CAL

In this lab, you will edit the Kraken app created by Altus to add CAL logging.

# Objectives

In this lab, you will learn to

* Add a CAL Transaction around the account Business Logic
* Add a CAL Error Event if the Business Logic fails.

We will edit the /controllers/account.js to surround the service calls which extract the users account activity. The business logic involves multiple service calls that run in parallel using the async.parallel() method call. Embed this method call in the CAL Transaction callback. In addition, if the async method detects an error, it returns the error as the first parameter in the async callback. If the error exists, send a CAL Event message indicating an error.

Also, if the login code detects an invalid login, report this using a CAL Event message.

# Set things up

Ensure that the Altus program created the Kraken app and that it builds successfully and runs on the staging server.

In multiple browser tabs:

1. Open Altus and select the application we are using.
2. Open the Fusion CI in a tab.
3. Open the application in a tab.
4. Open the CAL View log in a tab.

Use the CAL link below and search the page for your application name:

<http://mscal.qa.paypal.com/cgi/logview>

Start the application and login to see the account information. View the CAL messages in the browser.

This verifies that the application is working correctly.

# Add a CAL Transaction message

## Edit the account controller program

In github, edit the /controller/account.js file. At the top of the file, require the ‘cal’ module.

var cal = require(‘cal’);

In the exports section, surround the async.parallel() method with the CAL Transaction.

var transaction = cal.createTransaction('TRAN-name', 'TRAN-type');

transaction.addData('tranData','trandata');

transaction.flush(); // begin the transaction

async.parallel(…., callback(err, results) {

:

if (err) {

…

var event = cal.createEvent(‘name’,’type’);

event.status = cal.Status.ERROR;

event.addData( ‘msg’, ‘Error reading account data.’);

event.complete();

req.log('error', err);

transaction.status = cal.Status.ERROR;

transaction.complete();

next(err);

return;

}

…

transaction.complete(); // end of transaction

res.render('account', model);

});

};

## Commit the changes in github

At the bottom of the screen, enter a commit message and commit the changes.

## Build the app

Go to the Jenkins web interface and build the application again. Wait for CI to complete.

## Deploy to the staging server

Go to the Altus “Staging Server” screen for the application. Select the app and examine the “Manifests” table. The top entry should correspond to the latest build we just completed. Click the “Deploy” button, edit whatever changes you require (verify the email address). Continue when deployment completes.

## Start the application

Refresh the application browser screen.

## Login to see the account data

In the application screen, log in to see the account data. The account information should be the same.

## View the CAL log information

Open the CAL View Log and find your application (by name). Select the app, then the client machine, and examine the recent CAL logs. Notice that the CAL transaction we just programmed is in the log file. NOTE: sometimes it takes several minutes for the messages to arrive at the CAL database.

## Verify the new transaction exists

Verify that our new transaction exists inside the URL transaction for accessing the account data.

When you're seeing the messages that you expect, you have completed the lab!