package com.twitter.tweetypie

package federated.columns

import com.twitter.accounts.util.SafetyMetadataUtils

import com.twitter.ads.callback.thriftscala.EngagementRequest

import com.twitter.bouncer.thriftscala.{Bounce => BouncerBounce}

import com.twitter.stitch.Stitch

import com.twitter.strato.catalog.OpMetadata

import com.twitter.strato.config.AllOf

import com.twitter.strato.config.BouncerAccess

import com.twitter.strato.config.ContactInfo

import com.twitter.strato.config.Policy

import com.twitter.strato.data.Conv

import com.twitter.strato.data.Description.PlainText

import com.twitter.strato.data.Lifecycle.Production

import com.twitter.strato.fed.StratoFed

import com.twitter.strato.opcontext.OpContext

import com.twitter.strato.response.Err

import com.twitter.strato.thrift.ScroogeConv

import com.twitter.tweetypie.federated.columns.ApiErrors.\_

import com.twitter.tweetypie.federated.columns.CreateRetweetColumn.toCreateRetweetErr

import com.twitter.tweetypie.federated.context.GetRequestContext

import com.twitter.tweetypie.federated.prefetcheddata.PrefetchedDataRequest

import com.twitter.tweetypie.federated.prefetcheddata.PrefetchedDataResponse

import com.twitter.tweetypie.federated.promotedcontent.TweetPromotedContentLogger

import com.twitter.tweetypie.federated.promotedcontent.TweetPromotedContentLogger.RetweetEngagement

import com.twitter.tweetypie.thriftscala.TweetCreateState.\_

import com.twitter.tweetypie.thriftscala.{graphql => gql}

import com.twitter.tweetypie.{thriftscala => thrift}

import com.twitter.weaverbird.common.{GetRequestContext => WGetRequestContext}

class CreateRetweetColumn(

retweet: thrift.RetweetRequest => Future[thrift.PostTweetResult],

getRequestContext: GetRequestContext,

prefetchedDataRepository: PrefetchedDataRequest => Stitch[PrefetchedDataResponse],

logTweetPromotedContent: TweetPromotedContentLogger.Type,

statsReceiver: StatsReceiver,

) extends StratoFed.Column(CreateRetweetColumn.Path)

with StratoFed.Execute.StitchWithContext

with StratoFed.HandleDarkRequests {

override val policy: Policy = AllOf(

Seq(AccessPolicy.TweetMutationCommonAccessPolicies, BouncerAccess()))

// The underlying call to thriftTweetService.postRetweet is not idempotent

override val isIdempotent: Boolean = false

override type Arg = gql.CreateRetweetRequest

override type Result = gql.CreateRetweetResponseWithSubqueryPrefetchItems

override val argConv: Conv[Arg] = ScroogeConv.fromStruct

override val resultConv: Conv[Result] = ScroogeConv.fromStruct

override val contactInfo: ContactInfo = TweetypieContactInfo

override val metadata: OpMetadata = OpMetadata(

Some(Production),

Some(PlainText("Creates a retweet by the calling Twitter user of the given source tweet.")))

private val getWeaverbirdCtx = new WGetRequestContext()

override def execute(request: Arg, opContext: OpContext): Stitch[Result] = {

val ctx = getRequestContext(opContext)

// First, do any request parameter validation that can result in an error

// prior to calling into thriftTweetService.retweet.

val safetyLevel = ctx.safetyLevel.getOrElse(throw SafetyLevelMissingErr)

// Macaw-tweets returns ApiError.ClientNotPrivileged if the caller provides

// an impression\_id but lacks the PROMOTED\_TWEETS\_IN\_TIMELINE privilege.

val trackingId = request.engagementRequest match {

case Some(engagementRequest: EngagementRequest) if ctx.hasPrivilegePromotedTweetsInTimeline =>

TrackingId.parse(engagementRequest.impressionId, statsReceiver)

case Some(e: EngagementRequest) =>

throw ClientNotPrivilegedErr

case None =>

None

}

// DeviceSource is an oauth string computed from the ClientApplicationId.

// Macaw-tweets allows non-oauth callers, but GraphQL does not. An undefined

// ClientApplicationId is similar to TweetCreateState.DeviceSourceNotFound,

// which Macaw-tweets handles via a catch-all that returns

// ApiError.GenericAccessDenied

val deviceSource = ctx.deviceSource.getOrElse(throw GenericAccessDeniedErr)

// Macaw-tweets doesn't perform any parameter validation for the components

// used as input to makeSafetyMetaData.

val safetyMetadata = SafetyMetadataUtils.makeSafetyMetaData(

sessionHash = ctx.sessionHash,

knownDeviceToken = ctx.knownDeviceToken,

contributorId = ctx.contributorId

)

val thriftRetweetRequest = thrift.RetweetRequest(

sourceStatusId = request.tweetId,

userId = ctx.twitterUserId,

contributorUserId = None, // no longer supported, per tweet\_service.thrift

createdVia = deviceSource,

nullcast = request.nullcast,

trackingId = trackingId,

dark = ctx.isDarkRequest,

hydrationOptions = Some(HydrationOptions.writePathHydrationOptions(ctx.cardsPlatformKey)),

safetyMetaData = Some(safetyMetadata),

)

val stitchRetweet = Stitch.callFuture(retweet(thriftRetweetRequest))

request.engagementRequest.foreach { engagement =>

logTweetPromotedContent(engagement, RetweetEngagement, ctx.isDarkRequest)

}

stitchRetweet.flatMap { result: thrift.PostTweetResult =>

result.state match {

case thrift.TweetCreateState.Ok =>

val r = PrefetchedDataRequest(

tweet = result.tweet.get,

sourceTweet = result.sourceTweet,

quotedTweet = result.quotedTweet,

safetyLevel = safetyLevel,

requestContext = getWeaverbirdCtx()

)

prefetchedDataRepository(r)

.liftToOption()

.map((prefetchedData: Option[PrefetchedDataResponse]) => {

gql.CreateRetweetResponseWithSubqueryPrefetchItems(

data = Some(gql.CreateRetweetResponse(result.tweet.map(\_.id))),

subqueryPrefetchItems = prefetchedData.map(\_.value)

)

})

case errState =>

throw toCreateRetweetErr(errState, result.bounce, result.failureReason)

}

}

}

}

object CreateRetweetColumn {

val Path = "tweetypie/createRetweet.Tweet"

/\*\*

\* Ported from:

\* StatusesRetweetController#retweetStatus rescue block

\* TweetyPieStatusRepository.toRetweetException

\*/

def toCreateRetweetErr(

errState: thrift.TweetCreateState,

bounce: Option[BouncerBounce],

failureReason: Option[String]

): Err = errState match {

case CannotRetweetBlockingUser =>

BlockedUserErr

case AlreadyRetweeted =>

AlreadyRetweetedErr

case Duplicate =>

DuplicateStatusErr

case CannotRetweetOwnTweet | CannotRetweetProtectedTweet | CannotRetweetSuspendedUser =>

InvalidRetweetForStatusErr

case UserNotFound | SourceTweetNotFound | SourceUserNotFound | CannotRetweetDeactivatedUser =>

StatusNotFoundErr

case UserDeactivated | UserSuspended =>

UserDeniedRetweetErr

case RateLimitExceeded =>

RateLimitExceededErr

case UrlSpam =>

TweetUrlSpamErr

case Spam | UserReadonly =>

TweetSpammerErr

case SafetyRateLimitExceeded =>

SafetyRateLimitExceededErr

case Bounce if bounce.isDefined =>

accessDeniedByBouncerErr(bounce.get)

case DisabledByIpiPolicy =>

failureReason

.map(tweetEngagementLimitedErr)

.getOrElse(GenericAccessDeniedErr)

case TrustedFriendsRetweetNotAllowed =>

TrustedFriendsRetweetNotAllowedErr

case StaleTweetRetweetNotAllowed =>

StaleTweetRetweetNotAllowedErr

case \_ =>

GenericAccessDeniedErr

}

}