package com.twitter.tweetypie

package store

import com.twitter.gizmoduck.thriftscala.{CountsUpdateField => Field}

import com.twitter.tweetypie.backends.Gizmoduck

trait GizmoduckUserCountsUpdatingStore

extends TweetStoreBase[GizmoduckUserCountsUpdatingStore]

with InsertTweet.Store

with DeleteTweet.Store {

def wrap(w: TweetStore.Wrap): GizmoduckUserCountsUpdatingStore =

new TweetStoreWrapper(w, this)

with GizmoduckUserCountsUpdatingStore

with InsertTweet.StoreWrapper

with DeleteTweet.StoreWrapper

}

/\*\*

\* A TweetStore implementation that sends user-specific count updates to Gizmoduck.

\*/

object GizmoduckUserCountsUpdatingStore {

def isUserTweet(tweet: Tweet): Boolean =

!TweetLenses.nullcast.get(tweet) && TweetLenses.narrowcast.get(tweet).isEmpty

def apply(

incr: Gizmoduck.IncrCount,

hasMedia: Tweet => Boolean

): GizmoduckUserCountsUpdatingStore = {

def incrField(field: Field, amt: Int): FutureEffect[Tweet] =

FutureEffect[Tweet](tweet => incr((getUserId(tweet), field, amt)))

def incrAll(amt: Int): FutureEffect[Tweet] =

FutureEffect.inParallel(

incrField(Field.Tweets, amt).onlyIf(isUserTweet),

incrField(Field.MediaTweets, amt).onlyIf(t => isUserTweet(t) && hasMedia(t))

)

new GizmoduckUserCountsUpdatingStore {

override val insertTweet: FutureEffect[InsertTweet.Event] =

incrAll(1).contramap[InsertTweet.Event](\_.tweet)

override val deleteTweet: FutureEffect[DeleteTweet.Event] =

incrAll(-1)

.contramap[DeleteTweet.Event](\_.tweet)

.onlyIf(!\_.isUserErasure)

}

}

}