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

import com.twitter.clientapp.thriftscala.LogEvent

import com.twitter.logpipeline.client.common.EventPublisher

import com.twitter.product\_mixer.component\_library.side\_effect.ScribeClientEventSideEffect

import com.twitter.product\_mixer.component\_library.side\_effect.ScribeClientEventSideEffect.EventNamespace

import com.twitter.product\_mixer.component\_library.side\_effect.ScribeClientEventSideEffect.ClientEvent

import com.twitter.product\_mixer.core.model.common.identifier.SideEffectIdentifier

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

/\*\*

\* Config of a client event to be scribed under certain namespace.

\* @param eventNamespaceOverride overrides the default eventNamespace in the side effect.

\* Note that its fields (section/component/element/action) will

\* override the default namespace fields only if the fields are not

\* None. i.e. if you want to override the "section" field in the

\* default namespace with an empty section, you must specify

\* section = Some("")

\* in the override instead of

\* section = None

\*

\* @param metricFunction the function that extracts the metric value from a candidate.

\*/

case class EventConfig(

eventNamespaceOverride: EventNamespace,

metricFunction: CandidateMetricFunction)

/\*\*

\* Side effect to log client events server-side and to build metrics in the metric center.

\* By default will return "requests" event config.

\*/

class ScribeClientEventMetricsSideEffect[

Query <: PipelineQuery,

UnmarshalledResponseType <: HasMarshalling

](

override val identifier: SideEffectIdentifier,

override val logPipelinePublisher: EventPublisher[LogEvent],

override val page: String,

defaultEventNamespace: EventNamespace,

eventConfigs: Seq[EventConfig])

extends ScribeClientEventSideEffect[Query, UnmarshalledResponseType] {

override def buildClientEvents(

query: Query,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: UnmarshalledResponseType

): Seq[ScribeClientEventSideEffect.ClientEvent] = {

// count the number of client events of type "requests"

val requestClientEvent = ClientEvent(

namespace = buildEventNamespace(EventNamespace(action = Some("requests")))

)

eventConfigs

.map { config =>

ClientEvent(

namespace = buildEventNamespace(config.eventNamespaceOverride),

eventValue = Some(selectedCandidates.map(config.metricFunction(\_)).sum))

}

// scribe client event only when the metric sum is non-zero

.filter(clientEvent => clientEvent.eventValue.exists(\_ > 0L)) :+ requestClientEvent

}

private def buildEventNamespace(eventNamespaceOverride: EventNamespace): EventNamespace =

EventNamespace(

section = eventNamespaceOverride.section.orElse(defaultEventNamespace.section),

component = eventNamespaceOverride.component.orElse(defaultEventNamespace.component),

element = eventNamespaceOverride.element.orElse(defaultEventNamespace.element),

action = eventNamespaceOverride.action.orElse(defaultEventNamespace.action)

)

}