Dokumentacja wstępna projektu TIN „Rozgłaszacz internetowy”.

Autorzy: Joanna Raczyńska, Kamil Kacperski, Michał Mudel, Wojciech Zieliński

Prowadzący: dr inż. Piotr Gawkowski

# 1.Wstęp

Niniejszy dokument przedstawia projekt wstępny programu pt. „Rozsyłacz pocztowy” realizowanego w ramach przedmiotu TIN na wydziale EiTI Politechniki Warszawskiej. Głównym założeniem projektu jest stworzenie aplikacji, umożliwiającej proste i szybkie wysyłanie wiadomości – powiadomień do wybranej grupy odbiorców. Każdy klient posiada własne grupy i szablony wiadomości. Na ich podstawie wysyła wiadomości mailowe.

Użytkownik będzie miał możliwość rozsyłania wiadomości e-mail do zdefiniowanych wcześniej grup odbiorców (na przykład prowadzący mają dostęp do list mailingowych poszczególnych przedmiotów). Główną funkcjonalnością programu będą zdefiniowane wcześniej szablony wiadomości, które umożliwią bardzo szybkie utworzenie standardowych wiadomości (na przykład informacja o spóźnieniu czy odwołaniu spotkania). Użytkownik wybiera w oknie aplikacji wcześniej zdefiniowaną grupę, do której chce wysłać wiadomość, następnie dodaje wymagane do uzupełnienia szczegóły wiadomości (na przykład godzinę na którą chce przełożyć spotkanie) i klika wyślij. W razie potrzeby istnieje również możliwość modyfikacji domyślnego szablonu wiadomości czy też modyfikacja grup odbiorców.

Aplikacja zorganizowana będzie w strukturze klient – serwer. Każdy egzemplarz aplikacji użytkownika będzie klientem, który łączył się będzie z publicznym serwerem. Komunikacja między klientem a serwerem oparta będzie o protokół TCP i nadbudowanym na nim autorskim protokołem służącym do komunikacji w tworzonej aplikacji. Na serwerze przechowywane będą zdefiniowane grupy użytkowników (pogrupowane odpowiednio adresy e-mail) oraz szablony wiadomości. Każdy klient będzie mógł korzystać z danych dostępnych na serwerze, a także będzie miał możliwość ich modyfikacji. Autoryzacja odbywać się będzie na podstawie unikalnego loginu i hasła klienta. W celu zachowania poufności danych, połączenie między klientem a serwerem będzie szyfrowane. Serwer będzie wysyłał wiadomości w oparciu o protokół SMTP. Podstawowy schemat organizacji struktury programu przedstawiono na rysunku.



## Zarys implementacji

Aplikacja kliencka zorganizowana zostanie jako aplikacja okienkowa. Głównym założeniem jej interfejsu ma być prostota i szybka intuicyjna obsługa. Umożliwiać ma ona logowanie (uwierzytelnienie użytkownika), zarządzanie grupami odbiorców oraz wysyłanie wiadomości. Podstawową funkcjonalnością będzie wysyłanie wiadomości wcześniej zdefiniowanych jako szablony na serwerze. Wtedy konieczne będzie jedynie uzupełnienie niezbędnych danych (np. informacji o czasie spóźnienia czy godzinie na którą chcemy przełożyć spotkanie), co zdecydowanie przyspieszy proces komunikacji. Dla każdego klienta, po połączeniu się do serwera zostanie utworzony odrębny wątek kliencki, który zarządzał będzie dalszą komunikacją z serwerem.

Połączenie obsługiwać będzie wiele współpracujących ze sobą wątków. Każdy z nich wykonywał będzie ściśle określone zadanie, niezależnie od innych.

* Wątek nasłuchujący na połączenia – jest to wątek serwera, który nasłuchiwał będzie na połączenia od klientów.
* Wątek kliencki –wątek jaki powstaje po podłączeniu się klienta do serwera. Odpowiada on za autoryzację klienta, komunikację z wątkiem dostępu do danych oraz przesyłanie wiadomości jakie chce wysłać klient do kolejki wiadomości wątku rozsyłającego serwera
* Wątek rozsyłający serwera – odpowiadał będzie za rozsyłanie wiadomości pobranych z kolejki wiadomości za pomocą protokołu SMTP
* Wątek zarządzający – będzie nasłuchiwał na sygnały, że konieczne jest już zakończenie pracy. Będzie odpowiedzialny za zakończenie wszystkich wątków

# 2. Słownik dziedziny problemu

2.1 Klient – użytkownik systemu korzystający z części klienckiej aplikacji.

2.2 Administrator – użytkownik systemu zarządzający aplikacją po stronie serwera.

2.3 Grupa – zbiór adresów mailowych, będący podmiotem wysyłania wiadomości.

2.4 Szablon wiadomości – treść wiadomości mogąca posiadać pola wymagające uzupełnienia

2.5 Wiadomość – treść wiadomości z wypełnionymi wszystkimi polami, gotowa do wysłania.

2.6 Adres – adres mailowy który jest przechowywany w grupie.

2.7 Pole szablonu – tekst który musi zostać uzupełniony w szablonie wiadomości przed jej wysłaniem.

2.8 Sesja klienta – połączenie z danym klientem w danym czasie.

2.9 Aplikacja kliencka – aplikacja uruchamiana na komputerze klienta.

2.10 Aplikacja serwer – aplikacja uruchamiana przez administratora na serwerze, obsługująca klientów.

# 3. Wymagania funkcjonalne

3.1 Klient loguję się do systemu.

3.2 Klient tworzy nowy szablon wiadomości

3.3 Klient modyfikuje wcześniej utworzony szablon wiadomości.

3.4 Klient usuwa szablon wiadomości.

3.5 Klient wyświetla nazwy wszystkich szablonów wiadomości.

3.6 Klient wyświetla całą treść wybranego szablonu wiadomości.

3.7 Klient tworzy nową grupę.

3.8 Klient usuwa grupę.

3.9 Klient wyświetla grupę.

3.10 Klient dodaje adres do grupy.

3.11 Klient usuwa adres z grupy.

3.12 Klient wysyła wiadomość.

3.13 Klient zmienia swoje hasło.

3.13 Klient wylogowuje się z systemu.

3.14 Administrator wyświetla wszystkie obecnie otwarte sesje klientów.

3.15 Administrator kończy wybraną sesję z klientem.

3.16 Administrator blokuje wybranego klienta.

3.17 Administrator odblokowuje wybranego klienta.

3.18 Administrator tworzy nowego klienta.

3.19 Administrator usuwa wybranego klienta.

3.20 Administrator modyfikuje konto wybranego klienta.

# 4. Wymagania niefunkcjonalne

4.1 Aplikacja serwera będzie uruchamiana na platformach Windows 7, Windows 8, Windows 10, Windows Server 2012.

4.2 Aplikacja kliencka zostanie wykonana w technologii Windows Presentation Foundation, C# i .NET 4.6.

4.3 Aplikacja serwerowa zostanie wykonana za pomocą języka C++ 14.

4.4 Sesja kliencka jest kończona po 5 minutach braku komunikacji.

4.5 Sesja kliencka jest kończona jeśli wysłane przez klienta żądanie jest niezgodne z protokołem ASIA.

4.6 Sesja kliencka jest kończona jeśli wysyła więcej niż 10 żądań na sekundę.

4.7 Nie ma możliwości jednoczesnego nawiązania połączenia z serwerem przez liczbę użytkowników większą niż 1000.

4.8 W przypadku 4.7 konto klienta jest blokowane.

4.9 Natychmiast po utracie łączności wszystkie wątki składające się na komunikacje klienta z serwerem są zakańczane a gniazda zamykane.

4.10 Jako baza danych zostanie wykorzystany MS SQL Server 2014 Standard Edition with Service Pack 1.

# 5. Model komunikacji między modułami aplikacji

# 6. Opis komunikacji komponentów serwera

Funkcja main konfiguruje statyczny obiekt DependencyInjector, za pomocą którego pobierane będą wszelkie obecne w aplikacji oraz będą tworzone nowe instancje obiektów. Funkcja main uruchamia obiekt RootManager. RootManager uruchamia obiekty SessionsListener oraz SmtpLayer na osobnych wątkach. SessionsListener przekazuje RootManagerowi informacje o nowych połączeniach i dla każdego z nich które są dozwolone uruchamia obiekty ClientSession. Obiekty ClientSession uruchamiają obiekty Cipher które służą do szyfrowania i obiekty TcpLayer które służą do komunikacji z klientem. ClientSession komunikuje się z SmtpLayer poprzez kolekję obiektów MessagesQueue. Obiekty ClientSession i obiekt RootManager mają dostęp do DAL ze swoich wątków.

# 7. Model interfejsów i klas aplikacji

# 8. Opis protokołu SMTP

Dany protokół komunikacyjny służy do wysyłania wiadomości tekstowych złożonych z kopert oraz treści właściwej. Kanał transmisyjny SMTP bazuje na ustawionym wcześniej połączeniu TCP . Jest on połączeniem full-duplex które korzysta z domyślnego portu o numerze 25. Protokół SMTP składa się z 7-bitowych ciągów znaków ASCII , które są transmitowane za pomocą protokołu TCP w strumieniu danych podzielonym na 8-bitowe oktety. Każdy znak jest przesyłany jako 8-bitowy ciąg z najstarszym bitem ustawionym na 0.

**Schemat działania połączenia:**

Wątek rozsyłający zwany dalej klientem ustanawia połączenie z serwerem SMTP i czeka aż ten odeśle komunikat 220 READY FOR MAIL. Po odebraniu komunikatu klient wysyła komendę HELO a serwer odpowiada kodem 250 oraz własnym identyfikatorem. Po ustanowieniu połączenia wątek może w każdym momencie zakończyć połączenie.

Wymiana poczty rozpoczyna się poleceniem MAIL ( definiującym nadawcę ) FROM <adres > ( pod ten adres wysłane zostaną wszystkie informacje o błędach ) które powiadamia serwer, że nadchodzi nowa transakcja i powinien on wyczyścić tablice stanów i bufory danych. Po pomyślnym wykonaniu plecenia MAIL klient wysyła serię poleceń RCPT <adres> ( identyfikując odbiorcę lub wielu odbiorców danej wiadomości ) i czeka na ich potwierdzenie przez serwer. Następnie nadawca wysyła polecenie DATA określające gotowość do przesłania całej wiadomości. Serwer odpowiada komunikatem 354 Start mail input i określa ciąg znaków odpowiadających zakończeniu transmisji. Zazwyczaj tym ciągiem jest linia zawierająca tylko kropkę co odpowiada podanej sekwencji znaków - <CR><LF>. <CR><LF>.

Gdy klient kończy wysyłanie wiadomości do danego odbiorcy, może wysłać plecenie TURN, które odwraca połączenie i sprawia że odbiorca przejmuje nad nim kontrolę. Zakończenie połączenia można wykonać wysyłając polecenie QUIT i czekając na odpowiedzą 221 które pozwala na rozłączenie.

# 9. Opis autorskiego protokołu o nazwie ASIA (Automated Sending Information to Addresses)

|  |
| --- |
| KOD KOMUNIKATU |
| DŁUGOŚĆ DANEJ | |
| DANA  ..... | |
| ..... | |
| DŁUGOŚĆ DANEJ | |
| DANA  ...... | |

2 BAJTY

## 9.1 Typy komunikatów

Każdy komunikat posiada swój kod komunikatu po którym jest identyfikowany Kod zajmuje 1 bajt. Każdy komunikat posiada określoną liczbę danych. Pozostała część komunikatu zawiera dane, które same określają swoją długość. Oznaczenie {...} określa kolekcję danych, której długość jest określona w niej samej.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Numer | Nazwa | | Inicjator | | Odpowiedź | | Kod | | Dane |
| 9.1.1 | AUTH | | Serwer | | AUTH | | 0x00 | | SALT |
| Serwer wysyła salt potrzebny klientowi do bezpiecznego przesłania skrótu hasła (przesyłamy skrót skrótu) | | | | | | | | |
| 9.1.2 | AUTH | | Klient | | ACK | | 0x00 | | LOGIN, PASSWHASH |
| Klient wysyła dane do autoryzacji | | | | | | | | |
| 9.1.3 | ACK | | Serwer | | - | | 0x01 | | - |
| Serwer potwierdza poprzednie żądanie klienta | | | | | | | | |
| 9.1.4 | MSGCREATE | | Klient | | MSGGETONE | | 0x02 | | MSGTMPLT, MSGNAME |
| Klient tworzy nowy szablon wiadomości | | | | | | | | |
| 9.1.5 | MSGMODIFY | | Klient | | ACK | | 0x03 | | MSGID, MSGTMPLT, MSGNAME |
| Klient modyfikuje szablon wiadomości | | | | | | | | |
| 9.1.6 | MSGDELETE | | Klient | | ACK | | 0x04 | | MSGID |
| Klient usuwa szablon wiadomości | | | | | | | | |
| 9.1.7 | MSGGETALL | | Klient | | MSGGETALL | | 0x05 | | - |
| Klient pobiera wszystkie szablony wiadomosci | | | | | | | | |
| 9.1.8 | MSGGETALL | | Serwer | | - | | 0x05 | | {MSGID, MSGNAME} |
| Serwer wysyła wszystkie szablony wiadomosci | | | | | | | | |
| 9.1.9 | MSGGETONE | | Klient | | MSGGETONE | | 0x06 | | MSGID |
| Klient pobiera wybraną wiadomość | | | | | | | | |
| 9.1.10 | MSGGETONE | | Serwer | | - | | 0x06 | | MSGID, MSGTMPLT, MSGNAME |
| Serwer wysyła jedną wiadomość | | | | | | | | |
| 9.1.11 | GRPCREATE | | Klient | | GRPGETONE | | 0x07 | | GRPNAME |
| Klient tworzy grupę | | | | | | | | |
| 9.1.12 | GRPDELETE | | Klient | | ACK | | 0x08 | | GRPID |
| Klient usuwa grupę | | | | | | | | |
| 9.1.13 | GRPGETONE | | Klient | | GRPGETONE | | 0x09 | | GRPID |
| Klient pobiera jedną grupę | | | | | | | | |
| 9.1.14 | GRPGETONE | | Serwer | | - | | 0x09 | | GRPID, GRPNAME, {ADDRID, ADDRVAL} |
| Serwer wysyła jedną grupę | | | | | | | | |
| 9.1.15 | GRPGETALL | | Klient | | GRPGETALL | | 0x0A | | - |
| Klient pobiera wszystkie grupy | | | | | | | | |
| 9.1.16 | GRPGETALL | | Serwer | | - | | 0x0A | | {GRPID, GRPNAME} |
| Serwer wysyła wszystkie grupy | | | | | | | | |
| 9.1.17 | GRPADRADD | | Klient | | ACK | | 0x0B | | GRPID, ADDRVAL |
| Klient dodaje adres do grupy | | | | | | | | |
| 9.1.18 | GRPADRRMV | | Klient | | ACK | | 0x0C | | GRPID, ADDRID |
| Klient usuwa adres z grupy | | | | | | | | |
| 9.1.19 | SEND | | Klient | | ACK | | 0x0D | | GRPID, MSGID, {VAL} |
| Klient wysyła wiadomość | | | | | | | | |
| 9.1.20 | PSSWCHG | | Klient | | ACK | | 0x0E | | PASSWHASH, NEWPASSWHASH |
| Klient wysyła skróty starego i nowego hasła w celu jego zmiany | | | | | | | | |
| 9.1.21 | ERRUNAUTH | | Serwer | | - | | 0x0F | | - |
| Serwer wysyła informację o odmowie wykonania żądania – klient nie jest zautoryzowany. | | | | | | | | |
| 9.1.22 | ERRBUSY | | Serwer | | - | | 0x10 | | - |
| Serwer wysyła informację o odmowie połączenia – za dużo podłączonych użytkowników. | | | | | | | | |
| 9.1.23 | ERRLOGIN | | Serwer | | - | | 0x11 | | - |
| Błąd przy próbie logowania – zły login lub hasło | | | | | | | | |
| 9.1.24 | ERRBADREQ | | Serwer | | - | | 0x12 | | - |
| Serwer wysyła informację o odmowie – w protokole nie ma żądania o podanym numerze | | | | | | | | |
| 9.1.25 | ERRBADCONTENT | | Serwer | | - | | 0x13 | | - |
| Serwer wysyła informację o odmowie - niewłaściwy format przesłanych danych | | | | | | | | |
| 9.1.26 | ERRGROUP | Serwer | | - | | 0x14 | | - | |
| Błąd – nie ma grupy o zadanej nazwie | | | | | | | | |
| 9.1.27 | ERRLOGIN | Serwer | | - | | 0x15 | | - | |
| Błąd – nie ma szablonu o zadanej nazwie | | | | | | | | |

Każdy typ danych posiada pole określające długość danych. Pojęcie długości jest rozumiane zależnie od danej – może być to długość w bajtach, może być to liczba danych zawartych w tej danej (tylko dla danych złożonych). Długość może być też określona przez typ danej – wtedy wartości zawarte w polu DŁUGOŚĆ DANEJ nie mają znaczenia. Dane wystepuję w pakiecie w kolejności w jakiej są określone w tabeli 9.1.

|  |  |  |  |
| --- | --- | --- | --- |
| Numer | Nazwa | Opis | Typ |
| 9.2.1 | SALT | 32 bajtowy salt do funkcji skrótu | ASCII |
| 9.2.2 | LOGIN | Login klienta | ASCII |
| 9.2.3 | PASSWHASH | Hash hasła klienta | ASCII |
| 9.2.4 | MSGTMPLT | Treść wiadomości. Wypełnialne pola wiadomości są oznaczane w następujący sposób - „{nazwapola}”. Parzyste ciągi znaków „{„ są w szablonie oznaczają dwa razy krótsze ciągi znaków „{„. Dla „}” analogicznie | ASCII |
| 9.2.5 | MSGNAME | Nazwa szablonu wiadomości | ASCII |
| 9.2.6 | MSGID | ID szablonu wiadomości | INT |
| 9.2.7 | GRPNAME | Nazwa grupy | ASCII |
| 9.2.8 | GRPID | ID grupy | INT |
| 9.2.9 | ADDRVAL | Adres e-mail | ASCII |
| 9.2.10 | ADDRID | ID adresu e-mail | INT |
| 9.2.11 | VAL | Wartość pola szablonu wiadomości. | ASCII |
| 9.2.12 | NEWPASSWHASH | Hash nowego hasła klienta | ASCII |

Każda kolekcja danych w polu DŁUGOŚĆ DANEJ zawiera liczbę elementów w kolekcji które zawiera. Wszystkie dane wysyłane w komunikacie są podane w kolejności podanej w tabeli 9.1. Wszystkie pola wiadomości VAL wystepują w takiej kolejności w jakiej występują w szablonie wiadomości.

Przy autoryzacji klienta używana będzie funkcja skrótu DJBhash. Po połączeniu moduł ClientSession odsyłał będzie klientowi salt, który klient dołączy do hasła przed hashowaniem. Jako że cały czas stosujemy jedną funkcję skrótu, dodany salt zabezpieczał będzie przed atakami częstotliwościowymi.

## 9.2 Diagram stanów modułu ClientSession

# 

# 10. Opis protokołu szyfrującego

**10.1 Opis modułu Cipher**

Moduł Cipher odpowiedzialny będzie za zaszyfrowywanie oraz odszyfrowywanie przesyłanych przez niego danych. Podstawowym sposobem szyfrowania będzie szyfrowanie XOR. Będzie on udostępniał interfejs dla modułu TcpLayer, ClientSession oraz RootManager. Tworzony będzie jeden moduł Cipher per klient. Korzystał on będzie z dedykowanego protokołu szyfrującego.

**10.2 Format ramki**

|  |  |  |
| --- | --- | --- |
| Rodzaj kodowania  1B | Długość danych  2B | Dane |

Rodzaj kodowania określony będzie za pomocą jednego bajta. W aktualnej wersji zaimplementowany zostanie tylko jeden rodzaj kodowania – kodowanie XOR ze stałym ciągiem bitów dla każdego klienta.

|  |  |
| --- | --- |
| Rodzaj kodowania | Kod |
| XOR | 0x00 |

Długość danych to 2-bajtowe pole określające wielkość danych.

**10.3 Diagram stanów modułu Cipher**

Szyfrowanie XOR protokołu szyfrującego modułu Cipher jest bezstanowe. Ogranicza się ono do jednej operacji zaszyfrowania ciągu danych zawsze w ten sam sposób.

# 10. Opis modułu TCPLayer

Moduł TCPLayer po stronie serwera odpowiadał będzie za bezpośrednią komunikację z klientem.

Jej działanie będzie opartę o metody *send* i *receive.* Metoda send będzie wywoływana przez moduł szyfrujący. Będzie ona opakowywać otrzymane od modułu szyfrującego dane, tak aby były zgodne z ramką protokołu TCPLayer. Po wysłaniu danych funkcja send uruchamiać będzie prywatną funkcję receive, w której klient oczekiwać będzie na otrzymanie danych od serwera. Kiedy nadejdą dane od serwera, zostaną odpakowane z nagłówków i przekazane do modułu szyfrującego.

Ramka protokołu TCPLayer:



# 11. Diagram stanów sesji klienta

# UCklient.PNG12. Scenariusze użycia



# 13. Testowanie aplikacji

Na początku procesu testowania aplikacji odbędzie się testowanie jednostkowe dla poszczególnych komponentów aplikacji. Będą one prowadzone indywidualnie przez projektantów danych modułów dzięki metodzie wstrzykiwania zależności. Pozwoli to na uniknięcie zależności pomiędzy konkretnymi implementacjami współpracujących klas, a bazowaniu tylko na interfejsach pomiędzy nimi. Dzięki temu jesteśmy w stanie skupić się wyłącznie na testowaniu funkcjonalności wybranych komponentów.W tym celu będziemy wykorzystywać obiekty-zaślepki - proste pseudoimplementacje wymaganych argumentów

Następnie będą wstrzykiwane prawdziwe obiekty w procesie integracji komponentów. Testowanie integracyjne pozwoli na wykrycie niedopasowanych mechanizmów współpracy poszczególnych modułów aplikacji. Wszystkie wykryte błędy będą usuwane na bieżąco a proces trwał będzie do zintegrowania całości oprogramowania. Po zintegrowaniu całej aplikacji odbędą się testy funkcjonalne, które będą sprawdzały poprawność wszystkich funkcji udostępnionych użytkownikowi.

Wymagania niefunkcjonalne zostaną po części sprawdzone podczas testów wydajnościowych , przeciążeniowych oraz odpornościowych. Testy wydajnościowe zbadają poprawność i szybkość działania aplikacji przy przewidywanym obciążeniu. Jeśli aplikacja będzie się zawieszała bądź zbyt długo obsługiwała zlecone polecenia to nie przejdzie pozytywnie testów. Testy przeciążeniowe sprawdzą czy aplikacja zachowuje się poprawnie podczas maksymalnego obciążenia. Jeśli aplikacja przestanie działać poprawnie to testy będą ocenione negatywnie. Następnie przejdziemy do testów odpornościowych, które sprawdzą czy aplikacja postępuje zgodnie z przewidywanym schematem w sytuacjach krytycznych.

Ostatnim krokiem będą testy systemowe wykonane w przyszłym środowisku pracy aplikacji. Sprawdzą one zgodność sposobu działania wszystkich funkcji oraz weryfikacji innych właściwości systemu określonych przez wymagania niefunkcjonalne. Po przejściu przez wszystkie wyżej wymienione testy zostanie przeprowadzony test dokumentacji, który sprawdzi jej zgodność ze stanem właściwym.

# 14. Podział pracy pomiędzy członków zespołu

## 13.1 Joanna Raczyńska

13.1.1 Implementacja komponentu TcpLayer

13.1.2 Implementacja komponentu ClientSession

## 13.2 Kamil Kacperski

13.2.1 Implementacja komponentu DAL

13.2.2 Implementacja komponentu SmtpLayer

## 13.3 Michał Mudel

13.3.1 Implementacja komponentu SessionsListener

13.3.2 Implementacja komponentu Cipher

13.3.3 Implementacja klasy MessagesQueues

## 13.4 Wojciech Zieliński

13.4.1 Zarządzanie projektem

13.4.2 Implementacja aplikacji klienckiej

13.4.3 Implementacja komponentu RootManager

15. Analiza sytuacji krytycznych i propozycje ich rozwiązania

**14.1 Chwilowa lub trwała utrata łączności między węzłami**

Rozwiązanie: Serwer zamyka połączenie z klientem i natychmiast zakańcza wszystkie wątki związane z tym połączeniem. Aplikacja klienta informuje o utracie połączenia i zaleca ponowne połączenie.

**14.2 Otrzymanie błędnych pakietów**

Rozwiązanie: Serwer zamyka połączenie z klientem i natychmiast zakańcza wszystkie wątki związane z tym połączeniem.

**14.3 Przekroczenie limitu czasowego oczekiwania na odpowiedź klienta (5 min – wymaganie 4.5)**

Rozwiązanie: Serwer zamyka połączenie z klientem i natychmiast zakańcza wszystkie wątki związane z tym połączeniem. Aplikacja klienta informuje o utracie połączenia i zaleca ponowne połączenie.

**14.4 Uszkodzenie bądź utrata dostępu do bazy danych**

Rozwiązanie: Zaprzestanie pracy serwera. Dla próbujących się połączyć aplikacji klienckich wyświetlany jest komunikat o tymczasowym technicznym problemie, dopóki dostęp do bazy danych nie zostanie odzyskany. W przypadku nieodwracalnego uszkodzenia i braku możliwości odzyskania danych wymagane będzie ponowne założenie kont dla użytkowników.

**14.5 Utrata przez serwer połączenia z Internetem**

Rozwiązanie: Wszystkie sesje klienckie zostają zamknięte. Serwer nie funkcjonuje do czasu odzyskania połączenia z Internetem.

**14.6** **Przeciążenie systemu (duża liczba połączonych użytkowników)**

Rozwiązanie: Serwer nie przekracza pewnej, założonej liczby maksymalnych połączeń. W przypadku próby połączenia kolejnego klienta serwer informuje aplikację kliencką o tym fakcie i aplikacja kliencka wyświetla komunikat o zajętości serwera i zaleca spróbować ponownie za pewien czas.  
**14.7 Przeciążenie systemu (duża ilość żądań do przetworzenia od jednego użytkownika – wymaganie 4.7)**  
Rozwiązanie: Serwer zamyka połączenie z danym klientem. Dany klient zostaje również zablokowany.

**14.8 Przeciążenie systemu (duża ilość żądań do przetworzenia – wymaganie 4.8)**  
Rozwiązanie: Serwer zamyka wszystkie sesje klienckie i blokuje wszelką komunikację do czasu przeanalizowania sytuacji która doprowadziła do nagłej dużej ilości żądań.  
**14.9 Nagłe wyłączenie systemu**

Rozwiązanie: Wszystkie trwające na serwerze operacje zostają przerwane bez zapamiętania kontekstu. TcpLayer po stronie aplikacji klienckiej informuje aplikację o utracie połączenia i o tym, że żądane polecenie nie zostało wykonane.