package com.twitter.visibility.interfaces.des

import com.twitter.gizmoduck.thriftscala.User

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.visibility.VisibilityLibrary

import com.twitter.visibility.builder.VisibilityResult

import com.twitter.visibility.builder.tweets.CommunityTweetFeaturesV2

import com.twitter.visibility.builder.tweets.EditTweetFeatures

import com.twitter.visibility.builder.tweets.ExclusiveTweetFeatures

import com.twitter.visibility.builder.tweets.NilTweetLabelMaps

import com.twitter.visibility.builder.tweets.TrustedFriendsFeatures

import com.twitter.visibility.builder.tweets.TweetFeatures

import com.twitter.visibility.builder.users.AuthorFeatures

import com.twitter.visibility.builder.users.ViewerFeatures

import com.twitter.visibility.common.CommunitiesSource

import com.twitter.visibility.common.TrustedFriendsSource

import com.twitter.visibility.common.UserRelationshipSource

import com.twitter.visibility.common.UserSource

import com.twitter.visibility.models.ContentId

import com.twitter.visibility.models.SafetyLevel

import com.twitter.visibility.models.ViewerContext

import com.twitter.visibility.rules.Allow

import com.twitter.visibility.{thriftscala => vfthrift}

case class DESRealtimeVisibilityRequest(tweet: Tweet, author: User, viewer: Option[User])

object DESRealtimeVisibilityLibrary {

type Type = DESRealtimeVisibilityRequest => Stitch[vfthrift.Action]

private[this] val safetyLevel = SafetyLevel.DesRealtime

def apply(visibilityLibrary: VisibilityLibrary): Type = {

val libraryStatsReceiver = visibilityLibrary.statsReceiver

val vfEngineCounter = libraryStatsReceiver.counter("vf\_engine\_requests")

val tweetFeatures = new TweetFeatures(NilTweetLabelMaps, libraryStatsReceiver)

val authorFeatures = new AuthorFeatures(UserSource.empty, libraryStatsReceiver)

val viewerFeatures = new ViewerFeatures(UserSource.empty, libraryStatsReceiver)

val communityTweetFeatures = new CommunityTweetFeaturesV2(CommunitiesSource.empty)

val exclusiveTweetFeatures =

new ExclusiveTweetFeatures(UserRelationshipSource.empty, libraryStatsReceiver)

val trustedFriendsTweetFeatures = new TrustedFriendsFeatures(TrustedFriendsSource.empty)

val editTweetFeatures = new EditTweetFeatures(libraryStatsReceiver)

{ request: DESRealtimeVisibilityRequest =>

vfEngineCounter.incr()

val tweet = request.tweet

val author = request.author

val viewer = request.viewer

val viewerContext = ViewerContext.fromContext

val featureMap =

visibilityLibrary.featureMapBuilder(

Seq(

tweetFeatures.forTweetWithoutSafetyLabels(tweet),

authorFeatures.forAuthorNoDefaults(author),

viewerFeatures.forViewerNoDefaults(viewer),

communityTweetFeatures.forTweetOnly(tweet),

exclusiveTweetFeatures.forTweetOnly(tweet),

trustedFriendsTweetFeatures.forTweetOnly(tweet),

editTweetFeatures.forTweet(tweet),

)

)

val tweetResult = visibilityLibrary.runRuleEngine(

ContentId.TweetId(tweet.id),

featureMap,

viewerContext,

safetyLevel

)

val authorResult = visibilityLibrary.runRuleEngine(

ContentId.UserId(author.id),

featureMap,

viewerContext,

safetyLevel

)

Stitch.join(tweetResult, authorResult).map {

case (tweetResult, authorResult) => mergeResults(tweetResult, authorResult)

}

}

}

def mergeResults(

tweetResult: VisibilityResult,

authorResult: VisibilityResult,

): vfthrift.Action = {

Set(tweetResult.verdict, authorResult.verdict)

.find {

case Allow => false

case \_ => true

}

.map(\_.toActionThrift())

.getOrElse(Allow.toActionThrift())

}

}