package com.twitter.visibility.interfaces.tweets

import com.twitter.decider.Decider

import com.twitter.servo.util.Gate

import com.twitter.stitch.Stitch

import com.twitter.visibility.VisibilityLibrary

import com.twitter.visibility.builder.VisibilityResult

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

import com.twitter.visibility.builder.users.QuotedTweetFeatures

import com.twitter.visibility.builder.users.RelationshipFeatures

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

import com.twitter.visibility.common.UserRelationshipSource

import com.twitter.visibility.common.UserSource

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

import com.twitter.visibility.features.FeatureMap

import com.twitter.visibility.models.ContentId.QuotedTweetRelationship

import com.twitter.visibility.models.SafetyLevel

import com.twitter.visibility.models.UserUnavailableStateEnum

import com.twitter.visibility.models.ViewerContext

import com.twitter.visibility.rules.Drop

import com.twitter.visibility.rules.EvaluationContext

import com.twitter.visibility.rules.Reason.AuthorBlocksViewer

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

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

import com.twitter.visibility.rules.Reason.OffboardedAuthor

import com.twitter.visibility.rules.Reason.ProtectedAuthor

import com.twitter.visibility.rules.Reason.SuspendedAuthor

import com.twitter.visibility.rules.Reason.ViewerBlocksAuthor

import com.twitter.visibility.rules.Reason.ViewerHardMutedAuthor

import com.twitter.visibility.rules.Reason.ViewerMutesAuthor

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

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

case class TweetAndAuthor(tweetId: Long, authorId: Long)

case class QuotedTweetVisibilityRequest(

quotedTweet: TweetAndAuthor,

outerTweet: TweetAndAuthor,

viewerContext: ViewerContext,

safetyLevel: SafetyLevel)

object QuotedTweetVisibilityLibrary {

type Type = QuotedTweetVisibilityRequest => Stitch[VisibilityResult]

def apply(

visibilityLibrary: VisibilityLibrary,

userSource: UserSource,

userRelationshipSource: UserRelationshipSource,

decider: Decider,

userStateVisibilityLibrary: UserUnavailableStateVisibilityLibrary.Type,

enableVfFeatureHydration: Gate[Unit] = Gate.False

): Type = {

val libraryStatsReceiver = visibilityLibrary.statsReceiver

val visibilityDeciderGates = VisibilityDeciderGates(decider)

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

{

case QuotedTweetVisibilityRequest(quotedTweet, outerTweet, viewerContext, safetyLevel) =>

vfEngineCounter.incr()

val contentId = QuotedTweetRelationship(

outer = outerTweet.tweetId,

inner = quotedTweet.tweetId

)

val innerAuthorId = quotedTweet.authorId

val outerAuthorId = outerTweet.authorId

val viewerId = viewerContext.userId

val isFeatureHydrationInShimEnabled = enableVfFeatureHydration()

val authorFeatures = new AuthorFeatures(userSource, libraryStatsReceiver)

val viewerFeatures = new ViewerFeatures(userSource, libraryStatsReceiver)

val relationshipFeatures =

new RelationshipFeatures(userRelationshipSource, libraryStatsReceiver)

val quotedTweetFeatures =

new QuotedTweetFeatures(relationshipFeatures, libraryStatsReceiver)

val featureMap = visibilityLibrary.featureMapBuilder(

Seq(

viewerFeatures.forViewerContext(viewerContext),

authorFeatures.forAuthorId(innerAuthorId),

relationshipFeatures.forAuthorId(innerAuthorId, viewerId),

quotedTweetFeatures.forOuterAuthor(outerAuthorId, innerAuthorId)

)

)

val resp = if (isFeatureHydrationInShimEnabled) {

val evaluationContext = ProvidedEvaluationContext.injectRuntimeRulesIntoEvaluationContext(

evaluationContext = EvaluationContext(

SafetyLevel.QuotedTweetRules,

visibilityLibrary.getParams(viewerContext, SafetyLevel.QuotedTweetRules),

visibilityLibrary.statsReceiver)

)

val preFilteredFeatureMap =

ShimUtils.preFilterFeatureMap(

featureMap,

SafetyLevel.QuotedTweetRules,

contentId,

evaluationContext)

FeatureMap.resolve(preFilteredFeatureMap, libraryStatsReceiver).flatMap {

resolvedFeatureMap =>

visibilityLibrary

.runRuleEngine(

contentId,

resolvedFeatureMap,

viewerContext,

SafetyLevel.QuotedTweetRules

)

}

} else {

visibilityLibrary

.runRuleEngine(

contentId,

featureMap,

viewerContext,

SafetyLevel.QuotedTweetRules

)

}

resp.flatMap { visResult =>

val userStateOpt = visResult.verdict match {

case Drop(DeactivatedAuthor, \_) => Some(UserUnavailableStateEnum.Deactivated)

case Drop(OffboardedAuthor, \_) => Some(UserUnavailableStateEnum.Offboarded)

case Drop(ErasedAuthor, \_) => Some(UserUnavailableStateEnum.Erased)

case Drop(ProtectedAuthor, \_) => Some(UserUnavailableStateEnum.Protected)

case Drop(SuspendedAuthor, \_) => Some(UserUnavailableStateEnum.Suspended)

case Drop(AuthorBlocksViewer, \_) => Some(UserUnavailableStateEnum.AuthorBlocksViewer)

case Drop(ViewerBlocksAuthor, \_) => Some(UserUnavailableStateEnum.ViewerBlocksAuthor)

case Drop(ViewerMutesAuthor, \_) => Some(UserUnavailableStateEnum.ViewerMutesAuthor)

case Drop(ViewerHardMutedAuthor, \_) => Some(UserUnavailableStateEnum.ViewerMutesAuthor)

case \_ => None

}

userStateOpt

.map(userState =>

userStateVisibilityLibrary(

UserUnavailableStateVisibilityRequest(

safetyLevel,

quotedTweet.tweetId,

viewerContext,

userState,

isRetweet = false,

isInnerQuotedTweet = true,

))).getOrElse(Stitch.value(visResult))

}

}

}

}