Mojolicious

Marcos Rebelo (oleber@gmail.com)

Mojolicious

- An amazing real-time web framework supporting a simplified single file mode through Mojolicious::Lite.
- Very clean, portable and Object Oriented pure Perl API without any hidden magic and no requirements besides Perl 5.10.1 (although 5.12+ is recommended, and optional CPAN modules will be used to provide advanced functionality if they are installed).

Mojolicious

- Full stack HTTP 1.1 and WebSocket client/server implementation with IPv6, TLS, Bonjour, IDNA, Comet (long polling), chunking and multipart support.
- Built-in non-blocking I/O web server supporting libev and hot deployment, perfect for embedding.
- Automatic CGI and PSGI detection.
- JSON and HTML5/XML parser with CSS3 selector support.

Marcos Rebelo

- 10 years Perl Developer
- Test-driven development fan
- Mojolicious experience:
  - I'm not a Mojolicious developer.
  - A group of JSON Back-ends
  - Short effort on the Front-end

Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius -- and a lot of courage -- to move in the opposite direction.

Albert Einstein

Installation

```
$ sudo cpan Mojolicious
```

Hello World

```
use Mojolicious::Lite;
get '/' => sub {
    shift->render(text => 'Hello World!')
};
app->start;
```

- ```
  $ hello.pl daemon
  Server available at http://127.0.0.1:
  3000.
  ```
- ```
  $ curl http://127.0.0.1:3000/
  Hello World!
  ```

Generator

- There is a helper command to generate a small example application. You may generate multiple things, but two are very interesting.
- `$ mojo generate app`
  Generate Mojolicious application directory structure.
- `$ mojo generate lite_app`
  Generate Mojolicious::Lite application.

Mojolicious::Lite

```
#!/usr/bin/env perl
use Mojolicious::Lite;

# Documentation browser under "/perldoc"
plugin 'PODRenderer';

get '/welcome' => sub {
  my $self = shift;
  $self->render('index');
};

app->start;
```

Mojolicious::Lite

```
__DATA__

@@ index.html.ep
% layout 'default';
% title 'Welcome';
Welcome to Mojolicious!

@@ layouts/default.html.ep
<!doctype html><html>
  <head><title><%= title %></title></head>
  <body><%= content %></body>
</html>
```

Routes

```
get '/welcome' => sub { … };

post '/user' => sub { … };

any '/baz' => sub { … };

any ['get', 'post', 'delete'] =>
  '/bye' => sub { … };
```

GET/POST parameters

```
# /foo?user=Peter
get '/foo' => sub {
  my $self = shift;
  my $user = $self->param('user');
  $self->render(
    text => "Hello $user.");
};
```

Placeholders

```
# /foo/peter
get '/foo/:user' => sub {
  my $self = shift;
  my $user = $self->param('user');
  $self->render(
    text => "Hello $user.");
};
```

- Much more can be told about Placeholders, see the documentation.

Under

```
under sub {
  # Global logic shared by all routes
  my $self = shift;
  return 1 if
    $self->req->headers->header('X-Bender');
  $self->render(text=>"You're not Bender.");
  return;
};

# GET /welcome
get '/welcome' => { text => 'Hi Bender.' };
```

Under

```
group {
  # shared only by routes in this group
  under '/admin' => sub {
    my $self = shift;
    return 1 if login_ok( $self );
    $self->redirect_to('/login_page');
    return
  };
  # GET /admin/dashboard
  get '/dashboard' => { text => 'logged' };
};
```

Sessions

```
get '/counter' => sub {
  my $self = shift;
  $self->session->{counter}++;
};

__DATA__

@@ counter.html.ep
Counter: <%= session 'counter' %>
```

- Signed cookie based sessions just work out of the box as soon as you start using them.

Flash

```
get '/save' => sub {
   my $self = shift;

   $self->flash('success' => 1);
   $c->redirect_to('/show_user');
};
```

- Data storage persistent only for the next request, stored in the session.

Stash

```
# /bar
get '/bar' => sub {
  my $self = shift;
  $self->stash(one => 23);
  $self->render('baz', two => 24);
};

__DATA__
@@ baz.html.ep
Magic numbers: <%= $one %> and <%= $two %>.
```

- The stash is used to pass data to templates.

Log

```
my $log = $self->app->log;

$log->debug("Why isn't this working?");
$log->info("FYI: it happened again");
$log->warn("This might be a problem");
$log->error("Garden variety error");
$log->fatal("Boom!");
```

- Messages will be automatically written to `STDERR` or a '`$mode.log`' file if a log directory exists.

Render

- Rendering text: Perl characters can be rendered with the text stash value, the given content will be automatically encoded to bytes.

```
$self->render(text => 'Hello World!');
```

- Rendering data: Raw bytes can be rendered, no encoding will be performed.

```
$self->render(data => $octets);
```

Render

- Rendering JSON: The json stash value allows you to pass Perl structures to the renderer which get directly encoded to JSON.

```
$self->render(
    json => {foo => [1, 2, 3]});
```

Rendering templates

```
get '/bar' => sub {
  my $self = shift;
  $self->render(template => 'bar');
};

__DATA__
@@ bar.html.ep
Hi <%= param('name') %>
```

- The templates shall in the `__DATA__` session or in the templates directory with the file name `name.format.handler`.

Rendering templates

- The renderer does some magic to find the templates.
- Since we are processing '`/bar`', all this are similar:
  - `$self->render(template => 'bar');`
  - `$self->render('bar');`
  - `$self->render();`
  - You don't even need it. If there is no rendering done, Mojolicious will do it by default.

Embedded Perl

```
<% Perl code %>
<%= Perl expression, replaced with XML escaped result %>
<%== Perl expression, replaced with result %>
<%# Comment, useful for debugging %>
<%% Replaced with "<%", useful for generating templates %>
% Perl code line, treated as "<% line =%>"
%= Perl expression line, treated as "<%= line %>"
%== Perl expression line, treated as "<%== line %>"
%# Comment line, treated as "<%# line =%>"
%% Replaced with "%", useful for generating templates
```

Examples

```
<% my $count = 10; %>
<ul>
    <% for my $index (1 .. $count) { %>
        <li>
            <%= $index %>
        </li>
    <% } %>
</ul>
```

Examples

```
% my $count = 10;
<ul>
    % for my $index (1 .. $count) {
        <li>
            %= $index
        </li>
    % }
</ul>
```

Examples

```
<%= 'lalala' %>          <%# XML escaped %>
<%== '<p>test</p>' %>    <%# not escaped %>
```

Layouts

```
@@ foo/bar.html.ep
% layout 'mylayout', title => 'Hi there';
Hello World!

@@ layouts/mylayout.html.ep
<!DOCTYPE html>
<html>
  <head><title><%= $title %></title></head>
  <body><%= content %></body>
</html>
```

- Most of the time you want to wrap your generated content in a HTML skeleton.

Helpers

```
get '/bar' => sub {
  my $self = shift;
  $self->app->log->debug(
    $self->dumper( [1,2,3] ) );
};

__DATA__

@@ bar.html.ep
<%= dumper( { 'a' => 'b' } ) %>
```

- Helpers are little functions you can use in templates and controller code.

Helpers examples

- **`dumper`**: Dump a Perl data structure using Data::Dumper
- **`app`**: Alias for "app" in Mojolicious::Controller
- **`param`**: Alias for "param".
- **`session`**: Alias for "session".
- **`stash`**: Alias for "stash".
- **`layout`**: Render this template with a layout.
- **`content`**: Insert content into a layout template.

Creating a Helper

```
helper 'prefix' => sub {
  my ( $self, $text, $length ) = @_;
  return length( $text ) > $length - 3
    ? substr($text, 0, $length) . '...'
    : $text;
};

get '/bar' => sub {
  shift->stash('str' => '123456789');
};

__DATA__
@@ bar.html.ep
value: <%= prefix( $str, 5 ) %>
```

Growing

Generator

```
$ mojo generate app MyApp
```

```
my_app                          # Application directory
  |- script                     # Script directory
  |  `- my_app                  # Application script
  |- lib                        # Library directory
  |  |- MyApp.pm                # Application class
  |  `- MyApp                   # Application namespace
  |     `- Example.pm           # Controller class
  |- t                          # Test directory
  |  `- basic.t                 # Random test
  |- log                        # Log directory
  |  `- development.log         # Development mode log file
  |- public                     # Static file directory
  |  `- index.html              # Static HTML file
  `- templates                  # Template directory
     |- layouts                 # Template directory for layouts
     |  `- default.html.ep      # Layout template
     `- example                 # Tmpl dir for "Example"
controller
        `- welcome.html.ep      # Template for "welcome" action
```

my_app/lib/MyApp.pm

```
package MyApp;
use Mojo::Base 'Mojolicious';

# This method will run once at server start
sub startup {
  my $self = shift;

  # Documentation browser under "/perldoc"
  $self->plugin('PODRenderer');

  # Routes
  my $r = $self->routes;

  # Normal route to controller
  $r->route('/welcome')->to('example#welcome');
}

1;
```

Routing

```
# GET /user/123
$r->get('/user/:user_id')
  ->to(cb => sub { ... });

# POST /user/123
$r->post('/user/:user_id')->to(
  controller => 'example',
  action     => 'post_user'
); # Will call: MyApp::Example::post_user

$r->post('/user/:user_id')->to(
  'example#post_user');
```

Route Bridge

```
# POST /auth/user/123
my $r_auth = $r->bridge('/auth')
  ->to( cb => sub { ... } );
$r_auth->post('/user/:user_id')
  ->to('example#hdl_post_user');
```

Controller: lib/MyApp/Example.pm

```
package MyApp::Example;
use Mojo::Base 'Mojolicious::Controller';

# This action will render a template
sub welcome {
  my $self = shift;

  # Render "example/welcome.html.ep"
  $self->render( message => 'Welcome!');
}

1;
```

Testing

```
use Mojo::Base -strict;

use Test::More tests => 4;
use Test::Mojo;

use_ok 'MyApp';

my $t = Test::Mojo->new('MyApp');
$t->get_ok('/welcome')
  ->status_is(200)
  ->content_like(qr/Welcome/i);
```

Testing

- Request:

```
$t->delete_ok('/foo');
$t->get_ok('/foo');
$t->head_ok('/foo');
$t->post_ok('/foo');
$t->post_form_ok(
    '/foo' => {test => 123});
$t->put_ok('/foo');
```

- Header

```
$t->header_is(Expect => 'fun');
$t->header_isnt(Expect => 'fun');
$t->header_like(Expect => qr/fun/);
$t->header_unlike(Expect => qr/fun/);
```

Testing

- Status

```
$t->status_is(200);
$t->status_isnt(200);
```

- Content Type

```
$t->content_type_is('text/html');
$t->content_type_isnt('text/html');
$t->content_type_like(qr/text/);
$t->content_type_unlike(qr/text/);
```

Testing

- Response content:

```
$t->content_is('working!');
$t->content_isnt('working!');
$t->content_like(qr/working!/);
$t->content_unlike(qr/working!/);
```

- CSS3 selectors

```
$t->element_exists('div.foo[x=y]');
$t->element_exists_not('div.foo[x=y]');
$t->text_is('div.foo[x=y]' => 'Hello!');
$t->text_isnt('div.foo[x=y]' => 'Hello!');
$t->text_like('div.foo[x=y]' => qr/Hello/);
$t->text_unlike('div.foo[x=y]' =>
qr/Hello/);
```

Testing

- JSON

```
$t->json_content_is([1, 2, 3]);
$t->json_is('/foo' => {bar => [1, 3]});
$t->json_has('/minibar');
$t->json_hasnt('/minibar');
```

Questions and Answers