Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

Informatyka, Aplikacje internetowe i mobilne, semestr 5

Projektowanie serwisów internetowych

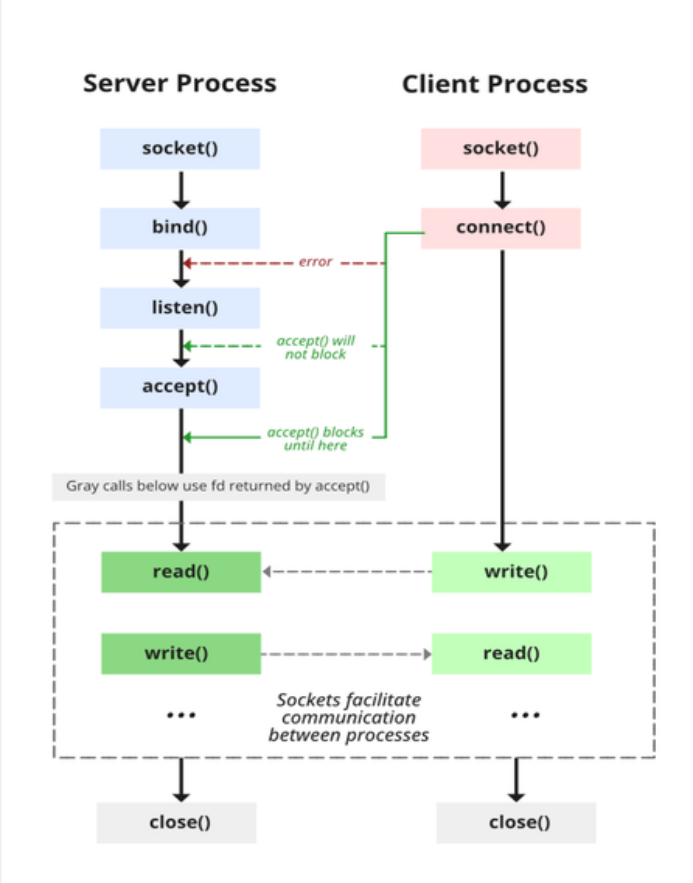
Laboratorium nr 6

PHP – Projektowanie serwisów z użyciem gniazd

Celem laboratorium jest zapoznanie studentów z projektowaniem serwisów internetowych za pomocą gniazd. Rozwiązując kolejne zadania można korzystać ze źródeł poprzednich wykonując odpowiednie modyfikacje. Z tego powodu należy podczas rozpoczęcia kolejnego zadania skopiować wersję poprzednią w celu nietracenia poprzedniego rozwiązania zadania (jest to ważne ze względu na prawidłowe rozliczenie się z tych zdań w sprawerze) .

Co to jest programowanie gniazd?

Programowanie gniazd to sposób na połączenie dwóch węzłów w sieci w celu komunikowania się ze sobą. Jedno gniazdo (węzeł) nasłuchuje na określonym porcie jakiegoś urządzenia o konkretnym adresie IP, podczas gdy inne gniazdo dociera do drugiego, aby utworzyć połączenie. Serwer tworzy gniazdo słuchacza, podczas gdy klient kontaktuje się z serwerem. Poniżej został przedstawiony diagram stanu dla procesu klienta i serwera z użyciem gniazd sieciowych (network socket).



Katedra Systemów Informacyjnych

Zadanie 1

Zadaniem naszym będzie opracowanie klienta, aby wysłać wiadomość tekstową np. „Hello” do serwera (działającym sieciowo na nr IP: 127.0.0.1 i porcie 10000) na którą serwer powinien odpowiedzieć w stronę klienta odpowiednią wiadomością zwrotną będącą zaszyfrowanym tekstem za pomocą algorytmu md5.

W podkatalogu Zadanie1 przygotuj odpowiedni plik PHP o nazwie server.php oraz client.php.

Przygotowanie kodu serwera – plik server.php:

Krok 1: Ustawienie zmiennych takich jak host i port

```
$host = “127.0.0.1”; // twoje lokalne IP w systemie
$port = 10000; // Numer portu może być dowolną dodatnią liczbą
// całkowitą z przedziału od 1024 do 65535.
set_time_limit(0);
```

set_time_limit - Ogranicza maksymalny czas wykonania skryptu, wartość zero oznacza brak limitu (jest to potrzebne do stanu nasłuchiwania serwera w trybie oczekiwania na połączenie się klienta, a czas takiego połączenia nie jest nam znany) .

Krok 2: Utworzenie gniazda

W celu utworzenia gniazda należy posłużyć się funkcją socket_create:

```
// create socket
$socket = socket_create(AF_INET, SOCK_STREAM, 0) or die("Could
not create socket\n");
```

Krok 3: Powiązanie gniazda z portem i hostem

W celu powiązania gniazda z portem i hostem należy użyć funkcji socket_bind:

```
$result = socket_bind($socket, $host, $port) or die("Could not
bind to socket\n");
echo "bind socket to port: $port on host: $host\n";
```

Krok 4: Rozpoczęcie nasłuchiwania na gniezdzie

Za pomocą funkcji socket_listen serwer czeka na połączenia przychodzące na gnieździe.

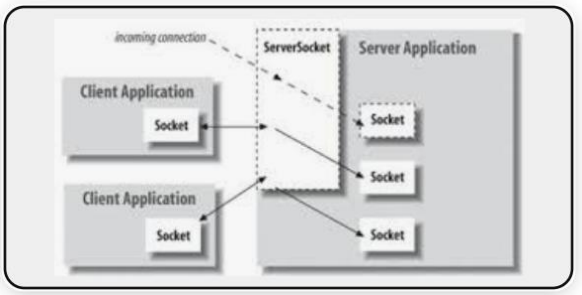
```
$result = socket_listen($socket, 3) or die("Could not set up
socket listener\n");
echo "start listening for connections - socket listen\n";
```

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

Krok 5: Zaakceptuj połączenie przychodzące

Użycie funkcji socket_accept akceptuje przychodzące żądanie połączenia w utworzonym gnieździe serwera. Po pomyślnym nawiązaniu połączenia zwracana jest nowa instancja Socket, która służyć do komunikacji z klientem. Zatem zmienna„$spawn" o typie socket to zasób gniazda, który jest odpowiedzialny za komunikację z gniazdem klienta.

```
$spawn = socket_accept($socket) or die("Could not accept incoming
connection\n");
echo "spawn another socket to handle communication\n";
```



Obok na rysunku przedstawiono mechanizm tworzenia przez serwer nowych gniazd do komunikacji z klientami (jest to standardowa architektura klient-serwer). Do tej pory przygotowaliśmy nasze gniazdo serwera, ale skrypt tak naprawdę nic nie robi. Zgodnie z naszym celem w następnych krokach odczytamy wiadomość z utworzonego nowego gniazda klienta, a następnie odeślemy odebraną wiadomość tekstową do klienta zakodowaną pomocą algorytmu md5.

Krok 6: Przeczytaj wiadomość z gniazda klienta (Client)

Za pomocą funkcji socket_read odczytujemy maks. liczbę 1024 bajtów z gniazda.

```
$input = socket_read($spawn, 1024) or die("Could not read
input\n");
echo "read client input from socket\n";
```

Step 7: Wygenerowanie zakodowanej odpowiedzi za pomocą md5

```
$output = md5($input). "\n";
```

Krok 8: Wysłanie odpowiedzi do gniazda klienta (client)

Za pomocą funkcji socket_write wysyłamy odpowiedź zwrotną do klienta.

```
socket_write($spawn, $output, strlen ($output)) or die("Could not
write output\n");
```

Krok 9: Zamknięcie gniazd/a

Za pomocą funkcji socket_write zamykamy najpierw gniazdo komunikacji z klientem, a na sam koniec następuje zamknięcie gniazda serwera.

```
socket_close($spawn); // zamkniecie gniazda komunikacji z klientem
socket_close($socket); // zamkniecie gniazda serwera /koniec nasłuchiwania
```

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

Krok 10: Uruchomienie serwera i weryfikacja jego stanu

Uruchomienie serwera:

Uruchom linię poleceń i przejdź do katalogu z zadaniem1.

Za pomocą komendy: php -q nazwa_skryptu.php uruchom swój serwer.

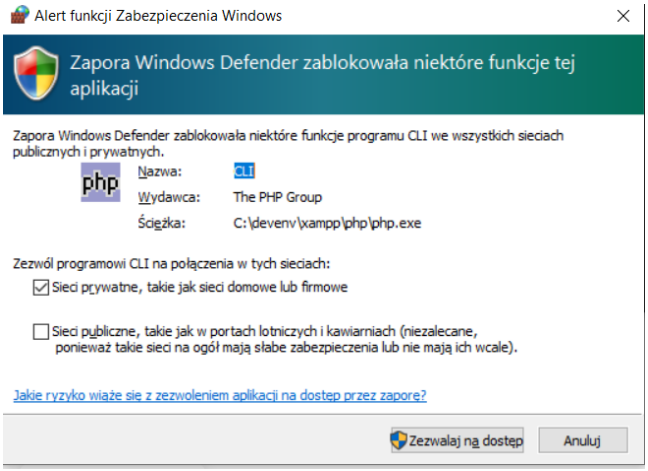
```
C:\DevEnv\xampp\htdocs\55543\PSI_LAB6\zadanie1>php -q server.php
bind socket to port: 10000 on host: 127.0.0.1
start listening for connections - socket listen
```

Uwaga: Jeśli otrzymasz komunikat, że nieodnaleziono php to w celu prawidłowego wywołania php w linii poleceń w twoim systemie powinna być zdefiniowana ścieżka dostępu do php. Sprawdź wartość zmiennej środowiskowej PATH:

`echo %PATH%` Ścieżka powinna zawierać: `C:\DevEnv\xampp\php;`

Jeśli ścieżka nie zawiera C:\DevEnv\xampp\php to dodaj ją w systemie operacyjnym.

Uwaga: Podczas pierwszego uruchamiania skryptu php serwera następuje próba otworzenia portu 10000 po stronie systemu operacyjnego. W takim przypadku może pojawić się monit zapory systemu operacyjnego np. Defender lub innego oprogramowania np. ESET o zezwolenie na ten dostęp w ramach odpowiednich sieci. Potwierdź „Zezwalaj na dostęp”



Weryfikacja stanu:

Jeśli udało się uruchomić serwer to za pomocą polecenia:

```
netstat -an | find ":10000"
```

sprawdź, czy na porcie 10000 nasłuchuje uruchomiony serwer.

```
C:\DevEnv\xampp\htdocs\55543\PSI_LAB6\zadanie1>netstat -an | find ":10000"
  TCP    127.0.0.1:10000        0.0.0.0:0              LISTENING
```

Jeśli tak to teraz przejdziemy do przetestowania serwera za pomocą klienta telnet

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

Krok 11: Przetestowanie serwera za pomocą klienta telnet

Wykorzystamy klasyczne testowanie usług sieciowych za pomocą klienta telnet.

W nowym oknie linii poleceń za pomocą polecenia telnet (*jeśli polecenie telnet nie jest znane w systemie do doinstaluj klienta telnet*) przetestuj działanie usługi swojego serwera.

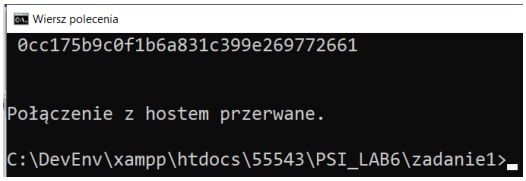
```
telnet  localhost  10000
```

Po wyświetleniu pustego okna linii poleceń:

Sprawdź, że po stronie serwera (w innym oknie CMD) pojawił się komunikat:

```
spawn another socket to handle communication
```

Wróć do linii poleceń, w którym uruchomiono polecenie telnet i po wciśnięciu jednego klawisza np. „a” uzyskasz odpowiedź zaszyfrowanej wiadomości.



Sprawdź co się działo po stronie serwera.

```
C:\DevEnv\xampp\htdocs\55543\PSI_LAB6\zadanie1>php -q server.php
bind socket to port: 10000 on host: 127.0.0.1
start listening for connections - socket listen
spawn another socket to handle communication
read client input from socket
Client Message : a
```

Ponowne testowanie serwera

Na tym etapie zauważ, że po jednym testowaniu nastąpił koniec pracy serwera. Zatem jeśli chcesz ponownie przetestować jego działanie to musisz go ponownie uruchomić (tak jak w kroku 10).

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

Zaprojektowanie klienta usługi za pomocą PHP

Do tej pory został zbudowany serwer za pomocą PHP i był on testowany za pomocą klienta telnet. Teraz czas na zbudowanie kodu klienta za pomocą PHP. Dlatego w tym samym katalogu roboczym utwórz plik client.php i przejdź do kolejnych kroków w celu jego zaprogramowania. Pierwsze kroki przypominają kod serwera. Zakładamy, że klient do serwera wyśle wiadomość
„Hello Server"

Krok 1: Ustawienie zmiennych host i port oraz czasu wykonania i komunikatu

```
$host = "127.0.0.1";
$port = 10000; // tutaj port i host muszą być takie same jak zdefiniowano w server.php.
// No Timeout
set_time_limit(0);
$message = "Hello Server";
```

Krok 2: Utworzenie gniazda klienta

```
$socket = socket_create(AF_INET, SOCK_STREAM, 0) or die("Could
not create socket\n");
echo "Socket Create\n";
```

Krok 3: Podłączenie się do serwera

```
$result = socket_connect($socket, $host, $port) or die("Could not
connect to server\n");
echo "Socket Connect to port $port on host/server $host\n";
```

Krok 4: Napisz do gniazda serwera

W tym kroku dane gniazda klienta są wysyłane do gniazda serwera.

```
socket_write($socket, $message, strlen($message)) or die("Could
not send data to server\n");
echo "Message to server : $message\n";
```

Krok 5: Odczytanie odpowiedzi z serwera

```
$result = socket_read ($socket, 1024) or die("Could not read
server response\n");
echo "Reply From Server  :".$result;
```

Krok 6: Zamknięcie gniazda klienta

Zamknięcie gniazda klienta kończy działanie klienta.

```
socket_close($socket);
```

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

Krok 7: Przetestowanie działania klienta

Uruchom serwer tak jak poprzednio, a następnie w nowym oknie linii poleceń będąc w katalogu roboczym zadanie1 uruchom swojego klienta.

```
C:\DevEnv\xampp\htdocs\55543\PSI_LAB6\zadanie1>php -q client.php
Socket Create
Socket Connect to port 10000 on host/server 127.0.0.1
Message to server : Hello Server
Reply From Server  :c09fd7e6885f0aef3ea998f1738f9335
```

Opracowywanie błędów połączenia klienta z serwerem

Poniżej informacyjnie przedstawiono wyświetlenie błędów podczas działania socket_connect.

```
$result = socket_connect ($socket, $host, $port);
if ($result < 0) {
    echo "socket_connect() failed.\nReason: ($result) " .
socket_strerror($result) . "\n";
} else {
    echo  "The connection to the server '$host' has been
established\n"
}
```

Zadanie 2

Utwórz katalog zadanie2 i skopiuj do niego pliki server.php oraz client.php. Zmień ich nazwy na server2.php oraz client2.php

W zdaniu tym należy:

1) dopracować kod serwera tak, aby realizując architekturę klient-serwer usługa naszego serwera po obsłużeniu jednego klienta kontynuowała działanie w celu obsłużenia kolejnych klientów (a nie kończyła się po obsłużeniu jednego połączenia tak jak w zadaniu 1). Tym razem serwer nie wysyła kodu md5 tylko przesłany tekst odsyła klientowi w odwrotnej kolejności znaków. Przykład: klient wysyła wiadomość Gdynia, a serwer odsyła odpowiedź: ainydG. Otrzymanie tekstu/komendy „shutdown” powoduje zakończenie pracy serwera. Otrzymanie komunikatu „quit” generuje komunikat po stronie serwera, że nastąpiło zakończenie działania klienta.
2) dopracować kod klienta tak, aby ciągle powtarzał dialog:

   Welcome to server on 127.0.0.1
   Client: The 'shutdown' command terminates server and client
   Client: The 'quit' command terminates the client
   Enter a message / command: ?????

   do czasu, aż po wydaniu komendy shutdown następuje zatrzymanie zakończenie działania klienta i nasłuchującego serwera. Samo wydanie quit oznacza zakończenie pracy klienta i dalsze prawidłowe funkcjonowanie serwera.

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

Poniżej przedstawiono w tabeli jak powinien wyglądać dialog klienta z serwerem. Klient wysyła kolejno wiadomość: Gdynia, Sopot i kończy komendą shutdown.

server2.php	client2.php
bind socket to port: 10000 on host: 127.0.0.1 start listening for connections - socket listen Continuous listening state for the socket - for incoming connection spawn another socket to handle communication read client input from socket Client Message : Gdynia Continuous listening state for the socket - for incoming connection spawn another socket to handle communication read client input from socket Client Message : Sopot Continuous listening state for the socket - for incoming connection spawn another socket to handle communication read client input from socket Client Message : shutdownShutdown server on 127.0.0.1 End of the server	--------------------------- Welcome to server on 127.0.0.1 Client: The 'shutdown' command terminates server and client Client: The 'quit' command terminates the client Enter a message / command: Gdynia Message to server : Gdynia Reply rrom server :ainydG --------------------------- Welcome to server on 127.0.0.1 Client: The 'shutdown' command terminates server and client Client: The 'quit' command terminates the client Enter a message / command: Sopot Message to server : Sopot Reply rrom server :topoS --------------------------- Welcome to server on 127.0.0.1 Client: The 'shutdown' command terminates server and client Client: The 'quit' command terminates the client Enter a message / command: shutdown Message to server : shutdown End of the Client
bind socket to port: 10000 on host: 127.0.0.1 start listening for connections - socket listen Continuous listening state for the socket - for incoming connection spawn another socket to handle communication read client input from socket Client Message : Gdynia Continuous listening state for the socket - for incoming connection spawn another socket to handle communication read client input from socket Client Message : Sopot Continuous listening state for the socket - for incoming connection spawn another socket to handle communication read client input from socket Client Message : quit Client quit on host: 127.0.0.1 Continuous listening state for the socket - for incoming connection	--------------------------- Welcome to server on 127.0.0.1 Client: The 'shutdown' command terminates server and client Client: The 'quit' command terminates the client Enter a message / command: Gdynia Message to server : Gdynia Reply rrom server :ainydG --------------------------- Welcome to server on 127.0.0.1 Client: The 'shutdown' command terminates server and client Client: The 'quit' command terminates the client Enter a message / command: Sopot Message to server : Sopot Reply rrom server :topoS --------------------------- Welcome to server on 127.0.0.1 Client: The 'shutdown' command terminates server and client Client: The 'quit' command terminates the client Enter a message / command: quit Message to server : quit End of the Client

Zadanie 3

Utwórz katalog zadanie3 i skopiuj do niego pliki server2.php oraz client2.php. Zmień ich nazwy na server3.php oraz client3.php.

Należy tak zaprojektować kod serwera oraz klienta tak , aby

- po stronie klienta pod odczytaniu liczby naturalnej n (n >=1) należy ją przekazać do serwera,
- serwer po trzymaniu od klienta liczby n za pomocą wydzielonej funkcji losuj (umieszczonej w oddzielnym pliku losuj.inc) losuje n liczb naturalnych z przedziału od 1 do 100, aby następnie po ich posortowaniu od najmniejszej do największej przekazać je klientowi,
- klient otrzymuje odpowiedź od serwera (wylosowane liczby) i wypisuje formie:
  Wylosowane liczby: [3, 6, 15, 56, 78] dla n = 5

Powyższe rozwiązanie reaguje na podanie n=quit -> następuje koniec działania klienta i serwera.

Rozwiązanie należy zabezpieczyć, aby sprawdzenie, czy podane n jest liczbą naturalną i n jest >= 1 następowało po stronie klienta lub serwera.

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

Zadanie 4

Utwórz katalog zadanie4 i skopiuj do niego pliki server3.php oraz client3.php. Zmień ich nazwy na server4.php oraz client4.php.

Należy tak zaprojektować kod serwera server4.php oraz webowego klienta PHP client4.php z elementami formularza HTML oraz podłączonymi stylami CSS tak , aby

- po stronie klienta webowego pobraną liczbę naturalną n (n >=1) z pola formularza HTML za pomocą metody POST należy przekazać do serwera wartość n po ówczesnym nawiązaniu połączenia z serwerem,
- serwer po trzymaniu od klienta liczby n za pomocą wydzielonej funkcji losuj (umieszczonej w oddzielnym pliku losuj.inc) losuje n liczb naturalnych z przedziału od 1 do 100, aby następnie po ich posortowaniu od najmniejszej do największej przekazać je klientowi,
- klient webowy otrzymuje odpowiedź od serwera (wylosowane liczby) i wypisuje je zaraz pod formularzem (pod przyciskiem Losuj liczby) w formie: Wylosowane liczby: [3, 6, 15, 56, 78] dla n = 5

Rozwiązanie należy zabezpieczyć, aby sprawdzenie, czy podane n jest liczbą naturalną i n jest >= 1 następowało po stronie usługi serwera.

Ponieważ klient będzie uruchamiany jako aplikacja na serwerze Aapache2 to kod źródłowy klienta nie może mieć tak jak poprzednio pętli, gdyż naszym interfejsem wejściowym będzie teraz formularz.

W celu przetestowania swojej aplikacji należy uruchomić xampp-control.exe z C:\DevEnv\xampp, aby następnie uruchomić usługę serwera Apache2 oraz swój serwer4.php w linii poleceń za pomocą:

```
php  -q  server4.php
```

Następnie wywołaj w przeglądarce aplikacji klienta client4.php.

W efekcie końcowym katalog zadanie4 powinien zawierać pliki: server4.php, client4.php, client4.css, losuj.inc.

Uwaga: Wygenerowany kod HTML i podpięte style CSS powinny być zgodne ze standardami W3C tzn. powinny przechodzić odpowiednią walidację.

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

Zadanie 5

Do nowego folderu zadanie5 pobierz z Iliasa pliku zadanie5.zip i rozpakuj go otrzymując w ten sposób plik osoby.php zawierający propozycję definicji klas: ***person oraz osoby*** wraz z odpowiednimi metodami, a także utworzonymi danymi przykładowych osób.

Zaprojektuj kod serwera server5.php nasłuchującego na porcie 5000 i korzystającego z dostępu do danych przykładowych osób (plik osoby.php), który po otrzymaniu od połączonego za pomocą gniazd klienta client5.php komunikatu/żądania w formie:

- znajdz:imie:Zofia wyszukał i zwrócił klientowi pełne dane wszystkich osób o imieniu Zofia,
- znajdz:imie:Zofia wyszukał i zwrócił klientowi pełne dane wszystkich osób o imieniu Zofia,
- znajdz:plec:K wyszukał i zwrócił klientowi pełne dane wszystkich kobiet spośród przykładowych osób,
- znajdz:plec:M wyszukał i zwrócił klientowi pełne dane wszystkich mężczyzn spośród przykładowych osób,
- komendy dodatkowe, które wynikają z możliwości klasy osoby oraz person.

przekaże odpowiedni wynik klientowi, który z punktu widzenia klienta (uruchomionego w linii poleceń) może wyświetlić odpowiedź serwera w postaci np.

```
Podaj komende: wyszukaj:plec:M
Sending command request
Reading response:
Server response:
wyszukaj:plec:M
Efekt wyszukiwania M

[Jan,Nowak,17,190,81,M]
[John,Smith,36,180,78,M]
[Waldemar,Kowalski,30,195,88,M]
```

Podpowiedzi:

- Możesz dopisywać metody w klasie person i osoby,
- W kodzie serwera skorzystaj z funkcji explode.

Zadanie 6

W nowym folderze zadanie6 zaprojektuj kod webowego klienta z odpowiednim formularzem HTML (wyszukaj według wydawnictwa, tytułu itd.) dającym możliwość za pomocą gniazd nawiązania połączenia z serwerem (port 7777) realizującego po swojej stronie wyszukiwanie książek w bazie postgresql (w znanym schemacie). Znalezione przez serwer rekordy z postgresql wyszukanych książek w formie tabeli HTML zostają przesłane przez serwer klientowi, który następnie prezentuje je w formie tabeli HTML pod formularzem wyszukiwania.

Rozliczenie w sprawerze

Wszystkie foldery zadanie1-6 zawierające rozwiązania poszczególnych zadań spakuj za pomocą jednego pliku ZIP w formcie PSI_lab6_nr_albumu.zip i wyślij jako rozliczenie w sprawerze.