package com.twitter.frigate.pushservice.predicate

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.pushservice.model.ListRecommendationPushCandidate

import com.twitter.frigate.pushservice.params.PushFeatureSwitchParams

import com.twitter.hermit.predicate.socialgraph.Edge

import com.twitter.hermit.predicate.socialgraph.RelationEdge

import com.twitter.hermit.predicate.socialgraph.SocialGraphPredicate

import com.twitter.hermit.predicate.NamedPredicate

import com.twitter.hermit.predicate.Predicate

import com.twitter.socialgraph.thriftscala.RelationshipType

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

object ListPredicates {

def listNameExistsPredicate(

)(

implicit stats: StatsReceiver

): NamedPredicate[ListRecommendationPushCandidate] = {

Predicate

.fromAsync { candidate: ListRecommendationPushCandidate =>

candidate.listName.map(\_.isDefined)

}

.withStats(stats)

.withName("list\_name\_exists")

}

def listAuthorExistsPredicate(

)(

implicit stats: StatsReceiver

): NamedPredicate[ListRecommendationPushCandidate] = {

Predicate

.fromAsync { candidate: ListRecommendationPushCandidate =>

candidate.listOwnerId.map(\_.isDefined)

}

.withStats(stats)

.withName("list\_owner\_exists")

}

def listAuthorAcceptableToTargetUser(

edgeStore: ReadableStore[RelationEdge, Boolean]

)(

implicit statsReceiver: StatsReceiver

): NamedPredicate[ListRecommendationPushCandidate] = {

val name = "list\_author\_acceptable\_to\_target\_user"

val sgsPredicate = SocialGraphPredicate

.anyRelationExists(

edgeStore,

Set(

RelationshipType.Blocking,

RelationshipType.BlockedBy,

RelationshipType.Muting

)

)

.withStats(statsReceiver.scope("list\_sgs\_any\_relation\_exists"))

.withName("list\_sgs\_any\_relation\_exists")

Predicate

.fromAsync { candidate: ListRecommendationPushCandidate =>

candidate.listOwnerId.flatMap {

case Some(ownerId) =>

sgsPredicate.apply(Seq(Edge(candidate.target.targetId, ownerId))).map(\_.head)

case \_ => Future.True

}

}

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

.withName(name)

}

/\*\*

\* Checks if the list is acceptable to Target user =>

\* - Is Target not following the list

\* - Is Target not muted the list

\*/

def listAcceptablePredicate(

)(

implicit stats: StatsReceiver

): NamedPredicate[ListRecommendationPushCandidate] = {

val name = "list\_acceptable\_to\_target\_user"

Predicate

.fromAsync { candidate: ListRecommendationPushCandidate =>

candidate.apiList.map {

case Some(apiList) =>

!(apiList.following.contains(true) || apiList.muting.contains(true))

case \_ => false

}

}

.withStats(stats.scope(name))

.withName(name)

}

def listSubscriberCountPredicate(

)(

implicit stats: StatsReceiver

): NamedPredicate[ListRecommendationPushCandidate] = {

val name = "list\_subscribe\_count"

Predicate

.fromAsync { candidate: ListRecommendationPushCandidate =>

candidate.apiList.map { apiListOpt =>

apiListOpt.exists { apiList =>

apiList.subscriberCount >= candidate.target.params(

PushFeatureSwitchParams.ListRecommendationsSubscriberCount)

}

}

}

.withStats(stats.scope(name))

.withName(name)

}

}