package com.twitter.tweetypie.repository

import com.twitter.spam.rtf.thriftscala.FilteredReason

import com.twitter.spam.rtf.thriftscala.{SafetyLevel => ThriftSafetyLevel}

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.TweetId

import com.twitter.tweetypie.core.FilteredState.HasFilteredReason

import com.twitter.tweetypie.core.FilteredState.Unavailable.BounceDeleted

import com.twitter.tweetypie.core.FilteredState.Unavailable.SourceTweetNotFound

import com.twitter.tweetypie.core.FilteredState.Unavailable.TweetDeleted

import com.twitter.tweetypie.repository.VisibilityResultToFilteredState.toFilteredStateUnavailable

import com.twitter.visibility.interfaces.tweets.DeletedTweetVisibilityLibrary

import com.twitter.visibility.models.SafetyLevel

import com.twitter.visibility.models.TweetDeleteReason

import com.twitter.visibility.models.TweetDeleteReason.TweetDeleteReason

import com.twitter.visibility.models.ViewerContext

/\*\*

\* Generate FilteredReason for tweet entities in following delete states:

\* com.twitter.tweetypie.core.FilteredState.Unavailable

\* - SourceTweetNotFound(true)

\* - TweetDeleted

\* - BounceDeleted

\*

\* Callers of this repository should be ready to handle empty response (Stitch.None)

\* from the underlying VF library when:

\* 1.the tweet should not NOT be filtered for the given safety level

\* 2.the tweet is not a relevant content to be handled by the library

\*/

object DeletedTweetVisibilityRepository {

type Type = VisibilityRequest => Stitch[Option[FilteredReason]]

case class VisibilityRequest(

filteredState: Throwable,

tweetId: TweetId,

safetyLevel: Option[ThriftSafetyLevel],

viewerId: Option[Long],

isInnerQuotedTweet: Boolean)

def apply(

visibilityLibrary: DeletedTweetVisibilityLibrary.Type

): Type = { request =>

toVisibilityTweetDeleteState(request.filteredState, request.isInnerQuotedTweet)

.map { deleteReason =>

val safetyLevel = SafetyLevel.fromThrift(

request.safetyLevel.getOrElse(ThriftSafetyLevel.FilterDefault)

)

val isRetweet = request.filteredState == SourceTweetNotFound(true)

visibilityLibrary(

DeletedTweetVisibilityLibrary.Request(

request.tweetId,

safetyLevel,

ViewerContext.fromContextWithViewerIdFallback(request.viewerId),

deleteReason,

isRetweet,

request.isInnerQuotedTweet

)

).map(toFilteredStateUnavailable)

.map {

//Accept FilteredReason

case Some(fs) if fs.isInstanceOf[HasFilteredReason] =>

Some(fs.asInstanceOf[HasFilteredReason].filteredReason)

case \_ => None

}

}

.getOrElse(Stitch.None)

}

/\*\*

\* @return map an error from tweet hydration to a VF model TweetDeleteReason,

\* None when the error is not related to delete state tweets.

\*/

private def toVisibilityTweetDeleteState(

tweetDeleteState: Throwable,

isInnerQuotedTweet: Boolean

): Option[TweetDeleteReason] = {

tweetDeleteState match {

case TweetDeleted => Some(TweetDeleteReason.Deleted)

case BounceDeleted => Some(TweetDeleteReason.BounceDeleted)

case SourceTweetNotFound(true) if !isInnerQuotedTweet => Some(TweetDeleteReason.Deleted)

case \_ => None

}

}

}