package com.twitter.recosinjector.edges

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.Stats.track

import com.twitter.util.Future

/\*\*

\* This is the generic interface that converts incoming Events (ex. TweetEvent, FavEvent, etc)

\* into Edge for a specific output graph. It applies the following flow:

\*

\* event -> update event stats -> build edges -> filter edges

\*

\* Top-level statistics are provided for each step, such as latency and number of events

\*/

trait EventToMessageBuilder[Event, E <: Edge] {

implicit val statsReceiver: StatsReceiver

private lazy val processEventStats = statsReceiver.scope("process\_event")

private lazy val numEventsStats = statsReceiver.counter("num\_process\_event")

private lazy val rejectEventStats = statsReceiver.counter("num\_reject\_event")

private lazy val buildEdgesStats = statsReceiver.scope("build")

private lazy val numAllEdgesStats = buildEdgesStats.counter("num\_all\_edges")

private lazy val filterEdgesStats = statsReceiver.scope("filter")

private lazy val numValidEdgesStats = statsReceiver.counter("num\_valid\_edges")

private lazy val numRecosHoseMessageStats = statsReceiver.counter("num\_RecosHoseMessage")

/\*\*

\* Given an incoming event, process and convert it into a sequence of RecosHoseMessages

\* @param event

\* @return

\*/

def processEvent(event: Event): Future[Seq[Edge]] = {

track(processEventStats) {

shouldProcessEvent(event).flatMap {

case true =>

numEventsStats.incr()

updateEventStatus(event)

for {

allEdges <- track(buildEdgesStats)(buildEdges(event))

filteredEdges <- track(filterEdgesStats)(filterEdges(event, allEdges))

} yield {

numAllEdgesStats.incr(allEdges.size)

numValidEdgesStats.incr(filteredEdges.size)

numRecosHoseMessageStats.incr(filteredEdges.size)

filteredEdges

}

case false =>

rejectEventStats.incr()

Future.Nil

}

}

}

/\*\*

\* Pre-process filter that determines whether the given event should be used to build edges.

\* @param event

\* @return

\*/

def shouldProcessEvent(event: Event): Future[Boolean]

/\*\*

\* Update cache/event logging related to the specific event.

\* By default, no action will be taken. Override when necessary

\* @param event

\*/

def updateEventStatus(event: Event): Unit = {}

/\*\*

\* Given an event, extract info and build a sequence of edges

\* @param event

\* @return

\*/

def buildEdges(event: Event): Future[Seq[E]]

/\*\*

\* Given a sequence of edges, filter and return the valid edges

\* @param event

\* @param edges

\* @return

\*/

def filterEdges(event: Event, edges: Seq[E]): Future[Seq[E]]

}