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.users.AuthorFeatures

import com.twitter.visibility.common.DmId

import com.twitter.visibility.common.UserId

import com.twitter.visibility.common.UserSource

import com.twitter.visibility.features.FeatureMap

import com.twitter.visibility.models.ContentId.{DmId => DmContentId}

import com.twitter.visibility.models.SafetyLevel.DirectMessages

import com.twitter.visibility.models.SafetyLevel

import com.twitter.visibility.models.ViewerContext

import com.twitter.visibility.rules.Drop

import com.twitter.visibility.rules.Reason.DeactivatedAuthor

import com.twitter.visibility.rules.Reason.ErasedAuthor

import com.twitter.visibility.rules.Reason.Nsfw

object DmVisibilityLibrary {

type Type = DmVisibilityRequest => Stitch[DmVisibilityResponse]

case class DmVisibilityRequest(

dmId: DmId,

dmAuthorUserId: UserId,

viewerContext: ViewerContext)

case class DmVisibilityResponse(isMessageNsfw: Boolean)

val DefaultSafetyLevel: SafetyLevel = DirectMessages

def apply(

visibilityLibrary: VisibilityLibrary,

stratoClient: StratoClient,

userSource: UserSource,

enableVfFeatureHydrationInShim: Gate[Unit] = Gate.False

): Type = {

val libraryStatsReceiver = visibilityLibrary.statsReceiver

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

val authorFeatures = new AuthorFeatures(userSource, libraryStatsReceiver)

{ r: DmVisibilityRequest =>

vfEngineCounter.incr()

val contentId = DmContentId(r.dmId)

val dmAuthorUserId = r.dmAuthorUserId

val isVfFeatureHydrationEnabled = enableVfFeatureHydrationInShim()

val featureMap =

visibilityLibrary.featureMapBuilder(

Seq(authorFeatures.forAuthorId(dmAuthorUserId))

)

val resp = if (isVfFeatureHydrationEnabled) {

FeatureMap.resolve(featureMap, libraryStatsReceiver).flatMap { resolvedFeatureMap =>

visibilityLibrary.runRuleEngine(

contentId,

resolvedFeatureMap,

r.viewerContext,

DefaultSafetyLevel

)

}

} else {

visibilityLibrary

.runRuleEngine(

contentId,

featureMap,

r.viewerContext,

DefaultSafetyLevel

)

}

resp.map(buildResponse)

}

}

private[this] def buildResponse(visibilityResult: VisibilityResult) =

visibilityResult.verdict match {

case Drop(Nsfw | ErasedAuthor | DeactivatedAuthor, \_) =>

DmVisibilityResponse(isMessageNsfw = true)

case \_ =>

DmVisibilityResponse(isMessageNsfw = false)

}

}