package com.twitter.visibility.interfaces.tweets

import com.twitter.decider.Decider

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.stitch.Stitch

import com.twitter.visibility.VisibilityLibrary

import com.twitter.visibility.builder.VisibilityResult

import com.twitter.visibility.builder.users.UserUnavailableFeatures

import com.twitter.visibility.common.actions.converter.scala.DropReasonConverter

import com.twitter.visibility.configapi.configs.VisibilityDeciderGates

import com.twitter.visibility.features.TweetIsInnerQuotedTweet

import com.twitter.visibility.features.TweetIsRetweet

import com.twitter.visibility.generators.LocalizedInterstitialGenerator

import com.twitter.visibility.generators.TombstoneGenerator

import com.twitter.visibility.models.ContentId.UserUnavailableState

import com.twitter.visibility.models.UserUnavailableStateEnum

import com.twitter.visibility.rules.Drop

import com.twitter.visibility.rules.Interstitial

import com.twitter.visibility.rules.Reason

import com.twitter.visibility.rules.Tombstone

import com.twitter.visibility.thriftscala.UserVisibilityResult

object UserUnavailableStateVisibilityLibrary {

type Type = UserUnavailableStateVisibilityRequest => Stitch[VisibilityResult]

def apply(

visibilityLibrary: VisibilityLibrary,

decider: Decider,

tombstoneGenerator: TombstoneGenerator,

interstitialGenerator: LocalizedInterstitialGenerator

): Type = {

val libraryStatsReceiver = visibilityLibrary.statsReceiver.scope("user\_unavailable\_vis\_library")

val defaultDropScope = visibilityLibrary.statsReceiver.scope("default\_drop")

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

val userUnavailableFeatures = UserUnavailableFeatures(libraryStatsReceiver)

val visibilityDeciderGates = VisibilityDeciderGates(decider)

{ r: UserUnavailableStateVisibilityRequest =>

vfEngineCounter.incr()

val contentId = UserUnavailableState(r.tweetId)

val featureMap =

visibilityLibrary.featureMapBuilder(

Seq(

\_.withConstantFeature(TweetIsInnerQuotedTweet, r.isInnerQuotedTweet),

\_.withConstantFeature(TweetIsRetweet, r.isRetweet),

userUnavailableFeatures.forState(r.userUnavailableState)

)

)

val language = r.viewerContext.requestLanguageCode.getOrElse("en")

val reason = visibilityLibrary

.runRuleEngine(

contentId,

featureMap,

r.viewerContext,

r.safetyLevel

).map(defaultToDrop(r.userUnavailableState, defaultDropScope))

.map(tombstoneGenerator(\_, language))

.map(visibilityResult => {

if (visibilityDeciderGates.enableLocalizedInterstitialInUserStateLibrary()) {

interstitialGenerator(visibilityResult, language)

} else {

visibilityResult

}

})

reason

}

}

def defaultToDrop(

userUnavailableState: UserUnavailableStateEnum,

defaultDropScope: StatsReceiver

)(

result: VisibilityResult

): VisibilityResult =

result.verdict match {

case \_: Drop | \_: Tombstone => result

case \_: Interstitial => result

case \_ =>

result.copy(verdict =

Drop(userUnavailableStateToDropReason(userUnavailableState, defaultDropScope)))

}

private[this] def userUnavailableStateToDropReason(

userUnavailableState: UserUnavailableStateEnum,

stats: StatsReceiver

): Reason =

userUnavailableState match {

case UserUnavailableStateEnum.Erased =>

stats.counter("erased").incr()

Reason.ErasedAuthor

case UserUnavailableStateEnum.Protected =>

stats.counter("protected").incr()

Reason.ProtectedAuthor

case UserUnavailableStateEnum.Offboarded =>

stats.counter("offboarded").incr()

Reason.OffboardedAuthor

case UserUnavailableStateEnum.AuthorBlocksViewer =>

stats.counter("author\_blocks\_viewer").incr()

Reason.AuthorBlocksViewer

case UserUnavailableStateEnum.Suspended =>

stats.counter("suspended\_author").incr()

Reason.SuspendedAuthor

case UserUnavailableStateEnum.Deactivated =>

stats.counter("deactivated\_author").incr()

Reason.DeactivatedAuthor

case UserUnavailableStateEnum.Filtered(result) =>

stats.counter("filtered").incr()

userVisibilityResultToDropReason(result, stats.scope("filtered"))

case UserUnavailableStateEnum.Unavailable =>

stats.counter("unspecified").incr()

Reason.Unspecified

case \_ =>

stats.counter("unknown").incr()

stats.scope("unknown").counter(userUnavailableState.name).incr()

Reason.Unspecified

}

private[this] def userVisibilityResultToDropReason(

result: UserVisibilityResult,

stats: StatsReceiver

): Reason =

result.action

.flatMap(DropReasonConverter.fromAction)

.map { dropReason =>

val reason = Reason.fromDropReason(dropReason)

stats.counter(reason.name).incr()

reason

}.getOrElse {

stats.counter("empty")

Reason.Unspecified

}

}