package com.twitter.visibility.interfaces.dms

import com.twitter.stitch.Stitch

import com.twitter.strato.client.{Client => StratoClient}

import com.twitter.visibility.VisibilityLibrary

import com.twitter.visibility.builder.VisibilityResult

import com.twitter.visibility.builder.dms.DmConversationFeatures

import com.twitter.visibility.builder.dms.DmEventFeatures

import com.twitter.visibility.builder.dms.InvalidDmEventFeatureException

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

import com.twitter.visibility.common.UserSource

import com.twitter.visibility.common.dm\_sources.DmConversationSource

import com.twitter.visibility.common.dm\_sources.DmEventSource

import com.twitter.visibility.common.stitch.StitchHelpers

import com.twitter.visibility.models.ContentId.DmEventId

import com.twitter.visibility.rules.Drop

import com.twitter.visibility.rules.Reason

import com.twitter.visibility.rules.RuleBase

object DmEventVisibilityLibrary {

type Type = DmEventVisibilityRequest => Stitch[VisibilityResult]

def apply(

visibilityLibrary: VisibilityLibrary,

stratoClient: StratoClient,

userSource: UserSource

): Type = {

val libraryStatsReceiver = visibilityLibrary.statsReceiver

val stratoClientStatsReceiver = visibilityLibrary.statsReceiver.scope("strato")

val vfLatencyStatsReceiver = visibilityLibrary.statsReceiver.scope("vf\_latency")

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

val dmConversationSource = {

DmConversationSource.fromStrato(stratoClient, stratoClientStatsReceiver)

}

val dmEventSource = {

DmEventSource.fromStrato(stratoClient, stratoClientStatsReceiver)

}

val authorFeatures = new AuthorFeatures(userSource, libraryStatsReceiver)

val dmConversationFeatures = new DmConversationFeatures(dmConversationSource, authorFeatures)

val dmEventFeatures =

new DmEventFeatures(

dmEventSource,

dmConversationSource,

authorFeatures,

dmConversationFeatures,

libraryStatsReceiver)

{ req: DmEventVisibilityRequest =>

val dmEventId = req.dmEventId

val contentId = DmEventId(dmEventId)

val safetyLevel = req.safetyLevel

if (!RuleBase.hasDmEventRules(safetyLevel)) {

Stitch.value(VisibilityResult(contentId = contentId, verdict = Drop(Reason.Unspecified)))

} else {

vfEngineCounter.incr()

val viewerContext = req.viewerContext

val viewerIdOpt = viewerContext.userId

viewerIdOpt match {

case Some(viewerId) =>

val featureMap = visibilityLibrary.featureMapBuilder(

Seq(dmEventFeatures.forDmEventId(dmEventId, viewerId)))

val resp = visibilityLibrary

.runRuleEngine(

contentId,

featureMap,

viewerContext,

safetyLevel

)

StitchHelpers.profileStitch(resp, Seq(vfLatencyStatsReceiver))

case None => Stitch.exception(InvalidDmEventFeatureException("Viewer id is missing"))

}

}

}

}

}