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

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

import com.twitter.product\_mixer.core.functional\_component.side\_effect.PipelineResultSideEffect

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.scrooge.ThriftStruct

import com.twitter.stitch.Stitch

/\*\*

\* A [[PipelineResultSideEffect]] that logs [[Thrift]] data that's already available to Scribe

\*/

trait ScribeLogEventSideEffect[

Thrift <: ThriftStruct,

Query <: PipelineQuery,

ResponseType <: HasMarshalling]

extends PipelineResultSideEffect[Query, ResponseType] {

/\*\*

\* Build the log 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

\* @return LogEvent in thrift

\*/

def buildLogEvents(

query: Query,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: ResponseType

): Seq[Thrift]

val logPipelinePublisher: EventPublisher[Thrift]

final override def apply(

inputs: PipelineResultSideEffect.Inputs[Query, ResponseType]

): Stitch[Unit] = {

val logEvents = buildLogEvents(

query = inputs.query,

selectedCandidates = inputs.selectedCandidates,

remainingCandidates = inputs.remainingCandidates,

droppedCandidates = inputs.droppedCandidates,

response = inputs.response

)

Stitch

.collect(

logEvents

.map { logEvent =>

Stitch.callFuture(logPipelinePublisher.publish(logEvent))

}

).unit

}

}