package com.twitter.visibility.builder.common

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.gizmoduck.thriftscala.MuteOption

import com.twitter.gizmoduck.thriftscala.MuteSurface

import com.twitter.gizmoduck.thriftscala.{MutedKeyword => GdMutedKeyword}

import com.twitter.servo.util.Gate

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.visibility.builder.FeatureMapBuilder

import com.twitter.visibility.common.\_

import com.twitter.visibility.features.\_

import com.twitter.visibility.models.{MutedKeyword => VfMutedKeyword}

import java.util.Locale

class MutedKeywordFeatures(

userSource: UserSource,

userRelationshipSource: UserRelationshipSource,

keywordMatcher: KeywordMatcher.Matcher = KeywordMatcher.TestMatcher,

statsReceiver: StatsReceiver,

enableFollowCheckInMutedKeyword: Gate[Unit] = Gate.False) {

private[this] val scopedStatsReceiver: StatsReceiver =

statsReceiver.scope("muted\_keyword\_features")

private[this] val requests = scopedStatsReceiver.counter("requests")

private[this] val viewerMutesKeywordInTweetForHomeTimeline =

scopedStatsReceiver.scope(ViewerMutesKeywordInTweetForHomeTimeline.name).counter("requests")

private[this] val viewerMutesKeywordInTweetForTweetReplies =

scopedStatsReceiver.scope(ViewerMutesKeywordInTweetForTweetReplies.name).counter("requests")

private[this] val viewerMutesKeywordInTweetForNotifications =

scopedStatsReceiver.scope(ViewerMutesKeywordInTweetForNotifications.name).counter("requests")

private[this] val excludeFollowingForMutedKeywordsRequests =

scopedStatsReceiver.scope("exclude\_following").counter("requests")

private[this] val viewerMutesKeywordInTweetForAllSurfaces =

scopedStatsReceiver.scope(ViewerMutesKeywordInTweetForAllSurfaces.name).counter("requests")

def forTweet(

tweet: Tweet,

viewerId: Option[Long],

authorId: Long

): FeatureMapBuilder => FeatureMapBuilder = { featureMapBuilder =>

requests.incr()

viewerMutesKeywordInTweetForHomeTimeline.incr()

viewerMutesKeywordInTweetForTweetReplies.incr()

viewerMutesKeywordInTweetForNotifications.incr()

viewerMutesKeywordInTweetForAllSurfaces.incr()

val keywordsBySurface = allMutedKeywords(viewerId)

val keywordsWithoutDefinedSurface = allMutedKeywordsWithoutDefinedSurface(viewerId)

featureMapBuilder

.withFeature(

ViewerMutesKeywordInTweetForHomeTimeline,

tweetContainsMutedKeyword(

tweet,

keywordsBySurface,

MuteSurface.HomeTimeline,

viewerId,

authorId

)

)

.withFeature(

ViewerMutesKeywordInTweetForTweetReplies,

tweetContainsMutedKeyword(

tweet,

keywordsBySurface,

MuteSurface.TweetReplies,

viewerId,

authorId

)

)

.withFeature(

ViewerMutesKeywordInTweetForNotifications,

tweetContainsMutedKeyword(

tweet,

keywordsBySurface,

MuteSurface.Notifications,

viewerId,

authorId

)

)

.withFeature(

ViewerMutesKeywordInTweetForAllSurfaces,

tweetContainsMutedKeywordWithoutDefinedSurface(

tweet,

keywordsWithoutDefinedSurface,

viewerId,

authorId

)

)

}

def allMutedKeywords(viewerId: Option[Long]): Stitch[Map[MuteSurface, Seq[GdMutedKeyword]]] =

viewerId

.map { id => userSource.getAllMutedKeywords(id) }.getOrElse(Stitch.value(Map.empty))

def allMutedKeywordsWithoutDefinedSurface(viewerId: Option[Long]): Stitch[Seq[GdMutedKeyword]] =

viewerId

.map { id => userSource.getAllMutedKeywordsWithoutDefinedSurface(id) }.getOrElse(

Stitch.value(Seq.empty))

private def mutingKeywordsText(

mutedKeywords: Seq[GdMutedKeyword],

muteSurface: MuteSurface,

viewerIdOpt: Option[Long],

authorId: Long

): Stitch[Option[String]] = {

if (muteSurface == MuteSurface.HomeTimeline && mutedKeywords.nonEmpty) {

Stitch.value(Some(mutedKeywords.map(\_.keyword).mkString(",")))

} else {

mutedKeywords.partition(kw =>

kw.muteOptions.contains(MuteOption.ExcludeFollowingAccounts)) match {

case (\_, mutedKeywordsFromAnyone) if mutedKeywordsFromAnyone.nonEmpty =>

Stitch.value(Some(mutedKeywordsFromAnyone.map(\_.keyword).mkString(",")))

case (mutedKeywordsExcludeFollowing, \_)

if mutedKeywordsExcludeFollowing.nonEmpty && enableFollowCheckInMutedKeyword() =>

excludeFollowingForMutedKeywordsRequests.incr()

viewerIdOpt match {

case Some(viewerId) =>

userRelationshipSource.follows(viewerId, authorId).map {

case true =>

case false => Some(mutedKeywordsExcludeFollowing.map(\_.keyword).mkString(","))

}

case \_ => Stitch.None

}

case (\_, \_) => Stitch.None

}

}

}

private def mutingKeywordsTextWithoutDefinedSurface(

mutedKeywords: Seq[GdMutedKeyword],

viewerIdOpt: Option[Long],

authorId: Long

): Stitch[Option[String]] = {

mutedKeywords.partition(kw =>

kw.muteOptions.contains(MuteOption.ExcludeFollowingAccounts)) match {

case (\_, mutedKeywordsFromAnyone) if mutedKeywordsFromAnyone.nonEmpty =>

Stitch.value(Some(mutedKeywordsFromAnyone.map(\_.keyword).mkString(",")))

case (mutedKeywordsExcludeFollowing, \_)

if mutedKeywordsExcludeFollowing.nonEmpty && enableFollowCheckInMutedKeyword() =>

excludeFollowingForMutedKeywordsRequests.incr()

viewerIdOpt match {

case Some(viewerId) =>

userRelationshipSource.follows(viewerId, authorId).map {

case true =>

case false => Some(mutedKeywordsExcludeFollowing.map(\_.keyword).mkString(","))

}

case \_ => Stitch.None

}

case (\_, \_) => Stitch.None

}

}

def tweetContainsMutedKeyword(

tweet: Tweet,

mutedKeywordMap: Stitch[Map[MuteSurface, Seq[GdMutedKeyword]]],

muteSurface: MuteSurface,

viewerIdOpt: Option[Long],

authorId: Long

): Stitch[VfMutedKeyword] = {

mutedKeywordMap.flatMap { keywordMap =>

if (keywordMap.isEmpty) {

Stitch.value(VfMutedKeyword(None))

} else {

val mutedKeywords = keywordMap.getOrElse(muteSurface, Nil)

val matchTweetFn: KeywordMatcher.MatchTweet = keywordMatcher(mutedKeywords)

val locale = tweet.language.map(l => Locale.forLanguageTag(l.language))

val text = tweet.coreData.get.text

matchTweetFn(locale, text).flatMap { results =>

mutingKeywordsText(results, muteSurface, viewerIdOpt, authorId).map(VfMutedKeyword)

}

}

}

}

def tweetContainsMutedKeywordWithoutDefinedSurface(

tweet: Tweet,

mutedKeywordSeq: Stitch[Seq[GdMutedKeyword]],

viewerIdOpt: Option[Long],

authorId: Long

): Stitch[VfMutedKeyword] = {

mutedKeywordSeq.flatMap { mutedKeyword =>

if (mutedKeyword.isEmpty) {

Stitch.value(VfMutedKeyword(None))

} else {

val matchTweetFn: KeywordMatcher.MatchTweet = keywordMatcher(mutedKeyword)

val locale = tweet.language.map(l => Locale.forLanguageTag(l.language))

val text = tweet.coreData.get.text

matchTweetFn(locale, text).flatMap { results =>

mutingKeywordsTextWithoutDefinedSurface(results, viewerIdOpt, authorId).map(

VfMutedKeyword

)

}

}

}

}

def spaceTitleContainsMutedKeyword(

spaceTitle: String,

spaceLanguageOpt: Option[String],

mutedKeywordMap: Stitch[Map[MuteSurface, Seq[GdMutedKeyword]]],

muteSurface: MuteSurface,

): Stitch[VfMutedKeyword] = {

mutedKeywordMap.flatMap { keywordMap =>

if (keywordMap.isEmpty) {

Stitch.value(VfMutedKeyword(None))

} else {

val mutedKeywords = keywordMap.getOrElse(muteSurface, Nil)

val matchTweetFn: KeywordMatcher.MatchTweet = keywordMatcher(mutedKeywords)

val locale = spaceLanguageOpt.map(l => Locale.forLanguageTag(l))

matchTweetFn(locale, spaceTitle).flatMap { results =>

if (results.nonEmpty) {

Stitch.value(Some(results.map(\_.keyword).mkString(","))).map(VfMutedKeyword)

} else {

Stitch.None.map(VfMutedKeyword)

}

}

}

}

}

}