167 lines (146 sloc) | 5.19 KB ...

```
/* This module exports a set of utilities to support new database
 * population or in-place data migrations utilizing Node.js file streaming
 * and asynchronous methods to support large data sets.
 */

const fs = require('fs')
const util = require('util')

const argon2 = require('argon2')
const color = require('kleur')
const config = require('config')
const cryptoRandomString = require('crypto-random-string')
const JSONStream = require('JSONStream')
const mongoose = require('mongoose')
const prompt = require('prompt-sync')({ sigint: true })
const uuid = require('uuid')

const errors = require('../utils/error')
const logger = require('../middleware/logger')
const utils = require('../utils/utils')
const CONSTANTS = require('../constants')

const apiKeyFile = 'user-secret.txt'

const error = new errors.IDRError()


async function newOrgTransform (org) {
  org.UUID = uuid.v4()
```

```
  org.inUse = false
  return org
}

async function preprocessUserSecrets () {
  if (process.env.NODE_ENV === 'development') {
    const secretKey = process.env.LOCAL_KEY
    const hash = await argon2.hash(secretKey)

    // provide secret to user in development
    console.log(color.bold().black().bgWhite(
      'Use the following API secret for all users:') + ' ' +
      color.bold().black().italic().bgGreen(secretKey)
    )

    return hash
  } else {
    // delete file for user secrets if one already exists
    fs.unlink(apiKeyFile, err => {
      if (err && err.code !== 'ENOENT') {
        logger.error(error.fileDeleteError(err))
        mongoose.connection.close()
      }
    })

    console.log(
      color.bold().black().bgWhite('The users\' API secret can be found in:') + ' ' +
      color.bold().black().italic().bgGreen(apiKeyFile)
    )
  }
}

async function newUserTransform (user, hash) {
  const tmpOrgUUID = await utils.getOrgUUID(user.cna_short_name)
  user.org_UUID = tmpOrgUUID
  user.UUID = uuid.v4()
  user.authority = { active_roles: [] }

  // shared secret key in development environments
  if (process.env.NODE_ENV === 'development') {
    user.secret = hash
  } else {
    const randomKey = cryptoRandomString({ length: CONSTANTS.CRYPTO_RANDOM_STRING_LENGTH })
    user.secret = await argon2.hash(randomKey)

    // write each user's API key to file
    // necessary when standing up any new shared instance of the system
    const payload = { username: user.username, secret: randomKey }
    fs.writeFile(apiKeyFile, JSON.stringify(payload) + '\n', { flag: 'a' }, (err) => {
```

```
      if (err) {
        logger.error(error.fileWriteError(err))
        mongoose.connection.close()
      }
    })
  }

  return user
}

async function newCveIdTransform (cveId) {
  const tmpRequestingCnaUUID = await utils.getOrgUUID(cveId.requested_by.cna)
  const tmpOwningCnaUUID = await utils.getOrgUUID(cveId.owning_cna)
  const tmpUserUUID = await utils.getUserUUID(cveId.requested_by.user, tmpRequestingCnaUUID)
  cveId.requested_by.cna = tmpRequestingCnaUUID
  cveId.owning_cna = tmpOwningCnaUUID
  cveId.requested_by.user = tmpUserUUID
  return cveId
}

async function newCveTransform (cve) {
  const orgUUID = await utils.getOrgUUID(cve.cve.cveMetadata.assignerShortName) // find uuid based
  cve.cve.cveMetadata.assignerOrgId = orgUUID // assign the assigner to be the org uuid
  return cve
}

function getUserPopulateInput (collectionNames) {
  const appEnv = process.env.NODE_ENV
  const dbName = config.get(`${appEnv}.database`)
  const promptString = (
    `Are you sure you wish to pre-populate the database for the ${appEnv} environment? ` +
    `Doing so will drop the ${collectionNames.join(', ')} collection(s) ` +
    `in the ${dbName} database. (y/n) `
  )

  let userInput = prompt(promptString)
  while (userInput.toLowerCase() !== 'n' && userInput.toLowerCase() !== 'y') {
    console.log('Unrecognized Input')
    userInput = prompt(promptString)
  }
  return userInput
}

/* populates any collection given the file path for an input JSON,
 * the data model for the collection, a function that transforms the
 * data, and a hash that only gets use in the User collection
 */
function populateCollection (filePath, dataModel, dataTransform = async function (x) { return x },
  const dataName = dataModel.collection.collectionName
```

```
  logger.info(`Populating ${dataName} collection...`)
  return new Promise(function (resolve, reject) {
    // let batchData = []
    const promises = []
    fs.createReadStream(filePath, { encoding: 'utf8' })
      .pipe(JSONStream.parse('*'))
      .on('data', async function (data) {
        this.pause()
        data = await dataTransform(data, hash)
        promises.push(dataModel.insertMany(data))
        this.resume()
        promisesShifter(promises)
      })
      .on('close', function () {
        Promise.all(promises).then(function () {
          logger.info(`${dataName} populated!`)
          resolve()
        })
      })
      .on('error', reject)
  })
}

function promisesShifter (promises) {
  if (promises) {
    while (!util.inspect(promises[0]).includes('pending')) {
      promises.shift()
      if (promises.length === 0) { break }
    }
  }
}

module.exports = {
  getUserPopulateInput,
  newCveIdTransform,
  newOrgTransform,
  newUserTransform,
  newCveTransform,
  populateCollection,
  preprocessUserSecrets
}
```