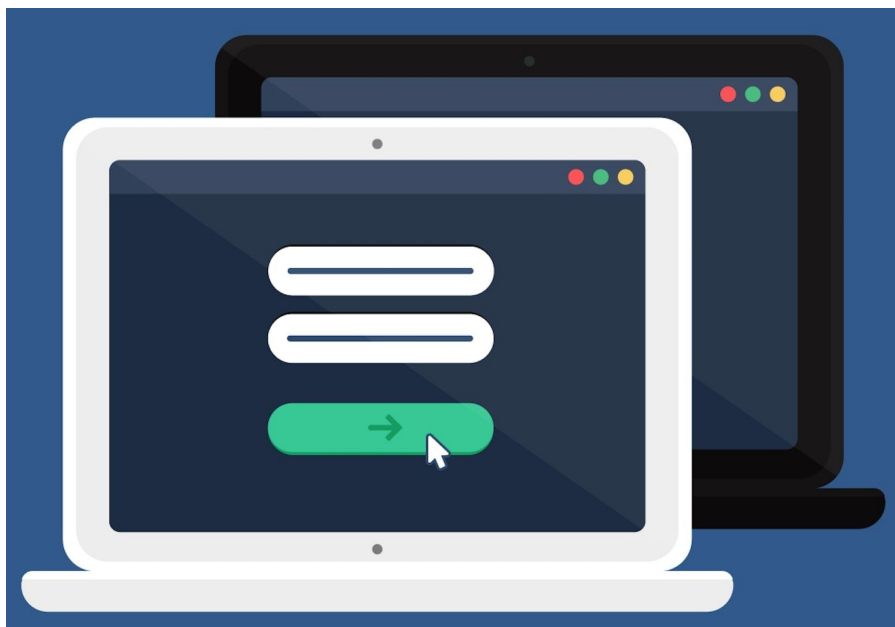


## Prosta i bezpieczna strona logowania w PHP

# Tutorial krok po kroku

---



## Strona logowania PHP w 10 minut

Chcesz stworzyć własną stronę logowania w PHP? Zrobić to w prosty, ale skuteczny sposób, pamiętając o bezpieczeństwie?

Krok po kroku przeprowadzę Cię przez ten proces, co będzie prawdziwą zabawą.

Zacznijmy od utworzenia pustego folderu na dysku lokalnym. Wybierz swój ulubiony edytor kodu (mój to Visual Studio Code) i zacznijmy od podstaw.

## KROK 1: Formularz logowania w HTML

Jeśli nie chce Ci się czytać, zobacz ten fragment w postaci video na Youtube:


<https://www.youtube.com/watch?v=G8CkEBnOTQI>

Utwórz pusty plik `index.php` i przygotuj formularz HTML. Ponieważ ma to być prosty tutorial, zrobimy to klasycznie bez żadnych udziwnień.

Dodaj strukturę dokumentu HTML do pliku, a wewnątrz tagów `<body>` `</body>` wstaw formularz z dwoma polami typu `input` i przyciskiem. Te pola wejściowe wykorzystamy do podawania loginu i hasła, więc wybierz dla nich odpowiednie typy. Login powinien być typu „text”, a hasło to „password”.

Po prostu zobacz kod, a my przejdziemy przez niego wiersz po wierszu:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>login tutorial</title>
</head>
<body>
  <form action="login.php" method="post">
    <input type="text" name="login" />
    <br/>
    <input type="password" name="password" />
    <br/>
    <button type="submit">log in</button>
  </form>
</body>
</html>
```



Nie znajdziesz tutaj nic szczególnego. To tylko zwykła biała strona z dwoma polami i przyciskiem.

Dwa najważniejsze atrybuty tagu formularza to **action** i **method**. W scenariuszu logowania chcemy, aby w polu **action** znalazł się plik przetwarzający dane logowania i tworzący sesję użytkownika (jeśli dane są prawidłowe). **Metoda** musi być POST, ponieważ nie chcesz umieszczać hasła użytkownika w parametrach adresu URL (jak robi to metoda GET).

Co również ważne w naszym przypadku, zwróć uwagę na atrybut **name** obok inputów. Będziemy używać tych nazw podczas odczytywania danych po stronie serwera, więc miej je na uwadze. Użyłem „login” do wprowadzenia loginu i „password” do hasła.

Poza tym nie ma nic więcej.

Możesz oczywiście dodać style i więcej elementów do struktury DOM. Jest to jednak poza zakresem tego przewodnika.

Ponieważ formularz jest gotowy, a użytkownik może wprowadzić dane, nadszedł czas, aby utworzyć plik `login.php`, który może obsłużyć dane wejściowe.

## KROK 2: Odczyt danych po stronie backendu PHP

Mamy pusty plik `login.php`. Dodajmy do niego trochę kodu.

Przede wszystkim musimy przeczytać, jakie dane użytkownik wprowadził do inputów.

Jak zapewne wiesz, możesz to zrobić za pomocą [zmiennej superglobalnej](#). Bardziej precyzyjnie, tablica `$_POST`. To miejsce, w którym lądują wszystkie dane z POST.

```
if (!empty($_POST['login']) && !empty($_POST['password']))  
{  
    var_dump($_POST);  
}
```


Aby sprawdzić, co naprawdę jest w środku tej tablicy, możesz najpierw rzucić całą jej zawartość przy pomocy instrukcji `var_dump` (jeśli pola nie są puste). Skorzystamy tu z metody `empty` do sprawdzenia każdego pola. Jeśli wykonałeś to ćwiczenie, prawdopodobnie zauważyłeś te same wartości, które wpisałeś w formularzu HTML.

Dobra robota!

Teraz musimy porównać je z odpowiednimi wartościami. Ponieważ tym razem nie łączymy się z żadną bazą danych, przechowamy wartości w skrypcie. Możemy założyć, że zostały pobrane z bazy danych.

Hardkodowanie tych wartości nie jest dobrą praktyką! Zdecydowanie unikajcie tego w środowisku produkcyjnym. Robimy to wyłącznie w celach demonstracyjnych.

```
$loginToTest = "admin";  
$passToTest = "pass123";
```



Pamiętaj, że o ile trzymanie loginu w postaci zwykłego tekstu jest w porządku, o tyle z hasłem już nie powinienes tego robić.

W przypadku haseł sprawy mają się zupełnie inaczej.

**Nigdzie nie wolno ich przechowywać w czystej postaci!** Nawet Ty, jako właściciel aplikacji i główny programista, nigdy nie powinienes wiedzieć, jakich haseł używają Twoi użytkownicy.

Ponadto, gdy ktoś używa tego samego hasła co inny użytkownik, nie powinienes być tego świadomy. Dlatego też hasła przechowywane w bazie danych są różne różne dla każdego użytkownika, nawet jeśli używają tych samych danych do logowania.

Czekaj, że co?

Jak to osiągnąć?

## KROK 3: Hasło w plain tekście i korzystanie z hashy

Nie martw się, mój przyjacielu.

Nagrałem kolejny film, specjalnie dla Ciebie:

<https://www.youtube.com/watch?v=6jojFNA44xo>

Jest to całkowicie wykonalne dzięki zastosowaniu dwóch mechanizmów (faktycznie zaimplementowanych w jednej metodzie haszującej):

1 . Hash

2 . Sól

Hash jest jednokierunkowym mechanizmem szyfrowania (więc nie jest to szyfrowanie, ale ludziom łatwiej jest sobie to wyobrazić, gdy w ten sposób staram się tłumaczyć, czym jest hash).

Dostępnych jest wiele algorytmów mieszających. Pomysł na wszystkie z nich jest prosty. Pozwól przekształcić tekst w inny tekst (relatywnie losowy – bez znaczenia) w stosunkowo krótkim czasie i za każdym razem, gdy przekształcam ten sam tekst, otrzymuję ten sam wynik hashowania.

W ten sposób zawsze mogę sprawdzić, czy użytkownik używa prawidłowego hasła, ponieważ porównuję jego hash z haszem zapisanym w bazie danych (wygenerowanym podczas rejestracji konta użytkownika). Jeśli są one równe, użytkownik używa tego samego, prawidłowego hasła.

Mała uwaga na temat kolizji w algorytmach HASH.

Istnieje niewielka mikro szansa (niezwykle mała, bliska zero), że dwa różne hasła wygenerują ten sam wynik mieszania. Szczerze mówiąc, nie powinienes się tym przejmować.

Jednak, gdy korzystamy wyłącznie z samego haszowania, pojawia się problem. Kiedy używam hasła „P@ssw0rd” i ktoś inny używa tego samego, powinniśmy uzyskać ten sam hasz przechowywany w bazie. Dlatego, jeśli mam dostęp do danych, mogę sprawdzić, czy ktoś ma ten sam hasz, co ja, i zalogować się przy użyciu mojego hasła do czyjegoś konta.

Wbrew pozorom, nie jest to ekstremalnie rzadka sytuacja. Zdziwiłbyś się, jak dużo ludzi używa prostych haseł w stylu „kasja1”.

Dlatego wprowadzamy drugi mechanizm – sól.

## Sól przychodzi nam z pomocą

Solenie jest prostym procesem dodawania unikatowego stringa do hasła przed hashowaniem.


Jeśli na przykład używam P@ssw0rd jako hasła, P@ssw0rd powinno być opatrzone dodatkowym ciągiem znaków, unikalnym dla każdego użytkownika (np. losowymi znakami w liczbie X). Wtedy dostałbym coś takiego:

*P@ssword6h8Sjms9(7sa*

*Hasło + sól*

a następnie, dopiero ten połączony ciąg jest haszowany i zapisywany do bazy danych.

Pamiętaj, że **musisz również przechowywać sól**, ponieważ bez niej nie możesz sprawdzić, czy użytkownik może się zalogować.



Teraz, gdy ktoś próbuje się zalogować, najpierw pobierasz sól z bazy danych, a następnie dodajesz sól do hasła z inputu, haszujesz wynik i porównujesz z hashem przechowywanym w bazie danych.

Brzmi trochę skomplikowanie, ale wcale takie nie jest. Zaufaj mi.



## KROK 4: Bcrypt złotym środkiem na hasła

Mówiłem ci, że w PHP masz jedną metodę, którą to wszystko obsłużysz.

Mam tutaj na myśli **bcrypt**.

Bcrypt to algorytm haszujący, który wymaga zastosowania soli.

Co ciekawe, w efekcie otrzymujemy sól jako część wygenerowanego hasza, więc nie ma potrzeby tworzenia dodatkowej kolumny tylko z solą. Dodatkowo bcrypt jest na tyle powolny przy wyliczaniu hasza, że stanowi ogromną trudność dla hackerów przy zmasowanym ataku brute-force.

W parze z bryptem idą dwie metody wbudowane w PHP, które w domyśle korzystają z algorytmu bcrypt (choć mogą korzystać również z innych): `password_hash()` i `password_verify()`.

Pierwsza z nich jest używana podczas pierwszego styku aplikacji z hasłem (gdy użytkownik tworzy nowe konto w aplikacji). Drugi służy do porównania ciągu znaków wprowadzanego przez użytkownika z pożądanym hasłem (przechowywanym jako hasz w DB).

W naszym skrypcie również przechowamy hasło w postaci bezpiecznej. Później zawsze możesz zaimplementować połączenie z bazą danych i pobrać z niej dane.

```
$hashToTest = password_hash($passToTest, PASSWORD_DEFAULT);
```

Także, po otrzymaniu wyniku w `$hashToTest`, tylko ten wynik powinien być przechowywany jako rzecz do porównania hasła.

W naszym przykładzie generujemy go w locie, ale za chwilę się to zmieni.

## KROK 5: Ustawiamy zmienną sesyjną użytkownika

Nareszcie... uff.

Myślałem, że nigdy nie przebrniemy przez hasze.

Mówienie o hashach zawsze mnie nakręca.

Po sprawdzeniu poprawności nazwy użytkownika i hasła możesz utworzyć zmienną sesji dla bieżącego użytkownika. Trzeci film możesz zobaczyć tutaj:

<https://www.youtube.com/watch?v=uZyOWmle62c>

Żeby to zrobić, będziemy musieli wykonać kilka kroków. Przede wszystkim dodaj metodę `session_start` na początku pliku. Poinformuje to nasz interpreter PHP, że sesja powinna zostać wznowiona lub powinna zostać utworzona nowa.


```
<?php
session_start();
```

Następnie, po sprawdzeniu haseł metodą `password_verify`, dodaj nową [zmienną sesyjną](#) o poprawnej nazwie użytkownika:

```
if (password_verify($_POST['password'], $hashToTest))
    $_SESSION['user'] = htmlspecialchars($_POST['login']);
```

Ponownie skłaniamy się ku zmiennym superglobalnym – w tym przypadku `$_SESSION`.

Tym razem tworzymy w niej nowy wpis. Zauważ, że użyłem również metody `htmlspecialchars` na `$_POST['login']`, ponieważ chcę wyświetlać tę wartość w strukturze HTML. Nie chcesz, aby ktokolwiek wstrzykiwał jakiś zły kod do twojego skryptu, więc zrób sobie przysługę i zawsze filtruj potencjalne zagrożenia.



`htmlspecialchars` przekształca wszystkie tagi HTML w ich bezpieczną wersję. Escapuje wszystkie tagi HTML. Alternatywnie możesz użyć metody `strip_tags`, która po prostu całkiem usuwa tagi ze stringa. Możesz użyć jednej lub drugiej metody, jak wolisz. Ważne, żeby zabezpieczyć się przed atakiem.

Ok, sesja gotowa, użytkownik jest zalogowany.

Ostatnią rzeczą do zrobienia jest przekierowanie użytkownika z powrotem na stronę `index.php`.

Możesz to osiągnąć dzięki wbudowanej funkcji PHP – `header`.

Zobacz listing:

```
header("Location: http://localhost:8001/index.php");
```

I to byłoby wszystko w sprawie pliku `login.php`.

Teraz czas sprawdzić sesję w pliku `index.php` i wyświetlić komunikat powitalny, gdy ktoś się zaloguje.

## KROK 6: Czy istnieje już sesja użytkownika?

Sprawdźmy, czy sesja użytkownika nie jest już przypadkiem utworzona, gdy ładujesz na formularzu logowania. Przede wszystkim musisz rozpocząć sesję w taki sam sposób, jak w pliku login.php.

Po prostu zacznij od otwarcia tagu PHP i uruchom `session_start()`. Następnie, w strukturze HTML sprawdź warunek IF tuż przed znacznikiem formularza:

```
<body>
    <?php if (empty($_SESSION['user'])) : ?>
        <form action="login.php" method="post">
    <?php //Next, after the form, close the if and write an else clause: ?>
    </form>
    <?php else : ?>
        <p>Hello, <?=$_SESSION['user']?></p>
    <?php endif; ?>
</body>
```

W ten sposób, jeśli użytkownik ma pustą zmienną sesji (jeśli jeszcze się nie zalogował), należy przedstawić mu formularz HTML do zalogowania. W przeciwnym razie przywitaj zalogowanego użytkownika, ponieważ ma już dostęp do Twojej witryny.

Teraz widzisz, dlaczego potraktowaliśmy login metodą zabezpieczającą tagi HTML.

Umieszczamy go bezpośrednio w strukturze HTML, a każdy tag HTML wewnątrz tej zmiennej spowodowałby scalenie ze strukturą HTML (pogrubienie, zerwanie, połączenie itp.)

Od teraz możesz sprawdzić gdzie tylko chcesz, czy istnieje zmienna sesyjna, Na tej podstawie jesteś w stanie stwierdzić, czy użytkownik jest zalogowany i czy ma dostęp do danego zasobu.

Jeśli chodzi o samo logowanie, to chyba dobrnęliśmy do końca. Ostateczną rzeczą jest wylogowanie.

## KROK 7: Wylogowanie i zabicie sesji

Możesz zalogować się do systemu i w ten sposób utworzyć zmienną sesji.

Teraz nadszedł czas, aby zezwolić użytkownikom na wylogowanie się, gdy nie potrzebują już dostępu do systemu.

Utwórzmy kolejny pusty plik o nazwie `logout.php`:

```
<?php
session_start();
unset($_SESSION['user']);
session_destroy();
header("Location: http://localhost:8001/index.php");
```

Ponownie musimy rozpocząć (lub kontynuować) sesję metodą `session_start`.

Następnie używamy `unset` (to funkcja usuwająca zmienną), dzięki której wiemy, że użytkownik nie jest już trzymany w tablicy `$_SESSION`.

Następnie, gdy sesja użytkownika zostanie usunięta, musisz zniszczyć całą sesję, aby zdecydowanie ją zakończyć. Możesz to zrobić za pomocą metody `session_destroy()`.

Na koniec przekieruj użytkownika do witryny `index.php`, tak jak na stronie `login.php`.

## Podsumujmy

To wszystko, co chciałem Ci przekazać, na temat prostego, ale bezpiecznego systemu logowania napisanego w PHP.

Teraz chciałbym usłyszeć co nieco od Ciebie.

Czy po wykonaniu wszystkich kroków udało Ci się zrobić działający mechanizm logowania? A może po drodze napotkałeś jakieś problemy, które Cię powstrzymały?

Napisz do mnie na [marcin@kursphp.com](mailto:marcin@kursphp.com).

Jeśli szukasz dalszych materiałów do nauki, sprawdź [Praktyczne PHP: Stwórz kompletny projekt w PHP od zera](https://praktycznephphp.pl) (praktycznephphp.pl), gdzie znajdziesz darmowy fragment książki “Praktyczne PHP”, poświęcony w całości Programowaniu Obiektowemu w PHP. Tam też możesz kupić swój egzemplarz książki (elektroniczny lub papierowy).

Oczywiście, zapraszam do dalszej nauki na [kursphp.com](https://kursphp.com).