package com.twitter.tweetypie

package hydrator

import com.twitter.stitch.NotFound

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.\_

import com.twitter.tweetypie.repository.\_

import com.twitter.tweetypie.thriftscala.\_

object MediaTagsHydrator {

type Type = ValueHydrator[Option[TweetMediaTags], TweetCtx]

/\*\*

\* TweetMediaTags contains a map of MediaId to Seq[MediaTag].

\* The outer traverse maps over each MediaId, while the inner

\* traverse maps over each MediaTag.

\*

\* A MediaTag has four fields:

\*

\* 1: MediaTagType tag\_type

\* 2: optional i64 user\_id

\* 3: optional string screen\_name

\* 4: optional string name

\*

\* For each MediaTag, if the tag type is MediaTagType.User and the user id is defined

\* (see mediaTagToKey) we look up the tagged user, using the tagging user (the tweet

\* author) as the viewer id (this means that visibility rules between the tagged user

\* and tagging user are applied).

\*

\* If we get a taggable user back, we fill in the screen name and name fields. If not,

\* we drop the tag.

\*/

def apply(repo: UserViewRepository.Type): Type =

ValueHydrator[TweetMediaTags, TweetCtx] { (tags, ctx) =>

val mediaTagsByMediaId: Seq[(MediaId, Seq[MediaTag])] = tags.tagMap.toSeq

Stitch

.traverse(mediaTagsByMediaId) {

case (mediaId, mediaTags) =>

Stitch.traverse(mediaTags)(tag => hydrateMediaTag(repo, tag, ctx.userId)).map {

ValueState.sequence(\_).map(tags => (mediaId, tags.flatten))

}

}

.map {

// Reconstruct TweetMediaTags(tagMap: Map[MediaId, SeqMediaTag])

ValueState.sequence(\_).map(s => TweetMediaTags(s.toMap))

}

}.onlyIf { (\_, ctx) =>

!ctx.isRetweet && ctx.tweetFieldRequested(Tweet.MediaTagsField)

}.liftOption

/\*\*

\* A function to hydrate a single `MediaTag`. The return type is `Option[MediaTag]`

\* because we may return `None` to filter out a `MediaTag` if the tagged user doesn't

\* exist or isn't taggable.

\*/

private[this] def hydrateMediaTag(

repo: UserViewRepository.Type,

mediaTag: MediaTag,

authorId: UserId

): Stitch[ValueState[Option[MediaTag]]] =

mediaTagToKey(mediaTag) match {

case None => Stitch.value(ValueState.unmodified(Some(mediaTag)))

case Some(key) =>

repo(toRepoQuery(key, authorId))

.map {

case user if user.mediaView.exists(\_.canMediaTag) =>

ValueState.modified(

Some(

mediaTag.copy(

userId = Some(user.id),

screenName = user.profile.map(\_.screenName),

name = user.profile.map(\_.name)

)

)

)

// if `canMediaTag` is false, drop the tag

case \_ => ValueState.modified(None)

}

.handle {

// if user is not found, drop the tag

case NotFound => ValueState.modified(None)

}

}

private[this] val queryFields: Set[UserField] = Set(UserField.Profile, UserField.MediaView)

def toRepoQuery(userKey: UserKey, forUserId: UserId): UserViewRepository.Query =

UserViewRepository.Query(

userKey = userKey,

// view is based on tagging user, not tweet viewer

forUserId = Some(forUserId),

visibility = UserVisibility.MediaTaggable,

queryFields = queryFields

)

private[this] def mediaTagToKey(mediaTag: MediaTag): Option[UserKey] =

mediaTag match {

case MediaTag(MediaTagType.User, Some(taggedUserId), \_, \_) => Some(UserKey(taggedUserId))

case \_ => None

}

}