package com.twitter.visibility.rules.generators

import com.twitter.visibility.models.SafetyLevel

import com.twitter.visibility.models.SafetyLevelGroup

import com.twitter.visibility.models.ViolationLevel

import com.twitter.visibility.rules.FreedomOfSpeechNotReachActions

import com.twitter.visibility.rules.FreedomOfSpeechNotReachRules

import com.twitter.visibility.rules.Rule

import com.twitter.visibility.rules.generators.TweetRuleGenerator.violationLevelPolicies

object TweetRuleGenerator {

private val level3LimitedActions: Seq[String] = Seq(

"like",

"reply",

"retweet",

"quote\_tweet",

"share\_tweet\_via",

"add\_to\_bookmarks",

"pin\_to\_profile",

"copy\_link",

"send\_via\_dm")

private val violationLevelPolicies: Map[

ViolationLevel,

Map[UserType, TweetVisibilityPolicy]

] = Map(

ViolationLevel.Level1 -> Map(

UserType.Follower -> TweetVisibilityPolicy

.builder()

.addGlobalRule(FreedomOfSpeechNotReachActions.SoftInterventionAvoidAction())

.addSafetyLevelGroupRule(

SafetyLevelGroup.Notifications,

FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelGroupRule(

SafetyLevelGroup.Recommendations,

FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelGroupRule(

SafetyLevelGroup.Search,

FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelGroupRule(

SafetyLevelGroup.TopicRecommendations,

FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelRule(

SafetyLevel.TimelineHomeRecommendations,

FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelRule(

SafetyLevel.TrendsRepresentativeTweet,

FreedomOfSpeechNotReachActions.DropAction())

.build,

UserType.Author -> TweetVisibilityPolicy

.builder()

.addGlobalRule(FreedomOfSpeechNotReachActions.AppealableAction())

.build,

UserType.Other -> TweetVisibilityPolicy

.builder()

.addGlobalRule(FreedomOfSpeechNotReachActions.SoftInterventionAvoidAction())

.addSafetyLevelGroupRule(

SafetyLevelGroup.Notifications,

FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelGroupRule(

SafetyLevelGroup.Recommendations,

FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelGroupRule(

SafetyLevelGroup.TimelineHome,

FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelGroupRule(

SafetyLevelGroup.Search,

FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelGroupRule(

SafetyLevelGroup.TopicRecommendations,

FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelRule(

SafetyLevel.TrendsRepresentativeTweet,

FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelRule(

SafetyLevel.ConversationReply,

FreedomOfSpeechNotReachActions.SoftInterventionAvoidAbusiveQualityReplyAction())

.build,

),

ViolationLevel.Level3 -> Map(

UserType.Follower -> TweetVisibilityPolicy

.builder()

.addGlobalRule(FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelGroupRule(

SafetyLevelGroup.TimelineProfile,

FreedomOfSpeechNotReachActions.SoftInterventionAvoidLimitedEngagementsAction(

limitedActionStrings = Some(level3LimitedActions))

)

.addSafetyLevelGroupRule(

SafetyLevelGroup.TweetDetails,

FreedomOfSpeechNotReachActions.SoftInterventionAvoidLimitedEngagementsAction(

limitedActionStrings = Some(level3LimitedActions))

)

.addSafetyLevelRule(

SafetyLevel.ConversationReply,

FreedomOfSpeechNotReachActions.SoftInterventionAvoidLimitedEngagementsAction(

limitedActionStrings = Some(level3LimitedActions))

)

.addSafetyLevelRule(

SafetyLevel.ConversationFocalTweet,

FreedomOfSpeechNotReachActions.SoftInterventionAvoidLimitedEngagementsAction(

limitedActionStrings = Some(level3LimitedActions))

)

.build,

UserType.Author -> TweetVisibilityPolicy

.builder()

.addGlobalRule(

FreedomOfSpeechNotReachActions.AppealableAvoidLimitedEngagementsAction(

limitedActionStrings = Some(level3LimitedActions))

)

.build,

UserType.Other -> TweetVisibilityPolicy

.builder()

.addGlobalRule(FreedomOfSpeechNotReachActions.DropAction())

.addSafetyLevelGroupRule(

SafetyLevelGroup.TimelineProfile,

FreedomOfSpeechNotReachActions

.InterstitialLimitedEngagementsAvoidAction(limitedActionStrings =

Some(level3LimitedActions))

)

.addSafetyLevelGroupRule(

SafetyLevelGroup.TweetDetails,

FreedomOfSpeechNotReachActions

.InterstitialLimitedEngagementsAvoidAction(limitedActionStrings =

Some(level3LimitedActions))

)

.addSafetyLevelRule(

SafetyLevel.ConversationReply,

FreedomOfSpeechNotReachActions

.InterstitialLimitedEngagementsAvoidAction(limitedActionStrings =

Some(level3LimitedActions))

)

.addSafetyLevelRule(

SafetyLevel.ConversationFocalTweet,

FreedomOfSpeechNotReachActions

.InterstitialLimitedEngagementsAvoidAction(limitedActionStrings =

Some(level3LimitedActions))

)

.build,

),

)

}

sealed trait UserType

object UserType {

case object Author extends UserType

case object Follower extends UserType

case object Other extends UserType

}

class TweetRuleGenerator extends RuleGenerator {

private[rules] val tweetRulesForSurface: Map[SafetyLevel, Seq[Rule]] = generateTweetPolicies()

private[rules] def getViolationLevelPolicies = violationLevelPolicies

override def rulesForSurface(safetyLevel: SafetyLevel): Seq[Rule] =

tweetRulesForSurface.getOrElse(safetyLevel, Seq())

private def generateRulesForPolicy(

violationLevel: ViolationLevel,

userType: UserType,

tweetVisibilityPolicy: TweetVisibilityPolicy

): Seq[(SafetyLevel, Rule)] = {

tweetVisibilityPolicy

.getRules()

.map {

case (safetyLevel, actionBuilder) =>

safetyLevel -> (userType match {

case UserType.Author =>

FreedomOfSpeechNotReachRules.ViewerIsAuthorAndTweetHasViolationOfLevel(

violationLevel = violationLevel,

actionBuilder = actionBuilder.withViolationLevel(violationLevel = violationLevel))

case UserType.Follower =>

FreedomOfSpeechNotReachRules.ViewerIsFollowerAndTweetHasViolationOfLevel(

violationLevel = violationLevel,

actionBuilder = actionBuilder.withViolationLevel(violationLevel = violationLevel))

case UserType.Other =>

FreedomOfSpeechNotReachRules.ViewerIsNonFollowerNonAuthorAndTweetHasViolationOfLevel(

violationLevel = violationLevel,

actionBuilder = actionBuilder.withViolationLevel(violationLevel = violationLevel))

})

}.toSeq

}

private def generatePoliciesForViolationLevel(

violationLevel: ViolationLevel

): Seq[(SafetyLevel, Rule)] = {

getViolationLevelPolicies

.get(violationLevel).map { policiesPerUserType =>

Seq(UserType.Author, UserType.Follower, UserType.Other).foldLeft(

List.empty[(UserType, SafetyLevel, Rule)]) {

case (rulesForAllUserTypes, userType) =>

rulesForAllUserTypes ++ generateRulesForPolicy(

violationLevel = violationLevel,

userType = userType,

tweetVisibilityPolicy = policiesPerUserType(userType)).map {

case (safetyLevel, rule) => (userType, safetyLevel, rule)

}

}

}

.map(policy => optimizePolicy(policy = policy, violationLevel = violationLevel))

.getOrElse(List())

}

private def injectFallbackRule(rules: Seq[Rule]): Seq[Rule] = {

rules :+ FreedomOfSpeechNotReachRules.TweetHasViolationOfAnyLevelFallbackDropRule

}

private def optimizePolicy(

policy: Seq[(UserType, SafetyLevel, Rule)],

violationLevel: ViolationLevel

): Seq[(SafetyLevel, Rule)] = {

val policiesByUserType = policy.groupBy { case (userType, \_, \_) => userType }.map {

case (userType, aggregated) =>

(userType, aggregated.map { case (\_, safetyLevel, rules) => (safetyLevel, rules) })

}

val followerPolicies = aggregateRulesBySafetyLevel(

policiesByUserType.getOrElse(UserType.Follower, Seq()))

val otherPolicies = aggregateRulesBySafetyLevel(

policiesByUserType.getOrElse(UserType.Other, Seq()))

policiesByUserType(UserType.Author) ++

followerPolicies.collect {

case (safetyLevel, rule) if !otherPolicies.contains(safetyLevel) =>

(safetyLevel, rule)

} ++

otherPolicies.collect {

case (safetyLevel, rule) if !followerPolicies.contains(safetyLevel) =>

(safetyLevel, rule)

} ++

followerPolicies.keySet

.intersect(otherPolicies.keySet).foldLeft(List.empty[(SafetyLevel, Rule)]) {

case (aggr, safetyLevel)

if followerPolicies(safetyLevel).actionBuilder == otherPolicies(

safetyLevel).actionBuilder =>

(

safetyLevel,

FreedomOfSpeechNotReachRules.ViewerIsNonAuthorAndTweetHasViolationOfLevel(

violationLevel = violationLevel,

actionBuilder = followerPolicies(safetyLevel).actionBuilder

)) :: aggr

case (aggr, safetyLevel) =>

(safetyLevel, followerPolicies(safetyLevel)) ::

(safetyLevel, otherPolicies(safetyLevel)) :: aggr

}

}

private def aggregateRulesBySafetyLevel(

policy: Seq[(SafetyLevel, Rule)]

): Map[SafetyLevel, Rule] = {

policy

.groupBy {

case (safetyLevel, \_) => safetyLevel

}.map {

case (safetyLevel, Seq((\_, rule))) =>

(safetyLevel, rule)

case \_ => throw new Exception("Policy optimization failure")

}

}

private def generateTweetPolicies(): Map[SafetyLevel, Seq[Rule]] = {

Seq(ViolationLevel.Level4, ViolationLevel.Level3, ViolationLevel.Level2, ViolationLevel.Level1)

.foldLeft(List.empty[(SafetyLevel, Rule)]) {

case (rulesForAllViolationLevels, violationLevel) =>

rulesForAllViolationLevels ++

generatePoliciesForViolationLevel(violationLevel)

}

.groupBy { case (safetyLevel, \_) => safetyLevel }

.map {

case (safetyLevel, list) =>

(safetyLevel, injectFallbackRule(list.map { case (\_, rule) => rule }))

}

}

}