowanego systemu komunikacji, który umożliwia użytkownikom realizację następujących funkcji:

- Rejestracja: Umożliwienie nowym użytkownikom tworzenia kont w systemie.
- Logowanie: Uwierzytelnianie użytkowników i dostęp do funkcji systemu.
- Wymiana wiadomości tekstowych: Tworzenie i zarządzanie dialogiem między użytkownikami.
- Transfer plików: Przesyłanie danych z/na serwer plików.

Struktura Systemu:

System będzie się składać z czterech głównych komponentów:

1. Interfejs Użytkownika (CLI):
 - **Opis:** Konsolowy interfejs użytkownika umożliwiający interakcję z systemem.
 - **Funkcje:**
 - Rejestracja i logowanie użytkowników.
 - Wysyłanie wiadomości tekstowych.
 - Przeglądanie tablicy wiadomości.
 - Zarządzanie plikami: wysyłanie i pobieranie.
2. API Gateway:
 - **Opis:** Centralny punkt dostępu do mikroserwisów. Odpowiada za:
 - Przekierowywanie żądań użytkownika do odpowiednich serwisów.
 - Implementację mechanizmów bezpieczeństwa, takich jak szyfrowanie i uwierzytelnianie.
 - **Funkcje:**
 - Szyfrowanie i deszyfrowanie wiadomości.
 - Obsługa podpisów cyfrowych.
 - Autoryzacja użytkowników.
3. Mikroserwisy: Niezależne serwisy zapewniające specyficzne funkcje:
 - Serwis Logowania: Uwierzytelnianie użytkownika i sesji
 - Serwis Rejestracji: Tworzenie nowych kont użytkowników
 - Serwis Tablica: Wyświetlanie ostatnich 10 komunikatów użytkowników
 - Serwis Czat: Wymiana komunikatów między zalogowanymi użytkownikami.
 - Serwis Transferu Plików: Wysyłanie i odbieranie plików
4. Back-end jako usługa (BaaS): Zestaw usług back-endowych w tym:
 - Bazy Danych: Przechowywanie danych użytkowników oraz historii czatów

- Serwer Plików(Folder): Przechowywanie i zarządzanie plikami przesłanymi przez użytkowników.

Technologie:

Projekt będzie wykorzystywał następujące technologie:

- Język programowania: Java
- Protokoły komunikacyjne: TCP/IP dla komunikacji sieciowej
- Zarządzanie bazą danych: MySQL
- Mechanizmy Bezpieczeństwa:
 - RSA (2048-bit) do podpisów cyfrowych.
 - AES (256-bit) do szyfrowania wiadomości.
 - Wymiana kluczy publicznych między CLI a API Gateway.

Szczegółowy Opis Operacji Kryptograficznych w Systemie

1. Początkowa Wymiana Kluczy i Uwierzytelnienie

Przygotowanie:

- Klient generuje własną parę kluczy RSA: klucz prywatny (K-CLI) i publiczny (K+CLI), te klucze są unikalne dla każdego uruchomienia aplikacji i służą do uwierzytelnienia oraz podpisywania wiadomości.
- API Gateway posiada swoją stałą parę kluczy RSA: klucz prywatny (K-API) i publiczny (K+API), te klucze są stałe dla serwera.

Proces Wymiany gdy klient chce połączyć się z serwerem,:

1. Klient przesyła swój klucz publiczny (K+CLI) do API Gateway
2. API Gateway przesyła swój klucz publiczny (K+API) do klienta
3. Obie strony przechowują klucze publiczne partnera do późniejszego użycia.

Uwierzytelnienie Dwustronne:

1. API Gateway → Klient (Serwer weryfikuje klienta):
 - API Gateway generuje losowy ciąg (nonce)
 - Wysyła go do klienta
 - Klient podpisuje go swoim kluczem prywatnym (K-CLI) i odsyła podpisaną wartość
 - API Gateway weryfikuje podpis kluczem publicznym klienta (K+CLI)
 - Jeśli weryfikacja się powiedzie, serwer ma pewność, że klient jest autentyczny
2. Klient → API Gateway (Klient weryfikuje serwer):
 - Klient generuje losowy ciąg (nonce)
 - Wysyła go do serwera

- API Gateway podpisuje go swoim kluczem prywatnym (K-API) i odsyła podpisaną wartość
- Klient weryfikuje podpis kluczem publicznym API Gateway (K+API)
- Jeśli weryfikacja się powiedzie, klient ma pewność, że serwer jest autentyczny

2. Szyfrowanie Komunikacji

Inicjalizacja Sesji:

1. Klient generuje klucz symetryczny AES dla sesji
2. Klucz AES jest szyfrowany kluczem publicznym API Gateway (K+API)
3. Zasyfrowany klucz AES jest przesyłany do API Gateway
4. API Gateway deszyfruje klucz AES swoim kluczem prywatnym (K-API)
5. Od tego momentu obie strony mają ten sam klucz AES do szybkiej komunikacji, który będzie używany do szyfrowania wszystkich dalszych wiadomości w tej sesji

Proces Szyfrowania Wiadomości:

1. Nadawca:
 - Tworzy oryginalną wiadomość
 - Szyfruje ją kluczem AES
 - Tworzy podpis cyfrowy swoim kluczem prywatnym
 - Łączy zasyfrowaną wiadomość i podpis
2. Odbiorca:
 - Odbiera zasyfrowaną wiadomość i podpis
 - Deszyfruje wiadomość kluczem AES
 - Weryfikuje podpis kluczem publicznym nadawcy
 - Przetwarza wiadomość jeśli weryfikacja się powiedzie

3. Transfer Plików

Wysyłanie Pliku:

1. Plik jest dzielony na segmenty o wielkości 1024B
2. Dla każdego segmentu:
 - Szyfrowanie segmentu kluczem AES
 - Tworzenie podpisu segmentu kluczem prywatnym nadawcy
 - Przesłanie zasyfrowanego segmentu i podpisu

Odbieranie Pliku:

1. Dla każdego odebranego segmentu:

- Weryfikacja podpisu kluczem publicznym nadawcy
- Deszyfrowanie segmentu kluczem AES
- Składanie segmentów w całość

4. Komunikacja z Mikrouslugami

Przepływ Danych:

1. Klient → API Gateway:
 - Klient szyfruje dane kluczem AES
 - Dodaje swój podpis cyfrowy (RSA)
 - Wysyła zabezpieczony pakiet
2. API Gateway → Mikrousluga:
 - API Gateway deszyfruje dane
 - Sprawdza podpis klienta
 - Przekazuje czytelne dane do odpowiedniej mikrouslugi
3. Mikrousluga → API Gateway:
 - Mikrousluga przetwarza żądanie
 - Wysyła odpowiedź do API Gateway
 - Dane są w formie niezaszyfrowanej (komunikacja wewnętrzna)
4. API Gateway → Klient:
 - API Gateway szyfruje odpowiedź kluczem AES
 - Dodaje swój podpis cyfrowy (RSA)
 - Wysyła zabezpieczony pakiet do klienta

5. Zakończenie Sesji (Gdy klient kończy prace).

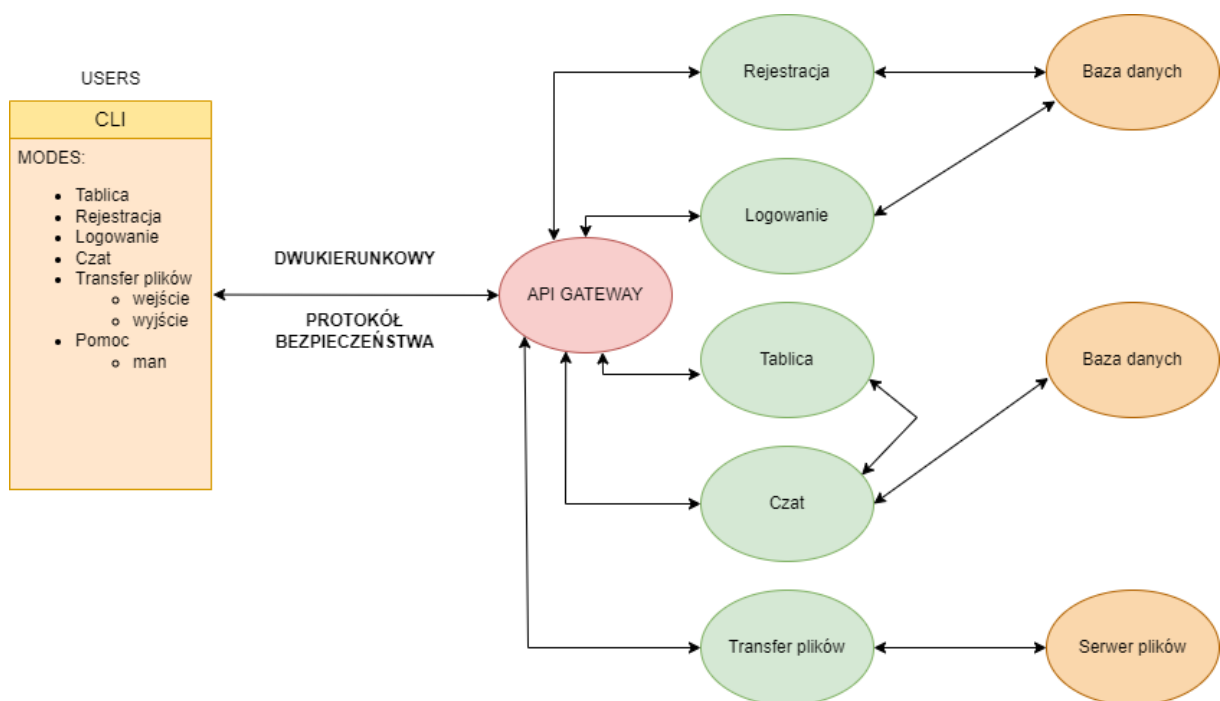
- System usuwa tymczasowy klucz AES
- Czyści wszystkie dane sesji z pamięci
- Zamyka bezpiecznie wszystkie połączenia
- Usuwa tymczasowe klucze i dane uwierzytelniające

Ten system zapewnia:

- Poufność danych (szyfrowanie AES)
- Autentyczność (podpisy RSA)
- Integralność (weryfikacja podpisów)
- Bezpieczną wymianę kluczy (RSA)

- Wydajność (AES do dużych danych)

2. Schemat blokowy



Rysunek 1 Schemat blokowy aplikacji

3.Opis poszczególnych aplikacji

Client

Opis: Aplikacja klienta (CLI) jest uruchamiana po stronie użytkownika i umożliwia interakcję z systemem poprzez konsolowy interfejs użytkownika. Obsługuje różne funkcje systemu, takie jak rejestracja, logowanie, wyświetlanie tablicy wiadomości, czat oraz transfer plików. Komunikacja z serwerem odbywa się za pomocą protokołu TCP/IP z wykorzystaniem szyfrowania AES oraz podpisów cyfrowych opartych na RSA.

Funkcje:

1. Rejestracja:

- Użytkownik wprowadza dane (login i hasło), które są przesyłane do serwera za pośrednictwem API Gateway.
- Dane są szyfrowane kluczem AES i podpisywane kluczem prywatnym klienta.

2. Logowanie:

- Uwierzytelnienie użytkownika na podstawie loginu i hasła.
- Sesja jest uwierzytelniana dwukierunkowo (klient i serwer wymieniają wyzwania).

3. Wyświetlanie tablicy:

- Pobranie ostatnich 10 wiadomości z bazy danych za pośrednictwem API Gateway.
- Dane są deszyfrowywane na kliencie po odebraniu.

4. Czat:

- Wysyłanie wiadomości tekstowych do innych użytkowników.
- Wiadomości są przechowywane w bazie danych i dostępne na żądanie.

5. Transfer plików:

- Wysyłanie i odbieranie plików z serwera.
- Pliki są dzielone na pakiety, szyfrowane i przesyłane w formacie Base64.

6. Pomoc:

- Wbudowany system pomocy dostępny za pomocą komendy man.

- Wyświetla listę dostępnych funkcji i ich opisów, ułatwiając korzystanie z aplikacji.

Architektura Bezpieczeństwa:

- **Szyfrowanie AES:** Wszystkie dane przesyłane między klientem a serwerem są szyfrowane kluczem AES.
- **Podpisy cyfrowe RSA:** Wiadomości są podpisywane kluczem prywatnym klienta i weryfikowane kluczem publicznym serwera.
- **Wymiana kluczy:**
 - Klucz AES jest szyfrowany kluczem publicznym serwera i przesyłany do klienta.
 - Klient używa klucza prywatnego RSA do odszyfrowania klucza AES.

Przykłady z klasy Client:

```
Uruchamianie klienta...
Inicjalizacja zabezpieczeń zakończona
Połączono z serwerem!
Zweryfikowano tożsamość serwera

Wpisz numer funkcji:
1. Login
2. Register
3. Tablica
man. Jeżeli chcesz uzyskać pomoc wpisz 'man'
Wybór:
```

Rysunek 2 Ekran powitalny klienta

```
man
System pomocy:
1. Logowanie - służy do logowania użytkownika
2. Rejestracja - służy do rejestracji nowego użytkownika
3. Pobierz tablicę - pobiera listę wpisów z tablicy
4. Dodaj wpis - dodaje nowy wpis na tablicy
5. Pobierz plik - pobiera plik z serwera
6. Wyślij plik - wysyła plik na serwer
man - wyświetla system pomocy
exit - wyjście z programu
```

Rysunek 3 Menu po wybraniu funkcji man (pomoc)

Server (API Gateway)

Opis: API Gateway pełni funkcję centralnego punktu dostępowego do systemu mikroserwisów. Jego głównym zadaniem jest obsługa żądań od klientów (aplikacji CLI), przekierowywanie ich do odpowiednich mikroserwisów oraz zarządzanie odpowiedziami. API Gateway zapewnia bezpieczeństwo komunikacji poprzez szyfrowanie danych oraz uwierzytelnianie dwukierunkowe między klientem a serwerem.

Funkcje:

1. **Obsługa zapytań od klientów:**
 - API Gateway odbiera zapytania przesyłane przez klienta.
 - Weryfikuje integralność wiadomości poprzez weryfikację podpisów cyfrowych RSA.
 - Odszyfrowuje dane zaszyfrowane kluczem AES.
2. **Przekierowywanie do mikroserwisów:**
 - Na podstawie typu zapytania (login, register, chat, tablica, FTI, FTO), API Gateway kieruje zapytanie do odpowiedniego mikroserwisu.
 - Komunikacja z mikroserwisami odbywa się poprzez protokół TCP/IP.
3. **Przetwarzanie odpowiedzi od mikroserwisów:**
 - API Gateway odbiera odpowiedź od mikroserwisu, podpisuje ją swoim kluczem prywatnym RSA i szyfruje kluczem AES, zanim przekaże ją klientowi.
4. **Mechanizmy bezpieczeństwa:**
 - Wymiana kluczy RSA z klientem na początku sesji.
 - Wysyłanie zaszyfrowanego klucza AES do klienta.
 - Podpisywanie odpowiedzi i weryfikacja podpisów wiadomości od klienta.

Przykłady z klasy Server:

```
Nowy klient połączony: 127.0.0.1
Inicjalizacja zabezpieczeń dla klienta: 127.0.0.1
Otrzymano klucz publiczny od klienta
Wysłano klucz publiczny do klienta
Wysłano wyzwanie do klienta
Zweryfikowano tożsamość klienta
Wysłano odpowiedź na wyzwanie klienta
Wysłano zaszyfrowany klucz AES
```

Rysunek 4 Przykład pracy serwera po połączeniu nowego klienta

Login

Opis: Mikroserwis Login umożliwia użytkownikom uwierzytelnianie poprzez wprowadzenie loginu i hasła. Jego głównym zadaniem jest sprawdzanie poprawności danych logowania w bazie danych i zwracanie odpowiedniego statusu operacji do API Gateway.

Funkcje:

1. Weryfikacja danych logowania:

- Otrzymuje zaszyfrowane żądanie logowania od API Gateway.
- Po odszyfrowaniu sprawdza w bazie danych, czy podany login i hasło są prawidłowe.
- Weryfikuje dane przy użyciu zapytania SQL.

2. Obsługa błędów:

- Zwraca status status:OK, jeśli dane logowania są poprawne.
- Zwraca status status:NO, jeśli login lub hasło są nieprawidłowe.

3. Bezpieczeństwo:

- Dane logowania są odbierane po wcześniejszym odszyfrowaniu w API Gateway.
- Żadne dane uwierzytelniające nie są przesyłane w postaci czystego tekstu.

Przykłady z klasy Login:

```
Wpisz numer funkcji:
1. Login
2. Register
3. Tablica
man. Jeżeli chcesz uzyskać pomoc wpisz 'man'
Wybór: 1
Wpisz nazwę użytkownika: karol
Wpisz hasło: karol
Wysyłanie pakietu logowania: type:login#login:karol#haslo:karol
Otrzymano odpowiedź: J9rw3FZA+9rVLP0S/MsidA==#signature:WhXd2iPP/JBa/TDT4JgTU3vL/NRIv0i
Odszyfrowana wiadomość: status:OK
Zalogowano pomyślnie
```

Rysunek 4 Logowanie

Register

Opis: Mikroserwis Register jest odpowiedzialny za obsługę rejestracji nowych użytkowników. Jego głównym zadaniem jest weryfikacja, czy nazwa użytkownika jest unikalna, a następnie zapisanie nowego konta w bazie danych. Mikroserwis komunikuje się z API Gateway, który przesyła odpowiednie żądania od klientów.

Funkcje:

1. Weryfikacja unikalności nazwy użytkownika:

- Przed zapisaniem nowego użytkownika w bazie danych, mikroserwis sprawdza, czy podany login już istnieje.
- W przypadku kolizji nazwy użytkownika zwraca status status:NO.

2. Rejestracja użytkownika:

- Po pozytywnej weryfikacji unikalności, dane nowego użytkownika (login i hasło) są zapisywane w bazie danych.

3. Obsługa błędów:

- Jeśli w trakcie rejestracji wystąpi błąd (np. problem z bazą danych), mikroserwis zwraca status status:NO.

Przykłady z klasy Register:

```
2
Podaj login: karol
Podaj haslo: karol
W bazie istnieje użytkownik z tą samą nazwą użytkownika
Wpisz numer funkcji:
1. Login
2. Register
3. Tablica
man. Jeżeli chcesz uzyskać pomoc wpisz 'man'
|
```

Rysunek 5 Przykład próby rejestracji użytkownika pod nazwą która istnieje

```
2
Podaj login: karol4
Podaj haslo: karol4
Zarejestrowano pomyślnie
Wpisz numer funkcji:
1. Login
2. Register
3. Tablica
man. Jeżeli chcesz uzyskać pomoc wpisz 'man'
|
```

Rysunek 6 Przykład rejestracji użytkownika

Tablica

Opis: Mikroserwis Tablica umożliwia wyświetlenie ostatnich dziesięciu wpisów z tablicy ogłoszeń. Jest dostępny zarówno dla użytkowników zalogowanych, jak i niezalogowanych, dzięki czemu zapewnia ogólnodostępny wgląd w aktywność systemu.

Funkcje:

1. Pobieranie wpisów:

- Mikroserwis odczytuje z bazy danych ostatnie 10 wpisów uporządkowanych według daty dodania.
- Obsługuje zapytania w trybie "tylko do odczytu".

2. Obsługa zapytań:

- Mikroserwis odbiera żądania od API Gateway w celu zwrócenia najnowszych wpisów.
- Dane są zwracane w formacie zgodnym z wymaganiami klienta (CLI).

3. Brak konieczności logowania:

- Usługa jest dostępna dla wszystkich użytkowników, niezależnie od ich statusu logowania.

```
3
karol napisał: Dzień dobry
karol1 napisał: witam
karol2 napisał: czesc
karol3 napisał: hej
Wpisz numer funkcji:
1. Login
2. Register
3. Tablica
man. Jeżeli chcesz uzyskać pomoc wpisz 'man'
|
```

Rysunek 7 Przykład wiadomości w tablicy

Chat

Opis: Mikroserwis Chat umożliwia zalogowanym użytkownikom dodawanie nowych wpisów do tablicy ogłoszeń. Wpisy są przechowywane w bazie danych i stają się widoczne dla innych użytkowników korzystających z funkcji tablicy.

Funkcje:

1. Dodawanie wpisów:

- Mikroserwis odbiera wiadomości tekstowe od API Gateway i zapisuje je w bazie danych jako nowy wpis.
- Każdy wpis zawiera nazwę autora oraz treść wiadomości.

2. Zarządzanie liczbą wpisów:

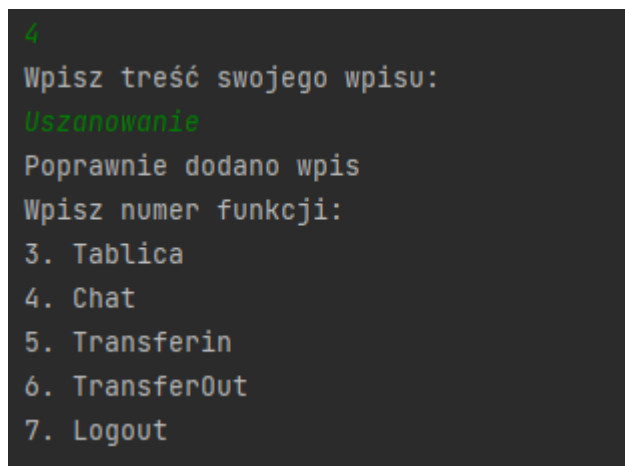
- Jeśli liczba wpisów na tablicy przekracza 10, najstarszy wpis jest automatycznie usuwany, aby utrzymać limit.

3. Obsługa błędów:

- W przypadku problemów z bazą danych mikroserwis zwraca odpowiedni status błędu (status:NO).

4. Ograniczenie dostępu:

- Funkcja jest dostępna wyłącznie dla użytkowników zalogowanych, co jest weryfikowane przez API Gateway przed przesłaniem żądania.



```
4
Wpisz treść swojego wpisu:
Uszanowanie
Poprawnie dodano wpis
Wpisz numer funkcji:
3. Tablica
4. Chat
5. Transferin
6. TransferOut
7. Logout
```

Rysunek 8 Przykład dodawania wpisu na chacie

Transfer plików wejście

Opis: Mikroserwis FileTransferIn umożliwia użytkownikom pobieranie plików z serwera. Użytkownik podaje nazwę pliku, a mikroserwis przesyła dane w formie pakietów, zapewniając jednocześnie informację o statusie operacji.

Funkcje:

1. Obsługa żądania pobrania pliku:

- Mikroserwis przyjmuje od API Gateway nazwę pliku do pobrania.
- Weryfikuje, czy plik istnieje na serwerze.

2. Podział pliku na pakiety:

- Plik jest dzielony na mniejsze części (maksymalnie 1024 bajty), które są wysyłane pojedynczo w formacie Base64.
- Każdy pakiet zawiera dane o:
 - Nazwie pliku.
 - Liczbie pakietów do przesłania.
 - Zawartości bieżącego pakietu.

3. Informacja o statusie:

- Jeśli plik istnieje i zostanie poprawnie przesłany, mikroserwis zwraca status status:OK.
- W przypadku błędu (np. brak pliku), zwracany jest status status:NO.

```
5
Podaj nazwę pliku:
plik1.txt
odebrano wszystkie paczki
Poprawnie pobrano plik.
Wpisz numer funkcji:
3. Tablica
4. Chat
5. Transferin
6. TransferOut
7. Logout
```


Transfer plików wyjście

Opis: Mikroserwis FileTransferOut umożliwia użytkownikom wysyłanie plików do serwera. Użytkownik określa nazwę pliku i jego zawartość, która następnie jest przesyłana w formie pakietów danych. Każdy pakiet zawiera szczegółowe metadane, takie jak nazwa pliku, liczba pakietów i treść bieżącego pakietu.

Funkcje:

1. Odbieranie pliku od użytkownika:

- Mikroserwis odbiera dane w formie pakietów przesyłanych przez API Gateway.
- Obsługuje wielopartyjne przesyłanie danych, co umożliwia przesyłanie dużych plików.

2. Składanie pliku:

- Po odebraniu wszystkich pakietów mikroserwis składa je w jeden plik i zapisuje go w katalogu docelowym.

3. Informacja o statusie:

- Jeśli plik zostanie poprawnie zapisany, mikroserwis zwraca status status:OK.
- W przypadku błędu (np. brak pakietów lub problem z zapisem) zwracany jest status status:NO.

```
0
1. plik1.txt
2. plik2.txt
3. plik3.txt
Wybierz plik który chcesz wysłać:
3
0
OK
Plik zapisano pomyślnie.
```

Rysunek 10 Wysłanie pliku na serwer

SecurityUtils(Narzędzia bezpieczeństwa)

Opis: Klasa SecurityUtils jest centralnym modułem obsługującym mechanizmy bezpieczeństwa w systemie. Odpowiada za szyfrowanie i deszyfrowanie danych, zarządzanie kluczami kryptograficznymi (AES i RSA), generowanie i weryfikację podpisów cyfrowych, a także uwierzytelnianie dwukierunkowe między klientem a serwerem. Jest używana przez API Gateway oraz mikroserwisy, zapewniając integralność, poufność i autentyczność danych.

Funkcje:

1. Zarządzanie kluczami kryptograficznymi:

- Generowanie par kluczy RSA (klucz publiczny i prywatny).
- Generowanie i zarządzanie kluczem AES do szyfrowania symetrycznego.
- Ustawianie klucza publicznego partnera komunikacji.

2. Szyfrowanie i deszyfrowanie danych:

- **AES:** Szyfrowanie i deszyfrowanie wiadomości za pomocą klucza AES.
- **RSA:** Szyfrowanie klucza AES kluczem publicznym partnera i deszyfrowanie klucza AES za pomocą klucza prywatnego.

3. Podpisy cyfrowe:

- Generowanie podpisów cyfrowych wiadomości za pomocą klucza prywatnego RSA.
- Weryfikacja podpisów wiadomości za pomocą klucza publicznego partnera.

4. Uwierzytelnianie dwukierunkowe:

- Generowanie wyzwań (nonce) do weryfikacji tożsamości.
- Podpisywanie i weryfikacja wyzwań w celu zapewnienia autentyczności komunikacji.

5. Kodowanie i dekodowanie wiadomości:

- Łączenie zaszyfrowanej wiadomości i podpisu w jeden komunikat.
- Rozdzielanie i deszyfrowanie otrzymanej wiadomości oraz weryfikacja podpisu.

Procesy i Algorytmy:

1. Szyfrowanie danych (AES):

- Dane są szyfrowane za pomocą klucza AES w trybie szyfrowania blokowego.
- Wynik szyfrowania jest kodowany w Base64 przed przesłaniem.

2. Podpisy cyfrowe (RSA):

- Dane są hashowane za pomocą SHA-256.
- Hash jest podpisywany kluczem prywatnym, generując podpis cyfrowy.

3. Uwierzytelnianie:

- Serwer generuje nonce (liczbę jednorazową), którą klient podpisuje i zwraca w celu weryfikacji.
- Analogiczny proces działa w odwrotnym kierunku.

4. Monitorowanie i Analiza Ruchu Sieciowego za pomocą WireShark w Projekcie

WireShark w tym projekcie jest wykorzystywany do analizy ruchu sieciowego pomiędzy klientem (CLI), API Gateway oraz mikroserwisami. Jego głównym celem jest weryfikacja implementacji mechanizmów bezpieczeństwa (szyfrowania i podpisów cyfrowych) oraz identyfikacja potencjalnych słabości w przesyłaniu danych.

Analiza Ruchu Sieciowego za pomocą Wiresharka

Przeprowadzona została analiza ruchu sieciowego w celu weryfikacji bezpieczeństwa i integralności komunikacji w projekcie. Głównym celem było sprawdzenie, czy przesyłane dane są odpowiednio szyfrowane i podpisywane cyfrowo.

Kroki Testowe

Przygotowanie:

1. Uruchomiono Wiresharka jako administrator.
2. Wybrano interfejs sieciowy **Localhost**, ponieważ komunikacja w projekcie odbywa się lokalnie.
3. Ustawiono filtr: `tcp.port == 1410` (port używany przez API Gateway).
4. Uruchomiono wszystkie komponenty programu w kolejności:
 - Mikroserwisy (Login, Register, Tablica, Chat, FileTransferIn, FileTransferOut).
 - Server.java (API Gateway).
 - Client.java (interfejs użytkownika).

Przeprowadzone Testy i Wyniki

1. Test Rejestracji

- **Opis:** Użytkownik wprowadził dane rejestracyjne (login i hasło).
- **Wynik w Wireshark:**
 - Przechwycono zaszyfrowaną wiadomość rejestracyjną w formacie type:register#login:<login>#haslo:<hasło> zaszyfrowaną AES.
 - Wysłane dane były nieczytelne w przechwyconych pakietach.
 - Każda wiadomość zawierała podpis cyfrowy, który został poprawnie zweryfikowany.

2. Test Logowania

- **Opis:** Użytkownik wprowadził dane logowania.
- **Wynik w Wireshark:**
 - Przechwycono wymianę kluczy RSA na początku sesji.
 - Klucz AES został przesłany w zaszyfrowanej formie za pomocą RSA.
 - Wymiana challenge-response była widoczna jako zaszyfrowane wiadomości.
 - Właściwe dane logowania były zaszyfrowane AES i podpisane cyfrowo.

3. Test Wysłania Wiadomości na Tablicę

- **Opis:** Użytkownik dodał wpis do tablicy.
- **Wynik w Wireshark:**
 - Przechwycono zaszyfrowaną wiadomość type:chat#login:<login>#tresc:<treść wiadomości> przesłaną do serwera.
 - Odpowiedź serwera (status:OK) była również zaszyfrowana i podpisana cyfrowo.
 - Treść wiadomości nie była widoczna w przechwyconych pakietach.

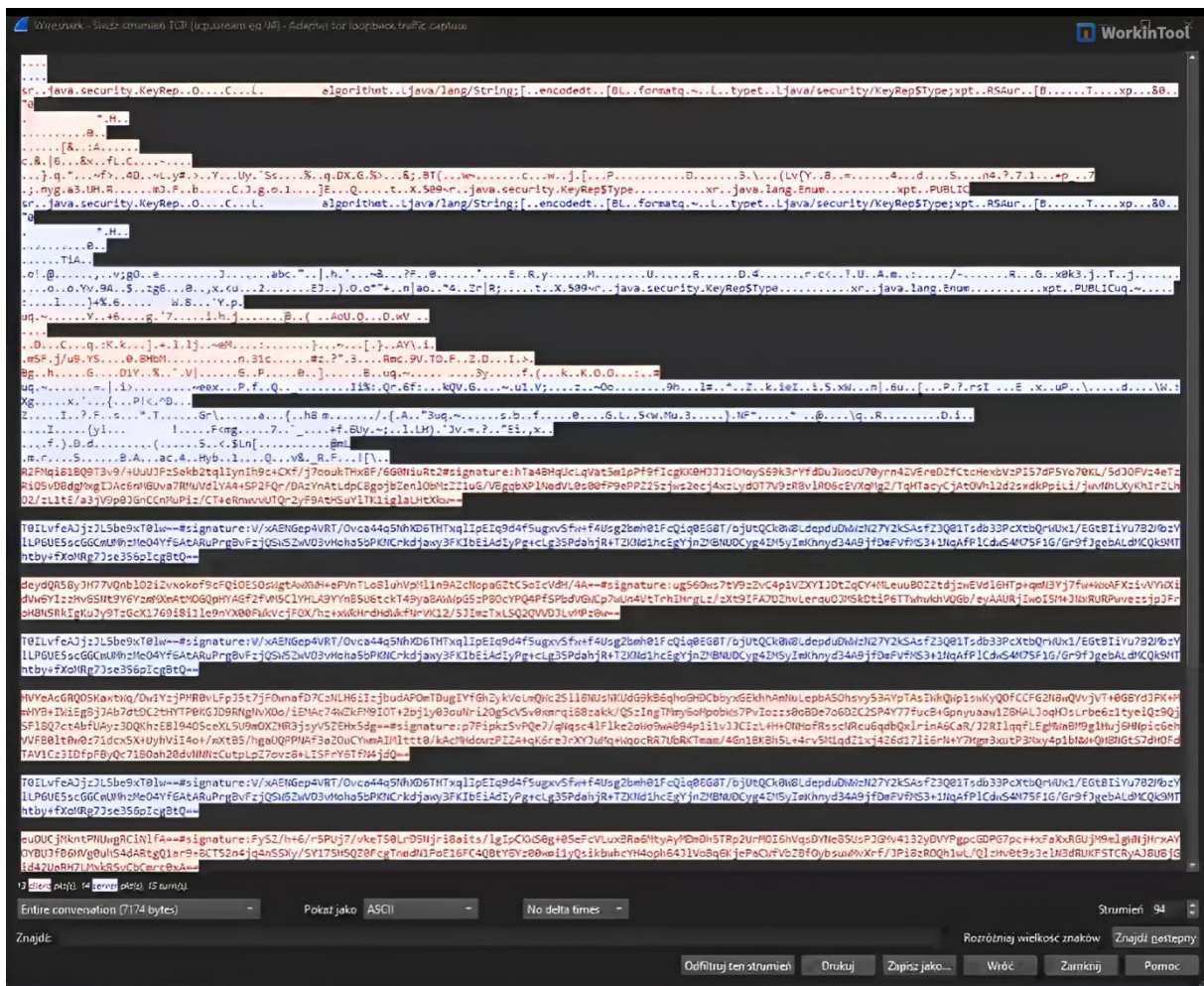
4. Test Transferu Pliku

- **Opis:** Użytkownik wysłał plik do serwera.
- **Wynik w Wireshark:**
 - Przechwycono wiele pakietów, ponieważ plik został podzielony na mniejsze fragmenty (1024 bajty).
 - Każdy fragment pliku był zaszyfrowany AES i podpisany cyfrowo.
 - Odpowiedź serwera (status:OK) potwierdzająca odbiór pliku była zaszyfrowana.

5. Test Pobierania Pliku

- **Opis:** Użytkownik pobrał wcześniej przesłany plik.
- **Wynik w Wireshark:**

- Przechwycono żądanie pobrania pliku w formacie type:FTI#login:<login>#filename:<nazwa_pliku>.
- Odpowiedź serwera zawierała zaszyfrowane fragmenty pliku.
- Pakiety z danymi pliku były dłuższe, co wynika z większej ilości przesyłanych danych.



Rysunek 11 Podgląd każdej wykonanej operacji podczas testowania bezpieczeństwa.

Powyższy zrzut ekranu przedstawia analizę przechwyconej komunikacji sieciowej pomiędzy klientem a API Gateway w projekcie. Widoczny przepływ danych obrazuje wymianę informacji w trakcie realizacji różnych operacji, takich jak logowanie, rejestracja, czy transfer plików.

Czerwony: pakiety od klienta do serwera

Niebieski: pakiety od serwera do klienta

5. Kody Źródłowe

SecurityUtils

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.security.*;
import java.util.Base64;

public class SecurityUtils { 10 usages
    private KeyPair keyPair; 5 usages
    private PublicKey partnerPublicKey; 1 usage
    private SecretKey aesKey; 5 usages
    private final SecureRandom secureRandom; 2 usages

    public SecurityUtils() throws NoSuchAlgorithmException { 5 usages
        this.keyPair = generateKeyPair();
        this.secureRandom = new SecureRandom();
        this.aesKey = generateAESKey();
    }

    // Generowanie pary kluczy RSA
    private KeyPair generateKeyPair() throws NoSuchAlgorithmException { 1 usage
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
        keyPairGenerator.initialize(2048);
        return keyPairGenerator.generateKeyPair();
    }

    // Generowanie klucza AES
    private SecretKey generateAESKey() throws NoSuchAlgorithmException { 1 usage
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(256); // używamy 256-bitowego klucza
        return keyGen.generateKey();
    }

    // Generowanie wyzwania do uwierzytelnienia
    public byte[] generateChallenge() { 2 usages
        byte[] challenge = new byte[32];
        secureRandom.nextBytes(challenge);
        return challenge;
    }
}
```

```

// Podpisywanie wyzwania
public byte[] signChallenge(byte[] challenge) throws Exception { 2 usages
    Signature signature = Signature.getInstance("SHA256withRSA");
    signature.initSign(keyPair.getPrivate());
    signature.update(challenge);
    return signature.sign();
}

// Weryfikacja podpisu wyzwania
public boolean verifyChallenge(byte[] challenge, byte[] signedChallenge, PublicKey publicKey) throws Exception { 2 usages
    Signature signature = Signature.getInstance("SHA256withRSA");
    signature.initVerify(publicKey);
    signature.update(challenge);
    return signature.verify(signedChallenge);
}

// Szyfrowanie danych za pomocą AES
public byte[] encryptAES(String data) throws Exception { 1 usage
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, aesKey);
    return cipher.doFinal(data.getBytes());
}

// Deszyfrowanie danych za pomocą AES
public String decryptAES(byte[] encryptedData) throws Exception { 1 usage
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.DECRYPT_MODE, aesKey);
    byte[] decryptedBytes = cipher.doFinal(encryptedData);
    return new String(decryptedBytes);
}

// Szyfrowanie klucza AES za pomocą klucza publicznego RSA
public byte[] encryptAESKey(PublicKey publicKey) throws Exception { 1 usage
    Cipher cipher = Cipher.getInstance("RSA");
    cipher.init(Cipher.ENCRYPT_MODE, publicKey);
    return cipher.doFinal(aesKey.getEncoded());
}

```

```

// Ustawianie klucza AES z zaszyfrowanego klucza
public void setAESKeyFromEncrypted(byte[] encryptedKey) throws Exception { 1 usage
    Cipher cipher = Cipher.getInstance( transformation: "RSA");
    cipher.init(Cipher.DECRYPT_MODE, keyPair.getPrivate());
    byte[] decryptedKey = cipher.doFinal(encryptedKey);
    this.aesKey = new SecretKeySpec(decryptedKey, algorithm: "AES");
}

// Ustawianie klucza publicznego partnera
public void setPartnerPublicKey(PublicKey publicKey) { 2 usages
    this.partnerPublicKey = publicKey;
}

// Pobieranie klucza publicznego
public PublicKey getPublicKey() { 2 usages
    return keyPair.getPublic();
}

// Kodowanie wiadomości z podpisem
public String encodeMessage(String message, byte[] signature) throws Exception { 13 usages
    byte[] encryptedData = encryptAES(message);
    String encodedData = Base64.getEncoder().encodeToString(encryptedData);
    String encodedSignature = Base64.getEncoder().encodeToString(signature);
    return encodedData + "#signature:" + encodedSignature;
}

// Dekodowanie wiadomości z podpisem
public String[] decodeMessage(String encodedMessage) throws Exception { 9 usages
    String[] parts = encodedMessage.split( regex: "#signature:");
    if (parts.length != 2) {
        throw new IllegalArgumentException("Invalid message format");
    }
    // Deszyfrujemy wiadomość używając AES
    byte[] encryptedData = Base64.getDecoder().decode(parts[0]);
    String decryptedMessage = decryptAES(encryptedData);
    System.out.println("Odszyfrowana wiadomość: " + decryptedMessage);
    return new String[]{decryptedMessage, parts[1]};
}

```



```

// Podpisywanie danych
public byte[] sign(String data) throws Exception { 13 usages
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] hashedData = digest.digest(data.getBytes(StandardCharsets.UTF_8));

    Signature signature = Signature.getInstance("SHA256withRSA");
    signature.initSign(keyPair.getPrivate());
    signature.update(hashedData);
    return signature.sign();
}

// Weryfikacja podpisu danych
public boolean verify(String data, byte[] signedData, PublicKey publicKey) throws Exception { 8 usages
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] hashedData = digest.digest(data.getBytes(StandardCharsets.UTF_8));

    Signature signature = Signature.getInstance("SHA256withRSA");
    signature.initVerify(publicKey);
    signature.update(hashedData);
    return signature.verify(signedData);
}
}

```

Client

```
1 import java.io.*;
2 import java.net.*;
3 import java.security.*;
4 import java.util.*;
5
6 public class Client {
7     private static SecurityUtils securityUtils; 32 usages
8     private static PublicKey serverPublicKey; 9 usages
9
10    public static void main(String[] args) {
11        Scanner sc = new Scanner(System.in);
12        String line = null;
13        boolean czyZalogowany = false;
14        String loginZalogowanego = null;
15
16        System.out.println("Uruchamianie klienta...");
17
18        try {
19            // Inicjalizacja narzędzi kryptograficznych
20            securityUtils = new SecurityUtils();
21            System.out.println("Inicjalizacja zabezpieczeń zakończona");
22
23            // Połączenie z serwerem
24            Socket socket = new Socket("localhost", 1410);
25            System.out.println("Połączono z serwerem!");
26
27            ObjectOutputStream objOut = new ObjectOutputStream(socket.getOutputStream());
28            ObjectInputStream objIn = new ObjectInputStream(socket.getInputStream());
29
30            // Krok 1: Wymiana kluczy publicznych
31            objOut.writeObject(securityUtils.getPublicKey());
32            serverPublicKey = (PublicKey) objIn.readObject();
33            securityUtils.setPartnerPublicKey(serverPublicKey);
34
35            // Krok 2: Uwierzytelnienie serwera
36            byte[] serverChallenge = (byte[]) objIn.readObject();
37            byte[] serverResponse = securityUtils.signChallenge(serverChallenge);
38            objOut.writeObject(serverResponse);
```

```

39
40 // Krok 3: Uwierzytelnienie klienta
41 byte[] clientChallenge = securityUtils.generateChallenge();
42 objOut.writeObject(clientChallenge);
43 byte[] signedServerChallenge = (byte[]) objIn.readObject();
44 boolean serverVerified = securityUtils.verifyChallenge(clientChallenge, signedServerChallenge, serverPublicKey);
45
46 if (!serverVerified) {
47     throw new SecurityException("Nie udało się zweryfikować tożsamości serwera!");
48 }
49 System.out.println("Zweryfikowano tożsamość serwera");
50
51 // Krok 4: Wymiana klucza AES
52 byte[] encryptedAESKey = (byte[]) objIn.readObject();
53 securityUtils.setAESKeyFromEncrypted(encryptedAESKey);
54
55 PrintWriter out = new PrintWriter(socket.getOutputStream(), autoFlush: true);
56 BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
57
58 // Główna petla obsługująca interakcje użytkownika
59 while (!"exit".equalsIgnoreCase(line)) {
60     menu(czyZalogowany);
61     line = sc.nextLine();

```

```

63     switch (line) {
64         case "1": {
65             // Logowanie użytkownika
66             System.out.print("Wpisz nazwę użytkownika: ");
67             String login = sc.nextLine();
68             System.out.print("Wpisz hasło: ");
69             String haslo = sc.nextLine();
70             StringJoiner loginPacket = new StringJoiner(delimiter: "#");
71             loginPacket.add("type:login");
72             loginPacket.add("login:" + login);
73             loginPacket.add("haslo:" + haslo);
74
75             String packet = loginPacket.toString();
76             byte[] signature = securityUtils.sign(packet);
77             String encodedMessage = securityUtils.encodeMessage(packet, signature);
78             out.println(encodedMessage);
79
80             String encodedResponse = in.readLine();
81             String[] parts = securityUtils.decodeMessage(encodedResponse);
82             String response = parts[0];
83             byte[] responseSignature = Base64.getDecoder().decode(parts[1]);
84
85             if (securityUtils.verify(response, responseSignature, serverPublicKey)) {
86                 String[] splittedResponse = response.split(regex: "#");
87                 if (splittedResponse[1].equals("OK")) {
88                     czyZalogowany = true;
89                     loginZalogowanego = login;
90                     System.out.println("Zalogowano pomyślnie");
91                 } else {
92                     System.out.println("Błąd logowania - nieprawidłowe dane");
93                 }
94             } else {
95                 System.out.println("Ostrzeżenie: Otrzymano nieprawidłowo podpisaną odpowiedź!");
96             }
97             break;
98         }

```

```

99         case "2": {
100             // Rejestracja nowego użytkownika
101             System.out.print("Podaj login: ");
102             String nowyLogin = sc.nextLine();
103             System.out.print("Podaj hasło: ");
104             String nowyHasło = sc.nextLine();
105             StringJoiner registerPacket = new StringJoiner("delimiter: "#");
106             registerPacket.add("type:register");
107             registerPacket.add("login:" + nowyLogin);
108             registerPacket.add("hasło:" + nowyHasło);
109
110             String packet = registerPacket.toString();
111             byte[] signature = securityUtils.sign(packet);
112             out.println(securityUtils.encodeMessage(packet, signature));
113
114             String encodedResponse = in.readLine();
115             String[] parts = securityUtils.decodeMessage(encodedResponse);
116             String response = parts[0];
117             byte[] responseSignature = Base64.getDecoder().decode(parts[1]);
118
119             if (securityUtils.verify(response, responseSignature, serverPublicKey)) {
120                 String[] splittedResponse = response.split("regex: ":"");
121                 if (splittedResponse[1].equals("OK")) {
122                     System.out.println("Zarejestrowano pomyślnie");
123                 } else {
124                     System.out.println("Błąd rejestracji - login już istnieje");
125                 }
126             }
127             break;
128         }

```

```

129         case "3": {
130             // Pobieranie tablicy wiadomości
131             String tablicaPacket = "type:tablica";
132             byte[] signature = securityUtils.sign(tablicaPacket);
133             out.println(securityUtils.encodeMessage(tablicaPacket, signature));
134
135             String encodedResponse = in.readLine();
136             String[] parts = securityUtils.decodeMessage(encodedResponse);
137             String response = parts[0];
138             byte[] responseSignature = Base64.getDecoder().decode(parts[1]);
139
140             if (securityUtils.verify(response, responseSignature, serverPublicKey)) {
141                 String[] splittedResponse = response.split("regex: ":"");
142                 if (splittedResponse[1].equals("OK")) {
143                     odczytajTablice(response);
144                 } else {
145                     System.out.println("Nie udało pobrać się danych z tablicy");
146                 }
147             }
148             break;
149         }

```

```

150     case "4": {
151         // Dodawanie nowej wiadomości na tablicy
152         if (!czyZalogowany) {
153             System.out.println("Musisz być zalogowany aby dodać wpis");
154             break;
155         }
156         System.out.println("Wpisz treść swojego wpisu: ");
157         String tresc = sc.nextLine();
158         StringJoiner chatPacket = new StringJoiner("delimiter: "#");
159         chatPacket.add("type:chat");
160         chatPacket.add("login:" + loginZalogowanego);
161         chatPacket.add("tresc:" + tresc);
162
163         String packet = chatPacket.toString();
164         byte[] signature = securityUtils.sign(packet);
165         out.println(securityUtils.encodeMessage(packet, signature));
166
167         String encodedResponse = in.readLine();
168         String[] parts = securityUtils.decodeMessage(encodedResponse);
169         String response = parts[0];
170         byte[] responseSignature = Base64.getDecoder().decode(parts[1]);
171
172         if (securityUtils.verify(response, responseSignature, serverPublicKey)) {
173             String[] splittedResponse = response.split("regex: ":"");
174             if (splittedResponse[1].equals("OK")) {
175                 System.out.println("Dodano wpis pomyślnie");
176             } else {
177                 System.out.println("Błąd podczas dodawania wpisu");
178             }
179         }
180         break;
181     }

```

```

182         case "5": {
183             // Pobieranie pliku z serwera
184             if (!czyZalogowany) {
185                 System.out.println("Musisz być zalogowany aby pobrać plik");
186                 break;
187             }
188             System.out.println("Podaj nazwę pliku:");
189             String nazwaPliku = sc.nextLine();
190             StringJoiner newFTIPacket = new StringJoiner("delimiter: #");
191             newFTIPacket.add("type:FTI");
192             newFTIPacket.add("login:" + loginZalogowanego);
193             newFTIPacket.add("filename:" + nazwaPliku);
194
195             String packet = newFTIPacket.toString();
196             byte[] signature = securityUtils.sign(packet);
197             out.println(securityUtils.encodeMessage(packet, signature));
198
199             ArrayList<String> packetsReceivedList = new ArrayList<String>();
200             String encodedResponse = in.readLine();
201             String[] parts = securityUtils.decodeMessage(encodedResponse);
202             String response = parts[0];
203             byte[] responseSignature = Base64.getDecoder().decode(parts[1]);
204
205             if (securityUtils.verify(response, responseSignature, serverPublicKey)) {
206                 String[] receivedPacket = response.split("regex: #|:");
207                 if (receivedPacket[1].equals("OK")) {
208                     int packetNo = Integer.parseInt(receivedPacket[5]);
209                     int packetsReceived = 1;
210                     while (packetsReceived < packetNo) {
211                         packetsReceivedList.add(response);
212                         encodedResponse = in.readLine();
213                         parts = securityUtils.decodeMessage(encodedResponse);
214                         response = parts[0];
215                         packetsReceived++;
216                     }
217                     if (packetNo == 1) {
218                         packetsReceivedList.add(response);
219                     }

```

```

219     }
220     System.out.println("Odebrano wszystkie paczki");
221
222     String[] splittedPacket;
223     String newFileName = "";
224     ArrayList<byte[]> newFileContentList = new ArrayList<>();
225     for (int i = 0; i < packetsRecivedList.size(); i++) {
226         splittedPacket = packetsRecivedList.get(i).split(regex: "#|:");
227         byte[] recivedContent = Base64.getDecoder().decode(splittedPacket[7]);
228         newFileContentList.add(recivedContent);
229         if (i == 0) {
230             newFileName = splittedPacket[3];
231         }
232     }
233     String newPath = "C:\\Moje_pliki\\" + newFileName;
234     File file = new File(newPath);
235     boolean isExsisting = file.exists();
236     if (isExsisting) {
237         System.out.println("Istnieje plik o tej samej nazwie");
238         break;
239     }
240     try (OutputStream outputStream = new FileOutputStream(file)) {
241         for (byte[] content : newFileContentList) {
242             outputStream.write(content);
243         }
244     } catch (IOException e) {
245         e.printStackTrace();
246     }
247     System.out.println("Poprawnie pobrano plik.");
248 } else {
249     System.out.println("Nie udało się pobrać pliku sprawdź logi w FileTransferIn");
250 }
251 }
252 break;
253 }

```

```

254     case "0": {
255         // Wysyłanie pliku na serwer
256         if (!czyZalogowany) {
257             System.out.println("Musisz być zalogowany aby wysłać plik");
258             break;
259         }
260         String[] pathnames;
261         String path = "C:\\Moje_pliki\\";
262         File file = new File(path);
263         pathnames = file.list();
264         int fileNo = 1;
265         if (pathnames == null) {
266             System.out.println("w tym folderze nie ma plików");
267             break;
268         } else {
269             for (String pathname : pathnames) {
270                 System.out.println(fileNo + ". " + pathname);
271                 fileNo++;
272             }
273         }
274         System.out.println("Wybierz plik który chcesz wysłać:");
275         int wybor = sc.nextInt() - 1;
276         File fileToSend = new File(pathname: "C:\\Moje_pliki\\" + pathnames[wybor]);
277         long fileSize = fileToSend.length();
278         int packetCount = (int) Math.ceil((double) fileSize / 1024);
279         InputStream fileInputStream = new FileInputStream(fileToSend);
280         byte[] buffer = new byte[1024];
281         StringJoiner newFTOPacket = new StringJoiner(delimiter: "#");
282         int bytesRead;
283         for (int i = 0; i < packetCount; i++) {
284             bytesRead = fileInputStream.read(buffer);
285             byte[] packetToAdd = new byte[bytesRead];
286             System.arraycopy(buffer, srcPos: 0, packetToAdd, destPos: 0, bytesRead);
287             newFTOPacket.add("type:FT0");
288             newFTOPacket.add("login:" + loginZalogowanego);
289             newFTOPacket.add("filename:" + fileToSend.getName());
290             newFTOPacket.add("packetsNo:" + packetCount);
291             newFTOPacket.add("content:" + Base64.getEncoder().encodeToString(packetToAdd));
292
293             String packet = newFTOPacket.toString();
294             byte[] signature = securityUtils.sign(packet);
295             out.println(securityUtils.encodeMessage(packet, signature));
296             newFTOPacket = new StringJoiner(delimiter: "#");
297         }
298
299         String encodedResponse = in.readLine();
300         String[] parts = securityUtils.decodeMessage(encodedResponse);
301         String response = parts[0];
302         byte[] responseSignature = Base64.getDecoder().decode(parts[1]);
303
304         if (securityUtils.verify(response, responseSignature, serverPublicKey)) {
305             String[] splittedResponse = response.split(regex: "#");
306             if (splittedResponse[1].equals("OK")) {
307                 System.out.println("Plik wysłano pomyślnie");
308             } else {
309                 System.out.println("Wystąpił problem z przesyłaniem pliku sprawdź logi w klasie FileTransferOut");
310             }
311         }
312         break;
313     }

```



```

314         case "7":
315             // Wylogowanie użytkownika
316             if (czyZalogowany) {
317                 czyZalogowany = false;
318                 loginZalogowanego = null;
319                 System.out.println("Wylogowano pomyślnie");
320             } else {
321                 System.out.println("Nie jesteś zalogowany");
322             }
323             break;
324         case "man":
325             // Wyświetlenie systemu pomocy
326             wyswietlPomoc();
327             break;
328         case "exit":
329             // Zamykanie aplikacji
330             System.out.println("Zamykanie aplikacji...");
331             socket.close();
332             break;
333         default:
334             System.out.println("Nieprawidłowa opcja. Wybierz ponownie.");
335     }
336 }
337 sc.close();
338 } catch (Exception e) {
339     System.out.println("Błąd połączenia z serwerem: " + e.getMessage());
340     e.printStackTrace();
341 }
342 }

```

```

344 // Wyświetlenie menu w zależności od statusu logowania
345 public static void menu(boolean czyZalogowany) { 1 usage
346     if (czyZalogowany) {
347         System.out.println("\nWpisz numer funkcji:");
348         System.out.println("3. Tablica");
349         System.out.println("4. Chat");
350         System.out.println("5. Transferin");
351         System.out.println("6. TransferOut");
352         System.out.println("7. Logout");
353         System.out.print("Wybór: ");
354     } else {
355         System.out.println("\nWpisz numer funkcji:");
356         System.out.println("1. Login");
357         System.out.println("2. Register");
358         System.out.println("3. Tablica");
359         System.out.println("man. Jeżeli chcesz uzyskać pomoc wpisz 'man'");
360         System.out.print("Wybór: ");
361     }
362 }
363
364 // Wyświetlenie systemu pomocy
365 private static void wyswietlPomoc() { 1 usage
366     System.out.println("\nSystem pomocy:");
367     System.out.println("1. Logowanie - służy do logowania użytkownika");
368     System.out.println("2. Rejestracja - służy do rejestracji nowego użytkownika");
369     System.out.println("3. Pobierz tablicę - pobiera listę wpisów z tablicy");
370     System.out.println("4. Dodaj wpis - dodaje nowy wpis na tablicy");
371     System.out.println("5. Pobierz plik - pobiera plik z serwera");
372     System.out.println("6. Wyślij plik - wysyła plik na serwer");
373     System.out.println("man - wyświetla system pomocy");
374     System.out.println("exit - wyjście z programu");
375 }
376
377 // Odczytanie i wyświetlenie tablicy wiadomości
378 @ public static void odczytajTablice(String odpowiedz) { 1 usage
379     String[] entries = odpowiedz.split(regex: "#");
380     for (int i = 1; i < entries.length; i++) {
381         String[] entryDetails = entries[i].split(regex: ":");
382         if (entryDetails.length == 2) {
383             System.out.println(entryDetails[0] + " napisał: " + entryDetails[1]);
384         }
385     }
386 }
387 }

```

Server

```
1 import java.io.*;
2 import java.net.*;
3 import java.security.*;
4 import java.util.Base64;
5
6 public class Server {
7     public static void main(String[] args) {
8         ServerSocket server = null;
9         try {
10             // Tworzenie serwera nasłuchującego na porcie 1410
11             server = new ServerSocket(port: 1410);
12             server.setReuseAddress(true);
13             System.out.println("Serwer uruchomiony i oczekuje na połączenia...");
14
15             while (true) {
16                 // Akceptowanie połączenia od klienta
17                 Socket client = server.accept();
18                 System.out.println("Nowy klient połączony: " + client.getInetAddress().getHostAddress());
19                 ClientHandler clientSock = new ClientHandler(client);
20                 new Thread(clientSock).start();
21             }
22         } catch (IOException e) {
23             e.printStackTrace();
24         } finally {
25             if (server != null) {
26                 try {
27                     server.close();
28                 } catch (IOException e) {
29                     e.printStackTrace();
30                 }
31             }
32         }
33     }
34
35     private static class ClientHandler implements Runnable { 2 usages
36         private final Socket clientSocket; 8 usages
37         private SecurityUtils securityUtils; 25 usages
38         private PublicKey clientPublicKey; 6 usages
39
40         public ClientHandler(Socket socket) { 1 usage
41             this.clientSocket = socket;
42         }
43     }
```

```

43
44 public void run() {
45     PrintWriter out = null;
46     BufferedReader in = null;
47
48     try {
49         // Inicjalizacja narzędzi kryptograficznych
50         securityUtils = new SecurityUtils();
51         System.out.println("Inicjalizacja zabezpieczeń dla klienta: " +
52             clientSocket.getInetAddress().getHostAddress());
53
54         ObjectInputStream objIn = new ObjectInputStream(clientSocket.getInputStream());
55         ObjectOutputStream objOut = new ObjectOutputStream(clientSocket.getOutputStream());
56
57         // Krok 1: Wymiana kluczy publicznych
58         clientPublicKey = (PublicKey) objIn.readObject();
59         securityUtils.setPartnerPublicKey(clientPublicKey);
60         System.out.println("Otrzymano klucz publiczny od klienta");
61
62         objOut.writeObject(securityUtils.getPublicKey());
63         System.out.println("Wysłano klucz publiczny do klienta");
64
65         // Krok 2: Uwierzytelnienie klienta
66         byte[] challenge = securityUtils.generateChallenge();
67         objOut.writeObject(challenge);
68         System.out.println("Wysłano wyzwanie do klienta");
69
70         byte[] clientResponse = (byte[]) objIn.readObject();
71         boolean clientVerified = securityUtils.verifyChallenge(challenge, clientResponse, clientPublicKey);
72
73         if (!clientVerified) {
74             throw new SecurityException("Nie udało się zweryfikować tożsamości klienta!");
75         }
76         System.out.println("Zweryfikowano tożsamość klienta");
77
78         // Krok 3: Uwierzytelnienie serwera
79         byte[] clientChallenge = (byte[]) objIn.readObject();
80         byte[] serverResponse = securityUtils.signChallenge(clientChallenge);
81         objOut.writeObject(serverResponse);
82         System.out.println("Wysłano odpowiedź na wyzwanie klienta");
83
84         // Krok 4: Wysłanie klucza AES (tylko jeśli uwierzytelnienie się powiodło)
85         byte[] encryptedAESKey = securityUtils.encryptAESKey(clientPublicKey);
86         objOut.writeObject(encryptedAESKey);
87         System.out.println("Wysłano zaszyfrowany klucz AES");
88
89         out = new PrintWriter(clientSocket.getOutputStream(), true);
90         in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
91
92         String line;
93         while ((line = in.readLine()) != null) {
94             System.out.println("Otrzymano wiadomość: " + line);
95
96             String[] parts = securityUtils.decodeMessage(line);
97             String message = parts[0];
98             byte[] signature = Base64.getDecoder().decode(parts[1]);
99
100             if (!securityUtils.verify(message, signature, clientPublicKey)) {
101                 System.out.println("Ostrzeżenie: Nieprawidłowy podpis wiadomości!");
102                 continue;
103             }
104
105             String[] splitted = message.split(regex: "#|:");
106             String response = null;

```

```

108         switch (splitted[1]) {
109             case "login": {
110                 // Przekazanie wiadomości do serwera logowania
111                 try (Socket loginSocket = new Socket(host: "localhost", port: 2137);
112                     PrintWriter loginOut = new PrintWriter(loginSocket.getOutputStream(), autoFlush: true);
113                     BufferedReader loginIn = new BufferedReader(new InputStreamReader(loginSocket.getInputStream())) {
114                     loginOut.println(message);
115                     response = loginIn.readLine();
116                     if (response != null) {
117                         byte[] responseSignature = securityUtils.sign(response);
118                         out.println(securityUtils.encodeMessage(response, responseSignature));
119                     }
120                 }
121                 break;
122             }
123             case "register": {
124                 // Przekazanie wiadomości do serwera rejestracji
125                 try (Socket registerSocket = new Socket(host: "localhost", port: 2138);
126                     PrintWriter registerOut = new PrintWriter(registerSocket.getOutputStream(), autoFlush: true);
127                     BufferedReader registerIn = new BufferedReader(new InputStreamReader(registerSocket.getInputStream())) {
128                     registerOut.println(message);
129                     response = registerIn.readLine();
130                     if (response != null) {
131                         byte[] responseSignature = securityUtils.sign(response);
132                         out.println(securityUtils.encodeMessage(response, responseSignature));
133                     }
134                 }
135                 break;
136             }
137             case "tablica": {
138                 // Przekazanie wiadomości do serwera tablicy
139                 try (Socket tablicaSocket = new Socket(host: "localhost", port: 2139);
140                     PrintWriter tablicaOut = new PrintWriter(tablicaSocket.getOutputStream(), autoFlush: true);
141                     BufferedReader tablicaIn = new BufferedReader(new InputStreamReader(tablicaSocket.getInputStream())) {
142                     tablicaOut.println(message);
143                     response = tablicaIn.readLine();
144                     if (response != null) {
145                         byte[] responseSignature = securityUtils.sign(response);
146                         out.println(securityUtils.encodeMessage(response, responseSignature));
147                     }
148                 }
149                 break;
150             }
151             case "chat": {
152                 // Przekazanie wiadomości do serwera czatu
153                 try (Socket chatSocket = new Socket(host: "localhost", port: 2120);
154                     PrintWriter chatOut = new PrintWriter(chatSocket.getOutputStream(), autoFlush: true);
155                     BufferedReader chatIn = new BufferedReader(new InputStreamReader(chatSocket.getInputStream())) {
156                     chatOut.println(message);
157                     response = chatIn.readLine();
158                     if (response != null) {
159                         byte[] responseSignature = securityUtils.sign(response);
160                         out.println(securityUtils.encodeMessage(response, responseSignature));
161                     }
162                 }
163                 break;
164             }

```

```

165         case "FT0": {
166             // Przekazanie wiadomości do serwera transferu plików (out)
167             try (Socket FT0Socket = new Socket(host: "localhost", port: 3333);
168                 PrintWriter FT0Out = new PrintWriter(FT0Socket.getOutputStream(), autoFlush: true);
169                 BufferedReader FT0In = new BufferedReader(new InputStreamReader(FT0Socket.getInputStream())) {
170                 String packetToSend = message;
171                 int packetNo = Integer.parseInt(message.split(regex: "#|:")[7]);
172                 FT0Out.println(packetToSend);
173
174                 int packetsSended = 1;
175                 while (packetsSended < packetNo) {
176                     String nextPacket = in.readLine();
177                     String[] nextParts = securityUtils.decodeMessage(nextPacket);
178                     if (securityUtils.verify(nextParts[0], Base64.getDecoder().decode(nextParts[1]), clientPublicKey)) {
179                         FT0Out.println(nextParts[0]);
180                     }
181                     packetsSended++;
182                 }
183
184                 response = FT0In.readLine();
185                 if (response != null) {
186                     byte[] responseSignature = securityUtils.sign(response);
187                     out.println(securityUtils.encodeMessage(response, responseSignature));
188                 }
189             }
190             break;
191         }
192
193         case "FT1": {
194             // Przekazanie wiadomości do serwera transferu plików (in)
195             try (Socket FT1Socket = new Socket(host: "localhost", port: 9921);
196                 PrintWriter FT1Out = new PrintWriter(FT1Socket.getOutputStream(), autoFlush: true);
197                 BufferedReader FT1In = new BufferedReader(new InputStreamReader(FT1Socket.getInputStream())) {
198                 FT1Out.println(message);
199                 String receivedPacket = FT1In.readLine();
200
201                 if (receivedPacket != null) {
202                     byte[] responseSignature = securityUtils.sign(receivedPacket);
203                     out.println(securityUtils.encodeMessage(receivedPacket, responseSignature));
204
205                     int packetNo = Integer.parseInt(receivedPacket.split(regex: "#|:")[5]);
206                     int packetsSended = 1;
207
208                     while (packetsSended < packetNo) {
209                         receivedPacket = FT1In.readLine();
210                         responseSignature = securityUtils.sign(receivedPacket);
211                         out.println(securityUtils.encodeMessage(receivedPacket, responseSignature));
212                         packetsSended++;
213                     }
214                 }
215             }
216             break;
217         }
218     }

```

```

219     } catch (Exception e) {
220         System.out.println("Błąd podczas obsługi klienta: " + e.getMessage());
221         e.printStackTrace();
222     } finally {
223         try {
224             if (out != null) {
225                 out.close();
226             }
227             if (in != null) {
228                 in.close();
229                 clientSocket.close();
230             }
231         } catch (IOException e) {
232             e.printStackTrace();
233         }
234         System.out.println("Zakończono połączenie z klientem: " +
235             clientSocket.getInetAddress().getHostAddress());
236     }
237 }
238 }
239 }

```

Login

```
1  import java.io.*;
2  import java.net.*;
3  import java.sql.*;
4  import java.security.NoSuchAlgorithmException;
5
6  public class Login {
7      private static SecurityUtils securityUtils; 1 usage
8
9      public static void main(String[] args) throws IOException, SQLException, ClassNotFoundException, NoSuchAlgorithmException {
10         // Tworzenie serwera nasłuchującego na porcie 2137
11         ServerSocket loginSocket = new ServerSocket(port: 2137);
12         securityUtils = new SecurityUtils();
13         System.out.println("Serwer logowania uruchomiony na porcie 2137");
14
15         while (true) {
16             try {
17                 // Akceptowanie połączenia od klienta
18                 Socket clientSocket = loginSocket.accept();
19                 PrintWriter loginOut = new PrintWriter(clientSocket.getOutputStream(), autoFlush: true);
20                 BufferedReader loginIn = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
21
22                 // Odczytanie zakodowanej wiadomości od klienta
23                 String encodedMessage = loginIn.readLine();
24                 String[] parts = encodedMessage.split(regex: "#:");
25
26                 // Login i hasło są już w formie niezaszyfrowanej, bo Server.java je rozszyfrował
27                 String login = parts[3];
28                 String haslo = parts[5];
29                 String odpowiedz;
30
31                 // Połączenie z bazą danych i weryfikacja użytkownika
32                 try (Connection con = DriverManager.getConnection(url: "jdbc:mysql://localhost:3306/ts", user: "root", password: "karol1")) {
33                     Class.forName(className: "com.mysql.cj.jdbc.Driver");
34                     PreparedStatement statement = con.prepareStatement(sql: "select * from users where login=? and haslo=?");
35                     statement.setString(parameterIndex: 1, login);
36                     statement.setString(parameterIndex: 2, haslo);
37                     ResultSet result = statement.executeQuery();
38
39                     // Sprawdzenie, czy użytkownik istnieje w bazie danych
40                     if (result.next()) {
41                         odpowiedz = "status:OK";
42                         System.out.println("Udane logowanie dla: " + login);
43                     } else {
44                         odpowiedz = "status:NO";
45                         System.out.println("Nieudane logowanie dla: " + login);
46                     }
47                 } catch (Exception e) {
48                     e.printStackTrace();
49                     odpowiedz = "status:NO";
50                 }
51
52                 // Wysłanie odpowiedzi do klienta
53                 loginOut.println(odpowiedz);
54             } catch (Exception e) {
55                 e.printStackTrace();
56             }
57         }
58     }
59 }
```


Register

```
1 import java.io.*;
2 import java.net.*;
3 import java.sql.*;
4
5 public class Register {
6     public static void main(String[] args) throws IOException, SQLException, ClassNotFoundException {
7         // Tworzenie serwera nasłuchującego na porcie 2138
8         ServerSocket registerSocket = new ServerSocket(port: 2138);
9         System.out.println("Serwer rejestracji uruchomiony na porcie 2138");
10
11         while (true) {
12             try {
13                 // Akceptowanie połączenia od klienta
14                 Socket clientSocket = registerSocket.accept();
15                 PrintWriter registerOut = new PrintWriter(clientSocket.getOutputStream(), autoFlush: true);
16                 BufferedReader registerIn = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
17
18                 // Odczytanie wiadomości rejestracyjnej od klienta
19                 String message = registerIn.readLine();
20                 System.out.println("Otrzymano żądanie rejestracji: " + message);
21                 String[] odebrane = message.split(regex: "#");
22                 String odpowiedz;
23
24                 // Połączenie z bazą danych i weryfikacja użytkownika
25                 try (Connection con = DriverManager.getConnection(url: "jdbc:mysql://localhost:3306/ts", user: "root", password: "karol1")) {
26                     Class.forName(className: "com.mysql.cj.jdbc.Driver");
27                     PreparedStatement checkStmt = con.prepareStatement(sql: "select * from users where login=?");
28                     checkStmt.setString(parameterIndex: 1, odebrane[3]);
29                     ResultSet result = checkStmt.executeQuery();
30
31                     // Sprawdzenie, czy użytkownik już istnieje
32                     if (!result.next()) {
33                         // Rejestracja nowego użytkownika
34                         PreparedStatement insertStmt = con.prepareStatement(sql: "INSERT INTO users (login,haslo) VALUES (?,?)");
35                         insertStmt.setString(parameterIndex: 1, odebrane[3]);
36                         insertStmt.setString(parameterIndex: 2, odebrane[5]);
37                         insertStmt.executeUpdate();
38                         odpowiedz = "status:OK";
39                         System.out.println("Zarejestrowano nowego użytkownika: " + odebrane[3]);
40                     } else {
41                         // Użytkownik już istnieje
42                         odpowiedz = "status:NO";
43                         System.out.println("Próba rejestracji istniejącego użytkownika: " + odebrane[3]);
44                     }
45                 } catch (Exception e) {
46                     e.printStackTrace();
47                     odpowiedz = "status:NO";
48                 }
49
50                 // Wysłanie odpowiedzi do klienta
51                 registerOut.println(odpowiedz);
52             } catch (Exception e) {
53                 e.printStackTrace();
54             }
55         }
56     }
57 }
```

Tablica

```
1  import java.io.*;
2  import java.net.*;
3  import java.sql.*;
4  import java.security.NoSuchAlgorithmException;
5
6  public class Tablica {
7      private static SecurityUtils securityUtils; 1 usage
8
9      public static void main(String[] args) throws IOException, SQLException, ClassNotFoundException, NoSuchAlgorithmException {
10         // Tworzenie serwera nasłuchującego na porcie 2139
11         ServerSocket tablicaSocket = new ServerSocket(port: 2139);
12         securityUtils = new SecurityUtils();
13         System.out.println("Serwer tablicy uruchomiony na porcie 2139");
14
15         while (true) {
16             try {
17                 // Akceptowanie połączenia od klienta
18                 Socket clientSocket = tablicaSocket.accept();
19                 PrintWriter tablicaOut = new PrintWriter(clientSocket.getOutputStream(), autoFlush: true);
20                 String odpowiedz = "status:OK#tresc:#";
21
22                 // Połączenie z bazą danych i pobranie ostatnich 10 wpisów z tablicy
23                 try (Connection con = DriverManager.getConnection(url: "jdbc:mysql://localhost:3306/ts", user: "root", password: "karol1")) {
24                     Class.forName(className: "com.mysql.cj.jdbc.Driver");
25                     Statement statement = con.createStatement();
26                     String loginTworzacego;
27                     String trescPosta;
28                     ResultSet result = statement.executeQuery(sql: "select * from tablica order by id desc LIMIT 10");
29                     int count = 0;
30                     while (result.next()) {
31                         count++;
32                         loginTworzacego = result.getString(columnLabel: "autor");
33                         trescPosta = result.getString(columnLabel: "tresc");
34                         odpowiedz += loginTworzacego + ".*" + trescPosta + "#";
35                     }
36                     System.out.println("Pobrano " + count + " wpisów z tablicy");
37                 } catch (Exception e) {
38                     e.printStackTrace();
39                     odpowiedz = "status:NO";
40                 }
41
42                 // Wysłanie odpowiedzi do klienta
43                 tablicaOut.println(odpowiedz);
44             } catch (Exception e) {
45                 e.printStackTrace();
46             }
47         }
48     }
49 }
```

Chat

```
1 import java.io.*;
2 import java.net.*;
3 import java.sql.*;
4 import java.security.NoSuchAlgorithmException;
5
6 public class Chat {
7     private static SecurityUtils securityUtils; 1usage
8
9     public static void main(String[] args) throws IOException, SQLException, ClassNotFoundException, NoSuchAlgorithmException {
10         // Tworzenie serwera nasłuchującego na porcie 2120
11         ServerSocket chatSocket = new ServerSocket(port: 2120);
12         securityUtils = new SecurityUtils();
13         System.out.println("Serwer czatu uruchomiony na porcie 2120");
14
15         while (true) {
16             try {
17                 // Akceptowanie połączenia od klienta
18                 Socket clientSocket = chatSocket.accept();
19                 PrintWriter chatOut = new PrintWriter(clientSocket.getOutputStream(), autoFlush: true);
20                 BufferedReader chatIn = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
21
22                 // Odczytanie wiadomości od klienta
23                 String message = chatIn.readLine();
24                 System.out.println("Otrzymano wiadomość: " + message);
25
26                 // Wiadomość jest już odszyfrowana przez Server.java
27                 String[] parts = message.split(regex: "#|:");
28                 String odpowiedz;
29
30                 // Połączenie z bazą danych i dodanie wpisu do tablicy
31                 try (Connection con = DriverManager.getConnection(url: "jdbc:mysql://localhost:3306/ts", user: "root", password: "karol1")) {
32                     Class.forName(className: "com.mysql.cj.jdbc.Driver");
33                     PreparedStatement insertStmt = con.prepareStatement(sql: "INSERT INTO tablica (autor, tresc) VALUES (?, ?)");
34                     insertStmt.setString(parameterIndex: 1, parts[3]); // autor
35                     insertStmt.setString(parameterIndex: 2, parts[5]); // tresc
36                     insertStmt.executeUpdate();
37
38                     // Sprawdzenie liczby wpisów i usunięcie najstarszego, jeśli jest ich więcej niż 10
39                     Statement countStmt = con.createStatement();
40                     ResultSet result = countStmt.executeQuery(sql: "SELECT COUNT(*) FROM tablica");
41                     result.next();
42                     int count = result.getInt(columnIndex: 1);
43
44                     if (count > 10) {
45                         Statement deleteStmt = con.createStatement();
46                         deleteStmt.executeUpdate(sql: "DELETE FROM tablica WHERE ID IN (SELECT ID FROM (SELECT ID FROM tablica ORDER BY ID ASC LIMIT 1) AS t)");
47                     }
48
49                     odpowiedz = "status:OK";
50                     System.out.println("Dodano wpis od użytkownika: " + parts[3]);
51                 } catch (Exception e) {
52                     e.printStackTrace();
53                     odpowiedz = "status:NO";
54                 }
55
56                 // Wyśłanie odpowiedzi do klienta
57                 chatOut.println(odpowiedz);
58             } catch (Exception e) {
59                 e.printStackTrace();
60             }
61         }
62     }
63 }
```

FileTransferIn

```
1 import java.io.*;
2 import java.net.*;
3 import java.sql.SQLException;
4 import java.util.Base64;
5 import java.util.StringJoiner;
6
7 public class FileTransferIn {
8     public static void main(String[] args) throws IOException, SQLException, ClassNotFoundException {
9         // Tworzenie serwera nasłuchującego na porcie 9921
10        ServerSocket ftiSocket = new ServerSocket(port: 9921);
11        System.out.println("Serwer transferu plików (in) uruchomiony na porcie 9921");
12        System.out.println("Oczekiwanie na ządania w katalogu: C:\\Serwer-plikow\\");
13
14        while (true) {
15            try {
16                // Akceptowanie połączenia od klienta
17                Socket clientSocket = ftiSocket.accept();
18                System.out.println("\n== Nowe połączenie od: " + clientSocket.getInetAddress().getHostAddress() + " ==");
19                PrintWriter FTIOut = new PrintWriter(clientSocket.getOutputStream(), autoFlush: true);
20                BufferedReader FTIIn = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
21
22                // Odczytanie ządania od klienta
23                String receivedPacket = FTIIn.readLine();
24                String requestedFile = receivedPacket.split(regex: "#.")[5];
25                System.out.println("Otrzymano ządanie pliku: " + requestedFile);
26
27                // Ścieżka do pliku na serwerze
28                String path = "C:\\Serwer-plikow\\" + requestedFile;
29                File file = new File(path);
30                StringJoiner newFTIPacket = new StringJoiner(delimiter: "#");
31
32                if (!file.exists()) {
33                    // Obsługa przypadku, gdy plik nie istnieje
34                    System.out.println("BŁĄD: Plik nie istnieje: " + requestedFile);
35                    newFTIPacket.add("status:NO");
36                    newFTIPacket.add("filename:_");
37                    newFTIPacket.add("packetsNo:0");
38                    newFTIPacket.add("content:_");
39                    FTIOut.println(newFTIPacket.toString());
40                } else {
41                    // Obsługa przypadku, gdy plik istnieje
42                    System.out.println("Znaleziono plik: " + file.getName());
43                    System.out.println("Rozmiar pliku: " + file.length() + " bajtów");
44
45                    long fileSize = file.length();
46                    int packetCount = (int) Math.ceil((double) fileSize / 1024);
47                    System.out.println("Liczba pakietów do wysłania: " + packetCount);
48
49                    byte[] buffer = new byte[1024];
50                    InputStream fileInputStream = new FileInputStream(file);
51                    int bytesRead;
52                    int totalBytesSent = 0;
53
54                    for (int i = 0; i < packetCount; i++) {
55                        // Odczytywanie danych z pliku i wysyłanie ich do klienta
56                        bytesRead = fileInputStream.read(buffer);
57                        totalBytesSent += bytesRead;
58                        byte[] packetToAdd = new byte[bytesRead];
59                        System.arraycopy(buffer, srcPos: 0, packetToAdd, destPos: 0, bytesRead);
60                        newFTIPacket.add("status:OK");
61                        newFTIPacket.add("filename:" + file.getName());
62                        newFTIPacket.add("packetsNo:" + packetCount);
63                        newFTIPacket.add("content:" + Base64.getEncoder().encodeToString(packetToAdd));
64
65                        FTIOut.println(newFTIPacket.toString());
66                        System.out.println("Wysłano pakiet " + (i+1) + "/" + packetCount +
67                            " (" + totalBytesSent + "/" + fileSize + " bajtów)");
68                        newFTIPacket = new StringJoiner(delimiter: "#");
69                    }
70                    fileInputStream.close();
71                    System.out.println("Transfer zakończony pomyślnie");
72                }
73                System.out.println("== Zakończono połączenie ==\n");
74            } catch (Exception e) {
75                // Obsługa błędów podczas transferu
76                System.out.println("BŁĄD podczas transferu: " + e.getMessage());
77                e.printStackTrace();
78            }
79        }
80    }
81}
```

FileTransferOut

```
1 import java.io.*;
2 import java.net.ServerSocket;
3 import java.net.Socket;
4 import java.sql.SQLException;
5 import java.util.ArrayList;
6 import java.util.Base64;
7
8 public class FileTransferOut {
9     public static void main(String[] args) throws IOException, SQLException, ClassNotFoundException {
10         // Tworzenie serwera nasłuchującego na porcie 3333
11         ServerSocket FTOSocket = new ServerSocket(port: 3333);
12         System.out.println("Serwer transferu plików (out) uruchomiony na porcie 3333");
13         System.out.println("Katalog docelowy: C:\\Serwer-plikow\\");
14
15         while (true) {
16             try {
17                 // Akceptowanie połączenia od klienta
18                 Socket clientSocket = FTOSocket.accept();
19                 System.out.println("\n=== Nowe połączenie od: " + clientSocket.getInetAddress().getHostAddress() + " ===");
20
21                 // Inicjalizacja strumieni do komunikacji z klientem
22                 PrintWriter ftoOut = new PrintWriter(clientSocket.getOutputStream(), autoFlush: true);
23                 BufferedReader FTIn = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
24                 ArrayList<String> packetsReceivedList = new ArrayList<String>();
25
26                 // Odczytanie pierwszego pakietu od klienta
27                 String receivedPacket = FTIn.readLine();
28                 String[] initialPacketInfo = receivedPacket.split(regex: "#[.]*");
29                 String fileName = initialPacketInfo[5];
30                 int packetNo = Integer.parseInt(initialPacketInfo[7]);
31
32                 System.out.println("Rozpoczęto odbieranie pliku: " + fileName);
33                 System.out.println("Oczekiwana liczba pakietów: " + packetNo);
34
35                 // Odbieranie kolejnych pakietów
36                 int packetsReceived = 1;
37                 System.out.println("Odebrano pakiet: 1/" + packetNo);
```

```
38
39                 while (packetsReceived < packetNo) {
40                     packetsReceivedList.add(receivedPacket);
41                     receivedPacket = FTIn.readLine();
42                     packetsReceived++;
43                     System.out.println("Odebrano pakiet: " + packetsReceived + "/" + packetNo);
44                 }
45                 if (packetNo == 1) {
46                     packetsReceivedList.add(receivedPacket);
47                 }
48                 System.out.println("Odebrano wszystkie pakiety. Rozpocznym składanie pliku...");
49
50                 // Składanie pliku z odebranych pakietów
51                 String[] splittedPacket;
52                 String newFileName = "";
53                 ArrayList<byte[]> newFileContentList = new ArrayList<>();
54                 long totalSize = 0;
55
56                 for (int i = 0; i < packetsReceivedList.size(); i++) {
57                     splittedPacket = packetsReceivedList.get(i).split(regex: "#[.]*");
58                     byte[] receivedContent = Base64.getDecoder().decode(splittedPacket[9]);
59                     totalSize += receivedContent.length;
60                     newFileContentList.add(receivedContent);
61                     if (i == 0) {
62                         newFileName = splittedPacket[5];
63                     }
64                 }
65
66                 // Zapisanie pliku na serwerze
67                 String newPath = "C:\\Serwer-plikow\\" + newFileName;
68                 File file = new File(newPath);
69                 String response;
70                 boolean isExisting = file.exists();
71
```

```

70         boolean isExisting = file.exists();
71
72         if (newFileName.equals("")) {
73             response = "status:NO";
74             System.out.println("BłAD: Otrzymano pustą nazwę pliku");
75         } else if (isExisting) {
76             response = "status:NO";
77             System.out.println("BłAD: Plik już istnieje w lokalizacji: " + newPath);
78         } else {
79             try (OutputStream outputStream = new FileOutputStream(file)) {
80                 for (byte[] content : newFileContentList) {
81                     outputStream.write(content);
82                 }
83                 response = "status:OK";
84                 System.out.println("Sukces: Zapisano plik " + newFileName);
85                 System.out.println("Lokalizacja: " + newPath);
86                 System.out.println("Rozmiar: " + totalSize + " bajtów");
87             } catch (IOException e) {
88                 response = "status:NO";
89                 System.out.println("BłAD podczas zapisywania pliku: " + e.getMessage());
90                 e.printStackTrace();
91             }
92         }
93
94         // Wysłanie odpowiedzi do klienta
95         ftoOut.println(response);
96         System.out.println("=== Zakończono połączenie ===\n");
97     } catch (Exception e) {
98         // Obsługa błędów podczas obsługi połączenia
99         System.out.println("BłAD podczas obsługi połączenia: " + e.getMessage());
100         e.printStackTrace();
101     }
102 }
103 }
104 }

```