package com.twitter.visibility.rules

import com.twitter.datatools.entityservice.entities.thriftscala.FleetInterstitial

import com.twitter.scrooge.ThriftStruct

import com.twitter.visibility.common.actions.LocalizedMessage

import com.twitter.visibility.common.actions.\_

import com.twitter.visibility.common.actions.converter.scala.AppealableReasonConverter

import com.twitter.visibility.common.actions.converter.scala.AvoidReasonConverter

import com.twitter.visibility.common.actions.converter.scala.ComplianceTweetNoticeEventTypeConverter

import com.twitter.visibility.common.actions.converter.scala.DownrankHomeTimelineReasonConverter

import com.twitter.visibility.common.actions.converter.scala.DropReasonConverter

import com.twitter.visibility.common.actions.converter.scala.InterstitialReasonConverter

import com.twitter.visibility.common.actions.converter.scala.LimitedActionsPolicyConverter

import com.twitter.visibility.common.actions.converter.scala.LimitedEngagementReasonConverter

import com.twitter.visibility.common.actions.converter.scala.LocalizedMessageConverter

import com.twitter.visibility.common.actions.converter.scala.SoftInterventionDisplayTypeConverter

import com.twitter.visibility.common.actions.converter.scala.SoftInterventionReasonConverter

import com.twitter.visibility.common.actions.converter.scala.TombstoneReasonConverter

import com.twitter.visibility.features.Feature

import com.twitter.visibility.logging.thriftscala.HealthActionType

import com.twitter.visibility.models.ViolationLevel

import com.twitter.visibility.strato.thriftscala.NudgeActionType.EnumUnknownNudgeActionType

import com.twitter.visibility.strato.thriftscala.{Nudge => StratoNudge}

import com.twitter.visibility.strato.thriftscala.{NudgeAction => StratoNudgeAction}

import com.twitter.visibility.strato.thriftscala.{NudgeActionType => StratoNudgeActionType}

import com.twitter.visibility.strato.thriftscala.{NudgeActionPayload => StratoNudgeActionPayload}

import com.twitter.visibility.thriftscala

import com.twitter.visibility.util.NamingUtils

sealed trait Action {

lazy val name: String = NamingUtils.getFriendlyName(this)

lazy val fullName: String = NamingUtils.getFriendlyName(this)

val severity: Int

def toActionThrift(): thriftscala.Action

def isComposable: Boolean = false

def toHealthActionTypeThrift: Option[HealthActionType]

}

sealed trait Reason {

lazy val name: String = NamingUtils.getFriendlyName(this)

}

sealed abstract class ActionWithReason(reason: Reason) extends Action {

override lazy val fullName: String = s"${this.name}/${reason.name}"

}

object Reason {

case object Bounce extends Reason

case object ViewerReportedAuthor extends Reason

case object ViewerReportedTweet extends Reason

case object DeactivatedAuthor extends Reason

case object OffboardedAuthor extends Reason

case object ErasedAuthor extends Reason

case object ProtectedAuthor extends Reason

case object SuspendedAuthor extends Reason

case object ViewerIsUnmentioned extends Reason

case object Nsfw extends Reason

case object NsfwMedia extends Reason

case object NsfwViewerIsUnderage extends Reason

case object NsfwViewerHasNoStatedAge extends Reason

case object NsfwLoggedOut extends Reason

case object PossiblyUndesirable extends Reason

case object AbuseEpisodic extends Reason

case object AbuseEpisodicEncourageSelfHarm extends Reason

case object AbuseEpisodicHatefulConduct extends Reason

case object AbuseGlorificationOfViolence extends Reason

case object AbuseGratuitousGore extends Reason

case object AbuseMobHarassment extends Reason

case object AbuseMomentOfDeathOrDeceasedUser extends Reason

case object AbusePrivateInformation extends Reason

case object AbuseRightToPrivacy extends Reason

case object AbuseThreatToExpose extends Reason

case object AbuseViolentSexualConduct extends Reason

case object AbuseViolentThreatHatefulConduct extends Reason

case object AbuseViolentThreatOrBounty extends Reason

case object MutedKeyword extends Reason

case object Unspecified extends Reason

case object UntrustedUrl extends Reason

case object SpamReplyDownRank extends Reason

case object LowQualityTweet extends Reason

case object LowQualityMention extends Reason

case object SpamHighRecallTweet extends Reason

case object TweetLabelDuplicateContent extends Reason

case object TweetLabelDuplicateMention extends Reason

case object PdnaTweet extends Reason

case object TweetLabeledSpam extends Reason

case object OneOff extends Reason

case object VotingMisinformation extends Reason

case object HackedMaterials extends Reason

case object Scams extends Reason

case object PlatformManipulation extends Reason

case object FirstPageSearchResult extends Reason

case object MisinfoCivic extends Reason

case object MisinfoCrisis extends Reason

case object MisinfoGeneric extends Reason

case object MisinfoMedical extends Reason

case object Misleading extends Reason

case object ExclusiveTweet extends Reason

case object CommunityNotAMember extends Reason

case object CommunityTweetHidden extends Reason

case object CommunityTweetCommunityIsSuspended extends Reason

case object CommunityTweetAuthorRemoved extends Reason

case object InternalPromotedContent extends Reason

case object TrustedFriendsTweet extends Reason

case object Toxicity extends Reason

case object StaleTweet extends Reason

case object DmcaWithheld extends Reason

case object LegalDemandsWithheld extends Reason

case object LocalLawsWithheld extends Reason

case object HatefulConduct extends Reason

case object AbusiveBehavior extends Reason

case object NotSupportedOnDevice extends Reason

case object IpiDevelopmentOnly extends Reason

case object InterstitialDevelopmentOnly extends Reason

case class FosnrReason(appealableReason: AppealableReason) extends Reason

def toDropReason(reason: Reason): Option[DropReason] =

reason match {

case AuthorBlocksViewer => Some(DropReason.AuthorBlocksViewer)

case CommunityTweetHidden => Some(DropReason.CommunityTweetHidden)

case CommunityTweetCommunityIsSuspended => Some(DropReason.CommunityTweetCommunityIsSuspended)

case DmcaWithheld => Some(DropReason.DmcaWithheld)

case ExclusiveTweet => Some(DropReason.ExclusiveTweet)

case InternalPromotedContent => Some(DropReason.InternalPromotedContent)

case LegalDemandsWithheld => Some(DropReason.LegalDemandsWithheld)

case LocalLawsWithheld => Some(DropReason.LocalLawsWithheld)

case Nsfw => Some(DropReason.NsfwAuthor)

case NsfwLoggedOut => Some(DropReason.NsfwLoggedOut)

case NsfwViewerHasNoStatedAge => Some(DropReason.NsfwViewerHasNoStatedAge)

case NsfwViewerIsUnderage => Some(DropReason.NsfwViewerIsUnderage)

case ProtectedAuthor => Some(DropReason.ProtectedAuthor)

case StaleTweet => Some(DropReason.StaleTweet)

case SuspendedAuthor => Some(DropReason.SuspendedAuthor)

case Unspecified => Some(DropReason.Unspecified)

case ViewerBlocksAuthor => Some(DropReason.ViewerBlocksAuthor)

case ViewerHardMutedAuthor => Some(DropReason.ViewerMutesAuthor)

case ViewerMutesAuthor => Some(DropReason.ViewerMutesAuthor)

case TrustedFriendsTweet => Some(DropReason.TrustedFriendsTweet)

case \_ => Some(DropReason.Unspecified)

}

def fromDropReason(dropReason: DropReason): Reason =

dropReason match {

case DropReason.AuthorBlocksViewer => AuthorBlocksViewer

case DropReason.CommunityTweetHidden => CommunityTweetHidden

case DropReason.CommunityTweetCommunityIsSuspended => CommunityTweetCommunityIsSuspended

case DropReason.DmcaWithheld => DmcaWithheld

case DropReason.ExclusiveTweet => ExclusiveTweet

case DropReason.InternalPromotedContent => InternalPromotedContent

case DropReason.LegalDemandsWithheld => LegalDemandsWithheld

case DropReason.LocalLawsWithheld => LocalLawsWithheld

case DropReason.NsfwAuthor => Nsfw

case DropReason.NsfwLoggedOut => NsfwLoggedOut

case DropReason.NsfwViewerHasNoStatedAge => NsfwViewerHasNoStatedAge

case DropReason.NsfwViewerIsUnderage => NsfwViewerIsUnderage

case DropReason.ProtectedAuthor => ProtectedAuthor

case DropReason.StaleTweet => StaleTweet

case DropReason.SuspendedAuthor => SuspendedAuthor

case DropReason.ViewerBlocksAuthor => ViewerBlocksAuthor

case DropReason.ViewerMutesAuthor => ViewerMutesAuthor

case DropReason.TrustedFriendsTweet => TrustedFriendsTweet

case DropReason.Unspecified => Unspecified

}

def toAppealableReason(reason: Reason, violationLevel: ViolationLevel): Option[AppealableReason] =

reason match {

case HatefulConduct => Some(AppealableReason.HatefulConduct(violationLevel.level))

case AbusiveBehavior => Some(AppealableReason.AbusiveBehavior(violationLevel.level))

case \_ => Some(AppealableReason.Unspecified(violationLevel.level))

}

def fromAppealableReason(appealableReason: AppealableReason): Reason =

appealableReason match {

case AppealableReason.HatefulConduct(level) => HatefulConduct

case AppealableReason.AbusiveBehavior(level) => AbusiveBehavior

case AppealableReason.Unspecified(level) => Unspecified

}

def toSoftInterventionReason(appealableReason: AppealableReason): SoftInterventionReason =

appealableReason match {

case AppealableReason.HatefulConduct(level) =>

SoftInterventionReason.FosnrReason(appealableReason)

case AppealableReason.AbusiveBehavior(level) =>

SoftInterventionReason.FosnrReason(appealableReason)

case AppealableReason.Unspecified(level) =>

SoftInterventionReason.FosnrReason(appealableReason)

}

def toLimitedEngagementReason(appealableReason: AppealableReason): LimitedEngagementReason =

appealableReason match {

case AppealableReason.HatefulConduct(level) =>

LimitedEngagementReason.FosnrReason(appealableReason)

case AppealableReason.AbusiveBehavior(level) =>

LimitedEngagementReason.FosnrReason(appealableReason)

case AppealableReason.Unspecified(level) =>

LimitedEngagementReason.FosnrReason(appealableReason)

}

val NSFW\_MEDIA: Set[Reason] = Set(Nsfw, NsfwMedia)

def toInterstitialReason(reason: Reason): Option[InterstitialReason] =

reason match {

case r if NSFW\_MEDIA.contains(r) => Some(InterstitialReason.ContainsNsfwMedia)

case PossiblyUndesirable => Some(InterstitialReason.PossiblyUndesirable)

case MutedKeyword => Some(InterstitialReason.MatchesMutedKeyword(""))

case ViewerReportedAuthor => Some(InterstitialReason.ViewerReportedAuthor)

case ViewerReportedTweet => Some(InterstitialReason.ViewerReportedTweet)

case ViewerBlocksAuthor => Some(InterstitialReason.ViewerBlocksAuthor)

case ViewerMutesAuthor => Some(InterstitialReason.ViewerMutesAuthor)

case ViewerHardMutedAuthor => Some(InterstitialReason.ViewerMutesAuthor)

case InterstitialDevelopmentOnly => Some(InterstitialReason.DevelopmentOnly)

case DmcaWithheld => Some(InterstitialReason.DmcaWithheld)

case LegalDemandsWithheld => Some(InterstitialReason.LegalDemandsWithheld)

case LocalLawsWithheld => Some(InterstitialReason.LocalLawsWithheld)

case HatefulConduct => Some(InterstitialReason.HatefulConduct)

case AbusiveBehavior => Some(InterstitialReason.AbusiveBehavior)

case FosnrReason(appealableReason) => Some(InterstitialReason.FosnrReason(appealableReason))

case \_ => None

}

def fromInterstitialReason(interstitialReason: InterstitialReason): Reason =

interstitialReason match {

case InterstitialReason.ContainsNsfwMedia => Reason.NsfwMedia

case InterstitialReason.PossiblyUndesirable => Reason.PossiblyUndesirable

case InterstitialReason.MatchesMutedKeyword(\_) => Reason.MutedKeyword

case InterstitialReason.ViewerReportedAuthor => Reason.ViewerReportedAuthor

case InterstitialReason.ViewerReportedTweet => Reason.ViewerReportedTweet

case InterstitialReason.ViewerBlocksAuthor => Reason.ViewerBlocksAuthor

case InterstitialReason.ViewerMutesAuthor => Reason.ViewerMutesAuthor

case InterstitialReason.DevelopmentOnly => Reason.InterstitialDevelopmentOnly

case InterstitialReason.DmcaWithheld => Reason.DmcaWithheld

case InterstitialReason.LegalDemandsWithheld => Reason.LegalDemandsWithheld

case InterstitialReason.LocalLawsWithheld => Reason.LocalLawsWithheld

case InterstitialReason.HatefulConduct => Reason.HatefulConduct

case InterstitialReason.AbusiveBehavior => Reason.AbusiveBehavior

case InterstitialReason.FosnrReason(reason) => Reason.fromAppealableReason(reason)

}

}

sealed trait Epitaph {

lazy val name: String = NamingUtils.getFriendlyName(this)

}

object Epitaph {

case object Unavailable extends Epitaph

case object Blocked extends Epitaph

case object BlockedBy extends Epitaph

case object Reported extends Epitaph

case object BounceDeleted extends Epitaph

case object Deleted extends Epitaph

case object NotFound extends Epitaph

case object PublicInterest extends Epitaph

case object Bounced extends Epitaph

case object Protected extends Epitaph

case object Suspended extends Epitaph

case object Offboarded extends Epitaph

case object Deactivated extends Epitaph

case object MutedKeyword extends Epitaph

case object Underage extends Epitaph

case object NoStatedAge extends Epitaph

case object LoggedOutAge extends Epitaph

case object SuperFollowsContent extends Epitaph

case object Moderated extends Epitaph

case object ForEmergencyUseOnly extends Epitaph

case object UnavailableWithoutLink extends Epitaph

case object CommunityTweetHidden extends Epitaph

case object CommunityTweetMemberRemoved extends Epitaph

case object CommunityTweetCommunityIsSuspended extends Epitaph

case object UserSuspended extends Epitaph

case object DevelopmentOnly extends Epitaph

case object AdultMedia extends Epitaph

case object ViolentMedia extends Epitaph

case object OtherSensitiveMedia extends Epitaph

case object DmcaWithheldMedia extends Epitaph

case object LegalDemandsWithheldMedia extends Epitaph

case object LocalLawsWithheldMedia extends Epitaph

case object ToxicReplyFiltered extends Epitaph

}

sealed trait IsInterstitial {

def toInterstitialThriftWrapper(): thriftscala.AnyInterstitial

def toInterstitialThrift(): ThriftStruct

}

sealed trait IsAppealable {

def toAppealableThrift(): thriftscala.Appealable

}

sealed trait IsLimitedEngagements {

def policy: Option[LimitedActionsPolicy]

def getLimitedEngagementReason: LimitedEngagementReason

}

object IsLimitedEngagements {

def unapply(

ile: IsLimitedEngagements

): Option[(Option[LimitedActionsPolicy], LimitedEngagementReason)] = {

Some((ile.policy, ile.getLimitedEngagementReason))

}

}

sealed abstract class ActionWithEpitaph(epitaph: Epitaph) extends Action {

override lazy val fullName: String = s"${this.name}/${epitaph.name}"

}

case class Appealable(

reason: Reason,

violationLevel: ViolationLevel,

localizedMessage: Option[LocalizedMessage] = None)

extends ActionWithReason(reason)

with IsAppealable {

override val severity: Int = 17

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.Appealable(toAppealableThrift())

override def toAppealableThrift(): thriftscala.Appealable =

thriftscala.Appealable(

Reason.toAppealableReason(reason, violationLevel).map(AppealableReasonConverter.toThrift),

localizedMessage.map(LocalizedMessageConverter.toThrift)

)

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(

HealthActionType.Appealable)

}

case class Drop(reason: Reason, applicableCountries: Option[Seq[String]] = None)

extends ActionWithReason(reason) {

override val severity: Int = 16

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.Drop(

thriftscala.Drop(

Reason.toDropReason(reason).map(DropReasonConverter.toThrift),

applicableCountries

))

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(HealthActionType.Drop)

}

case class Interstitial(

reason: Reason,

localizedMessage: Option[LocalizedMessage] = None,

applicableCountries: Option[Seq[String]] = None)

extends ActionWithReason(reason)

with IsInterstitial {

override val severity: Int = 10

override def toInterstitialThriftWrapper(): thriftscala.AnyInterstitial =

thriftscala.AnyInterstitial.Interstitial(

toInterstitialThrift()

)

override def toInterstitialThrift(): thriftscala.Interstitial =

thriftscala.Interstitial(

Reason.toInterstitialReason(reason).map(InterstitialReasonConverter.toThrift),

localizedMessage.map(LocalizedMessageConverter.toThrift)

)

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.Interstitial(toInterstitialThrift())

def toMediaActionThrift(): thriftscala.MediaAction =

thriftscala.MediaAction.Interstitial(toInterstitialThrift())

override def isComposable: Boolean = true

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(

HealthActionType.TweetInterstitial)

}

case class InterstitialLimitedEngagements(

reason: Reason,

limitedEngagementReason: Option[LimitedEngagementReason],

localizedMessage: Option[LocalizedMessage] = None,

policy: Option[LimitedActionsPolicy] = None)

extends ActionWithReason(reason)

with IsInterstitial

with IsLimitedEngagements {

override val severity: Int = 11

override def toInterstitialThriftWrapper(): thriftscala.AnyInterstitial =

thriftscala.AnyInterstitial.InterstitialLimitedEngagements(

toInterstitialThrift()

)

override def toInterstitialThrift(): thriftscala.InterstitialLimitedEngagements =

thriftscala.InterstitialLimitedEngagements(

limitedEngagementReason.map(LimitedEngagementReasonConverter.toThrift),

localizedMessage.map(LocalizedMessageConverter.toThrift)

)

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.InterstitialLimitedEngagements(toInterstitialThrift())

override def isComposable: Boolean = true

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(

HealthActionType.LimitedEngagements)

def getLimitedEngagementReason: LimitedEngagementReason = limitedEngagementReason.getOrElse(

LimitedEngagementReason.NonCompliant

)

}

case object Allow extends Action {

override val severity: Int = -1

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.Allow(thriftscala.Allow())

override def toHealthActionTypeThrift: Option[HealthActionType] = None

}

case object NotEvaluated extends Action {

override val severity: Int = -1

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.NotEvaluated(thriftscala.NotEvaluated())

override def toHealthActionTypeThrift: Option[HealthActionType] = None

}

case class Tombstone(epitaph: Epitaph, applicableCountryCodes: Option[Seq[String]] = None)

extends ActionWithEpitaph(epitaph) {

override val severity: Int = 15

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.Tombstone(thriftscala.Tombstone())

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(HealthActionType.Tombstone)

}

case class LocalizedTombstone(reason: TombstoneReason, message: LocalizedMessage) extends Action {

override lazy val fullName: String = s"${this.name}/${NamingUtils.getFriendlyName(reason)}"

override val severity: Int = 15

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.Tombstone(

thriftscala.Tombstone(

reason = TombstoneReasonConverter.toThrift(Some(reason)),

message = Some(LocalizedMessageConverter.toThrift(message))

))

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(HealthActionType.Tombstone)

}

case class DownrankHomeTimeline(reason: Option[DownrankHomeTimelineReason]) extends Action {

override val severity: Int = 9

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.DownrankHomeTimeline(toDownrankThrift())

def toDownrankThrift(): thriftscala.DownrankHomeTimeline =

thriftscala.DownrankHomeTimeline(

reason.map(DownrankHomeTimelineReasonConverter.toThrift)

)

override def isComposable: Boolean = true

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(HealthActionType.Downrank)

}

case class Avoid(avoidReason: Option[AvoidReason] = None) extends Action {

override val severity: Int = 1

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.Avoid(toAvoidThrift())

def toAvoidThrift(): thriftscala.Avoid =

thriftscala.Avoid(

avoidReason.map(AvoidReasonConverter.toThrift)

)

override def isComposable: Boolean = true

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(HealthActionType.Avoid)

}

case object Downrank extends Action {

override val severity: Int = 0

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.Downrank(thriftscala.Downrank())

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(HealthActionType.Downrank)

}

case object ConversationSectionLowQuality extends Action {

override val severity: Int = 4

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.ConversationSectionLowQuality(thriftscala.ConversationSectionLowQuality())

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(

HealthActionType.ConversationSectionLowQuality)

}

case object ConversationSectionAbusiveQuality extends Action {

override val severity: Int = 5

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.ConversationSectionAbusiveQuality(

thriftscala.ConversationSectionAbusiveQuality())

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(

HealthActionType.ConversationSectionAbusiveQuality)

def toConversationSectionAbusiveQualityThrift(): thriftscala.ConversationSectionAbusiveQuality =

thriftscala.ConversationSectionAbusiveQuality()

}

case class LimitedEngagements(

reason: LimitedEngagementReason,

policy: Option[LimitedActionsPolicy] = None)

extends Action

with IsLimitedEngagements {

override val severity: Int = 6

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.LimitedEngagements(toLimitedEngagementsThrift())

def toLimitedEngagementsThrift(): thriftscala.LimitedEngagements =

thriftscala.LimitedEngagements(

Some(LimitedEngagementReasonConverter.toThrift(reason)),

policy.map(LimitedActionsPolicyConverter.toThrift),

Some(reason.toLimitedActionsString)

)

override def isComposable: Boolean = true

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(

HealthActionType.LimitedEngagements)

def getLimitedEngagementReason: LimitedEngagementReason = reason

}

case class EmergencyDynamicInterstitial(

copy: String,

linkOpt: Option[String],

localizedMessage: Option[LocalizedMessage] = None,

policy: Option[LimitedActionsPolicy] = None)

extends Action

with IsInterstitial

with IsLimitedEngagements {

override val severity: Int = 11

override def toInterstitialThriftWrapper(): thriftscala.AnyInterstitial =

thriftscala.AnyInterstitial.EmergencyDynamicInterstitial(

toInterstitialThrift()

)

override def toInterstitialThrift(): thriftscala.EmergencyDynamicInterstitial =

thriftscala.EmergencyDynamicInterstitial(

copy,

linkOpt,

localizedMessage.map(LocalizedMessageConverter.toThrift)

)

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.EmergencyDynamicInterstitial(toInterstitialThrift())

override def isComposable: Boolean = true

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(

HealthActionType.TweetInterstitial)

def getLimitedEngagementReason: LimitedEngagementReason = LimitedEngagementReason.NonCompliant

}

case class SoftIntervention(

reason: SoftInterventionReason,

engagementNudge: Boolean,

suppressAutoplay: Boolean,

warning: Option[String] = None,

detailsUrl: Option[String] = None,

displayType: Option[SoftInterventionDisplayType] = None,

fleetInterstitial: Option[FleetInterstitial] = None)

extends Action {

override val severity: Int = 7

def toSoftInterventionThrift(): thriftscala.SoftIntervention =

thriftscala.SoftIntervention(

Some(SoftInterventionReasonConverter.toThrift(reason)),

engagementNudge = Some(engagementNudge),

suppressAutoplay = Some(suppressAutoplay),

warning = warning,

detailsUrl = detailsUrl,

displayType = SoftInterventionDisplayTypeConverter.toThrift(displayType)

)

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.SoftIntervention(toSoftInterventionThrift())

override def isComposable: Boolean = true

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(

HealthActionType.SoftIntervention)

}

case class TweetInterstitial(

interstitial: Option[IsInterstitial],

softIntervention: Option[SoftIntervention],

limitedEngagements: Option[LimitedEngagements],

downrank: Option[DownrankHomeTimeline],

avoid: Option[Avoid],

mediaInterstitial: Option[Interstitial] = None,

tweetVisibilityNudge: Option[TweetVisibilityNudge] = None,

abusiveQuality: Option[ConversationSectionAbusiveQuality.type] = None,

appealable: Option[Appealable] = None)

extends Action {

override val severity: Int = 12

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.TweetInterstitial(

thriftscala.TweetInterstitial(

interstitial.map(\_.toInterstitialThriftWrapper()),

softIntervention.map(\_.toSoftInterventionThrift()),

limitedEngagements.map(\_.toLimitedEngagementsThrift()),

downrank.map(\_.toDownrankThrift()),

avoid.map(\_.toAvoidThrift()),

mediaInterstitial.map(\_.toMediaActionThrift()),

tweetVisibilityNudge.map(\_.toTweetVisbilityNudgeThrift()),

abusiveQuality.map(\_.toConversationSectionAbusiveQualityThrift()),

appealable.map(\_.toAppealableThrift())

)

)

override def toHealthActionTypeThrift: Option[HealthActionType] = Some(

HealthActionType.TweetInterstitial)

}

sealed trait LocalizedNudgeActionType

object LocalizedNudgeActionType {

case object Reply extends LocalizedNudgeActionType

case object Retweet extends LocalizedNudgeActionType

case object Like extends LocalizedNudgeActionType

case object Share extends LocalizedNudgeActionType

case object Unspecified extends LocalizedNudgeActionType

def toThrift(

localizedNudgeActionType: LocalizedNudgeActionType

): thriftscala.TweetVisibilityNudgeActionType =

localizedNudgeActionType match {

case Reply => thriftscala.TweetVisibilityNudgeActionType.Reply

case Retweet => thriftscala.TweetVisibilityNudgeActionType.Retweet

case Like => thriftscala.TweetVisibilityNudgeActionType.Like

case Share => thriftscala.TweetVisibilityNudgeActionType.Share

case Unspecified =>

thriftscala.TweetVisibilityNudgeActionType.EnumUnknownTweetVisibilityNudgeActionType(5)

}

def fromStratoThrift(stratoNudgeActionType: StratoNudgeActionType): LocalizedNudgeActionType =

stratoNudgeActionType match {

case StratoNudgeActionType.Reply => Reply

case StratoNudgeActionType.Retweet => Retweet

case StratoNudgeActionType.Like => Like

case StratoNudgeActionType.Share => Share

case EnumUnknownNudgeActionType(\_) => Unspecified

}

}

case class LocalizedNudgeActionPayload(

heading: Option[String],

subheading: Option[String],

iconName: Option[String],

ctaTitle: Option[String],

ctaUrl: Option[String],

postCtaText: Option[String]) {

def toThrift(): thriftscala.TweetVisibilityNudgeActionPayload = {

thriftscala.TweetVisibilityNudgeActionPayload(

heading = heading,

subheading = subheading,

iconName = iconName,

ctaTitle = ctaTitle,

ctaUrl = ctaUrl,

postCtaText = postCtaText

)

}

}

object LocalizedNudgeActionPayload {

def fromStratoThrift(

stratoNudgeActionPayload: StratoNudgeActionPayload

): LocalizedNudgeActionPayload =

LocalizedNudgeActionPayload(

heading = stratoNudgeActionPayload.heading,

subheading = stratoNudgeActionPayload.subheading,

iconName = stratoNudgeActionPayload.iconName,

ctaTitle = stratoNudgeActionPayload.ctaTitle,

ctaUrl = stratoNudgeActionPayload.ctaUrl,

postCtaText = stratoNudgeActionPayload.postCtaText

)

}

case class LocalizedNudgeAction(

nudgeActionType: LocalizedNudgeActionType,

nudgeActionPayload: Option[LocalizedNudgeActionPayload]) {

def toThrift(): thriftscala.TweetVisibilityNudgeAction = {

thriftscala.TweetVisibilityNudgeAction(

tweetVisibilitynudgeActionType = LocalizedNudgeActionType.toThrift(nudgeActionType),

tweetVisibilityNudgeActionPayload = nudgeActionPayload.map(\_.toThrift)

)

}

}

object LocalizedNudgeAction {

def fromStratoThrift(stratoNudgeAction: StratoNudgeAction): LocalizedNudgeAction =

LocalizedNudgeAction(

nudgeActionType =

LocalizedNudgeActionType.fromStratoThrift(stratoNudgeAction.nudgeActionType),

nudgeActionPayload =

stratoNudgeAction.nudgeActionPayload.map(LocalizedNudgeActionPayload.fromStratoThrift)

)

}

case class LocalizedNudge(localizedNudgeActions: Seq[LocalizedNudgeAction])

case object LocalizedNudge {

def fromStratoThrift(stratoNudge: StratoNudge): LocalizedNudge =

LocalizedNudge(localizedNudgeActions =

stratoNudge.nudgeActions.map(LocalizedNudgeAction.fromStratoThrift))

}

case class TweetVisibilityNudge(

reason: TweetVisibilityNudgeReason,

localizedNudge: Option[LocalizedNudge] = None)

extends Action {

override val severity: Int = 3

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.TweetVisibilityNudge(

localizedNudge match {

case Some(nudge) =>

thriftscala.TweetVisibilityNudge(

tweetVisibilityNudgeActions = Some(nudge.localizedNudgeActions.map(\_.toThrift()))

)

case \_ => thriftscala.TweetVisibilityNudge(tweetVisibilityNudgeActions = None)

}

)

override def toHealthActionTypeThrift: Option[HealthActionType] =

Some(HealthActionType.TweetVisibilityNudge)

def toTweetVisbilityNudgeThrift(): thriftscala.TweetVisibilityNudge =

thriftscala.TweetVisibilityNudge(tweetVisibilityNudgeActions =

localizedNudge.map(\_.localizedNudgeActions.map(\_.toThrift())))

}

trait BaseComplianceTweetNotice {

val complianceTweetNoticeEventType: ComplianceTweetNoticeEventType

val details: Option[String]

val extendedDetailsUrl: Option[String]

}

case class ComplianceTweetNoticePreEnrichment(

reason: Reason,

complianceTweetNoticeEventType: ComplianceTweetNoticeEventType,

details: Option[String] = None,

extendedDetailsUrl: Option[String] = None)

extends Action

with BaseComplianceTweetNotice {

override val severity: Int = 2

def toComplianceTweetNoticeThrift(): thriftscala.ComplianceTweetNotice =

thriftscala.ComplianceTweetNotice(

ComplianceTweetNoticeEventTypeConverter.toThrift(complianceTweetNoticeEventType),

ComplianceTweetNoticeEventTypeConverter.eventTypeToLabelTitle(complianceTweetNoticeEventType),

details,

extendedDetailsUrl

)

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.ComplianceTweetNotice(

toComplianceTweetNoticeThrift()

)

override def toHealthActionTypeThrift: Option[HealthActionType] = None

def toComplianceTweetNotice(): ComplianceTweetNotice = {

ComplianceTweetNotice(

complianceTweetNoticeEventType = complianceTweetNoticeEventType,

labelTitle = ComplianceTweetNoticeEventTypeConverter.eventTypeToLabelTitle(

complianceTweetNoticeEventType),

details = details,

extendedDetailsUrl = extendedDetailsUrl

)

}

}

case class ComplianceTweetNotice(

complianceTweetNoticeEventType: ComplianceTweetNoticeEventType,

labelTitle: Option[String] = None,

details: Option[String] = None,

extendedDetailsUrl: Option[String] = None)

extends Action

with BaseComplianceTweetNotice {

override val severity: Int = 2

def toComplianceTweetNoticeThrift(): thriftscala.ComplianceTweetNotice =

thriftscala.ComplianceTweetNotice(

ComplianceTweetNoticeEventTypeConverter.toThrift(complianceTweetNoticeEventType),

labelTitle,

details,

extendedDetailsUrl

)

override def toActionThrift(): thriftscala.Action =

thriftscala.Action.ComplianceTweetNotice(

toComplianceTweetNoticeThrift()

)

override def toHealthActionTypeThrift: Option[HealthActionType] = None

}

object Action {

def toThrift[T <: Action](action: T): thriftscala.Action =

action.toActionThrift()

def getFirstInterstitial(actions: Action\*): Option[IsInterstitial] =

actions.collectFirst {

case ile: InterstitialLimitedEngagements => ile

case edi: EmergencyDynamicInterstitial => edi

case i: Interstitial => i

}

def getFirstSoftIntervention(actions: Action\*): Option[SoftIntervention] =

actions.collectFirst {

case si: SoftIntervention => si

}

def getFirstLimitedEngagements(actions: Action\*): Option[LimitedEngagements] =

actions.collectFirst {

case le: LimitedEngagements => le

}

def getAllLimitedEngagements(actions: Action\*): Seq[IsLimitedEngagements] =

actions.collect {

case ile: IsLimitedEngagements => ile

}

def getFirstDownrankHomeTimeline(actions: Action\*): Option[DownrankHomeTimeline] =

actions.collectFirst {

case dr: DownrankHomeTimeline => dr

}

def getFirstAvoid(actions: Action\*): Option[Avoid] =

actions.collectFirst {

case a: Avoid => a

}

def getFirstMediaInterstitial(actions: Action\*): Option[Interstitial] =

actions.collectFirst {

case i: Interstitial if Reason.NSFW\_MEDIA.contains(i.reason) => i

}

def getFirstTweetVisibilityNudge(actions: Action\*): Option[TweetVisibilityNudge] =

actions.collectFirst {

case n: TweetVisibilityNudge => n

}

}

sealed trait State {

lazy val name: String = NamingUtils.getFriendlyName(this)

}

object State {

case object Pending extends State

case object Disabled extends State

final case class MissingFeature(features: Set[Feature[\_]]) extends State

final case class FeatureFailed(features: Map[Feature[\_], Throwable]) extends State

final case class RuleFailed(throwable: Throwable) extends State

case object Skipped extends State

case object ShortCircuited extends State

case object Heldback extends State

case object Evaluated extends State

}

case class RuleResult(action: Action, state: State)