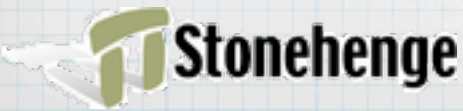
Managing Complexity with CGI::Prototype

brian d foy, brian@stonehenge.com
Stonehenge Consulting Services
Nordic Perl Workshop • June 16, 2006



Web application flow

* Start up
* Receive input
* Process input
* Return output
* Clean up
* Shut down

Looks simple, right?

* Is the request valid?
* Should I process the transaction?
* Do I have all the necessary input?
* Is this part of a multi-step process?
* Which output do I give?

Language agnostic

Current fad is
Model-View-Controller
(MVC)

MVC Architecture

* Model -> data store -> database
* View -> data display -> web page
* Controller -> user interaction -> form widgets

Choose your poison

* Mason
* CGI::Application
* Maypole, Catalyst, Catalyst++
* CGI::Prototype

Using CGI::Prototype

* Model -> DBI, Class::DBI, XML
* View -> Template Toolkit, XSLT -> HTML
* Controller -> HTML form widgets

Application state

* Each state is a namespace
* Each state has a Template file
* Change state at any time by changing namespace

Application flow

* Inherit from CGI::Prototype
* Single URL
* WebApp->activate;
* The rest is in packages and templates
* CGI::Prototype handles the flow

```
sub activate {
  my $self = shift;
  eval {
    $self->app_enter;
    my $this_page = $self->dispatch;
    $this_page->control_enter;
    $this_page->respond_enter;
    my $next_page = $this_page->respond;
    $this_page->respond_leave;
    if ($this_page ne $next_page) {
      $this_page->control_leave;
      $next_page->control_enter;
    }
    $next_page->render_enter;
    $next_page->render;
    $next_page->render_leave;
    $next_page->control_leave;
    $self->app_leave;
  };
  $self->error($@) if $@;
}
```

```
package TPR::Dashboard::FindByName;
use base qw(TPR::Dashboard);

my $users = [];

sub respond {
    my $self = shift;
    my $q = $self->CGI;

    my $name = $q->param('subscriber_name');

    $users = $self->db->get_subscriber_by_name( $name ) || [];
    my $count = @$users;

    if( @$users == 1 ){
        $q->param( 'subscriber_id', $users->[0]->pk );
        "TPR::CGI::ShowSubscriber";
        }
    elsif( @$users == 0 ) {
        "TPR::CGI::StartPage";
        }
    else { __PACKAGE__; }
    }

sub users { $users }
```

Flexibility

* Change action by changing namespace
* CGI::Prototype object handles the input and data (Class::Prototyped) in $self
* Choose template with template()
* Template has access to self.foo

Handling Errors

* Respond with Error namespace
* Rest of transaction does Error stuff
* Still has access to all data
* CGI::Prototype has safe mode (no 500s, hopefully)

Multiple Responses

* Decide which template to use as late as possible
* No search results -> show search form
* One search result -> show details
* Many search results -> show list
* Database error -> show error page

```
[% self.CGI.header %]
<!DOCTYPE ...>
<html>

<head>
    <title>[% self.title %]</title>
    <link rel="stylesheet" type="text/css" href="... "/>
</head>

<body>

<div id="header" class="menubar">
  ...
</div>

<div id="content" class="middle">
<!-- BEGIN Content [% template %] -->
[% PROCESS $template %]
<!-- END Content [% template %] -->
</div>

<div id="footer">
</div>

</body></html>
```

Add accessors...

```
sub action { "add_subscriber" }
sub title  { "Add a subscriber" }

my $db = TPR::Subscribers::Database::SQLite->new( $file  );
sub db { $db }

sub subscriber
    {
    $_[0]->db->get_subscriber_by_pk( $_[1] );
    }
```

...use them in Template

```
<title>[% self.title %]</title>

[% self.CGI.start_form(
    Method => "GET",
    Action => self.CGI.url,
    )
    %]

[% self.CGI.hidden(
    Name     => 'action',
    Value    => self.action,
    Override => 1,
    ) -%]
```

The View just views

* Accessors are just accessors
* Do not change the state
* Change the state in the Controller
* All views then get the feature

Change behavior

```
__PACKAGE__->reflect->addSlots(
    [qw(engine FIELD autoload)] => sub {
        my $self = shift;
        require Template;

        Template->new( {
            PROCESS      => [ $self->config_wrapper ],
            INCLUDE_PATH => "/web/cgi-bin/local-templates",
            } );
        }
    );

sub config_wrapper { 'Wrapper.tt' }

sub template {
    (my $package = ref $_[0] || $_[0]) =~ s/.*::/;
    my $file = "$package.tt";

    -e "/web/cgi-bin/local-templates/$file" ?
        $file : \ "I could not find $package.tt";
    }
```

CGI::Prototype::Hidden

is even easier

References

* Introduction to CGI::Prototype
  http://www.ourmedia.org/node/1644
* Introduction to Class::Prototyped
  http://www.ourmedia.org/node/1728
* Various articles on stonehenge.com