package com.twitter.follow\_recommendations.common.clients.interests\_service

import com.google.inject.Inject

import com.google.inject.Singleton

import com.twitter.finagle.stats.NullStatsReceiver

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.store.InterestedInInterestsFetchKey

import com.twitter.inject.Logging

import com.twitter.interests.thriftscala.InterestId

import com.twitter.interests.thriftscala.InterestRelationship

import com.twitter.interests.thriftscala.InterestedInInterestModel

import com.twitter.interests.thriftscala.UserInterest

import com.twitter.interests.thriftscala.UserInterestData

import com.twitter.interests.thriftscala.UserInterestsResponse

import com.twitter.stitch.Stitch

import com.twitter.strato.client.Client

import com.twitter.strato.thrift.ScroogeConvImplicits.\_

@Singleton

class InterestServiceClient @Inject() (

stratoClient: Client,

statsReceiver: StatsReceiver = NullStatsReceiver)

extends Logging {

val interestsServiceStratoColumnPath = "interests/interestedInInterests"

val stats = statsReceiver.scope("interest\_service\_client")

val errorCounter = stats.counter("error")

private val interestsFetcher =

stratoClient.fetcher[InterestedInInterestsFetchKey, UserInterestsResponse](

interestsServiceStratoColumnPath,

checkTypes = true

)

def fetchUttInterestIds(

userId: Long

): Stitch[Seq[Long]] = {

fetchInterestRelationships(userId)

.map(\_.toSeq.flatten.flatMap(extractUttInterest))

}

def extractUttInterest(

interestRelationShip: InterestRelationship

): Option[Long] = {

interestRelationShip match {

case InterestRelationship.V1(relationshipV1) =>

relationshipV1.interestId match {

case InterestId.SemanticCore(semanticCoreInterest) => Some(semanticCoreInterest.id)

case \_ => None

}

case \_ => None

}

}

def fetchCustomInterests(

userId: Long

): Stitch[Seq[String]] = {

fetchInterestRelationships(userId)

.map(\_.toSeq.flatten.flatMap(extractCustomInterest))

}

def extractCustomInterest(

interestRelationShip: InterestRelationship

): Option[String] = {

interestRelationShip match {

case InterestRelationship.V1(relationshipV1) =>

relationshipV1.interestId match {

case InterestId.FreeForm(freeFormInterest) => Some(freeFormInterest.interest)

case \_ => None

}

case \_ => None

}

}

def fetchInterestRelationships(

userId: Long

): Stitch[Option[Seq[InterestRelationship]]] = {

interestsFetcher

.fetch(

InterestedInInterestsFetchKey(

userId = userId,

labels = None,

None

))

.map(\_.v)

.map {

case Some(response) =>

response.interests.interests.map { interests =>

interests.collect {

case UserInterest(\_, Some(interestData)) =>

getInterestRelationship(interestData)

}.flatten

}

case \_ => None

}

.rescue {

case e: Throwable => // we are swallowing all errors

logger.warn(s"interests could not be retrieved for user $userId due to ${e.getCause}")

errorCounter.incr

Stitch.None

}

}

private def getInterestRelationship(

interestData: UserInterestData

): Seq[InterestRelationship] = {

interestData match {

case UserInterestData.InterestedIn(interestModels) =>

interestModels.collect {

case InterestedInInterestModel.ExplicitModel(model) => model

}

case \_ => Nil

}

}

}