“Even though I am in the thralls of Perl 6, I still do all my web development in Perl 5 because the ecology of modules is so mature.”

http://blogs.perl.org/users/ken_youens-clark/2016/10/web-development-with-perl-5.html

Web development and Perl 6

Bailador

BreakDancer	

Crust

Web	

Web::App::Ballet

Web::App::MVC	

Web::RF

Bailador	Nov 2016
BreakDancer	Mar 2014
Crust	Jan 2016
Web	May 2016
Web::App::Ballet	Jun 2015
Web::App::MVC	Mar 2013
Web::RF	Nov 2015

“Even though I am in the thralls of Perl 6, I still do all my web development in Perl 5 because the ecology of modules is so mature.”

http://blogs.perl.org/users/ken_youens-clark/2016/10/web-development-with-perl-5.html

Crust

Web

Bailador

to the rescue

Bailador config

```
my %settings;
multi sub setting(Str $name) {
    %settings{$name}
}
multi sub setting(Pair $pair) {
    %settings{$pair.key} = $pair.value
}

setting 'database'     => $*TMPDIR.child('dancr.db');
# webscale authentication method
setting 'username'     => 'admin';
setting 'password'     => 'password';
setting 'layout'       => 'main';
```

Bailador DB

```
sub connect_db() {
    my $dbh = DBIish.connect(
        'SQLite',
        :database(setting('database').Str)
    );
    return $dbh;
}

sub init_db() {
    my $db = connect_db;
    my $schema = slurp 'schema.sql';
    $db.do($schema);
}
```

Bailador handler

```
get '/' => {
    my $db = connect_db();
    my $sth = $db.prepare(
        'select id, title, text from entries order by id desc'
    );
    $sth.execute;
    layout template 'show_entries.tt', {
        msg           => get_flash(),
        add_entry_url => uri_for('/add'),
        entries       => $sth.allrows(:array-of-hash)
                             .map({$_<id> => $_}).hash,
        session       => session,
    };
}
```

Bailador - Logging in

```
post '/login' => sub (*@a) {
    my $err;

    # process form input
    if request.params<username> ne setting('username') {
        $err = "Invalid username";
    }
    elsif request.params<password> ne setting('password') {
        $err = "Invalid password";
    }
    else {
        session<logged_in> = True;
        set_flash('You are logged in.');
        return redirect '/';
    }

    # display login form
    layout template 'login.tt', {err => $err};
}
```

Bailador helpers

```
sub uri_for($path) {
    return $path;
}

sub layout($content) {
    template 'layouts/' ~ setting('layout') ~ '.tt', {
        css_url    => '/css/style.css',
        login_url  => uri_for('/login'),
        logout_url => uri_for('/logout'),
        session    => session,
        content    => $content,
    };
}
```

“Even though I am in the thralls of Perl 6, I still do all my web development in Perl 5 because the ecology of modules is so mature.”

http://blogs.perl.org/users/ken_youens-clark/2016/10/web-development-with-perl-5.html


SINCE 2009
PERL
DANCER
is FUN
PERL
AGAIN

First dancing steps

```
use Dancer2:from<Perl5>;

get '/' => sub {
    'Hello World!';
};

start;
```

First dancing steps

```
use Dancer2:from<Perl5>;

get '/' => sub ($app) {
    'Hello World!';
};

start;
```

First dancing steps

```
use Dancer2:from<Perl5>;

get '/' => {
    'Hello World!';
};

start;
```

Dancr

```
use v6.c;

unit class Dancr;

use Dancer2:from<Perl5>;
use DBI:from<Perl5>;
use Template:from<Perl5>;

set 'database'     => $*TMPDIR.child('dancr.db');
set 'session'      => 'Simple';
set 'template'     => 'template_toolkit';
set 'logger'       => 'console';
set 'log'          => 'debug';
set 'show_errors'  => 1;
set 'startup_info' => 1;
set 'warnings'     => 1;
set 'username'     => 'admin';
set 'password'     => 'password';
set 'layout'       => 'main';
```

DBI

```
sub connect_db() {
    return DBI.connect(
        'dbi:SQLite:dbname='
            ~ setting('database'),
        Any,
        Any,
        ${sqlite_unicode => 1},
    ) or die %*PERL5<$DBI::errstr>;
}
```

Hooks

```
hook before_template_render =>
sub (%tokens) {
    %tokens<css_url>
        = request.base ~ 'css/style.css';
    %tokens<login_url>
        = uri_for('/login');
    %tokens<logout_url>
        = uri_for('/logout');
}
```

Performance

```
get '/' => {
    my $db  = connect_db();
    my $sql = 'select * from entries order by id desc';
    my $sth = $db.prepare($sql) or die $db.errstr;
    $sth.execute or die $sth.errstr;

    template 'show_entries.tt', Map.new((
        msg           => get_flash(),
        add_entry_url => uri_for('/add'),
        entries       => $sth.fetchall_hashref('id'),
    )).item;
}
```

Context

```
post '/add' => {
    send_error("Not logged in", 401)
        unless session('logged_in');

    my $db = connect_db();
    my $sql = 'insert into entries (title, text)'
         ~ 'values (?, ?)';
    my $sth = $db.prepare($sql) or die $db.errstr;
    $sth.execute(
        body_parameters.get('title'),
        body_parameters.get('text')
    ) or die $sth.errstr;

    set_flash('New entry posted!');
    redirect '/';
}
```


mojolicious

Mojolicious spaces

```
use Mojolicious::Lite:from<Perl5>;

get '/' => sub ($c) {
    $c.render(text => 'Hello World!');
}

app.start;
```

Mojolicious templates

```
use Mojolicious::Lite:from<Perl5>;

get '/' => sub ($c) {
    $c.stash(:one<23>);
    $c.render(:template<magic>, :two<24>);
}

app.start;

=finish

@@ magic.html.ep
The magic numbers are <%= $one %> and <%= $two %>.
```

Mojolicious fat comma

```
# Render the template "index.html.ep"
get '/' => sub ($c) {
    $c.render;
}, 'index';

# Render the template "hello.html.ep"
get '/hello';
```

Mojolicious Regexes

```
# /1

# /123

any '/:bar' => [bar => rx/\d+/], sub ($c) {
    my $bar = $c.param('bar');
    $c.render(
        text => "Our :bar placeholder matched $bar",
    );
};
```

Mojolicious Websockets

```
websocket '/echo' => sub ($c) {
    $c.on(json => sub ($c, $hash) {
        $hash<msg> = "echo: $hash<msg>";
        $c.send(${json => $hash});
    });
}
```


catalyst
Web Framework

Catalyst Controller

```
package XStats::Controller::Root;
use Moose;
use namespace::autoclean;

BEGIN { extends 'Catalyst::Controller'; }

#
# Sets the actions in this controller to be registered with no prefix
# so they function identically to actions created in MyApp.pm
#
__PACKAGE__->config(namespace => '');

__PACKAGE__→meta→make_immutable;

...
```

Catalyst Controller cont.

```
. . .
use v6::inline;
use CatalystX::Perl6::Component::Perl5Attributes;

method index($c) is Path is Args(0) {
    $c.stash({
        template  => 'index.zpt',
        uri_graph => $c.uri_for('graph'),
    });
}
```

CatalystX::Perl6::Component::Perl5Attributes

```
use Inline::Perl5;

multi trait_mod:<is>(Routine $declarand, :$Path!) is export {
    unless $declarand.does(Inline::Perl5::Perl5Attributes) {
        $declarand does Inline::Perl5::Perl5Attributes;
    }
    $declarand.attributes.push(
        'Path' ~ ($Path.isa(Bool) ?? '' !! "('$Path')")
    );
}
```

“~~Even though~~ As I am in the thralls of Perl 6, I still do all my web development ~~in~~ with Perl 5 because the ecology of modules is so mature.”

Thank You!

http://niner.name/talks/
http://github.com/niner/

A couple of weeks ago, Perl weekly linked to a blog post about web development with Perl 5.
It started like this:

“Even though I am in the thralls of Perl 6, I still do all my web development in Perl 5 because the ecology of modules is so mature.”

http://blogs.perl.org/users/ken_youens-clark/2016/10/web-development-with-perl-5.html

When I read this sentence, it kind of rubbed me the wrong way. Here’s someone who likes Perl 6, yet cannot use it for what he needs to be doing.

This is unacceptable.

I cannot believe that after so many people poured so much time into Perl 6, it should not be ready for something as mundane as web development.

So I set out on my quest to show that this sentence is misguided or just obsolete.

Web development and Perl 6

So, where do we begin?
How do we show that everything is not as bad as this blogger believes?
Well I did a little survey of modules.perl6.org and had a look at everything that sounded like a web framework to me.
These are the results:

Bailador
BreakDancer
Crust
Web
Web::App::Ballet
Web::App::MVC
Web::RF

As you can see, it's a reasonably short list. In itself that's not a bad thing as the list of actively developed web frameworks for Perl 5 probably is not much longer.

Of course, there are lots and lots and lots of failed attempts or abandoned frameworks on CPAN.

This raises the second question: to which category do these belong?

Bailador	Nov 2016
BreakDancer	Mar 2014
Crust	Jan 2016
Web	May 2016
Web::App::Ballet	Jun 2015
Web::App::MVC	Mar 2013
Web::RF	Nov 2015

To answer this question, I had a look at the date of the last commit.

Keep in mind that the first stable release of Perl 6 as a languages was cut on December 25th 2015 and in the months immediately before this release, we did a whole lot of semantic changes that we knew we had to do before we want to commit to stability in the language.

So everything that has not been changed in or after those months can be considered either very lucky or probably bitrotted and unmaintained.

This leaves just 3 contenders.

“Even though I am in the thralls of Perl 6, I still do all my web development in Perl 5 because the ecology of modules is so mature.”

http://blogs.perl.org/users/ken_youens-clark/2016/10/web-development-with-perl-5.html

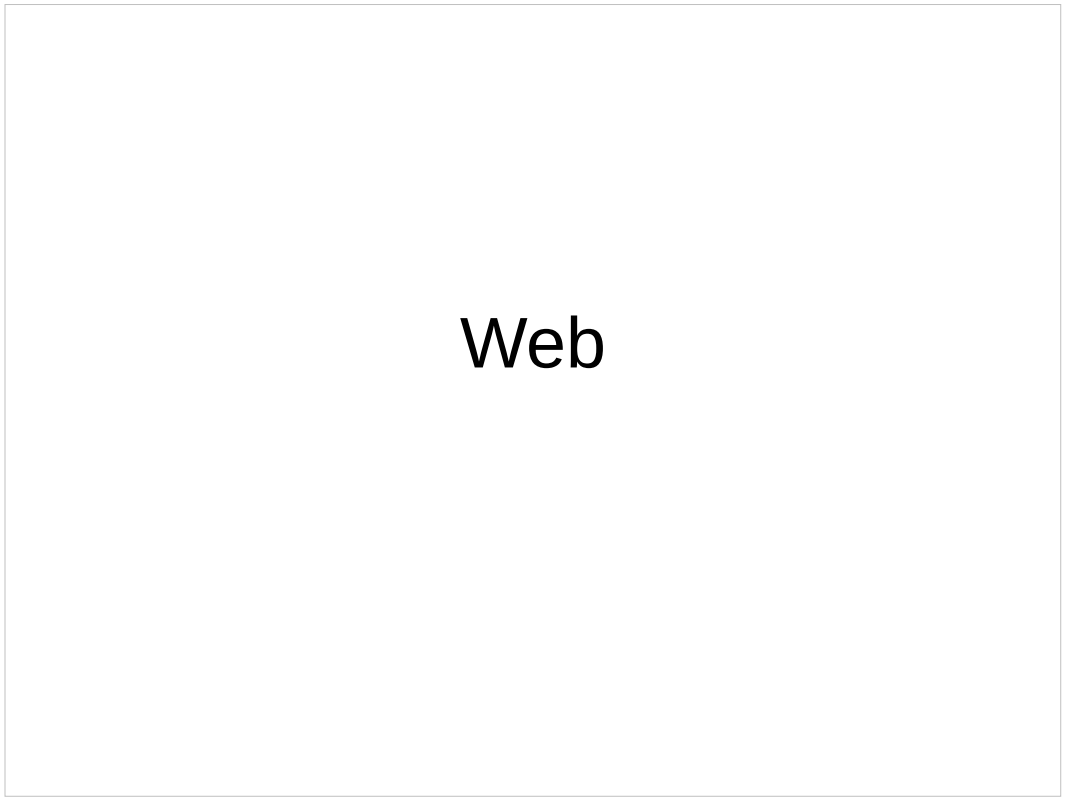
No, no, no. Let's not jump to conclusions.
3 can still be a very decent pool to chose from.
Just have a look.

Crust

Crust is in short a PSGI implementation in Perl 6.

It is a glue between the web framework and the web server.

Though certainly very useful, this is probably not the first thing, you'll be looking for. But let's keep it in mind for now.



Web can be considered one level above Crust in the stack. It gives you request and response objects and a dispatcher.

Not luxurious, but sounds like a start. Except for that it apparently does not even support fancy features like file uploads yet.

There's another thing missing that I'd consider a somewhat essential part. I will come back to this later.

For now, let's move on to the first thing we should have considered.

Bailador

to the rescue

- Bailador is pretty much a straight forward port of the popular Dancer framework to Perl 6. It's light weight, is being actively developed (though slowly) and brings much of what you'd expect from a web framework.
- It is a little thin on the documentation and examples. But since it's a Dancer port, I resolved to just porting a canonical Dancer example to Bailador to see how well it does.

Bailador config

```
my %settings;
multi sub setting(Str $name) {
    %settings{$name}
}
multi sub setting(Pair $pair) {
    %settings{$pair.key} = $pair.value
}

setting 'database'     => $*TMPDIR.child('dancr.db');
# webscale authentication method
setting 'username'     => 'admin';
setting 'password'     => 'password';
setting 'layout'       => 'main';
```

Bailador does not have any configuration system integrated, so first off, I wrote a very minimalistic one, so I could stick as close as possible to the Dancer example.

Bailador DB

```
sub connect_db() {
    my $dbh = DBIish.connect(
        'SQLite',
        :database(setting('database').Str)
    );
    return $dbh;
}

sub init_db() {
    my $db = connect_db;
    my $schema = slurp 'schema.sql';
    $db.do($schema);
}
```

Here we have just a bit of database set up, nothing special. Though I have to mention that I very much like that DBIish uses named arguments instead of parsing some connection string. Though its odd, that the SQLite driver doesn’t cope too well with getting a proper IO::Path object for the database so I have to explicitly turn it into a string here.

Bailador handler

```
get '/' => {
    my $db = connect_db();
    my $sth = $db.prepare(
        'select id, title, text from entries order by id desc'
    );
    $sth.execute;
    layout template 'show_entries.tt', {
        msg           => get_flash(),
        add_entry_url => uri_for('/add'),
        entries       => $sth.allrows(:array-of-hash)
                             .map({$_<id> => $_}).hash,
        session       => session,
    };
}
```

Now if you know a bit of Dancer, you'll feel right at home here, as this is pretty much the same as you'd write in Perl 5.

We define a route for GET requests to the root and give Bailador a code block to execute when the route matches. In Dancer you'd have to give it a subroutine reference instead of just a block.

Again DBIish's interface is a bit cleaner. The advantage of being able to start over really.

5 minutes

Bailador - Logging in

```
post '/login' => sub (*@a) {
    my $err;

    # process form input
    if request.params<username> ne setting('username') {
        $err = "Invalid username";
    }
    elsif request.params<password> ne setting('password') {
        $err = "Invalid password";
    }
    else {
        session<logged_in> = True;
        set_flash('You are logged in.');
        return redirect '/';
    }

    # display login form
    layout template 'login.tt', {err => $err};
}
```

- The login handler seems a good example for how to access parameters, sessions and redirects.
- Very straight forward, easy to use, very familiar from Dancer.
- So have I convinced you, that Perl 6 is absolutely ready?
- Who of you noticed the interesting bit in the last line? Instead of just calling the template function as you'd do in Dancer, I pass the result to a function called layout. Why is that?

Bailador helpers

```
sub uri_for($path) {
    return $path;
}

sub layout($content) {
    template 'layouts/' ~ setting('layout') ~ '.tt', {
        css_url    => '/css/style.css',
        login_url  => uri_for('/login'),
        logout_url => uri_for('/logout'),
        session    => session,
        content    => $content,
    };
}
```

Well Bailador simply hasn’t seen as much development as Dancer or Mojolicious.

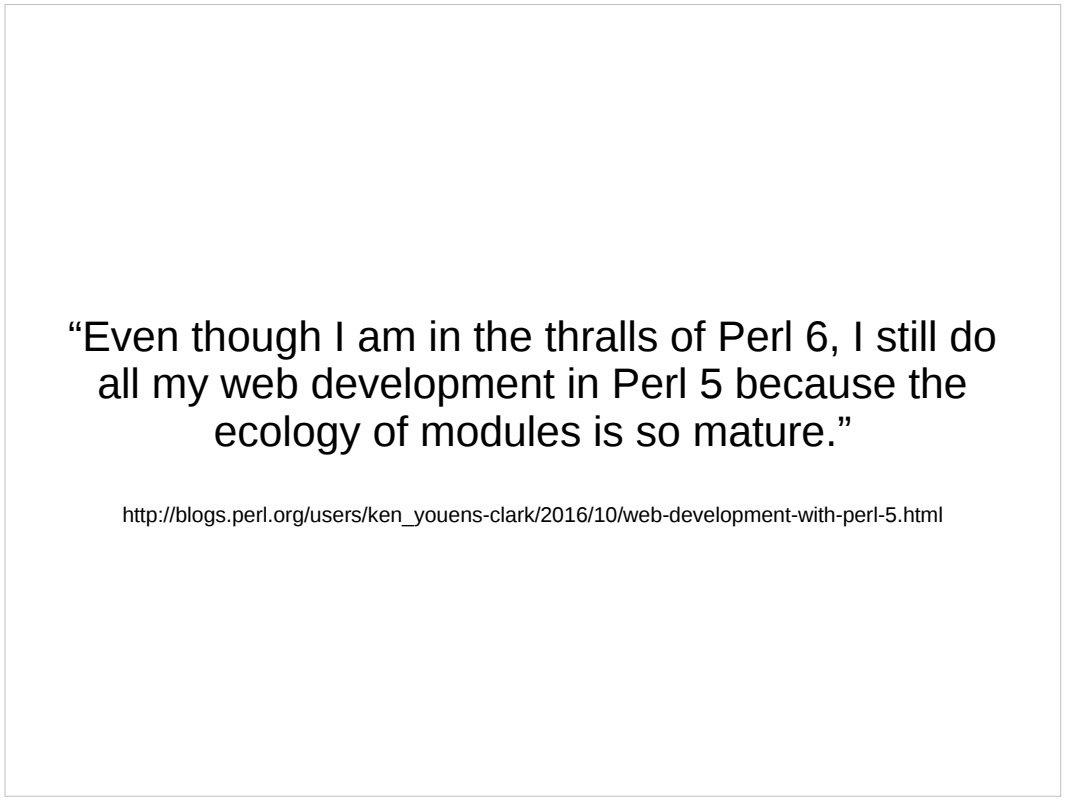
It has yet to gain builtin support for layout templates or hooks, so I had to write a little helper. Nothing bad really.

There’s also no facility to construct URIs. So for now our little example application cannot be run with a path prefix.

Again probably not a deal breaker.

It just shows that Bailador is not as mature a framework as Dancer or Mojolicious.

Wait a minute...



Doh!

Do you remember Crust, the PSGI framework for Perl 6? The cool thing about PSGI is that it allows for sharing so called middlewares between applications and frameworks.

There’s tons of middlewares from support for running behind a front end proxy to awesome debug panels.

In theory, you could have the same with Crust.

But middlewares have yet to appear.

So is there no hope?

Are we stuck really with the choice between a mature ecology of modules and the devotion to our new love?

Screw it, I want to have both.

If Dancer2 is where the features are, then Dancer2 I will use.

First dancing steps

```
use Dancer2:from<Perl5>;

get '/' => sub {
    'Hello World!';
};

start;
```

Now this is kind of like the canonical Hello World in Dancer2 taken straight out of its tutorial.

This might actually still be simple enough to run.

Of course, I wouldn't show it to you unless, I actually tried it.

So I go out on a limb and do my first live demo in years!

First dancing steps

```
use Dancer2:from<Perl5>;

get '/' => sub ($app) {
    'Hello World!';
};

start;
```

So what happened?

Well the documentation does not tell you this, but Dancer2 actually passes the application object to your handler routine.

Since Perl 6 does have proper subroutine signatures, it expects you to declare parameters and complains if arguments were passed, but none were expected.

So we could do that to make Perl 6 happy.

But that would also make it more tedious and less pretty!

And we cannot have that, can we?

First dancing steps

```
use Dancer2:from<Perl5>;

get '/' => {
    'Hello World!';
};

start;
```

Luckily, tadzik, who first tried this, found a really pretty way around this issue.

In Perl 6, code blocks are not just syntactic elements, but first class objects.

You can think of subroutines as code blocks with signatures.

From that point of view, it's only natural, that naked code blocks don't care about arguments. So in a way, they are actually closer to Perl 5's anonymous subroutines.

Now let’s have a look at how well we do at the original Dancr example I ported to Bailador.

Dancr

```
use v6.c;

unit class Dancr;

use Dancer2:from<Perl5>;
use DBI:from<Perl5>;
use Template:from<Perl5>;

set 'database'     => $*TMPDIR.child('dancr.db');
set 'session'      => 'Simple';
set 'template'     => 'template_toolkit';
set 'logger'       => 'console';
set 'log'          => 'debug';
set 'show_errors'  => 1;
set 'startup_info' => 1;
set 'warnings'     => 1;
set 'username'     => 'admin';
set 'password'     => 'password';
set 'layout'       => 'main';
```

I'm not gonna show the full source code, that would mostly be just boring like this bunch of code.

Interesting bits may just be to point out that I do stick with the tutorial closely, so I do use Perl 5's DBI and Template Toolkit.

Also one of the little improvements of Perl 6 that matter so much in daily life as a programmer is that we have File::Spec's functionality integrated.

So we can just use the $*TMPDIR variable and its convenient child method for platform independent storing of our database file.

DBI

```
sub connect_db() {
    return DBI.connect(
        'dbi:SQLite:dbname='
            ~ setting('database'),
        Any,
        Any,
        ${sqlite_unicode => 1},
    ) or die %*PERL5<$DBI::errstr>;
}
```

As I mentioned, I just use the DBI which all of you should know.
So this will look familiar to you but also a little strange.
First of all, undef is gone.
Even undefined values are typed in Perl 6.
The closest equivalent to a plain undef is an undefined Any.
Now what the hell is this Dollar-Hash thingy?
The answer is, that it's an itemized hash.
In other words, it's a hash that should be treated like a single item and that's the important part, not flattened.
Especially not into the argument list of the method call.
Lastly, this slide shows how to access global Perl 5 variables from within Perl 6.

10 minutes

Hooks

```
hook before_template_render =>
sub (%tokens) {
    %tokens<css_url>
        = request.base ~ 'css/style.css';
    %tokens<login_url>
        = uri_for('/login');
    %tokens<logout_url>
        = uri_for('/logout');
}
```

On to a more mundane piece of code, the before_template_render hook.

As you probably know much better than me, this is just a sub that gets passed a hash in which we can store additional values.

This is mostly just translated syntax.

Now please, a quick show of hands: how many of you have forgotten to add the semicolon after the closing curly of the sub at one point or another?

Happens to me all the time, which is why I'm so glad, that Perl 6 has fixed this for us.

Yes, indeed, we do not actually need the semicolon there anymore.

Performance

```
get '/' => {
    my $db  = connect_db();
    my $sql = 'select * from entries order by id desc';
    my $sth = $db.prepare($sql) or die $db.errstr;
    $sth.execute or die $sth.errstr;

    template 'show_entries.tt', Map.new((
        msg           => get_flash(),
        add_entry_url => uri_for('/add'),
        entries       => $sth.fetchall_hashref('id'),
    )).item;
}
```

Now this is the heart of our program, the code that actually delivers the page.

The only specialty is the Map thingy there.

Now what is that about?

When we pass a hash from Perl 6 to Perl 5, we expect writes to this hash to be visible in Perl 6.

The way to achieve this is by tieing a Perl 5 hash to a Perl 6 hash.

This causes all access to be slower which is quite visible in benchmarks.

Maps now are immutable hashes in Perl 6. Since they are immutable, we can get away with just copying its contents to a pure Perl 5 hash for added speed.

Context

```
post '/add' => {
    send_error("Not logged in", 401)
        unless session('logged_in');

    my $db = connect_db();
    my $sql = 'insert into entries (title, text)'
         ~ 'values (?, ?)';
    my $sth = $db.prepare($sql) or die $db.errstr;
    $sth.execute(
        body_parameters.get('title'),
        body_parameters.get('text')
    ) or die $sth.errstr;

    set_flash('New entry posted!');
    redirect '/';
}
```

Now the final piece of Dancer code I'm gonna show is this bit.
There's nothing out of the ordinary here.
In fact, it's actually closer to what the Dancer documentation suggests than the tutorial itself.
I'm talking about using the body_parameters accessor instead of the param function.
Other than this being the recommended way anyway, there's a reason for this deviation.
param is context sensitive.
It behaves different in list context than in scalar context.
Now Perl 6 does not have this distinction, so Inline::Perl5 calls all functions in list context because it's the most general.

Next I had a look at Mojolicious.

As Mojolicious is quite similar to Dancer, the results were similar as well.

So instead of a boring introduction, I will focus on a couple of stumbling blocks I discovered when porting all the tutorial examples.

Mojolicious spaces

```
use Mojolicious::Lite:from<Perl5>;

get '/' => sub ($c) {
    $c.render(text => 'Hello World!');
}

app.start;
```

This hello world should be pretty much what you expected.

Just note the space before the argument list of the anonymous subroutine. If you leave that out, Perl 6 will think you want to call a function called “sub” and complain about Variable '$c' not being declared.

Mojolicious templates

```
use Mojolicious::Lite:from<Perl5>;

get '/' => sub ($c) {
    $c.stash(:one<23>);
    $c.render(:template<magic>, :two<24>);
}

app.start;

=finish

@@ magic.html.ep
The magic numbers are <%= $one %> and <%= $two %>.
```

It would be really cool if this example worked as shown. Mojolicous supports storing templates in the __DATA__ section of your program.

In Perl 6 instead of the magic DATA label, you use the =finish POD command to declare the rest of the file as data. But this is not (yet) passed on to Perl 5 code.

So instead we have to move the template code to an external template file. But quite honestly, you should do that anyway. Least of all to get proper syntax highlighting for HTML code.

Mojolicious fat comma

```
# Render the template "index.html.ep"
get '/' => sub ($c) {
    $c.render;
}, 'index';

# Render the template "hello.html.ep"
get '/hello';
```

When preparing the example showing how to give a route a name, I stumbled over another subtle difference between Perl 5 and 6.

The fat comma operator is now syntax for constructing a Pair object consisting of a key and a value.

If you chain those, you will actually create two nested pairs while the Perl 5 code actually just expects a flat list of values.

#Talk: Don't use => here!

Mojolicious Regexes

```
# /1
# /123
any '/:bar' => [bar => rx/\d+/], sub ($c) {
    my $bar = $c.param('bar');
    $c.render(
        text => "Our :bar placeholder matched $bar",
    );
};
```

Now this is the first example that really doesn't work. Right now it's unfortunately impossible to pass a regex from Perl 6 to Perl 5.

I actually do have a patch to implement this support, but it's still not enough for Mojolicious, as the latter relies on stringification of the regex to embed it in a larger dispatch matcher.

I think it's a problem that can be solved. Just for now, this is a restriction.

Mojolicious Websockets

```
websocket '/echo' => sub ($c) {
    $c.on(json => sub ($c, $hash) {
        $hash<msg> = "echo: $hash<msg>";
        $c.send(${json => $hash});
    });
}
```

What does work on the other hand is websockets. And I can't help but find this incredibly cool.

16 minutes

For something that looks a bit different than the last 3 frameworks, Catalyst is an interesting candidate.

Catalyst puts a bit more pressure on you to stick to an MVC architecture.

It automatically loads your module, view and controller modules and of course it expects those to be written in Perl 5.

It even goes a step further and generates the boilerplate code for you.

Catalyst Controller

```
package XStats::Controller::Root;
use Moose;
use namespace::autoclean;

BEGIN { extends 'Catalyst::Controller'; }

#
# Sets the actions in this controller to be registered with no prefix
# so they function identically to actions created in MyApp.pm
#
__PACKAGE__->config(namespace => '');

__PACKAGE__→meta→make_immutable;

...
```

This is code which we probably better just leave as it is.

So we also won't waste much time on it.

The far more interesting question is, how we get our Perl 6 code in there?

Catalyst Controller cont.

```
. . .
use v6::inline;
use CatalystX::Perl6::Component::Perl5Attributes;

method index($c) is Path is Args(0) {
    $c.stash({
        template  => 'index.zpt',
        uri_graph => $c.uri_for('graph'),
    });
}
```

The answer is by declaring that the rest of the file is written in Perl 6.

Everything from the “use v6::inline” statement on is plain Perl 6 code.

We have an index method with appropriate Catalyst attributes making the method a Catalyst action.

Of course we get passed the context object and can use the usual methods to get our work done.

This works pretty much out of the box, except for the Catalyst attributes.

In Perl 6, there’s no such thing as subroutine attributes.

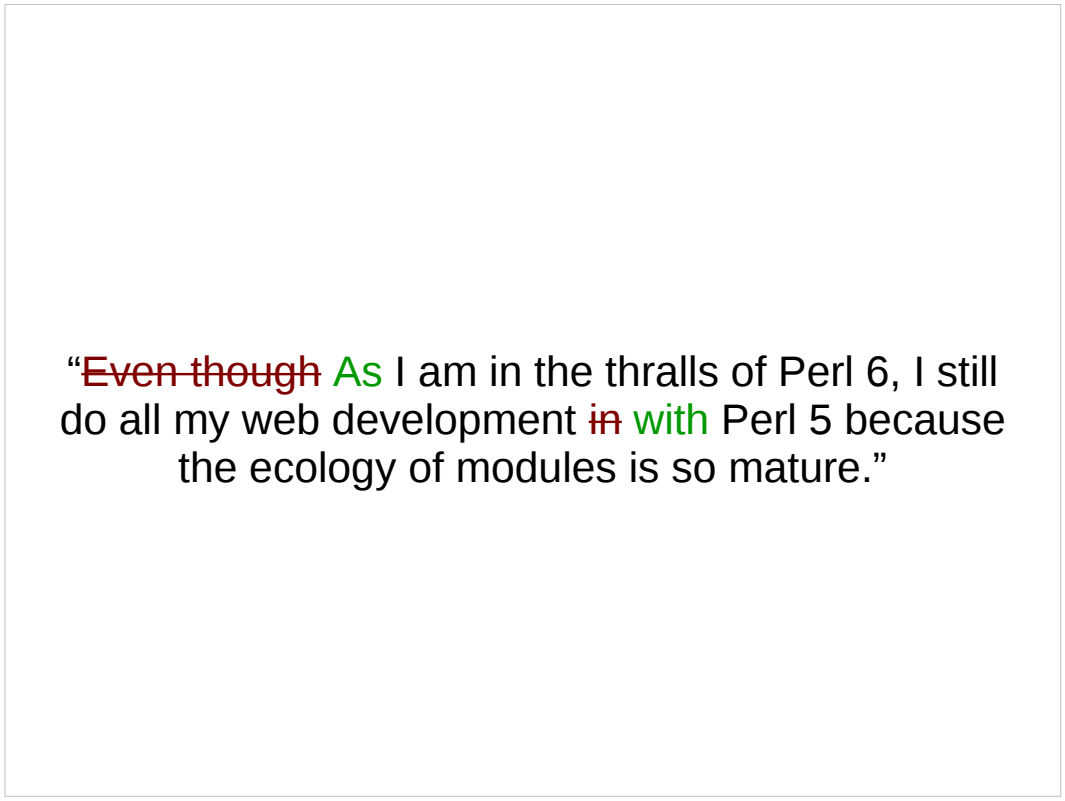
Instead, there’s a more general mechanism called Traits.

CatalystX::Perl6::Component::Perl5Attributes

```
use Inline::Perl5;

multi trait_mod:<is>(Routine $declarand, :$Path!) is export {
    unless $declarand.does(Inline::Perl5::Perl5Attributes) {
        $declarand does Inline::Perl5::Perl5Attributes;
    }
    $declarand.attributes.push(
        'Path' ~ ($Path.isa(Bool) ?? '' !! "('$Path')")
    );
}
```

- Traits can for example be a convenient way to add a role to those method objects.
- For Catalyst actions, we want to mix in the Inline::Perl5::Perl5Attributes role.
- This role provides an “attributes” attribute, which is just a list of attributes to apply to the Perl 5 wrapper method that gets generated for our inlined Perl 6 methods.
- The helper module provides two more traid_mod functions for supporting the Args and ActionClass attributes. They look pretty much exactly like this one.
- And that’s it.
- That’s all you should have to know to be able to use Catalyst in a Perl 6 web application.



And now I know what’s been bothering me about the sentence that haunted us so far.

It takes just two really tiny changes for me to be able to make peace with it.

Because even though it is certainly fun building a new web framework, I sometimes just need the reliable tools, I’m so familiar with.

Luckily, they are still there for me.

Thank You!

http://niner.name/talks/
http://github.com/niner/

What's really impossible is to find any high resolution logos of any of the covered web frameworks.