package com.twitter.tweetypie

package repository

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

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.FilteredState

import com.twitter.tweetypie.media.Media

import com.twitter.tweetypie.media.MediaUrl

import com.twitter.tweetypie.thriftscala.\_

import com.twitter.tweetypie.util.MediaId

import java.nio.ByteBuffer

case class PastedMedia(mediaEntities: Seq[MediaEntity], mediaTags: Map[MediaId, Seq[MediaTag]]) {

/\*\*

\* Updates the copied media entities to have the same indices as the given UrlEntity.

\*/

def updateEntities(urlEntity: UrlEntity): PastedMedia =

if (mediaEntities.isEmpty) this

else copy(mediaEntities = mediaEntities.map(Media.copyFromUrlEntity(\_, urlEntity)))

def merge(that: PastedMedia): PastedMedia =

PastedMedia(

mediaEntities = this.mediaEntities ++ that.mediaEntities,

mediaTags = this.mediaTags ++ that.mediaTags

)

/\*\*

\* Return a new PastedMedia that contains only the first maxMediaEntities media entities

\*/

def take(maxMediaEntities: Int): PastedMedia = {

val entities = this.mediaEntities.take(maxMediaEntities)

val mediaIds = entities.map(\_.mediaId)

val pastedTags = mediaTags.filterKeys { id => mediaIds.contains(id) }

PastedMedia(

mediaEntities = entities,

mediaTags = pastedTags

)

}

def mergeTweetMediaTags(ownedTags: Option[TweetMediaTags]): Option[TweetMediaTags] = {

val merged = ownedTags.map(\_.tagMap).getOrElse(Map.empty) ++ mediaTags

if (merged.nonEmpty) {

Some(TweetMediaTags(merged))

} else {

None

}

}

}

object PastedMedia {

import MediaUrl.Permalink.hasTweetId

val empty: PastedMedia = PastedMedia(Nil, Map.empty)

/\*\*

\* @param tweet: the tweet whose media URL was pasted.

\*

\* @return the media that should be copied to a tweet that has a

\* link to the media in this tweet, along with its protection

\* status. The returned media entities will have sourceStatusId

\* and sourceUserId set appropriately for inclusion in a different

\* tweet.

\*/

def getMediaEntities(tweet: Tweet): Seq[MediaEntity] =

getMedia(tweet).collect {

case mediaEntity if hasTweetId(mediaEntity, tweet.id) =>

setSource(mediaEntity, tweet.id, getUserId(tweet))

}

def setSource(mediaEntity: MediaEntity, tweetId: TweetId, userId: TweetId): MediaEntity =

mediaEntity.copy(

sourceStatusId = Some(tweetId),

sourceUserId = Some(mediaEntity.sourceUserId.getOrElse(userId))

)

}

object PastedMediaRepository {

type Type = (TweetId, Ctx) => Stitch[PastedMedia]

case class Ctx(

includeMediaEntities: Boolean,

includeAdditionalMetadata: Boolean,

includeMediaTags: Boolean,

extensionsArgs: Option[ByteBuffer],

safetyLevel: SafetyLevel) {

def asTweetQueryOptions: TweetQuery.Options =

TweetQuery.Options(

enforceVisibilityFiltering = true,

extensionsArgs = extensionsArgs,

safetyLevel = safetyLevel,

include = TweetQuery.Include(

tweetFields =

Set(Tweet.CoreDataField.id) ++

(if (includeMediaEntities) Set(Tweet.MediaField.id) else Set.empty) ++

(if (includeMediaTags) Set(Tweet.MediaTagsField.id) else Set.empty),

mediaFields = if (includeMediaEntities && includeAdditionalMetadata) {

Set(MediaEntity.AdditionalMetadataField.id)

} else {

Set.empty

},

// don't recursively load pasted media

pastedMedia = false

)

)

}

/\*\*

\* A Repository of PastedMedia fetched from other tweets. We query the tweet with

\* default global visibility filtering enabled, so we won't see entities for users that

\* are protected, deactivated, suspended, etc.

\*/

def apply(tweetRepo: TweetRepository.Type): Type =

(tweetId, ctx) =>

tweetRepo(tweetId, ctx.asTweetQueryOptions)

.flatMap { t =>

val entities = PastedMedia.getMediaEntities(t)

if (entities.nonEmpty) {

Stitch.value(PastedMedia(entities, getMediaTagMap(t)))

} else {

Stitch.NotFound

}

}

.rescue {

// drop filtered tweets

case \_: FilteredState => Stitch.NotFound

}

}