Веб-приложения изнутри

Содержание

Протокол HTTP

HTTP/1.1 messages (HTTP/1.1 сообщения)

```
HTTP-message   = Request | Response

generic-message = start-line
*(message-header CRLF)
CRLF
[ message-body ]
start-line      = Request-Line | Status-Line
```

Заголовки

```
message-header = field-name ":" [ field-value ]
field-name     = token
field-value    = *( field-content | LWS )
field-content  = <the octets making up the field-
value and consisting of either *text or combination
of token, separators, and quoted-string>
```

Протокол HTTP

HTTP = HyperText Transfer Protocol

Стандарт http://tools.ietf.org/html/rfc2616

Протокол HTTP

Формат ответа

```
Response      = Status-Line  ; Section 6.1
*(( general-header           ; Section 4.5
| response-header            ; Section 6.2
| entity-header ) CRLF)      ; Section 7.1
CRLF
[ message-body ]             ; Section 7.2

Status-Line = HTTP-Version SP Status-Code SP
              Reason-Phrase CRLF
```

Статус

- 1** - информационные сообщения от сервера клиенту
- 2** - успешная обработка запроса
- 3** - контент находится в другом месте или не изменялся
- 4** - ошибка обработки запроса (клиент неправильно сформировал пакет)
- 5** - ошибка обработки запроса (проблема на сервере)

Протокол HTTP

Пример простейшего запроса

```
say 'GET / HTTP/1.1\r\nHost:search.cpan.org\r\n\r\n
```

```
GET / HTTP/1.1
Host: search.cpan.org
```

Ответ

```
HTTP/1.1 200 OK
Date: Mon, 13 Apr 2015 20:19:35 GMT
Server: Plack/Starman (Perl)
Content-Length: 3623
Content-Type: text/html

__CONTENT__
```

Протокол HTTP

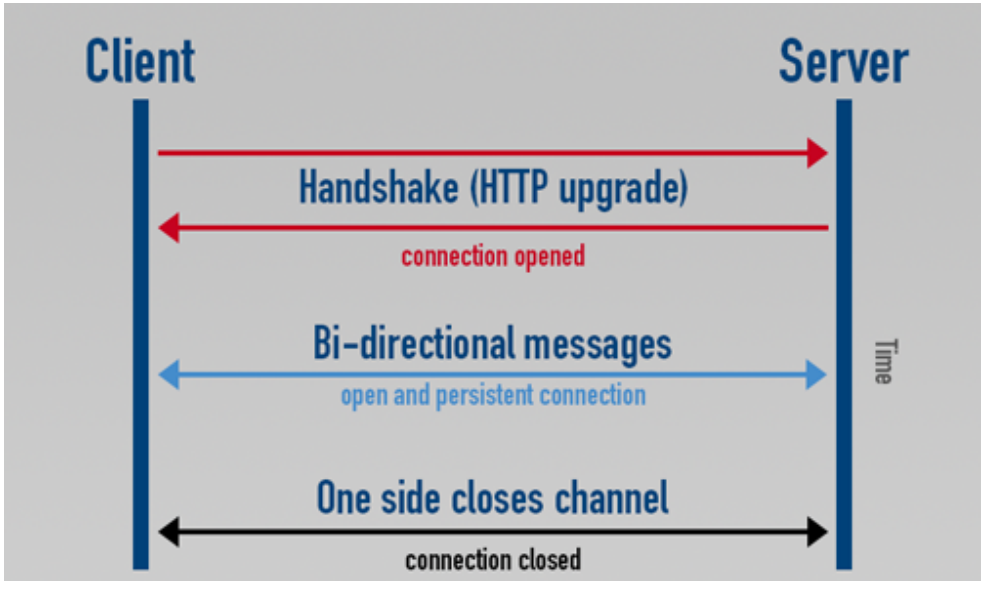
Распространённые методы

Метод	Пояснение
GET	позволяет получить информацию от сервера, тело запроса всегда остается пустым;
HEAD	аналогичен GET, но тело ответа остается всегда пустым, позволяет проверить доступность запрашиваемого ресурса и прочитать HTTP-заголовки ответа;
POST	позволяет загрузить информацию на сервер, по смыслу изменяет ресурс на сервере, но зачастую используется и для создания ресурса на сервере, тело запроса содержит изменяемый/создаваемый ресурс;
PUT	аналогичен POST, но по смыслу занимается созданием ресурса, а не его изменением, тело запроса содержит создаваемый ресурс;
DELETE	удаляет ресурс с сервера.

Протокол HTTP

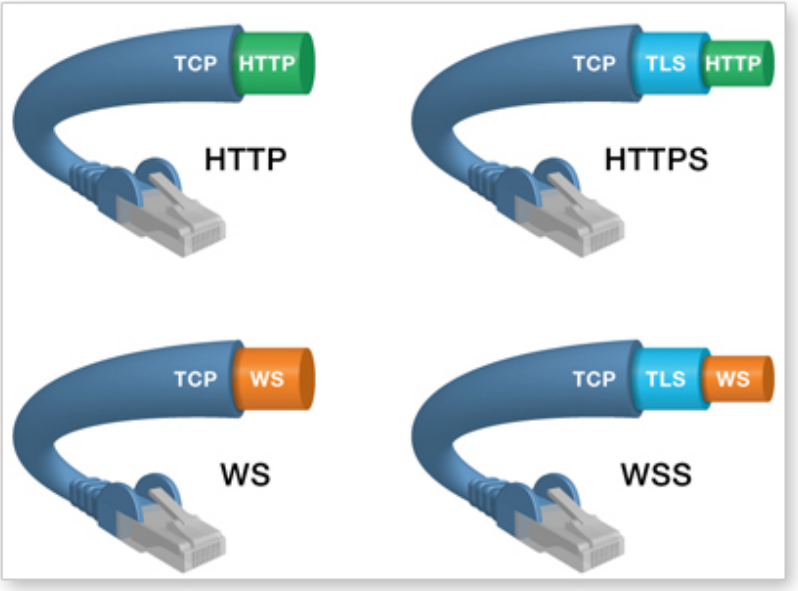
Использование HTTP-протокола как транспорт

- XML-RPC
- SOAP
- WebSocket



Протокол HTTP

Безопасность



Содержание

CGI, mod_perl, FastCGI, PSGI

Взаимодействие сервера и приложения

CGI (Common Gateway Interface) — это не язык, а лишь тип интерфейса

mod_perl - это надстройка над CGI для единоразовой компиляции скриптов

FastCGI - это развитие технологии CGI, запускается отдельным процессом и общается по сети с сервером

PSGI - Перловый суперклей для веб-фреймворков и веб-серверов

CGI, mod_perl, FastCGI, PSGI

CGI.pm

Первый большой модуль для работы с CGI интерфейсом

При помощи этого модуля можно сделать многое. Это комбайн.

Генерация HTML/XHTML

Поддержка старых браузеров IE 3.01

Очень много магии (174+ модуля/245KB кода)

Книга на 245 страниц

Он устарел. Он очень громоздкий.

121 открытый баг!

Однако он встречается!

CGI, mod_perl, FastCGI, PSGI

mod_perl

Самый частовстречающийся вариант

На него легко перенести ранее написанное CGI приложение

Самым большим превосходством является единоразовая компиляция приложения

Он тоже устарел.

CGI, mod_perl, FastCGI, PSGI

Пример

CGI (/perl/test.pl):

```
print "Content-type: text/html\r\n\r\n";
print "<h1>Hello world!</h1>";
```

Mod_perl:

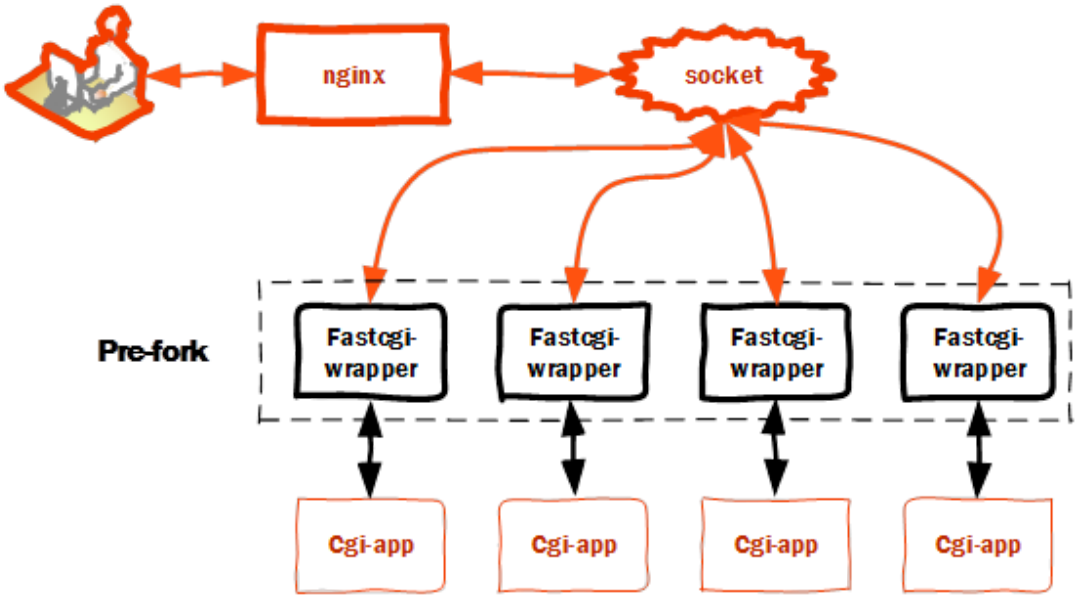
```
package Apache::ROOT::perl::test_2epl;
use Apache qw(exit);

sub handler {
    #line 1 test.pl
    my $r = shift;
    $r->content_type("text/html");
    $r->send_http_header;
    print "Hello world!";
}
```

CGI, mod_perl, FastCGI, PSGI

FastCGI

Современный вариант CGI программирования

Приложение запускается отдельно от сервера



CGI, mod_perl, FastCGI, PSGI

Пример

```
#!/usr/bin/perl

use strict;
use FCGI;

my $socket = FCGI::OpenSocket(":9000", 5);

my $request = FCGI::Request(
    \*STDIN, \*STDOUT, \*STDERR, \%ENV, $socket
);

my $count = 1;

while($request->Accept() >= 0) {
    print "Content-Type: text/html\r\n\r\n";
    print $count++;
}
```

CGI, mod_perl, FastCGI, PSGI

Plack

Простейшее приложение

```
my $app = sub {
    my $env = shift;
    return [
        200,
        ['Content-Type' => 'text/plain'],
        ["hello, world\n"]
    ];
};
```

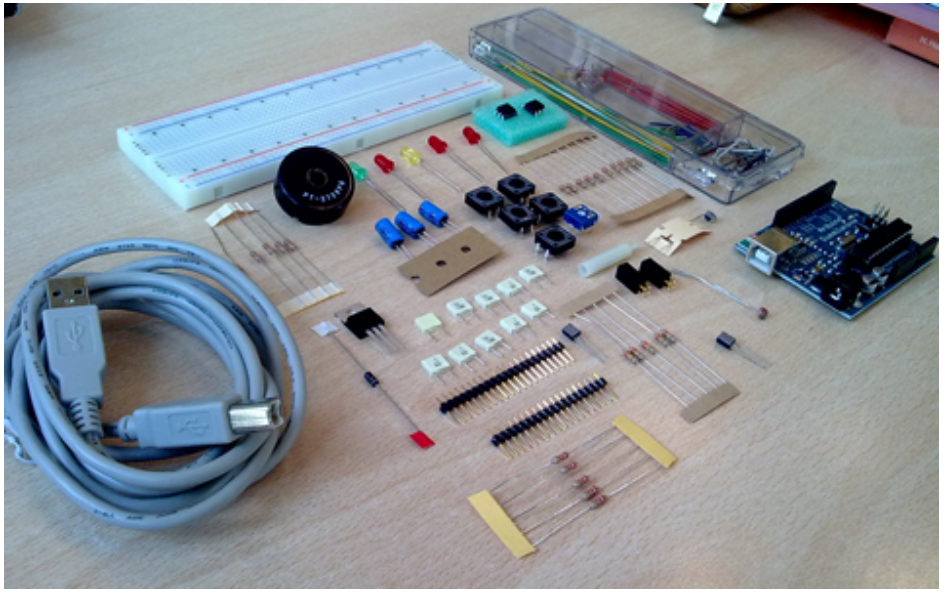
CGI, mod_perl, FastCGI, PSGI

```
use strict;
use Plack;
use Plack::Request;

my $app = sub {
    my $env = shift;
    my $req = Plack::Request->new($env);
    my $res = $req->new_response(200);

    $res->body('Hello World!');

    return $res->finalize();
};
```

Содержание

Веб-фреймворки

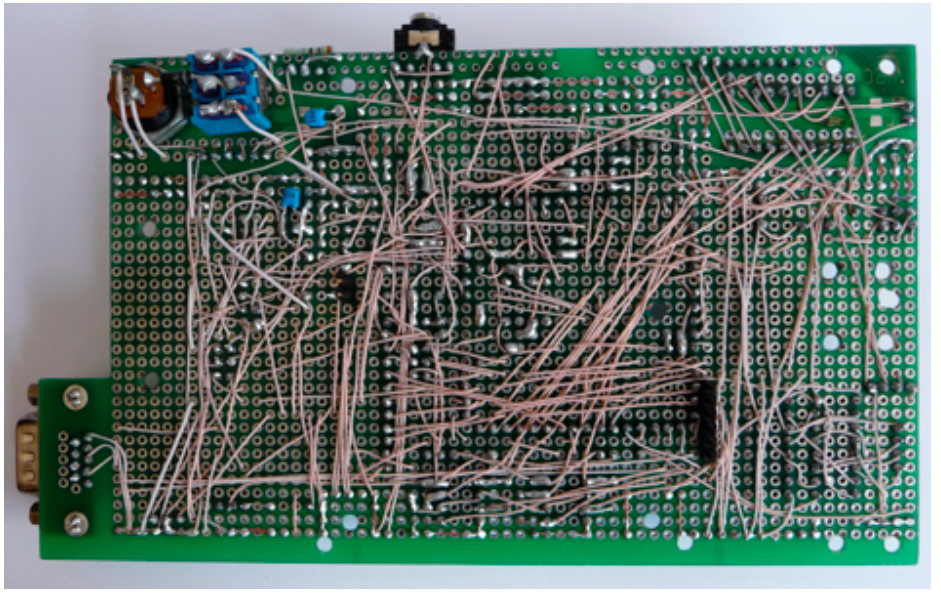
Веб-фреймворк vs CMS



Программная платформа, определяющая структуру программной системы

Веб-фреймворки

Веб-фреймворк vs CMS



Система управления содержимым

Веб-фреймворки

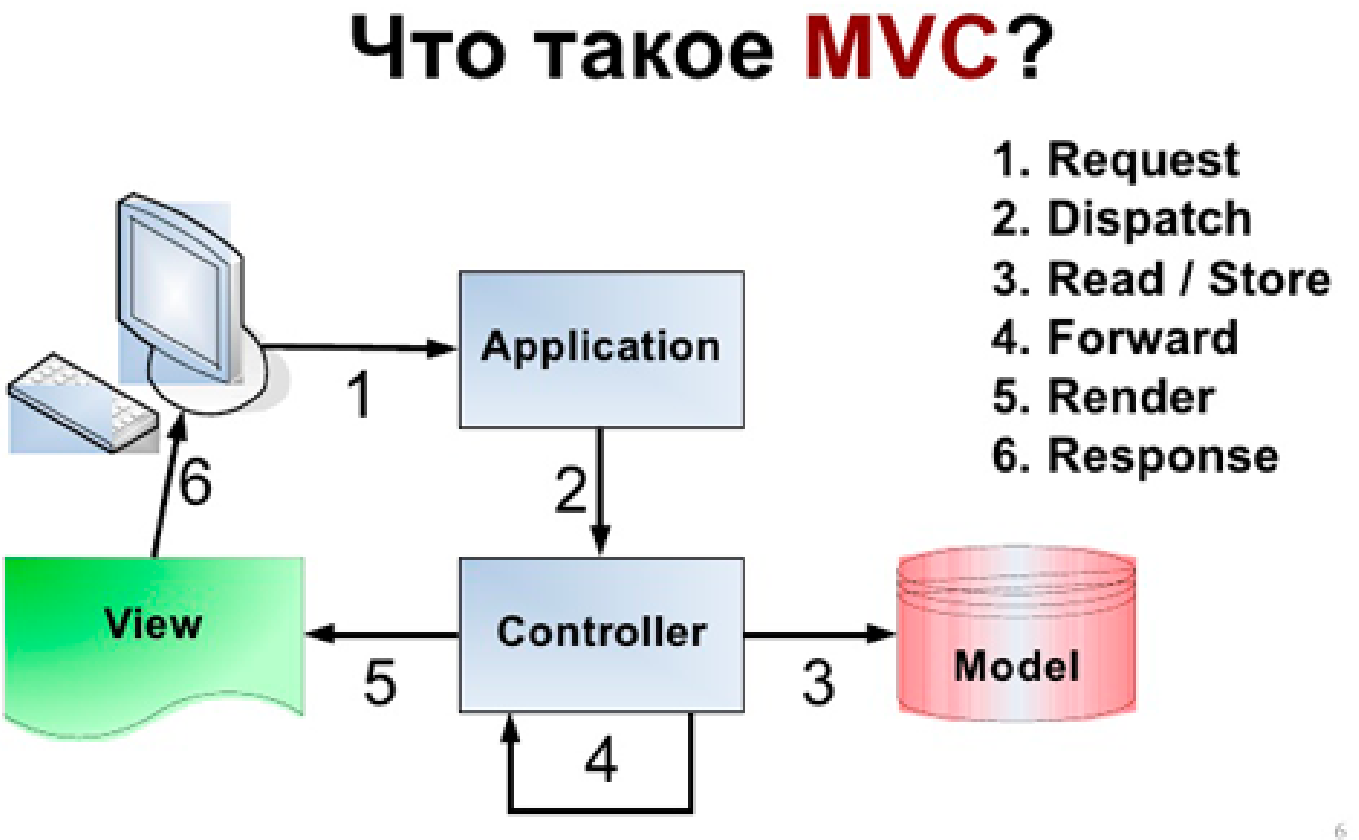
MVC (Модель-Представление-Контроллер)

Модель живёт сама по себе и содержит бизнеслогику

За формат результата отвечает представление

Контроллер обрабатывает запрос пользователя, занимается аутентификацией/авторизацией

Веб-фреймворки



Dancer

Установка:

```
%cpan Dancer2
...
...
F:\Strawberry\c\bin\dmake.exe install UNINST=1 -- O

%dancer2 -a sferamailapp
+ sferamailapp
+ sferamailapp\config.yml
+ sferamailapp\cpanfile
+ sferamailapp\Makefile.PL
+ sferamailapp\MANIFEST.SKIP
+ sferamailapp\bin
+ sferamailapp\bin\app.psgi
+ sferamailapp\environments
+ sferamailapp\environments\development.yml
+ sferamailapp\environments\production.yml
+ sferamailapp\lib
+ sferamailapp\lib\sferamailapp.pm
+ sferamailapp\public
```

Dancer

Запуск:

```
%plackup -p 5000 bin\app.psgi
HTTP::Server::PSGI: Accepting connections at http:/
```

Можно открывать браузер и заходить на http://127.0.0.1:5000

Так же генерируются файлы для запуска в режимах cgi и fcgi

sferamailapp\public\dispatch.cgi

sferamailapp\public\dispatch.fcgi

Dancer

Маршруты/Роутинг

```
get '/main/:page' => sub {
    return "Hello ".params->{page};
};
```

```
any ['get', 'post'] => '/main/*.*' => sub {
    my ($page, $ext) = splat;
    return pass if $ext = 'mp3';
    ...
};
```

```
get qr{ /main/([\w]+) }x => sub {
    my ($login) = splat;
    return "Hello $login";
};
```

Dancer

```
set views => path(dirname(__FILE__), 'templates');
set engines => {
    template => {
    template_toolkit => {
        extension => 'foo',
    },
    },
};
```

```
hook before_template_render => sub {
    ...
}
```

```
hook after_template_render => sub {
    # do something with $content
};
```

Dancer

```
hook before_layout_render => sub {
    ...
};
```

```
hook before => sub {
    if (!session('user') &&
        request->dispatch_path !~ m{^/login/}) {
        # Pass the original path requested along
        # to the handler:
        forward '/login/', {path => request->path};
    }
};
```

```
set session => 'YAML';
session varname => 'value';
session('varname');
$app->destroy_session;
```

Dancer

```
hook before => sub { if (!session('user') &&
    request->dispatch_path !~ /^\/login/) {
        forward '/login', {
          requested_path => request->dispatch_path
}}};

get '/secret' => sub { return "Top Secret" };

get '/login' => sub { template 'login', {
      path => param('requested_path')
}};

post '/login' => sub {
    if (param('user') eq 'bob' &&
        param('pass') eq 'letmein') {
      session user => param('user');
      redirect param('path') || '/';
    } else { redirect '/login?failed=1' }
};
```

Dancer

```
set serializer => 'JSON';

get '/user/:id/' => sub {
    { foo => 42,
      number => 100234,
      list => [qw(one two three)],
    }
};
```

```
post '/upload/:file' => sub {
    my $filename     = params->{file};
    my $uploadedFile = request->upload($filename);
};
```

Dancer

```
plugins:
 Database:
  driver: 'mysql'
  database: 'test'
  host: 'localhost'
  port: 3306
  username: 'myusername'
  password: 'mypassword'
  connection_check_threshold: 10
  dbi_params:
   RaiseError: 1
   AutoCommit: 1
   on_connect_do: ["SET NAMES 'utf8'"]
   log_queries: 1
   handle_class: 'My::Super::Database::Handle'
```

Dancer

```
use Dancer2::Plugin::Database;

my $user = database->quick_select(
    'users',
    {username => params->{user}}
);
```

DBIx::Class

```
use Dancer2;
use Dancer2::Plugin::DBIC qw(schema resultset);

get '/users/:user_id' => sub {
  my $user = schema('default')->resultset('User')
   ->find(param 'user_id');
  template user_profile => {
    user => $user
  };
};
```

Dancer

```
# param.t
use strict; use warnings;
use Test::More; use Plack::Test;
use MyApp;

my $test = Plack::Test->create( MyApp->to_app );
subtest 'An empty request' => sub {
  my $res = $test->request( GET '/' );
  ok( $res->is_success, 'Successful request' );
  is( $res->content, '{}', 'Empty response back' );
};

subtest 'Request with user' => sub {
  my $res = $test->request(GET '/?user=sawyer_x');
  ok($res->is_success, 'Successful request');
  is($res->content, '{"user":"sawyer_x"}', 'Rsp');
};

done_testing();
```

Mojolicious

```
%cpan Mojolicious
...
F:\Strawberry\c\bin\dmake.exe install UNINST=1 -- O

%mojo generate app SferaMailMojo

[mkdir] sfera_mail_mojo\script
[write] sfera_mail_mojo\script\sfera_mail_mojo
[chmod] sfera_mail_mojo\script\sfera_mail_mojo 744
[mkdir] sfera_mail_mojo\lib
[write] sfera_mail_mojo\lib\SferaMailMojo.pm
[mkdir] sfera_mail_mojo\lib\SferaMailMojo\Controlle
[write] sfera_mail_mojo\lib\SferaMailMojo\Controlle
[mkdir] sfera_mail_mojo\t
[write] sfera_mail_mojo\t\basic.t
[mkdir] sfera_mail_mojo\log
[mkdir] sfera_mail_mojo\public
[write] sfera_mail_mojo\public\index.html
[mkdir] sfera_mail_mojo\templates\layouts
[write] sfera_mail_mojo\templates\layouts\default.h
[mkdir] sfera_mail_mojo\templates\example
```

Mojolicious

Запуск

- В lib хранится код,
- В log логи
- В public статические файлы, которые раздаются отдельным сервером
- В script скрипт запуска сервера
- В t лежат тесты
- В templates шаблоны.

```
%perl script\sfera_mail_mojo daemon -m development
Server available at http://127.0.0.1:3000
```

Mojolicious

Mojo::Lite

```
%mojo generate lite_app SferaMailLiteApp
[write] SferaMailLiteApp
[chmod] SferaMailLiteApp 744
```

Запуск

```
%perl SferaMailLiteApp daemon
[Sat Apr 18 14:57:59 2015] [info] Listening at "htt
```

Mojolicious

Маршруты

```
my $r = Mojolicious::Routes::Route->new;
$r->get('/hello')->to(
    namespace  => 'SferaMailApp::Special',
    controller => 'foo',
    action     => 'hello'
);
$r->put('/hello')->to(controller => 'foo', action =
$r->any('/whatever')->to(controller => 'foo', actio
```

Плейсхолдеры

Route	/:path/test	/(:path)test	/#path/test	/*path/test
regexp	([^/.]+)	([^/.]+)	([^/]+)	(.+)
Good	/sfera/test	/sferatest	/sfera.23/test	/sfera/23/test
Wrong	/sfera.1/test	/sfera/test	/sfera/hello	/sfera/hi
Stash	path => 'sfera'	path => 'sfera'	path => 'sfera.23'	path => 'sfera/23'

Mojolicious

Шаблоны

```
sfera_mail_mojo\templates
```

```
sub welcome {
    my $self = shift;
    # Render response
    $self->stash(mymessage => 'Welcome');
    $self->render(text => 'Hello there.');
}
```

```
    $r->get('/foo/:user')
      ->to('foo#bar')
      ->name('baz');

    my $url = $c->url_for('baz');
```

Mojolicious

Сессии

```
my $sessions = Mojolicious::Sessions->new;
$sessions->cookie_name('myapp');
$sessions->default_expiration(86400);
```

```
my $auth = $r->under('/' => sub {
  my $c = shift;
  return 1 if $c->session->{is_auth};
  $c->render(text => "You're not authenticated.");
  return undef;
});
$auth->get('/admin')->to('admin#welcome');
```

Catalyst

```
%cpan Catalyst
...
F:\Strawberry\c\bin\dmake.exe install UNINST=1 -- O

%cpan -i Catalyst::Devel
...
```

```
% catalyst.pl SferaMail::Ctalyst
created "SferaMail-Ctalyst"
created "SferaMail-Ctalyst\script"
created "SferaMail-Ctalyst\lib"
created "SferaMail-Ctalyst\root"
created "SferaMail-Ctalyst\root\static"
created "SferaMail-Ctalyst\root\static\images"
created "SferaMail-Ctalyst\t"
created "SferaMail-Ctalyst\lib\SferaMail\Ctalyst"
created "SferaMail-Ctalyst\lib\SferaMail\Ctalyst\Mo
created "SferaMail-Ctalyst\lib\SferaMail\Ctalyst\Vi
created "SferaMail-Ctalyst\lib\SferaMail\Ctalyst\Co
created "SferaMail-Ctalyst\sferamail_ctalyst.conf"
```

Catalyst

Кодогенерация

```
%script\sferamail_ctalyst_create.pl controller Stud
%script\sferamail_ctalyst_create.pl view Stud
```

Заготовленные рецепты

```
requires 'Catalyst::Plugin::Authentication';
requires 'Catalyst::Plugin::Session';
requires 'Catalyst::Plugin::Session::Store::File';
requires 'Catalyst::Plugin::Session::State::Cookie'
# Configure SimpleDB Authentication
__PACKAGE__->config(
  'Plugin::Authentication' => {
    default => {
      class           => 'SimpleDB',
      user_model      => 'DB::User',
      password_type   => 'clear',
}});
```

Catalyst

DBIx::Class внутри

```
$ script/myapp_create.pl model DB DBIC::Schema \
SferaMail::Catalyst::Schema create=static \
components=TimeStamp,PassphraseColumn \
dbi:SQLite:sferamail.db \
on_connect_do="PRAGMA foreign_keys = ON"
```

Аутентификация

```
requires 'Catalyst::Plugin::Authentication';
$c->authenticate({
  username => $username,
  password => $password
})
```

```
$ DBIC_TRACE=1 perl -Ilib set_hashed_passwords.pl
```

Catalyst

Авторизация

```
requires 'Catalyst::Plugin::Authorization::Roles';
if ($c->check_user_roles('admin')) {
    ...
}
```

Содержание

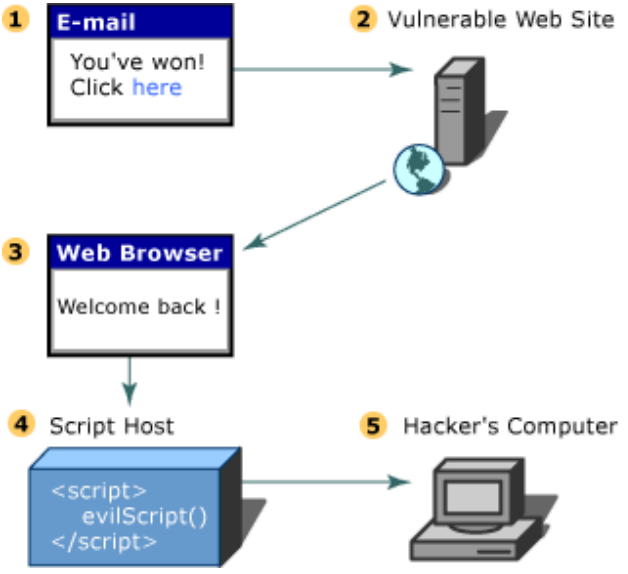
1. Протокол HTTP
2. CGI, mod_perl, FastCGI, PSGI
3. Веб-фреймворки
4. *Механизмы безопасности в приложениях*

Безопасности в приложениях

- XSS
- CSRF
- Response-splitting
- Innumeration
- Sql-injection
- Code-including
- .....

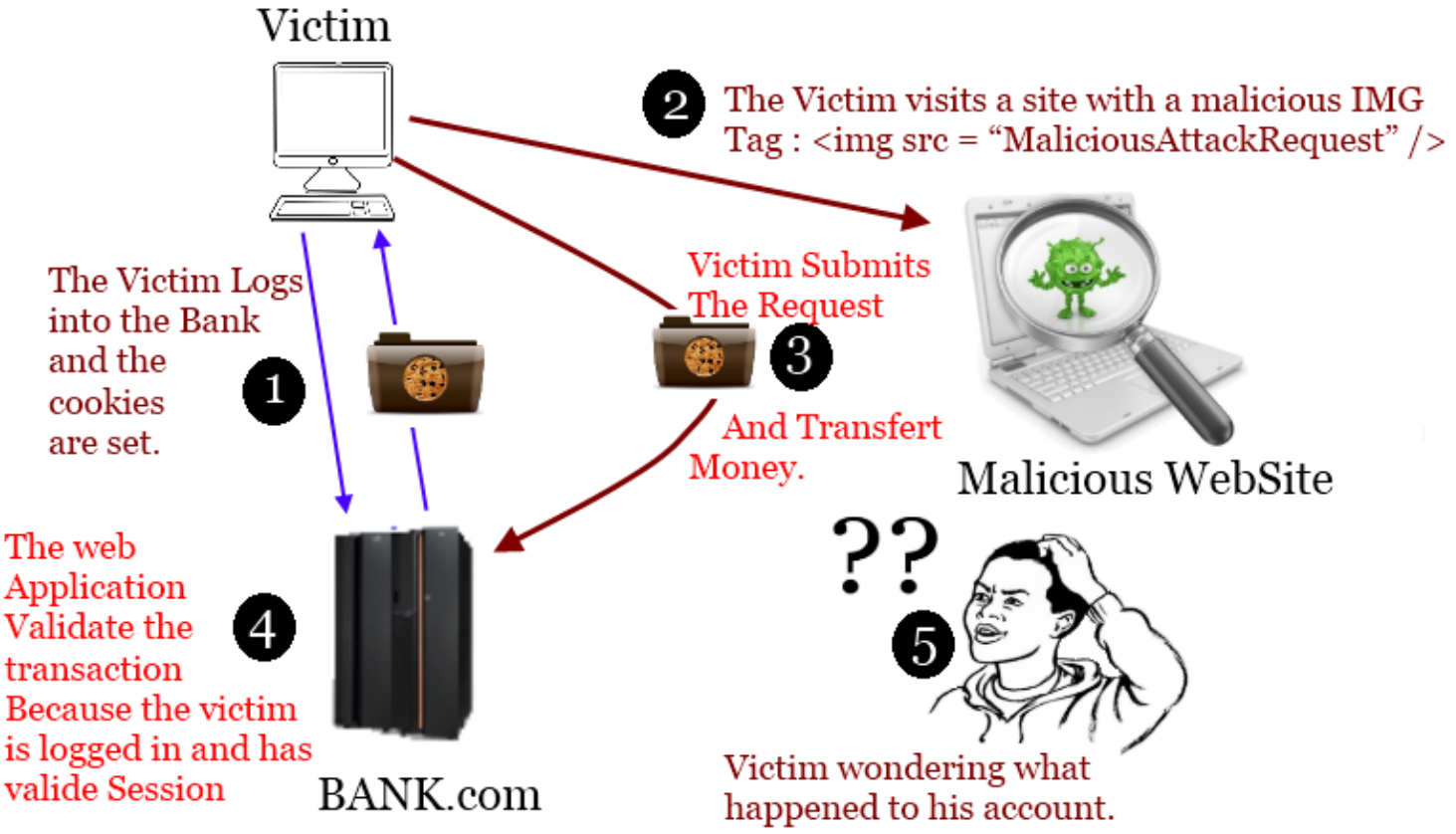
Безопасности в приложениях

XSS - Cross-site scripting



Безопасности в приложениях

CSRF - Межсайтовая фальсификация запросов



Безопасности в приложениях

Response-splitting

```
HTTP/1.1 200 OK
Date: Thu, 13 Mar 2014 18:08:14 GMT
Content-Language: pl

Jestem w body
Connection: close
Content-Type: text/html; charset=UTF-8

Hej!
```

Безопасности в приложениях

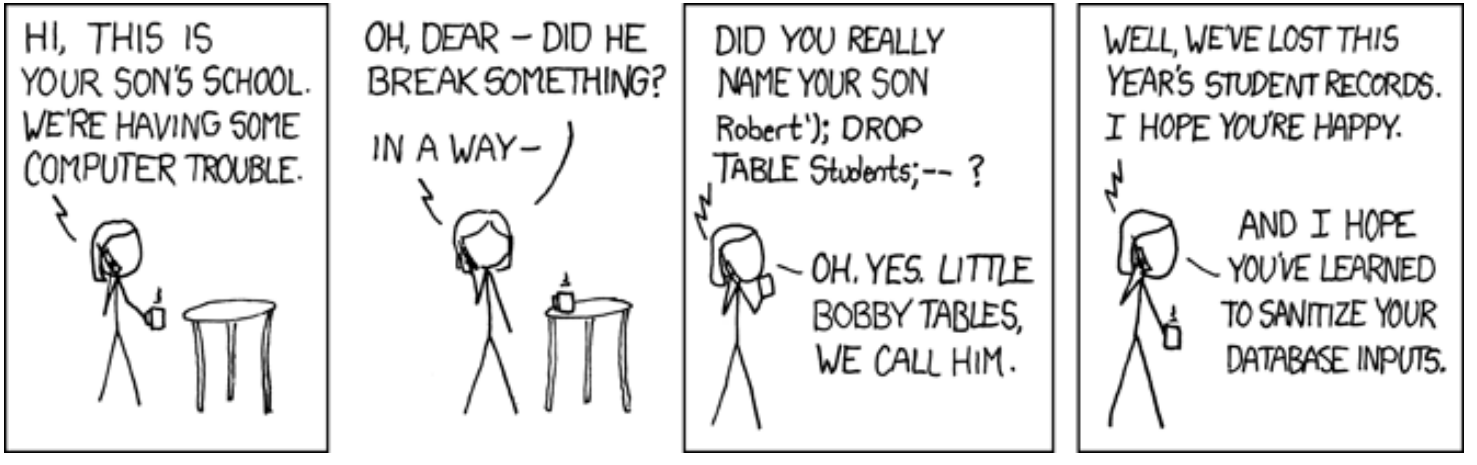
Innumeration

- Получение результатов заказа по id
- Получение данных пользователя по id

Перебор id в запросах с целью получения личных данных

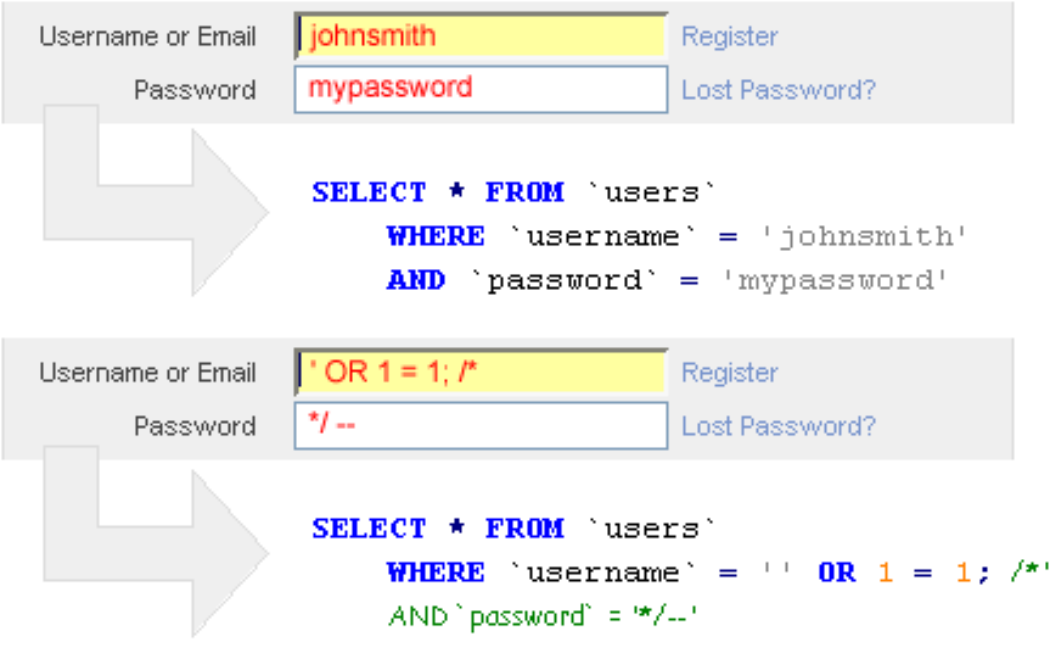
Безопасности в приложениях

Sql-injection



Безопасности в приложениях

Sql-injection



Безопасности в приложениях

Remote Code Execution

```
eval $str;

open(my $fh, $str);

system($str);
```

Домашнее задание

Создать WEB-приложение с 3 разделами, пользовательским, административным и XML-RPC.

- Пользовательский раздел:
  - страница регистрации (Nick, Пороль, ФИО, ссылка на проект)
  - страница авторизации
  - страница, редактирования личных данных, с возможностью удаления аккаунта.
  - получение токена для XML-RPC
- XML-RPC
  - аутентификация через Basic-auth, вместо пароля используется токен
  - удалённый вызов процедур калькулятора
  - рейт-лимиты вызовов процедуры для одного пользователя
- Административный раздел:
  - страница просмотра/редактирования рейт-лимитов
  - страница просмотра зарегичстрированных пользователей
  - страница удаления пользователя