Boilerplate Web Apps with Catalyst::::Helper

Presented By
Jamie Pitts

Thursday, April 8, 2010
Concept

Set up a “clone-me” script and templates that can generate catalyst web apps...

☐ Ready to develop with
☐ Architecture implemented
  ☐ Controllers
  ☐ Data model
  ☐ Views / templates
  ☐ Shared resources
☐ Plugins configured

The RepRap
Use Cases

- **Web consulting business:** template application allowing developers to work on new projects in a standard way.

- **Portfolio of similar web sites:** library providing common functionality and a shared data model, but allowing for extensive modifications in the UI.

- **Large-scale web application:** components that are part of the same user experience and share that same library, data model, and UI, yet are running in their own web containers.

- **Tinkerer’s web sites:** you have a catalyst set-up that you want to re-use and refine.
Web Applications In Perl

Before Web Frameworks

- Controller is a “go” method in a cgi script
  > Progresses to: Controller is a mod_perl handler

- URLs follow the cgi directory structure
  > Progresses to: URLs follow the mod_perl handler hierarchy

- Home-brew HTML template system
  > Progresses to: Template Toolkit

- DBI calls in the controller
  > Progresses to: DBI calls in a separate data model
  > Progresses to: DBIx
Problems with Proto-Frameworks

- often a single developer’s brain-child
- often built for a particular use
- often built in a short period of time
- often poorly documented
- can’t wait to start a new one

But... we keep building them because they are fun to build!
Era of Web Frameworks

- Segmented architecture:
  - Controller
  - Data model
  - Views / templates
- Conveniences:
  - URL Routes
  - SQL abstraction
  - Documentation
  - Community
Web Framework Frustrations

- Frameworks are opinionated: architectural decisions are already made.
- Frameworks are constraining: edge cases take too long to build.
- Over-advocacy: crowds out alternatives, drowns out rational discussions.
- You failed to trademark the “X on Rails” snowclone
Catalyst
The Un-opinionated Framework

- ... so that you can be opinionated

- Define your architecture using
  Catalyst Plugins:
  - auth credential
  - auth store
  - session store
  - data model
  - caching
  - logging
  - template system
  - static files
Catalyst
The Un-opinionated Framework

- Highly opinionated parts:
  - Authentication
  - Template naming conventions
  - Configuration style:
    - YAML, JSON, XML, Perl
  - URL dispatch types:
    - path, chained, and regex
Share Your Opinions
Creating the Clone-Me

Step 1: The Bootstrap

☐ Who will be using the Clone-Me?

☐ Create a bootstrap application with catalyst.pl
  ☐ Or use an existing one that has a good set-up

☐ Wire in the critical moving parts:
  ☐ configuration style: perl
  ☐ dispatch style: REST
  ☐ model: DBIC
  ☐ authentication, authorization, session store
  ☐ template naming conventions, php tag style

Thursday, April 8, 2010
Creating the Clone-Me
Step 2: The Common Library

- Move controllers into the common library, then subclass
  - Root controller
  - Auth controller
- Move models into the common library, then subclass
  - User model
  - Consider implementing an API
- Context class
  - Can be moved and subclassed, but it is simpler to just create a Catalyst::Helper template
Creating the Clone-Me
Step 3: The Clone-Me System

- catalyst.pl is an interface to Catalyst::Helper

- Uses template toolkit to build a skeleton application

- Create a Clone-Me directory: CLONEME_web_service

- cloneme.pl script

- lib directory: for the library templates

- templates directory: for the templates templates

  [% tags %] are used by Catalyst::Helper

  <? tags ?> are used by the bootstrap application
Creating the Clone-Me
Step 4: The Clone-Me Script

- CLONEME.pl creates the component directory
- Interacts with your implementation of Catalyst::Helper
- Prompts for variables:
  - $component_dir: user_mgm, video_list
  - $class_name: TigerLead::UserManager, TigerLead::VideoList
  - $default_realm: client, lead, support
- Adds files to svn
Creating the Clone-Me
Step 5: The Helper

- Subclass Catalyst::Helper in the common library
- Overload the mk_app method
  - add relevant variables: dir, realm
  - add new mk calls
- Implement your own template renderer
- Overload or create new mk methods
  - Cover all aspects of your catalyst bootstrap application
    - dirs, models, appclass, rootcontroller, auth controller, readme, symlinks, templates, favicon, images
Creating the Clone-Me
Step 6: Helper Templates

☐ CLONEME

☐ Create templates where your mk methods require it
  ☐ Make your bootstrap templates generic
  ☐ Create a README or welcome page

☐ Helper template variables:

  [% dir %] for referring to the generated component directory

  [% name %] for package declarations, configurations
Creating the Clone-Me
Step 8: Refinement

1. Clone a test_app using the CLONEME.pl
   > overwrite test clones during refinement

2. Make the test_app
   $ cd ..;/test_app
   $ perl MakeFile.PL

3. Run and test the test_app
   # script/test_app_server.pl -p 8080
   http://localhost:8080

4. Go back to ../CLONEME_webservice and refine
Find Some Catalyst Newbies
And Get Them Started