package com.twitter.follow\_recommendations.common.predicates.sgs

import com.google.common.annotations.VisibleForTesting

import com.twitter.finagle.stats.NullStatsReceiver

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.base.Predicate

import com.twitter.follow\_recommendations.common.base.PredicateResult

import com.twitter.follow\_recommendations.common.models.CandidateUser

import com.twitter.follow\_recommendations.common.models.HasProfileId

import com.twitter.follow\_recommendations.common.models.FilterReason.FailOpen

import com.twitter.follow\_recommendations.common.models.FilterReason.InvalidRelationshipTypes

import com.twitter.product\_mixer.core.model.marshalling.request.HasClientContext

import com.twitter.socialgraph.thriftscala.ExistsRequest

import com.twitter.socialgraph.thriftscala.ExistsResult

import com.twitter.socialgraph.thriftscala.LookupContext

import com.twitter.socialgraph.thriftscala.Relationship

import com.twitter.socialgraph.thriftscala.RelationshipType

import com.twitter.stitch.Stitch

import com.twitter.stitch.socialgraph.SocialGraph

import com.twitter.timelines.configapi.HasParams

import com.twitter.util.TimeoutException

import com.twitter.util.logging.Logging

import javax.inject.Inject

import javax.inject.Singleton

case class RelationshipMapping(

relationshipType: RelationshipType,

includeBasedOnRelationship: Boolean)

class SgsRelationshipsPredicate(

socialGraph: SocialGraph,

relationshipMappings: Seq[RelationshipMapping],

statsReceiver: StatsReceiver = NullStatsReceiver)

extends Predicate[(HasClientContext with HasParams, CandidateUser)]

with Logging {

private val stats: StatsReceiver = statsReceiver.scope(this.getClass.getSimpleName)

override def apply(

pair: (HasClientContext with HasParams, CandidateUser)

): Stitch[PredicateResult] = {

val (target, candidate) = pair

val timeout = target.params(SgsPredicateParams.SgsRelationshipsPredicateTimeout)

SgsRelationshipsPredicate

.extractUserId(target)

.map { id =>

val relationships = relationshipMappings.map { relationshipMapping: RelationshipMapping =>

Relationship(

relationshipMapping.relationshipType,

relationshipMapping.includeBasedOnRelationship)

}

val existsRequest = ExistsRequest(

id,

candidate.id,

relationships = relationships,

context = SgsRelationshipsPredicate.UnionLookupContext

)

socialGraph

.exists(existsRequest).map { existsResult: ExistsResult =>

if (existsResult.exists) {

PredicateResult.Invalid(Set(InvalidRelationshipTypes(relationshipMappings

.map { relationshipMapping: RelationshipMapping =>

relationshipMapping.relationshipType

}.mkString(", "))))

} else {

PredicateResult.Valid

}

}

.within(timeout)(com.twitter.finagle.util.DefaultTimer)

}

// if no user id is present, return true by default

.getOrElse(Stitch.value(PredicateResult.Valid))

.rescue {

case e: TimeoutException =>

stats.counter("timeout").incr()

Stitch(PredicateResult.Invalid(Set(FailOpen)))

case e: Exception =>

stats.counter(e.getClass.getSimpleName).incr()

Stitch(PredicateResult.Invalid(Set(FailOpen)))

}

}

}

object SgsRelationshipsPredicate {

// OR Operation

@VisibleForTesting

private[follow\_recommendations] val UnionLookupContext = Some(

LookupContext(performUnion = Some(true)))

private def extractUserId(target: HasClientContext with HasParams): Option[Long] = target match {

case profRequest: HasProfileId => Some(profRequest.profileId)

case userRequest: HasClientContext with HasParams => userRequest.getOptionalUserId

case \_ => None

}

}

@Singleton

class InvalidTargetCandidateRelationshipTypesPredicate @Inject() (

socialGraph: SocialGraph)

extends SgsRelationshipsPredicate(

socialGraph,

InvalidRelationshipTypesPredicate.InvalidRelationshipTypes) {}

@Singleton

class NoteworthyAccountsSgsPredicate @Inject() (

socialGraph: SocialGraph)

extends SgsRelationshipsPredicate(

socialGraph,

InvalidRelationshipTypesPredicate.NoteworthyAccountsInvalidRelationshipTypes)

object InvalidRelationshipTypesPredicate {

val InvalidRelationshipTypesExcludeFollowing: Seq[RelationshipMapping] = Seq(

RelationshipMapping(RelationshipType.HideRecommendations, true),

RelationshipMapping(RelationshipType.Blocking, true),

RelationshipMapping(RelationshipType.BlockedBy, true),

RelationshipMapping(RelationshipType.Muting, true),

RelationshipMapping(RelationshipType.MutedBy, true),

RelationshipMapping(RelationshipType.ReportedAsSpam, true),

RelationshipMapping(RelationshipType.ReportedAsSpamBy, true),

RelationshipMapping(RelationshipType.ReportedAsAbuse, true),

RelationshipMapping(RelationshipType.ReportedAsAbuseBy, true)

)

val InvalidRelationshipTypes: Seq[RelationshipMapping] = Seq(

RelationshipMapping(RelationshipType.FollowRequestOutgoing, true),

RelationshipMapping(RelationshipType.Following, true),

RelationshipMapping(

RelationshipType.UsedToFollow,

true

) // this data is accessible for 90 days.

) ++ InvalidRelationshipTypesExcludeFollowing

val NoteworthyAccountsInvalidRelationshipTypes: Seq[RelationshipMapping] = Seq(

RelationshipMapping(RelationshipType.Blocking, true),

RelationshipMapping(RelationshipType.BlockedBy, true),

RelationshipMapping(RelationshipType.Muting, true),

RelationshipMapping(RelationshipType.MutedBy, true),

RelationshipMapping(RelationshipType.ReportedAsSpam, true),

RelationshipMapping(RelationshipType.ReportedAsSpamBy, true),

RelationshipMapping(RelationshipType.ReportedAsAbuse, true),

RelationshipMapping(RelationshipType.ReportedAsAbuseBy, true)

)

}