package com.twitter.follow\_recommendations.configapi

import com.google.common.annotations.VisibleForTesting

import com.google.inject.Inject

import com.twitter.decider.Decider

import com.twitter.featureswitches.v2.FeatureSwitches

import com.twitter.featureswitches.{Recipient => FeatureSwitchRecipient}

import com.twitter.follow\_recommendations.common.models.DisplayLocation

import com.twitter.product\_mixer.core.model.marshalling.request.ClientContext

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.timelines.configapi.FeatureContext

import com.twitter.timelines.configapi.FeatureValue

import com.twitter.timelines.configapi.ForcedFeatureContext

import com.twitter.timelines.configapi.OrElseFeatureContext

import com.twitter.timelines.configapi.featureswitches.v2.FeatureSwitchResultsFeatureContext

import javax.inject.Singleton

/\*

\* Request Context Factory is used to build RequestContext objects which are used

\* by the config api to determine the param overrides to apply to the request.

\* The param overrides are determined per request by configs which specify which

\* FS/Deciders/AB translate to what param overrides.

\*/

@Singleton

class RequestContextFactory @Inject() (featureSwitches: FeatureSwitches, decider: Decider) {

def apply(

clientContext: ClientContext,

displayLocation: DisplayLocation,

featureOverrides: Map[String, FeatureValue]

): RequestContext = {

val featureContext = getFeatureContext(clientContext, displayLocation, featureOverrides)

RequestContext(clientContext.userId, clientContext.guestId, featureContext)

}

private[configapi] def getFeatureContext(

clientContext: ClientContext,

displayLocation: DisplayLocation,

featureOverrides: Map[String, FeatureValue]

): FeatureContext = {

val recipient =

getFeatureSwitchRecipient(clientContext)

.withCustomFields("display\_location" -> displayLocation.toFsName)

// userAgeOpt is going to be set to None for logged out users and defaulted to Some(Int.MaxValue) for non-snowflake users

val userAgeOpt = clientContext.userId.map { userId =>

SnowflakeId.timeFromIdOpt(userId).map(\_.untilNow.inDays).getOrElse(Int.MaxValue)

}

val recipientWithAccountAge =

userAgeOpt

.map(age => recipient.withCustomFields("account\_age\_in\_days" -> age)).getOrElse(recipient)

val results = featureSwitches.matchRecipient(recipientWithAccountAge)

OrElseFeatureContext(

ForcedFeatureContext(featureOverrides),

new FeatureSwitchResultsFeatureContext(results))

}

@VisibleForTesting

private[configapi] def getFeatureSwitchRecipient(

clientContext: ClientContext

): FeatureSwitchRecipient = {

FeatureSwitchRecipient(

userId = clientContext.userId,

userRoles = clientContext.userRoles,

deviceId = clientContext.deviceId,

guestId = clientContext.guestId,

languageCode = clientContext.languageCode,

countryCode = clientContext.countryCode,

isVerified = None,

clientApplicationId = clientContext.appId,

isTwoffice = clientContext.isTwoffice

)

}

}