"use strict";

var express = require("express");

var fs = require('fs');

var channelObjects = require("../BusinessServices/channelObjects.js");

var schedule = require('node-schedule');

const util = require('util');

var userEnrollmentService, cachingService, commonDataService, fccrService, blockService, requestLccXmlService, loggingService;

var appLogger, xmlPushLogger, respPullLogger, cachingLogger;

createLoggerService();

function createLoggerService() {

    //logging

    loggingService = require("../BusinessServices/loggingService.js")();

    loggingService.createLoggingDirIfNotExists();

    appLogger = loggingService.getAppLogger();

    appLogger.info("appLogger works")

    xmlPushLogger = loggingService.getXmlPushLogger();

    xmlPushLogger.info("xmlPushLogger works")

    respPullLogger = loggingService.getRespPullLogger();

    respPullLogger.info("respPullLogger works")

    cachingLogger = loggingService.getCachingLogger();

    cachingLogger.info("cachingLogger works")

}

setTimeout(function () {

    //userEnrollmentService = require("../BusinessServices/userEnrollmentService.js")();

    cachingService = require("../BusinessServices/cachingService.js")(cachingLogger);

    commonDataService = require("../BusinessServices/commonDataService.js")();

    fccrService = require("../BusinessServices/fccrService.js")(channelObjects.fabric\_client, channelObjects.channels, channelObjects.peers, channelObjects.eventHubPeers, channelObjects.orderer, channelObjects.usersForTransaction, appLogger);

    blockService = require("../BusinessServices/blockService.js")(fccrService, appLogger);

    requestLccXmlService = require("../BusinessServices/requestLccXmlService.js")(fccrService, xmlPushLogger, respPullLogger);

    cachingService.getCachedData();

    var ruleJobCaching = new schedule.RecurrenceRule();

    ruleJobCaching.minute = [0, new schedule.Range(0, 60, 15)];

    var cronJobCaching = schedule.scheduleJob(ruleJobCaching, function () {

        cachingLogger.info('Run every 15 minutes: Caching Job');

        if (cachingService.isMasterDataChanged() || cachingService.isLegalEntityDataChanged()

            || cachingService.isCcHierarchyDataChanged() || cachingService.isMDMValidationDataChanged()) {

            cachingService.getCachedData();

        }

    });

    var ruleXMLCreation = new schedule.RecurrenceRule();

    ruleXMLCreation.minute = [0, new schedule.Range(0, 60, 5)];

    var cronJobXMLCreation = schedule.scheduleJob(ruleXMLCreation, async function () {

        xmlPushLogger.info('Executing Scheduler: XML File Creation Job');

        try {

            await requestLccXmlService.generateRequestLccXmlFiles();

            xmlPushLogger.info('XML files successfully generated');

        } catch (err) {

            xmlPushLogger.error("Error while generating XMLs " + err.message);

        }

    });

Env.json

{

"DEV": {

"UL\_SITE\_URL": "",

"UL\_USERNAME": "",

"UL\_PWD": "",

"UL\_DOMAIN": "",

"UL\_SP\_MDG\_INPUT\_DIR": "",

"UL\_SP\_MDG\_OUTUPUT\_DIR": "",

"UL\_SP\_NONMDG\_INPUT\_DIR": "",

"UL\_SP\_NONMDG\_OUPUT\_DIR": "",

"LOCAL\_DIR\_FOR\_SP\_DOWNLOAD": "UL\_FCCR\_xml\_Files/downloadFromSharePoint/",

"LOCAL\_DIR\_FOR\_SP\_UPLOAD": "UL\_FCCR\_xml\_Files/uploadToSharePoint/",

"LOGGING\_DIR": "UL\_FCCR\_xml\_Files",

"MAX\_ATTEMPTS\_SP\_UPLOAD": 0

},

Logging service.json

'use strict';

var fs = require('fs');

var helper = require('./helper.js');

const path = require('path');

const { createLogger, format, transports } = require('winston');

require('winston-daily-rotate-file');

const { combine, timestamp, label, printf } = format;

var environment\_config = require('../env.json')[process.env.NODE\_ENV || 'DEV'];

//Directory for creating log files

var envLogDir = environment\_config.LOGGING\_DIR;

const logDir = envLogDir + '/logs';

const pushInSPDir = logDir + '/push\_in\_share\_point';

const pullFromSPDir = logDir + '/pull\_from\_share\_point';

const cacheDir = logDir + '/cache';

const appDir = logDir + '/app';

const xmlPush = path.join(pushInSPDir, 'xmlPush');

const caching = path.join(cacheDir, 'caching');

const app = path.join(appDir, 'app');

const respPull = path.join(pullFromSPDir, 'respPull');

//printing format of logs

var myFormat = format.printf(info => {

if (typeof (info.message) == "object") {

let msg = JSON.stringify(info.message);

return `${info.timestamp} ${info.level}: ${msg} `;

}

return `${info.timestamp} ${info.level}: ${info.message}`;

});

//For writing the logs of the xml scheduler

var xmlPushTransport = new (transports.DailyRotateFile)({

level: 'debug',

filename: xmlPush + '%DATE%.log',

datePattern: 'YYYY-MM-DD',

maxSize: '20m',

//zippedArchive: true,

maxFiles: '7d',

format: format.combine(

format.timestamp({

format: 'YYYY-MM-DD HH:mm:ss'

}),

myFormat

)

});

var respPullTransport = new (transports.DailyRotateFile)({

level: 'debug',

filename: respPull + '%DATE%.log',

datePattern: 'YYYY-MM-DD',

maxSize: '20m',

//zippedArchive: true,

maxFiles: '7d',

format: format.combine(

format.timestamp({

format: 'YYYY-MM-DD HH:mm:ss'

}),

myFormat

)

});

var cacheTransport = new (transports.DailyRotateFile)({

level: 'debug',

filename: caching + '%DATE%.log',

datePattern: 'YYYY-MM-DD',

maxSize: '20m',

maxFiles: '7d',

format: format.combine(

format.timestamp({

format: 'YYYY-MM-DD HH:mm:ss'

}),

myFormat

)

});

var appTransport = new (transports.DailyRotateFile)({

level: 'debug',

filename: app + '%DATE%.log',

datePattern: 'YYYY-MM-DD',

maxSize: '20m',

maxFiles: '7d',

format: format.combine(

format.timestamp({

format: 'YYYY-MM-DD HH:mm:ss'

}),

myFormat

)

});

var consoleTransport = new (transports.Console)({

level: 'debug',

format: format.combine(

format.colorize(),

format.timestamp({

format: 'YYYY-MM-DD HH:mm:ss'

}),

myFormat

)

});

var respPullLogger = createLogger({

transports: [

consoleTransport,

respPullTransport

]

});

var xmlPushLogger = createLogger({

transports: [

consoleTransport,

xmlPushTransport

]

});

var cachingLogger = createLogger({

transports: [

consoleTransport,

cacheTransport

]

});

var appLogger = createLogger({

transports: [

consoleTransport,

appTransport

]

});

module.exports = function () {

var loggingService = {};

loggingService.createLoggingDirIfNotExists = function () {

helper.createDirIfNotExists(logDir);

}

loggingService.getAppLogger = function () {

console.log("inside getAppLogger")

return appLogger;

}

loggingService.getCachingLogger = function () {

return cachingLogger;

}

loggingService.getXmlPushLogger = function () {

return xmlPushLogger;

}

loggingService.getRespPullLogger = function () {

return respPullLogger;

}

return loggingService;

};

Blockservice.js

'use strict';

var fccrService;

var appLogger

const util=require('util')

module.exports = function (fccrService,appLogger) {

appLogger=appLogger;

var blockService = {};

blockService.queryInfo = function (role) {

if (role.toUpperCase() === 'UNILEVER') {

return fccrService.queryInfo()

.then((results) => {

return results;

}).catch((err) => {

appLogger.error(util.format('%o',err));

throw err;

});

}

appLogger.info("queryInfo");

}

blockService.getRecentBlocks = function (role, blockNumber) {

var blockList = new Array(0);

return blockService.queryInfo(role)

.then((results) => {

if (blockNumber == -1) {

blockNumber = results.height.low;

}

return blockService.getBlock(blockList, role, blockNumber - 1)

.then((results) => {

return results;

}).catch((err) => {

appLogger.error(util.format('%o',err));

throw err;

});

}).catch((err) => {

appLogger.error(util.format('%o',err));

throw err;

});

}

return blockService;

}