## NAME

Catalyst::Manual::Tutorial::06_Authorization - Catalyst Tutorial - Chapter 6: Authorization

## OVERVIEW

This is **Chapter 6 of 10** for the Catalyst tutorial.

Tutorial Overview

1. Introduction
2. Catalyst Basics
3. More Catalyst Basics
4. Basic CRUD
5. Authentication
6. **06_Authorization**
7. Debugging
8. Testing
9. Advanced CRUD
10. Appendices

## DESCRIPTION

This chapter of the tutorial adds role-based authorization to the existing authentication implemented in Chapter 5. It provides simple examples of how to use roles in both TT templates and controller actions. The first half looks at basic authorization concepts. The second half looks at how moving your authorization code to your model can simplify your code and make things easier to maintain.

Source code for the tutorial in included in the */home/catalyst/Final* directory of the Tutorial Virtual machine (one subdirectory per chapter). There are also instructions for downloading the code in Catalyst::Manual::Tutorial::01_Intro.

## BASIC AUTHORIZATION

In this section you learn the basics of how authorization works under Catalyst.

### Update Plugins to Include Support for Authorization

Edit `lib/MyApp.pm` and add `Authorization::Roles` to the list:

```
# Load plugins
use Catalyst qw/
    -Debug
    ConfigLoader
    Static::Simple

    StackTrace

    Authentication
    Authorization::Roles

    Session
    Session::Store::File
    Session::State::Cookie

    StatusMessage
/;
```

Once again, include this additional plugin as a new dependency in the Makefile.PL file like this:

```
requires 'Catalyst::Plugin::Authorization::Roles';
```

## Add Role-Specific Logic to the "Book List" Template

Open `root/src/books/list.tt2` in your editor and add the following lines to the bottom of the file:

```
...
<p>Hello [% c.user.username %], you have the following roles:</p>

<ul>
  [% # Dump list of roles -%]
  [% FOR role = c.user.roles %]<li>[% role %]</li>[% END %]
</ul>

<p>
[% # Add some simple role-specific logic to template %]
[% # Use $c->check_user_roles() to check authz -%]
[% IF c.check_user_roles('user') %]
  [% # Give normal users a link for 'logout' %]
  <a href="[% c.uri_for('/logout') %]">User Logout</a>
[% END %]

[% # Can also use $c->user->check_roles() to check authz -%]
[% IF c.check_user_roles('admin') %]
  [% # Give admin users a link for 'create' %]
  <a href="[% c.uri_for(c.controller.action_for('form_create')) %]">Admin Create</a>
[% END %]
</p>
```

This code displays a different combination of links depending on the roles assigned to the user.

## Limit Books::add to 'admin' Users

`IF` statements in TT templates simply control the output that is sent to the user's browser; it provides no real enforcement (if users know or guess the appropriate URLs, they are still perfectly free to hit any action within your application). We need to enhance the controller logic to wrap restricted actions with role-validation logic.

For example, we might want to restrict the "formless create" action to admin-level users by editing `lib/MyApp/Controller/Books.pm` and updating `url_create` to match the following code:

```
    =head2 url_create

    Create a book with the supplied title and rating,
    with manual authorization

    =cut

    sub url_create :Chained('base') :PathPart('url_create') :Args(3) {
        # In addition to self & context, get the title, rating & author_id args
        # from the URL.  Note that Catalyst automatically puts extra information
        # after the "/<controller_name>/<action_name/" into @_
        my ($self, $c, $title, $rating, $author_id) = @_;

        # Check the user's roles
        if ($c->check_user_roles('admin')) {
            # Call create() on the book model object. Pass the table
            # columns/field values we want to set as hash values
            my $book = $c->model('DB::Book')->create({
                    title   => $title,
                    rating  => $rating
                });

            # Add a record to the join table for this book, mapping to
            # appropriate author
            $book->add_to_book_authors({author_id => $author_id});
            # Note: Above is a shortcut for this:
            # $book->create_related('book_authors', {author_id => $author_id});

            # Assign the Book object to the stash and set template
            $c->stash(book     => $book,
                      template => 'books/create_done.tt2');
        } else {
            # Provide very simple feedback to the user.
            $c->response->body('Unauthorized!');
        }
    }
```

To add authorization, we simply wrap the main code of this method in an `if` statement that calls `check_user_roles`. If the user does not have the appropriate permissions, they receive an "Unauthorized!" message. Note that we intentionally chose to display the message this way to demonstrate that TT templates will not be used if the response body has already been set. In reality you would probably want to use a technique that maintains the visual continuity of your template layout (for example, using Catalyst::Plugin::StatusMessage as shown in the last chapter to redirect to an "unauthorized" page).

**TIP**: If you want to keep your existing `url_create` method, you can create a new copy and comment out the original by making it look like a Pod comment. For example, put something like `=begin` before `sub add : Local {` and `=end` after the closing `}`.

## Try Out Authentication And Authorization

Make sure the development server is running:

```
    $ script/myapp_server.pl -r
```

Now trying going to http://localhost:3000/books/list and you should be taken to the login page (you might have to `Shift+Reload` or `Ctrl+Reload` your browser and/or click the "User Logout" link on the book list page). Try logging in with both `test01` and `test02` (both use a password of `mypass`) and notice how the roles information updates at the bottom of the "Book List" page. Also try the "User Logout" link on the book list page.

Now the "url_create" URL will work if you are already logged in as user `test01`, but receive an authorization failure if you are logged in as `test02`. Try:

```
    http://localhost:3000/books/url_create/test/1/6
```

while logged in as each user. Use one of the "logout" links (or go to http://localhost:3000/logout in your browser directly) when you are done.

## ENABLE MODEL-BASED AUTHORIZATION

Hopefully it's fairly obvious that adding detailed permission checking logic to our controllers and view templates isn't a very clean or scalable way to build role-based permissions into out application. As with many other aspects of MVC web development, the goal is to have your controllers and views be an "thin" as possible, with all of the "fancy business logic" built into your model.

For example, let's add a method to our `Books.pm` Result Class to check if a user is allowed to delete a book. Open `lib/MyApp/Schema/Result/Book.pm` and add the following method (be sure to add it below the "`DO NOT MODIFY ...`" line):

```
    =head2 delete_allowed_by

    Can the specified user delete the current book?

    =cut

    sub delete_allowed_by {
        my ($self, $user) = @_;

        # Only allow delete if user has 'admin' role
        return $user->has_role('admin');
    }
```

Here we call a `has_role` method on our user object, so we should add this method to our Result Class. Open `lib/MyApp/Schema/Result/User.pm` and add the following method below the "`DO NOT MODIFY ...`" line:

```
    =head2 has_role

    Check if a user has the specified role

    =cut

    use Perl6::Junction qw/any/;
    sub has_role {
        my ($self, $role) = @_;

        # Does this user posses the required role?
        return any(map { $_->role } $self->roles) eq $role;
    }
```

Let's also add `Perl6::Junction` to the requirements listed in Makefile.PL:

```
    requires 'Perl6::Junction';
```

**Note:** Feel free to use `grep` in lieu of `Perl6::Junction::any` if you prefer. Also, please don't let the use of the `Perl6::Junction` module above lead you to believe that Catalyst is somehow dependent on Perl 6... we are simply using that module for its easy-to-read `any` function.

Now we need to add some enforcement inside our controller. Open `lib/MyApp/Controller/Books.pm` and update the `delete` method to match the following code:

```
    =head2 delete

    Delete a book

    =cut

    sub delete :Chained('object') :PathPart('delete') :Args(0) {
        my ($self, $c) = @_;

        # Check permissions
        $c->detach('/error_noperms')
            unless $c->stash->{object}->delete_allowed_by($c->user->get_object);

        # Saved the PK id for status_msg below
        my $id = $c->stash->{object}->id;

        # Use the book object saved by 'object' and delete it along
        # with related 'book_authors' entries
        $c->stash->{object}->delete;

        # Redirect the user back to the list page
        $c->response->redirect($c->uri_for($self->action_for('list'),
            {mid => $c->set_status_msg("Deleted book $id")}));
    }
```

Here, we `detach` to an error page if the user is lacking the appropriate permissions. For this to work, we need to make arrangements for the '/error_noperms' action to work. Open `lib/MyApp/Controller/Root.pm` and add this method:

```
    =head2 error_noperms

    Permissions error screen

    =cut

    sub error_noperms :Chained('/') :PathPart('error_noperms') :Args(0) {
        my ($self, $c) = @_;

        $c->stash(template => 'error_noperms.tt2');
    }
```

And also add the template file by putting the following text into `root/src/error_noperms.tt2`:

```
<span class="error">Permission Denied</span>
```

Log in as `test01` and create several new books using the `url_create` feature:

```
http://localhost:3000/books/url_create/Test/1/4
```

Then, while still logged in as `test01`, click the "Delete" link next to one of these books. The book should be removed and you should see the usual green "Book deleted" message. Next, click the "User Logout" link and log back in as `test02`. Now try deleting one of the books. You should be taken to the red "Permission Denied" message on our error page.

Use one of the 'Logout' links (or go to the http://localhost:3000/logout URL directly) when you are done.

You can jump to the next chapter of the tutorial here: Debugging

## AUTHOR

Kennedy Clark, `hkclark@gmail.com`

Feel free to contact the author for any errors or suggestions, but the best way to report issues is via the CPAN RT Bug system at https://rt.cpan.org/Public/Dist/Display.html?Name=Catalyst-Manual.

Copyright 2006-2011, Kennedy Clark, under the Creative Commons Attribution Share-Alike License Version 3.0 (http://creativecommons.org/licenses/by-sa/3.0/us/).

syntax highlighting: no syntax highlighting ▼

120190 Uploads, 34929 Distributions
178154 Modules, 12986 Uploaders

hosted by YellowBot