package com.twitter.tsp.common

import com.twitter.abdecider.LoggingABDecider

import com.twitter.abdecider.UserRecipient

import com.twitter.contentrecommender.thriftscala.DisplayLocation

import com.twitter.discovery.common.configapi.FeatureContextBuilder

import com.twitter.featureswitches.FSRecipient

import com.twitter.featureswitches.Recipient

import com.twitter.featureswitches.UserAgent

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.interests.thriftscala.TopicListingViewerContext

import com.twitter.timelines.configapi

import com.twitter.timelines.configapi.Params

import com.twitter.timelines.configapi.RequestContext

import com.twitter.timelines.configapi.abdecider.LoggingABDeciderExperimentContext

case class ParamsBuilder(

featureContextBuilder: FeatureContextBuilder,

abDecider: LoggingABDecider,

overridesConfig: configapi.Config,

statsReceiver: StatsReceiver) {

def buildFromTopicListingViewerContext(

topicListingViewerContext: Option[TopicListingViewerContext],

displayLocation: DisplayLocation,

userRoleOverride: Option[Set[String]] = None

): Params = {

topicListingViewerContext.flatMap(\_.userId) match {

case Some(userId) =>

val userRecipient = ParamsBuilder.toFeatureSwitchRecipientWithTopicContext(

userId,

userRoleOverride,

topicListingViewerContext,

Some(displayLocation)

)

overridesConfig(

requestContext = RequestContext(

userId = Some(userId),

experimentContext = LoggingABDeciderExperimentContext(

abDecider,

Some(UserRecipient(userId, Some(userId)))),

featureContext = featureContextBuilder(

Some(userId),

Some(userRecipient)

)

),

statsReceiver

)

case \_ =>

throw new IllegalArgumentException(

s"${this.getClass.getSimpleName} tried to build Param for a request without a userId"

)

}

}

}

object ParamsBuilder {

def toFeatureSwitchRecipientWithTopicContext(

userId: Long,

userRolesOverride: Option[Set[String]],

context: Option[TopicListingViewerContext],

displayLocationOpt: Option[DisplayLocation]

): Recipient = {

val userRoles = userRolesOverride match {

case Some(overrides) => Some(overrides)

case \_ => context.flatMap(\_.userRoles.map(\_.toSet))

}

val recipient = FSRecipient(

userId = Some(userId),

userRoles = userRoles,

deviceId = context.flatMap(\_.deviceId),

guestId = context.flatMap(\_.guestId),

languageCode = context.flatMap(\_.languageCode),

countryCode = context.flatMap(\_.countryCode),

userAgent = context.flatMap(\_.userAgent).flatMap(UserAgent(\_)),

isVerified = None,

isTwoffice = None,

tooClient = None,

highWaterMark = None

)

displayLocationOpt match {

case Some(displayLocation) =>

recipient.withCustomFields(displayLocationCustomFieldMap(displayLocation))

case None =>

recipient

}

}

private val DisplayLocationCustomField = "display\_location"

def displayLocationCustomFieldMap(displayLocation: DisplayLocation): (String, String) =

DisplayLocationCustomField -> displayLocation.toString

}