package com.twitter.tweetypie

package repository

import com.twitter.logging.Logger

import com.twitter.spam.rtf.thriftscala.{SafetyLevel => ThriftSafetyLevel}

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.\_

import com.twitter.tweetypie.repository.VisibilityResultToFilteredState.toFilteredState

import com.twitter.tweetypie.thriftscala.Tweet

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

import com.twitter.visibility.interfaces.tweets.TweetVisibilityLibrary

import com.twitter.visibility.interfaces.tweets.TweetVisibilityRequest

import com.twitter.visibility.models.SafetyLevel.DeprecatedSafetyLevel

import com.twitter.visibility.models.SafetyLevel

import com.twitter.visibility.models.ViewerContext

/\*\*

\* This repository handles visibility filtering of tweets

\*

\* i.e. deciding whether to drop/suppress tweets based on viewer

\* and safety level for instance. Rules in VF library can be thought as:

\*

\* (SafetyLevel)(Viewer, Content, Features) => Action

\*

\* SafetyLevel represents the product context in which the Viewer is

\* requesting to view the Content. Example: TimelineHome, TweetDetail,

\* Recommendations, Notifications

\*

\* Content here is mainly tweets (can be users, notifications, cards etc)

\*

\* Features might include safety labels and other metadata of a Tweet,

\* flags set on a User (including the Viewer), relationships between Users

\* (e.g. block, follow), relationships between Users and Content

\* (e.g. reported for spam)

\*

\* We initialize VisibilityLibrary using UserSource and UserRelationshipSource:

\* Stitch interfaces that provide methods to retrieve user and relationship

\* information in Gizmoduck and SocialGraph repositories, respectively.

\* This user and relationship info along with Tweet labels, provide necessary

\* features to take a filtering decision.

\*

\* Actions supported in Tweetypie right now are Drop and Suppress.

\* In the future, we might want to surface other granular actions such as

\* Tombstone and Downrank which are supported in VF lib.

\*

\* The TweetVisibilityRepository has the following format:

\*

\* Request(Tweet, Option[SafetyLevel], Option[UserId]) => Stitch[Option[FilteredState]]

\*

\* SafetyLevel is plumbed from the tweet query options.

\*

\* In addition to the latency stats and rpc counts from VF library, we also capture

\* unsupported and deprecated safety level stats here to inform the relevant clients.

\*

\* go/visibilityfiltering, go/visibilityfilteringdocs

\*

\*/

object TweetVisibilityRepository {

type Type = Request => Stitch[Option[FilteredState]]

case class Request(

tweet: Tweet,

viewerId: Option[UserId],

safetyLevel: ThriftSafetyLevel,

isInnerQuotedTweet: Boolean,

isRetweet: Boolean,

hydrateConversationControl: Boolean,

isSourceTweet: Boolean)

def apply(

visibilityLibrary: TweetVisibilityLibrary.Type,

visibilityDeciderGates: VisibilityDeciderGates,

log: Logger,

statsReceiver: StatsReceiver

): TweetVisibilityRepository.Type = {

val noTweetRulesCounter = statsReceiver.counter("no\_tweet\_rules\_requests")

val deprecatedScope = statsReceiver.scope("deprecated\_safety\_level")

request: Request =>

SafetyLevel.fromThrift(request.safetyLevel) match {

case DeprecatedSafetyLevel =>

deprecatedScope.counter(request.safetyLevel.name.toLowerCase()).incr()

log.warning("Deprecated SafetyLevel (%s) requested".format(request.safetyLevel.name))

Stitch.None

case safetyLevel: SafetyLevel =>

if (!TweetVisibilityLibrary.hasTweetRules(safetyLevel)) {

noTweetRulesCounter.incr()

Stitch.None

} else {

visibilityLibrary(

TweetVisibilityRequest(

tweet = request.tweet,

safetyLevel = safetyLevel,

viewerContext = ViewerContext.fromContextWithViewerIdFallback(request.viewerId),

isInnerQuotedTweet = request.isInnerQuotedTweet,

isRetweet = request.isRetweet,

hydrateConversationControl = request.hydrateConversationControl,

isSourceTweet = request.isSourceTweet

)

).map(visibilityResult =>

toFilteredState(

visibilityResult = visibilityResult,

disableLegacyInterstitialFilteredReason =

visibilityDeciderGates.disableLegacyInterstitialFilteredReason()))

}

}

}

/\*\*

\* We can skip visibility filtering when any of the following is true:

\*

\* - SafetyLevel is deprecated

\* - SafetyLevel has no tweet rules

\*/

def canSkipVisibilityFiltering(thriftSafetyLevel: ThriftSafetyLevel): Boolean =

SafetyLevel.fromThrift(thriftSafetyLevel) match {

case DeprecatedSafetyLevel =>

true

case safetyLevel: SafetyLevel =>

!TweetVisibilityLibrary.hasTweetRules(safetyLevel)

}

}