

CGI 2

CGI Web App Strategy

Form on client side -- presents a question

On submit, form bindings are sent to server as client request

Server runs CGI which looks at bindings, make response

Movie Search Problem

Present a search form

Client enters a string, we return a table of rows from the movie database that included the search string

Form/action trick

A form with no action= URL, is sent back to the URL that produced the form

Take advantage of this: when invoked with no bindings, produce the search form

e.g. .../cgi-bin/moviesthrough.pl --> search form

Then on form submit, the bindings will be sent back to the same URL

Simplicity -- the client specifies a url to start things off, and that url is used thereafter

In this way, the form and the script do not have to know each-others filename, the name of the server etc., and it's easier to keep the field name's consistent since it's all in the one file.

It would be more structured to maintain the form in its own HTML file that is separate from the perl script file, especially if the two are maintained by separate people.

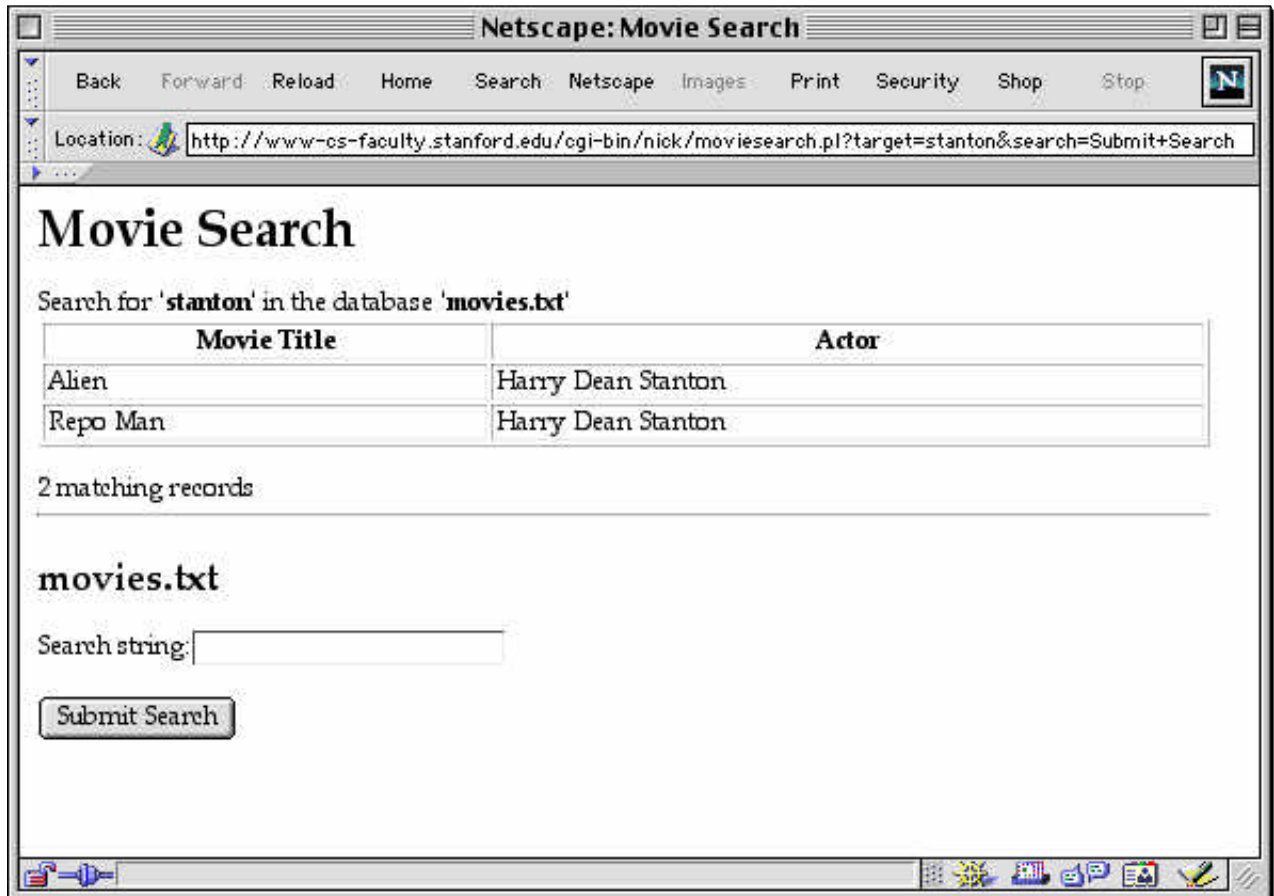
Moviesthrough Points

Uses CGI.pm to extract the value of the "target" binding

If the target string is defined, opens the local file "movies.txt", searches it, and prints the results to a table.

If the target binding is not defined, generates a search form

Movie Search Example



```

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

# Implements a simple search on a local tab-delimited database
# and returns the results in an HTML table.
# When called with no arguments, returns a simple search form.
# A submit of the search form returns a table of all the
# records which contain the target search string.
# Uses the CGI perl module.

use CGI;

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

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

#### Start Output
## $| = 1;          ## sets STDOUT to unbuffered (optional)
print "Content-type: text/html\n\n";
print $html_start;

#### Check client query, put together response
my $query = new CGI;

my($target, $filename);

$target = $query->param("target");
$filename = "movies.txt";

#### Search Form
my $searchForm = <<EOT;
<h2>$filename</h2>
<p>

<form method=get>
Search string:<input type=text name=target size=20><br>

<p>
<input type=submit name=search value="Submit Search">

</form>
EOT

```

```

#### Response logic: look at what we got on the request ...
#### put together our response.
#### If the target is not defined, give them the search form
if (!defined($target)) {
    print $searchForm;
    print $html_end;
    exit(0);
}
else {
    print "<p>Search for '<b>$target</b>' in the database '<b>$filename</b>'\n";

    open(DATA, "$filename") || ReportError("Cannot open '$filename'");
    ## Security: do not pass a string from the client to open()
    ## note that $filename is trusted -- it's our data, not from the client

    ## Make a table out of all the rows which matched...
    print "<table border=1 width = 100%>\n";

    my($labels);
    $labels = <DATA>;          # grab the labels from the first line
    chop($labels);
    TableRow($labels, "th");

    my($count, $line);
    $count = 0;
    while (defined($line = <DATA>)) {
        $target = quotemeta($target); ## '\''escape chars like '*', '('...'
        if ($line =~ /$target/i) {    ## Case insens search for target
            chop($line);
            TableRow($line, "td");
            $count++;
        }
    }

    close(DATA);

    print"</table>\n";

    # print count with the pluralization correct
    if ($count == 1) { print "<p>1 matching record\n"; }
    else { print "<p>$count matching records\n"; }

    print "<hr>$searchForm\n";

    print $html_end;
    exit(0);
}

```

```

# Helper which prints out one row of a table.
# The text should be tab delimited, and the type
# should be "th" for table header, or "td" for
# a table row (the default).
sub TableRow {
    my($text, $type) = @_;

    my(@fields, $elem);

    @fields = split(/\t/, $text, -1);

    print "<tr>\n";
    foreach $elem (@fields) {
        print "<$type>$elem</$type>";
    }
    print "\n</tr>\n";
}

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

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

```

HTTP vs. Continuity

There is one misconception that plagues all web-app developers...

1. GUI Applications

We are all used to the GUI app structure

State in local memory

GUI displays that state

Edit operations change the state -- there's only one copy of the state

Going "back" shows the same state

2. Web Applications

User View

See page 1

The user makes a choice -- Submits a form or click a link

See page 2

The user makes a choice -- Submits a form or click a link

See page 3

Continuity illusion

The user thinks of page1, page2, page3, as all being part of a unified, overall dialog

The user thinks that choices made on page1 are remembered on page3

HTTP -- Stateless

However, HTTP requests are "stateless" (aka "connectionless")

Each connection is made independently as a request/response pair

Stateless protocols are easier to implement, but it means that the continuity illusion will require deliberate effort by the server.

Server View

Request 1-> page 1 -- our user's first request

Request 2 -- requests by other users

Request 3

....

Request100 -> page 2 -- our user's second request

Request 101

Request 102

...

Request 196 -> page 3 -- our user's third request

Server -- No Continuity

When Request100 comes in, there is nothing in HTTP that formally connects it to Request1

The server will need to do extra work to create the illusion that page1, page2, page3, all go together

Client -- False Sense of Continuity

We run and interact with clients all the time, so we get the false illusion that the page1, page2, page3 continuity is just natural.

To be a good server programmer, you have to see through the illusion.

Server -- Breadcrumbs

When returning page1, the server puts extra information in page1 as a reminder about what was going on. (we'll see other solutions later)

Then, when the user does a click on page1 (making request100), the server can recover the information to see what was going on, and so produce page2 with the right context.

Problem -- Planning Ahead

The server needs reminder information at the time of Request100

For that to happen, earlier in time, the server needs to have placed the information in page1

The server needs to plan ahead, so the information is there when the request comes in

Movie Analogies

Monty Python's Holy Grail -- the Trojan Rabbit

Memento

Attach a note to an element at one time, so when that element re-appears later, you can look at the note and remember what was going on.

Technique: hidden fields

Our first technique will be to use hidden fields in the HTML -- use them to store state in a form -- so the state is available later when the form is submitted.

Troll Example

The Troll is a simple CGI that demonstrates continuity...

Page 1: You pick a color

Page 2: You give it something you are afraid of

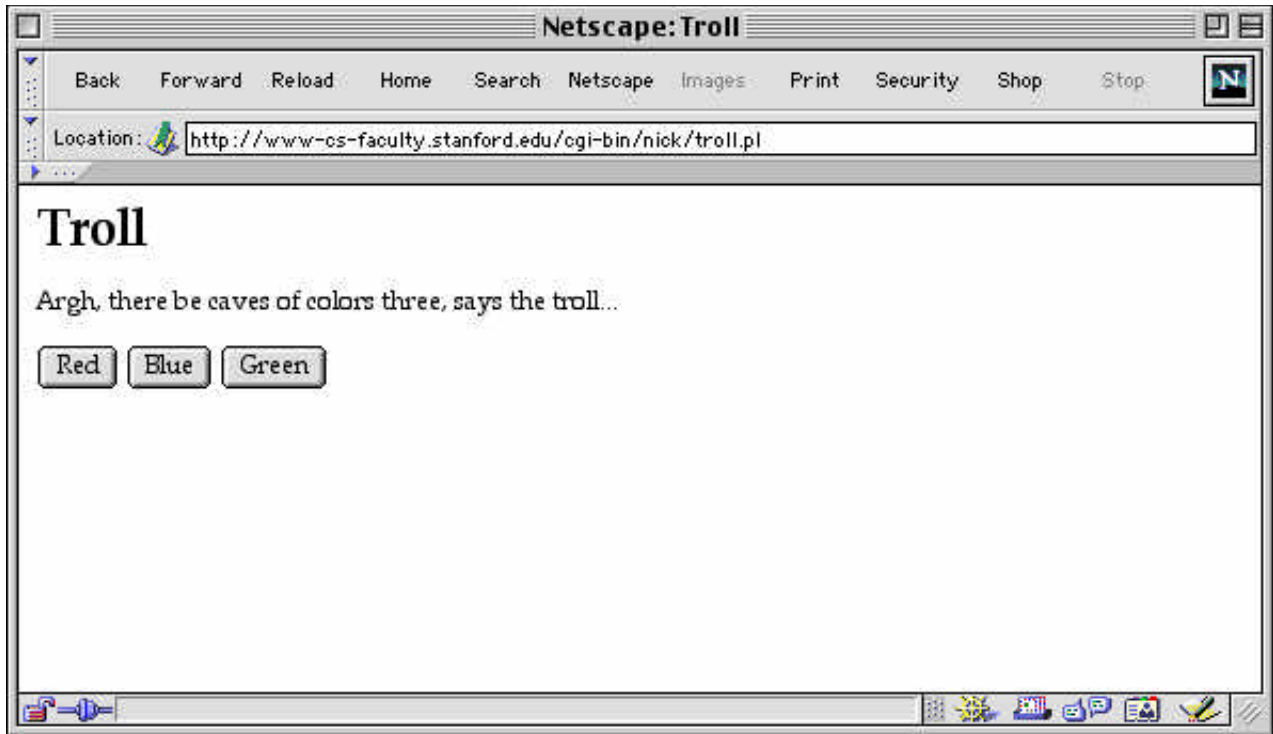
Page 3: The troll makes a scary surprise, using elements from the previous pages

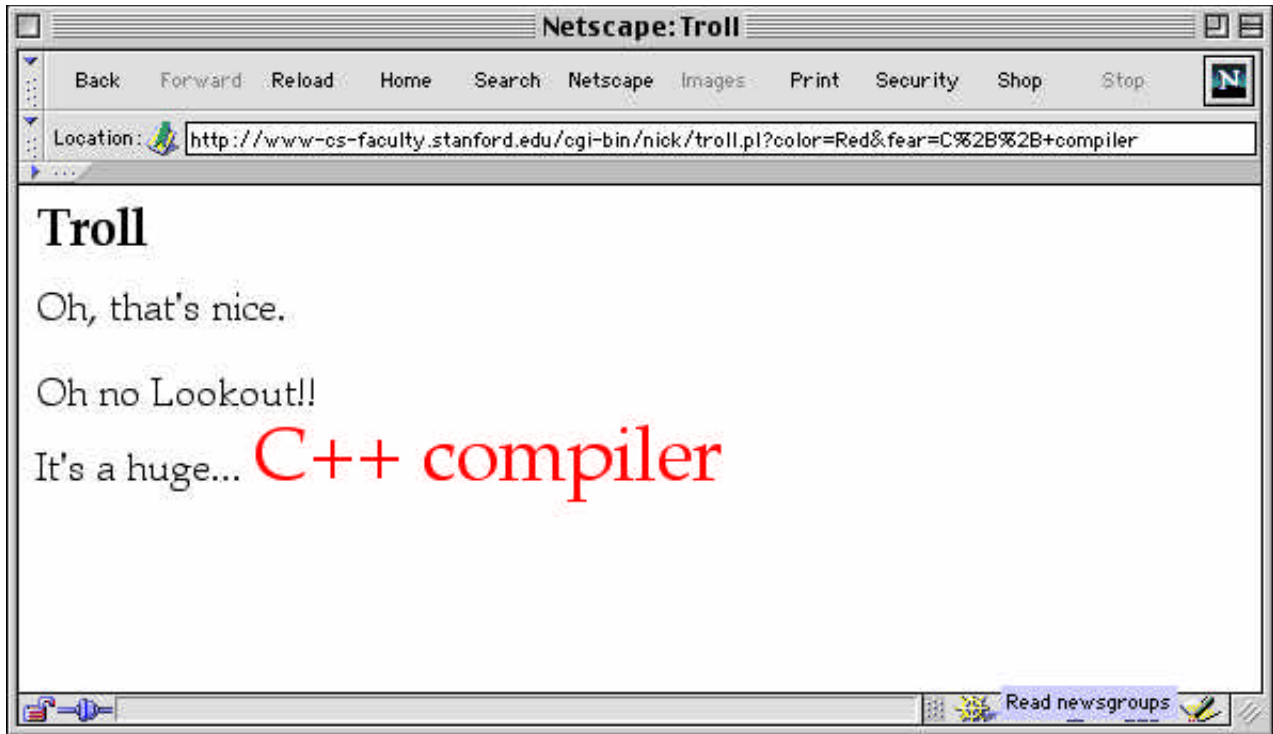
Troll Points

Page1 tells us color, store that as a hidden field in page2

Page2 tells us the fear -- at that point we have both color and fear, so we can generate page3

Point: this looks trivial from the client side, but it required some care to construct the continuity from the server side.





(note: the "C++ compiler" is red)

troll.pl

```

#!/usr/bin/perl -w

use strict 'vars';

use CGI;

my $query = new CGI;

my $color = $query->param("color");
my $fear = $query->param("fear");

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

#### HTML head
print <<EOT;
<html>
<head><title>Troll</title></head>
<body bgcolor=white>
<h1>Troll</h1>
EOT

#### No color -> ask color
if (!defined($color)) {
    print <<EOT;
<form method=get>
<p>
Argh, there be caves of colors three, says the troll...
<p>
<input type=submit name=color value="Red">
<input type=submit name=color value="Blue">
<input type=submit name=color value="Green">
</form>
EOT
} #### No fear -> ask fear
elsif (!defined($fear)) {
    print <<EOT;
<form>
<input type=hidden name=color value="\$color\">
Tell the nice troll what be the
<font size=+2>Scariest Thing</font> on earth...
<input type=text name=fear size=20>
<input type=submit value="Tell Fear To Troll">
</form>
EOT
} #### Have color+fear -> scary page
else {
    $| = 1; ## set STDOUT to be unbuffered, so the data gets sent before the sleep
    print "<p><font size=+2>Oh, that's nice.<p>Oh no Lookout!!\n";
    sleep(2);
    print "<br>It's a huge...\n";
    sleep(2);
    print "<font size=+4 color='\$color\">$fear</font></font>";
}

print "</body></html>\n";

```