package com.twitter.product\_mixer.component\_library.filter.list\_visibility

import com.twitter.product\_mixer.component\_library.model.candidate.TwitterListCandidate

import com.twitter.product\_mixer.core.functional\_component.filter.Filter

import com.twitter.product\_mixer.core.functional\_component.filter.FilterResult

import com.twitter.product\_mixer.core.model.common.CandidateWithFeatures

import com.twitter.product\_mixer.core.model.common.UniversalNoun

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

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

import com.twitter.socialgraph.thriftscala.SocialgraphList

import com.twitter.stitch.Stitch

import com.twitter.strato.catalog.Fetch

import com.twitter.strato.generated.client.lists.reads.CoreOnListClientColumn

/\* This Filter queries the core.List.strato column

\* on Strato, and filters out any lists that are not

\* returned. core.List.strato performs an authorization

\* check, and does not return lists the viewer is not authorized

\* to have access to. \*/

class ListVisibilityFilter[Candidate <: UniversalNoun[Long]](

listsColumn: CoreOnListClientColumn)

extends Filter[PipelineQuery, Candidate] {

override val identifier: FilterIdentifier = FilterIdentifier("ListVisibility")

def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[Candidate]]

): Stitch[FilterResult[Candidate]] = {

val listCandidates = candidates.collect {

case CandidateWithFeatures(candidate: TwitterListCandidate, \_) => candidate

}

Stitch

.traverse(

listCandidates.map(\_.id)

) { listId =>

listsColumn.fetcher.fetch(listId)

}.map { fetchResults =>

fetchResults.collect {

case Fetch.Result(Some(list: SocialgraphList), \_) => list.id

}

}.map { allowedListIds =>

val (kept, excluded) = candidates.map(\_.candidate).partition {

case candidate: TwitterListCandidate => allowedListIds.contains(candidate.id)

case \_ => true

}

FilterResult(kept, excluded)

}

}

}