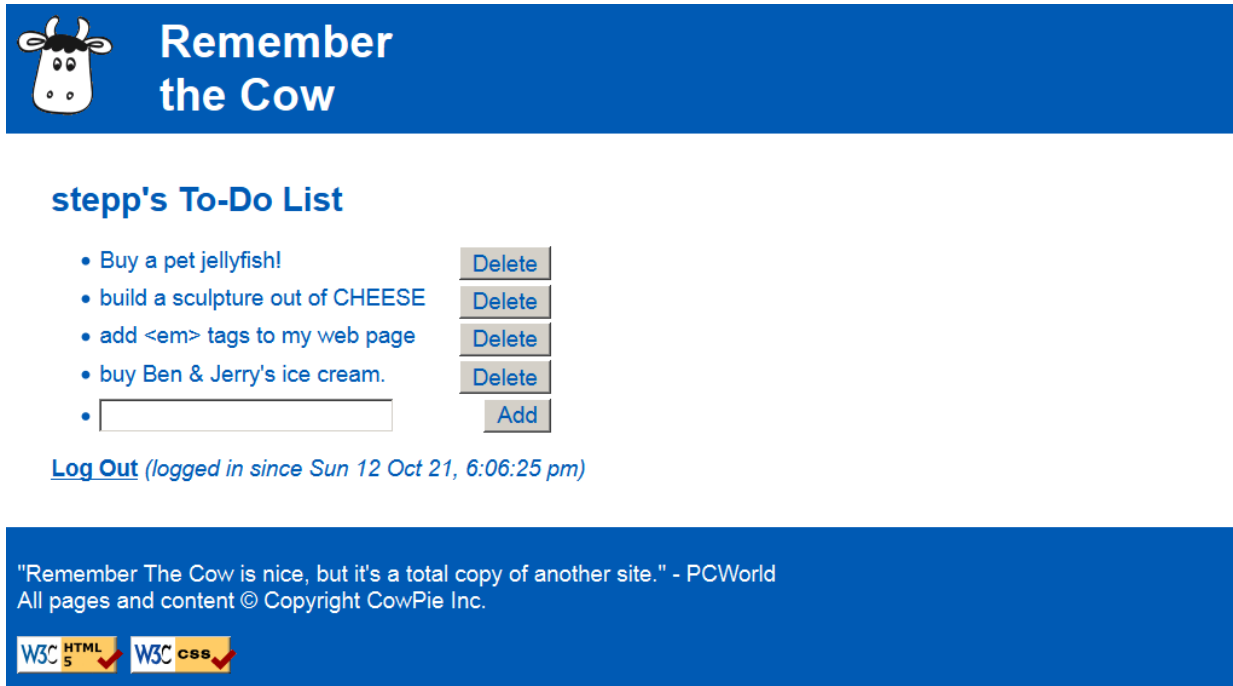


University of Washington, CSE 154 Homework Assignment 7: To-Do List

In this assignment you will write a web application for an online to-do list. The assignment tests your understanding of user login sessions and cookies, and building a large site with many connected pages.

You will create a web page for a fictional online to-do list site called "Remember the Cow", which is a parody of the real to-do list web site "Remember the Milk". Your site requires the user to log in or create an account first. After logging in, the user can manipulate his/her to-do list by adding or deleting items. Any changes made to the list are saved to the web server, so that if the user leaves the page and returns later, the state of the list is remembered.



Remember the Cow

stepp's To-Do List

- Buy a pet jellyfish!
- build a sculpture out of CHEESE
- add `` tags to my web page
- buy Ben & Jerry's ice cream.
-

[Log Out](#) (logged in since Sun 12 Oct 21, 6:06:25 pm)

"Remember The Cow is nice, but it's a total copy of another site." - PCWorld
All pages and content © Copyright CowPie Inc.

W3C HTML 5 W3C CSS

You will write and turn in the following files (*see the following pages for more details about each file*):

- **start.php**: the initial web page describing the site with a form for the user to log in or register
- **login.php**: the target where **start.php** submits its login form data to log the user in
- **logout.php**: logs the user out and returns them to the front page **start.php**
- **todolist.php**: the page that shows the user's to-do list and lets them add/delete items from it
- **submit.php**: the target where **todolist.php** submits requests to add/delete items from the list
- **common.php**: a PHP file containing any shared PHP code or functions used by multiple other files
- **cow.css**: the style sheet that describes the appearance and presentation of all your pages (*optional*)

On the course web site we have provided a **skeleton** of the HTML output from **start.php** and **todolist.php**, to help you get started. The HTML should not necessarily stay in those files exactly as written (e.g., some of it might go into your "common" file), but the skeleton is an indication of what the output should be. You don't have to produce exactly the same HTML output as in these skeletons, but they can serve as a starting point.

Extra Features:

Additional features that can be done for an extra late day can be found in a separate spec on the class web site.

Pages' Appearance:

The site consists of two pages that actually output HTML content for the user to see: [start.php](#) and [todolist.php](#). Both pages share a common appearance and theme. Your pages' appearance must match the following specification; but several aspects of the page appearance are intentionally not mentioned in this spec and are left up to you. The intent is for you to **be creative and make your own unique page appearance**, so long as you satisfy the requirements in this spec. If you want to exactly match our expected output, that's fine, but we'd rather see you make something unique and creative. *(We require you to match our top/bottom blue bars and some page behavior, but the exact appearance of the content in the main section between the blue bars is basically up to you. Through the rest of this spec, we will mention aspects of our own page's appearance in italic for reference, but you are not required to match those italic aspects.)*

Here is the [start.php](#) page's initial appearance:

The screenshot shows the 'Remember the Cow' website. At the top is a blue header bar with a cow logo on the left and the text 'Remember the Cow' in white. Below the header is a central area with a white background. It contains the text: 'The best way to manage your tasks. Never forget the cow (or anything else) again!' followed by 'Log in now to manage your to-do list. If you do not have an account, one will be created for you.' Below this is a login form with two input fields labeled 'User Name' and 'Password', and a 'Log in' button. At the bottom is a blue footer bar with a quote: '"Remember The Cow is nice, but it's a total copy of another site." - PCWorld' and 'All pages and content © Copyright CowPie Inc.' Below the quote are two W3C logos: 'W3C HTML 5' and 'W3C CSS'. Annotations with arrows point to the blue bars and the central area.

The **overall page body** has no margin or padding, so that the page content is able to stretch to the very edges of the browser window. Text in the page body uses font Arial, falling back to Helvetica, then to the default sans-serif font available on the system. The default body font size is 14pt and all other text (headings, etc.) are relative to this. All form controls (text boxes, buttons, etc.) on both pages should also use these fonts and sizes.

A **blue bar** appears along the top and bottom of each page; the bars use a blue background color of #005AB4 and white text. Both bars have 0.5em of space between the edge of the content and the edge of the blue background.

The **top bar** contains the site's "cow" logo image, and 1em to the right of the image, the site's name, "Remember the Cow", as a level-1 heading split across two lines. The heading has no vertical margin. The logo image is found at:

- <https://webster.cs.washington.edu/images/todolist/logo.gif>

The **bottom blue bar** on each page displays a quote about the web site and a copyright notice, along with the standard two W3C images. The URLs of these images and their link targets are the same as in the past. Note that the W3C image links sometimes don't work well on pages that require GET or POST query parameters, so you may need to View Source on your page and copy/paste the output into the W3C validator to validate your page.

Image:

<https://webster.cs.washington.edu/w3c-html.png>
<https://webster.cs.washington.edu/w3c-css.png>

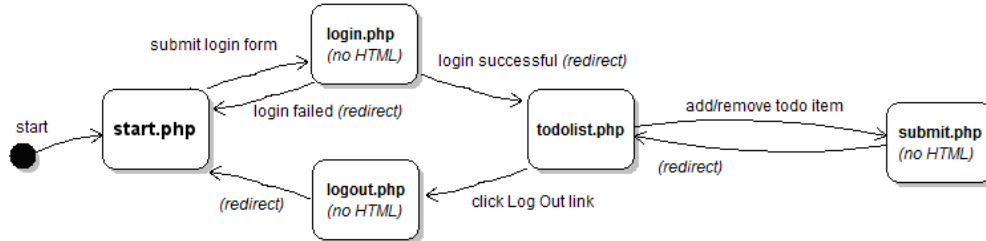
Links to:

<https://webster.cs.washington.edu/validate-html.php>
<https://webster.cs.washington.edu/validate-css.php>

Between the top/bottom blue bars is a **main section of content** on both pages. This main section's appearance is up to you, as long as it has the functionality required by this spec. Be creative! *(Our version has a white background and blue text in the color of #005AB4. The form controls also display their text in this color. Our main area has 2em of padding on all four sides around the edge of its content.)*

Page Relationships and Behavior:

Various actions cause the user to transition from one page to another, as summarized by the following diagram and described in more detail later. Certain pages should only be viewable under certain states. If the user tries to visit [start.php](#) or [login.php](#) while already logged in, they should be immediately redirected to [todolist.php](#). Similarly, the user tries to visit [todolist.php](#), [submit.php](#), or [logout.php](#) while not yet logged in, they should be redirected to [start.php](#). You may assume that the user will not directly try to visit [common.php](#) or any of the [*.txt](#) files.



start.php: This is the initial page that the user visits upon entering the site. The page displays a login form with text boxes for a user name and password (*ours are each 8 characters in size*). The user types a name and password and presses a submit button to log in, which causes the form to submit to [login.php](#) as a POST request.

<input type="text"/>	User Name
<input type="password"/>	Password
<input type="submit" value="Log in"/>	

login.php: This file never directly produces any HTML output. Instead, it accepts the user name and password parameters from [start.php](#) and based on their values, redirects the user to another page. If either of the required parameters are not passed, display an error message of your choice and exit the page (`die`). Recall that you can use the PHP `header` function discussed in class to redirect from one page to another, such as:

```
header("Location: foo.php");
die();
```

Your site will maintain a list of existing user names and passwords, stored in the file [users.txt](#). The file contains one user per line, with the user name and password separated by a colon (:). For example:

```
aobourn:4ever!
daisy:1PrinceSS@
```

When the user submits the login form, you should look through this file to see if the user account exists and whether the password given by the user is correct. If the account exists and the user submitted the correct password, start a **login session** so the site will remember the user's information, and redirect the user to [todolist.php](#). Store session data in PHP's `$_SESSION` global array. (*Hint: See textbook Chapter 14 for help on implementing user login sessions, particularly the case study.*) From the user's perspective it may look as though [start.php](#) submits directly to [todolist.php](#), since the redirect will happen quickly.

If the user name already exists but the password submitted is incorrect, redirect the user back to [start.php](#). Unless you choose to do this as an extra feature, you don't need to display any kind of error message; just put the user back at the start page.

If no such user exists in the file, assume that the user wants to **create a new user account**. To create an account, first validate the user input based on the following (contrived and arbitrary) rules using **regular expressions**:

- The **user name** must be 3-8 characters long, begin with a lowercase letter, and consist entirely of lowercase letters and numbers. Valid examples are "stepp", "jsmith42", "bradpitt", "j3ss1ca", and "x1234567".
- The **password** must be 6-12 characters long, begin with a number, and end with any character that is *not* a letter or number. Valid examples are "4ever!", "123abc\$\$", "2% milk?", and "01234567891*".

If the user input is valid for the new account, **add it to the end of the [users.txt](#) file** in the format above. Also start a login session for the newly created user and redirect them to [todolist.php](#). If the input is invalid, redirect the user back to [start.php](#). Unless you choose to do this as an extra feature, you don't need to display any kind of error message. (No [users.txt](#) file is provided on the class web site, but you can create your own in a text editor or through your PHP code.)

todolist.php: This is the page the user sees after logging in that displays their to-do list and allows them to add/delete from it. If the user visits [todolist.php](#) without being logged in, they are redirected to [start.php](#).

The page should have a heading including the user's name (*ours says "username's To-Do List"*), followed by a bullet list of to-do items. Each item displays its text and a button to Delete that item from the list. After any existing items, there is a text box (*ours is 25 characters wide*) and a button for the user to Add a new item to the end of the list.



The **Add** button adds a new item to the end of the list using the text currently in the box. The **Delete** button next to a given item erases that item from the list. (*More details about adding and deleting are below.*)

Each user account maintains a separate to-do list of their own personal items. The list is stored in a file based on the user name; for example, user **obourn** stores her to-do list in a file **todo_obourn.txt**. The to-do list files contain each item on its own line without any special formatting or encoding. Your file must follow this format exactly. For example, the list of items in the screenshot on the first page of this document would be stored as the following text:

```
Buy a pet jellyfish!  
build a sculpture out of CHEESE  
add <em> tags to my web page  
buy Ben & Jerry's ice cream.
```

The to-do list for any user who has not previously visited the site is initially empty. (The to-do list is empty when there is no **todo_username.txt** file on the server, or when its contents are blank.) If the user has visited the site previously and added items to his/her to-do list, those items are read from the appropriate text file on the server and shown on the page. To-do items might contain special characters like **<** or **&**; your PHP code should **HTML-encode** them before outputting them, but they should be stored in the text file un-encoded.

Underneath the list of items, the page contains a "Log Out" link. When the user clicks this link, it goes to **logout.php**. As with the login file, **logout.php** is not a file that produces any HTML output. Instead, it ends the user's login session and immediately redirects to **start.php**, which will show the start page and login form. If the user manually directs their browser to **logout.php** while they are not logged in, just redirect them to **start.php**.

submit.php: Each Add or Delete action the user takes on the **todolist.php** page should be implemented as a form POST to the page **submit.php**, which will modify the user's to-do text file appropriately and then immediately redirect back to **todolist.php**. Place a small form into each to-do item that POSTs to **submit.php** to delete the item. Also place a form into the last item containing the text box for the user to type a new item to add to the list.

To add an item, submit a POST to **submit.php** with a query parameter of **action**, set to "add"; and another named **item**, containing the text of the item to add. If no to-do list file exists for this user, create one. If it exists, the new item should be appended to the end on its own line. Any item text is acceptable, including special characters or an empty string. It is allowed for two items in the list to have exactly the same text; do not assume they are unique.

To delete an item, submit a POST to **submit.php** with a query parameter of **action**, set to "delete"; and another named **index**, set to the 0-based index of the item to delete. For example, if the file contains four to-do items, they should be considered to have the indexes 0-3 from top to bottom. If the user passes an index of 2, they are indicating that they want the third out of four items to be deleted.

If the index passed is not a number or out of bounds, or if any request to **submit.php** is missing a required parameter, your code should emit a short error message and exit (**die**). Errors are not expected in general usage but could occur if a malicious user manually modifies the page in their browser.

Your code can tell whether the user is submitting an Add or Delete based on the value of the **action** parameter. The **submit.php** page is used for POSTs only; you do not need to write any code to handle GET requests. The submit code doesn't need to produce any HTML output, but it should redirect back to **todolist.php**, which will.

Cookie for Last Login Date/Time:

Most of the state that exists between pages should be kept using session variables. For example, when the user logs in, you should create a session to remember who is logged in and maintain that session until the user logs out.

Separately from the session, you should also create a **cookie** that remembers the last date/time at which the user logged into the site from this particular computer and browser. Set your cookie to **expire after 7 days**. You can get a string representing **the current date/time** at the moment the user logs in using PHP's **date** function:

```
date("D y M d, g:i:s a") # returns a string such as "Sun 14 Oct 21, 6:49:14 pm"
```

On the [todolist.php](#) page, next to the "Log Out" link, display text indicating how long the user has been logged in during their current session based on the value of this cookie. On the [start.php](#) page, if the cookie exists, display the last date/time the user logged in from this computer.



A screenshot of a login form. It features two input fields: the top one is labeled "User Name" and the bottom one is labeled "Password". Below the password field is a button labeled "Log in".

(last login from this computer was Sun 12 Oct 21, 6:49:14 pm)

start.php displaying last-login cookie

stepp's To-Do List



A screenshot of a to-do list form. It shows a bullet point followed by an empty input field and a button labeled "Add".

[Log Out](#) (logged in since Sun 12 Oct 21, 6:49:14 pm)

todolist.php displaying last-login cookie

It is important to understand the difference between cookies and session variables. Your cookie will not be lost when the user logs out or closes the browser. But the other user information will, because it is part of the session that is terminated when the user logs out. If the user explicitly clears the cookies in their browser, or uses a different computer or browser, or uses an "Incognito Mode" tab, the cookie will be gone. (See PHP's `setcookie` function.)

Hints and Debugging:

While writing into text files you may see an error, *"Permission denied"*. If so, use your SFTP software (e.g. FileZilla) to give "write" permissions on the text file, and "write/execute" permissions on your overall [hw7/](#) folder.

Since some of your pages produce no output and immediately redirect, it can be hard to debug them. Consider temporarily disabling the redirection command and instead printing some output for debugging.

You can view the contents of [todo_*.txt](#) and [users.txt](#) directly by typing their URLs into your browser.

Implementation and Grading:

Your page must pass the W3C HTML/CSS **validators**. Use HTTP GET vs. POST requests properly on your forms. You should separate content (HTML), presentation (CSS), and behavior (PHP). Follow our **style guide**.

You should also follow reasonable style guidelines similar to those of a CSE 14x programming assignment. In particular, minimize global variables, avoid **redundant code**, and use parameters and return values properly. Do not use the PHP `global` keyword. Use sessions properly for storing data related to the current user's login; do not abuse the global session array or store values into it that are not needed across the entire login session. Your PHP code should generate no error or warning messages when run using reasonable sets of parameters.

Perform user input validation using **regular expressions** rather than standard string functions as much as possible.

All of the files you submit should have adequate **commenting**. The top of every file should have a descriptive comment header describing yourself, the assignment, and that file's purpose. PHP files should also have descriptive comment headers on each function and on each complex section of code. Make sure to follow the proper comment syntax for each language; in particular, don't put HTML comments in PHP files that do not produce HTML output.

Format your code similarly to the examples from class. Properly use whitespace and indentation. Use good variable and method names. Avoid lines of PHP code more than 100 characters wide. In your HTML, do not place more than one block element on the same line or begin any block element past the 100th character on a line.

Do not place a solution to this assignment on a public web site. Upload your files to the **Webster** server at:

https://webster.cs.washington.edu/students/your_uwnetid/hw7/