package com.twitter.visibility.interfaces.dms

import com.twitter.servo.util.Gate

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.users.AuthorFeatures

import com.twitter.visibility.common.UserSource

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

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

import com.twitter.visibility.features.FeatureMap

import com.twitter.visibility.models.ContentId.DmConversationId

import com.twitter.visibility.rules.Drop

import com.twitter.visibility.rules.EvaluationContext

import com.twitter.visibility.rules.Reason

import com.twitter.visibility.rules.RuleBase

import com.twitter.visibility.rules.providers.ProvidedEvaluationContext

import com.twitter.visibility.rules.utils.ShimUtils

object DmConversationVisibilityLibrary {

type Type = DmConversationVisibilityRequest => Stitch[VisibilityResult]

def apply(

visibilityLibrary: VisibilityLibrary,

stratoClient: StratoClient,

userSource: UserSource,

enableVfFeatureHydrationInShim: Gate[Unit] = Gate.False

): 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 authorFeatures = new AuthorFeatures(userSource, libraryStatsReceiver)

val dmConversationFeatures = new DmConversationFeatures(dmConversationSource, authorFeatures)

{ req: DmConversationVisibilityRequest =>

val dmConversationId = req.dmConversationId

val contentId = DmConversationId(dmConversationId)

val safetyLevel = req.safetyLevel

if (!RuleBase.hasDmConversationRules(safetyLevel)) {

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

} else {

vfEngineCounter.incr()

val viewerContext = req.viewerContext

val viewerId = viewerContext.userId

val isVfFeatureHydrationEnabled: Boolean =

enableVfFeatureHydrationInShim()

val featureMap = visibilityLibrary.featureMapBuilder(

Seq(dmConversationFeatures.forDmConversationId(dmConversationId, viewerId)))

val resp = if (isVfFeatureHydrationEnabled) {

val evaluationContext = ProvidedEvaluationContext.injectRuntimeRulesIntoEvaluationContext(

evaluationContext = EvaluationContext(

safetyLevel,

visibilityLibrary.getParams(viewerContext, safetyLevel),

visibilityLibrary.statsReceiver)

)

val preFilteredFeatureMap =

ShimUtils.preFilterFeatureMap(featureMap, safetyLevel, contentId, evaluationContext)

FeatureMap.resolve(preFilteredFeatureMap, libraryStatsReceiver).flatMap {

resolvedFeatureMap =>

visibilityLibrary

.runRuleEngine(

contentId,

resolvedFeatureMap,

viewerContext,

safetyLevel

)

}

} else {

visibilityLibrary

.runRuleEngine(

contentId,

featureMap,

viewerContext,

safetyLevel

)

}

StitchHelpers.profileStitch(resp, Seq(vfLatencyStatsReceiver))

}

}

}

}