package com.twitter.product\_mixer.component\_library.candidate\_source.ann

import com.twitter.ann.common.\_

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.CandidateSource

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

import com.twitter.stitch.Stitch

import com.twitter.util.{Time => \_, \_}

import com.twitter.finagle.util.DefaultTimer

/\*\*

\* @param annQueryableById Ann Queryable by Id client that returns nearest neighbors for a sequence of queries

\* @param identifier Candidate Source Identifier

\* @tparam T1 type of the query.

\* @tparam T2 type of the result.

\* @tparam P runtime parameters supported by the index.

\* @tparam D distance function used in the index.

\*/

class AnnCandidateSource[T1, T2, P <: RuntimeParams, D <: Distance[D]](

val annQueryableById: QueryableById[T1, T2, P, D],

val batchSize: Int,

val timeoutPerRequest: Duration,

override val identifier: CandidateSourceIdentifier)

extends CandidateSource[AnnIdQuery[T1, P], NeighborWithDistanceWithSeed[T1, T2, D]] {

implicit val timer = DefaultTimer

override def apply(

request: AnnIdQuery[T1, P]

): Stitch[Seq[NeighborWithDistanceWithSeed[T1, T2, D]]] = {

val ids = request.ids

val numOfNeighbors = request.numOfNeighbors

val runtimeParams = request.runtimeParams

Stitch

.collect(

ids

.grouped(batchSize).map { batchedIds =>

annQueryableById

.batchQueryWithDistanceById(batchedIds, numOfNeighbors, runtimeParams).map {

annResult => annResult.toSeq

}.within(timeoutPerRequest).handle { case \_ => Seq.empty }

}.toSeq).map(\_.flatten)

}

}