package com.twitter.tweetypie

package config

import com.twitter.featureswitches.v2.FeatureSwitches

import com.twitter.stitch.repo.Repo

import com.twitter.tweetypie.backends.LimiterService.Feature

import com.twitter.tweetypie.handler.\_

import com.twitter.tweetypie.jiminy.tweetypie.NudgeBuilder

import com.twitter.tweetypie.repository.RelationshipKey

import com.twitter.tweetypie.store.TotalTweetStore

import com.twitter.tweetypie.thriftscala.\_

import com.twitter.tweetypie.tweettext.TweetText

import com.twitter.visibility.common.TrustedFriendsSource

import com.twitter.visibility.common.UserRelationshipSource

import com.twitter.visibility.writer.interfaces.tweets.TweetWriteEnforcementLibrary

trait TweetBuilders {

val retweetBuilder: RetweetBuilder.Type

val tweetBuilder: TweetBuilder.Type

}

object TweetBuilders {

def validateCardRefAttachmentByUserAgentGate(

android: Gate[Unit],

nonAndroid: Gate[Unit]

): Gate[Option[String]] =

Gate[Option[String]] { (userAgent: Option[String]) =>

if (userAgent.exists(\_.startsWith("TwitterAndroid"))) {

android()

} else {

nonAndroid()

}

}

def apply(

settings: TweetServiceSettings,

statsReceiver: StatsReceiver,

deciderGates: TweetypieDeciderGates,

featureSwitchesWithExperiments: FeatureSwitches,

clients: BackendClients,

caches: Caches,

repos: LogicalRepositories,

tweetStore: TotalTweetStore,

hasMedia: Tweet => Boolean,

unretweetEdits: TweetDeletePathHandler.UnretweetEdits,

): TweetBuilders = {

val urlShortener =

UrlShortener.scribeMalware(clients.guano) {

UrlShortener.fromTalon(clients.talon.shorten)

}

val urlEntityBuilder = UrlEntityBuilder.fromShortener(urlShortener)

val geoBuilder =

GeoBuilder(

repos.placeRepo,

ReverseGeocoder.fromGeoduck(clients.geoduckGeohashLocate),

statsReceiver.scope("geo\_builder")

)

val replyCardUsersFinder: CardUsersFinder.Type = CardUsersFinder(repos.cardUsersRepo)

val selfThreadBuilder = SelfThreadBuilder(statsReceiver.scope("self\_thread\_builder"))

val replyBuilder =

ReplyBuilder(

repos.userIdentityRepo,

repos.optionalTweetRepo,

replyCardUsersFinder,

selfThreadBuilder,

repos.relationshipRepo,

repos.unmentionedEntitiesRepo,

deciderGates.enableRemoveUnmentionedImplicitMentions,

statsReceiver.scope("reply\_builder"),

TweetText.MaxMentions

)

val mediaBuilder =

MediaBuilder(

clients.mediaClient.processMedia,

CreateMediaTco(urlShortener),

statsReceiver.scope("media\_builder")

)

val validateAttachments =

AttachmentBuilder.validateAttachments(

statsReceiver,

validateCardRefAttachmentByUserAgentGate(

android = deciderGates.validateCardRefAttachmentAndroid,

nonAndroid = deciderGates.validateCardRefAttachmentNonAndroid

)

)

val attachmentBuilder =

AttachmentBuilder(

repos.optionalTweetRepo,

urlShortener,

validateAttachments,

statsReceiver.scope("attachment\_builder"),

deciderGates.denyNonTweetPermalinks

)

val validatePostTweetRequest: FutureEffect[PostTweetRequest] =

TweetBuilder.validateAdditionalFields[PostTweetRequest]

val validateRetweetRequest =

TweetBuilder.validateAdditionalFields[RetweetRequest]

val tweetIdGenerator =

() => clients.snowflakeClient.get()

val retweetSpamChecker =

Spam.gated(deciderGates.checkSpamOnRetweet) {

Spam.allowOnException(

ScarecrowRetweetSpamChecker(

statsReceiver.scope("retweet\_builder").scope("spam"),

repos.retweetSpamCheckRepo

)

)

}

val tweetSpamChecker =

Spam.gated(deciderGates.checkSpamOnTweet) {

Spam.allowOnException(

ScarecrowTweetSpamChecker.fromSpamCheckRepository(

statsReceiver.scope("tweet\_builder").scope("spam"),

repos.tweetSpamCheckRepo

)

)

}

val duplicateTweetFinder =

DuplicateTweetFinder(

settings = settings.duplicateTweetFinderSettings,

tweetSource = DuplicateTweetFinder.TweetSource.fromServices(

tweetRepo = repos.optionalTweetRepo,

getStatusTimeline = clients.timelineService.getStatusTimeline

)

)

val validateUpdateRateLimit =

RateLimitChecker.validate(

clients.limiterService.hasRemaining(Feature.Updates),

statsReceiver.scope("rate\_limits", Feature.Updates.name),

deciderGates.rateLimitByLimiterService

)

val tweetBuilderStats = statsReceiver.scope("tweet\_builder")

val updateUserCounts =

TweetBuilder.updateUserCounts(hasMedia)

val filterInvalidData =

TweetBuilder.filterInvalidData(

validateTweetMediaTags = TweetBuilder.validateTweetMediaTags(

tweetBuilderStats.scope("media\_tags\_filter"),

RateLimitChecker.getMaxMediaTags(

clients.limiterService.minRemaining(Feature.MediaTagCreate),

TweetBuilder.MaxMediaTagCount

),

repos.optionalUserRepo

),

cardReferenceBuilder = TweetBuilder.cardReferenceBuilder(

CardReferenceValidationHandler(clients.expandodo.checkAttachmentEligibility),

urlShortener

)

)

val rateLimitFailures =

PostTweet.RateLimitFailures(

validateLimit = RateLimitChecker.validate(

clients.limiterService.hasRemaining(Feature.TweetCreateFailure),

statsReceiver.scope("rate\_limits", Feature.TweetCreateFailure.name),

deciderGates.rateLimitTweetCreationFailure

),

clients.limiterService.incrementByOne(Feature.Updates),

clients.limiterService.incrementByOne(Feature.TweetCreateFailure)

)

val countFailures =

PostTweet.CountFailures[TweetBuilderResult](statsReceiver)

val tweetBuilderFilter: PostTweet.Filter[TweetBuilderResult] =

rateLimitFailures.andThen(countFailures)

val conversationControlBuilder = ConversationControlBuilder.fromUserIdentityRepo(

statsReceiver = statsReceiver.scope("conversation\_control\_builder"),

userIdentityRepo = repos.userIdentityRepo

)

val conversationControlValidator = ConversationControlBuilder.Validate(

useFeatureSwitchResults = deciderGates.useConversationControlFeatureSwitchResults,

statsReceiver = statsReceiver

)

val communitiesValidator: CommunitiesValidator.Type = CommunitiesValidator()

val collabControlBuilder: CollabControlBuilder.Type = CollabControlBuilder()

val userRelationshipSource = UserRelationshipSource.fromRepo(

Repo[UserRelationshipSource.Key, Unit, Boolean] { (key, \_) =>

repos.relationshipRepo(

RelationshipKey(key.subjectId, key.objectId, key.relationship)

)

}

)

val trustedFriendsSource =

TrustedFriendsSource.fromStrato(clients.stratoserverClient, statsReceiver)

val validateTweetWrite = TweetWriteValidator(

convoCtlRepo = repos.conversationControlRepo,

tweetWriteEnforcementLibrary = TweetWriteEnforcementLibrary(

userRelationshipSource,

trustedFriendsSource,

repos.userIsInvitedToConversationRepo,

repos.stratoSuperFollowEligibleRepo,

repos.tweetRepo,

statsReceiver.scope("tweet\_write\_enforcement\_library")

),

enableExclusiveTweetControlValidation = deciderGates.enableExclusiveTweetControlValidation,

enableTrustedFriendsControlValidation = deciderGates.enableTrustedFriendsControlValidation,

enableStaleTweetValidation = deciderGates.enableStaleTweetValidation

)

val nudgeBuilder = NudgeBuilder(

clients.stratoserverClient,

deciderGates.jiminyDarkRequests,

statsReceiver.scope("nudge\_builder")

)

val editControlBuilder = EditControlBuilder(

tweetRepo = repos.tweetRepo,

card2Repo = repos.card2Repo,

promotedTweetRepo = repos.stratoPromotedTweetRepo,

subscriptionVerificationRepo = repos.stratoSubscriptionVerificationRepo,

disablePromotedTweetEdit = deciderGates.disablePromotedTweetEdit,

checkTwitterBlueSubscription = deciderGates.checkTwitterBlueSubscriptionForEdit,

setEditWindowToSixtyMinutes = deciderGates.setEditTimeWindowToSixtyMinutes,

stats = statsReceiver,

)

val validateEdit = EditValidator(repos.optionalTweetRepo)

// TweetBuilders builds two distinct TweetBuilders (Tweet and Retweet builders).

new TweetBuilders {

val tweetBuilder: TweetBuilder.Type =

tweetBuilderFilter[PostTweetRequest](

TweetBuilder(

stats = tweetBuilderStats,

validateRequest = validatePostTweetRequest,

validateEdit = validateEdit,

validateUpdateRateLimit = validateUpdateRateLimit,

tweetIdGenerator = tweetIdGenerator,

userRepo = repos.userRepo,

deviceSourceRepo = repos.deviceSourceRepo,

communityMembershipRepo = repos.stratoCommunityMembershipRepo,

communityAccessRepo = repos.stratoCommunityAccessRepo,

urlShortener = urlShortener,

urlEntityBuilder = urlEntityBuilder,

geoBuilder = geoBuilder,

replyBuilder = replyBuilder,

mediaBuilder = mediaBuilder,

attachmentBuilder = attachmentBuilder,

duplicateTweetFinder = duplicateTweetFinder,

spamChecker = tweetSpamChecker,

filterInvalidData = filterInvalidData,

updateUserCounts = updateUserCounts,

validateConversationControl = conversationControlValidator,

conversationControlBuilder = conversationControlBuilder,

validateTweetWrite = validateTweetWrite,

nudgeBuilder = nudgeBuilder,

communitiesValidator = communitiesValidator,

collabControlBuilder = collabControlBuilder,

editControlBuilder = editControlBuilder,

featureSwitches = featureSwitchesWithExperiments,

)

)

val retweetBuilder: RetweetBuilder.Type =

tweetBuilderFilter[RetweetRequest](

RetweetBuilder(

validateRequest = validateRetweetRequest,

tweetIdGenerator = tweetIdGenerator,

tweetRepo = repos.tweetRepo,

userRepo = repos.userRepo,

tflock = clients.tflockWriteClient,

deviceSourceRepo = repos.deviceSourceRepo,

validateUpdateRateLimit = validateUpdateRateLimit,

spamChecker = retweetSpamChecker,

updateUserCounts = updateUserCounts,

superFollowRelationsRepo = repos.stratoSuperFollowRelationsRepo,

unretweetEdits = unretweetEdits,

setEditWindowToSixtyMinutes = deciderGates.setEditTimeWindowToSixtyMinutes

)

)

}

}

}