package com.twitter.visibility.builder.users

import com.twitter.finagle.stats.Counter

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.gizmoduck.thriftscala.Label

import com.twitter.gizmoduck.thriftscala.Safety

import com.twitter.gizmoduck.thriftscala.UniversalQualityFiltering

import com.twitter.gizmoduck.thriftscala.User

import com.twitter.gizmoduck.thriftscala.UserType

import com.twitter.stitch.NotFound

import com.twitter.stitch.Stitch

import com.twitter.visibility.builder.FeatureMapBuilder

import com.twitter.visibility.common.UserId

import com.twitter.visibility.common.UserSource

import com.twitter.visibility.features.\_

import com.twitter.visibility.interfaces.common.blender.BlenderVFRequestContext

import com.twitter.visibility.interfaces.common.search.SearchVFRequestContext

import com.twitter.visibility.models.UserAge

import com.twitter.visibility.models.ViewerContext

class ViewerFeatures(userSource: UserSource, statsReceiver: StatsReceiver) {

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

private[this] val requests = scopedStatsReceiver.counter("requests")

private[this] val viewerIdCount =

scopedStatsReceiver.scope(ViewerId.name).counter("requests")

private[this] val requestCountryCode =

scopedStatsReceiver.scope(RequestCountryCode.name).counter("requests")

private[this] val requestIsVerifiedCrawler =

scopedStatsReceiver.scope(RequestIsVerifiedCrawler.name).counter("requests")

private[this] val viewerUserLabels =

scopedStatsReceiver.scope(ViewerUserLabels.name).counter("requests")

private[this] val viewerIsDeactivated =

scopedStatsReceiver.scope(ViewerIsDeactivated.name).counter("requests")

private[this] val viewerIsProtected =

scopedStatsReceiver.scope(ViewerIsProtected.name).counter("requests")

private[this] val viewerIsSuspended =

scopedStatsReceiver.scope(ViewerIsSuspended.name).counter("requests")

private[this] val viewerRoles =

scopedStatsReceiver.scope(ViewerRoles.name).counter("requests")

private[this] val viewerCountryCode =

scopedStatsReceiver.scope(ViewerCountryCode.name).counter("requests")

private[this] val viewerAge =

scopedStatsReceiver.scope(ViewerAge.name).counter("requests")

private[this] val viewerHasUniversalQualityFilterEnabled =

scopedStatsReceiver.scope(ViewerHasUniversalQualityFilterEnabled.name).counter("requests")

private[this] val viewerIsSoftUserCtr =

scopedStatsReceiver.scope(ViewerIsSoftUser.name).counter("requests")

def forViewerBlenderContext(

blenderContext: BlenderVFRequestContext,

viewerContext: ViewerContext

): FeatureMapBuilder => FeatureMapBuilder =

forViewerContext(viewerContext)

.andThen(

\_.withConstantFeature(

ViewerOptInBlocking,

blenderContext.userSearchSafetySettings.optInBlocking)

.withConstantFeature(

ViewerOptInFiltering,

blenderContext.userSearchSafetySettings.optInFiltering)

)

def forViewerSearchContext(

searchContext: SearchVFRequestContext,

viewerContext: ViewerContext

): FeatureMapBuilder => FeatureMapBuilder =

forViewerContext(viewerContext)

.andThen(

\_.withConstantFeature(

ViewerOptInBlocking,

searchContext.userSearchSafetySettings.optInBlocking)

.withConstantFeature(

ViewerOptInFiltering,

searchContext.userSearchSafetySettings.optInFiltering)

)

def forViewerContext(viewerContext: ViewerContext): FeatureMapBuilder => FeatureMapBuilder =

forViewerId(viewerContext.userId)

.andThen(

\_.withConstantFeature(RequestCountryCode, requestCountryCode(viewerContext))

).andThen(

\_.withConstantFeature(RequestIsVerifiedCrawler, requestIsVerifiedCrawler(viewerContext))

)

def forViewerId(viewerId: Option[UserId]): FeatureMapBuilder => FeatureMapBuilder = { builder =>

requests.incr()

val builderWithFeatures = builder

.withConstantFeature(ViewerId, viewerId)

.withFeature(ViewerIsProtected, viewerIsProtected(viewerId))

.withFeature(

ViewerHasUniversalQualityFilterEnabled,

viewerHasUniversalQualityFilterEnabled(viewerId)

)

.withFeature(ViewerIsSuspended, viewerIsSuspended(viewerId))

.withFeature(ViewerIsDeactivated, viewerIsDeactivated(viewerId))

.withFeature(ViewerUserLabels, viewerUserLabels(viewerId))

.withFeature(ViewerRoles, viewerRoles(viewerId))

.withFeature(ViewerAge, viewerAgeInYears(viewerId))

.withFeature(ViewerIsSoftUser, viewerIsSoftUser(viewerId))

viewerId match {

case Some(\_) =>

viewerIdCount.incr()

builderWithFeatures

.withFeature(ViewerCountryCode, viewerCountryCode(viewerId))

case \_ =>

builderWithFeatures

}

}

def forViewerNoDefaults(viewerOpt: Option[User]): FeatureMapBuilder => FeatureMapBuilder = {

builder =>

requests.incr()

viewerOpt match {

case Some(viewer) =>

builder

.withConstantFeature(ViewerId, viewer.id)

.withConstantFeature(ViewerIsProtected, viewerIsProtectedOpt(viewer))

.withConstantFeature(ViewerIsSuspended, viewerIsSuspendedOpt(viewer))

.withConstantFeature(ViewerIsDeactivated, viewerIsDeactivatedOpt(viewer))

.withConstantFeature(ViewerCountryCode, viewerCountryCode(viewer))

case None =>

builder

.withConstantFeature(ViewerIsProtected, false)

.withConstantFeature(ViewerIsSuspended, false)

.withConstantFeature(ViewerIsDeactivated, false)

}

}

private def checkSafetyValue(

viewerId: Option[UserId],

safetyCheck: Safety => Boolean,

featureCounter: Counter

): Stitch[Boolean] =

viewerId match {

case Some(id) =>

userSource.getSafety(id).map(safetyCheck).ensure {

featureCounter.incr()

}

case None => Stitch.False

}

private def checkSafetyValue(

viewer: User,

safetyCheck: Safety => Boolean

): Boolean = {

viewer.safety.exists(safetyCheck)

}

def viewerIsProtected(viewerId: Option[UserId]): Stitch[Boolean] =

checkSafetyValue(viewerId, s => s.isProtected, viewerIsProtected)

def viewerIsProtected(viewer: User): Boolean =

checkSafetyValue(viewer, s => s.isProtected)

def viewerIsProtectedOpt(viewer: User): Option[Boolean] =

viewer.safety.map(\_.isProtected)

def viewerIsDeactivated(viewerId: Option[UserId]): Stitch[Boolean] =

checkSafetyValue(viewerId, s => s.deactivated, viewerIsDeactivated)

def viewerIsDeactivated(viewer: User): Boolean =

checkSafetyValue(viewer, s => s.deactivated)

def viewerIsDeactivatedOpt(viewer: User): Option[Boolean] =

viewer.safety.map(\_.deactivated)

def viewerHasUniversalQualityFilterEnabled(viewerId: Option[UserId]): Stitch[Boolean] =

checkSafetyValue(

viewerId,

s => s.universalQualityFiltering == UniversalQualityFiltering.Enabled,

viewerHasUniversalQualityFilterEnabled

)

def viewerUserLabels(viewerIdOpt: Option[UserId]): Stitch[Seq[Label]] =

viewerIdOpt match {

case Some(viewerId) =>

userSource

.getLabels(viewerId).map(\_.labels)

.handle {

case NotFound => Seq.empty

}.ensure {

viewerUserLabels.incr()

}

case \_ => Stitch.value(Seq.empty)

}

def viewerAgeInYears(viewerId: Option[UserId]): Stitch[UserAge] =

(viewerId match {

case Some(id) =>

userSource

.getExtendedProfile(id).map(\_.ageInYears)

.handle {

case NotFound => None

}.ensure {

viewerAge.incr()

}

case \_ => Stitch.value(None)

}).map(UserAge)

def viewerIsSoftUser(viewerId: Option[UserId]): Stitch[Boolean] = {

viewerId match {

case Some(id) =>

userSource

.getUserType(id).map { userType =>

userType == UserType.Soft

}.ensure {

viewerIsSoftUserCtr.incr()

}

case \_ => Stitch.False

}

}

def requestCountryCode(viewerContext: ViewerContext): Option[String] = {

requestCountryCode.incr()

viewerContext.requestCountryCode

}

def requestIsVerifiedCrawler(viewerContext: ViewerContext): Boolean = {

requestIsVerifiedCrawler.incr()

viewerContext.isVerifiedCrawler

}

def viewerCountryCode(viewerId: Option[UserId]): Stitch[String] =

viewerId match {

case Some(id) =>

userSource

.getAccount(id).map(\_.countryCode).flatMap {

case Some(countryCode) => Stitch.value(countryCode.toLowerCase)

case \_ => Stitch.NotFound

}.ensure {

viewerCountryCode.incr()

}

case \_ => Stitch.NotFound

}

def viewerCountryCode(viewer: User): Option[String] =

viewer.account.flatMap(\_.countryCode)

}