package com.twitter.frigate.pushservice.predicate

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.\_

import com.twitter.frigate.pushservice.model.PushTypes.PushCandidate

import com.twitter.frigate.thriftscala.\_

import com.twitter.hermit.predicate.NamedPredicate

import com.twitter.hermit.predicate.scarecrow.{ScarecrowPredicate => HermitScarecrowPredicate}

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

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

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

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

import com.twitter.storehaus.ReadableStore

object ScarecrowPredicate {

val name = ""

def candidateToEvent(candidate: PushCandidate): Event = {

val recommendedUserIdOpt = candidate match {

case tweetCandidate: TweetCandidate with TweetAuthor =>

tweetCandidate.authorId

case userCandidate: UserCandidate =>

Some(userCandidate.userId)

case \_ => None

}

val hashtagsInTweet = candidate match {

case tweetCandidate: TweetCandidate with TweetDetails =>

tweetCandidate.tweetyPieResult

.flatMap { tweetPieResult =>

tweetPieResult.tweet.hashtags.map(\_.map(\_.text))

}.getOrElse(Nil)

case \_ =>

Nil

}

val urlsInTweet = candidate match {

case tweetCandidate: TweetCandidate with TweetDetails =>

tweetCandidate.tweetyPieResult

.flatMap { tweetPieResult =>

tweetPieResult.tweet.urls.map(\_.flatMap(\_.expanded))

}

case \_ => None

}

val tweetIdOpt = candidate match {

case tweetCandidate: TweetCandidate =>

Some(tweetCandidate.tweetId)

case \_ =>

None

}

val urlOpt = candidate match {

case candidate: UrlCandidate =>

Some(candidate.url)

case \_ =>

None

}

val scUserIds = candidate match {

case hasSocialContext: SocialContextActions => Some(hasSocialContext.socialContextUserIds)

case \_ => None

}

val eventTitleOpt = candidate match {

case eventCandidate: EventCandidate with EventDetails =>

Some(eventCandidate.eventTitle)

case \_ =>

None

}

val urlTitleOpt = candidate match {

case candidate: UrlCandidate =>

candidate.title

case \_ =>

None

}

val urlDescriptionOpt = candidate match {

case candidate: UrlCandidate with UrlCandidateWithDetails =>

candidate.description

case \_ =>

None

}

Event(

"magicrecs\_recommendation\_write",

Map(

"targetUserId" -> FeatureData(Some(FeatureValue.LongValue(candidate.target.targetId))),

"type" -> FeatureData(

Some(

FeatureValue.StrValue(candidate.commonRecType.name)

)

),

"recommendedUserId" -> FeatureData(recommendedUserIdOpt map { id =>

FeatureValue.LongValue(id)

}),

"tweetId" -> FeatureData(tweetIdOpt map { id =>

FeatureValue.LongValue(id)

}),

"url" -> FeatureData(urlOpt map { url =>

FeatureValue.StrValue(url)

}),

"hashtagsInTweet" -> FeatureData(Some(FeatureValue.StrListValue(hashtagsInTweet))),

"urlsInTweet" -> FeatureData(urlsInTweet.map(FeatureValue.StrListValue)),

"socialContexts" -> FeatureData(scUserIds.map { sc =>

FeatureValue.LongListValue(sc)

}),

"eventTitle" -> FeatureData(eventTitleOpt.map { eventTitle =>

FeatureValue.StrValue(eventTitle)

}),

"urlTitle" -> FeatureData(urlTitleOpt map { title =>

FeatureValue.StrValue(title)

}),

"urlDescription" -> FeatureData(urlDescriptionOpt map { des =>

FeatureValue.StrValue(des)

})

)

)

}

def candidateToPossibleEvent(c: PushCandidate): Option[Event] = {

if (c.frigateNotification.notificationDisplayLocation == NotificationDisplayLocation.PushToMobileDevice) {

Some(candidateToEvent(c))

} else {

None

}

}

def apply(

scarecrowCheckEventStore: ReadableStore[Event, TieredActionResult]

)(

implicit statsReceiver: StatsReceiver

): NamedPredicate[PushCandidate] = {

HermitScarecrowPredicate(scarecrowCheckEventStore)

.optionalOn(

candidateToPossibleEvent,

missingResult = true

)

.withStats(statsReceiver.scope(s"predicate\_$name"))

.withName(name)

}

}