package com.twitter.visibility.builder.users

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.visibility.builder.FeatureMapBuilder

import com.twitter.visibility.common.user\_result.UserVisibilityResultHelper

import com.twitter.visibility.features.AuthorBlocksViewer

import com.twitter.visibility.features.AuthorIsDeactivated

import com.twitter.visibility.features.AuthorIsErased

import com.twitter.visibility.features.AuthorIsOffboarded

import com.twitter.visibility.features.AuthorIsProtected

import com.twitter.visibility.features.AuthorIsSuspended

import com.twitter.visibility.features.AuthorIsUnavailable

import com.twitter.visibility.features.ViewerBlocksAuthor

import com.twitter.visibility.features.ViewerMutesAuthor

import com.twitter.visibility.models.UserUnavailableStateEnum

case class UserUnavailableFeatures(statsReceiver: StatsReceiver) {

private[this] val scopedStatsReceiver = statsReceiver.scope("user\_unavailable\_features")

private[this] val suspendedAuthorStats = scopedStatsReceiver.scope("suspended\_author")

private[this] val deactivatedAuthorStats = scopedStatsReceiver.scope("deactivated\_author")

private[this] val offboardedAuthorStats = scopedStatsReceiver.scope("offboarded\_author")

private[this] val erasedAuthorStats = scopedStatsReceiver.scope("erased\_author")

private[this] val protectedAuthorStats = scopedStatsReceiver.scope("protected\_author")

private[this] val authorBlocksViewerStats = scopedStatsReceiver.scope("author\_blocks\_viewer")

private[this] val viewerBlocksAuthorStats = scopedStatsReceiver.scope("viewer\_blocks\_author")

private[this] val viewerMutesAuthorStats = scopedStatsReceiver.scope("viewer\_mutes\_author")

private[this] val unavailableStats = scopedStatsReceiver.scope("unavailable")

def forState(state: UserUnavailableStateEnum): FeatureMapBuilder => FeatureMapBuilder = {

builder =>

builder

.withConstantFeature(AuthorIsSuspended, isSuspended(state))

.withConstantFeature(AuthorIsDeactivated, isDeactivated(state))

.withConstantFeature(AuthorIsOffboarded, isOffboarded(state))

.withConstantFeature(AuthorIsErased, isErased(state))

.withConstantFeature(AuthorIsProtected, isProtected(state))

.withConstantFeature(AuthorBlocksViewer, authorBlocksViewer(state))

.withConstantFeature(ViewerBlocksAuthor, viewerBlocksAuthor(state))

.withConstantFeature(ViewerMutesAuthor, viewerMutesAuthor(state))

.withConstantFeature(AuthorIsUnavailable, isUnavailable(state))

}

private[this] def isSuspended(state: UserUnavailableStateEnum): Boolean =

state match {

case UserUnavailableStateEnum.Suspended =>

suspendedAuthorStats.counter().incr()

true

case UserUnavailableStateEnum.Filtered(result)

if UserVisibilityResultHelper.isDropSuspendedAuthor(result) =>

suspendedAuthorStats.counter().incr()

suspendedAuthorStats.counter("filtered").incr()

true

case \_ => false

}

private[this] def isDeactivated(state: UserUnavailableStateEnum): Boolean =

state match {

case UserUnavailableStateEnum.Deactivated =>

deactivatedAuthorStats.counter().incr()

true

case \_ => false

}

private[this] def isOffboarded(state: UserUnavailableStateEnum): Boolean =

state match {

case UserUnavailableStateEnum.Offboarded =>

offboardedAuthorStats.counter().incr()

true

case \_ => false

}

private[this] def isErased(state: UserUnavailableStateEnum): Boolean =

state match {

case UserUnavailableStateEnum.Erased =>

erasedAuthorStats.counter().incr()

true

case \_ => false

}

private[this] def isProtected(state: UserUnavailableStateEnum): Boolean =

state match {

case UserUnavailableStateEnum.Protected =>

protectedAuthorStats.counter().incr()

true

case UserUnavailableStateEnum.Filtered(result)

if UserVisibilityResultHelper.isDropProtectedAuthor(result) =>

protectedAuthorStats.counter().incr()

protectedAuthorStats.counter("filtered").incr()

true

case \_ => false

}

private[this] def authorBlocksViewer(state: UserUnavailableStateEnum): Boolean =

state match {

case UserUnavailableStateEnum.AuthorBlocksViewer =>

authorBlocksViewerStats.counter().incr()

true

case UserUnavailableStateEnum.Filtered(result)

if UserVisibilityResultHelper.isDropAuthorBlocksViewer(result) =>

authorBlocksViewerStats.counter().incr()

authorBlocksViewerStats.counter("filtered").incr()

true

case \_ => false

}

private[this] def viewerBlocksAuthor(state: UserUnavailableStateEnum): Boolean =

state match {

case UserUnavailableStateEnum.ViewerBlocksAuthor =>

viewerBlocksAuthorStats.counter().incr()

true

case UserUnavailableStateEnum.Filtered(result)

if UserVisibilityResultHelper.isDropViewerBlocksAuthor(result) =>

viewerBlocksAuthorStats.counter().incr()

viewerBlocksAuthorStats.counter("filtered").incr()

true

case \_ => false

}

private[this] def viewerMutesAuthor(state: UserUnavailableStateEnum): Boolean =

state match {

case UserUnavailableStateEnum.ViewerMutesAuthor =>

viewerMutesAuthorStats.counter().incr()

true

case UserUnavailableStateEnum.Filtered(result)

if UserVisibilityResultHelper.isDropViewerMutesAuthor(result) =>

viewerMutesAuthorStats.counter().incr()

viewerMutesAuthorStats.counter("filtered").incr()

true

case \_ => false

}

private[this] def isUnavailable(state: UserUnavailableStateEnum): Boolean =

state match {

case UserUnavailableStateEnum.Unavailable =>

unavailableStats.counter().incr()

true

case UserUnavailableStateEnum.Filtered(result)

if UserVisibilityResultHelper.isDropUnspecifiedAuthor(result) =>

unavailableStats.counter().incr()

unavailableStats.counter("filtered").incr()

true

case \_ => false

}

}