package com.twitter.follow\_recommendations.common.clients.graph\_feature\_service

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

import com.twitter.graph\_feature\_service.thriftscala.PresetFeatureTypes.WtfTwoHop

import com.twitter.graph\_feature\_service.thriftscala.EdgeType

import com.twitter.graph\_feature\_service.thriftscala.GfsIntersectionResponse

import com.twitter.graph\_feature\_service.thriftscala.GfsPresetIntersectionRequest

import com.twitter.graph\_feature\_service.thriftscala.{Server => GraphFeatureService}

import com.twitter.stitch.Stitch

import javax.inject.{Inject, Singleton}

@Singleton

class GraphFeatureServiceClient @Inject() (

graphFeatureService: GraphFeatureService.MethodPerEndpoint) {

import GraphFeatureServiceClient.\_

def getIntersections(

userId: Long,

candidateIds: Seq[Long],

numIntersectionIds: Int

): Stitch[Map[Long, FollowProof]] = {

Stitch

.callFuture(

graphFeatureService.getPresetIntersection(

GfsPresetIntersectionRequest(userId, candidateIds, WtfTwoHop, Some(numIntersectionIds))

)

).map {

case GfsIntersectionResponse(gfsIntersectionResults) =>

(for {

candidateId <- candidateIds

gfsIntersectionResultForCandidate =

gfsIntersectionResults.filter(\_.candidateUserId == candidateId)

followProof <- for {

result <- gfsIntersectionResultForCandidate

intersection <- result.intersectionValues

if leftEdgeTypes.contains(intersection.featureType.leftEdgeType)

if rightEdgeTypes.contains(intersection.featureType.rightEdgeType)

intersectionIds <- intersection.intersectionIds.toSeq

} yield FollowProof(intersectionIds, intersection.count.getOrElse(0))

} yield {

candidateId -> followProof

}).toMap

}

}

}

object GraphFeatureServiceClient {

val leftEdgeTypes: Set[EdgeType] = Set(EdgeType.Following)

val rightEdgeTypes: Set[EdgeType] = Set(EdgeType.FollowedBy)

}