package com.twitter.tweetypie

package handler

import com.twitter.botmaker.thriftscala.BotMakerResponse

import com.twitter.bouncer.thriftscala.Bounce

import com.twitter.finagle.tracing.Trace

import com.twitter.relevance.feature\_store.thriftscala.FeatureData

import com.twitter.relevance.feature\_store.thriftscala.FeatureValue.StrValue

import com.twitter.service.gen.scarecrow.thriftscala.TieredAction

import com.twitter.service.gen.scarecrow.thriftscala.TieredActionResult

import com.twitter.tweetypie.core.TweetCreateFailure

import com.twitter.tweetypie.thriftscala.TweetCreateState

object Spam {

sealed trait Result

case object Allow extends Result

case object SilentFail extends Result

case object DisabledByIpiPolicy extends Result

val AllowFuture: Future[Allow.type] = Future.value(Allow)

val SilentFailFuture: Future[SilentFail.type] = Future.value(SilentFail)

val DisabledByIpiPolicyFuture: Future[DisabledByIpiPolicy.type] =

Future.value(DisabledByIpiPolicy)

def DisabledByIpiFailure(

userName: Option[String],

customDenyMessage: Option[String] = None

): TweetCreateFailure.State = {

val errorMsg = (customDenyMessage, userName) match {

case (Some(denyMessage), \_) => denyMessage

case (\_, Some(name)) => s"Some actions on this ${name} Tweet have been disabled by Twitter."

case \_ => "Some actions on this Tweet have been disabled by Twitter."

}

TweetCreateFailure.State(TweetCreateState.DisabledByIpiPolicy, Some(errorMsg))

}

type Checker[T] = T => Future[Result]

/\*\*

\* Dummy spam checker that always allows requests.

\*/

val DoNotCheckSpam: Checker[AnyRef] = \_ => AllowFuture

def gated[T](gate: Gate[Unit])(checker: Checker[T]): Checker[T] =

req => if (gate()) checker(req) else AllowFuture

def selected[T](gate: Gate[Unit])(ifTrue: Checker[T], ifFalse: Checker[T]): Checker[T] =

req => gate.select(ifTrue, ifFalse)()(req)

def withEffect[T](check: Checker[T], effect: T => Unit): T => Future[Result] = { t: T =>

effect(t)

check(t)

}

/\*\*

\* Wrapper that implicitly allows retweet or tweet creation when spam

\* checking fails.

\*/

def allowOnException[T](checker: Checker[T]): Checker[T] =

req =>

checker(req).rescue {

case e: TweetCreateFailure => Future.exception(e)

case \_ => AllowFuture

}

/\*\*

\* Handler for scarecrow result to be used by a Checker.

\*/

def handleScarecrowResult(

stats: StatsReceiver

)(

handler: PartialFunction[(TieredActionResult, Option[Bounce], Option[String]), Future[Result]]

): Checker[TieredAction] =

result => {

stats.scope("scarecrow\_result").counter(result.resultCode.name).incr()

Trace.record("com.twitter.tweetypie.Spam.scarecrow\_result=" + result.resultCode.name)

/\*

\* A bot can return a custom DenyMessage

\*

\* If it does, we substitute this for the 'message' in the ValidationError.

\*/

val customDenyMessage: Option[String] = for {

botMakeResponse: BotMakerResponse <- result.botMakerResponse

outputFeatures <- botMakeResponse.outputFeatures

denyMessageFeature: FeatureData <- outputFeatures.get("DenyMessage")

denyMessageFeatureValue <- denyMessageFeature.featureValue

denyMessage <- denyMessageFeatureValue match {

case stringValue: StrValue =>

Some(stringValue.strValue)

case \_ =>

None

}

} yield denyMessage

handler.applyOrElse(

(result.resultCode, result.bounce, customDenyMessage),

withEffect(DoNotCheckSpam, (\_: AnyRef) => stats.counter("unexpected\_result").incr())

)

}

}