package com.twitter.visibility.builder.tweets

import com.twitter.communities.moderation.thriftscala.CommunityTweetModerationState

import com.twitter.communities.moderation.thriftscala.CommunityUserModerationState

import com.twitter.communities.visibility.thriftscala.CommunityVisibilityFeatures

import com.twitter.communities.visibility.thriftscala.CommunityVisibilityFeaturesV1

import com.twitter.communities.visibility.thriftscala.CommunityVisibilityResult

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.visibility.builder.FeatureMapBuilder

import com.twitter.visibility.common.CommunitiesSource

import com.twitter.visibility.features.CommunityTweetAuthorIsRemoved

import com.twitter.visibility.features.CommunityTweetCommunityNotFound

import com.twitter.visibility.features.CommunityTweetCommunityDeleted

import com.twitter.visibility.features.CommunityTweetCommunitySuspended

import com.twitter.visibility.features.CommunityTweetCommunityVisible

import com.twitter.visibility.features.CommunityTweetIsHidden

import com.twitter.visibility.features.TweetIsCommunityTweet

import com.twitter.visibility.features.ViewerIsCommunityAdmin

import com.twitter.visibility.features.ViewerIsCommunityMember

import com.twitter.visibility.features.ViewerIsCommunityModerator

import com.twitter.visibility.features.ViewerIsInternalCommunitiesAdmin

import com.twitter.visibility.models.CommunityTweet

import com.twitter.visibility.models.ViewerContext

class CommunityTweetFeaturesV2(communitiesSource: CommunitiesSource)

extends CommunityTweetFeatures {

private[this] def forCommunityTweet(

communityTweet: CommunityTweet

): FeatureMapBuilder => FeatureMapBuilder = { builder: FeatureMapBuilder =>

{

val communityVisibilityFeaturesStitch =

communitiesSource.getCommunityVisibilityFeatures(communityTweet.communityId)

val communityTweetModerationStateStitch =

communitiesSource.getTweetModerationState(communityTweet.tweet.id)

val communityTweetAuthorModerationStateStitch =

communitiesSource.getUserModerationState(

communityTweet.authorId,

communityTweet.communityId

)

def getFlagFromFeatures(f: CommunityVisibilityFeaturesV1 => Boolean): Stitch[Boolean] =

communityVisibilityFeaturesStitch.map {

case Some(CommunityVisibilityFeatures.V1(v1)) => f(v1)

case \_ => false

}

def getFlagFromCommunityVisibilityResult(

f: CommunityVisibilityResult => Boolean

): Stitch[Boolean] = getFlagFromFeatures { v =>

f(v.communityVisibilityResult)

}

builder

.withConstantFeature(

TweetIsCommunityTweet,

true

)

.withFeature(

CommunityTweetCommunityNotFound,

getFlagFromCommunityVisibilityResult {

case CommunityVisibilityResult.NotFound => true

case \_ => false

}

)

.withFeature(

CommunityTweetCommunitySuspended,

getFlagFromCommunityVisibilityResult {

case CommunityVisibilityResult.Suspended => true

case \_ => false

}

)

.withFeature(

CommunityTweetCommunityDeleted,

getFlagFromCommunityVisibilityResult {

case CommunityVisibilityResult.Deleted => true

case \_ => false

}

)

.withFeature(

CommunityTweetCommunityVisible,

getFlagFromCommunityVisibilityResult {

case CommunityVisibilityResult.Visible => true

case \_ => false

}

)

.withFeature(

ViewerIsInternalCommunitiesAdmin,

getFlagFromFeatures { \_.viewerIsInternalAdmin }

)

.withFeature(

ViewerIsCommunityAdmin,

getFlagFromFeatures { \_.viewerIsCommunityAdmin }

)

.withFeature(

ViewerIsCommunityModerator,

getFlagFromFeatures { \_.viewerIsCommunityModerator }

)

.withFeature(

ViewerIsCommunityMember,

getFlagFromFeatures { \_.viewerIsCommunityMember }

)

.withFeature(

CommunityTweetIsHidden,

communityTweetModerationStateStitch.map {

case Some(CommunityTweetModerationState.Hidden(\_)) => true

case \_ => false

}

)

.withFeature(

CommunityTweetAuthorIsRemoved,

communityTweetAuthorModerationStateStitch.map {

case Some(CommunityUserModerationState.Removed(\_)) => true

case \_ => false

}

)

}

}

def forTweet(

tweet: Tweet,

viewerContext: ViewerContext

): FeatureMapBuilder => FeatureMapBuilder = {

CommunityTweet(tweet) match {

case None => forNonCommunityTweet()

case Some(communityTweet) => forCommunityTweet(communityTweet)

}

}

}