package com.twitter.unified\_user\_actions.adapter.client\_event

import com.twitter.clientapp.thriftscala.EventNamespace

import com.twitter.clientapp.thriftscala.Item

import com.twitter.clientapp.thriftscala.ItemType.User

import com.twitter.clientapp.thriftscala.LogEvent

import com.twitter.clientapp.thriftscala.{Item => LogEventItem}

import com.twitter.unified\_user\_actions.adapter.common.AdapterUtils

import com.twitter.unified\_user\_actions.thriftscala.AuthorInfo

import com.twitter.unified\_user\_actions.thriftscala.ClientEventNamespace

import com.twitter.unified\_user\_actions.thriftscala.EventMetadata

import com.twitter.unified\_user\_actions.thriftscala.ProductSurface

import com.twitter.unified\_user\_actions.thriftscala.SourceLineage

import com.twitter.unified\_user\_actions.thriftscala.TweetAuthorFollowClickSource

import com.twitter.unified\_user\_actions.thriftscala.TweetAuthorUnfollowClickSource

import com.twitter.unified\_user\_actions.thriftscala.TweetInfo

/\*\*

\* Comprises helper methods that:

\* 1. need not be overridden by subclasses of `BaseClientEvent`

\* 2. need not be invoked by instances of subclasses of `BaseClientEvent`

\* 3. need to be accessible to subclasses of `BaseClientEvent` and other utils

\*/

object ClientEventCommonUtils {

def getBasicTweetInfo(

actionTweetId: Long,

ceItem: LogEventItem,

ceNamespaceOpt: Option[EventNamespace]

): TweetInfo = TweetInfo(

actionTweetId = actionTweetId,

actionTweetTopicSocialProofId = getTopicId(ceItem, ceNamespaceOpt),

retweetingTweetId = ceItem.tweetDetails.flatMap(\_.retweetingTweetId),

quotedTweetId = ceItem.tweetDetails.flatMap(\_.quotedTweetId),

inReplyToTweetId = ceItem.tweetDetails.flatMap(\_.inReplyToTweetId),

quotingTweetId = ceItem.tweetDetails.flatMap(\_.quotingTweetId),

// only set AuthorInfo when authorId is present

actionTweetAuthorInfo = getAuthorInfo(ceItem),

retweetingAuthorId = ceItem.tweetDetails.flatMap(\_.retweetAuthorId),

quotedAuthorId = ceItem.tweetDetails.flatMap(\_.quotedAuthorId),

inReplyToAuthorId = ceItem.tweetDetails.flatMap(\_.inReplyToAuthorId),

tweetPosition = ceItem.position,

promotedId = ceItem.promotedId

)

def getTopicId(

ceItem: LogEventItem,

ceNamespaceOpt: Option[EventNamespace] = None,

): Option[Long] =

ceNamespaceOpt.flatMap {

TopicIdUtils.getTopicId(item = ceItem, \_)

}

def getAuthorInfo(

ceItem: LogEventItem,

): Option[AuthorInfo] =

ceItem.tweetDetails.flatMap(\_.authorId).map { authorId =>

AuthorInfo(

authorId = Some(authorId),

isFollowedByActingUser = ceItem.isViewerFollowsTweetAuthor,

isFollowingActingUser = ceItem.isTweetAuthorFollowsViewer,

)

}

def getEventMetadata(

eventTimestamp: Long,

logEvent: LogEvent,

ceItem: LogEventItem,

productSurface: Option[ProductSurface] = None

): EventMetadata = EventMetadata(

sourceTimestampMs = eventTimestamp,

receivedTimestampMs = AdapterUtils.currentTimestampMs,

sourceLineage = SourceLineage.ClientEvents,

// Client UI language or from Gizmoduck which is what user set in Twitter App.

// Please see more at https://sourcegraph.twitter.biz/git.twitter.biz/source/-/blob/finatra-internal/international/src/main/scala/com/twitter/finatra/international/LanguageIdentifier.scala

// The format should be ISO 639-1.

language = logEvent.logBase.flatMap(\_.language).map(AdapterUtils.normalizeLanguageCode),

// Country code could be IP address (geoduck) or User registration country (gizmoduck) and the former takes precedence.

// We don’t know exactly which one is applied, unfortunately,

// see https://sourcegraph.twitter.biz/git.twitter.biz/source/-/blob/finatra-internal/international/src/main/scala/com/twitter/finatra/international/CountryIdentifier.scala

// The format should be ISO\_3166-1\_alpha-2.

countryCode = logEvent.logBase.flatMap(\_.country).map(AdapterUtils.normalizeCountryCode),

clientAppId = logEvent.logBase.flatMap(\_.clientAppId),

clientVersion = logEvent.clientVersion,

clientEventNamespace = logEvent.eventNamespace.map(en => toClientEventNamespace(en)),

traceId = getTraceId(productSurface, ceItem),

requestJoinId = getRequestJoinId(productSurface, ceItem),

clientEventTriggeredOn = logEvent.eventDetails.flatMap(\_.triggeredOn)

)

def toClientEventNamespace(eventNamespace: EventNamespace): ClientEventNamespace =

ClientEventNamespace(

page = eventNamespace.page,

section = eventNamespace.section,

component = eventNamespace.component,

element = eventNamespace.element,

action = eventNamespace.action

)

/\*\*

\* Get the profileId from Item.id, which itemType = 'USER'.

\*

\* The profileId can be also be found in the event\_details.profile\_id.

\* However, the item.id is more reliable than event\_details.profile\_id,

\* in particular, 45% of the client events with USER items have

\* Null for event\_details.profile\_id while 0.13% item.id is Null.

\* As such, we only use item.id to populate the profile\_id.

\*/

def getProfileIdFromUserItem(item: Item): Option[Long] =

if (item.itemType.contains(User))

item.id

else None

/\*\*

\* TraceId is going to be deprecated and replaced by requestJoinId.

\*

\* Get the traceId from LogEventItem based on productSurface.

\*

\* The traceId is hydrated in controller data from backend. Different product surfaces

\* populate different controller data. Thus, the product surface is checked first to decide

\* which controller data should be read to ge the requestJoinId.

\*/

def getTraceId(productSurface: Option[ProductSurface], ceItem: LogEventItem): Option[Long] =

productSurface match {

case Some(ProductSurface.HomeTimeline) => HomeInfoUtils.getTraceId(ceItem)

case Some(ProductSurface.SearchResultsPage) => { new SearchInfoUtils(ceItem) }.getTraceId

case \_ => None

}

/\*\*

\* Get the requestJoinId from LogEventItem based on productSurface.

\*

\* The requestJoinId is hydrated in controller data from backend. Different product surfaces

\* populate different controller data. Thus, the product surface is checked first to decide

\* which controller data should be read to get the requestJoinId.

\*

\* Support Home / Home\_latest / SearchResults for now, to add other surfaces based on requirement.

\*/

def getRequestJoinId(productSurface: Option[ProductSurface], ceItem: LogEventItem): Option[Long] =

productSurface match {

case Some(ProductSurface.HomeTimeline) => HomeInfoUtils.getRequestJoinId(ceItem)

case Some(ProductSurface.SearchResultsPage) => {

new SearchInfoUtils(ceItem)

}.getRequestJoinId

case \_ => None

}

def getTweetAuthorFollowSource(

eventNamespace: Option[EventNamespace]

): TweetAuthorFollowClickSource = {

eventNamespace

.map(ns => (ns.element, ns.action)).map {

case (Some("follow"), Some("click")) => TweetAuthorFollowClickSource.CaretMenu

case (\_, Some("follow")) => TweetAuthorFollowClickSource.ProfileImage

case \_ => TweetAuthorFollowClickSource.Unknown

}.getOrElse(TweetAuthorFollowClickSource.Unknown)

}

def getTweetAuthorUnfollowSource(

eventNamespace: Option[EventNamespace]

): TweetAuthorUnfollowClickSource = {

eventNamespace

.map(ns => (ns.element, ns.action)).map {

case (Some("unfollow"), Some("click")) => TweetAuthorUnfollowClickSource.CaretMenu

case (\_, Some("unfollow")) => TweetAuthorUnfollowClickSource.ProfileImage

case \_ => TweetAuthorUnfollowClickSource.Unknown

}.getOrElse(TweetAuthorUnfollowClickSource.Unknown)

}

}