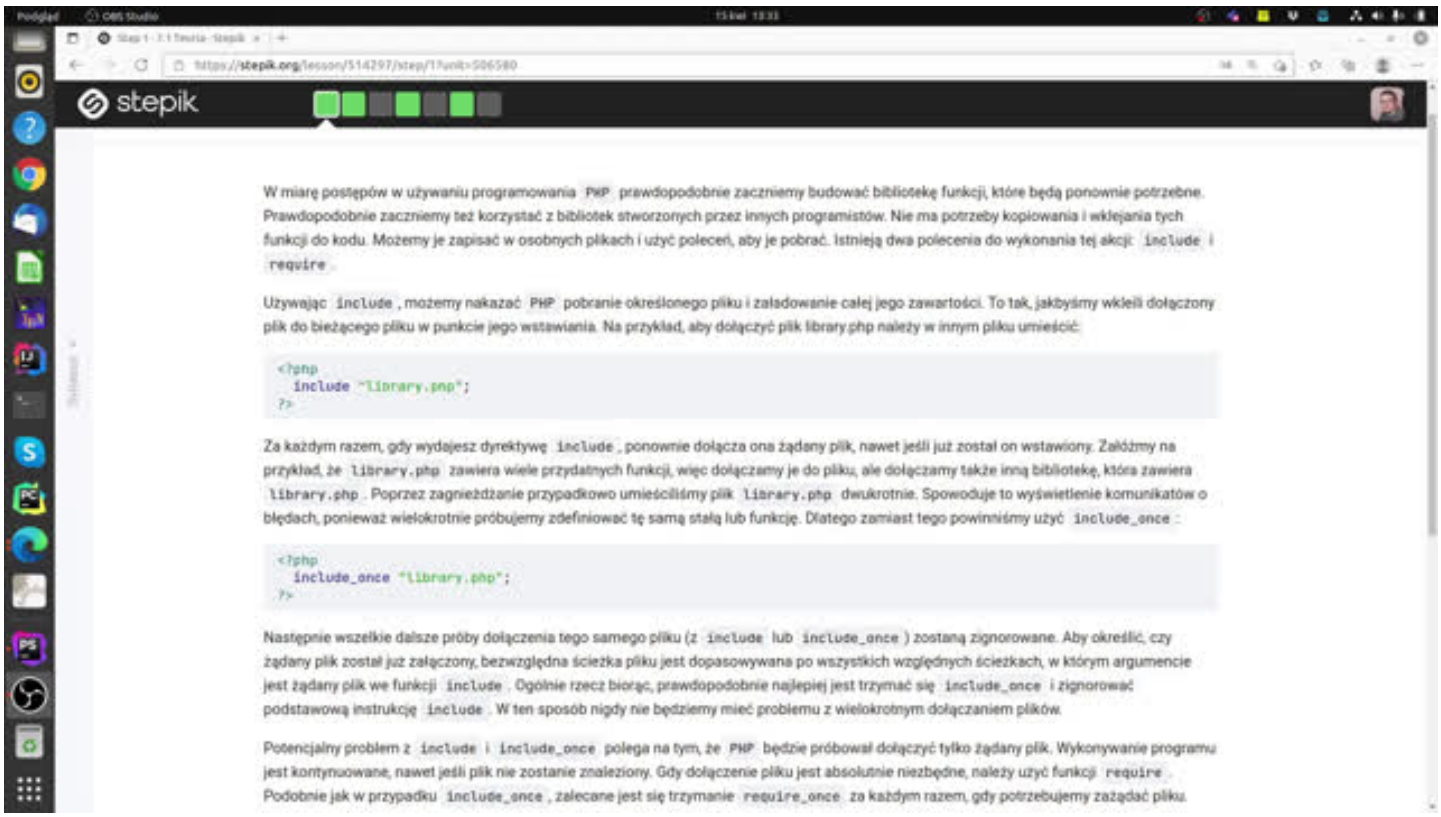
11.1 11.2 Teoria

Step 1



To watch this video please visit https://stepik.org/lesson/514297/step/1

Step 2

W miarę postępów w używaniu programowania `PHP` prawdopodobnie zaczniemy budować bibliotekę funkcji, które będą ponownie potrzebne. Prawdopodobnie zaczniemy też korzystać z bibliotek stworzonych przez innych programistów. Nie ma potrzeby kopiowania i wklejania tych funkcji do kodu. Możemy je zapisać w osobnych plikach i użyć poleceń, aby je pobrać. Istnieją dwa polecenia do wykonania tej akcji: `include` i `require` .

Używając `include` , możemy nakazać `PHP` pobranie określonego pliku i załadowanie całej jego zawartości. To tak, jakbyśmy wkleili dołączony plik do bieżącego pliku w punkcie jego wstawiania. Na przykład, aby dołączyć plik library.php należy w innym pliku umieścić:

```
<?php
  include "library.php";
?>
```

Za każdym razem, gdy wydajesz dyrektywę `include` , ponownie dołącza ona żądany plik, nawet jeśli już został on wstawiony. Załóżmy na przykład, że `library.php` zawiera wiele przydatnych funkcji, więc dołączamy je do pliku, ale dołączamy także inną bibliotekę, która zawiera `library.php` . Poprzez zagnieżdżanie przypadkowo umieściliśmy plik `library.php` dwukrotnie. Spowoduje to wyświetlenie komunikatów o błędach, ponieważ wielokrotnie próbujemy zdefiniować tę samą stałą lub funkcję. Dlatego zamiast tego powinniśmy użyć `include_once` :

```
<?php
  include_once "library.php";
?>
```

Następnie wszelkie dalsze próby dołączenia tego samego pliku (z `include` lub `include_once`) zostaną zignorowane. Aby określić, czy żądany plik został już załączony, bezwzględna ścieżka pliku jest dopasowywana po wszystkich względnych ścieżkach, w którym argumencie jest żądany plik we funkcji `include`. Ogólnie rzecz biorąc, prawdopodobnie najlepiej jest trzymać się `include_once` i zignorować podstawową instrukcję `include`. W ten sposób nigdy nie będziemy mieć problemu z wielokrotnym dołączaniem plików.

Potencjalny problem z `include` i `include_once` polega na tym, że `PHP` będzie próbował dołączyć tylko żądany plik. Wykonywanie programu jest kontynuowane, nawet jeśli plik nie zostanie znaleziony. Gdy dołączenie pliku jest absolutnie niezbędne, należy użyć funkcji `require`. Podobnie jak w przypadku `include_once`, zalecane jest się trzymanie `require_once` za każdym razem, gdy potrzebujemy zażądać pliku.

```
<?php
  require_once "library.php";
?>
```

Step 3

Odpowiedź `HTTP`, którą serwer wysyła z powrotem do klienta, zawiera nagłówki, które identyfikują typ treści w treści odpowiedzi, serwer, który wysłał odpowiedź, ile bajtów znajduje się w treści, gdy odpowiedź została wysłana i tak dalej. `PHP` i `Apache` zwykle zajmują się nagłówkami (identyfikacja dokumentu jako `HTML`, obliczanie długości strony `HTML` itp.). Większość aplikacji internetowych nigdy nie musi samodzielnie ustawiać nagłówków. Jeśli jednak chcemy odesłać coś, co nie jest `HTML`, ustawić czas wygaśnięcia strony, przekierować przeglądarkę klienta lub wygenerować określony błąd `HTTP`, będziemy musieli użyć funkcji `header()`. Jedynym haczykiem związanym z ustawianiem nagłówków jest to, że musimy to zrobić przed wygenerowaniem jakiejkolwiek treści. Oznacza to, że wszystkie wywołania funkcji `header()` (lub `setcookie()`, jeśli ustawiamy pliki cookie) muszą mieć miejsce na samej górze pliku, nawet przed znacznikiem `<html>`.

```
<?php header("Content-Type: text/plain"); ?>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Hangman Game</title>
</head>
<body>
</body>
</html>
```

Natomiast próba ustawienia nagłówków po uruchomieniu dokumentu powoduje wyświetlenie następującego ostrzeżenia:

```
Warning: Cannot add header information - headers already sent
```

Nagłówek `Content-Type` identyfikuje typ zwracanego dokumentu. Zwykle jest to `text/html`, wskazujący na dokument `HTML`, ale są też inne przydatne typy dokumentów. Na przykład `text/plain` zmusza przeglądarkę do traktowania strony jako zwykłego tekstu. Ten typ przypomina automatyczne źródło widoku strony i jest przydatny podczas debugowania. Aby wysłać przeglądarkę pod nowy adres `URL`, zwany **przekierowaniem**, należy ustawić nagłówek `Location`. Wówczas, powinniśmy użyć również funkcji `exit()`, która spowoduje, że skrypt nie zawraca sobie głowy generowaniem i wyświetlaniem pozostałej części listy kodu. Gdy podamy częściowy adres `URL` (np. `/elsewhere.html`), serwer `WWW` obsługuje to przekierowanie wewnętrznie. Jest to rzadko przydatne, ponieważ przeglądarka zazwyczaj nie dowiaduje się, że nie pobiera żądanej strony. Jeśli w nowym dokumencie znajdują się względne adresy `URL`, przeglądarka interpretuje te adresy `URL` jako odnoszące się do żądanego dokumentu, a nie do dokumentu, który został ostatecznie wysłany. Ogólnie rzecz biorąc, będziemy zawsze chcieli przekierować do bezwzględnego adresu `URL`.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Title page</title>
</head>
<body>
 <h1>This is a index.php page</h1>
    <?php
        header("Location: about.php"); #Lub inaczej header("Location:
http://localhost:63342/test_php/about.php");
        exit();
    ?>
</body>
</html>
```

Serwer może jawnie poinformować przeglądarkę i wszelkie pamięci podręczne proxy, które mogą znajdować się między serwerem a przeglądarką, o określonej dacie i godzinie wygaśnięcia dokumentu. Pamięć podręczna serwera proxy i przeglądarki może przechowywać dokument do tego czasu lub wygasać go wcześniej. Wielokrotne przeładowania zbuforowanego dokumentu nie kontaktują się z serwerem. Jednak próba pobrania wygasłego dokumentu kontaktuje się z serwerem. Aby ustawić czas wygaśnięcia dokumentu, należy użyć nagłówka `Expires` :

```
header("Expires: Tue, 02 Jul 2022 05:30:00 GMT");
```

Aby wymusić wygaśnięcie dokumentu po trzech godzinach od wygenerowania strony, należy użyć `time()` i `gmstrftime()` do wygenerowania ciągu daty wygaśnięcia dokumentu:

```
<?php
    $now = time();
    $then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 60 * 60 * 3);
    header("Expires: {$then}");
?>
```

Aby wskazać, że dokument „nigdy” wygasa, należy użyć czasu od obecnego roku do teraz:

```
<?php
    $now = time();
    $then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 365 * 86440);
    header("Expires: {$then}");
?>
```

Aby oznaczyć dokument jako wygasły, należy użyć aktualnej lub przeszłej godziny:

```
<?php
    $then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT");
    header("Expires: {$then}");
?>
```

To najlepszy sposób, aby zapobiec przechowywaniu dokumentu w pamięci podręcznej przeglądarki lub serwera proxy:

```
<?php
    header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
    header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
    header("Cache-Control: no-store, no-cache, must-revalidate");
    header("Cache-Control: post-check=0, pre-check=0", false);
    header("Pragma: no-cache");
?>
```

Step 4

Uwierzytelnianie `HTTP` działa poprzez nagłówki żądań i statusy odpowiedzi. Przeglądarka może wysłać nazwę użytkownika i hasło (poświadczenia) w nagłówkach żądań. Jeśli dane uwierzytelniające nie są wysyłane lub nie są satysfakcjonujące, serwer wysyła odpowiedź `401 Unauthorized` i identyfikuje dziedzinę uwierzytelniania (ciąg, taki jak `Mary's Pictures` lub `Your Shopping Cart`)

za pośrednictwem nagłówka `WWW-Authenticate`. Zwykle pojawia się komunikat `Wprowadź nazwę użytkownika i hasło dla . . .` w przeglądarce, a następnie ponownie zażądana zostanie strona ze zaktualizowanymi poświadczeniami w nagłówku. Aby obsłużyć uwierzytelnianie w `PHP`, należy sprawdzić nazwę użytkownika i hasło (elementy `PHP_AUTH_USER` i `PHP_AUTH_PW` w tablicy `$_SERVER`) i należy wywołać odpowiedni nagłówek, aby ustawić dziedzinę i wysłać odpowiedź `401 Unauthorized`:

```
header('WWW-Authenticate: Basic realm="Top Secret Files"');
header("HTTP/1.0 401 Unauthorized");
```

Możemy zrobić wszystko, co chcemy, aby uwierzytelnić nazwę użytkownika i hasło; na przykład możemy skonsultować się z bazą danych, przeczytać plik z prawidłowymi użytkownikami lub skonsultować się z serwerem domeny Microsoft. W tym przykładzie sprawdzamy, czy hasło jest odwróconą nazwą użytkownika (nie jest to najbezpieczniejsza metoda uwierzytelniania, to dla pewności!):

```
<?php
    $_SERVER['PHP_AUTH_USER'] = 'admin'; // W serwerze cgi nie ma na stałe podanych tych wartości
    $_SERVER['PHP_AUTH_PW'] = 'password1'; // W serwerze cgi nie ma na stałe podanych tych wartości
    $authOK = false;
    $user = $_SERVER['PHP_AUTH_USER'];
    $password = $_SERVER['PHP_AUTH_PW'];
    if (isset($user) && isset($password) && $user === 'admin' && $password === 'password') {
        $authOK = true;
    }
    if (!$authOK) {
        header('WWW-Authenticate: Basic realm="Top Secret Files"');
        header('HTTP/1.0 401 Unauthorized');
        echo "No auth";
    // anything else printed here is only seen if the client hits "Cancel"
        exit;
    }
?>
```

Jeśli chronimy więcej niż jedną stronę, umieść poprzedni kod w osobnym pliku i umieść go na górze każdej chronionej strony. Jeśli nasz host używa wersji `CGI PHP` zamiast modułu `Apache`, te zmienne nie mogą być ustawione i będziemy musieli użyć innej formy uwierzytelniania — na przykład poprzez zbieranie nazwy użytkownika i hasła za pomocą formularza `HTML`.

HTTP jest protokołem bezstanowym, co oznacza, że gdy serwer `WWW` zakończy żądanie klienta dotyczące strony internetowej, połączenie między nimi znika. Innymi słowy, serwer nie może rozpoznać, że sekwencja żądań pochodzi od tego samego klienta. Stan jest jednak przydatny. Nie możemy na przykład zbudować aplikacji koszyka na zakupy, na przykład, jeśli nie możemy śledzić sekwencji żądań od jednego użytkownika. Musimy wiedzieć, kiedy użytkownik dodaje produkty do koszyka lub je usuwa oraz co znajduje się w koszyku, gdy użytkownik decyduje się na zakup. Aby obejść brak stanu sieci, programiści wymyślili wiele sztuczek do śledzenia informacji o stanie między żądaniami (znanych również jako **śledzenie sesji**). Jedną z takich technik jest użycie ukrytych pól formularzy do przekazywania informacji. `PHP` traktuje ukryte pola formularzy jak normalne pola formularzy, więc wartości są dostępne w tablicach `$_GET` i `$_POST`. Korzystając z ukrytych pól formularzy, możemy przekazywać całą zawartość koszyka. Jednak bardziej powszechne jest przypisywanie każdemu użytkownikowi unikalnego identyfikatora i przekazywanie go za pomocą jednego ukrytego pola formularza. Chociaż ukryte pola formularzy działają we wszystkich przeglądarkach, działają tylko dla sekwencji formularzy generowanych dynamicznie, więc nie są tak ogólnie przydatne, jak niektóre inne techniki. Inną techniką jest przepisywanie adresów `URL`, gdzie każdy lokalny adres URL, na który użytkownik może kliknąć, jest dynamicznie modyfikowany, aby zawierał dodatkowe informacje. Te dodatkowe informacje są często określane jako parametr w adresie `URL`. Na przykład, jeśli przypiszemy każdemu użytkownikowi unikalny identyfikator, możemy umieścić ten identyfikator we wszystkich adresach `URL` w następujący sposób:

```
http://www.example.com/catalog.php?userid=123
```

Jeśli zadbamy o dynamiczną modyfikację wszystkich łączy lokalnych, aby zawierały identyfikator użytkownika, możemy teraz śledzić poszczególnych użytkowników w aplikacji. Przepisywanie adresów `URL` działa dla wszystkich dynamicznie generowanych dokumentów, nie tylko formularzy, ale faktyczne przepisywanie tych samych adresów może być żmudne. Trzecią i najbardziej rozpowszechnioną techniką utrzymywania stanu jest użycie plików `cookie`. `Cookie` to trochę informacji, które serwer może przekazać klientowi. Przy każdym kolejnym żądaniu klient przekaże te informacje serwerowi, identyfikując się w ten sposób. Pliki `cookie` są przydatne do przechowywania informacji podczas wielokrotnych wizyt przeglądarki, ale nie są pozbawione własnych problemów. Głównym problemem jest to, że większość przeglądarek umożliwia użytkownikom wyłączenie plików cookie. Tak więc każda aplikacja, która używa plików `cookie` do utrzymania stanu, **musi używać innej techniki jako mechanizmu awaryjnego**.

Najlepszym sposobem na utrzymanie stanu w `PHP` jest użycie **wbudowanego systemu śledzenia sesji**. Ten system umożliwia tworzenie trwałych zmiennych, które są dostępne z różnych stron aplikacji, a także podczas różnych wizyt na stronie przez tego samego użytkownika. Za kulisami mechanizm śledzenia sesji `PHP` wykorzystuje pliki `cookie` (lub adresy URL), aby w elegancki sposób rozwiązać większość problemów wymagających stanu, dbając o wszystkie szczegóły za nas. Plik `cookie` to w zasadzie ciąg znaków, który zawiera kilka pól. Serwer może wysłać jeden lub więcej plików `cookie` do przeglądarki w nagłówkach odpowiedzi. Niektóre pola pliku `cookie` wskazują strony, dla których przeglądarka powinna wysłać plik `cookie` w ramach żądania. Pole wartości pliku `cookie` to ładunek — serwery mogą tam przechowywać dowolne dane (w granicach), takie jak unikalny kod identyfikujący użytkownika, preferencje i tym podobne. Aby wysłać plik cookie do przeglądarki należy użyć funkcji `setcookie()`:

```
setcookie(name [, value [, expires [, path [, domain [, secure [,httponly ]]]]]]);
```

Ta funkcja tworzy ciąg znaków `cookie` z podanych argumentów i tworzy nagłówek `Cookie` z tym ciągiem jako wartością. Ponieważ ciasteczka są wysyłane jako nagłówki w odpowiedzi, `setcookie()` musi zostać wywołana przed wysłaniem jakiejkolwiek części dokumentu. Parametry `setcookie()` to:

- `name` - Unikalna nazwa konkretnego pliku `cookie`. Możemy mieć wiele plików `cookie` o różnych nazwach i atrybutach. Nazwa nie może zawierać spacji ani średników.
- `value` - Dowolna wartość ciągu dołączona do tego pliku `cookie`. Oryginalna specyfikacja `Netscape` ograniczała całkowity rozmiar pliku `cookie` (w tym nazwę, datę wygaśnięcia i inne informacje) do `4 KB`, więc chociaż nie ma określonego limitu rozmiaru wartości pliku `cookie`, prawdopodobnie nie może on być znacznie większy niż `3,5 KB`.
- `expires` - Data ważności tego pliku `cookie`. Jeśli nie określono daty ważności, przeglądarka zapisuje plik `cookie` w pamięci, a nie na dysku. Po zamknięciu przeglądarki plik `cookie` znika. Data ważności jest określona jako liczba sekund od północy 1 stycznia 1970 (GMT). Na przykład `time() + 60 * 60 * 2`, powoduje, że plik cookie wygaśnie za dwie godziny.
- `path` - Przeglądarka zwróci plik `cookie` tylko dla adresów `URL` podanej ścieżki. Domyślnie jest to katalog, w którym znajduje się bieżąca strona. Na przykład, jeśli `/store/front/cart.php` ustawi ciasteczko i nie określi ścieżki, ciasteczko zostanie odesłane z powrotem do serwera dla wszystkich stron, których ścieżka `URL` zaczyna się od `/store/front/`.
- `domain` - Przeglądarka zwróci plik `cookie` tylko dla adresów `URL` w tej domenie. Wartość domyślna to nazwa hosta serwera.
- `secure` - Przeglądarka prześle plik `cookie` tylko przez połączenia `https`. Wartość domyślna to `false`, co oznacza, że można wysłać plik `cookie` przez niezabezpieczone połączenia.
- `httponly` - Jeśli ten parametr zostanie ustawiony na `true`, plik `cookie` będzie dostępny tylko za pośrednictwem protokołu `HTTP`, a tym samym niedostępny za pośrednictwem innych środków, takich jak `JavaScript`. To, czy pozwala to na bezpieczniejszy plik `cookie`, jest nadal przedmiotem dyskusji, więc należy korzystać z tego parametru ostrożnie.

Funkcja `setcookie()` ma również alternatywną składnię:

```
setcookie ($name [, $value = "" [, $options = [] ]] )
```

gdzie `$options` to tablica, która przechowuje inne parametry następujące po zawartości `$value`. Oszczędza to trochę na długości wiersza kodu dla funkcji `setcookie()`, ale tablica `$options` będzie musiała zostać zbudowana przed jej użyciem. Gdy przeglądarka wysyła plik `cookie` z powrotem do serwera, możemy uzyskać dostęp do tego pliku `cookie` za pośrednictwem tablicy `$_COOKIE`. Kluczem jest nazwa ciasteczka, a wartością jest pole wartości ciasteczka. Na przykład poniższy kod śledzi, ile razy strona była otwierana przez tego klienta:

```
$pageAccesses = $_COOKIE['accesses'];
setcookie('accesses', ++$pageAccesses);
```

Podczas dekodowania plików `cookie` wszelkie kropki (`.`) w nazwie pliku cookie są zamieniane na podkreślenia. Na przykład plik cookie o nazwie `tip.top` jest dostępny jako `$_COOKIE['tip_top']`. Przyjrzyjmy się plikom cookie w akcji. Poniższy przykład pokazuje stronę `HTML`, która zawiera szereg opcji kolorów tła i pierwszego planu.

```
<html>
<head><title>Set Your Preferences</title></head>
<body>
<form action="prefs.php" method="post">
    <p>Background:
        <select name="background">
            <option value="black">Black</option>
            <option value="white">White</option>
            <option value="red">Red</option>
            <option value="blue">Blue</option>
        </select><br />
        Foreground:
        <select name="foreground">
            <option value="black">Black</option>
            <option value="white">White</option>
            <option value="red">Red</option>
            <option value="blue">Blue</option>
        </select></p>
    <input type="submit" value="Change Preferences">
</form>
</body>
</html>
```

Formularz jest przesyłany do skryptu `PHP` `prefs.php`. Skrypt ten następnie ustawia pliki `cookie` dla preferencji kolorów określonych w formularzu. Wywołania `setcookie()` są wykonywane po uruchomieniu strony `HTML`.

```
<html>
<head><title>Preferences Set</title></head>
<body>
<?php
  $colors = array(
      'black' => "#000000",
      'white' => "#ffffff",
      'red' => "#ff0000",
      'blue' => "#0000ff"
  );

  $backgroundName = $_POST['background'];
  $foregroundName = $_POST['foreground'];

  setcookie('bg', $colors[$backgroundName]);
  setcookie('fg', $colors[$foregroundName]);
  ?>
  <p>Thank you. Your preferences have been changed to:<br />
Background: <?php echo $backgroundName; ?><br />
Foreground: <?php echo $foregroundName; ?></p>
<p>Click <a href="prefs_demo.php">here</a> to see the preferences in action.</p>
</body>
</html>
```

Powyższa strona zawiera łącze do innej strony, która korzysta z preferencji kolorów, uzyskując dostęp do tablicy `$_COOKIE`.

```
<html>
<head><title>Front Door</title></head>
<?php
    $backgroundName = $_COOKIE['bg'];
    $foregroundName = $_COOKIE['fg'];
?>
<body bgcolor="<?php echo $backgroundName; ?>" text="<?php echo $foregroundName; ?>">
<h1>Welcome to the Store</h1>
<p>We have many fine products for you to view. Please feel free to browse
    the aisles and stop an assistant at any time. But remember, you break it
    you bought it!</p>
<p>Would you like to <a href="index.php">change your preferences?</a></p>
</body>
</html>
```

Istnieje wiele zastrzeżeń dotyczących korzystania z plików `cookie`. Nie wszyscy klienci (przeglądarki) obsługują lub akceptują pliki `cookie`, a nawet jeśli klient obsługuje pliki `cookie`, użytkownik może je wyłączyć. Ponadto specyfikacja plików `cookie` mówi, że żaden plik `cookie` nie może przekroczyć `4 KB`, tylko 20 plików `cookie` jest dozwolonych na domenę, a po stronie klienta może być przechowywanych łącznie 300 plików `cookie`. Niektóre przeglądarki mogą mieć wyższe limity, ale nie możemy na tym polegać. Wreszcie, nie mamy kontroli nad tym, kiedy przeglądarki faktycznie wygasają pliki `cookie` — jeśli przeglądarka działa i musi dodać nowy plik `cookie`, może odrzucić plik `cookie`, który jeszcze nie wygasł. Należy również uważać, aby pliki `cookie` szybko wygasały. Czas wygaśnięcia zależy od tego, czy zegar klienta jest tak dokładny jak nasz. Wiele osób nie ma dokładnie ustawionych zegarów systemowych, więc nie można liczyć na szybkie wygaśnięcie. Pomimo tych ograniczeń pliki `cookie` są bardzo przydatne do przechowywania informacji podczas wielokrotnych wizyt przeglądarki.

Step 5

Napiszmy bardzo prosty mechanizm zapamiętywania danych użytkownika na naszej stronie. Będzie się on składał z następujących plików:

- `index.php` - skrypt zarządzający
- `form.html` - kod formularza
- `header.html` - kod nagłówka strony
- `footer.html` - kod stopki strony

```
<!--form.html-->
<form method="post" action="index.php" xmlns="http://www.w3.org/1999/html">
  <div>
    <label>Wprowadź imię i nazwisko: </label>
    <input type="text" name="nazwa"/><br/>
    <input type="submit"/>
  </div>
</form>
```

```
<!--header.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Zapamiętywanie danych</title>
</head>
```

```
<!--footer.html-->
<body>

</body>
<footer>Stopka</footer>
</html>
```

```
<!--index.php-->
<?php
    if(!isset($_COOKIE['nazwa']) && !isset($_POST['nazwa'])){
        include("header.html");
        include("form.html");
        include("footer.html");
    }
    else if(isset($_POST['nazwa'])){
        setcookie("nazwa",$_POST['nazwa'],time() + 60 * 60 * 24 * 365 );
        include("header.html");
        echo "<p>Dziękujemy za podanie danych.</p>";
        include("footer.html");
    }
    else{
        include("header.html");
        echo "Witamy, zostałeś rozpoznany jako {$_COOKIE['nazwa']}.";
        include("footer.html");
    }
?>
```

Step 6

`PHP` ma wbudowaną obsługę **sesji**, obsługując całą manipulację plikami `cookie`, aby zapewnić trwałe zmienne, które są dostępne z różnych stron i podczas wielu wizyt w witrynie. **Sesje** umożliwiają łatwe tworzenie wielostronicowych formularzy (takich jak koszyki zakupów), zapisywanie informacji uwierzytelniających użytkownika ze strony na stronę oraz przechowywanie trwałych preferencji użytkownika w witrynie. Każdy odwiedzający po raz pierwszy otrzymuje **unikalny identyfikator sesji**. Domyślnie identyfikator sesji jest przechowywany w pliku cookie o nazwie `PHPSESSID`. Jeśli przeglądarka użytkownika nie obsługuje plików `cookie` lub ma wyłączone pliki `cookie`, identyfikator sesji jest propagowany w adresach `URL` w obrębie witryny. Z każdą sesją powiązany jest magazyn danych. Można zarejestrować zmienne, które mają być ładowane z magazynu danych po rozpoczęciu każdej strony i zapisywane z powrotem w magazynie danych po zakończeniu strony. Zarejestrowane zmienne utrzymują się między stronami, a zmiany zmiennych dokonane na jednej stronie są widoczne z innych. Na przykład link `dodaj to do koszyka` może przenieść użytkownika na stronę, która dodaje element do zarejestrowanej tablicy elementów w koszyku. Ta zarejestrowana tablica może być następnie użyta na innej stronie do wyświetlenia zawartości koszyka. Sesje rozpoczynają się automatycznie, gdy skrypt zaczyna działać. W razie potrzeby generowany jest nowy identyfikator sesji, prawdopodobnie tworząc plik `cookie`, który ma zostać wysłany do przeglądarki, i ładuje wszelkie trwałe zmienne ze sklepu. Możemy zarejestrować zmienną w sesji, przekazując nazwę zmiennej do tablicy `$_SESSION[]`. Na przykład, oto podstawowy licznik trafień:

```
<?php
  session_start();
?>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Title page</title>
</head>
<body>
<?php
    $_SESSION['hits'] = $_SESSION['hits'] + 1;
    echo "This page has been viewed {$_SESSION['hits']} times.";
?>
</body>
</html>
```

Funkcja `session_start()` ładuje zarejestrowane zmienne do tablicy asocjacyjnej `$_SESSION`. Kluczami są nazwy zmiennych (np. `$_SESSION['hits']`). Funkcja `session_id()` zwraca bieżący identyfikator sesji. Aby zakończyć sesję, należy wywołać funkcję `session_destroy()`. Spowoduje to usunięcie magazynu danych dla bieżącej sesji, ale nie spowoduje usunięcia pliku `cookie` z pamięci podręcznej przeglądarki. Oznacza to, że podczas kolejnych wizyt na stronach obsługujących sesje użytkownik będzie miał ten sam identyfikator sesji, co przed wywołaniem `session_destroy()`, ale żadnych danych. Przeróbmy poprzedni przykład z użyciem ciasteczek na używanie sesji:

```
<?php session_start(); ?>
<html>
<head><title>Preferences Set</title></head>
<body>
<?php
$colors = array(
    'black' => "#000000",
    'white' => "#ffffff",
    'red' => "#ff0000",
    'blue' => "#0000ff"
);
$bg = $colors[$_POST['background']];
$fg = $colors[$_POST['foreground']];
$_SESSION['bg'] = $bg;
$_SESSION['fg'] = $fg;
?>
<p>Thank you. Your preferences have been changed to:<br />
    Background: <?php echo $_POST['background']; ?><br />
    Foreground: <?php echo $_POST['foreground']; ?></p>
<p>Click <a href="prefs_sessions.php">here</a> to see the preferences
    in action.</p>
</body>
</html>
```

```
<?php
    session_start() ;
    $backgroundName = $_SESSION['bg'];
    $foregroundName = $_SESSION['fg'];
?>
<html>
<head><title>Front Door</title></head>
<body bgcolor="<?php echo $backgroundName; ?>" text="<?php echo $foregroundName; ?>">
<h1>Welcome to the Store</h1>
<p>We have many fine products for you to view. Please feel free to browse
    the aisles and stop an assistant at any time. But remember, you break it
    you bought it!</p>
<p>Would you like to <a href="index.php">change your preferences?</a></p>
</body>
</html>
```

Sesje nie są zachowywane po tym, jak przeglądarka przestaje istnieć. Aby to zmienić, musimy ustawić opcję `session.cookie_lifetime` w `php.ini` na czas życia pliku `cookie` w sekundach.

Step 7

Oto kolejny przykład z wykorzystania sesji w języku `PHP`.

```
<!--index.php-->
<?php
    session_start();
    $_SESSION['zmienna_sesji'] = 'abcd';
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Przykład sesji</title>
</head>
<body>
    <div>
        Witamy na stronie. Została tutaj rozpoczęta sesja.<br/>
        Identyfikatorem sesji jest: <?php echo session_id(); ?><br/>
        Została ustawiona zmienna o nazwie: zmienna_sesji<br/>
        Wartością zmiennej zmienna_sesji jest: <?php echo $_SESSION['zmienna_sesji']; ?><br/>
    </div>
</body>
</html>
```

```
<!--index2.php-->
<?php
    session_start();
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Implementacja sesji</title>
</head>
<body>
<div>
    Witamy na drugiej stronie sesji.<br/>
    Identyfikatorem sesji jest: <?php echo session_id(); ?><br/>
    Została ustawiona zmienna o nazwie: zmienna_sesji<br/>
    Wartością zmiennej zmienna_sesji jest: <?php echo $_SESSION['zmienna_sesji']; ?><br/>
    <a href="index3.php">Następna strona</a>
</div>
</body>
</html>
```

```
<!--index3.php-->
<?php
    session_start();
    unset($_SESSION['zmienna_sesji']);
    if (isset($_COOKIE[session_name()])){
        setcookie(session_name(),'',time() - 360);
    }
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Przykład sesji</title>
</head>
<body>
<div>
    Witamy na trzeciej stronie sesji.<br/>
    Identyfikatorem sesji jest: <?php echo session_id(); ?><br/>
    Została ustawiona zmienna o nazwie: zmienna_sesji<br/>
    Wartością zmiennej zmienna_sesji jest:
    <?php
        echo $_SESSION['zmienna_sesji'] . "<br/>";
        if (session_destroy())
            echo "Sesja została zakończona";
        else
            echo "Próba zakończenia sesji nie powiodła się";
    ?>
</div>
</body>
</html>
```

Step 8

Domyślnie identyfikator sesji jest przekazywany ze strony do strony w pliku `cookie` `PHPSESSID`. Jednak system sesji `PHP` obsługuje dwie alternatywy: **pola formularzy** i **adresy** `URL`. Przekazywanie identyfikatora sesji przez ukryte pola formularzy jest niezwykle niewygodne, ponieważ zmuszają nas, aby każdy link między stronami był przyciskiem przesyłania formularza. Jednak system adresów `URL` służący do przekazywania identyfikatora sesji jest nieco bardziej elegancki. `PHP` może przepisać pliki `HTML`, dodając identyfikator sesji do każdego względnego łącza. Jednak aby to zadziałało, `PHP` musi być skonfigurowane z opcją `-enable-trans-id` podczas kompilacji. Wiąże się to z obniżką wydajności, ponieważ `PHP` musi analizować i przepisywać każdą stronę. Ruchliwe witryny mogą chcieć trzymać się plików `cookie`, ponieważ nie powodują spowolnienia spowodowanego przepisywaniem stron. Ponadto ujawnia to identyfikatory sesji, potencjalnie umożliwiając ataki typu `man-in-the-middle`. Domyślnie `PHP` przechowuje informacje o sesji w plikach w katalogu tymczasowym serwera. Zmienne każdej sesji są przechowywane w osobnym pliku. Każda zmienna jest serializowana do pliku w zastrzeżonym formacie. Wszystkie te wartości możemy zmienić w pliku `php.ini`. Możemy zmienić lokalizację plików sesji, ustawiając wartość `session.save_path` w `php.ini`. `PHP` może przechowywać informacje o sesji

w jednym z dwóch formatów w bieżącym magazynie sesji — albo we wbudowanym formacie `PHP`, albo w **Web Distributed Data eXchange** (`WDDX`). Możemy zmienić format, ustawiając wartość `session.serialize_handler` w pliku `php.ini` na `php` dla zachowania domyślnego lub `wddx` dla formatu `WDDX`.

Korzystając z kombinacji plików `cookie` i własnego modułu obsługi sesji, możemy zachować stan podczas wszystkich wizyt. Każdy stan, który powinien zostać zapomniany, gdy użytkownik opuszcza witrynę, na przykład na której stronie jest użytkownik, może zostać pozostawiony wbudowanym sesjom `PHP`. Każdy stan, który powinien utrzymywać się między wizytami użytkownika, taki jak unikalny identyfikator użytkownika, może być przechowywany w pliku `cookie`. Dzięki identyfikatorowi użytkownika możemy pobrać bardziej trwały stan użytkownika (preferencje wyświetlania, adres pocztowy itp.) ze stałego magazynu, takiego jak baza danych. Kolejny przykład pozwala użytkownikowi wybrać kolory tekstu i tła oraz przechowuje te wartości w pliku `cookie`. Wszelkie wizyty na stronie w ciągu następnego tygodnia wysyłają wartości kolorów w pliku `cookie`.

```
<?php
if($_POST['bgcolor']) {setcookie('bgcolor', $_POST['bgcolor'], time() + (60 * 60 * 24 * 7));
}
if (isset($_COOKIE['bgcolor'])) {
    $backgroundName = $_COOKIE['bgcolor'];
}
else if (isset($_POST['bgcolor'])) {
    $backgroundName = $_POST['bgcolor'];
}
else {
    $backgroundName = "gray";
} ?>
<html>
<head><title>Save It</title></head>
<body bgcolor="<?php echo $backgroundName; ?>">
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
    <p>Background color:
        <select name="bgcolor">
            <option value="gray">Gray</option>
            <option value="white">White</option>
            <option value="black">Black</option>
            <option value="blue">Blue</option>
            <option value="green">Green</option>
            <option value="red">Red</option>
        </select></p>
    <input type="submit" />
</form>
</body>
</html>
```

Protokół **Secure Sockets Layer** (`SSL`) zapewnia bezpieczny kanał, przez który mogą przepływać zwykłe żądania i odpowiedzi `HTTP`. `PHP` nie zajmuje się konkretnie `SSL`, więc nie możemy w żaden sposób kontrolować szyfrowania z `PHP`. Adres `URL` `https://` wskazuje bezpieczne połączenie dla tego dokumentu, w przeciwieństwie do adresu `URL` `http://`. Wpis `HTTPS` w tablicy `$_SERVER` jest ustawiony na `'on'`, jeśli strona `PHP` została wygenerowana w odpowiedzi na żądanie przez połączenie `SSL`. Aby zapobiec generowaniu strony przez nieszyfrowane połączenie, należy użyć następującego kodu:

```
if ($_SERVER['HTTPS'] !== 'on') {
  die("Must be a secure connection.");
}
```

Częstym błędem jest wysyłanie formularza przez bezpieczne połączenie (np. `https://www.example.com/form.html`), ale wysyłanie akcji formularza do adresu `URL` `http://`. Wszelkie parametry formularza wprowadzone następnie przez użytkownika są wysyłane przez niezabezpieczone połączenie, a trywialny sniffer pakietów może je ujawnić.