

Overview

Heroku app to handle long running request: importing ticket audits.

Please see [US258](#) for more information.

Running the app:

Locally:

Run the following command locally and open your browser to <http://localhost:5000/canvas>

```
. setupConfigVars.sh
```

then run the following:

```
heroku local web
```

To run the worker

```
heroku local worker
```

Heroku:

<https://ticket-scale-import.herokuapp.com/canvas>

Visualforce (using signed request)

/apex/TEST__TicketScaleImport

Connected App

[Connected App 1](#)

[Connected App 2](#)

Debugging the app

run node-inspector in a separate browser window

then run heroku local webdebug

Debugging Unit Tests

To debug unit tests, the same steps apply but the actual NODE code to be called (instead of that from the proc) is actually from grunt

As always, make sure that node-inspector is installed / running

```
npm install -g node-inspector
```

then

```
node-inspector  
# in some new tab
```

ex:

```
node --debug-brk node_modules/grunt/bin/grunt testOnly
```

thats it.

Running Unit Tests

To run unit tests, run the following:

```
grunt test  
# please note that this calls jshint, jscs, ejslint and mochaTest
```

or from watch

```
grunt watch:test
```

to JUST run unit tests

```
grunt testOnly
```

Writing Unit Tests

There are three main things to keep in mind for writing unit tests:

- assertions (this is done through chai)
- spies (whether the method was called)
- stubs / mocks - when the code calls X, don't call anything and return 42
 - (please note that stubbing web requests uses `nock`)

Assertions

Always use chai assertions like follows:

```
var chai = require( 'chai' );  
var assert = chai.assert;
```

then make the request within the code

```
assert.isOk('everything', 'everything is ok');  
assert.isOk(false, 'this will fail');
```

for more information on asserts, please see: [Chai](#)

Spies

Testing this function can be quite elegantly achieved with a test spy:

```
it('calls the original function', function () {  
  var callback = sinon.spy();  
  var proxy = once(callback);  
  
  proxy();  
  
  assert(callback.called);  
});
```

The fact that the function was only called once is important:

```
it('calls the original function only once', function () {
  var callback = sinon.spy();
  var proxy = once(callback);

  proxy();
  proxy();

  assert(callback.calledOnce);
  // ...or:
  // assert.equals(callback.callCount, 1);
});
```

We also care about the this value and arguments:

```
it('calls original function with right this and args', function () {
  var callback = sinon.spy();
  var proxy = once(callback);
  var obj = {};

  proxy.call(obj, 1, 2, 3);

  assert(callback.calledOn(obj));
  assert(callback.calledWith(1, 2, 3));
});
```

For more information, please see [Sinon](#)

Stubs / Mocks

The function returned by once should return whatever the original function returns. To test this, we create a stub:

```
it("returns the return value from the original function", function () {
  var callback = sinon.stub().returns(42);
  var proxy = once(callback);

  assert.equals(proxy(), 42);
});
```

Conveniently, stubs can also be used as spies, e.g. we can query them for their callCount, received args and more.

Learn more about stubs.

For more information, please see:

- [sinon#Stubs](#)
- [mock-express-request](#)
- [mock-express-response](#)

Nock (network mocks)

To do so, comment out any nock code within the test and include the following:

```
var nock:any = require('nock');
nock.recorder.rec();
```

When you then run your test again, any calls will be echoed out to the console with the exact nock code to intercept it and provide a stub response.

For more information, please see [Nock](#)

Fixing permissions

Sometimes people have different umask settings on their machines, meaning files generated appear modified when instead they just have different permissions.

Navigate to the base of the repo and run this shellscript:

```
git config --global --add alias.permission-reset '!git diff -p -R --no-color | grep -E "^(diff|(old|new) mode)" --color=never | git apply'
```

Now, you can run this git command

```
git permission-reset
```

All files are reset back to the permission known within git.

Common Links

For more information about using Node.js on Heroku, see these Dev Center articles:

- [Getting Started with Node.js on Heroku](#)
- [Heroku Node.js Support](#)

- [Node.js on Heroku](#)
- [Best Practices for Node.js Development](#)
- [Using WebSockets on Heroku with Node.js](#)
- [Platform connect](#)
- [JSForce](#)
- [Lightning Design System - LDS](#)

how was this made

- [nodejs buildpack](#)
- [canvas developer guide](#)
- [made a canvas app](#)
- [authorize the canvas app for users/profiles/perm sets](#)
- [expose out the canvas app in vf](#)
- [demo on dev0](#)

Example canvas tag in vf

```
<apex:canvasApp applicationName="TEST Canvas Demo"
  namespacePrefix="" onCanvasAppLoad="onCanvasLoad"
  height="600px" width="100%"
  parameters="{ eventId: '{! EventDateTime__c.Event__c }', eventDateTimeId: '{! Event
DateTime__c.Id }', eventName: '{! JSENCODE( EventDateTime__c.Event__r.EventName__c ) }
' }"
/>
```