package com.twitter.follow\_recommendations.common.clients.dismiss\_store

import com.twitter.follow\_recommendations.common.constants.GuiceNamedConstants

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.onboarding.relevance.store.thriftscala.WhoToFollowDismissEventDetails

import com.twitter.stitch.Stitch

import com.twitter.strato.catalog.Scan.Slice

import com.twitter.strato.client.Scanner

import com.twitter.util.logging.Logging

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

/\*\*

\* this store gets the list of dismissed candidates since a certain time

\* primarily used for filtering out accounts that a user has explicitly dismissed

\*

\* we fail open on timeouts, but loudly on other errors

\*/

@Singleton

class DismissStore @Inject() (

@Named(GuiceNamedConstants.DISMISS\_STORE\_SCANNER)

scanner: Scanner[(Long, Slice[

(Long, Long)

]), Unit, (Long, (Long, Long)), WhoToFollowDismissEventDetails],

stats: StatsReceiver)

extends Logging {

private val MaxCandidatesToReturn = 100

// gets a list of dismissed candidates. if numCandidatesToFetchOption is none, we will fetch the default number of candidates

def get(

userId: Long,

negStartTimeMs: Long,

maxCandidatesToFetchOption: Option[Int]

): Stitch[Seq[Long]] = {

val maxCandidatesToFetch = maxCandidatesToFetchOption.getOrElse(MaxCandidatesToReturn)

scanner

.scan(

(

userId,

Slice(

from = None,

to = Some((negStartTimeMs, Long.MaxValue)),

limit = Some(maxCandidatesToFetch)

)

)

)

.map {

case s: Seq[((Long, (Long, Long)), WhoToFollowDismissEventDetails)] if s.nonEmpty =>

s.map {

case ((\_: Long, (\_: Long, candidateId: Long)), \_: WhoToFollowDismissEventDetails) =>

candidateId

}

case \_ => Nil

}

}

}