LAB 2 - łagodne wejście w PHP

- AJAX za pomocą jQuery; tablice $_POST, $_GET, listing katalogu, array, foreach
- obsługa żądań post i get w jednym pliku php
- include, require;
- przesłanie pliku na serwer

0. Przygotowanie

Pobrać i ulokować do katalogu `D:\Rep\src\web\` pliki wejściowe:
- `1.html`,
- podkatalogi `E_negative` oraz `E_positive` wraz z zawartością.

Katalog MojeEmotki należy skopiować gdzieś lokalnie poza ścieżką deweloperską (będą plikami lokalnymi klienta do uploadu).

Ścieżka `D:\Rep` powinna być katalogiem współdzielonym dla maszyny debian-wai.
Uruchomić maszynę debian-wai.
Otworzyć w Notepad++ i przeanalizować zawartość pliku `1.html`
Przetestuj czy strona odpowiada http://192.168.56.10:8080/1.html

1. Pobranie emotikonek za pomocą żądania asynchronicznego

a) Dodamy wywołanie funkcji ajax, które wywoła metodą POST po stronie serwera skrypt E.php. Jednak chcemy pobrać emotki pozytywne lub negatywne które są przechowywane w różnych katalogach więc należy wraz z żądaniem przekazać w polu data parametr żądania np. o nazwie E_typ przekazujący wartość wybranego elementu radio

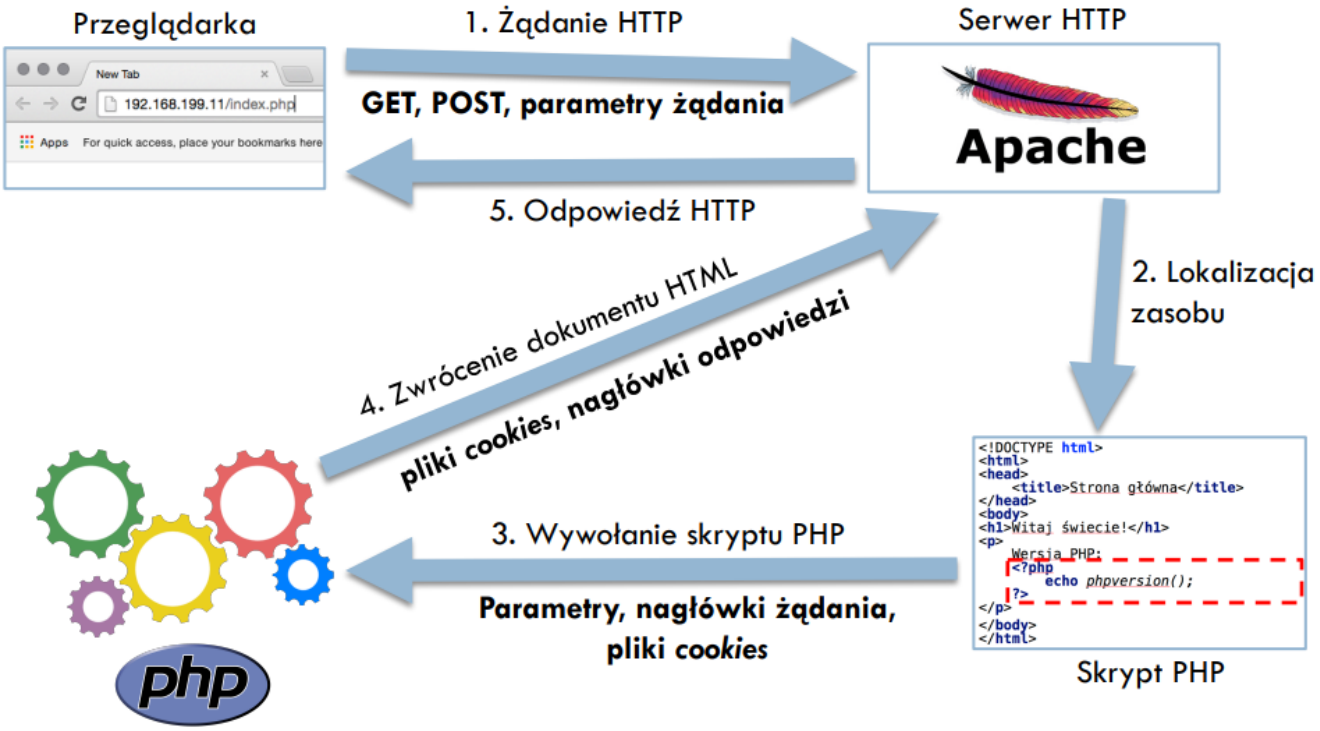
```
$.ajax({
        method: "POST",
        url: "E.php",
        data: { E_typ: param.value }    //wartość z inputa typu
radio
       })
      .done(function( msg ) {
        $('#result').html(msg);
     });
```

Metoda done() wywołuje funkcję anonimową, która umieszcza za pomocą DOM w ciele strony w elemencie o id=’result’ odpowiedź otrzymaną w zmiennej msg.

Wykonaj w przeglądarce żądanie http://192.168.56.10:8080/1.html
Przejrzyj w zakładce Network/Sieć generowane żądania.
Rozpoznajesz żądanie asynchroniczne? - xhr
Czy widzisz przekazywany parametr?

b) zapoznajmy się z przepływem sterowania przy żądaniach o zasób PHP
- Przeglądarka wysyła żądanie o zasób np. index.php
- Apache odbiera żądanie, szuka zasobu i widzi że jest to zasób typu php (czyli nie html od razu do odesłania klientowi), więc należy plik ten przekazać do interpretera PHP

- Upraszczając trochę - interpreter interpretuje kod zawarty w znacznikach <?php … ?>. Jeżeli napotka komendę header, to ustawia odpowiedni nagłówek w sekcji nagłówków paczki protokołu HTTP, jeśli napotka komendę echo “ cos> ” to komenda ta pisze do bufora odpowiedzi (ciało odpowiedzi http) znaczniki html, jeśli wyjdzie z bloku interpretacji i w pliku php napotka kod html to od razu też go wyrzuca w bufor wyjściowy;
- bufor wyjściowy z zawartością html jest przekazywany przez Apache jako paczka protokołu HTTP do klienta



c) Teraz należy utworzyć plik E.php który będzie: odbierał otrzymany parametr i zależnie od niego odczytywał do tablicy zawartość katalogu pozytywnych lub negatywnych emotek oraz na podstawie zawartości tablicy generował odpowiednią odpowiedź html ze znacznikami img.

Na początek należy dać sygnał parserowi, że umieszczamy kod php umieszczając długie przełączniki php

```
<?php

?>
```

Następnie dodajemy wewnątrz blok if, który sprawdzi czy parametr `E_typ` odczytany z tablicy superglobalnej $_POST jest dostępny i czy nie jest on null.

```
if (isset($_POST['E_typ'])) {

}
```

Do odbioru żądań mamy tablicę $_POST, $_GET, $_REQUEST, dlaczego używamy tu $_POST? Dlaczego indekser tablicy wskazuje na `E_typ` a nie na np. `param`. Zerknij w

jaki sposób było wysyłane żądanie asynchroniczne.

Skoro przekazywany parametr jest dostępny to odczytajmy jego wartość do zmiennej i wypiszmy wraz z komunikatem na wyjście do bufora.

```
if (isset($_POST['E_typ'])) {
   $rodzaj = $_POST['E_typ'];
    echo "OK, odebrałem: ". $rodzaj;
}
```

Elementy generowane do bufora stanowią treść odpowiedzi odsyłanej następnie przez serwer Apache do klienta.
Zapisz kod i wywołaj ponownie plik 1.html w przeglądarce wybierając któryś z elementów radio. Prześledź generowane żądanie i otrzymywaną odpowiedź. Zobacz jaki content-type odpowiedzi jest ustawiany.

Następnie zakomentujmy instrukcję echo za pomocą //
Dodamy teraz po niej warunek który na podstawie wartości zmiennej $rodzaj przypisze zmiennej $directory odpowiedni katalog (blik if oraz eleseif).W przypadku gdy parametr posiada inną wartość spoza dopuszczalnego należy umieścić kod wymuszenia przekierowania do strony wyjściowej i przerwać bezwarunkowo wykonanie skryptu php bez generowania odpowiedzi do bufora.

```
// echo "OK, odebrałem: ". $rodzaj;
if ( $rodzaj=="P" )  { $directory= './E_positive'; }
    elseif  ( $rodzaj=="N" ) { $directory= './E_negative'; }
      else  {
         exit;
      }
```

Uruchom i sprawdź poprawność działania kodu.

Doświadczenie.

- Zmień sposób komunikacji z POST na GET (popraw generowanie żądania po str.klienta + tablica superglobalna w php).
- Zaobserwuj jaką teraz ma postać Request URL.
- Skopiuj go, wklej w drugim oknie przeglądarki w pasek adresu zmieniając wartość parametru na np. X. (Możesz też zamiast tego dodać trzeci przycisk radio z wartością X w stronie html.)
- Zaobserwuj co jest teraz generowane przez skrypt php.
- Zaobserwuj występujący ciąg przekierowań żądań HTTP. Zauważ że serwer odpowiedział kodem 302 i ustawił w odpowiedzi nagłówek Location, co wygenerowało nowe żądanie o zasób 1.html.

Powróćmy do żądania POST zamiast GET (popraw generowanie żądania po str.klienta + tablica superglobalna w php).

Skoro już wiemy w kodzie jakiego użyć katalogu , to należy odczytać zawarte w nim nazwy plików. Zatem użyjmy w tym celu funkcji scandir(). Wynik jest tablicą która zawiera również odwołanie do katalogu bieżącego i nadrzędnego. Aby wykluczyć je z

naszej tablicy plików wykonamy różnicę tablic wyniku listingu i tablicy z elementami do pozbycia się.

```
$scanned_directory = array_diff(scandir($directory),
array('..', '.'));
print_r( $scanned_directory);
```

Komenda print_r wyświetli w przyjazny sposób zawartość wynikowej tablicy.
np. dla pozytywnych emotikonek:

Array ([2] => 1.PNG [3] => 2.PNG [4] => 4.PNG [5] => 5.PNG)

Zaobserwuj działanie kodu.

Zakomentuj komendę print_r. Teraz zajmiemy się wygenerowaniem tylu znaczników img z atrybutem src wskazującym na nazwę pliku ile mamy elementów odczytanych w tablicy.

```
//print_r( $scanned_directory);
foreach ($scanned_directory as $plik) {
   echo '<img src="' . $directory . '/'. $plik . '"
alt="emotka" ><br />';

}
```

Przetestuj. Czy zakończenie skryptu `?>` jest konieczne na końcu pliku php?

2. Dwa w jednym- czyli rezygnujemy z plików html

Teraz połączymy pliki 1.html i E.php w jeden plik Emoti.php o tym samym działaniu.

a) główną konstrukcję pliku Emoti.php uzależniamy od rozróżnienia jakiego typu żądanie zostało odebrane - post czy get.

```
<?php

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
//jest żądanie przesłania danych AJAX
}
else {
 //jest żądanie typu GET - pobrania strony do wyświetlenia
 // tu w sumie nie musimy nic robić
}
?>
```

b) całą zawartość skopiowaną z pliku 1.html umieść w pliku Emoti.php po znaku domknięciu przełącznika php - `?>`
Wywołaj w psku adresu żądanie: http://192.168.56.10:8080/Emoti.php

Uwaga! następuje żądanie typu GET (zobacz w komunikacji sieciowej narzędzi

deweloperskich)

W odpowiedzi otrzymano tą samą stronę co z pliku 1.html, bo w php po sprawdzeniu że nie jest to żądanie typu POST tylko GET nic nie ma w bloku else. Można by było cały blok else pominąć. Dalej następuje przekazanie kodu html do strumienia wyjściowego.

c) obsłużymy teraz naszego AJAX'a, czyli żądania typu POST

Cały nasz dotychczasowy kod z pliku E.php umieszczamy w bloku if:

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
//jest żądanie przesłania danych AJAX
 if (isset($_GET['E_typ'])) {
     $rodzaj = $_GET['E_typ'];
     // echo "OK". $rodzaj;
     if ( $rodzaj=="P" )  { $directory= './E_positive'; }
      elseif  ( $rodzaj=="N" ) { $directory= './E_negative'; }
        else  {
           header("Location: 1.html");
           exit;
        }

    $scanned_directory = array_diff(scandir($directory),
array('..', '.'));
      //print_r( $scanned_directory);

    foreach ($scanned_directory as $plik) {
          echo '<img src="' . $directory . '/'. $plik . '"
alt="emotikonka" ><br />';

    }
 }
exit;
}
```

Dodatkowo przed końcem bloku obsługi żądania post przerywamy metodą exit działanie skryptu, by html nie został wypchnięty w output_buffer.
Testujemy działanie strony Emoti.php.

AJAX nie działa. Należy oczywiście poprawić adresata żądań asynchronicznych- już nie `url: "E.php"`, a Emoti.php

Uwaga! Pierwsze żądanie o zasób `Emoti.php` idzie metodą GET, ale już po kliknięciu w przycisk radio jest w JS wysyłane żądanie asynchroniczne metodą POST, więc w kodzie PHP wejdziemy do bloku if, wykona się kod generujący ciąg znaczników <img>, który zostanie przekazany do bufora wyjściowego. Następnie Apache odeśle ten kod do klienta jako odpowiedź na żądanie. Kod PHP w bloku if kończy się komendą exit dlatego html'owy kod strony z końca pliku nie zostanie wystawiony do bufora.

3. **Include i require**
   a) Include - dołączaj tak do kodu pliku potrzebne, ale nie krytyczne

Zduplikuj plik Emoti.php i nazwij go Emoti2.php. Teraz wytnij część html od <!doctype aż po koniec </head> i umieść w pliku Nagl.php. Cała zawartość tego pliku zostanie zainkludowana (dołączona) dynamicznie w obsłudze żądania GET.

```
...
else {
 //jest żądanie typu GET - pobrania strony do wyświetlenia
 include 'Nagl.php';
}
?>

<body>
...
```

Przetestuj działanie strony. Pamiętaj o poprawieniu url: na Emoti2.php w żądaniu AJAX. Dołączanie do kodu php kawałkow innych plików rozwiązuje problem jaki mieliśmy w html z tym samym kodem na wielu stronach np. stopką strony czy menu.

A co się stanie jeśli plik ten nie będzi dostępny? Zrób literówkę w nazwie dołączanego pliku np. `include '1Nagl.php';`

Otrzymaliśmy jedynie ostrzeżenia:
Warning: include(1Nagl.php): failed to open stream: No such file or directory in /var/www/dev/src/web/Emoti2.php
Warning: include(): Failed opening '1Nagl.php' for inclusion (include_path='.:/usr/share/php') in /var/www/dev/src/web/Emoti2.php

Jednak strona się załadowała i by dodatkowo działała (pomijając brak formatowania JQuery UI) jeśli załączenie skryptów jQuery i jQuery UI było na końcu dokumentu po div id=”result” a nie w sekcji head. Możesz przećwiczyć jeśli wydaje Ci się to dziwne.

b) Require - dołączaj tak plik krytycznie potrzebne z uwagi na działanie kodu

Skopiuj wylistowanie katalogu i umieść je w pliku AllEmoti.php po znaczniku otwierającym <?php

```
<?php
$scanned_directory = array_diff(scandir($directory),
array('..', '.'));
```

W miejsce tej linii wstaw załączenie pliku z dodatkowym dołączanym kodem:

```
require 'AllEmoti.php';
```

Przetestuj działanie. Zauważ iż plik AllEmoti.php nie kończy się zamykającym znacznikiem php postaci `?>`
Nie ma konieczności jego wstawiania wciąż jesteśmy w trybie interpretera- tak jakbyśmy rozsunęli linie kodu i włożyli w to miejsce kod z zewnętrznego pliku.

Teraz proszę wprowadzić literówkę w nazwie dołączanego zasobu (np 1AllEmoti.php) i zaobserwować wynik działania kodu.

Warning**: require(1AllEmoti.php): failed to open stream: No such file or directory in **/var/www/dev/src/web/Emoti2.php** on line **...

Fatal error**: require(): Failed opening required '1AllEmoti.php' (include_path='.:/usr/share/php') in **/var/www/dev/src/web//Emoti2.php** on line **...

Wygenerowany został błąd krytyczny i działanie kodu od tego miejsca zostało wstrzymane. Zasadnie bo zmienna `$scanned_directory` nie byłaby znana w dalszych liniach kodu pętli.

Zamień instrukcję `require 'AllEmoti.php';` na include i zaobserwuj działanie. Otrzymujemy dodatkowo uwagi i ostrzeżenia:

***Notice**: Undefined variable: scanned_directory*
***Warning**: Invalid argument supplied for foreach()*

a kod próbuje się dalej wykonywać ale bez powodzenia.

c) Obie instrukcje mają jeszcze wersję dołączania jednokrotnego: include_once, require_once
Doczytaj tu:
- https://www.php.net/manual/en/function.include-once.php
- https://www.php.net/manual/en/function.require-once.php

d) Przećwiczymy teraz funkcje.
Załóż nowy plik o nazwie AJAX.php umieść w nim funkcję, a w niej poniższy kod którym obsługiwaliśmy żądanie ajax'owe z drobnymi modyfikacjami.

```
<?php
function robAJAX( $rodzaj) {
   if ( $rodzaj=="P" )  { $directory= './E_positive'; }
     elseif  ( $rodzaj=="N" ) { $directory= './E_negative'; }
      else  {
         header("Location: 1.html");
         exit;
      }

       $scanned_directory = array_diff(scandir($directory),
array('..', '.'));
     $zdjecia= '';
     foreach ($scanned_directory as $plik) {
        $zdjecia= $zdjecia . '<img src="' . $directory . '/'.
$plik . '" alt="emotikonka" ><br />';
      }
     return $zdjecia;
}
```

Modyfikacja polega na dodaniu elementów wytłuszczonych tj. zdefiniowaniu zmiennej `$zdjecia` do której to w pętli doklejamy za każdym przebiegiem pętli kolejny znacznik

img, a na końcu funkcji wartość zmienej `$zdjecia` jest zwracana jako wynik działania funkcji.

W pliku Emoti2.php na samej górze np w 2 linii kodu dołącz plik AJAX.php (jest to krytyczny zasób z punktu działania kodu więc użyjemy require)
`require 'AJAX.php';`

Następnie należy umieścić wywołanie funkcji `robAJAX()` w boku kodu if odpowiadającemu warunkowi gdy parametr E_typ jest ustawiony. Czyli po linii kodu `if (isset($_POST['E_typ'])) {`
Zauważ, iż funkcja ta wywoływana jest z parametrem. Przekazujemy do funkcji wybrany typ emotki czyli `$_POST['E_typ']` i jednocześnie funkcja zwraca jako wynik działania ciąg znaków (kolejne znaczniki img). Zwracany ciąg znaków należy przekazać do outputbuffora za pomocą `echo.` Zatem wywołanie tej funkcji w 2 linii kodu przyjmie postać :

```
echo robAJAX($_POST['E_typ']);
```

Teraz należy usunąć z tego bloku nadmiarowy kod (starej obsługi bez funkcji). Nasz blok kodu powinien mieć finalnie postać (boldem oznaczyłam nową zawartość):

```
<?php
require 'AJAX.php';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
//jest żądanie przesłania danych AJAX
 if (isset($_POST['E_typ'])) {
   echo robAJAX($_POST['E_typ']);
    }
 exit;
}
else {
 //jest żądanie typu GET - pobrania strony do wyświetlenia
    include 'Nagl.php';
}
?>
<!--dalej część html -->
```

Przetestuj czy obsługa żądania asynchronicznego po dokonanych modyfikacjach wciąż działa.

4. Przesłanie pliku emotki na serwer

Zduplikuj plik Emoti.php z pkt 2 i nazwij go EmotiUpload.php

a) dodamy formularz pozwalający na wskazanie pliku i przesłanie go na stronę serwera

Umieść poniższy kod formularza np. między sekcją a div'em na wynik asynchroniczny

```
...
</section>

 <form enctype="multipart/form-data" action="EmotiUpload.php"
```

```
method="POST">
   <input type="hidden" name="MAX_FILE_SIZE" value="30000" />
   <!--  Name input'a jest nazwą w tablicy $_FILES -->
    Wskaż plik emotiki: <input name="EmotiFile" type="file"
/>
    <input type="submit" value="Send File" />
 </form>

 <div id="result">
...
```

Zwróć uwagę na użyty `enctype`, `method` i do kogo wysyłamy plik (`action`).
Ukryty input MAX_FILE_SIZE musi być przed inputem wskazującym plik i ogranicza wielkość przesyłanego pliku, który można poprawnie dołączyć do formularza.
Natomiast nazwa inputa wyboru pliku z dysku jest kluczowa, bo pod taką właśnie nazwą będziemy odczytywać plik po stronie serwera z tablicy superglobalnej $_FILES[].

W sumie będziemy chcieli przesyłany plik umieścić w odpowiednim katalogu E_negative lub E_positive zależnie od zadeklarowanego typu emotikony. Zatem cały znacznik <section> też powinien znaleźć się w ramach formularza np.zaraz za elementem <form>. Skopiuj go tam.

b) odebrane pliku po stronie serwera z tablicy $_FILES i zapisanie emoitkony do właściwego katalogu

Najpierw należy uporządkować sekcję działania kodu odpowiedzialną za odbieranie komunikacji POST ponieważ mamy żądanie POST dla żądania asynchronicznego oraz żądanie POST w przypadku dobierania danych z formularza.
Wyczyśćmy najpierw całą tą sekcję z bieżącego kodu - będziemy na nowo kolejno go umieszczać. Zatem powinniśmy mieć postać:

```
...
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
   //tu wycięliśmy zawartość

} // od POST'a
else {
...
```

Teraz przystąpmy do ustalenia struktury warunków w ramach obsługi odebranego żądania POST (dodany kod oznaczono wytłuszczeniem)

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {

 if  (isset($_POST['E_typ'])) {
     // typ jest wybrany => więc tylko AJAX
     echo robAJAX($_POST['E_typ']);
     exit;
    }
elseif ( (isset($_FILES["EmotiFile"])) &&
(isset($_POST['radio-1'])) ) {
    //plik i typ emotikony jest ustawiony
    echo "tu będzie odbiór pliku";
```

```
            }
            else
              { echo  "Wybierz parametry <a href='EmotiUpload.php'
          >Wróć</a>";
                exit;
                 }

          } // od POST'a
```

Przetestuj działanie kodu, czy zachowuje się właściwie:

- zaraz po załadowaniu strony, nic nie jest wskazane - klikamy SendFile. Powinien wyświetlić -> wybierz parametry Wróć
- klikając różny typ emotki - AJAX powinien zadziałać dociągając odpowiednie grafiki emotek;
- typ emotki wybrany i wskazujemy plik do upladu, klikamy SendFile. Powinien wyświetlić - > tu będzie odbiór pliku
- po nowym załadowaniu EmotiUpload.php wybieramy plik do uploadu bez wskazywania typu emotki i klikamy SendFile. Powinien wyświetlić -> wybierz parametry Wróć

Teraz skupimy się na samym kodzie obsługi odbierania pliku. Zakomentuj linię `echo "tu będzie odbiór pliku";`

```
     elseif ((isset($_FILES["EmotiFile"])) &&
(isset($_POST['radio-1'])) ){
      //plik i typ emotikony jest ustawiony
      // echo "tu będzie odbiór pliku";

       if ($_FILES["EmotiFile"]["error"] == UPLOAD_ERR_OK ){
          // plik przeszedł poprawnie i typ emotki ustawiony
          echo "plik w katalogu tymczasowym uploadu";
        }
        else {
           //plik załączono ale jest problem z jego przesłaniem
           echo  "Jest problem z tym plikiem <a
href='EmotiUpload.php' >Wróć</a>";
         }



     }else
     ...
```

Zwróć uwagę iż dane o przesłanym pliku są odczytywane z tablicy `$_FILES["EmotiFile"]`, gdzie EmotiFile jest w formularzu nazwą pola inputu służącego do wskazania pliku do przesłania. Każdy element tej tablicy jest też tablicą w której są dodatkowe informacje. Zatem spod indeksu **["error"]** odczytamy czy wystąpiły jakieś problemy z przesyłanym plikiem.

Zaobserwujemy to dokładniej umieszczając poniższy wyboldowany kod:

```
if ($_FILES["EmotiFile"]["error"] == UPLOAD_ERR_OK ){
      // plik przeszedł poprawnie i typ emotki ustawiony ->ulokować na serwerze
        echo "plik w katalogu tymczasowym uploadu";
        echo '<pre> Here is some more debugging info:<br >';
       print_r($_FILES);
       print "</pre>";
}
```

Wyświetlona postać będzie zbliżona do:

```
  Here is some more debugging info:
 Array
 (
     [EmotiFile] => Array
         (
             [name] => k7.png
             [type] => image/png
             [tmp_name] => /tmp/phpsZnNfc
             [error] => 0
             [size] => 12376
         )

 )
```

Zastanów się przez chwilę jak wyglądałby kod jeśli w formularzu mielibyśmy kilka inputów przesłania pliku na serwer. Rozwiązanie takiej kwestii znajdziesz w Example#3 na stronie https://www.php.net/manual/en/features.file-upload.post-method.php

Należy podkreślić iż plik wciąż znajduje się w katalogu tymczasowym na serwerze. Więcej parametrów konfiguracji odnośnie tego katalogu znajdziesz w pliku php.ini (położenie katalogu, ograniczenie na wielkość plików które mogą być w nim umieszczone itp.).

Zajmijmy się teraz właściwym przekopiowaniem przesyłanego pliku z katalogu tymczasowego do katalogu docelowego.
Można zakomentować dodatkowe info, które już nie jest nam potrzebne.

```
if ($_FILES["EmotiFile"]["error"] == UPLOAD_ERR_OK ){
    //  plik przeszedł poprawnie i typ emotki ustawiony ->ulokować na serwerze
    // echo "plik w katalogu tymczasowym uploadu";
    // echo '<pre> Here is some more debugging info:<br >';
     // print_r($_FILES);
     // print "</pre>";

   $uploaddir = '/var/www/dev/src/web/';
   $uploadfile = $uploaddir . basename($_FILES['EmotiFile']['name']);

   if (move_uploaded_file($_FILES['EmotiFile']['tmp_name'],
$uploadfile)) {
          echo "Plik przesłany poprawnie \n";
```

```
    }
  else { echo "Possible file upload attack!\n"; }

}
```

Zmienna `$uploaddir` wskazuje na lokalizację katalogu docelowego, do którego należy przekopiować plik.
Zmienna `$uploadfile` jest połączeniem docelowej ścieżki i właściwej nazwy pliku uploadowanego.
Przekopiowanie pliku następuje za pomocą funkcji `move_uploaded_file(from,to),` funkcja zwraca true jeśli operacja przebiegła pomyślnie.

No dobrze ale w chwili obecnej przesyłany plik ląduje w głównym katalogu, a nie w odpowiednim wedle wskazania typu emotikony. Zatem zajmijmy się tą sprawą.
Dodatkowo zmienna `$uploaddir` ma na stałe ustaloną wartość `'/var/www/dev/src/web/'`. Lokalizacja ta może być inna na innym serwerze jeśli byśmy przenosili kod między różnymi środowiskami. Zatem poprawniejsze jest wykorzystanie $_SERVER['DOCUMENT_ROOT'] wskazującego aktualną ścieżkę głównego katalogu aplikacji na serwerze.

```
//$uploaddir = '/var/www/dev/src/web/';
$uploaddir = $_SERVER['DOCUMENT_ROOT'];

if ( $_POST['radio-1']=="P" ){
   $uploaddir = $uploaddir . '/E_positive/';
}
 else  { $uploaddir = $uploaddir . '/E_negative/'; }
echo $uploaddir;
```

Zweryfikujemy poprawność przesyłu pliku, czy faktycznie plik ląduje we właściwym miejscu.
Dwa testowe pliki do uploadu dostępne są w katalogu MojeEmotki.
Uwaga w środowisku produkcyjnym dodatkowo wymagane jest dodanie praw pisania do katalogu - wykorzystaj chmod z poziomu linii komend by kod php miał możliwość dostępu do docelowego katalogu.

5. Problem post-redirect-get