package com.twitter.visibility.interfaces.blender

import com.twitter.decider.Decider

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.mediaservices.media\_util.GenericMediaKey

import com.twitter.servo.util.Gate

import com.twitter.stitch.Stitch

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

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.util.Stopwatch

import com.twitter.visibility.VisibilityLibrary

import com.twitter.visibility.builder.VerdictLogger

import com.twitter.visibility.builder.VisibilityResult

import com.twitter.visibility.builder.media.MediaFeatures

import com.twitter.visibility.builder.media.StratoMediaLabelMaps

import com.twitter.visibility.builder.tweets.\_

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

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

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

import com.twitter.visibility.common.MediaSafetyLabelMapSource

import com.twitter.visibility.common.MisinformationPolicySource

import com.twitter.visibility.common.SafetyLabelMapSource

import com.twitter.visibility.common.TrustedFriendsSource

import com.twitter.visibility.common.UserRelationshipSource

import com.twitter.visibility.common.UserSource

import com.twitter.visibility.rules.ComposableActions.ComposableActionsWithInterstitial

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

import com.twitter.visibility.features.FeatureMap

import com.twitter.visibility.features.TweetIsInnerQuotedTweet

import com.twitter.visibility.features.TweetIsRetweet

import com.twitter.visibility.features.TweetIsSourceTweet

import com.twitter.visibility.logging.thriftscala.VFLibType

import com.twitter.visibility.models.ContentId

import com.twitter.visibility.models.ContentId.BlenderTweetId

import com.twitter.visibility.models.ContentId.TweetId

import com.twitter.visibility.models.SafetyLevel

import com.twitter.visibility.models.SafetyLevel.toThrift

import com.twitter.visibility.rules.Action

import com.twitter.visibility.rules.Allow

import com.twitter.visibility.rules.Drop

import com.twitter.visibility.rules.Interstitial

import com.twitter.visibility.rules.TweetInterstitial

object TweetType extends Enumeration {

type TweetType = Value

val ORIGINAL, SOURCE, QUOTED = Value

}

import com.twitter.visibility.interfaces.blender.TweetType.\_

object BlenderVisibilityLibrary {

def buildWithStratoClient(

visibilityLibrary: VisibilityLibrary,

decider: Decider,

stratoClient: StratoClient,

userSource: UserSource,

userRelationshipSource: UserRelationshipSource

): BlenderVisibilityLibrary = new BlenderVisibilityLibrary(

visibilityLibrary,

decider,

stratoClient,

userSource,

userRelationshipSource,

None

)

def buildWithSafetyLabelMapSource(

visibilityLibrary: VisibilityLibrary,

decider: Decider,

stratoClient: StratoClient,

userSource: UserSource,

userRelationshipSource: UserRelationshipSource,

safetyLabelMapSource: SafetyLabelMapSource

): BlenderVisibilityLibrary = new BlenderVisibilityLibrary(

visibilityLibrary,

decider,

stratoClient,

userSource,

userRelationshipSource,

Some(safetyLabelMapSource)

)

def createVerdictLogger(

enableVerdictLogger: Gate[Unit],

decider: Decider,

statsReceiver: StatsReceiver

): VerdictLogger = {

if (enableVerdictLogger()) {

VerdictLogger(statsReceiver, decider)

} else {

VerdictLogger.Empty

}

}

def scribeVisibilityVerdict(

result: CombinedVisibilityResult,

enableVerdictScribing: Gate[Unit],

verdictLogger: VerdictLogger,

viewerId: Option[Long],

safetyLevel: SafetyLevel

): Unit = if (enableVerdictScribing()) {

verdictLogger.scribeVerdict(

visibilityResult = result.tweetVisibilityResult,

viewerId = viewerId,

safetyLevel = toThrift(safetyLevel),

vfLibType = VFLibType.BlenderVisibilityLibrary)

result.quotedTweetVisibilityResult.map(quotedTweetVisibilityResult =>

verdictLogger.scribeVerdict(

visibilityResult = quotedTweetVisibilityResult,

viewerId = viewerId,

safetyLevel = toThrift(safetyLevel),

vfLibType = VFLibType.BlenderVisibilityLibrary))

}

}

class BlenderVisibilityLibrary(

visibilityLibrary: VisibilityLibrary,

decider: Decider,

stratoClient: StratoClient,

userSource: UserSource,

userRelationshipSource: UserRelationshipSource,

safetyLabelMapSourceOption: Option[SafetyLabelMapSource]) {

val libraryStatsReceiver = visibilityLibrary.statsReceiver

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

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

val bvlRequestCounter = libraryStatsReceiver.counter("bvl\_requests")

val vfLatencyOverallStat = libraryStatsReceiver.stat("vf\_latency\_overall")

val vfLatencyStitchBuildStat = libraryStatsReceiver.stat("vf\_latency\_stitch\_build")

val vfLatencyStitchRunStat = libraryStatsReceiver.stat("vf\_latency\_stitch\_run")

val visibilityDeciderGates = VisibilityDeciderGates(decider)

val verdictLogger = BlenderVisibilityLibrary.createVerdictLogger(

visibilityDeciderGates.enableVerdictLoggerBVL,

decider,

libraryStatsReceiver)

val tweetLabels = safetyLabelMapSourceOption match {

case Some(safetyLabelMapSource) =>

new StratoTweetLabelMaps(safetyLabelMapSource)

case None =>

new StratoTweetLabelMaps(

SafetyLabelMapSource.fromStrato(stratoClient, stratoClientStatsReceiver))

}

val mediaLabelMaps = new StratoMediaLabelMaps(

MediaSafetyLabelMapSource.fromStrato(stratoClient, stratoClientStatsReceiver))

val tweetFeatures = new TweetFeatures(tweetLabels, libraryStatsReceiver)

val blenderContextFeatures = new BlenderContextFeatures(libraryStatsReceiver)

val authorFeatures = new AuthorFeatures(userSource, libraryStatsReceiver)

val viewerFeatures = new ViewerFeatures(userSource, libraryStatsReceiver)

val relationshipFeatures =

new RelationshipFeatures(userRelationshipSource, libraryStatsReceiver)

val fonsrRelationshipFeatures =

new FosnrRelationshipFeatures(

tweetLabels = tweetLabels,

userRelationshipSource = userRelationshipSource,

statsReceiver = libraryStatsReceiver)

val misinfoPolicySource =

MisinformationPolicySource.fromStrato(stratoClient, stratoClientStatsReceiver)

val misinfoPolicyFeatures =

new MisinformationPolicyFeatures(misinfoPolicySource, stratoClientStatsReceiver)

val exclusiveTweetFeatures =

new ExclusiveTweetFeatures(userRelationshipSource, libraryStatsReceiver)

val mediaFeatures = new MediaFeatures(mediaLabelMaps, libraryStatsReceiver)

val trustedFriendsTweetFeatures = new TrustedFriendsFeatures(

trustedFriendsSource = TrustedFriendsSource.fromStrato(stratoClient, stratoClientStatsReceiver))

val editTweetFeatures = new EditTweetFeatures(libraryStatsReceiver)

def getCombinedVisibilityResult(

bvRequest: BlenderVisibilityRequest

): Stitch[CombinedVisibilityResult] = {

val elapsed = Stopwatch.start()

bvlRequestCounter.incr()

val (

requestTweetVisibilityResult,

quotedTweetVisibilityResultOption,

sourceTweetVisibilityResultOption

) = getAllVisibilityResults(bvRequest: BlenderVisibilityRequest)

val response: Stitch[CombinedVisibilityResult] = {

(

requestTweetVisibilityResult,

quotedTweetVisibilityResultOption,

sourceTweetVisibilityResultOption) match {

case (requestTweetVisResult, Some(quotedTweetVisResult), Some(sourceTweetVisResult)) => {

Stitch

.join(

requestTweetVisResult,

quotedTweetVisResult,

sourceTweetVisResult

).map {

case (requestTweetVisResult, quotedTweetVisResult, sourceTweetVisResult) => {

requestTweetVisResult.verdict match {

case Allow =>

CombinedVisibilityResult(sourceTweetVisResult, Some(quotedTweetVisResult))

case \_ =>

CombinedVisibilityResult(requestTweetVisResult, Some(quotedTweetVisResult))

}

}

}

}

case (requestTweetVisResult, None, Some(sourceTweetVisResult)) => {

Stitch

.join(

requestTweetVisResult,

sourceTweetVisResult

).map {

case (requestTweetVisResult, sourceTweetVisResult) => {

requestTweetVisResult.verdict match {

case Allow =>

CombinedVisibilityResult(sourceTweetVisResult, None)

case \_ =>

CombinedVisibilityResult(requestTweetVisResult, None)

}

}

}

}

case (requestTweetVisResult, Some(quotedTweetVisResult), None) => {

Stitch

.join(

requestTweetVisResult,

quotedTweetVisResult

).map {

case (requestTweetVisResult, quotedTweetVisResult) => {

CombinedVisibilityResult(requestTweetVisResult, Some(quotedTweetVisResult))

}

}

}

case (requestTweetVisResult, None, None) => {

requestTweetVisResult.map {

CombinedVisibilityResult(\_, None)

}

}

}

}

val runStitchStartMs = elapsed().inMilliseconds

val buildStitchStatMs = elapsed().inMilliseconds

vfLatencyStitchBuildStat.add(buildStitchStatMs)

response

.onSuccess(\_ => {

val overallMs = elapsed().inMilliseconds

vfLatencyOverallStat.add(overallMs)

val stitchRunMs = elapsed().inMilliseconds - runStitchStartMs

vfLatencyStitchRunStat.add(stitchRunMs)

})

.onSuccess(

BlenderVisibilityLibrary.scribeVisibilityVerdict(

\_,

visibilityDeciderGates.enableVerdictScribingBVL,

verdictLogger,

bvRequest.viewerContext.userId,

bvRequest.safetyLevel))

}

def getContentId(viewerId: Option[Long], authorId: Long, tweet: Tweet): ContentId = {

if (viewerId.contains(authorId))

TweetId(tweet.id)

else BlenderTweetId(tweet.id)

}

def getAllVisibilityResults(bvRequest: BlenderVisibilityRequest): (

Stitch[VisibilityResult],

Option[Stitch[VisibilityResult]],

Option[Stitch[VisibilityResult]]

) = {

val tweetContentId = getContentId(

viewerId = bvRequest.viewerContext.userId,

authorId = bvRequest.tweet.coreData.get.userId,

tweet = bvRequest.tweet)

val tweetFeatureMap =

buildFeatureMap(bvRequest, bvRequest.tweet, ORIGINAL)

vfEngineCounter.incr()

val requestTweetVisibilityResult = visibilityLibrary

.runRuleEngine(

tweetContentId,

tweetFeatureMap,

bvRequest.viewerContext,

bvRequest.safetyLevel

).map(handleComposableVisibilityResult)

val quotedTweetVisibilityResultOption = bvRequest.quotedTweet.map(quotedTweet => {

val quotedTweetContentId = getContentId(

viewerId = bvRequest.viewerContext.userId,

authorId = quotedTweet.coreData.get.userId,

tweet = quotedTweet)

val quotedInnerTweetFeatureMap =

buildFeatureMap(bvRequest, quotedTweet, QUOTED)

vfEngineCounter.incr()

visibilityLibrary

.runRuleEngine(

quotedTweetContentId,

quotedInnerTweetFeatureMap,

bvRequest.viewerContext,

bvRequest.safetyLevel

)

.map(handleComposableVisibilityResult)

.map(handleInnerQuotedTweetVisibilityResult)

})

val sourceTweetVisibilityResultOption = bvRequest.retweetSourceTweet.map(sourceTweet => {

val sourceTweetContentId = getContentId(

viewerId = bvRequest.viewerContext.userId,

authorId = sourceTweet.coreData.get.userId,

tweet = sourceTweet)

val sourceTweetFeatureMap =

buildFeatureMap(bvRequest, sourceTweet, SOURCE)

vfEngineCounter.incr()

visibilityLibrary

.runRuleEngine(

sourceTweetContentId,

sourceTweetFeatureMap,

bvRequest.viewerContext,

bvRequest.safetyLevel

)

.map(handleComposableVisibilityResult)

})

(

requestTweetVisibilityResult,

quotedTweetVisibilityResultOption,

sourceTweetVisibilityResultOption)

}

def buildFeatureMap(

bvRequest: BlenderVisibilityRequest,

tweet: Tweet,

tweetType: TweetType

): FeatureMap = {

val authorId = tweet.coreData.get.userId

val viewerId = bvRequest.viewerContext.userId

val isRetweet = if (tweetType.equals(ORIGINAL)) bvRequest.isRetweet else false

val isSourceTweet = tweetType.equals(SOURCE)

val isQuotedTweet = tweetType.equals(QUOTED)

val tweetMediaKeys: Seq[GenericMediaKey] = tweet.media

.getOrElse(Seq.empty)

.flatMap(\_.mediaKey.map(GenericMediaKey.apply))

visibilityLibrary.featureMapBuilder(

Seq(

viewerFeatures

.forViewerBlenderContext(bvRequest.blenderVFRequestContext, bvRequest.viewerContext),

relationshipFeatures.forAuthorId(authorId, viewerId),

fonsrRelationshipFeatures

.forTweetAndAuthorId(tweet = tweet, authorId = authorId, viewerId = viewerId),

tweetFeatures.forTweet(tweet),

mediaFeatures.forMediaKeys(tweetMediaKeys),

authorFeatures.forAuthorId(authorId),

blenderContextFeatures.forBlenderContext(bvRequest.blenderVFRequestContext),

\_.withConstantFeature(TweetIsRetweet, isRetweet),

misinfoPolicyFeatures.forTweet(tweet, bvRequest.viewerContext),

exclusiveTweetFeatures.forTweet(tweet, bvRequest.viewerContext),

trustedFriendsTweetFeatures.forTweet(tweet, viewerId),

editTweetFeatures.forTweet(tweet),

\_.withConstantFeature(TweetIsInnerQuotedTweet, isQuotedTweet),

\_.withConstantFeature(TweetIsSourceTweet, isSourceTweet),

)

)

}

def handleComposableVisibilityResult(result: VisibilityResult): VisibilityResult = {

if (result.secondaryVerdicts.nonEmpty) {

result.copy(verdict = composeActions(result.verdict, result.secondaryVerdicts))

} else {

result

}

}

private def composeActions(primary: Action, secondary: Seq[Action]): Action = {

if (primary.isComposable && secondary.nonEmpty) {

val actions = Seq[Action] { primary } ++ secondary

val interstitialOpt = Action.getFirstInterstitial(actions: \_\*)

val softInterventionOpt = Action.getFirstSoftIntervention(actions: \_\*)

val limitedEngagementsOpt = Action.getFirstLimitedEngagements(actions: \_\*)

val avoidOpt = Action.getFirstAvoid(actions: \_\*)

val numActions =

Seq[Option[\_]](interstitialOpt, softInterventionOpt, limitedEngagementsOpt, avoidOpt)

.count(\_.isDefined)

if (numActions > 1) {

TweetInterstitial(

interstitialOpt,

softInterventionOpt,

limitedEngagementsOpt,

None,

avoidOpt

)

} else {

primary

}

} else {

primary

}

}

def handleInnerQuotedTweetVisibilityResult(

result: VisibilityResult

): VisibilityResult = {

val newVerdict: Action =

result.verdict match {

case interstitial: Interstitial => Drop(interstitial.reason)

case ComposableActionsWithInterstitial(tweetInterstitial) => Drop(tweetInterstitial.reason)

case verdict => verdict

}

result.copy(verdict = newVerdict)

}

}