package com.twitter.visibility.interfaces.media

import com.twitter.stitch.Stitch

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

import com.twitter.util.Stopwatch

import com.twitter.visibility.VisibilityLibrary

import com.twitter.visibility.builder.VisibilityResult

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

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

import com.twitter.visibility.builder.media.MediaMetadataFeatures

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

import com.twitter.visibility.common.MediaMetadataSource

import com.twitter.visibility.common.MediaSafetyLabelMapSource

import com.twitter.visibility.common.UserSource

import com.twitter.visibility.features.FeatureMap

import com.twitter.visibility.generators.TombstoneGenerator

import com.twitter.visibility.models.ContentId.MediaId

import com.twitter.visibility.rules.EvaluationContext

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

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

object MediaVisibilityLibrary {

type Type = MediaVisibilityRequest => Stitch[VisibilityResult]

def apply(

visibilityLibrary: VisibilityLibrary,

userSource: UserSource,

tombstoneGenerator: TombstoneGenerator,

stratoClient: StratoClient,

): Type = {

val libraryStatsReceiver = visibilityLibrary.statsReceiver

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

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

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

val stratoClientStatsReceiver = libraryStatsReceiver.scope("strato")

val mediaMetadataFeatures = new MediaMetadataFeatures(

MediaMetadataSource.fromStrato(stratoClient, stratoClientStatsReceiver),

libraryStatsReceiver)

val mediaLabelMaps = new StratoMediaLabelMaps(

MediaSafetyLabelMapSource.fromStrato(stratoClient, stratoClientStatsReceiver))

val mediaFeatures = new MediaFeatures(mediaLabelMaps, libraryStatsReceiver)

val viewerFeatures = new ViewerFeatures(userSource, libraryStatsReceiver)

{ r: MediaVisibilityRequest =>

vfEngineCounter.incr()

val contentId = MediaId(r.mediaKey.toStringKey)

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

val featureMap = visibilityLibrary.featureMapBuilder(

Seq(

viewerFeatures.forViewerContext(r.viewerContext),

mediaFeatures.forGenericMediaKey(r.mediaKey),

mediaMetadataFeatures.forGenericMediaKey(r.mediaKey),

)

)

val evaluationContext = ProvidedEvaluationContext.injectRuntimeRulesIntoEvaluationContext(

evaluationContext = EvaluationContext(

r.safetyLevel,

visibilityLibrary.getParams(r.viewerContext, r.safetyLevel),

visibilityLibrary.statsReceiver)

)

val preFilteredFeatureMap =

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

val elapsed = Stopwatch.start()

FeatureMap.resolve(preFilteredFeatureMap, libraryStatsReceiver).flatMap {

resolvedFeatureMap =>

vfLatencyStitchRunStat.add(elapsed().inMilliseconds)

visibilityLibrary

.runRuleEngine(

contentId,

resolvedFeatureMap,

r.viewerContext,

r.safetyLevel

)

.map(tombstoneGenerator(\_, languageCode))

.onSuccess(\_ => vfLatencyOverallStat.add(elapsed().inMilliseconds))

}

}

}

}