package com.twitter.frigate.pushservice.util

import com.twitter.frigate.common.store.deviceinfo.DeviceInfo

import com.twitter.frigate.pushservice.model.PushTypes.Target

import com.twitter.frigate.pushservice.params.PushFeatureSwitchParams.EnableRuxLandingPage

import com.twitter.frigate.pushservice.params.PushParams.EnableRuxLandingPageAndroidParam

import com.twitter.frigate.pushservice.params.PushParams.EnableRuxLandingPageIOSParam

import com.twitter.frigate.pushservice.params.PushParams.RuxLandingPageExperimentKeyAndroidParam

import com.twitter.frigate.pushservice.params.PushParams.RuxLandingPageExperimentKeyIOSParam

import com.twitter.frigate.pushservice.params.PushParams.ShowRuxLandingPageAsModalOnIOS

import com.twitter.rux.common.context.thriftscala.MagicRecsNTabTweet

import com.twitter.rux.common.context.thriftscala.MagicRecsPushTweet

import com.twitter.rux.common.context.thriftscala.RuxContext

import com.twitter.rux.common.context.thriftscala.Source

import com.twitter.rux.common.encode.RuxContextEncoder

/\*\*

\* This class provides utility functions for email landing page for push

\*/

object EmailLandingPageExperimentUtil {

val ruxCxtEncoder = new RuxContextEncoder()

def getIbis2ModelValue(

deviceInfoOpt: Option[DeviceInfo],

target: Target,

tweetId: Long

): Map[String, String] = {

val enable = enablePushEmailLanding(deviceInfoOpt, target)

if (enable) {

val ruxCxt = if (deviceInfoOpt.exists(\_.isRuxLandingPageEligible)) {

val encodedCxt = getRuxContext(tweetId, target, deviceInfoOpt)

Map("rux\_cxt" -> encodedCxt)

} else Map.empty[String, String]

val enableModal = if (showModalForIOS(deviceInfoOpt, target)) {

Map("enable\_modal" -> "true")

} else Map.empty[String, String]

Map("land\_on\_email\_landing\_page" -> "true") ++ ruxCxt ++ enableModal

} else Map.empty[String, String]

}

def createNTabRuxLandingURI(screenName: String, tweetId: Long): String = {

val encodedCxt =

ruxCxtEncoder.encode(RuxContext(Some(Source.MagicRecsNTabTweet(MagicRecsNTabTweet(tweetId)))))

s"$screenName/status/${tweetId.toString}?cxt=$encodedCxt"

}

private def getRuxContext(

tweetId: Long,

target: Target,

deviceInfoOpt: Option[DeviceInfo]

): String = {

val isDeviceIOS = PushDeviceUtil.isPrimaryDeviceIOS(deviceInfoOpt)

val isDeviceAndroid = PushDeviceUtil.isPrimaryDeviceAndroid(deviceInfoOpt)

val keyOpt = if (isDeviceIOS) {

target.params(RuxLandingPageExperimentKeyIOSParam)

} else if (isDeviceAndroid) {

target.params(RuxLandingPageExperimentKeyAndroidParam)

} else None

val context = RuxContext(Some(Source.MagicRecsTweet(MagicRecsPushTweet(tweetId))), None, keyOpt)

ruxCxtEncoder.encode(context)

}

private def enablePushEmailLanding(

deviceInfoOpt: Option[DeviceInfo],

target: Target

): Boolean =

deviceInfoOpt.exists(deviceInfo =>

if (deviceInfo.isEmailLandingPageEligible) {

val isRuxLandingPageEnabled = target.params(EnableRuxLandingPage)

isRuxLandingPageEnabled && isRuxLandingEnabledBasedOnDeviceInfo(deviceInfoOpt, target)

} else false)

private def showModalForIOS(deviceInfoOpt: Option[DeviceInfo], target: Target): Boolean = {

deviceInfoOpt.exists { deviceInfo =>

deviceInfo.isRuxLandingPageAsModalEligible && target.params(ShowRuxLandingPageAsModalOnIOS)

}

}

private def isRuxLandingEnabledBasedOnDeviceInfo(

deviceInfoOpt: Option[DeviceInfo],

target: Target

): Boolean = {

val isDeviceIOS = PushDeviceUtil.isPrimaryDeviceIOS(deviceInfoOpt)

val isDeviceAndroid = PushDeviceUtil.isPrimaryDeviceAndroid(deviceInfoOpt)

if (isDeviceIOS) {

target.params(EnableRuxLandingPageIOSParam)

} else if (isDeviceAndroid) {

target.params(EnableRuxLandingPageAndroidParam)

} else true

}

}