package com.twitter.tweetypie

package federated.context

import com.twitter.common.ip\_address\_utils.ClientIpAddressUtils

import com.twitter.context.thriftscala.Viewer

import com.twitter.context.TwitterContext

import com.twitter.finagle.core.util.InetAddressUtil

import com.twitter.passbird.bitfield.clientprivileges.thriftscala.{Constants => ClientAppPrivileges}

import com.twitter.finatra.tfe.HttpHeaderNames

import com.twitter.spam.rtf.thriftscala.SafetyLevel

import com.twitter.strato.access.Access.ClientApplicationPrivilege

import com.twitter.strato.access.Access

import com.twitter.strato.access.ClientApplicationPrivilegeVariant

import com.twitter.strato.context.StratoContext

import com.twitter.strato.opcontext.OpContext

import com.twitter.strato.response.Err

import com.twitter.weaverbird.common.GetPlatformKey

/\*\*

\* [[RequestContext]] exists to avoid wiring the federated column

\* implementations directly to the request data that is derived from the

\* contextual environment. Columns should not directly reference

\* TwitterContext, StratoContext, strato.access.Access, HTTP headers, etc.

\* Each column operation operates on two input parameters: a request (i.e.

\* a column operation's Arg) and a [[RequestContext]].

\*/

private[federated] case class RequestContext(

clientApplicationId: Option[AppId] = None,

deviceSource: Option[String] = None,

knownDeviceToken: Option[KnownDeviceToken] = None,

remoteHost: Option[String] = None,

twitterUserId: UserId,

contributorId: Option[UserId] = None,

isDarkRequest: Boolean = false,

hasPrivilegeNullcastingAccess: Boolean = false,

hasPrivilegePromotedTweetsInTimeline: Boolean = false,

sessionHash: Option[String] = None,

cardsPlatformKey: Option[String] = None,

safetyLevel: Option[SafetyLevel] = None,

) {

def isContributorRequest = contributorId.exists(\_ != twitterUserId)

}

/\*\*

\* Provides a single place to derive request data from the contextual

\* environment. Defined as a sealed class (vs an object) to allow mocking

\* in unit tests.

\*/

private[federated] sealed class GetRequestContext() {

// Bring Tweetypie permitted TwitterContext into scope

private[this] val TwitterContext: TwitterContext =

com.twitter.context.TwitterContext(com.twitter.tweetypie.TwitterContextPermit)

/\*\*

\* When TwitterUserIdNotDefined is thrown, it's likely that the column

\* access control configuration lacks `AllowTwitterUserId` or other

\* Policy that ensures the caller is authenticated.

\*/

private[federated] val TwitterUserIdNotDefined =

Err(Err.Authentication, "User authentication is required for this operation.")

private[this] val SessionHashHeaderName = "x-tfe-session-hash"

private[this] def hasClientApplicationPrivilege(id: Int): Boolean =

Access.getPrincipals.contains(

ClientApplicationPrivilege(

ClientApplicationPrivilegeVariant

.byId(id.toShort).get))

private[this] def getRequestHeader(headerName: String): Option[String] =

StratoContext

.current()

.propagatedHeaders

.flatMap(\_.get(headerName))

def apply(opContext: OpContext): RequestContext = {

val twitterUserId = Access.getTwitterUserId match {

// Access.getTwitterUserId should return a value as long as the column

// policy includes AllowTwitterUserId, which guarantees the presence of

// the value.

case Some(twitterUser) => twitterUser.id

case None => throw TwitterUserIdNotDefined

}

// contributorId should only be defined when the authenticated user differs

// from the "Twitter user"

val contributorId =

Access.getAuthenticatedTwitterUserId.map(\_.id).filter(\_ != twitterUserId)

val twitterContext = TwitterContext().getOrElse(Viewer())

val deviceSource = twitterContext.clientApplicationId.map("oauth:" + \_)

// Ported from StatusesUpdateController#getBirdherdOptions and

// BirdherdOption.UserIp(request.clientHost)

val remoteHost: Option[String] =

getRequestHeader(HttpHeaderNames.X\_TWITTER\_AUDIT\_IP\_THRIFT.toLowerCase) // use the new header

.flatMap(ClientIpAddressUtils.decodeClientIpAddress(\_))

.flatMap(ClientIpAddressUtils.getString(\_))

.orElse(

getRequestHeader(

HttpHeaderNames.X\_TWITTER\_AUDIT\_IP.toLowerCase

) // fallback to old way before migration is completed

.map(h => InetAddressUtil.getByName(h.trim).getHostAddress)

)

val isDarkRequest = opContext.darkRequest.isDefined

val sessionHash = getRequestHeader(SessionHashHeaderName)

val cardsPlatformKey = twitterContext.clientApplicationId.map(GetPlatformKey(\_))

val safetyLevel = opContext.safetyLevel

RequestContext(

clientApplicationId = twitterContext.clientApplicationId,

deviceSource = deviceSource,

knownDeviceToken = twitterContext.knownDeviceToken,

remoteHost = remoteHost,

twitterUserId = twitterUserId,

contributorId = contributorId,

isDarkRequest = isDarkRequest,

hasPrivilegeNullcastingAccess =

hasClientApplicationPrivilege(ClientAppPrivileges.NULLCASTING\_ACCESS),

hasPrivilegePromotedTweetsInTimeline =

hasClientApplicationPrivilege(ClientAppPrivileges.PROMOTED\_TWEETS\_IN\_TIMELINE),

sessionHash = sessionHash,

cardsPlatformKey = cardsPlatformKey,

safetyLevel = safetyLevel,

)

}

}