package com.twitter.tweetypie

package core

import com.twitter.featureswitches.v2.FeatureSwitchResults

import com.twitter.tweetypie.thriftscala.\_

object TweetData {

object Lenses {

val tweet: Lens[TweetData, Tweet] = Lens[TweetData, Tweet](\_.tweet, \_.copy(\_))

val suppress: Lens[TweetData, Option[FilteredState.Suppress]] =

Lens[TweetData, Option[FilteredState.Suppress]](

\_.suppress,

(td, suppress) => td.copy(suppress = suppress)

)

val sourceTweetResult: Lens[TweetData, Option[TweetResult]] =

Lens[TweetData, Option[TweetResult]](

\_.sourceTweetResult,

(td, sourceTweetResult) => td.copy(sourceTweetResult = sourceTweetResult)

)

val quotedTweetResult: Lens[TweetData, Option[QuotedTweetResult]] =

Lens[TweetData, Option[QuotedTweetResult]](

\_.quotedTweetResult,

(td, quotedTweetResult) => td.copy(quotedTweetResult = quotedTweetResult)

)

val cacheableTweetResult: Lens[TweetData, Option[TweetResult]] =

Lens[TweetData, Option[TweetResult]](

\_.cacheableTweetResult,

(td, cacheableTweetResult) => td.copy(cacheableTweetResult = cacheableTweetResult)

)

val tweetCounts: Lens[TweetData, Option[StatusCounts]] =

Lens[TweetData, Option[StatusCounts]](

\_.tweet.counts,

(td, tweetCounts) => td.copy(tweet = td.tweet.copy(counts = tweetCounts))

)

}

def fromCachedTweet(cachedTweet: CachedTweet, cachedAt: Time): TweetData =

TweetData(

tweet = cachedTweet.tweet,

completedHydrations = cachedTweet.completedHydrations.toSet,

cachedAt = Some(cachedAt),

isBounceDeleted = cachedTweet.isBounceDeleted.contains(true)

)

}

/\*\*

\* Encapsulates a tweet and some hydration metadata in the hydration pipeline.

\*

\* @param cachedAt if the tweet was read from cache, `cachedAt` contains the time at which

\* the tweet was written to cache.

\*/

case class TweetData(

tweet: Tweet,

suppress: Option[FilteredState.Suppress] = None,

completedHydrations: Set[HydrationType] = Set.empty,

cachedAt: Option[Time] = None,

sourceTweetResult: Option[TweetResult] = None,

quotedTweetResult: Option[QuotedTweetResult] = None,

cacheableTweetResult: Option[TweetResult] = None,

storedTweetResult: Option[StoredTweetResult] = None,

featureSwitchResults: Option[FeatureSwitchResults] = None,

// The isBounceDeleted flag is only used when reading from an underlying

// tweet repo and caching records for not-found tweets. It only exists

// as a flag on TweetData to marshal bounce-deleted through the layered

// transforming caches injected into CachingTweetRepository, ultimately

// storing this flag in thrift on CachedTweet.

//

// During tweet hydration, TweetData.isBounceDeleted is unused and

// should always be false.

isBounceDeleted: Boolean = false) {

def addHydrated(fieldIds: Set[HydrationType]): TweetData =

copy(completedHydrations = completedHydrations ++ fieldIds)

def toCachedTweet: CachedTweet =

CachedTweet(

tweet = tweet,

completedHydrations = completedHydrations,

isBounceDeleted = if (isBounceDeleted) Some(true) else None

)

}