package com.twitter.product\_mixer.component\_library.side\_effect

import com.twitter.abdecider.ScribingABDeciderUtil

import com.twitter.clientapp.thriftscala.LogEvent

import com.twitter.product\_mixer.core.model.common.presentation.CandidateWithDetails

import com.twitter.product\_mixer.core.model.marshalling.HasMarshalling

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.scribelib.marshallers

import com.twitter.scribelib.marshallers.ClientDataProvider

import com.twitter.scribelib.marshallers.LogEventMarshaller

/\*\*

\* Side effect to log client events server-side. Create an implementation of this trait by

\* defining the `buildClientEvents` method, and the `page` val.

\* The ClientEvent will be automatically converted into a [[LogEvent]] and scribed.

\*/

trait ScribeClientEventSideEffect[

Query <: PipelineQuery,

UnmarshalledResponseType <: HasMarshalling]

extends ScribeLogEventSideEffect[LogEvent, Query, UnmarshalledResponseType] {

/\*\*

\* The page which will be defined in the namespace. This is typically the service name that's scribing

\*/

val page: String

/\*\*

\* Build the client events from query, selections and response

\*

\* @param query PipelineQuery

\* @param selectedCandidates Result after Selectors are executed

\* @param remainingCandidates Candidates which were not selected

\* @param droppedCandidates Candidates dropped during selection

\* @param response Result after Unmarshalling

\*/

def buildClientEvents(

query: Query,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: UnmarshalledResponseType

): Seq[ScribeClientEventSideEffect.ClientEvent]

final override def buildLogEvents(

query: Query,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: UnmarshalledResponseType

): Seq[LogEvent] = {

buildClientEvents(

query = query,

selectedCandidates = selectedCandidates,

remainingCandidates = remainingCandidates,

droppedCandidates = droppedCandidates,

response = response).flatMap { event =>

val clientData = clientContextToClientDataProvider(query)

val clientName = ScribingABDeciderUtil.clientForAppId(clientData.clientApplicationId)

val namespaceMap: Map[String, String] = Map(

"client" -> Some(clientName),

"page" -> Some(page),

"section" -> event.namespace.section,

"component" -> event.namespace.component,

"element" -> event.namespace.element,

"action" -> event.namespace.action

).collect { case (k, Some(v)) => k -> v }

val data: Map[Any, Any] = Seq(

event.eventValue.map("event\_value" -> \_),

event.latencyMs.map("latency\_ms" -> \_)

).flatten.toMap

val clientEventData = data +

("event\_namespace" -> namespaceMap) +

(marshallers.CategoryKey -> "client\_event")

LogEventMarshaller.marshal(

data = clientEventData,

clientData = clientData

)

}

}

/\*\*

\* Makes a [[ClientDataProvider]] from the [[PipelineQuery.clientContext]] from the [[query]]

\*/

private def clientContextToClientDataProvider(query: Query): ClientDataProvider = {

new ClientDataProvider {

override val userId = query.clientContext.userId

override val guestId = query.clientContext.guestId

override val personalizationId = None

override val deviceId = query.clientContext.deviceId

override val clientApplicationId = query.clientContext.appId

override val parentApplicationId = None

override val countryCode = query.clientContext.countryCode

override val languageCode = query.clientContext.languageCode

override val userAgent = query.clientContext.userAgent

override val isSsl = None

override val referer = None

override val externalReferer = None

}

}

}

object ScribeClientEventSideEffect {

case class EventNamespace(

section: Option[String] = None,

component: Option[String] = None,

element: Option[String] = None,

action: Option[String] = None)

case class ClientEvent(

namespace: EventNamespace,

eventValue: Option[Long] = None,

latencyMs: Option[Long] = None)

}