package com.twitter.tweetypie

package repository

import com.twitter.passbird.clientapplication.thriftscala.ClientApplication

import com.twitter.passbird.clientapplication.thriftscala.GetClientApplicationsResponse

import com.twitter.servo.cache.ScopedCacheKey

import com.twitter.stitch.MapGroup

import com.twitter.stitch.NotFound

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.thriftscala.DeviceSource

// converts the device source parameter value to lower-case, to make the cached

// key case-insensitive

case class DeviceSourceKey(param: String) extends ScopedCacheKey("t", "ds", 1, param.toLowerCase)

object DeviceSourceRepository {

type Type = String => Stitch[DeviceSource]

type GetClientApplications = FutureArrow[Seq[Long], GetClientApplicationsResponse]

val DefaultUrl = "https://help.twitter.com/en/using-twitter/how-to-tweet#source-labels"

def formatUrl(name: String, url: String): String = s"""<a href="$url">$name</a>"""

/\*\*

\* Construct an html a tag from the client application

\* name and url for the display field because some

\* clients depend on this.

\*/

def deviceSourceDisplay(

name: String,

urlOpt: Option[String]

): String =

urlOpt match {

case Some(url) => formatUrl(name = name, url = url) // data sanitized by passbird

case None =>

formatUrl(name = name, url = DefaultUrl) // data sanitized by passbird

}

def toDeviceSource(app: ClientApplication): DeviceSource =

DeviceSource(

// The id field used to represent the id of a row

// in the now deprecated device\_sources mysql table.

id = 0L,

parameter = "oauth:" + app.id,

internalName = "oauth:" + app.id,

name = app.name,

url = app.url.getOrElse(""),

display = deviceSourceDisplay(app.name, app.url),

clientAppId = Some(app.id)

)

def apply(

parseAppId: String => Option[Long],

getClientApplications: GetClientApplications

): DeviceSourceRepository.Type = {

val getClientApplicationsGroup = new MapGroup[Long, DeviceSource] {

def run(ids: Seq[Long]): Future[Long => Try[DeviceSource]] =

getClientApplications(ids).map { response => id =>

response.found.get(id) match {

case Some(app) => Return(toDeviceSource(app))

case None => Throw(NotFound)

}

}

}

appIdStr =>

parseAppId(appIdStr) match {

case Some(appId) =>

Stitch.call(appId, getClientApplicationsGroup)

case None =>

Stitch.exception(NotFound)

}

}

}