Testing tools

I can’t live without

Ilya Chesnokov

Ideal
Development Process

Design

Documentation

Tests

Code

Doing the tests afterwords
is like putting a condom on
after you’ve come

(someone on IRC)

- Design
- Documentation
- Tests
- Code

- Design
- Documentation
- Tests
- Code

API

PSGI

Test PSGI API

- Plack::Test
- LWP::Protocol::PSGI
- HTTP::Message::PSGI
- Test::WWW::Mechanize::PSGI
- Framework-specific tests: Dancer::Test, Test::Mojo, Catalyst::Test, etc…

Typical PSGI app test

```
use Plack::Test;
use Test::Deep ':v1';
use JSON::XS qw(decode_json);
use HTTP::Status qw(:constants);
use HTTP::Request::Common;

test_psgi $psgi_app, sub ($cb) {
    my $res = $cb->(GET '/item/nonexistent');

    is $res->status, HTTP_NOT_FOUND;
    cmp_deeply(
        decode_json($content),
        { error => 'Item not found: nonexistent' },
        'Querying nonexistent item results in a proper error'
    );
};
```

```
use Plack::Test;
use Test::Deep ':v1';
use JSON::XS qw(decode_json);
use HTTP::Status qw(:constants);
use HTTP::Request::Common;

test_psgi $psgi_app, sub ($cb) {
    my $res = $cb->(GET '/item/nonexistent');

    is $res->status, HTTP_NOT_FOUND;
    cmp_deeply(
        decode_json($content),
        { error => 'Item not found: nonexistent' },
        'Querying nonexistent item results in a proper error'
    );
};
```

```perl
use Plack::Test;
use Test::Deep ':v1';
use JSON::XS qw(decode_json);
use HTTP::Status qw(:constants);
use HTTP::Request::Common;

test_psgi $psgi_app, sub ($cb) {
    my $res = $cb->(GET '/item/nonexistent');

    is $res->status, HTTP_NOT_FOUND;
    cmp_deeply(
        decode_json($content),
        { error => 'Item not found: nonexistent' },
        'Querying nonexistent item results in a proper error'
    );
};
```

```
use Plack::Test;
use Test::Deep ':v1';
use JSON::XS qw(decode_json);
use HTTP::Status qw(:constants);
use HTTP::Request::Common;

test_psgi $psgi_app, sub ($cb) {
    my $res = $cb->(GET '/item/nonexistent');

    is $res->status, HTTP_NOT_FOUND;
    cmp_deeply(
        decode_json($content),
        { error => 'Item not found: nonexistent' },
        'Querying nonexistent item results in a proper error'
    );
};
```

```
use Plack::Test;
use Test::Deep ':v1';
use JSON::XS qw(decode_json);
use HTTP::Status qw(:constants);
use HTTP::Request::Common;

test_psgi $psgi_app, sub ($cb) {
    my $res = $cb->(GET '/item/nonexistent');

    is $res->status, HTTP_NOT_FOUND;
    cmp_deeply(
        decode_json($content),
        { error => 'Item not found: nonexistent' },
        'Querying nonexistent item results in a proper error'
    );
};
```

```
test_psgi $psgi_app, sub ($cb) {
    my $res = $cb->(GET '/item/nonexistent');
    is $res->status, HTTP_NOT_FOUND;
    cmp_deeply(
        decode_json($content),
        { error => 'Item not found: nonexistent' },
        'Querying nonexistent item results in a proper error'
    );

    $res = $cb->(GET "/item/$item_id");
    is $res->status, HTTP_OK;
    cmp_deeply(
        decode_json($content),
        {
          name => 'Pulse Rifle',
          weight => 10,
          price => 2000,
        }
        'Get real item details'
    );
};
```

```
test_psgi $psgi_app, sub ($cb) {
    my $res = $cb->(GET '/item/nonexistent');
    is $res->status, HTTP_NOT_FOUND;
    cmp_deeply(
        decode_json($content),
        { error => 'Item not found: nonexistent' },
        'Querying nonexistent item results in a proper error'
    );

    $res = $cb->(GET "/item/$item_id");
    is $res->status, HTTP_OK;
    cmp_deeply(
        decode_json($content),
        {
          name => 'Pulse Rifle',
          weight => 10,
          price => 2000,
        }
        'Get real item details'
    );
};
```

```perl
test_psgi $psgi_app, sub ($cb) {
  my $res = $cb->(GET '/item/nonexistent');
  is $res->status, HTTP_NOT_FOUND;
  cmp_deeply(
    decode_json($content),
    { error => 'Item not found: nonexistent' },
    'Querying nonexistent item results in a proper error'
  );

  $res = $cb->(GET "/item/$item_id");
  is $res->status, HTTP_OK;
  cmp_deeply(
    decode_json($content),
    {
      name   => 'Pulse Rifle',
      weight => 10,
      price  => 2000,
    },
    'Get real item details'
  );

  $res = $cb->(GET "/item");
  is $res->status, HTTP_OK;
  cmp_deeply(
    decode_json($content),
    {
      items => [
        {
          name   => 'Pulse Rifle',
          weight => 10,
          price  => 2000,
        },
        {
          name   => 'Hard Armor',
          weight => 12,
          price  => 3000,
        },
      ],
    }
  );

  $res = $cb->(
    POST "/item",
    Content => encode_json(
      {
        name  => 'Kitchen knife',
        mass  => 5,
        price => 100,
      }
    )
  );
  is $res->status, HTTP_OK;
  cmp_deeply(
    decode_json($content),
    {
      name   => 'Kitchen knife',
      weight => 5,
      price  => 100,
    }
  );
};
```

```perl
test_psgi $psgi_app, sub ($cb) {
  my $res = $cb->(GET '/item/nonexistent');
  is $res->status, HTTP_NOT_FOUND;
  cmp_deeply(
    decode_json($content),
    { error => 'Item not found: nonexistent' },
    'Querying nonexistent item results in a proper error'
  );

  $res = $cb->(GET "/item/$item_id");
  is $res->status, HTTP_OK;
  cmp_deeply(
    decode_json($content),
    {
      name   => 'Pulse Rifle',
      weight => 10,
      price  => 2000,
    },
    'Get real item details'
  );

  $res = $cb->(GET "/item");
  is $res->status, HTTP_OK;
  cmp_deeply(
    decode_json($content),
    {
      items => [
        {
          name   => 'Pulse Rifle',
          weight => 10,
          price  => 2000,
        },
        {
          name   => 'Hard Armor',
          weight => 12,
          price  => 3000,
        },
      ],
    }
  );

  $res = $cb->(
    POST "/item",
    Content => encode_json(
      {
        name  => 'Kitchen knife',
        mass  => 5,
        price => 100,
      }
    )
  );
  is $res->status, HTTP_OK;
  cmp_deeply(
    decode_json($content),
    {
      name   => 'Kitchen knife',
      weight => 5,
      price  => 100,
    }
  );
};
```

```perl
test_psgi $psgi_app, sub ($cb) {
  my $res = $cb->(GET '/item/nonexistent');
  is $res->status, HTTP_NOT_FOUND;
  cmp_deeply(
    decode_json($content),
    { error => 'Item not found: nonexistent' },
    'Querying nonexistent item results in a proper error'
  );

  $res = $cb->(GET "/item/$item_id");
  is $res->status, HTTP_OK;
  cmp_deeply(
    decode_json($content),
    {
      name   => 'Pulse Rifle',
      weight => 10,
      price  => 2000,
    },
    'Get real item details'
  );

  $res = $cb->(GET "/item");
  is $res->status, HTTP_OK;
  cmp_deeply(
    decode_json($content),
    {
      items => [
        {
          name   => 'Pulse Rifle',
          weight => 10,
          price  => 2000,
        },
        {
          name   => 'Hard Armor',
          weight => 12,
          price  => 3000,
        },
      ],
    }
  );

  $res = $cb->(
    POST "/item",
    Content => encode_json(
      {
        name  => 'Kitchen knife',
        mass  => 5,
        price => 100,
      }
    )
  );
  is $res->status, HTTP_OK;
  cmp_deeply(
    decode_json($content),
    {
      name   => 'Kitchen knife',
      weight => 5,
      price  => 100,
    }
  );
};
```

```perl
$res = $cb->(
  POST "/item",
  Content => encode_json(
    {
      name  => 'Kitchen knife',
      mass  => 5,
      price => 100,
    }
  )
);
is $res->status, HTTP_OK;
cmp_deeply(
  decode_json($content),
  {
    name   => 'Kitchen knife',
    weight => 5,
    price  => 100,
  },
  'Item created successfully'
);
```

```
$res = $cb->(
  POST "/item",
  Content => encode_json(
    {
      name  => 'Kitchen knife',
      mass  => 5,
      price => 100,
    }
  )
);
is $res->status, HTTP_OK;
cmp_deeply(
  decode_json($content),
  {
    name   => 'Kitchen knife',
    weight => 5,
    price  => 100,
  },
  'Item created successfully'
);
```

```
$res = $cb->(
  POST "/item",
  Content => encode_json(
    {
      name  => 'Kitchen knife',
      mass  => 5,
      price => 100,
    }
  )
);
is $res->status, HTTP_OK;
cmp_deeply(
  decode_json($content),
  {
    name   => 'Kitchen knife',
    weight => 5,
    price  => 100,
  },
  'Item created successfully'
);
```

```
$res = $cb->(
  POST "/item",
  Content => encode_json(
    {
      name  => 'Kitchen knife',
      mass  => 5,
      price => 100,
    }
  )
);
is $res->status, HTTP_OK;
cmp_deeply(
  decode_json($content),
  {
    name   => 'Kitchen knife',
    weight => 5,
    price  => 100,
  },
  'Item created successfully'
);
```

```
$res = $cb->(
  POST "/item",
  Content => encode_json(
    {
      name  => 'Kitchen knife',
      mass  => 5,
      price => 100,
    }
  )
);
is $res->status, HTTP_OK;
cmp_deeply(
  decode_json($content),
  {
    name   => 'Kitchen knife',
    weight => 5,
    price  => 100,
  },
  'Item created successfully'
);
```

```
$res = $cb->(
  POST "/item",
  Content => encode_json(
    {
      name  => 'Kitchen knife',
      mass  => 5,
      price => 100,
    }
  )
);
is $res->status, HTTP_OK;
cmp_deeply(
  decode_json($content),
  {
    name   => 'Kitchen knife',
    weight => 5,
    price  => 100,
  },
  'Item created successfully'
);
```

```
POST "/item",

  {
    name  => 'Kitchen knife',
    mass  => 5,
    price => 100,
  }


       status, HTTP_OK;


{
  name   => 'Kitchen knife',
  weight => 5,
  price  => 100,
},
'Item created successfully'
```

```
POST "/item",

  {
    name  => 'Kitchen knife',
    mass  => 5,
    price => 100,
  }

       status, HTTP_OK;


{
  name   => 'Kitchen knife',
  weight => 5,
  price  => 100,
},
'Item created successfully'
```

Request

```
POST "/item",

  {
    name  => 'Kitchen knife',
    mass  => 5,
    price => 100,
  }
```

Request

```
       status, HTTP_OK;


{
  name   => 'Kitchen knife',
  weight => 5,
  price  => 100,
},
'Item created successfully'
```

Response

```
POST "/item",

  {
    name  => 'Kitchen knife',
    mass  => 5,
    price => 100,
  }

       status, HTTP_OK;


{
  name   => 'Kitchen knife',
  weight => 5,
  price  => 100,
},
'Item created successfully'
```

Request

Response

Test title

```
api_ok(
    call   => [
        POST => "/item" => {
            name  => 'Kitchen knife',
            mass  => 5,
            price => 100,
        }
    ],
    expect => {
        status       => HTTP_OK,
        json_content => {
            name   => 'Kitchen knife',
            weight => 5,
            price  => 100,
        },
    },
    title => 'Item created successfully',
);
```

```
api_ok(
    title => 'Item created successfully',
    call  => [
        POST => "/item" => {
            name  => 'Kitchen knife',
            mass  => 5,
            price => 100,
        }
    ],
    expect => {
        status       => HTTP_OK,
        json_content => {
            name   => 'Kitchen knife',
            weight => 5,
            price  => 100,
        },
    },
);
```

```
api_ok(
    'Item created successfully',
    [
        POST => "/item" => {
            name  => 'Kitchen knife',
            mass  => 5,
            price => 100,
        }
    ],
    {
        status       => HTTP_OK,
        json_content => {
            name   => 'Kitchen knife',
            weight => 5,
            price  => 100,
        },
    },
);
```

Test::Class::Moose

Controller

<->

TestsFor::Controller

```
Controller::method()
<->
TestsFor::Controller::test_method()
```

```
$test->api_ok(
    'Item created successfully',
    [
        POST => "/item" => {
            name  => 'Kitchen knife',
            mass  => 5,
            price => 100,
        }
    ],
    {
        status       => HTTP_OK,  # optional
        json_content => {
            name   => 'Kitchen knife',
            weight => 5,
            price  => 100,
        },
    },
);
```

$test object

- Contains **psgi_client** as a property
    - initialised with $psgi_app
- Knows about
    - HTTP headers to apply to request
    - Route prefix, if any
- Can perform authentication or other preliminary steps

Test::TCM::Role::API

Test::TCM::Role::API

(draft name)

Achtung!

Draft code!

$test->api_ok()

- Generates request
- Sends it through PSGI client / app
- Parses response
- Tests parsed response against expected data

```
use Test::Deep qw( cmp_deeply );
use Test::Differences qw(eq_or_diff);
use Test::More;

my $json_content = {
    error => 'Item does not exist'
};
my $expected = {
    json_content => {
        error => 'Item does not exist: nonexistent',
    },
};

cmp_deeply(
    $json_content,
    $expected->{json_content},
    'Data is as expected'
) or eq_or_diff( $json_content, $expected->{json_content} );
```

```perl
use Test::Deep qw( cmp_deeply );
use Test::Differences qw(eq_or_diff);
use Test::More;

my $json_content = {
    error => 'Item does not exist'
};
my $expected = {
    json_content => {
        error => 'Item does not exist: nonexistent',
    },
};

cmp_deeply(
    $json_content,
    $expected->{json_content},
    'Data is as expected'
) or eq_or_diff( $json_content, $expected->{json_content} );
```

```
not ok 1 - Data is as expected

#   Failed test 'Data is as expected'
#   at 4-test-response.pl line 20.
# Compared $data->{"error"}
#    got : 'Item does not exist'
# expect : 'Item does not exist: nonexistent'
not ok 2

#   Failed test at 4-test-response.pl line 20.
# +----+-----------------------------------+-----------------------------------------------+
# | Elt|Got                                |Expected                                       |
# +----+-----------------------------------+-----------------------------------------------+
# |   0|{                                  |{                                              |
# *   1|  error => 'Item does not exist'   |  error => 'Item does not exist: nonexistent'  *
# |   2|}                                  |}                                              |
# +----+-----------------------------------+-----------------------------------------------+
not ok 1 - Data is as expected

#   Failed test 'Data is as expected'
#   at 4-test-response.pl line 20.
# Compared $data->{"error"}
#    got : 'Item does not exist'
# expect : 'Item does not exist: nonexistent'
not ok 2

#   Failed test at 4-test-response.pl line 20.
# +----+----------------------------------+-----------------------------------------------+
# | Elt|Got                               |Expected                                       |
# +----+----------------------------------+-----------------------------------------------+
# |   0|{                                 |{                                              |
# *   1|  error => 'Item does not exist'  |  error => 'Item does not exist: nonexistent'  *
# |   2|}                                 |}                                              |
# +----+----------------------------------+-----------------------------------------------+
```

```perl
use Test::Deep qw(:v1);
use Test::Differences qw(eq_or_diff);
use Test::More;

my $json_content = { success => ignore() };
my $expected = {
    json_content => {
        error => 'Item does not exist: nonexistent',
    },
};

cmp_deeply(
    $json_content,
    $expected->{json_content},
    'Data is as expected'
) or eq_or_diff( $json_content, $expected->{json_content} );
```

```
use Test::Deep qw(:v1);
use Test::Differences qw(eq_or_diff);
use Test::More;

my $json_content = { success => ignore() };
my $expected = {
    json_content => {
        error => 'Item does not exist: nonexistent',
    },
};

cmp_deeply(
    $json_content,
    $expected->{json_content},
    'Data is as expected'
) or eq_or_diff( $json_content, $expected->{json_content} );
```

```
not ok 1 - Data is as expected

#   Failed test 'Data is as expected'
#   at 5-test-response.pl line 17.
# Comparing hash keys of $data
# Missing: 'error'
# Extra: 'success'
not ok 2

#   Failed test at 5-test-response.pl line 17.
# +----+------------------------------------------------+------------------------------------------------+
# | Elt|Got                                             |Expected                                        |
# +----+------------------------------------------------+------------------------------------------------+
# |   0|{                                               |{                                               |
# *   1|  success => bless( {}, 'Test::Deep::Ignore' )  |  error => 'Item does not exist: nonexistent'   *
# |   2|}                                               |}                                               |
# +----+------------------------------------------------+------------------------------------------------+
1..2
# Looks like you failed 2 tests of 2.
```

```
not ok 1 - Data is as expected

#   Failed test 'Data is as expected'
#   at 5-test-response.pl line 17.
# Comparing hash keys of $data
# Missing: 'error'
# Extra: 'success'
not ok 2

#   Failed test at 5-test-response.pl line 17.
# +----+------------------------------------------------+-----------------------------------------------+
# | Elt|Got                                             |Expected                                       |
# +----+------------------------------------------------+-----------------------------------------------+
# |   0|{                                               |{                                              |
# *   1|  success => bless( {}, 'Test::Deep::Ignore' )  |  error => 'Item does not exist: nonexistent'  *
# |   2|}                                               |}                                              |
# +----+------------------------------------------------+-----------------------------------------------+
1..2
# Looks like you failed 2 tests of 2.
```

"Expectations-based tests"

```
expect($something)

some_code() # has side effects, e.g. writes logs

check_something_is_as_expected()
```

```
expect($something)

some_code() # has side effects, e.g. writes logs

check_something_is_as_expected()
```

```
expect($something)

some_code() # has side effects, e.g. writes logs

check_something_is_as_expected()
```

```
expect($something)

some_code() # has side effects, e.g. writes logs

check_something_is_as_expected()
```

Side effects

- Logs
- SQL queries
- Emails
- ...

Logs

Log::Log4perl

Test::Log::Log4perl

```perl
# Get the loggers
my $logger   = Log::Log4perl->get_logger("Foo::Bar");
my $expected = Test::Log::Log4perl->get_logger("Foo::Bar");

# Start testing
Test::Log::Log4perl->start();

# Declare we're going to log something
$expected->error("This is a test");

# Log that something
$logger->error("This is a test");

# Test that those things matched
Test::Log::Log4perl->end("Test that that logs okay");
```

```
# Get the loggers
my $logger   = Log::Log4perl->get_logger("Foo::Bar");
my $expected = Test::Log::Log4perl->get_logger("Foo::Bar");

# Start testing
Test::Log::Log4perl->start();

# Declare we're going to log something
$expected->error("This is a test");

# Log that something
$logger->error("This is a test");

# Test that those things matched
Test::Log::Log4perl->end("Test that that logs okay");
```

```
# Get the loggers
my $logger   = Log::Log4perl->get_logger("Foo::Bar");
my $expected = Test::Log::Log4perl->get_logger("Foo::Bar");

# Start testing
Test::Log::Log4perl->start();

# Declare we're going to log something
$expected->error("This is a test");

# Log that something
$logger->error("This is a test");

# Test that those things matched
Test::Log::Log4perl->end("Test that that logs okay");
```

```perl
# Get the loggers
my $logger  = Log::Log4perl->get_logger("Foo::Bar");
my $expected = Test::Log::Log4perl->get_logger("Foo::Bar");

# Start testing
Test::Log::Log4perl->start();

# Declare we're going to log something
$expected->error("This is a test");

# Log that something
$logger->error("This is a test");

# Test that those things matched
Test::Log::Log4perl->end("Test that that logs okay");
```

```
# Get the loggers
my $logger  = Log::Log4perl->get_logger("Foo::Bar");
my $expected = Test::Log::Log4perl->get_logger("Foo::Bar");

# Start testing
Test::Log::Log4perl->start();

# Declare we're going to log something
$expected->error("This is a test");

# Log that something
$logger->error("This is a test");

# Test that those things matched
Test::Log::Log4perl->end("Test that that logs okay");
```

```perl
# Get the loggers
my $logger   = Log::Log4perl->get_logger("Foo::Bar");
my $expected = Test::Log::Log4perl->get_logger("Foo::Bar");

# Start testing
Test::Log::Log4perl->start();

# Declare we're going to log something
$expected->error("This is a test");

# Log that something
$logger->error("This is a test");

# Test that those things matched
Test::Log::Log4perl->end("Test that that logs okay");
```

We've built our own wrapper...

...but it's not released

:-(

SQL

Count SQL queries

DBIx::Class

```
DBIx::Class::Storage->debug(1)
```

```
DBIx::Class::Storage
->debugcb(sub ($op, $query) { ... })
```

```perl
package My::Test;

use Test::Class::Moose;
use My::DBIC::Schema;

with qw(
 Test::TCM::Role::SQL
);

sub schema { My::DBIC::Schema->connected_schema; }

sub test_something ( $test, $ ) {
    $test->expect_sql_count(0);
    some_code();
    $test->sql_count_ok("some_code() did't call database");
}
```

```perl
package My::Test;

use Test::Class::Moose;
use My::DBIC::Schema;

with qw(
 Test::TCM::Role::SQL
);

sub schema { My::DBIC::Schema->connected_schema; }

sub test_something ( $test, $ ) {
    $test->expect_sql_count(0);
    some_code();
    $test->sql_count_ok("some_code() did't call database");
}
```

```
package My::Test;

use Test::Class::Moose;
use My::DBIC::Schema;

with qw(
 Test::TCM::Role::SQL
);

sub schema { My::DBIC::Schema->connected_schema; }

sub test_something ( $test, $ ) {
    $test->expect_sql_count(0);
    some_code();
    $test->sql_count_ok("some_code() did't call database");
}
```

```
package My::Test;

use Test::Class::Moose;
use My::DBIC::Schema;

with qw(
 Test::TCM::Role::SQL
);

sub schema { My::DBIC::Schema->connected_schema; }

sub test_something ( $test, $ ) {
    $test->expect_sql_count(0);
    some_code();
    $test->sql_count_ok("some_code() did't call database");
}
```

```
package My::Test;

use Test::Class::Moose;
use My::DBIC::Schema;

with qw(
 Test::TCM::Role::SQL
);

sub schema { My::DBIC::Schema->connected_schema; }

sub test_something ( $test, $ ) {
    $test->expect_sql_count(0);
    some_code();
    $test->sql_count_ok("some_code() did't call database");
}
```

```perl
package My::Test;

use Test::Class::Moose;
use My::DBIC::Schema;

with qw(
 Test::TCM::Role::SQL
);

sub schema { My::DBIC::Schema->connected_schema; }

sub test_something ( $test, $ ) {
    $test->expect_sql_count(0);
    some_code();
    $test->sql_count_ok("some_code() did't call database");
}
```

```
sub expect_sql_count ($test, $expected_sql_count) {
    local $Test::Builder::Level = $Test::Builder::Level + 1;

    $test->_expected_sql_count($expected_sql_count);

    my $storage = $test->schema->storage;
    $storage->debug(1);
    weaken(my $weak_test = $test);
    $storage->debugcb(
        sub {
            my ($op, $info) = @_;
            $weak_test->_inc_sql_count;
            $weak_test->_old_debugcb->(@_);
            $weak_test->_add_sql_call($info);
        }
    );
}
```

```
sub expect_sql_count ($test, $expected_sql_count) {
    local $Test::Builder::Level = $Test::Builder::Level + 1;

    $test->_expected_sql_count($expected_sql_count);

    my $storage = $test->schema->storage;
    $storage->debug(1);
    weaken(my $weak_test = $test);
    $storage->debugcb(
        sub {
            my ($op, $info) = @_;
            $weak_test->_inc_sql_count;
            $weak_test->_old_debugcb->(@_);
            $weak_test->_add_sql_call($info);
        }
    );
}
```

```perl
sub expect_sql_count ($test, $expected_sql_count) {
    local $Test::Builder::Level = $Test::Builder::Level + 1;

    $test->_expected_sql_count($expected_sql_count);

    my $storage = $test->schema->storage;
    $storage->debug(1);
    weaken(my $weak_test = $test);
    $storage->debugcb(
        sub {
            my ($op, $info) = @_;
            $weak_test->_inc_sql_count;
            $weak_test->_old_debugcb->(@_);
            $weak_test->_add_sql_call($info);
        }
    );
}
```

```perl
sub sql_count_ok ($test, $title = '') {
    if (!$test->_has_expected_sql_count) {
        croak 'expect_sql_count() must be called before sql_count_ok()';
    }

    my $result = is(
        $test->_sql_count,
        $test->_expected_sql_count,
        $title || 'SQL count is as expected'
    );
    if (!$result) {
        if ($test->_has_queries) {
            diag "Performed SQL queries: [\n"
              . join("\n", @{ $test->_queries }) . "\n";
        }
    }

    $test->_reset_sql_count;
    $test->_clear_expected_sql_count;
    $test->_clear_queries;
    $test->schema->storage->debug($test->_old_debug);
    $test->schema->storage->debugcb($test->_old_debugcb);
}
```

```
sub sql_count_ok ($test, $title = '') {
    if (!$test->_has_expected_sql_count) {
        croak 'expect_sql_count() must be called before sql_count_ok()';
    }

    my $result = is(
        $test->_sql_count,
        $test->_expected_sql_count,
        $title || 'SQL count is as expected'
    );
    if (!$result) {
        if ($test->_has_queries) {
            diag "Performed SQL queries: [\n"
              . join("\n", @{ $test->_queries }) . "\n";
        }
    }

    $test->_reset_sql_count;
    $test->_clear_expected_sql_count;
    $test->_clear_queries;
    $test->schema->storage->debug($test->_old_debug);
    $test->schema->storage->debugcb($test->_old_debugcb);
}
```

```
sub sql_count_ok ($test, $title = '') {
    if (!$test->_has_expected_sql_count) {
        croak 'expect_sql_count() must be called before sql_count_ok()';
    }

    my $result = is(
        $test->_sql_count,
        $test->_expected_sql_count,
        $title || 'SQL count is as expected'
    );
    if (!$result) {
        if ($test->_has_queries) {
            diag "Performed SQL queries: [\n"
              . join("\n", @{ $test->_queries }) . "\n";
        }
    }

    $test->_reset_sql_count;
    $test->_clear_expected_sql_count;
    $test->_clear_queries;
    $test->schema->storage->debug($test->_old_debug);
    $test->schema->storage->debugcb($test->_old_debugcb);
}
```

```
sub sql_count_ok ($test, $title = '') {
    if (!$test->_has_expected_sql_count) {
        croak 'expect_sql_count() must be called before sql_count_ok()';
    }

    my $result = is(
        $test->_sql_count,
        $test->_expected_sql_count,
        $title || 'SQL count is as expected'
    );
    if (!$result) {
        if ($test->_has_queries) {
            diag "Performed SQL queries: [\n"
              . join("\n", @{ $test->_queries }) . "\n";
        }
    }

    $test->_reset_sql_count;
    $test->_clear_expected_sql_count;
    $test->_clear_queries;
    $test->schema->storage->debug($test->_old_debug);
    $test->schema->storage->debugcb($test->_old_debugcb);
}
```

Test::TCM::Role::SQL

Test::TCM::Role::SQL

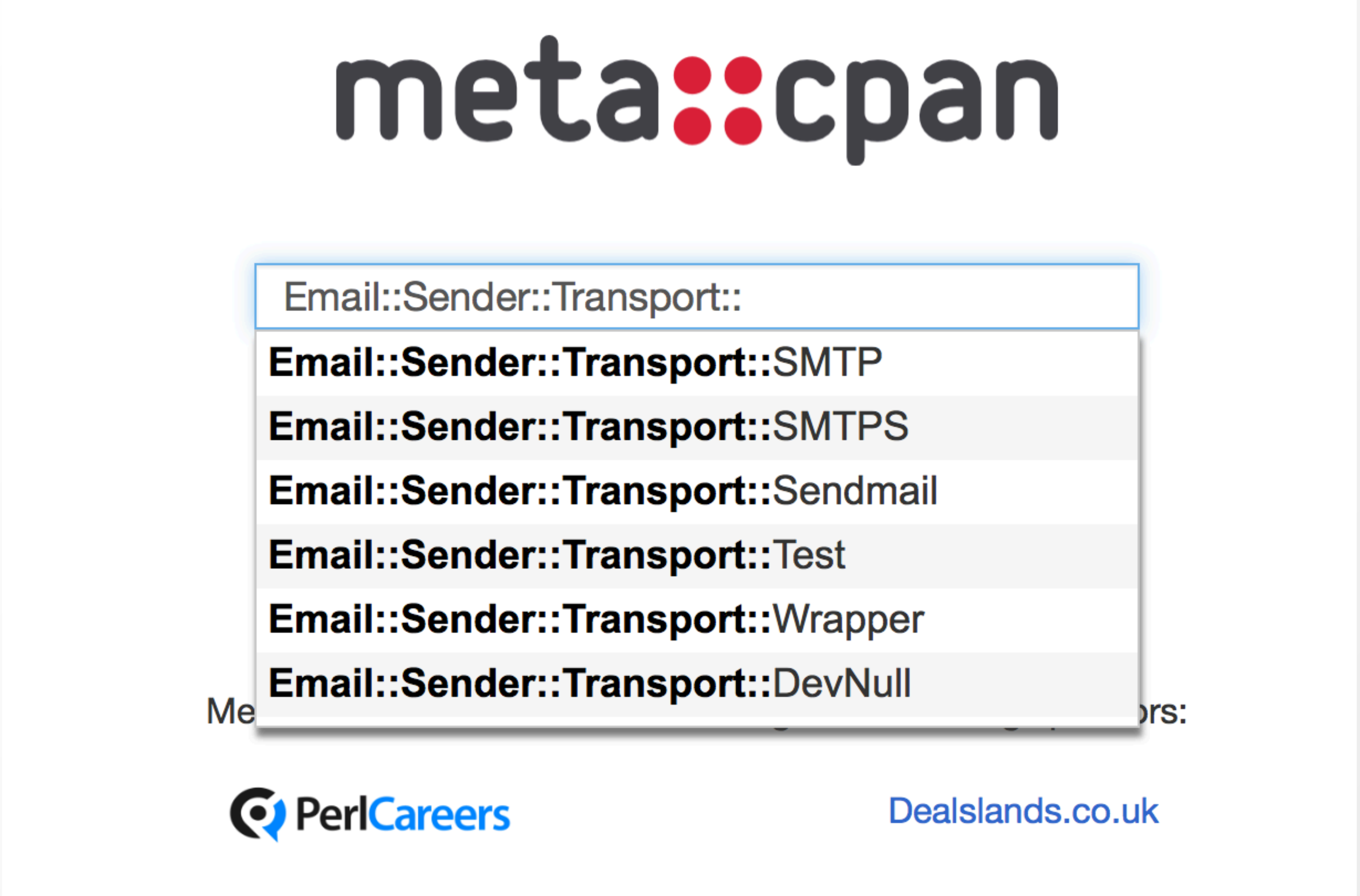
(draft name)

Took just several hours to create

...still might be useful to others

Email

Email::Sender::Simple



meta::cpan

Email::Sender::Transport::

Email::Sender::Transport::SMTP

Email::Sender::Transport::SMTPS

Email::Sender::Transport:: Sendmail

Email::Sender::Transport:: Test

Email::Sender::Transport:: Wrapper

Email::Sender::Transport:: DevNull

Me rs:

PerlCareers

Dealslands.co.uk

```perl
use Test::Class::Moose;
with qw(Test::TCM::Role::Email);

sub test_emails_sent ($test, $) {
    $test->expect_emails(
        {
            name => 'my_template',
            to   => 'someone@example.com',
            html => '<b>Hello!</b>',
            text => 'Hello!',
        }
    );

    # ... code that sends email via Email::Sender::Simple ...
    some_code();

    $test->emails_ok;
}
```

```
use Test::Class::Moose;
with qw(Test::TCM::Role::Email);

sub test_emails_sent ($test, $) {
    $test->expect_emails(
        {
            name => 'my_template',
            to   => 'someone@example.com',
            html => '<b>Hello!</b>',
            text => 'Hello!',
        }
    );

    # ... code that sends email via Email::Sender::Simple ...
    some_code();

    $test->emails_ok;
}
```

```
use Test::Class::Moose;
with qw(Test::TCM::Role::Email);

sub test_emails_sent ($test, $) {
    $test->expect_emails(
        {
            name => 'my_template',
            to   => 'someone@example.com',
            html => '<b>Hello!</b>',
            text => 'Hello!',
        }
    );

    # ... code that sends email via Email::Sender::Simple ...
    some_code();

    $test->emails_ok;
}
```

```
use Test::Class::Moose;
with qw(Test::TCM::Role::Email);

sub test_emails_sent ($test, $) {
    $test->expect_emails(
        {
            name => 'my_template',
            to   => 'someone@example.com',
            html => '<b>Hello!</b>',
            text => 'Hello!',
        }
    );

    # ... code that sends email via Email::Sender::Simple ...
    some_code();

    $test->emails_ok;
}
```

```
use Test::Class::Moose;
with qw(Test::TCM::Role::Email);

sub test_emails_sent ($test, $) {
    $test->expect_emails(
        {
            name => 'my_template',
            to   => 'someone@example.com',
            html => '<b>Hello!</b>',
            text => 'Hello!',
        }
    );

    # ... code that sends email via Email::Sender::Simple ...
    some_code();

    $test->emails_ok;
}
```

Test::TCM::Role::Email

Test::TCM::Role::Email

(draft name)

To reiterate...

- Test::TCM::Role::API
- Test::TCM::Role::SQL
- Test::TCM::Role::Email
- Test::TCM::Role::Logs

To reiterate...

- Test::TCM::Role::API - PSGI
- Test::TCM::Role::SQL - DBIx::Class
- Test::TCM::Role::Email - Email::Sender::Simple
- Test::TCM::Role::Logs - Test::Log::Log4perl
  - to be released (maybe sometime)

Thoughts / ideas

- Roles or Classes? (Not only use with TCM)
- Extract common behaviour for Expectations-based tests
  - make it easier to create new similar testing "frameworks"
  - parameterised role maybe?
- Other API output checking rules (not just Test::Deep)
  - custom DSL, Test2::Tools::Compare, JSON schema, etc

https://github.com/ichesnokov

Questions?

Thank you