KOMUNIKACJA POPRZEZ SIEĆ, PROTOKOŁY, UŻYCIE SSH, KOMUNIKACJA Z USŁUGAMI REST

ĆWICZENIA DO PREZENTACJI

MODUŁ 12

Altkom Akademia S.A., materiały własne

2 WYBRANE PROTOKOŁY: TELNET, SSH, SFTP

ĆWICZENIE 2.1:

Instalacja i praca z klientem SSH

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia poznasz wybrane programy będące klientami SSH

CELE I ZADANIA:

- Zainstaluj:
  - wbudowanego w system Windows klienta SSH
  - program PuTTY
- Korzystając z tych programów zaloguj się za pomocą SSH na serwery:
  - `test.rebex.net`
  - `tty.sdf.org`
- Wydaj kilka poleceń, aby sprawdzić działanie

ALGORYTM WYKONANIA:

- Uruchom okno wiersza poleceń
- Wydaj polecenie `ssh`, aby sprawdzić, czy dostępny jest wbudowany w system Windows klient SSH
- Jeśli polecenie nie zostanie rozpoznane, to zainstaluj klienta
- W tym celu:
  - wciśnij klawisze <Windows><I>
  - wybierz opcje *Aplikacje* → *Funkcje opcjonalne*
  - wciśnij przycisk *Wyświetl funkcje* w sekcji *Dodaj funkcję opcjonalną*
  - odszukaj pozycję *Klient OpenSSH* i zaznacz pole wyboru
  - wciśnij przycisk *Dalej*, a następnie *Zainstaluj*
- Sprawdź działanie zainstalowanego klienta logując się do serwerów:
  - `test.rebex.net`, login: *demo*, hasło: *password*
    wykorzystaj polecenie: `ssh demo@test.rebex.net`
  - `tty.sdf.org`, login i hasło zgodne z utworzonym kontem
- Wydaj kilka poleceń, aby sprawdzić działanie zdalnej sesji, np.: *ls, ls -l, pwd, whoami*, ...
- Zakończ pracę i wyloguj się (polecenie *exit*)
- Powtórz te same czynności z klientem PuTTY (program można pobrać spod adresu: `https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html`)

ĆWICZENIE 2.2:

Budowa klienta SSH z użyciem modułu paramiko

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia poznasz jak wykorzystać bibliotekę *paramiko* do stworzenia klienta SSH

CELE I ZADANIA:

- Napisz program, który umożliwi zdalną pracę terminalową z wykorzystaniem protokołu SSH
- Wykorzystaj bibliotekę *paramiko*
- Za pomocą programu zaloguj się na serwer *test.rebex.net* (patrz poprzednie ćwiczenie)
- Wprowadź z klawiatury kilka poleceń i wykonaj je

ALGORYTM WYKONANIA:

- Utwórz pakiet *ex02_config*
- Utwórz w nim plik konfiguracyjny *credentials.ini*, zawierający dane niezbędne do zalogowania do zdalnego serwera:

```
[DEFAULT]
port = 22

[rebex]
hostname = test.rebex.net
username = demo
password = password
```

- Uzupełnij ten plik o sekcję `[sdf]` zawierającą dane do zalogowania do serwera *tty.sdf.org*
- Ten plik i dane będą wykorzystywane w tym i w kolejnych ćwiczeniach
- Wykorzystując moduł *configparser* odczytaj parametry połączenia dla serwera *test.rebex.net* – szczegóły: p. moduł 9: *"Obsługa i przetwarzanie różnych typów danych, użycie wyrażeń regularnych"*
- Wzorując się na przykładzie z prezentacji napisz program umożliwiający pracę terminalową
- W wywołaniu metody *connect* użyj dodatkowej opcji *look_for_keys=False* – spowoduje ona pominięcie poszukiwania pliku z kluczem prywatnym w katalogu *.ssh*
- Program powinien działać w pętli i w każdej iteracji oczekiwać polecenia wprowadzanego z klawiatury
- To polecenie powinno być wykonane zdalnie, a wynik przedstawiony na ekranie
- Wprowadzenie pustego łańcucha powinno być sygnałem do zakończenia połączenia

ĆWICZENIE 2.3:

Budowa klienta SSH z użyciem modułu paramiko-expect

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia poznasz jak wykorzystać biblioteki *paramiko* oraz *paramiko-expect* do stworzenia klienta SSH

CELE I ZADANIA:

- Napisz program, który umożliwi zdalną pracę terminalową z wykorzystaniem protokołu SSH
- Wykorzystaj biblioteki *paramiko* oraz *paramiko-expect*
- Za pomocą programu zaloguj się na serwer *tty.sdf.org* (patrz poprzednie ćwiczenie)
- Wprowadź z klawiatury kilka poleceń i wykonaj je

ALGORYTM WYKONANIA:

- Skopiuj rozwiązanie poprzedniego ćwiczenia i sprawdź, czy działa ono także z serwerem *tty.sdf.org*
- Co jest powodem takiego zachowania?
- Wzorując się na przykładzie z prezentacji zmodyfikuj kod tak, aby program umożliwiał pracę terminalową
- Program powinien działać w pętli i w każdej iteracji oczekiwać polecenia wprowadzanego z klawiatury
- To polecenie powinno być wykonane zdalnie, a wynik przedstawiony na ekranie
- Wprowadzenie pustego łańcucha powinno być sygnałem do zakończenia połączenia
- W rozwiązaniu posłuż się biblioteką *paramiko-expect* – możesz wzorować się na przykładzie z prezentacji

ĆWICZENIE 2.4:

Użycie klienta SFTP

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia poznasz, jak wykorzystać bibliotekę *paramiko* do stworzenia klienta SFTP

CELE I ZADANIA:

- Napisz program, który umożliwi pobranie plików z serwera SFTP
- Wykorzystaj bibliotekę *paramiko* do utworzenia klienta SFTP
- Za pomocą programu zaloguj się na serwer *test.rebex.net* (patrz poprzednie ćwiczenia)
- Pobierz wybrane pliki

ALGORYTM WYKONANIA:

- Skopiuj rozwiązanie ćwiczenia z pakietu *ex02_02*
- Utwórz lokalnie podkatalog o nazwie *remote*, w którym zapiszemy pliki pobrane z serwera SFTP
- Wzorując się na prezentacji utwórz klienta SFTP na podstawie klienta SSH
- Wypisz zawartość bieżącego katalogu
- Przejdź do zdalnego podkatalogu *pub/example*
- Wypisz nazwę bieżącego katalogu roboczego
- Skopiuj do lokalnego katalogu *remote* wszystkie pliki, których nazwy nie kończą się na *Small.png*
- W trakcie kopiowania:
  - wypisz nazwę kopiowanego pliku
  - prezentuj postępy w kopiowaniu (możesz wykorzystać parametr *callback* metody *get*)
- Zakończ połączenie i sprawdź, czy pliki zostały pobrane

3 KOMUNIKACJA Z WYKORZYSTANIEM GNIAZD

ĆWICZENIE 3.1:

Przykład komunikacji przy użyciu gniazd

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia dowiesz się, jak można zrealizować komunikację pomiędzy serwerem, a klientem z wykorzystaniem gniazd

CELE I ZADANIA:

- Napisz dwa programy:
  - program startujący serwer, nasłuchujący połączenia na wybranym porcie
  - program klienta, który nawiąże połączenie z serwerem
- Oba programy powinny umożliwić wymianę komunikatów (chat) pomiędzy sobą
- Zaproponuj rozwiązanie umożliwiające zakończenie konwersacji (powinien to zainicjować klient)

ALGORYTM WYKONANIA:

- Wzorując się na schemacie komunikacji przedstawionym na prezentacji napisz dwa programy komunikujące się ze sobą – program serwera i program klienta
- W programie serwera:
  - utwórz gniazdo serwera na wybranym porcie (powyżej 1024)
  - zaplanuj możliwość nawiązania tylko jednego połączenia
  - rozpocznij nasłuch połączeń
  - po nawiązaniu połączenia i odebraniu wiadomości program powinien umożliwić wpisanie z klawiatury odpowiedzi i odesłanie jej klientowi
- W programie klienta:
  - utwórz gniazdo klienta powiązane z gniazdem podanego serwera na danym porcie
  - program powinien umożliwić wprowadzenie komunikatu z klawiatury i wysłanie go do serwera
  - następnie program oczekuje na odpowiedź i wyświetla ją na ekranie
  - program powinien działać w pętli na zasadzie: "wysłanie komunikatu – odebranie odpowiedzi" do momentu, gdy klient wyda komunikat zakończenia konwersacji (np. poleceniem *bye*)
- Wysłanie komunikatu kończącego powinno spowodować wyjście z pętli i zamknięcie połączenia i gniazda klienta, a to z kolei zamknięcie gniazda serwera
- Uruchom program serwera, a następnie program klienta i przetestuj komunikację
- Możesz też spróbować nawiązać komunikację pomiędzy dwoma programami działającymi na różnych komputerach

4 PROTOKÓŁ HTTP

ĆWICZENIE 4.1:

Komunikacja za pomocą protokołu HTTP

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz wiedzę nt. struktury wiadomości HTTP

CELE I ZADANIA:

- Zapoznanie się ze strukturą:
  - żądania HTTP
  - odpowiedzi HTTP

ALGORYTM WYKONANIA:

- Z pomocą trenera utwórz prosty serwer HTTP przy wykorzystaniu biblioteki *Flask*
- Zadaniem serwera będzie wypisanie:
  - użytej metody HTTP
  - listy nagłówków HTTP żądania
- Serwer będzie działał na porcie 5000
- Uruchom serwer
- Uruchom przeglądarkę i wpisz adres `http://127.0.0.1:5000`
- Zobacz jaką odpowiedź zwrócił serwer

- Uruchom *Firefox*'a i zainstaluj wtyczkę *RESTClient* – można ją znaleźć pod adresem `https://addons.mozilla.org/en-US/firefox/addon/restclient/`
- Korzystając z zainstalowanej wtyczki wyślij żądanie typu GET pod adres `http://127.0.0.1:5000`
- Zobacz, jakie nagłówki HTTP zostały zwrócone w odpowiedzi
- Wyślij kolejne żądanie pod ten sam adres, wykorzystując metodę HEAD
- Czym różni się ta odpowiedź od poprzedniej?
- Wyślij jeszcze raz żądanie pod ten sam adres, wykorzystując metodę OPTIONS
- Jaką informację zwróciła ta metoda?

5 ARCHITEKTURA REST

ĆWICZENIE 5.1:

Zapoznanie się z działaniem aplikacji REST

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność korzystania z aplikacji REST

CELE I ZADANIA:

- Korzystając z wtyczki *RESTClient* przeglądarki *Firefox* przestestuj funkcjonalność dostarczonej aplikacji REST
- Zwróć uwagę na ustawiane nagłówki HTTP i URI żądania

ALGORYTM WYKONANIA:

- Aplikacja zarządza danymi osobowymi i umożliwia realizację operacji CRUD
- Każda osoba jest opisywana przez:
  - unikalny identyfikator – atrybut *pid* (liczba całkowita)
  - imię – atrybut *fname*
  - nazwisko – atrybut *lname*
  - wiek – atrybut *age* (liczba całkowita)
- Aplikacja jest dostępna pod bazowym adresem:
  http://195.167.158.224:8080/REST/db/persons
- Dane mogą być zwracane i przyjmowane w formatach: tekstowym, XML oraz JSON
- Do **odczytu wszystkich osób**:
  - użyj metody GET
  - zwróć uwagę, jak zmieni się odpowiedź, gdy zmienisz wartość nagłówka *Accept* żądania (wybierz typ MIME odpowiadający jednemu ze wspieranych formatów)
- Aby **odczytać pojedynczą osobę**:
  - uzupełnij adres bazowy, dodając na końcu segment z numerem identyfikacyjnym danej osoby
  - ponownie użyj metody GET
- Aby **dodać nową osobę**:
  - użyj metody POST skierowanej pod adres bazowy
  - dane osoby umieść w ciele wiadomości
  - nie musisz ustawiać atrybutu *pid*
  - pamiętaj, aby ustawić nagłówek *Content-Type* określający format przesyłanych danych

  - zwróć uwagę na nagłówek *Location* odpowiedzi – powinien zawierać URI pod którym dostępne będą dane nowej osoby
  - przetestuj żądanie dla różnych formatów danych

- Aby **usunąć wybraną osobę**:
  - użyj metody DELETE
  - uzupełnij adres bazowy, dodając na końcu segment z numerem identyfikacyjnym danej osoby
  - w odpowiedzi nie jest zwracane ciało

- Aby **zaktualizować dane istniejącej osoby**:
- użyj metody PUT skierowanej pod adres bazowy
- zaktualizowane dane osoby (kompletne) umieść w ciele wiadomości
- atrybut *pid* powinien identyfikować osobę, której dane aktualizujemy
- pamiętaj, aby ustawić nagłówek *Content-Type* określający format przesyłanych danych
- w odpowiedzi nie jest zwracane ciało

6 MODUŁ REQUESTS

ĆWICZENIE 6.1:

Budowa klienta REST przy użyciu modułu requests

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność implemetacji klienta REST

CELE I ZADANIA:

- Utwórz program będący klientem REST
- Zaimplementuj operacje CRUD

ALGORYTM WYKONANIA:

- Wzorując się na poprzednim ćwiczeniu napisz program realizujący te same operacje, a więc umożliwiający:
  - pobranie danych wszystkich osób z usługi REST
  - pobranie danych osoby o podanym identyfikatorze
  - dodanie nowej osoby
  - usunięcie osoby o podanym identyfikatorze
  - zaktualizowanie danych istniejącej osoby o podanym identyfikatorze
- Do realizacji tych zadań użyj modułu *requests*
- Jako reprezentacji danych użyj formatu JSON