package com.twitter.tweetypie

package handler

import com.twitter.stitch.Stitch

import com.twitter.timelineservice.{thriftscala => tls}

import com.twitter.tweetypie.backends.TimelineService

import com.twitter.tweetypie.repository.TweetQuery

import com.twitter.tweetypie.repository.TweetRepository

import com.twitter.tweetypie.thriftscala.CardReference

import com.twitter.tweetypie.thriftscala.ConversationControl

import com.twitter.tweetypie.thriftscala.ConversationControlByInvitation

import com.twitter.tweetypie.thriftscala.ConversationControlCommunity

import com.twitter.tweetypie.thriftscala.ConversationControlFollowers

import com.twitter.tweetypie.thriftscala.EditControl

import com.twitter.tweetypie.thriftscala.EditOptions

import com.twitter.tweetypie.thriftscala.NoteTweetOptions

import com.twitter.tweetypie.thriftscala.PostTweetRequest

import com.twitter.tweetypie.thriftscala.TweetCreateConversationControl

import com.twitter.tweetypie.util.ConversationControls

import com.twitter.tweetypie.util.EditControlUtil

import com.twitter.util.Time

/\*\*

\* Used at tweet creation time to determine whether the tweet creation

\* request should be considered a duplicate of an existing tweet.

\*/

object DuplicateTweetFinder {

/\*\*

\* Return the ids of any tweets that are found to be duplicates of

\* this request.

\*/

type Type = RequestInfo => Future[Option[TweetId]]

final case class Settings(

// The number of tweets that are loaded from the user's timeline

// for the heuristic duplicate check

numTweetsToCheck: Int,

// The oldest that a tweet can be to still be considered a

// duplicate by the heuristic duplicate check

maxDuplicateAge: Duration)

// Takes a ConversationControl from a Tweet and converts to the equivalent

// TweetCreateConversationControl. Note: this is a lossy conversion because the

// ConversationControl contains additional data from the Tweet.

def toTweetCreateConversationControl(

conversationControl: ConversationControl

): TweetCreateConversationControl =

conversationControl match {

case ConversationControl.ByInvitation(

ConversationControlByInvitation(\_, \_, inviteViaMention)) =>

ConversationControls.Create.byInvitation(inviteViaMention)

case ConversationControl.Community(ConversationControlCommunity(\_, \_, inviteViaMention)) =>

ConversationControls.Create.community(inviteViaMention)

case ConversationControl.Followers(ConversationControlFollowers(\_, \_, inviteViaMention)) =>

ConversationControls.Create.followers(inviteViaMention)

case \_ => throw new IllegalArgumentException

}

/\*\*

\* The parts of the request that we need in order to perform

\* duplicate detection.

\*/

final case class RequestInfo(

userId: UserId,

isNarrowcast: Boolean,

isNullcast: Boolean,

text: String,

replyToTweetId: Option[TweetId],

mediaUploadIds: Seq[MediaId],

cardReference: Option[CardReference],

conversationControl: Option[TweetCreateConversationControl],

underlyingCreativesContainer: Option[CreativesContainerId],

editOptions: Option[EditOptions] = None,

noteTweetOptions: Option[NoteTweetOptions] = None) {

def isDuplicateOf(tweet: Tweet, oldestAcceptableTimestamp: Time): Boolean = {

val createdAt = getTimestamp(tweet)

val isDuplicateText = text == getText(tweet)

val isDuplicateReplyToTweetId = replyToTweetId == getReply(tweet).flatMap(\_.inReplyToStatusId)

val isDuplicateMedia = getMedia(tweet).map(\_.mediaId) == mediaUploadIds

val isDuplicateCardReference = getCardReference(tweet) == cardReference

val isDuplicateConversationControl =

tweet.conversationControl.map(toTweetCreateConversationControl) == conversationControl

val isDuplicateConversationContainerId = {

tweet.underlyingCreativesContainerId == underlyingCreativesContainer

}

val isDuplicateIfEditRequest = if (editOptions.isDefined) {

// We do not count an incoming edit request as creating a duplicate tweet if:

// 1) The tweet that is considered a duplicate is a previous version of this tweet OR

// 2) The tweet that is considered a duplicate is otherwise stale.

val tweetEditChain = tweet.editControl match {

case Some(EditControl.Initial(initial)) =>

initial.editTweetIds

case Some(EditControl.Edit(edit)) =>

edit.editControlInitial.map(\_.editTweetIds).getOrElse(Nil)

case \_ => Nil

}

val tweetIsAPreviousVersion =

editOptions.map(\_.previousTweetId).exists(tweetEditChain.contains)

val tweetIsStale = EditControlUtil.isLatestEdit(tweet.editControl, tweet.id) match {

case Return(false) => true

case \_ => false

}

!(tweetIsStale || tweetIsAPreviousVersion)

} else {

// If not an edit request, this condition is true as duplication checking is not blocked

true

}

// Note that this does not prevent you from tweeting the same

// image twice with different text, or the same text twice with

// different images, because if you upload the same media twice,

// we will store two copies of it, each with a different media

// URL and thus different t.co URL, and since the text that

// we're checking here has that t.co URL added to it already, it

// is necessarily different.

//

// We shouldn't have to check the user id or whether it's a

// retweet, because we loaded the tweets from the user's

// (non-retweet) timelines, but it doesn't hurt and protects

// against possible future changes.

(oldestAcceptableTimestamp <= createdAt) &&

getShare(tweet).isEmpty &&

(getUserId(tweet) == userId) &&

isDuplicateText &&

isDuplicateReplyToTweetId &&

isDuplicateMedia &&

isDuplicateCardReference &&

isDuplicateConversationControl &&

isDuplicateConversationContainerId &&

isDuplicateIfEditRequest &&

noteTweetOptions.isEmpty // Skip duplicate checks for NoteTweets

}

}

object RequestInfo {

/\*\*

\* Extract the information relevant to the DuplicateTweetFinder

\* from the PostTweetRequest.

\*/

def fromPostTweetRequest(req: PostTweetRequest, processedText: String): RequestInfo =

RequestInfo(

userId = req.userId,

isNarrowcast = req.narrowcast.nonEmpty,

isNullcast = req.nullcast,

text = processedText,

replyToTweetId = req.inReplyToTweetId,

mediaUploadIds = req.mediaUploadIds.getOrElse[Seq[MediaId]](Seq.empty),

cardReference = req.additionalFields.flatMap(\_.cardReference),

conversationControl = req.conversationControl,

underlyingCreativesContainer = req.underlyingCreativesContainerId,

editOptions = req.editOptions,

noteTweetOptions = req.noteTweetOptions

)

}

/\*\*

\* Encapsulates the external interactions that we need to do for

\* duplicate checking.

\*/

trait TweetSource {

def loadTweets(tweetIds: Seq[TweetId]): Future[Seq[Tweet]]

def loadUserTimelineIds(userId: UserId, maxCount: Int): Future[Seq[TweetId]]

def loadNarrowcastTimelineIds(userId: UserId, maxCount: Int): Future[Seq[TweetId]]

}

object TweetSource {

/\*\*

\* Use the provided services to access tweets.

\*/

def fromServices(

tweetRepo: TweetRepository.Optional,

getStatusTimeline: TimelineService.GetStatusTimeline

): TweetSource =

new TweetSource {

// only fields needed by RequestInfo.isDuplicateOf()

private[this] val tweetQueryOption =

TweetQuery.Options(

TweetQuery.Include(

tweetFields = Set(

Tweet.CoreDataField.id,

Tweet.MediaField.id,

Tweet.ConversationControlField.id,

Tweet.EditControlField.id

),

pastedMedia = true

)

)

private[this] def loadTimeline(query: tls.TimelineQuery): Future[Seq[Long]] =

getStatusTimeline(Seq(query)).map(\_.head.entries.map(\_.statusId))

override def loadUserTimelineIds(userId: UserId, maxCount: Int): Future[Seq[Long]] =

loadTimeline(

tls.TimelineQuery(

timelineType = tls.TimelineType.User,

timelineId = userId,

maxCount = maxCount.toShort

)

)

override def loadNarrowcastTimelineIds(userId: UserId, maxCount: Int): Future[Seq[Long]] =

loadTimeline(

tls.TimelineQuery(

timelineType = tls.TimelineType.Narrowcasted,

timelineId = userId,

maxCount = maxCount.toShort

)

)

override def loadTweets(tweetIds: Seq[TweetId]): Future[Seq[Tweet]] =

if (tweetIds.isEmpty) {

Future.value(Seq[Tweet]())

} else {

Stitch

.run(

Stitch.traverse(tweetIds) { tweetId => tweetRepo(tweetId, tweetQueryOption) }

)

.map(\_.flatten)

}

}

}

def apply(settings: Settings, tweetSource: TweetSource): Type = { reqInfo =>

if (reqInfo.isNullcast) {

// iff nullcast, we bypass duplication logic all together

Future.None

} else {

val oldestAcceptableTimestamp = Time.now - settings.maxDuplicateAge

val userTweetIdsFut =

tweetSource.loadUserTimelineIds(reqInfo.userId, settings.numTweetsToCheck)

// Check the narrowcast timeline iff this is a narrowcasted tweet

val narrowcastTweetIdsFut =

if (reqInfo.isNarrowcast) {

tweetSource.loadNarrowcastTimelineIds(reqInfo.userId, settings.numTweetsToCheck)

} else {

Future.value(Seq.empty)

}

for {

userTweetIds <- userTweetIdsFut

narrowcastTweetIds <- narrowcastTweetIdsFut

candidateTweets <- tweetSource.loadTweets(userTweetIds ++ narrowcastTweetIds)

} yield candidateTweets.find(reqInfo.isDuplicateOf(\_, oldestAcceptableTimestamp)).map(\_.id)

}

}

}