

Production Debugging in AWS ElasticBeanstalk

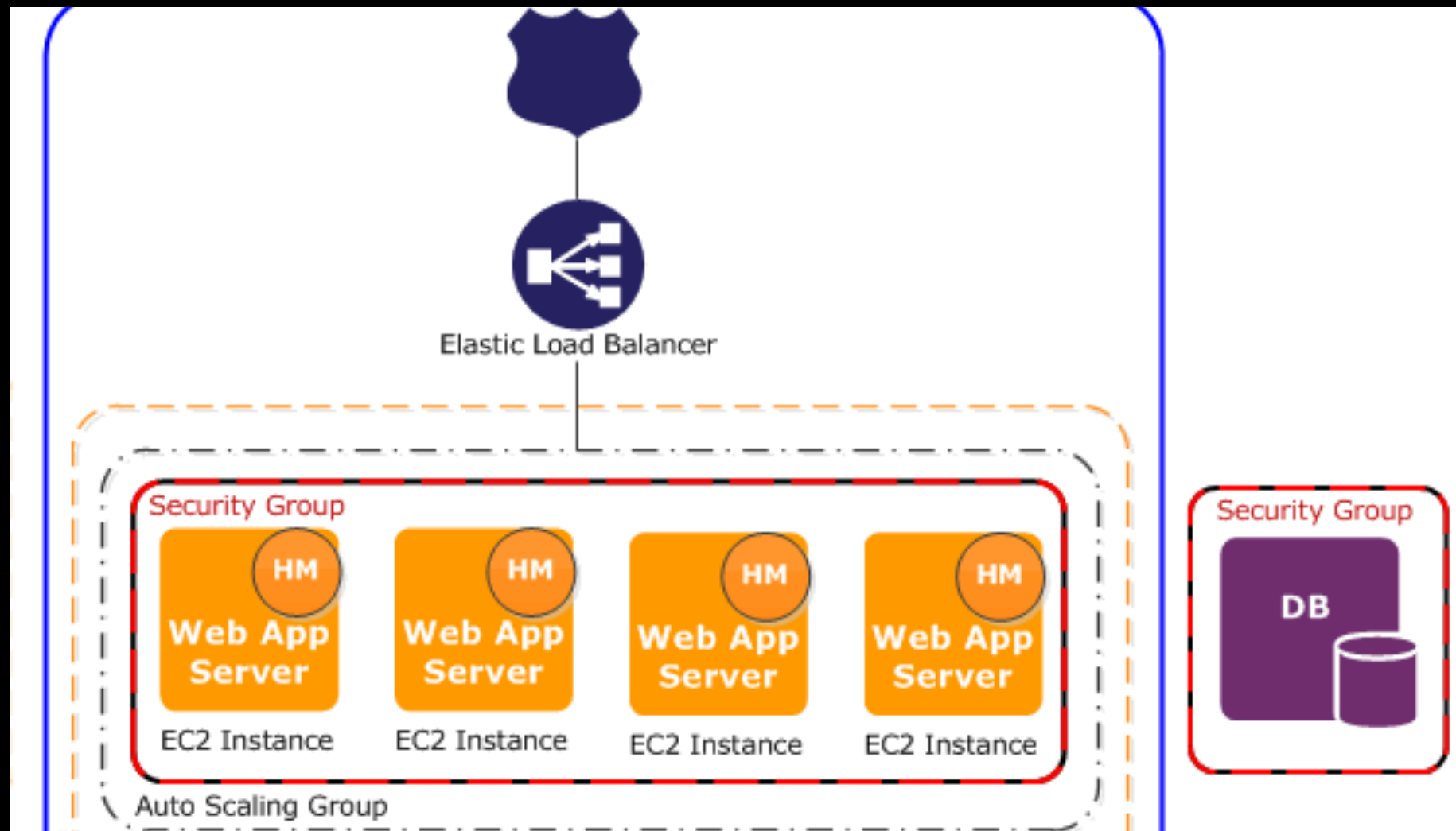
Mark Campbell

Team Lead at SERMO

Talk Goals

- Touch on some AWS technologies
- Operation of a system versus development
- Touch on web and rails debugging

AWS ElasticBeanstalk Architecture



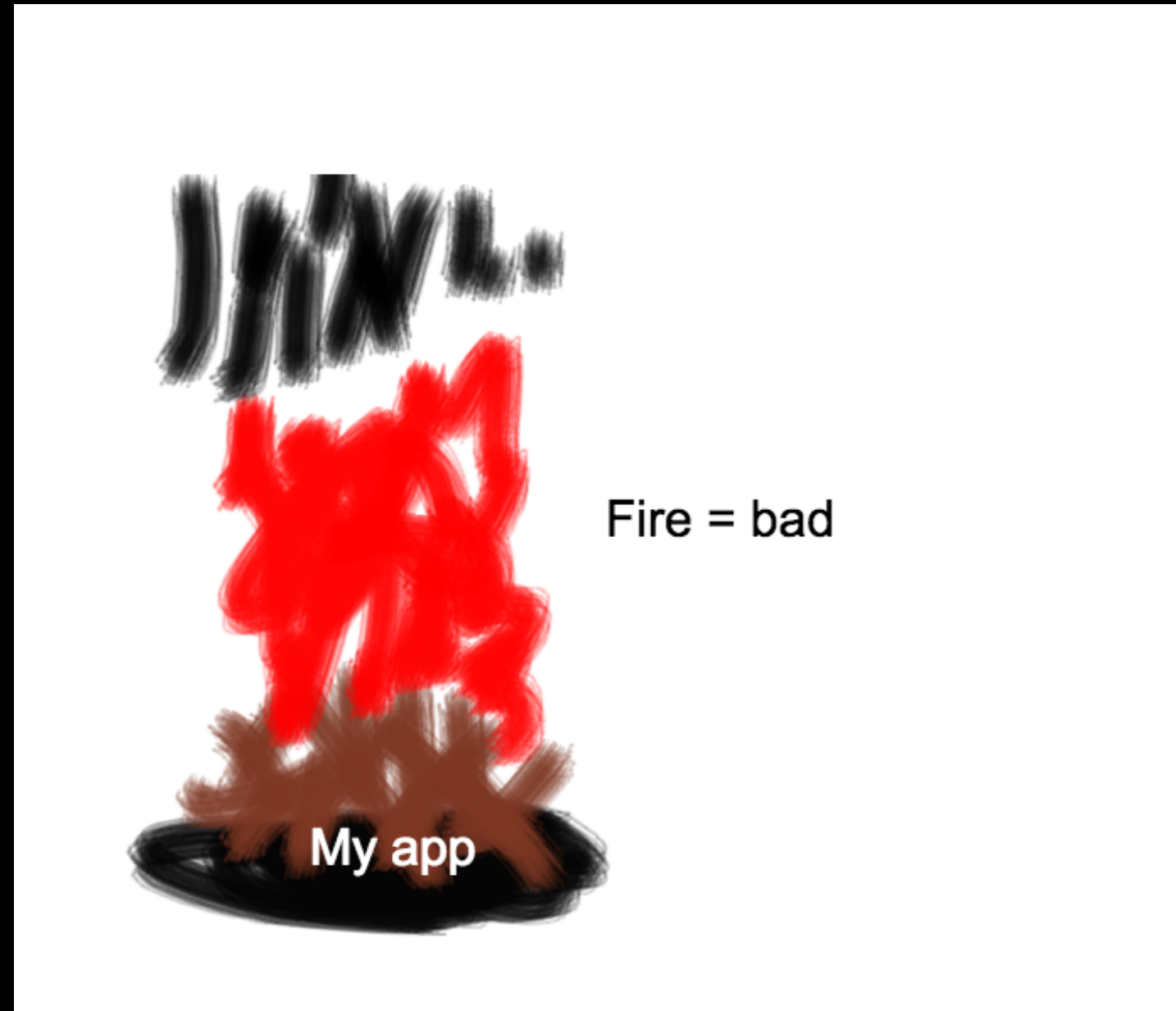
Application Design

- AWS ElasticBeanstalk, i.e. Heroku except less expensive
- Two apps
- JSON API secured by HTTPS and HTTP basic authentication

'Simple' problem

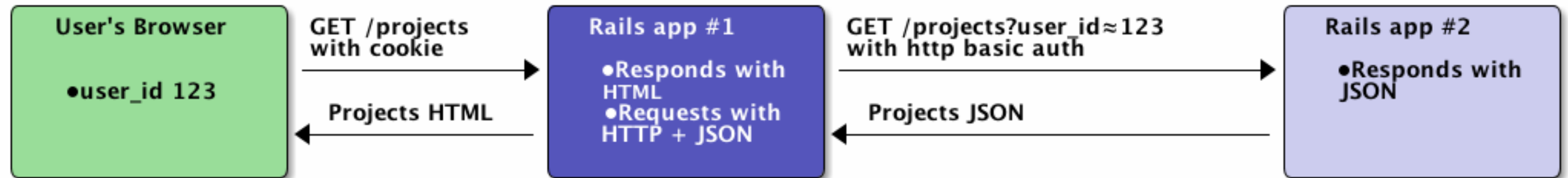
Smoke testing the app before releasing it to users resulted in a lot of smoke.

Lit AF is somehow not good



Request flow

Mental model is critical for debugging!

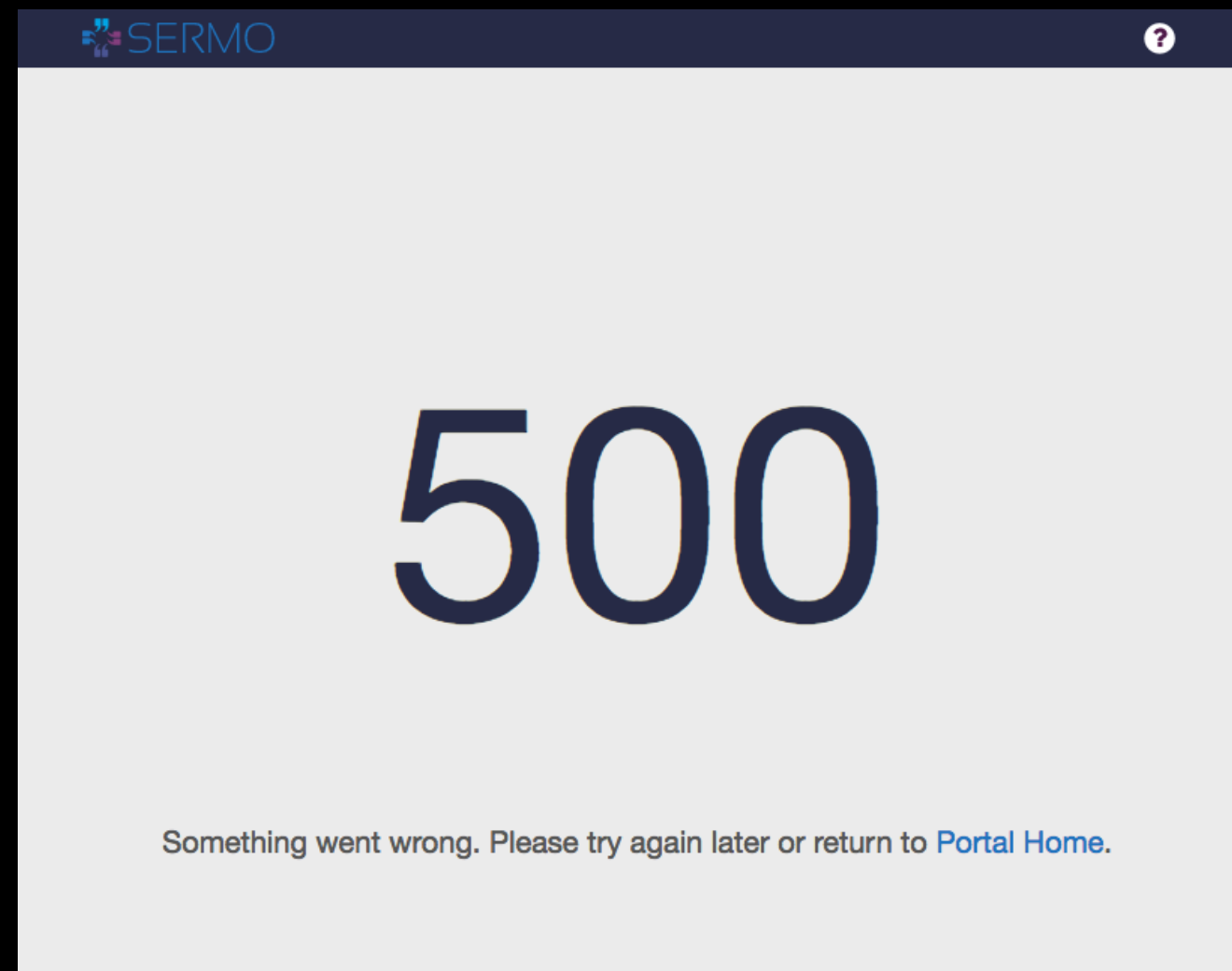


Common Tools

- SSH
- Rails and nginx logs
- Rails console
- curl

Symptoms

Log in. Go to projects. User see '500' page:



Debugging

- Exception tracker
- Logs

Exception tracker

Caveat: some classes of exceptions are completely ignored.

Exception tracker clients (gem `sentry-raven`) have default lists.

Logs

Caveat: logs don't have all data you could want.

Rule: logs will always lack the specific data you need at that point in time.

Example: *usernames and passwords*

Log Locations

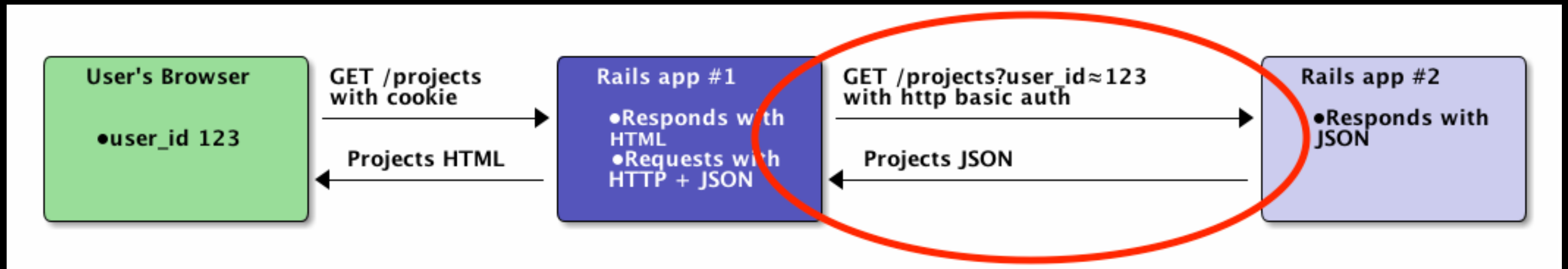
AWS ElasticBeanstalk default 'interesting' log locations:

- Rails: `/var/app/current/log`
- NGINX: `/var/log/nginx`
- Puma: `/var/log/puma`

Exception that was raised

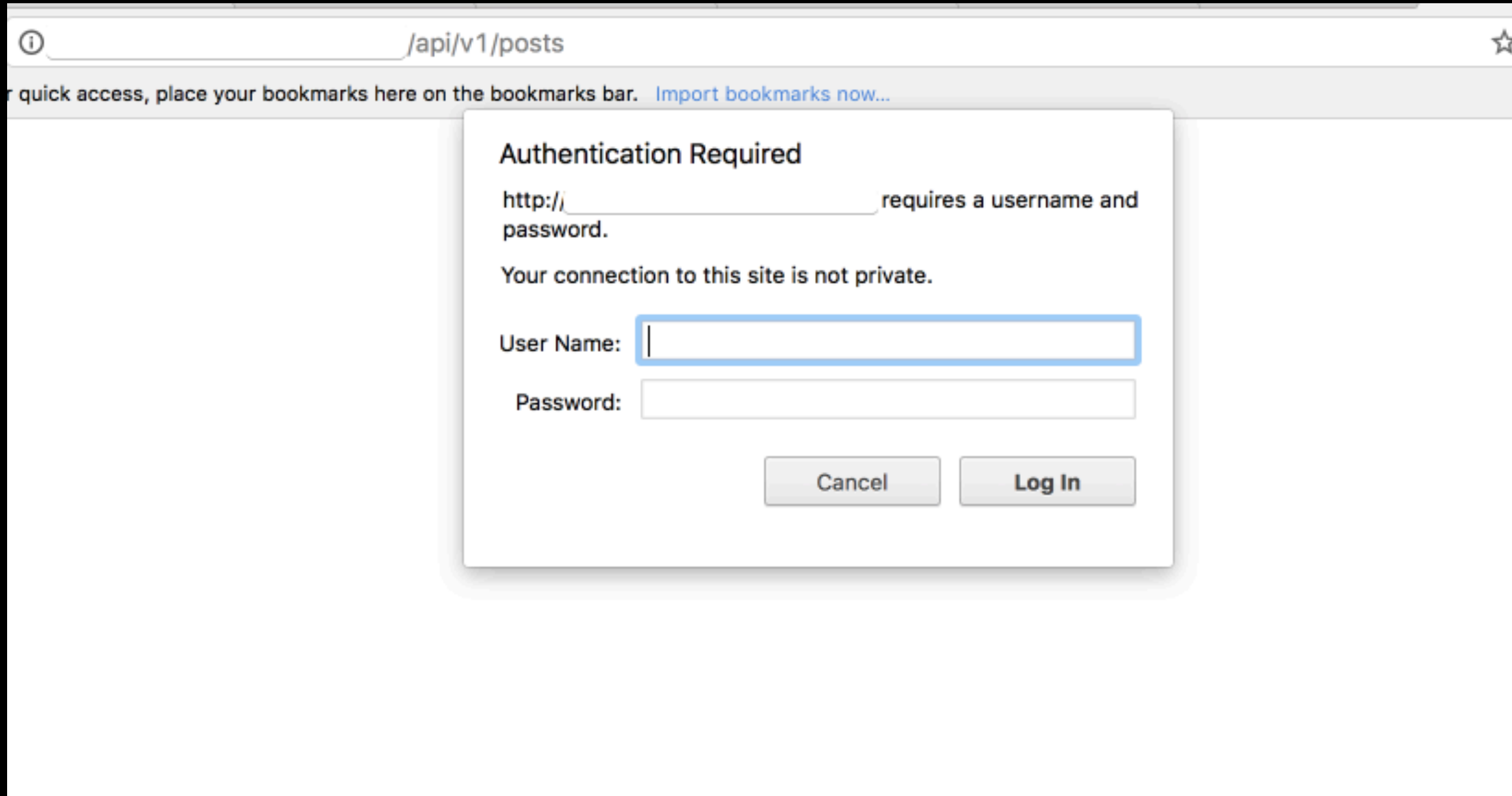
Class: `JsonApiClient::Errors::NotAuthorized`

Where from: `json_api_client` gem.



HTTP 401
Unauthorized

HTTP Basic Authentication



HTTP Basic Authentication (cont.)

Username and password are base64 encoded like this:

```
Base64.encode64(username + ':' + password)
```

In JavaScript: `btoa(username + ':' + password)`

HTTP Basic Authentication (cont.)

Username: Aladdin

Password: OpenSesame

HTTP header generated:

Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW11

Test with alternative methods!

Using `curl` to test makes reproducing this much easier to reproduce.

See `-u`, `--user <user:password>` in `man curl`

curling works, wtf?

Investigate other server

Dead end. Logs in Rails and NGINX don't have full credentials logged by default.

We also already know that it works!

Investigate the header

How do we set the header in our code? Like this:

```
connection_options[:headers] = {  
  'Authorization' => "Basic #{Base64.encode64(API_KEY + ':' + PASSWORD)}"  
}
```

Spot the bug

```
irb(main):003:0> Base64.encode64('mysecretusername:TX1TZWNyZXRVc2VybmFt...')  
  
=> "TEZ40VJvSi9tc3R3V3BodXpUanNkTFF1TURheFJvcmtBTExCYVd0TUZSeFFw\nU1JSR3NUVW  
55WGZwTENqYnFqVzpUWGxUW1d0eVpYU1ZjM1Z5Ym1GdFpUcE1S\nbmc1VW05S0wyMXpkSGRYY0do  
MWVsUnFjM1JNVVdWT1JHRjRVbT15YTBGTVRF\nSmhWMDVOUmxKNFVYQ1NVbEpIYzFSVmJubFlabk  
JNUTJwaWNXcFg=\n"
```

Realization

Mark Campbell huh

Mark Campbell one sec

Mark Campbell is `http` `basic` base64 encoded auth supposed to have newlines in it?

 no

Fix

```
Base64.strict_encode64( 'mysecretusername:TX1TZWNyZXRVc2VybmFt...' )
```

Smoke test!

Checklists

Operation of a system is very complex.

Without a checklist, you're going to miss something.

New systems, feature, and bugs should have checklists!

Lessons learned

- Checklist before system is operational
- Smoke tests are great
- Reproduce the error as simply as possible
- Mental models are critical

Questions?