package com.twitter.visibility.interfaces.cards

import com.twitter.appsec.sanitization.URLSafety

import com.twitter.decider.Decider

import com.twitter.servo.util.Gate

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.{thriftscala => tweetypiethrift}

import com.twitter.util.Stopwatch

import com.twitter.visibility.VisibilityLibrary

import com.twitter.visibility.builder.FeatureMapBuilder

import com.twitter.visibility.builder.VisibilityResult

import com.twitter.visibility.builder.tweets.CommunityTweetFeatures

import com.twitter.visibility.builder.tweets.CommunityTweetFeaturesV2

import com.twitter.visibility.builder.tweets.NilTweetLabelMaps

import com.twitter.visibility.builder.tweets.TweetFeatures

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.CommunitiesSource

import com.twitter.visibility.common.UserId

import com.twitter.visibility.common.UserRelationshipSource

import com.twitter.visibility.common.UserSource

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

import com.twitter.visibility.features.CardIsPoll

import com.twitter.visibility.features.CardUriHost

import com.twitter.visibility.features.FeatureMap

import com.twitter.visibility.models.ContentId.CardId

import com.twitter.visibility.models.ViewerContext

object CardVisibilityLibrary {

type Type = CardVisibilityRequest => Stitch[VisibilityResult]

private[this] def getAuthorFeatures(

authorIdOpt: Option[Long],

authorFeatures: AuthorFeatures

): FeatureMapBuilder => FeatureMapBuilder = {

authorIdOpt match {

case Some(authorId) => authorFeatures.forAuthorId(authorId)

case \_ => authorFeatures.forNoAuthor()

}

}

private[this] def getCardUriFeatures(

cardUri: String,

enableCardVisibilityLibraryCardUriParsing: Boolean,

trackCardUriHost: Option[String] => Unit

): FeatureMapBuilder => FeatureMapBuilder = {

if (enableCardVisibilityLibraryCardUriParsing) {

val safeCardUriHost = URLSafety.getHostSafe(cardUri)

trackCardUriHost(safeCardUriHost)

\_.withConstantFeature(CardUriHost, safeCardUriHost)

} else {

identity

}

}

private[this] def getRelationshipFeatures(

authorIdOpt: Option[Long],

viewerIdOpt: Option[Long],

relationshipFeatures: RelationshipFeatures

): FeatureMapBuilder => FeatureMapBuilder = {

authorIdOpt match {

case Some(authorId) => relationshipFeatures.forAuthorId(authorId, viewerIdOpt)

case \_ => relationshipFeatures.forNoAuthor()

}

}

private[this] def getTweetFeatures(

tweetOpt: Option[tweetypiethrift.Tweet],

tweetFeatures: TweetFeatures

): FeatureMapBuilder => FeatureMapBuilder = {

tweetOpt match {

case Some(tweet) => tweetFeatures.forTweet(tweet)

case \_ => identity

}

}

private[this] def getCommunityFeatures(

tweetOpt: Option[tweetypiethrift.Tweet],

viewerContext: ViewerContext,

communityTweetFeatures: CommunityTweetFeatures

): FeatureMapBuilder => FeatureMapBuilder = {

tweetOpt match {

case Some(tweet) => communityTweetFeatures.forTweet(tweet, viewerContext)

case \_ => identity

}

}

def apply(

visibilityLibrary: VisibilityLibrary,

userSource: UserSource = UserSource.empty,

userRelationshipSource: UserRelationshipSource = UserRelationshipSource.empty,

communitiesSource: CommunitiesSource = CommunitiesSource.empty,

enableVfParityTest: Gate[Unit] = Gate.False,

enableVfFeatureHydration: Gate[Unit] = Gate.False,

decider: Decider

): Type = {

val libraryStatsReceiver = visibilityLibrary.statsReceiver

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

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

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

val cardUriStats = libraryStatsReceiver.scope("card\_uri")

val visibilityDeciderGates = VisibilityDeciderGates(decider)

val authorFeatures = new AuthorFeatures(userSource, libraryStatsReceiver)

val viewerFeatures = new ViewerFeatures(userSource, libraryStatsReceiver)

val tweetFeatures = new TweetFeatures(NilTweetLabelMaps, libraryStatsReceiver)

val communityTweetFeatures = new CommunityTweetFeaturesV2(

communitiesSource = communitiesSource,

)

val relationshipFeatures =

new RelationshipFeatures(userRelationshipSource, libraryStatsReceiver)

val parityTest = new CardVisibilityLibraryParityTest(libraryStatsReceiver)

{ r: CardVisibilityRequest =>

val elapsed = Stopwatch.start()

var runStitchStartMs = 0L

val viewerId: Option[UserId] = r.viewerContext.userId

val featureMap =

visibilityLibrary

.featureMapBuilder(

Seq(

viewerFeatures.forViewerId(viewerId),

getAuthorFeatures(r.authorId, authorFeatures),

getCardUriFeatures(

cardUri = r.cardUri,

enableCardVisibilityLibraryCardUriParsing =

visibilityDeciderGates.enableCardVisibilityLibraryCardUriParsing(),

trackCardUriHost = { safeCardUriHost: Option[String] =>

if (safeCardUriHost.isEmpty) {

cardUriStats.counter("empty").incr()

}

}

),

getCommunityFeatures(r.tweetOpt, r.viewerContext, communityTweetFeatures),

getRelationshipFeatures(r.authorId, r.viewerContext.userId, relationshipFeatures),

getTweetFeatures(r.tweetOpt, tweetFeatures),

\_.withConstantFeature(CardIsPoll, r.isPollCardType)

)

)

val response = visibilityLibrary

.runRuleEngine(

CardId(r.cardUri),

featureMap,

r.viewerContext,

r.safetyLevel

)

.onSuccess(\_ => {

val overallStatMs = elapsed().inMilliseconds

vfLatencyOverallStat.add(overallStatMs)

val runStitchEndMs = elapsed().inMilliseconds

vfLatencyStitchRunStat.add(runStitchEndMs - runStitchStartMs)

})

runStitchStartMs = elapsed().inMilliseconds

val buildStitchStatMs = elapsed().inMilliseconds

vfLatencyStitchBuildStat.add(buildStitchStatMs)

lazy val hydratedFeatureResponse: Stitch[VisibilityResult] =

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

visibilityLibrary.runRuleEngine(

CardId(r.cardUri),

resolvedFeatureMap,

r.viewerContext,

r.safetyLevel

)

}

val isVfParityTestEnabled = enableVfParityTest()

val isVfFeatureHydrationEnabled = enableVfFeatureHydration()

if (!isVfParityTestEnabled && !isVfFeatureHydrationEnabled) {

response

} else if (isVfParityTestEnabled && !isVfFeatureHydrationEnabled) {

response.applyEffect { resp =>

Stitch.async(parityTest.runParityTest(hydratedFeatureResponse, resp))

}

} else {

hydratedFeatureResponse

}

}

}

}