package com.twitter.visibility.interfaces.users

import com.twitter.decider.Decider

import com.twitter.gizmoduck.thriftscala.User

import com.twitter.servo.decider.DeciderGateBuilder

import com.twitter.stitch.Stitch

import com.twitter.strato.client.Client

import com.twitter.visibility.VisibilityLibrary

import com.twitter.visibility.builder.users.AuthorFeatures

import com.twitter.visibility.builder.users.RelationshipFeatures

import com.twitter.visibility.builder.users.ViewerAdvancedFilteringFeatures

import com.twitter.visibility.builder.users.ViewerFeatures

import com.twitter.visibility.builder.users.ViewerSearchSafetyFeatures

import com.twitter.visibility.builder.VisibilityResult

import com.twitter.visibility.builder.users.SearchFeatures

import com.twitter.visibility.common.UserRelationshipSource

import com.twitter.visibility.common.UserSearchSafetySource

import com.twitter.visibility.common.UserSource

import com.twitter.visibility.configapi.configs.VisibilityDeciderGates

import com.twitter.visibility.context.thriftscala.UserVisibilityFilteringContext

import com.twitter.visibility.models.ContentId.UserId

import com.twitter.visibility.models.SafetyLevel

import com.twitter.visibility.models.ViewerContext

import com.twitter.visibility.rules.Reason.Unspecified

import com.twitter.visibility.rules.Allow

import com.twitter.visibility.rules.Drop

import com.twitter.visibility.rules.RuleBase

object UserVisibilityLibrary {

type Type =

(User, SafetyLevel, ViewerContext, UserVisibilityFilteringContext) => Stitch[VisibilityResult]

def apply(

visibilityLibrary: VisibilityLibrary,

userSource: UserSource = UserSource.empty,

userRelationshipSource: UserRelationshipSource = UserRelationshipSource.empty,

stratoClient: Client,

decider: Decider

): Type = {

val libraryStatsReceiver = visibilityLibrary.statsReceiver.scope("user\_library")

val stratoClientStatsReceiver = visibilityLibrary.statsReceiver.scope("strato")

val visibilityDeciderGates = VisibilityDeciderGates(decider)

val vfEngineCounter = libraryStatsReceiver.counter("vf\_engine\_requests")

val noUserRulesCounter = libraryStatsReceiver.counter("no\_user\_rules\_requests")

val viewerIsAuthorCounter = libraryStatsReceiver.counter("viewer\_is\_author\_requests")

val authorFeatures = new AuthorFeatures(userSource, libraryStatsReceiver)

val viewerFeatures = new ViewerFeatures(userSource, libraryStatsReceiver)

val relationshipFeatures =

new RelationshipFeatures(userRelationshipSource, libraryStatsReceiver)

val searchFeatures = new SearchFeatures(libraryStatsReceiver)

val viewerSafeSearchFeatures = new ViewerSearchSafetyFeatures(

UserSearchSafetySource.fromStrato(stratoClient, stratoClientStatsReceiver),

libraryStatsReceiver)

val deciderGateBuilder = new DeciderGateBuilder(decider)

val advancedFilteringFeatures =

new ViewerAdvancedFilteringFeatures(userSource, libraryStatsReceiver)

(user, safetyLevel, viewerContext, userVisibilityFilteringContext) => {

val contentId = UserId(user.id)

val viewerId = viewerContext.userId

if (!RuleBase.hasUserRules(safetyLevel)) {

noUserRulesCounter.incr()

Stitch.value(VisibilityResult(contentId = contentId, verdict = Allow))

} else {

if (viewerId.contains(user.id)) {

viewerIsAuthorCounter.incr()

Stitch.value(VisibilityResult(contentId = contentId, verdict = Allow))

} else {

vfEngineCounter.incr()

val featureMap =

visibilityLibrary.featureMapBuilder(

Seq(

viewerFeatures.forViewerContext(viewerContext),

viewerSafeSearchFeatures.forViewerId(viewerId),

relationshipFeatures.forAuthor(user, viewerId),

authorFeatures.forAuthor(user),

advancedFilteringFeatures.forViewerId(viewerId),

searchFeatures.forSearchContext(userVisibilityFilteringContext.searchContext)

)

)

visibilityLibrary.runRuleEngine(

contentId,

featureMap,

viewerContext,

safetyLevel

)

}

}

}

}

def Const(shouldDrop: Boolean): Type =

(user, \_, \_, \_) =>

Stitch.value(

VisibilityResult(

contentId = UserId(user.id),

verdict = if (shouldDrop) Drop(Unspecified) else Allow,

finished = true

)

)

}