package com.twitter.tweetypie.repository

import com.twitter.spam.rtf.thriftscala.FilteredReason

import com.twitter.spam.rtf.thriftscala.KeywordMatch

import com.twitter.spam.rtf.thriftscala.SafetyResult

import com.twitter.tweetypie.core.FilteredState

import com.twitter.tweetypie.core.FilteredState.Suppress

import com.twitter.tweetypie.core.FilteredState.Unavailable

import com.twitter.visibility.builder.VisibilityResult

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

import com.twitter.visibility.rules.Reason.\_

import com.twitter.visibility.rules.\_

import com.twitter.visibility.{thriftscala => vfthrift}

object VisibilityResultToFilteredState {

def toFilteredStateUnavailable(

visibilityResult: VisibilityResult

): Option[FilteredState.Unavailable] = {

val dropSafetyResult = Some(

Unavailable.Drop(FilteredReason.SafetyResult(visibilityResult.getSafetyResult))

)

visibilityResult.verdict match {

case Drop(ExclusiveTweet, \_) =>

dropSafetyResult

case Drop(NsfwViewerIsUnderage | NsfwViewerHasNoStatedAge | NsfwLoggedOut, \_) =>

dropSafetyResult

case Drop(TrustedFriendsTweet, \_) =>

dropSafetyResult

case \_: LocalizedTombstone => dropSafetyResult

case Drop(StaleTweet, \_) => dropSafetyResult

// legacy drop actions

case dropAction: Drop => unavailableFromDropAction(dropAction)

// not an unavailable state that can be mapped

case \_ => None

}

}

def toFilteredState(

visibilityResult: VisibilityResult,

disableLegacyInterstitialFilteredReason: Boolean

): Option[FilteredState] = {

val suppressSafetyResult = Some(

Suppress(FilteredReason.SafetyResult(visibilityResult.getSafetyResult))

)

val dropSafetyResult = Some(

Unavailable.Drop(FilteredReason.SafetyResult(visibilityResult.getSafetyResult))

)

visibilityResult.verdict match {

case \_: Appealable => suppressSafetyResult

case \_: Preview => suppressSafetyResult

case \_: InterstitialLimitedEngagements => suppressSafetyResult

case \_: EmergencyDynamicInterstitial => suppressSafetyResult

case \_: SoftIntervention => suppressSafetyResult

case \_: LimitedEngagements => suppressSafetyResult

case \_: TweetInterstitial => suppressSafetyResult

case \_: TweetVisibilityNudge => suppressSafetyResult

case Interstitial(

ViewerBlocksAuthor | ViewerReportedAuthor | ViewerReportedTweet | ViewerMutesAuthor |

ViewerHardMutedAuthor | MutedKeyword | InterstitialDevelopmentOnly | HatefulConduct |

AbusiveBehavior,

\_,

\_) if disableLegacyInterstitialFilteredReason =>

suppressSafetyResult

case Interstitial(

ViewerBlocksAuthor | ViewerReportedAuthor | ViewerReportedTweet |

InterstitialDevelopmentOnly,

\_,

\_) =>

suppressSafetyResult

case \_: ComplianceTweetNotice => suppressSafetyResult

case Drop(ExclusiveTweet, \_) =>

dropSafetyResult

case Drop(NsfwViewerIsUnderage | NsfwViewerHasNoStatedAge | NsfwLoggedOut, \_) =>

dropSafetyResult

case Drop(TrustedFriendsTweet, \_) =>

dropSafetyResult

case Drop(StaleTweet, \_) => dropSafetyResult

case \_: LocalizedTombstone => dropSafetyResult

case \_: Avoid => suppressSafetyResult

// legacy drop actions

case dropAction: Drop => unavailableFromDropAction(dropAction)

// legacy suppress actions

case action => suppressFromVisibilityAction(action, !disableLegacyInterstitialFilteredReason)

}

}

def toFilteredState(

userVisibilityResult: Option[vfthrift.UserVisibilityResult]

): FilteredState.Unavailable =

userVisibilityResult

.collect {

case blockedUser if UserVisibilityResultHelper.isDropAuthorBlocksViewer(blockedUser) =>

Unavailable.Drop(FilteredReason.AuthorBlockViewer(true))

/\*\*

\* Reuse states for author visibility issues from the [[UserRepository]] for consistency with

\* other logic for handling the same types of author visibility filtering.

\*/

case protectedUser if UserVisibilityResultHelper.isDropProtectedAuthor(protectedUser) =>

Unavailable.Author.Protected

case suspendedUser if UserVisibilityResultHelper.isDropSuspendedAuthor(suspendedUser) =>

Unavailable.Author.Suspended

case nsfwUser if UserVisibilityResultHelper.isDropNsfwAuthor(nsfwUser) =>

Unavailable.Drop(FilteredReason.ContainNsfwMedia(true))

case mutedByViewer if UserVisibilityResultHelper.isDropViewerMutesAuthor(mutedByViewer) =>

Unavailable.Drop(FilteredReason.ViewerMutesAuthor(true))

case blockedByViewer

if UserVisibilityResultHelper.isDropViewerBlocksAuthor(blockedByViewer) =>

Unavailable.Drop(

FilteredReason.SafetyResult(

SafetyResult(

None,

vfthrift.Action.Drop(

vfthrift.Drop(Some(vfthrift.DropReason.ViewerBlocksAuthor(true)))

))))

}

.getOrElse(FilteredState.Unavailable.Drop(FilteredReason.UnspecifiedReason(true)))

private def unavailableFromDropAction(dropAction: Drop): Option[FilteredState.Unavailable] =

dropAction match {

case Drop(AuthorBlocksViewer, \_) =>

Some(Unavailable.Drop(FilteredReason.AuthorBlockViewer(true)))

case Drop(Unspecified, \_) =>

Some(Unavailable.Drop(FilteredReason.UnspecifiedReason(true)))

case Drop(MutedKeyword, \_) =>

Some(Unavailable.Drop(FilteredReason.TweetMatchesViewerMutedKeyword(KeywordMatch(""))))

case Drop(ViewerMutesAuthor, \_) =>

Some(Unavailable.Drop(FilteredReason.ViewerMutesAuthor(true)))

case Drop(Nsfw, \_) =>

Some(Unavailable.Drop(FilteredReason.ContainNsfwMedia(true)))

case Drop(NsfwMedia, \_) =>

Some(Unavailable.Drop(FilteredReason.ContainNsfwMedia(true)))

case Drop(PossiblyUndesirable, \_) =>

Some(Unavailable.Drop(FilteredReason.PossiblyUndesirable(true)))

case Drop(Bounce, \_) =>

Some(Unavailable.Drop(FilteredReason.TweetIsBounced(true)))

/\*\*

\* Reuse states for author visibility issues from the [[UserRepository]] for consistency with

\* other logic for handling the same types of author visibility filtering.

\*/

case Drop(ProtectedAuthor, \_) =>

Some(Unavailable.Author.Protected)

case Drop(SuspendedAuthor, \_) =>

Some(Unavailable.Author.Suspended)

case Drop(OffboardedAuthor, \_) =>

Some(Unavailable.Author.Offboarded)

case Drop(DeactivatedAuthor, \_) =>

Some(Unavailable.Author.Deactivated)

case Drop(ErasedAuthor, \_) =>

Some(Unavailable.Author.Deactivated)

case \_: Drop =>

Some(Unavailable.Drop(FilteredReason.UnspecifiedReason(true)))

}

private def suppressFromVisibilityAction(

action: Action,

enableLegacyFilteredReason: Boolean

): Option[FilteredState.Suppress] =

action match {

case interstitial: Interstitial =>

interstitial.reason match {

case MutedKeyword if enableLegacyFilteredReason =>

Some(Suppress(FilteredReason.TweetMatchesViewerMutedKeyword(KeywordMatch(""))))

case ViewerMutesAuthor if enableLegacyFilteredReason =>

Some(Suppress(FilteredReason.ViewerMutesAuthor(true)))

case ViewerHardMutedAuthor if enableLegacyFilteredReason =>

Some(Suppress(FilteredReason.ViewerMutesAuthor(true)))

// Interstitial tweets are considered suppressed by Tweetypie. For

// legacy behavior reasons, these tweets should be dropped when

// appearing as a quoted tweet via a call to getTweets.

case Nsfw =>

Some(Suppress(FilteredReason.ContainNsfwMedia(true)))

case NsfwMedia =>

Some(Suppress(FilteredReason.ContainNsfwMedia(true)))

case PossiblyUndesirable =>

Some(Suppress(FilteredReason.PossiblyUndesirable(true)))

case \_ =>

Some(Suppress(FilteredReason.PossiblyUndesirable(true)))

}

case \_ => None

}

}