Sessions

Class 36

Cookies

- a cookie is a name-value pair
- created by the server
- labeled with server's domain name
- sent in HTTP headers from server to browser with a page
- stored by the browser
- sent back to server by browser
- two versions:
  - session cookies never "expire"; deleted when browser closes
  - persistent cookies have an explicit expiration date

Characteristics

- similar in concept to hidden input fields
- for convenience only
- easily blocked
- easily modified
- cannot harm browser (but can be used to track private actions)
- cannot harm server (unless stolen and used as part of attack)

Example

http://borax.truman.edu/315/c15/cookie.php

note the "lag" due to the request – server – response cycle

also note that we can navigate away and come back, and the cookie is still there

- cookies can be viewed in the browser
- easiest in Chrome; auto-updated
- menu $\rightarrow$ more tools $\rightarrow$ developer tools $\rightarrow$ application

Stateless

- a fundamental characteristic of the Web is the stateless interaction between browser and server
- HTTP is a stateless protocol
- every HTTP request is independent of every other request

- this is great for browsing and retrieving static collections of documents
- not so great for applications

Maintaining State

- we wish applications to be stateful
- the canonical example: a shopping cart

- as you browse through pages, products, details, you wish the shopping cart to remain uncorrupted
- if you put several items in your cart on your laptop, then come to the site later with your phone, you wish the cart to still contain your earlier items

- the technology that allows statefulness is termed the session
- sessions are (usually) implemented with cookies

Session Management

- a session has three components
  - a unique identifier implemented as a cookie
  - as many variables as are needed to store the state information
  - a duration which indicates how long the session should be idle before being terminated

- the ID is passed back and forth between browser and server at every request and every response, in a HTTP header
- all variables are stored on the server either
  - in one or more files
  - in a database
  - both

Timeout

- session state information is stored on the server
- how long should it be stored?
- because HTTP is stateless, there is no way to know for sure when a user has finished with a session
- if you're lucky, the user will log out; that means they're done
- users rarely do this

- abandoned sessions consume resources and are a security risk
- thus all sessions must time out at some point
- how long depends on the application and the designer (you)

Summary

- a session management system must
  1. create and keep track of unique IDs
  2. maintain state information associated with each ID
  3. (eventually) time each session out and free its resources

PHP Sessions

- (by default) PHP implements sessions with
  - a cookie named PHPSESSID that stores the session ID and that has timeout information
  - text files on the server that store session variable information (default: /var/lib/php/sessions on Linux Apache server) (large systems uses databases instead of text files)

Session Functions

- several functions are used in PHP to manipulate sessions
- session_start is used both to create a session and to attach to an existing session
- it is used in every PHP program that needs to access the session
- it must be called before any HTML is emitted, so it normally is put at the top of the program

- session_name, session_id, and session_destroy are used to kill a session

Session Variables

- once a session exists, you can store any value you wish in `$_SESSION`
- `$_SESSION['name'] = 'Jon Beck';`

- you can do all the normal things with `$_SESSION`
- `unset($_SESSION['name']);`

- `$_SESSION = array();` clears all data from this session, but does not destroy the session

Example

https://borax.truman.edu/c36/home.php

Usefulness

- the huge strength of sessions is that they allow secure sharing of data between different PHP programs
- reduced information passing in GET URLs
- reduced information passing in POST variables

- example: saving search results

```
if (isset($_POST['searchlist']))
{
  $_SESSION['searchlist'] = json_decode($_POST['searchlist']);
}
```

Reasons to Use Sessions

authentication

- a user only has to log in once, then every PHP program knows their user ID
- if `$_SESSION['userid']` is set, then you are guaranteed they logged in correctly and are authorized to be at that page
- furthermore, when they log in, you can save their access level, for example, guest, normal_user, admin

performance

- if an application needs to do an expensive DB lookup or computation, do it once and store the results in a session variable rather than re-computing it in every PHP program

Reasons to Use Sessions

saving form data

- applications must validate information before blindly storing it into a database
- validation must happen on the server: you can't trust anything on or from the browser
- problem: you submit a form, some fields are incorrect, you get sent back to fix them, the original information is gone, you have to re-enter
- saving entered information in session variables allows you to re-populate form fields with no need for the user to re-enter information

Reasons to Use Sessions

history

- track a user's pathway around a site, so you know what pages they've been to and what pages they haven't

customization

- let users choose their own colors, layout preferences, etc
- remember those choices in session variables, available to every page on the site