package com.twitter.visibility.models

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

import com.twitter.datatools.entityservice.entities.{thriftscala => t}

import com.twitter.escherbird.softintervention.thriftscala.MisinformationLocalizedPolicy

import com.twitter.escherbird.thriftscala.TweetEntityAnnotation

case class MisinformationPolicy(

semanticCoreAnnotation: SemanticCoreAnnotation,

priority: Long = MisinformationPolicy.DefaultPriority,

filteringLevel: Int = MisinformationPolicy.DefaultFilteringLevel,

publishedState: PublishedState = MisinformationPolicy.DefaultPublishedState,

engagementNudge: Boolean = MisinformationPolicy.DefaultEngagementNudge,

suppressAutoplay: Boolean = MisinformationPolicy.DefaultSuppressAutoplay,

warning: Option[String] = None,

detailsUrl: Option[String] = None,

displayType: Option[MisinfoPolicyDisplayType] = None,

applicableCountries: Seq[String] = Seq.empty,

fleetInterstitial: Option[FleetInterstitial] = None)

object MisinformationPolicy {

private val DefaultPriority = 0

private val DefaultFilteringLevel = 1

private val DefaultPublishedState = PublishedState.Published

private val DefaultEngagementNudge = true

private val DefaultSuppressAutoplay = true

def apply(

annotation: TweetEntityAnnotation,

misinformation: MisinformationLocalizedPolicy

): MisinformationPolicy = {

MisinformationPolicy(

semanticCoreAnnotation = SemanticCoreAnnotation(

groupId = annotation.groupId,

domainId = annotation.domainId,

entityId = annotation.entityId

),

priority = misinformation.priority.getOrElse(DefaultPriority),

filteringLevel = misinformation.filteringLevel.getOrElse(DefaultFilteringLevel),

publishedState = misinformation.publishedState match {

case Some(t.PublishedState.Draft) => PublishedState.Draft

case Some(t.PublishedState.Dogfood) => PublishedState.Dogfood

case Some(t.PublishedState.Published) => PublishedState.Published

case \_ => DefaultPublishedState

},

displayType = misinformation.displayType collect {

case t.MisinformationDisplayType.GetTheLatest => MisinfoPolicyDisplayType.GetTheLatest

case t.MisinformationDisplayType.StayInformed => MisinfoPolicyDisplayType.StayInformed

case t.MisinformationDisplayType.Misleading => MisinfoPolicyDisplayType.Misleading

case t.MisinformationDisplayType.GovernmentRequested =>

MisinfoPolicyDisplayType.GovernmentRequested

},

applicableCountries = misinformation.applicableCountries match {

case Some(countries) => countries.map(countryCode => countryCode.toLowerCase)

case \_ => Seq.empty

},

fleetInterstitial = misinformation.fleetInterstitial,

engagementNudge = misinformation.engagementNudge.getOrElse(DefaultEngagementNudge),

suppressAutoplay = misinformation.suppressAutoplay.getOrElse(DefaultSuppressAutoplay),

warning = misinformation.warning,

detailsUrl = misinformation.detailsUrl,

)

}

}

trait MisinformationPolicyTransform {

def apply(policies: Seq[MisinformationPolicy]): Seq[MisinformationPolicy]

def andThen(transform: MisinformationPolicyTransform): MisinformationPolicyTransform =

(policies: Seq[MisinformationPolicy]) => transform(this.apply(policies))

}

object MisinformationPolicyTransform {

def prioritize: MisinformationPolicyTransform =

(policies: Seq[MisinformationPolicy]) =>

policies

.sortBy(p => p.filteringLevel)(Ordering[Int].reverse)

.sortBy(p => p.priority)(Ordering[Long].reverse)

def filter(filters: Seq[MisinformationPolicy => Boolean]): MisinformationPolicyTransform =

(policies: Seq[MisinformationPolicy]) =>

policies.filter { policy => filters.forall { filter => filter(policy) } }

def filterLevelAndState(

filteringLevel: Int,

publishedStates: Seq[PublishedState]

): MisinformationPolicyTransform =

filter(

Seq(

hasFilteringLevelAtLeast(filteringLevel),

hasPublishedStates(publishedStates)

))

def filterLevelAndStateAndLocalized(

filteringLevel: Int,

publishedStates: Seq[PublishedState]

): MisinformationPolicyTransform =

filter(

Seq(

hasFilteringLevelAtLeast(filteringLevel),

hasPublishedStates(publishedStates),

hasNonEmptyLocalization,

))

def filterState(

publishedStates: Seq[PublishedState]

): MisinformationPolicyTransform =

filter(

Seq(

hasPublishedStates(publishedStates)

))

def filterStateAndLocalized(

publishedStates: Seq[PublishedState]

): MisinformationPolicyTransform =

filter(

Seq(

hasPublishedStates(publishedStates),

hasNonEmptyLocalization,

))

def filterApplicableCountries(

countryCode: Option[String],

): MisinformationPolicyTransform =

filter(Seq(policyAppliesToCountry(countryCode)))

def filterOutGeoSpecific(): MisinformationPolicyTransform =

filter(Seq(policyIsGlobal()))

def filterNonEngagementNudges(): MisinformationPolicyTransform =

filter(

Seq(

hasEngagementNudge,

))

def policyAppliesToCountry(countryCode: Option[String]): MisinformationPolicy => Boolean =

policy =>

policy.applicableCountries.isEmpty ||

(countryCode.nonEmpty && policy.applicableCountries.contains(countryCode.get))

def policyIsGlobal(): MisinformationPolicy => Boolean =

policy => policy.applicableCountries.isEmpty

def hasFilteringLevelAtLeast(filteringLevel: Int): MisinformationPolicy => Boolean =

\_.filteringLevel >= filteringLevel

def hasPublishedStates(

publishedStates: Seq[PublishedState]

): MisinformationPolicy => Boolean =

policy => publishedStates.contains(policy.publishedState)

def hasNonEmptyLocalization: MisinformationPolicy => Boolean =

policy => policy.warning.nonEmpty && policy.detailsUrl.nonEmpty

def hasEngagementNudge: MisinformationPolicy => Boolean =

policy => policy.engagementNudge

}

sealed trait PublishedState

object PublishedState {

case object Draft extends PublishedState

case object Dogfood extends PublishedState

case object Published extends PublishedState

val PublicPublishedStates = Seq(PublishedState.Published)

val EmployeePublishedStates = Seq(PublishedState.Published, PublishedState.Dogfood)

}

sealed trait MisinfoPolicyDisplayType

object MisinfoPolicyDisplayType {

case object GetTheLatest extends MisinfoPolicyDisplayType

case object StayInformed extends MisinfoPolicyDisplayType

case object Misleading extends MisinfoPolicyDisplayType

case object GovernmentRequested extends MisinfoPolicyDisplayType

}

object SemanticCoreMisinformation {

val domainId: Long = 148L

}