package com.twitter.tweetypie

package handler

import com.twitter.finagle.tracing.Trace

import com.twitter.relevance.feature\_store.thriftscala.FeatureData

import com.twitter.relevance.feature\_store.thriftscala.FeatureValue

import com.twitter.service.gen.scarecrow.thriftscala.TieredAction

import com.twitter.service.gen.scarecrow.thriftscala.TieredActionResult

import com.twitter.service.gen.scarecrow.thriftscala.TweetContext

import com.twitter.service.gen.scarecrow.thriftscala.TweetNew

import com.twitter.spam.features.thriftscala.SafetyMetaData

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.TweetCreateFailure

import com.twitter.tweetypie.handler.Spam.Checker

import com.twitter.tweetypie.repository.TweetSpamCheckRepository

import com.twitter.tweetypie.thriftscala.TweetCreateState

import com.twitter.tweetypie.thriftscala.TweetMediaTags

case class TweetSpamRequest(

tweetId: TweetId,

userId: UserId,

text: String,

mediaTags: Option[TweetMediaTags],

safetyMetaData: Option[SafetyMetaData],

inReplyToTweetId: Option[TweetId],

quotedTweetId: Option[TweetId],

quotedTweetUserId: Option[UserId])

/\*\*

\* Use the Scarecrow service as the spam checker for tweets.

\*/

object ScarecrowTweetSpamChecker {

val log: Logger = Logger(getClass)

private def requestToScarecrowTweet(req: TweetSpamRequest): TweetNew = {

// compile additional input features for the spam check

val mediaTaggedUserIds = {

val mediaTags = req.mediaTags.getOrElse(TweetMediaTags())

mediaTags.tagMap.values.flatten.flatMap(\_.userId).toSet

}

val additionalInputFeatures = {

val mediaTaggedUserFeatures = if (mediaTaggedUserIds.nonEmpty) {

Seq(

"mediaTaggedUsers" -> FeatureData(Some(FeatureValue.LongSetValue(mediaTaggedUserIds))),

"victimIds" -> FeatureData(Some(FeatureValue.LongSetValue(mediaTaggedUserIds)))

)

} else {

Seq.empty

}

val quotedTweetIdFeature = req.quotedTweetId.map { quotedTweetId =>

"quotedTweetId" -> FeatureData(Some(FeatureValue.LongValue(quotedTweetId)))

}

val quotedTweetUserIdFeature = req.quotedTweetUserId.map { quotedTweetUserId =>

"quotedTweetUserId" -> FeatureData(Some(FeatureValue.LongValue(quotedTweetUserId)))

}

val featureMap =

(mediaTaggedUserFeatures ++ quotedTweetIdFeature ++ quotedTweetUserIdFeature).toMap

if (featureMap.nonEmpty) Some(featureMap) else None

}

TweetNew(

id = req.tweetId,

userId = req.userId,

text = req.text,

additionalInputFeatures = additionalInputFeatures,

safetyMetaData = req.safetyMetaData,

inReplyToStatusId = req.inReplyToTweetId

)

}

private def tieredActionHandler(stats: StatsReceiver): Checker[TieredAction] =

Spam.handleScarecrowResult(stats) {

case (TieredActionResult.NotSpam, \_, \_) => Spam.AllowFuture

case (TieredActionResult.SilentFail, \_, \_) => Spam.SilentFailFuture

case (TieredActionResult.DenyByIpiPolicy, \_, \_) => Spam.DisabledByIpiPolicyFuture

case (TieredActionResult.UrlSpam, \_, denyMessage) =>

Future.exception(TweetCreateFailure.State(TweetCreateState.UrlSpam, denyMessage))

case (TieredActionResult.Deny, \_, denyMessage) =>

Future.exception(TweetCreateFailure.State(TweetCreateState.Spam, denyMessage))

case (TieredActionResult.Captcha, \_, denyMessage) =>

Future.exception(TweetCreateFailure.State(TweetCreateState.SpamCaptcha, denyMessage))

case (TieredActionResult.RateLimit, \_, denyMessage) =>

Future.exception(

TweetCreateFailure.State(TweetCreateState.SafetyRateLimitExceeded, denyMessage))

case (TieredActionResult.Bounce, Some(b), \_) =>

Future.exception(TweetCreateFailure.Bounced(b))

}

def fromSpamCheckRepository(

stats: StatsReceiver,

repo: TweetSpamCheckRepository.Type

): Spam.Checker[TweetSpamRequest] = {

val handler = tieredActionHandler(stats)

req => {

Trace.record("com.twitter.tweetypie.ScarecrowTweetSpamChecker.userId=" + req.userId)

Stitch.run(repo(requestToScarecrowTweet(req), TweetContext.Creation)).flatMap { resp =>

handler(resp.tieredAction)

}

}

}

}