package com.twitter.cr\_mixer.source\_signal

import com.twitter.cr\_mixer.config.TimeoutConfig

import com.twitter.cr\_mixer.model.ModuleNames

import com.twitter.cr\_mixer.model.SourceInfo

import com.twitter.cr\_mixer.param.FrsParams

import com.twitter.cr\_mixer.param.GlobalParams

import com.twitter.cr\_mixer.source\_signal.FrsStore.FrsQueryResult

import com.twitter.cr\_mixer.source\_signal.SourceFetcher.FetcherQuery

import com.twitter.cr\_mixer.thriftscala.SourceType

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.simclusters\_v2.common.UserId

import com.twitter.simclusters\_v2.thriftscala.InternalId

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

import javax.inject.Singleton

import javax.inject.Inject

import javax.inject.Named

@Singleton

case class FrsSourceSignalFetcher @Inject() (

@Named(ModuleNames.FrsStore) frsStore: ReadableStore[FrsStore.Query, Seq[FrsQueryResult]],

override val timeoutConfig: TimeoutConfig,

globalStats: StatsReceiver)

extends SourceSignalFetcher {

override protected val stats: StatsReceiver = globalStats.scope(identifier)

override type SignalConvertType = UserId

override def isEnabled(query: FetcherQuery): Boolean = {

query.params(FrsParams.EnableSourceParam)

}

override def fetchAndProcess(query: FetcherQuery): Future[Option[Seq[SourceInfo]]] = {

// Fetch raw signals

val rawSignals = frsStore

.get(FrsStore.Query(query.userId, query.params(GlobalParams.UnifiedMaxSourceKeyNum)))

.map {

\_.map {

\_.map {

\_.userId

}

}

}

// Process signals

rawSignals.map {

\_.map { frsUsers =>

convertSourceInfo(SourceType.FollowRecommendation, frsUsers)

}

}

}

override def convertSourceInfo(

sourceType: SourceType,

signals: Seq[SignalConvertType]

): Seq[SourceInfo] = {

signals.map { signal =>

SourceInfo(

sourceType = sourceType,

internalId = InternalId.UserId(signal),

sourceEventTime = None

)

}

}

}