package com.twitter.recos.decider

import com.twitter.decider.Decider

import com.twitter.decider.DeciderFactory

import com.twitter.decider.RandomRecipient

import com.twitter.decider.Recipient

import com.twitter.decider.SimpleRecipient

import com.twitter.recos.util.TeamUsers

case class GuestRecipient(id: Long) extends Recipient {

override def isGuest: Boolean = true

}

sealed trait BaseDecider {

def baseConfig: Option[String] = None

def overlayConfig: Option[String] = None

lazy val decider: Decider = DeciderFactory(baseConfig, overlayConfig)()

def isAvailable(feature: String, recipient: Option[Recipient]): Boolean =

decider.isAvailable(feature, recipient)

def isAvailable(feature: String): Boolean = isAvailable(feature, None)

def isAvailableExceptTeam(feature: String, id: Long, isUser: Boolean = true): Boolean = {

if (isUser) TeamUsers.team.contains(id) || isAvailable(feature, Some(SimpleRecipient(id)))

else isAvailable(feature, Some(GuestRecipient(id)))

}

}

case class RecosDecider(env: String, cluster: String = "atla") extends BaseDecider {

override val baseConfig = Some("/com/twitter/recos/config/decider.yml")

override val overlayConfig = Some(

s"/usr/local/config/overlays/recos/service/prod/$cluster/decider\_overlay.yml"

)

def shouldCompute(id: Long, displayLocation: String, isUser: Boolean = true): Boolean = {

isAvailableExceptTeam(RecosDecider.recosIncomingTraffic + "\_" + displayLocation, id, isUser)

}

def shouldReturn(id: Long, displayLocation: String, isUser: Boolean = true): Boolean = {

isAvailableExceptTeam(RecosDecider.recosShouldReturn + "\_" + displayLocation, id, isUser)

}

def shouldDarkmode(experiment: String): Boolean = {

isAvailable(RecosDecider.recosShouldDark + "\_exp\_" + experiment, None)

}

def shouldScribe(id: Long, isUser: Boolean = true): Boolean = {

if (isUser) (id > 0) && isAvailableExceptTeam(RecosDecider.recosShouldScribe, id, isUser)

else false // TODO: define the behavior for guests

}

def shouldWriteMomentCapsuleOpenEdge(): Boolean = {

val capsuleOpenDecider = env match {

case "prod" => RecosDecider.recosShouldWriteMomentCapsuleOpenEdge

case \_ => RecosDecider.recosShouldWriteMomentCapsuleOpenEdge + RecosDecider.testSuffix

}

isAvailable(capsuleOpenDecider, Some(RandomRecipient))

}

}

object RecosDecider {

val testSuffix = "\_test"

val recosIncomingTraffic: String = "recos\_incoming\_traffic"

val recosShouldReturn: String = "recos\_should\_return"

val recosShouldDark: String = "recos\_should\_dark"

val recosRealtimeBlacklist: String = "recos\_realtime\_blacklist"

val recosRealtimeDeveloperlist: String = "recos\_realtime\_developerlist"

val recosShouldScribe: String = "recos\_should\_scribe"

val recosShouldWriteMomentCapsuleOpenEdge: String = "recos\_should\_write\_moment\_capsule\_open\_edge"

}

trait GraphDecider extends BaseDecider {

val graphNamePrefix: String

override val baseConfig = Some("/com/twitter/recos/config/decider.yml")

override val overlayConfig = Some(

"/usr/local/config/overlays/recos/service/prod/atla/decider\_overlay.yml"

)

}

case class UserTweetEntityGraphDecider() extends GraphDecider {

override val graphNamePrefix: String = "user\_tweet\_entity\_graph"

def tweetSocialProof: Boolean = {

isAvailable("user\_tweet\_entity\_graph\_tweet\_social\_proof")

}

def entitySocialProof: Boolean = {

isAvailable("user\_tweet\_entity\_graph\_entity\_social\_proof")

}

}

case class UserUserGraphDecider() extends GraphDecider {

override val graphNamePrefix: String = "user\_user\_graph"

}

case class UserTweetGraphDecider(env: String, dc: String) extends GraphDecider {

override val graphNamePrefix: String = "user-tweet-graph"

override val baseConfig = Some("/com/twitter/recos/config/user-tweet-graph\_decider.yml")

override val overlayConfig = Some(

s"/usr/local/config/overlays/user-tweet-graph/user-tweet-graph/$env/$dc/decider\_overlay.yml"

)

}