Carl Durocher, Fotolia

Question and answer system for the web

QUIZ MASTER

Catalyst is the Ruby on Rails of the Perl universe. When you are developing a web application like a quiz, using the MVC framework is really convenient and helps keep the underlying components cleanly separated.

BY MICHAEL SCHILLI

Whether *guessthelogo.com* asks you to pick the right company logo out of strikingly similar variations, or Food.aol.com invites visitors to identify candy bars by their cross sections (Figure 1), an entertaining quiz is always welcome during a hard day at work. You can expect your colleagues to forward the URLs, and comparing scores later on will mix up the hacking order and trigger fascinating discussions.

Are you interested in compiling your own quiz? Figure 2 shows our homemade quiz at work. To make the code reusable, the application retrieves the questions and multiple choice answers from a YAML file (Figure 3). The example uses a selection of questions from the USA immigration test. For example, applicants need to know how many stars are displayed on the US flag and what they symbolize [4].

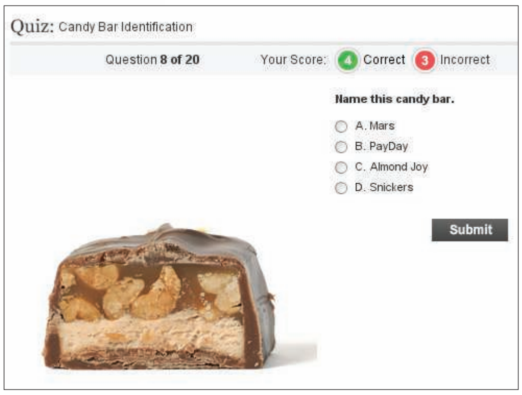
The web application parses the YAML file and displays each question individually on a new page. Whereas the YAML file always lists the correct answer first, the application will display possible choices in random order to keep things interesting.

The implementation is not particularly sophisticated, but there are quite a few things to think about: nicely designed HTML with dynamically managed fields, session management between the individual questions to prevent the application from forgetting the user's score, and a results page that tells the user the final score and invites them to try the next round (Figure 4). Finally, the server should never trust the client, because the client just might cheat.

Catalyst Framework

The Catalyst Framework [5] helps Perl programmers with projects of this kind by automatically creating a program code skeleton to which the developer simply adds the application-specific components.

The fact that the system gets split up into the model (data representation), view (HTML display), and controller (flow control), has proven to be very effective in web application development,



Quiz: Candy Bar Identification

Question 8 of 20 Your Score: 4 Correct 3 Incorrect

Name this candy bar.

- A. Mars
- B. PayDay
- C. Almond Joy
- D. Snickers

Submit

Figure 1: Guess the candy bar? A cross-section quiz on food.aol.com [3] invites web users to guess cal-orie-packed goodies.

as it supports clear code separation and thus easier maintenance.

Installing the Framework

The Catalyst modules are available from CPAN. Because of their sheer number, I recommend downloading a prebuilt package. On a Debian-based system, the command line

```
sudo apt-get install libcatalyst-perl libcatalyst-modules-perl
```

installs all the modules and a bunch of dependencies. To avoid having to start from scratch, call *catalyst.pl QuizShow* from the command line and Catalyst creates a new *QuizShow* directory for the newly created application. It drops about 30 files into various subdirectories to let you run the whole enchilada straight away. Among other things, this includes a *Makefile.PL* file, to package the application CPAN-style, predefined configuration files, module skeletons to fill in application-specific code, and various scripts to create new parts and launch the application in different ways.

Later, you can run it as a CGI script or using Mod_perl on an Apache server. During development, you might like to launch the web server included with the distribution:

```
cd QuizShow
script/quizshow_server.pl
```

This immediately launches the server as shown in Figure 5 and outputs nicely formatted information on the server configuration and the URL at which the browser can reach it.

The default setting is *http://localhost:3000*. If you enter it in a browser, you get to see the Catalyst welcome page. Production systems will later use an Apache server instead.

Http with a Memory

When a browser communicates with a web server, neither of them saves state between individual requests, unless session cookies and server-side session files take care of it explicitly. A quiz that forgets the score between questions wouldn't be all that useful.

Session management is boilerplate logic, and it's easy to get wrong, so Catalyst offers a turn-key solution, again as a Debian package.

The following command line installs the required Perl modules:

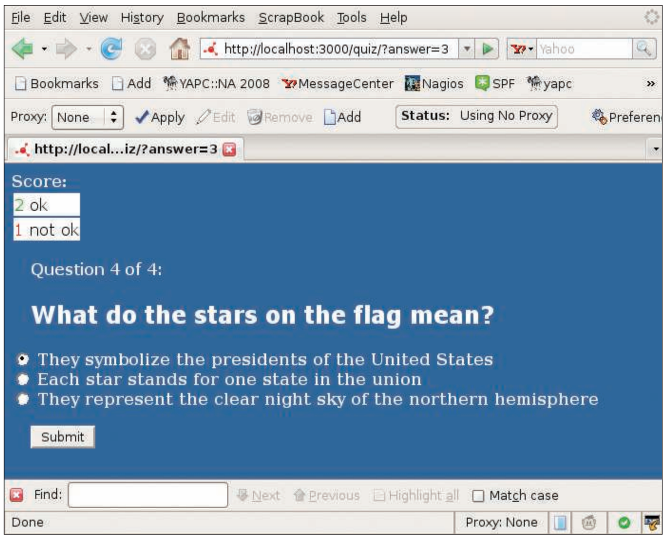
```
sudo apt-get install libcatalyst-plugin-session-fastmmap-perl
```

To make sure the quiz automatically feeds a session cookie to the browser on first contact, besides allocating a cache server-side and storing user data in that space, you need to change the code

```
use Catalyst qw/-Debug ConfigLoader Static::Simple/;
```

Listing 1: Ctrl-Quiz.pm

```
######################################
package QuizShow::Controller::Quiz;
# Mike Schilli, 2008 (m@perlmeister.com)
######################################
use strict;
use warnings;
use base 'Catalyst::Controller';

######################################
sub quiz : Global {
######################################
  my ( $self, $c, @args ) = @_;

  if((@args and $args[0] eq "reset") or
   !defined $c->session->{next_question} or
   $c->session->{"next_question"} == -1
    ) {
    $c->session->{"next_question"} = 0;
    $c->session->{"score_ok"}      = 0;
    $c->session->{"score_nok"}     = 0;
    $c->session->{"total"}         =
           $c->model('Questions')->total();
    $c->response->redirect($c->uri_for());
    $c->detach();
  }

  if(my $answer =
          $c->req->param("answer")) {

    if($answer ==
       $c->session()->{"correct_answer"}) {

        $c->session()->{"score_ok"}++;
    } else {

        $c->session()->{"score_nok"}++;
    }
  }

  my $next_question =
    $c->session()->{"next_question"} || 0;

  $c->stash->{template} = 'quiz.tt';

  my ($question, @answers) =
    $c->model('Questions')->
            get_question( $next_question );

  if(defined $question) {
    @answers = map { [$_, 'incorrect'] }
                              @answers;
    $answers[0]->[1] = 'correct';

    my $correct_answer;
    my $i = 0;

    while (@answers) {
      my $pick = splice(@answers,
                       rand @answers, 1);
      push @{ $c->stash->{answers} },
           { text => $pick->[0],
             num => ++$i};

      $c->session()->{"correct_answer"}= $i
          if $pick->[1] eq 'correct';
    }
    $c->session()->{"next_question"} =
                       $next_question + 1;
  } else {
      $c->session->{next_question} = -1;
  }

  $c->stash->{question} = $question;

  for(qw( total score_ok score_nok
          next_question)) {
    $c->stash->{ $_ } =
        $c->session()->{ $_ };
  }
}

1;
```


Score:
2 ok
1 not ok

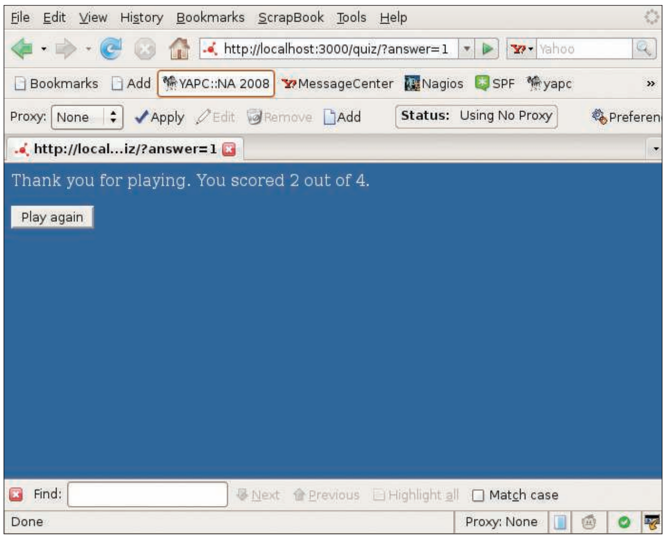
Question 4 of 4:

What do the stars on the flag mean?

- They symbolize the presidents of the United States
- Each star stands for one state in the union
- They represent the clear night sky of the northern hemisphere

Submit

Figure 2: A quiz application for new US citizens implemented with Catalyst. This question asks about the number of stars in the flag.


Thank you for playing. You scored 2 out of 4.

Play again

Figure 4: At the end of the quiz, the candidate is shown their score. The application needs to remember the history.

in the automatically generated *lib/QuizShow.pm* file to:

```
use Catalyst qw/-Debug ConfigLoader ⤦
Static::Simple Session ⤦
Session::State::Cookie ⤦
Session::Store::FastMmap/;
```

This allows the application to access a Perl hash with session information by simply calling the Catalyst context object's *session()* method. Catalyst stores this data automatically under the session ID of the Catalyst browser cookie and manages it on the server without requiring any development effort.

Of course, this approach will only work if the browser talks to the same server for each new request, and not to an arbitrary member of a server farm. Catalyst offers database-based sessions for more complex configurations to handle this.

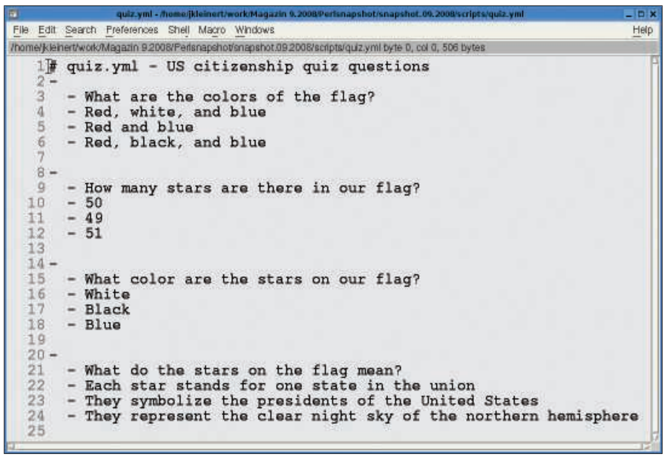

```
# quiz.yml - US citizenship quiz questions
-
  - What are the colors of the flag?
  - Red, white, and blue
  - Red and blue
  - Red, black, and blue

-
  - How many stars are there in our flag?
  - 50
  - 49
  - 51

-
  - What color are the stars on our flag?
  - White
  - Black
  - Blue

-
  - What do the stars on the flag mean?
  - Each star stands for one state in the union
  - They symbolize the presidents of the United States
  - They represent the clear night sky of the northern hemisphere
```

Figure 3: The multiple choice questions with three alternative answers are stored in a YAML file. The first choice is always the correct one, but the order will be mixed up later on.

Worth Viewing

Perl's Template Toolkit [6] helps implementing the "View" part of the Catalyst MVC. It defines a template language, which is deliberately kept simple and allows users to define dynamic fields in static HTML. Although it supports simple programming logic, such as conditions or loops, it deliberately avoids the features of a full-fledged programming language because inexperienced developers tend to add more code in the view layer, instead of relying on a clear separation of the flow controller and the view. The command

```
script/quizshow
create.pl ⤦
view TT TT
```

using the included *quizshow_create.pl* Catalyst helper script adds the *lib/QuizShow/View/TT.pm* module to the directory tree created previously. The first *TT* represents the name of the module created here (*TT.pm*); the second one ensures that the latter is a class derived from the Template Toolkit View. Alternatively, Catalyst supports the Mason and *HTML::Template* toolkits.

Catalyst also tells the *TT.pm* module to se the Template Toolkit processor to handle files ending with *.tt* before the web server delivers them. Figure 6 shows the *quiz.tt* template, which must reside in the root directory of the newly created Catalyst project.

First, the *[% IF %]* condition written in Template Toolkit syntax checks to see whether more questions exist. If not, the browser displays the final score, otherwise it uses the Template variables *score_ok* and *score_nok* to display the current score and the number of questions left.

The template then outputs the current question and uses the *FOREACH* loop to iterate through the randomly ordered answers and present them as clickable radio buttons.

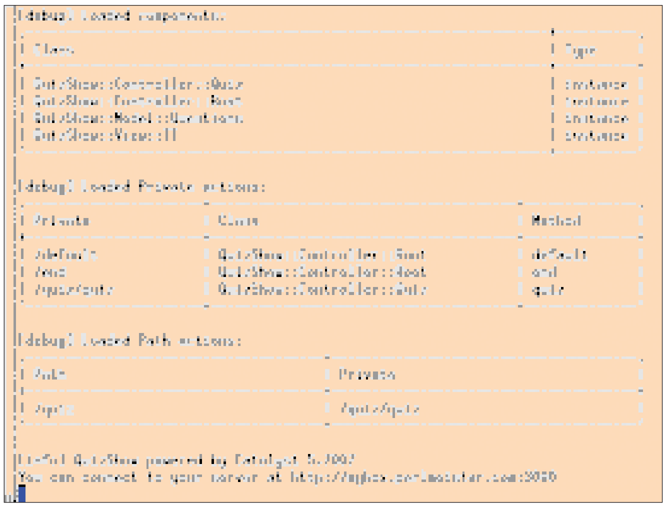
The *Submit* button sends the web form to the web server at the original URL because the HTML doesn't specify a form URL.

Control the Flow

To define the application's control flow, you also need a *Quiz.pm* Controller:

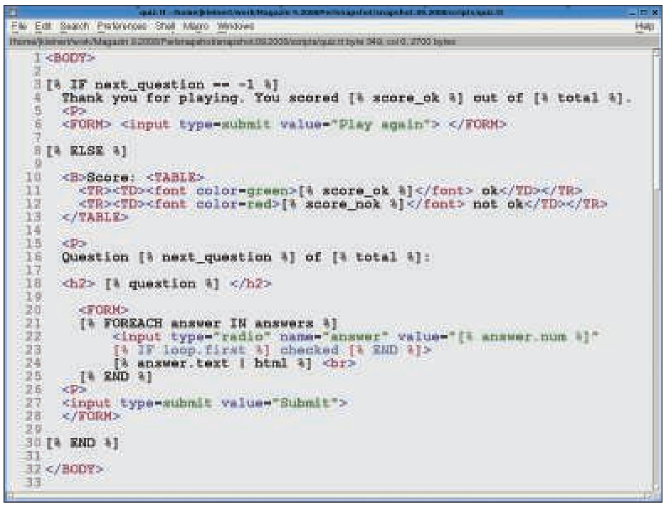
```
script/quizshow_create.pl ⤦
controller Quiz
```

This command line creates the *lib/QuizShow/Controller/Quiz.pm* file, to which you need to add the code shown in Listing 1, *Ctrl-Quiz.pm*.



```
[debug] Loaded components:
.----------------------------------------------------------------+----------.
| Class                                                          | Type     |
+----------------------------------------------------------------+----------+
| QuizShow::Controller::Quiz                                     | instance |
| QuizShow::Controller::Root                                     | instance |
| QuizShow::Model::Questions                                     | instance |
| QuizShow::View::TT                                             | instance |
'----------------------------------------------------------------+----------'

[debug] Loaded Private actions:
.----------------------+--------------------------------------+--------------.
| Private              | Class                                | Method       |
+----------------------+--------------------------------------+--------------+
| /default             | QuizShow::Controller::Root           | default      |
| /end                 | QuizShow::Controller::Root           | end          |
| /quiz/quiz           | QuizShow::Controller::Quiz           | quiz         |
'----------------------+--------------------------------------+--------------'

[debug] Loaded Path actions:
.-------------------------------------+--------------------------------------.
| Path                                | Private                              |
+-------------------------------------+--------------------------------------+
| /quiz                               | /quiz/quiz                           |
'-------------------------------------+--------------------------------------'

[info] QuizShow powered by Catalyst 5.7007
You can connect to your server at http://mybox.perlmeister.com:3000
```

Figure 5: The test server included with the Catalyst package launches and discloses runtime parameters of the application.



```
<BODY>

[% IF next_question == -1 %]
  Thank you for playing. You scored [% score_ok %] out of [% total %].
  <P>
  <FORM> <input type=submit value="Play again"> </FORM>

[% ELSE %]

  <B>Score: <TABLE>
    <TR><TD><font color=green>[% score_ok %]</font> ok</TD></TR>
    <TR><TD><font color=red>[% score_nok %]</font> not ok</TD></TR>
  </TABLE>

  <P>
  Question [% next_question %] of [% total %]:

  <h2> [% question %] </h2>

    <FORM>
    [% FOREACH answer IN answers %]
        <input type="radio" name="answer" value="[% answer.num %]"
        [% IF loop.first %] checked [% END %]>
        [% answer.text | html %] <br>
    [% END %]
  <P>
  <input type=submit value="Submit">
  </FORM>

[% END %]

</BODY>
```

Figure 6: The quiz.tt Catalyst template defines the application's look. The Toolkit processor is responsible for the interpretation.

Quiz.pm defines the *quiz()* method, which has the *:Global* attribute. This allows Catalyst to catch any requests below the *http://localhost:3000/quiz* URL and pass any subsequent paths in to the application as the *@args* parameter. For example, if the user adds */quiz/reset* to the URL, Catalyst calls the *quiz()* method and sets the first element of *@args* to *reset*.

On */reset*, the *quiz()* resets the session data to zero and asks the browser to redirect to the application start page. This changes the URL displayed in the browser from */quiz/reset* to */quiz*; the controller sets the counter for right and wrong answers to *0*, and a new quiz can begin.

The Catalyst object's *uri_for()* method generates absolute URLs from URLs entered relative to the application root, allowing the browser to redirect to them. The *redirect()* method itself only sets an http header, but does not interrupt the control flow; this is what makes the *$c->detach()* that follows in Line 24 so important.

Break the Flow

This causes Catalyst to finish processing the request. Incidentally, this is a practical method of stopping the control flow, even if you are in the middle of a nested loop construction. The *$c* variable points to the Catalyst system's context object; it is added to the controller's methods with the call and is useful for retrieving almost anything from the depths of the Catalyst system.

The Catalyst object's *$c*'s *session()* method reveals the session hash, which is made persistent by cookies and server-side storage, including the entries *next_question* (index of the next question to be asked in the YAML array), *score_ok* (the number of questions answered correctly), *score_nok* (the number of questions answered incorrectly), *total* (total number of questions), and *correct_answer* to let the server know which of the randomly ordered answers is the right one for the question just posed.

Catalyst uses

```
$c->req->param("answer")
```

in line 28 to retrieve the *answer* form parameter provided by the browser from the request object. This number is equivalent to the number *1*, *2*, or *3* for the radio button clicked by the user to select an answer. If the value matches the value stored in the session hash on the server before the web page was served up, the response was correct, and the controller increments the *score_ok* session variable.

In line 43 the controller defines *quiz.tt* as the template and then stores variable values in the template stash. If the controller sets *$c->stash->{score _ok}*, the template processor will replace the template entry *[% score_ok %]* with the value determined in the controller.

Stash variables can be arbitrarily nested data structures; for example, the stash entry *answers* contains a pointer to an array whose elements in

Listing 2: Mod-Questions.pm

```
01 ############################################
02 package QuizShow::Model::Questions;
03 # Mike Schilli, 2008 (m@perlmeister.com)
04 ############################################
05 use strict;
06 use warnings;
07 use base 'Catalyst::Model';
08 use YAML qw(LoadFile);
09
10 my $FILE = "/home/mschilli/data/quiz.yml";
11
12 ############################################
13 sub total {
14 ############################################
15     my $yml = LoadFile $FILE;
16     return scalar @$yml;
17 }
18
19 ############################################
20 sub get_question {
21 ############################################
22     my($m, $index) = @_;
23
24     my $yml = LoadFile $FILE;
25     return undef if $index > $#$yml;
26     return @{ $yml->[$index] };
27 }
28
29 1;
```

turn are pointers to hashes, which contain the text and number of an answer indexed by the *text* and *num* keys.

The *quiz.tt* template iterates over this array to display all possible answers. It assigns an alias named *answer* to the currently processed element and then uses *[% answer.text %]* and *[% answer.num %]* to access the underlying hash entries – that's a very practical template toolkit feature that saves a lot typing.

Lines 50 to 52 create a data structure from the answer array extracted from the YAML file. The array assigns a *correct* tag to the first entry, and *incorrect* to all others. To display the answers in random order, the *while* loop in line 57 picks up a random element from this array of arrays. If this is the answer tagged as correct, the controller remembers the number for the session hash later on.

Lines 60 to 62 generate a hash with the *text* and *num* entries from the answer and pushes it to the end of the *answers* stash. The *quiz.tt* template picks up the data and dynamically creates the HTML output.

YAML as a Model

Catalyst normally relies on database-backed data models; however, in this case the data is stored in a YAML file. It isn't too difficult to define your own model. The line

```
script/quizshow_create.pl ↩
model Questions
```

creates the *lib/QuizShow/Model/Questions.pm* file, which you need to populate with the code shown in Listing 2, *Mod-Questions.pm*.
The YAML file in Figure 3 defines an array of entries for each question posed by the test. Comment lines starting with *#* are ignored. The individual array entries start with a hyphen, and comprise sub-arrays with four elements each: the wording of the question, followed by the right answer, and two incorrect answers.

The *$FILE* variable in Listing 2 defines the path to the YAML file. The *total()* method parses the data and returns the total number of questions to allow the web application to display the number of questions remaining. *total()* provides the YAML array in a scalar context to this end; this returns the array length in Perl.

The *get_question()* method in line 20 retrieves the questions and answers for an array index (*0* to *n-1*) and returns them as a list. If the index does not point to a valid entry, it returns *undef*. This is the signal to the online quiz that it has run out of questions and has to display the final score.

If the controller wants to access the encapsulated data model, it grabs the Catalyst object and calls *$c->model('Questions')*. This gives it an instance of the *Questions* data model whose *get_question()* and *total()* methods it can then use.

Installation

To install the ready-to-run Catalyst application on the production server, simply enter the typical three commands for CPAN modules in the project directory:

```
cd QuizShow
perl Makefile.PL
make install
```

Catalyst then injects the modules, templates, and scripts required by the application into the Perl hierarchy defined on the used platform.

Instead of the Perl server included by the distribution, I would recommend using a Mod_perl installation to allow Apache 2 to execute the program more quickly:

```
PerlModule QuizShow
<Location />
  SetHandler modperl
  PerlResponseHandler QuizShow
</Location>
```

An Apache 1.3 configuration also is described by the exhaustive Catalyst documentation. If speed is not an issue, there is also a CGI script that Catalyst installs as *quizshow_cgi.pl*. If you drop the script into the web ser-ver's CGI directory and enter the URL *http://localhost/cgi/quizshow_cgi.pl/quiz*, the quiz launches.

If you migrate from the test server to the web server, note that the sessions reside below the */tmp/quizshow* directory and the web server will normally run under a different user account. If you modify the user privileges correspondingly, the new server can just go on using the old session store.

Apache 2 and Mod_perl 2 cause a similar problem. The Perl module is easily installed on Ubuntu using the *libapache2-mod-perl2* and *libcatalyst-engine-apache-perl* packages, however, *Session::Store::FastMmap* complains that it can't handle threads. The alternative *Session::Store::File* works perfectly; just modify *lib/QuizShow.pm* accordingly.

Unfortunately, Apache 2 uses the root account to create the directories and is unable to access them later when it spawns non-privileged child processes. *sudo chown -R www-data /tmp/quizshow* solves the problem by assigning the ownership of the session store to the web server user (can also be *nobody*).

Further Reading

Catalyst has much more to offer than just the functions I discussed. It is equally suited to small or large projects, in which team members split up the work to focus on different areas of the project. It offers a mature test framework that is automatically generated when you create a new project. Dynamic Ajax web pages are also supported. With this approach, the web application uses an extension module to send, for example, Json data to browser-side Javascript – and a cozy Web 2.0 feeling ensues because page reloads are no longer necessary to refresh the display.

Besides the online manual and tutorials [4], the book *Catalyst* [6] offers a very useful overview, although it is not a perfect reference manual as it lacks both an index and the in-depth detail a reference book requires. ■

INFO

[1] Listings for this article: *http://www.linux-magazine.com/resources/article_code*

[2] Candy bar identification: *http://food.aol.com/play-with-your-food/candy-bar-id-quiz/?u*

[3] US citizenship test: *http://www.washingtonpost.com/wp-srv/national/longterm/citizen/citizen.htm*

[4] Catalyst: *http://www.catalystframework.org*

[5] Template toolkit: *http://template-toolkit.org*

[6] *Catalyst*, by Jonathan Rockaway: *http://www.packtpub.com/catalyst-perl-web-application/book*