
open
apereo 2013

A node.js module for CAS validation
Apereo 2013

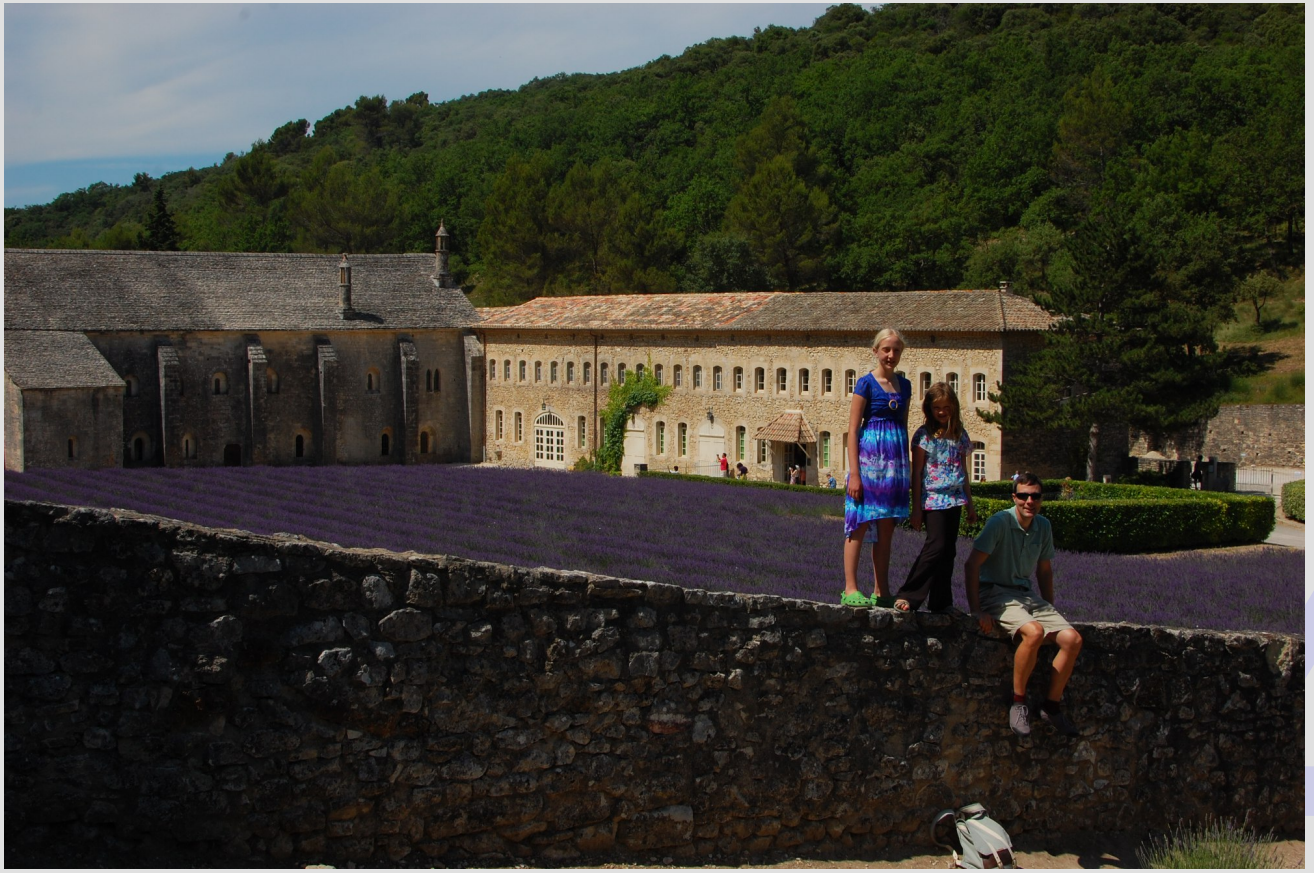
James E. Marca

June 5, 2013

Hello, my name is James

- Transportation engineering PhD
- Research scientist with UCI ITS
- https://github.com/jmarca
- http://contourline.wordpress.com
- james@activimetrics.com

Me, traveling!



CAS

I don't know anything about CAS internals.
This talk is about *using* CAS

Use case: CAS + ldap

- Used to use Drupal
- Drupal was set up to use ldap and CAS
- So we used ldap and CAS

Single sign on, single sign off

- Caltrans sponsors liked the website, but
- They didn't like signing in again to project sites
- Need single sign on, sign off

I use node.js

- JavaScript on the server.
- Fast because V8 is fast
- Clean, single-threaded non-blocking design
- What's not to like?

CAS support in node.js

- The node.js packaging system is npm.
- `npm search cas`: lots of options
- but none supported single sign out

CAS Validate

- repository: https://github.com/jmarca/cas_validate
- installation: npm install cas_validate
- a plugin to **Express** and **Connect**

Node.js idioms

(A brief digression)

Non-blocking by design

- file i/o
- web operations
- db access
- etc

Async needs callbacks

```
fs.readFile(filename, [encoding], [callback])
```

Asynchronously reads the entire contents of a file:

```
fs.readFile('/etc/passwd', function (err, data) {
  if (err) throw err;
  console.log(data);
});
```

function(err, data)

The callback is passed two arguments `(err, data)`. Check for errors, then handle `data`.

callback is the last argument

example from node-redis code

```
RedisClient.prototype.send_command =
        function (command, args, callback) {
    if (Array.isArray(args)) {
        if (typeof callback === "function") {
        } else if (! callback) {
            last_arg_type = typeof args[args.length-1];
            if (last_arg_type === "function"
                || last_arg_type === "undefined") {
                callback = args.pop();
            }
        } else { throw new Error("...");}
    } else { throw new Error("..."); }
}
```

Lots of work to pull the last argument of an optional list of arguments as the callback

Closures are used

```
var fs = require('fs')
function doSomething(done){
    function callback(err, data) {
        if (err) throw err
        return done(null,data)
    })
    fs.readFile('hpms/data.txt','utf8',
                callback)
}
```

Function `callback`, being declared inside function `doSomething` can "see" `done`, even after it is passed to `fs.readFile`

End of digression

Except:

- semicolons are optional
- comma first notation is popular

Notes on code examples

Note that the code examples in the following slides are **not** exactly what is in the current version of cas_validate. Rather they are simplified versions, omitting error checks and other details for clarity of presentation.

Program requirements

- Express is route based
- public routes:
  - check if logged in,
  - but don't *require* a login.
- restricted routes:
  - require a login,
  - check permissions, etc

Single Sign On/Off

- A login here=login everywhere
  - and vice versa
- A logout here=logout everywhere
  - and vice versa

CAS documentation is excellent

Thanks

Basic login task

1. establish a session with the client
2. ask the client to redirect to the CAS server
3. expect a reply back from the CAS server with a ticket
4. check the ticket's validity directly with the CAS server

Establish a session with a client

- use standard connect/express middleware

```
var express = require('express')
var RedisStore =
     require('connect-redis')(express);
var app = express()
app
 .use(express.logger({buffer:5000}))
 .use(express.bodyParser())
 .use(express.cookieParser('tall barley at Waterloo'))
 .use(express.session({ store: new RedisStore }))
```

Redirect to CAS server

- more complicated, but still easy

Prerequisites:

```
var querystring = require('querystring')
// configurable
var cas_host = 'https://my.cas.host'
var login_service = '/cas/login'
```

Redirect Code

```
var redirecter = function (req,res,next){
    // decide endpoint where CAS server
    // will return to
    var service = determine_service(req)
    var queryopts = {'service':service}
    res.writeHead(307,
        { 'location': cas_host+login_service +'?'
                +querystring.stringify(queryopts)
        })
    return res.end()
}
```

Listen for CAS reply

- parse incoming query for `ticket` parameter

```
function ticket_check(req,res,next){
    var url = parseUrl(req.url,true);
    if( url.query === undefined
       || url.query.ticket === undefined){
        logger.debug('moving along, no ticket');
        return next(); // next route
    }
    logger.debug('have ticket')
    ...
```

Check ticket validity

The request MUST include a valid service ticket, passed as the HTTP request parameter, “ticket”.

- ☐ Directly connect with CAS server to check validity of ticket
- ☐ Using the Request library

Code to connect to CAS server

```
function ticket_check(req,res,next){
    ...
    var ticket = url.query.ticket;
    var service = opt_service ?
        opt_service : determine_service(req)
    var cas_uri =
       cas_host+validation_service +'?'
       +querystring.stringify({'service':service,
                               'ticket':ticket});
    request({uri:cas_uri}, callback);
    return null;
});
```

Define the request callback

- ☐ current version uses a regular expression
- ☐ development version actually parses the XML
- ☐ callback is a defined in ticket_check, can see `ticket` and `next`
- ☐ Redis: use Redis to link CAS session ticket and local session

Prerequisites:

```
var redis = require("redis")
var redclient = redis.createClient();
redclient.on("error", function (err) {
    logger.error("Redis Client Error: " + err);
});
```

Code to handle response from CAS

```
function callback(err, resp, body) {
    if (!err && resp.statusCode === 200) {
        if(/cas:authenticationSuccess/.exec(body)){
            if(/<cas:user>(\w+)<\/cas:user>/.exec(body)){
                req.session.name = RegExp.$1;
            }
            req.session.st = ticket;
            redclient.set(ticket,req.sessionID);
            return next();
        }else{  next(new Error('authentication failed')); }
    }else{ next(new Error('authentication failed')); }
}
```

That's it

- ☐ We redirected to CAS
- ☐ consumed the service ticket
- ☐ verified the service ticket
- ☐ established a session
- ☐ stored the session in Redis with the ticket as the key

Example server

```
var app = express()
.use(express.cookieParser('barley Waterloo Napoleon'))
.use(express.session({ store: new RedisStore }))
.use(cas_validate.ticket({'cas_host':cas_host}))
.use(cas_validate.check_or_redirect({'cas_host':cas_host}))
.use(function(req, res, next){
        res.end('hello world')
        return null
    });
var server =app.listen(testport,testhost,done)
```

Single Sign Off

- very easy
- listen for a `POST` from the CAS server
- invalidate the session

SSOff code

```
function ssoff(req,res,next){
    var method = req.method.toLowerCase();
    if (method == 'post'){
        return invalidate(req,res,next);
    }
    return next()
}
```

And the invalidate function

```
function invalidate(req,res,next){
      var st = get_session_ticket(req)
      redclient.get(st,function(err,sid){
          req.sessionStore.destroy(sid
          ,function(err){
              redclient.del(st);
              return null
          }});
      res.end();
      return null;
  }
}
```

Demo

Sign on, sign off

CTMLabs banner

- Using this library to create a dynamic banner for our sites
- Researchers need only write (all on one line, of course)

```
<script src='http://menu.ctmlabs.net/menu.js?resetgateway=true
            &service=http%3a%2f%2fara.ctmlabs.net%2findex.html'>
</script>
```

http://ara.ctmlabs.net

Thank you

Questions? Comments? Rotten tomatoes?