package com.twitter.frigate.pushservice.model.ibis

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.TweetAuthorDetails

import com.twitter.frigate.common.base.TweetCandidate

import com.twitter.frigate.common.base.TweetDetails

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

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

import com.twitter.frigate.pushservice.params.SubtextForAndroidPushHeader

import com.twitter.frigate.pushservice.params.{PushFeatureSwitchParams => FS}

import com.twitter.frigate.pushservice.util.CopyUtil

import com.twitter.frigate.pushservice.util.EmailLandingPageExperimentUtil

import com.twitter.frigate.pushservice.util.InlineActionUtil

import com.twitter.frigate.pushservice.util.PushToHomeUtil

import com.twitter.frigate.pushservice.util.PushIbisUtil.mergeFutModelValues

import com.twitter.util.Future

trait TweetCandidateIbis2Hydrator

extends Ibis2HydratorForCandidate

with InlineActionIbis2Hydrator

with CustomConfigurationMapForIbis {

self: PushCandidate with TweetCandidate with TweetDetails with TweetAuthorDetails =>

lazy val scopedStats: StatsReceiver = statsReceiver.scope(getClass.getSimpleName)

lazy val tweetIdModelValue: Map[String, String] =

Map(

"tweet" -> tweetId.toString

)

lazy val authorModelValue: Map[String, String] = {

assert(authorId.isDefined)

Map(

"author" -> authorId.getOrElse(0L).toString

)

}

lazy val otherModelValues: Map[String, String] =

Map(

"show\_explanatory\_text" -> "true",

"show\_negative\_feedback" -> "true"

)

lazy val mediaModelValue: Map[String, String] =

Map(

"show\_media" -> "true"

)

lazy val inlineVideoMediaMap: Map[String, String] = {

if (hasVideo) {

val isInlineVideoEnabled = target.params(FS.EnableInlineVideo)

val isAutoplayEnabled = target.params(FS.EnableAutoplayForInlineVideo)

Map(

"enable\_inline\_video\_for\_ios" -> isInlineVideoEnabled.toString,

"enable\_autoplay\_for\_inline\_video\_ios" -> isAutoplayEnabled.toString

)

} else Map.empty

}

lazy val landingPageModelValues: Future[Map[String, String]] = {

for {

deviceInfoOpt <- target.deviceInfo

} yield {

PushToHomeUtil.getIbis2ModelValue(deviceInfoOpt, target, scopedStats) match {

case Some(pushToHomeModelValues) => pushToHomeModelValues

case \_ =>

EmailLandingPageExperimentUtil.getIbis2ModelValue(

deviceInfoOpt,

target,

tweetId

)

}

}

}

lazy val tweetDynamicInlineActionsModelValues = {

if (target.params(PushFeatureSwitchParams.EnableTweetDynamicInlineActions)) {

val actions = target.params(PushFeatureSwitchParams.TweetDynamicInlineActionsList)

InlineActionUtil.getGeneratedTweetInlineActions(target, statsReceiver, actions)

} else Map.empty[String, String]

}

lazy val tweetDynamicInlineActionsModelValuesForWeb: Map[String, String] = {

if (target.isLoggedOutUser) {

Map.empty[String, String]

} else {

InlineActionUtil.getGeneratedTweetInlineActionsForWeb(

actions = target.params(PushFeatureSwitchParams.TweetDynamicInlineActionsListForWeb),

enableForDesktopWeb =

target.params(PushFeatureSwitchParams.EnableDynamicInlineActionsForDesktopWeb),

enableForMobileWeb =

target.params(PushFeatureSwitchParams.EnableDynamicInlineActionsForMobileWeb)

)

}

}

lazy val copyFeaturesFut: Future[Map[String, String]] =

CopyUtil.getCopyFeatures(self, scopedStats)

private def getVerifiedSymbolModelValue: Future[Map[String, String]] = {

self.tweetAuthor.map {

case Some(author) =>

if (author.safety.exists(\_.verified)) {

scopedStats.counter("is\_verified").incr()

if (target.params(FS.EnablePushPresentationVerifiedSymbol)) {

scopedStats.counter("is\_verified\_and\_add").incr()

Map("is\_author\_verified" -> "true")

} else {

scopedStats.counter("is\_verified\_and\_NOT\_add").incr()

Map.empty

}

} else {

scopedStats.counter("is\_NOT\_verified").incr()

Map.empty

}

case \_ =>

scopedStats.counter("none\_author").incr()

Map.empty

}

}

private def subtextAndroidPushHeader: Map[String, String] = {

self.target.params(PushFeatureSwitchParams.SubtextInAndroidPushHeaderParam) match {

case SubtextForAndroidPushHeader.None =>

Map.empty

case SubtextForAndroidPushHeader.TargetHandler =>

Map("subtext\_target\_handler" -> "true")

case SubtextForAndroidPushHeader.TargetTagHandler =>

Map("subtext\_target\_tag\_handler" -> "true")

case SubtextForAndroidPushHeader.TargetName =>

Map("subtext\_target\_name" -> "true")

case SubtextForAndroidPushHeader.AuthorTagHandler =>

Map("subtext\_author\_tag\_handler" -> "true")

case SubtextForAndroidPushHeader.AuthorName =>

Map("subtext\_author\_name" -> "true")

case \_ =>

Map.empty

}

}

lazy val bodyPushMap: Map[String, String] = {

if (self.target.params(PushFeatureSwitchParams.EnableEmptyBody)) {

Map("enable\_empty\_body" -> "true")

} else Map.empty[String, String]

}

override def customFieldsMapFut: Future[Map[String, String]] =

for {

superModelValues <- super.customFieldsMapFut

copyFeaturesModelValues <- copyFeaturesFut

verifiedSymbolModelValue <- getVerifiedSymbolModelValue

} yield {

superModelValues ++ copyFeaturesModelValues ++

verifiedSymbolModelValue ++ subtextAndroidPushHeader ++ bodyPushMap

}

override lazy val senderId: Option[Long] = authorId

def tweetModelValues: Future[Map[String, String]] =

landingPageModelValues.map { landingPageModelValues =>

tweetIdModelValue ++ authorModelValue ++ landingPageModelValues ++ tweetDynamicInlineActionsModelValues ++ tweetDynamicInlineActionsModelValuesForWeb

}

override lazy val modelValues: Future[Map[String, String]] =

mergeFutModelValues(super.modelValues, tweetModelValues)

}