

CGI – Part 3

Survey Example

Presents table of movies/books, one per row

Users out on the Internet can vote for a row – their IP addr is added to the row

The file is read and written on the server side – in this way, the survey CGI is building a client/server solution

Filenum – Security, Hidden field

Client specifies a 'filenum'

Security issue vs. letting the client specify a filename

Use a hidden field to maintain continuity

Multiple Vote Buttons

Each row in the table has a “vote” button

How to distinguish which got clicked?

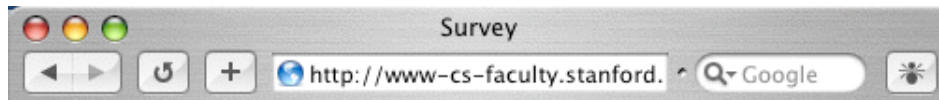
Cannot use value=, since that's the button title, and they all have the title “Vote”

Use the name of the button instead – v_0, v_1, ...

On server side, look through the bindings to figure out which button was clicked.

Use `$query->param()` to get an array of the binding names

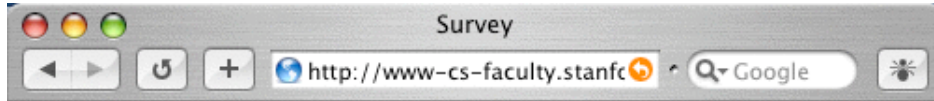
Survey Example



Survey

Choose Survey





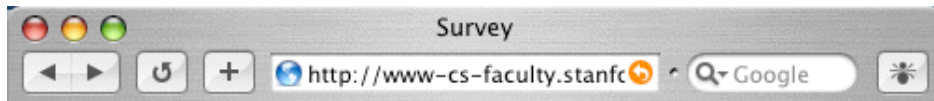
Survey

movie_survey.txt

Aliens	<input type="button" value="Vote"/>
Midnight Run	<input type="button" value="Vote"/>
Annie Hall	<input type="button" value="Vote"/>
The English Patient	<input type="button" value="Vote"/>
As Good As It Gets	<input type="button" value="Vote"/>
Jerry Maguire	<input type="button" value="Vote"/>

Fri May 9 12:27:25 2003

User votes for Aliens...



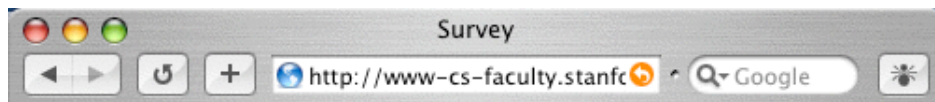
Survey

movie_survey.txt

Aliens	171.64.64.167	<input type="button" value="Vote"/>
Midnight Run		<input type="button" value="Vote"/>
Annie Hall		<input type="button" value="Vote"/>
The English Patient		<input type="button" value="Vote"/>
As Good As It Gets		<input type="button" value="Vote"/>
Jerry Maguire		<input type="button" value="Vote"/>

Fri May 9 12:27:40 2003

Votes for Aliens again...



Survey

Hey, you already voted for 'Aliens'

movie_survey.txt

Aliens	171.64.64.167	<input type="button" value="Vote"/>
Midnight Run		<input type="button" value="Vote"/>
Annie Hall		<input type="button" value="Vote"/>
The English Patient		<input type="button" value="Vote"/>
As Good As It Gets		<input type="button" value="Vote"/>
Jerry Maguire		<input type="button" value="Vote"/>

Fri May 9 12:28:15 2003

Survey Code

```
#!/usr/bin/perl -w
use strict 'vars';

## Survey CGI -- lets people vote for the row they like.
## Supports multiple files -- uses a hidden field to track
## which file the user is on.

## Each line of the text file looks like...
## title <tab> ip1 ip2 ip3
## for each ip that has voted for that row

use CGI;

#### HTML start and end
my $html_start = <<EOT;
<html><head>
<title>Survey</title>
</head>
<body bgcolor=white>
<h1>Survey</h1>
EOT

my $html_end = "</body></html>\n";
```

```

##### Start Output
## $| = 1; ## sets STDOUT to unbuffered
## Need unbuffered if the prints must be sent at once --
## not necessary for CGI output

print "Content-type: text/html\n";
print "\n";
print $html_start;

my $query = new CGI;

my $filenum = $query->param("filenum");
my $filename;

# Encode which file as a number index into our filanem array.
# This is very simple and effective security strategy to limit
# what we pass to open()
my @filenames = ("movie_survey.txt", "book_survey.txt");

my @lines;      ## global var -- the lines of text

if (defined($filenum)) {
    ## note: filename from OUR array
    ## note: what happens if some bad guy passes in filenum == 13?
    $filename = $filenames[$filenum];
}

# Security note: we do not send a string from the client
# side into an open() call. In perl, open() can do all sorts
# of things, such as "rm *", so we must be careful what we pass it.

##### Choose Form
my $chooseForm = <<EOT;
<h2>Choose Survey</h2>
<p>
<form method=get>
<select name=filenum>
<option value=0>Movies
<option value=1>Books
</select>
<input type=submit name=choose value="Choose Survey">
</form>
EOT

##### Response logic
if (!defined($filename)) { ## no file -> choose form
    print $chooseForm;
    print $html_end;
    exit(0);
}
else {
    ReadFile(); ## loads the global @lines array

    ## clear -> erase all votes
    if (defined($query->param("clear"))) {
        my $line;
    }
}

```

```

    foreach $line (@lines) {
        my($text, $votes) = ParseRow($line);
        $line = $text . "\n";
    }
    WriteFile();
}

## check for vote in the bindings -> update that row
my $num = FindVoteSubmit();
if ($num != -1) {
    my ($text, $votes) = ParseRow($lines[$num]);
    my $ip = $query->remote_addr;
    ## (above example of getting HTTP header field from CGI obj)
    ## add this ip as a vote, if it has not already voted
    if (index($votes, $ip) == -1) {
        $votes = $votes . " $ip";
        $lines[$num] = $text . "\t" . "$votes" . "\n";
        WriteFile();
    }
    else {
        print "<p>Hey, you already voted for '$text'\n";
    }
}

my $method = "post";
## The browser convention is that if method is "get", the
## browser will re-send the request if the user goes back
## and hits the reload button. If method is "post", the browser
## will ask the user first before re-submitting

## In any case, now print out the table
## Use hidden field to remember which file we're on
print "<h3>$filename</h3>\n";
print "<form method=$method>\n";
print "<input type=hidden name=filenum value=$filenum>\n";
print "<table border=1 width = 100%>\n";

my $line;
my $i;
for ($i=0; $i<scalar(@lines); $i++) {
    $line = $lines[$i];
    my ($text, $votes) = ParseRow($line);

    ## trick: use v_NNN as button name, where NNN is row index
    my $buttonName = "v_" . $i;

print <<EOT;
<tr><td>$text</td><td>$votes</td>
<td><input type=submit name=$buttonName value=Vote></td></tr>
EOT

}

print "</table>\n";
print "<input type=submit name=clear value=\"Clear All\" >\n";
print "<input type=submit name=show value=\"Get Latest\"></form>\n";

```

```

my $time = localtime();
print "<p><font size=-2>$time</font>\n";
print $html_end;
exit(0);
}

## Returns the number of a v_NNN submission,
## or -1 if none found.
sub FindVoteSubmit {
    my @params = $query->param();
    my $param;
    foreach $param (@params) {
        if ($param =~ /^v_(\d+)/) {
            return $1;
        }
    }
    return(-1);
}

## Given one line of text, parse it into text and vote strings
## and return them as an array length 2
sub ParseRow {
    my($line) = @_;
    chomp($line);
    my ($text, $votes) = split(/\t/, $line);
    if (!defined($votes)) { $votes = ""; }
    return($text, $votes);
}

## Reads $filename into @lines array
sub ReadFile {
    open(DATA, "$filename") || ReportError("Cannot open '$filename'");
    ## Security: do not pass a string from the client to open()

    @lines = <DATA>;      ## read whole file

    close(DATA);
}

## Writes @lines array to $filename
sub WriteFile {
    open(DATA, ">$filename") || ReportError("Cannot open '$filename' for writing");
    my($line);
    foreach $line (@lines) {
        print DATA $line;
    }
    close(DATA);
}

## Helper that prints an error and exits
sub ReportError {
    my($err) = @_;

    print "\n<h1>Error</h1><p>$err\n</html>\n";
    exit(0);
}

```

Many Forms / Rowindex Alternative

Problem: which vote button was clicked

Above: use button name

Alternate: have many little forms, one per row.

Each little form has the filename hidden field, a submit button, and a 2nd hidden field called "rowindex"

In some ways this is cleaner, but it involves a lot of repetition

Security

Strings from the "client side" – not trusted

open() problems

In Perl, open() can run a program, such as with the filename "rm -rf * |"

In any case, open() could be used to return an unintended file on the server

Therefore: be careful what we pass to open—one way is to only pass strings that server programmer has hard-coded

Filenum solution

Simple: the client can only index into our array

Effective, but possibly inconvenient, since array must be kept up to date

Filename field problems

What if we put hidden "filename" field in the form

Is that safe?

No, the client could just edit the form however they like, and then submit it

Filename syntax checking

Could get filename from client side, but check it syntactically (e.g. must be a word, followed by .txt). Non-word chars such as '.', '/', and '|' are worrisome in filenames.

```
if (!$fname =~ m/^[\\w\\-\\_]+\\.txt$/) {
    ReportError("Filename must not contain funny chars, but got '$fname'");
}
```

Get/Post and the Back button

Browser behavior varies with the back button

Survey seq: clear, vote0, vote1 – what happens when we hit the back button?

Mostly: if the GET method is used, the back button does not do a submit, it just shows the old HTML.

However, doing a reload/refresh on a page re-sends the request that lead to that page, essentially doing a form submit again.

If the POST method is used, the browser may ask the user if they would like to do a submit first, giving some sort of warning. But who actually reads/understands that warning?

The back button is fine with static pages, but it can play havoc with a web application. Users are gradually being trained not to use the back button with web apps.

Back Example

Survey app – consider this sequence

page1 -- empty

(vote for row 0)

page2 -- see vote on row0

(vote for row 1)

page3 -- see votes on row0 and row1

(hit back button)

page2 -- see vote on row0 (seeing the old state)

Reload

Now seeing page2, what if client hits "reload"?

This re-does the submit from page1 that lead to page2, in this case casting a vote for row0 again.

Our CGI screens out the duplicate vote and gives a little error message, but you can imagine all sorts of problems if the CGI is not ready for this case.

Problem – Double Order (multiple Submit)

Suppose the user is at the "Submit order" page of a web store

They hit the button – submits the order

Before getting the response (if the website is slow), they hit the button again

Or they use the back button to get back to the page, and then hit the button again

Or they are on the “order accepted” page, and hit the reload button

Problem: order submitted twice

Post Solution

Could use the POST method so that the browser will suggest to the user that they not submit again.

Not a complete solution, but it helps.

Prevents the user from using the back button to just look at the old state – an annoying loss, but maybe that's ok.

Serial Number Solution

When putting out the “submit order page” – include a serial number which is remembered on the server side.

Keep track of which numbers have already been submitted – notice if this is a second submit

Alternately: track recent orders, and compare this order to recent ones to see if it appears to be a duplicate

Back Problem 2 – Add Record

Suppose the user is at the “add record” page for a database, and they use it to add a record

Then they hit the back button, fill in different form values, and click add-record again, intending to add a 2nd record. From the user's point of view, this is a pretty reasonable interpretation of the interface.

Fundamental Problem

The server may get two submits, the 2nd of which is not intentional

The server may get two submits, the 2nd is intentional

Middleground Solution

Impossible to truly reconcile the two cases – decide on a case by case basis

Follow principle of “least damage”

e.g. losing their record data is bad, so take the 2nd submit for the record=add case

e.g. getting a dup order is bad, so screen out that case

Can always put up some sort of error page that explains the situation to the user and asks them what they really wanted. However, users get scared when presented with an unexpected page.

Remember – HTML forms are a bit of a hack – limits on how high quality an user experience we can build

Maybe users can be “trained” over time not to use the back button with web apps? But there will always be new users, new people to explain this to.