package com.twitter.tweetypie.util

import com.twitter.servo.util.Gate

import com.twitter.tweetypie.util.TweetEditFailure.TweetEditInvalidEditControlException

import com.twitter.tweetypie.util.TweetEditFailure.TweetEditUpdateEditControlException

import com.twitter.tweetypie.thriftscala.EditControl

import com.twitter.tweetypie.thriftscala.EditControlEdit

import com.twitter.tweetypie.thriftscala.EditControlInitial

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.util.Try

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Time

import com.twitter.util.Duration

object EditControlUtil {

val maxTweetEditsAllowed = 5

val oldEditTimeWindow = Duration.fromMinutes(30)

val editTimeWindow = Duration.fromMinutes(60)

def editControlEdit(

initialTweetId: TweetId,

editControlInitial: Option[EditControlInitial] = None

): EditControl.Edit =

EditControl.Edit(

EditControlEdit(initialTweetId = initialTweetId, editControlInitial = editControlInitial))

// EditControl for the tweet that is not an edit, that is, any regular tweet we create

// that can, potentially, be edited later.

def makeEditControlInitial(

tweetId: TweetId,

createdAt: Time,

setEditWindowToSixtyMinutes: Gate[Unit] = Gate(\_ => false)

): EditControl.Initial = {

val editWindow = if (setEditWindowToSixtyMinutes()) editTimeWindow else oldEditTimeWindow

val initial = EditControlInitial(

editTweetIds = Seq(tweetId),

editableUntilMsecs = Some(createdAt.plus(editWindow).inMilliseconds),

editsRemaining = Some(maxTweetEditsAllowed),

isEditEligible = defaultIsEditEligible,

)

EditControl.Initial(initial)

}

// Returns if a given latestTweetId is the latest edit in the EditControl

def isLatestEdit(

tweetEditControl: Option[EditControl],

latestTweetId: TweetId

): Try[Boolean] = {

tweetEditControl match {

case Some(EditControl.Initial(initial)) =>

isLatestEditFromEditControlInitial(Some(initial), latestTweetId)

case Some(EditControl.Edit(edit)) =>

isLatestEditFromEditControlInitial(

edit.editControlInitial,

latestTweetId

)

case \_ => Throw(TweetEditInvalidEditControlException)

}

}

// Returns if a given latestTweetId is the latest edit in the EditControlInitial

private def isLatestEditFromEditControlInitial(

initialTweetEditControl: Option[EditControlInitial],

latestTweetId: TweetId

): Try[Boolean] = {

initialTweetEditControl match {

case Some(initial) =>

Return(latestTweetId == initial.editTweetIds.last)

case \_ => Throw(TweetEditInvalidEditControlException)

}

}

/\* Create an updated edit control for an initialTweet given the id of the new edit \*/

def editControlForInitialTweet(

initialTweet: Tweet,

newEditId: TweetId

): Try[EditControl.Initial] = {

initialTweet.editControl match {

case Some(EditControl.Initial(initial)) =>

Return(EditControl.Initial(plusEdit(initial, newEditId)))

case Some(EditControl.Edit(\_)) => Throw(TweetEditUpdateEditControlException)

case \_ =>

initialTweet.coreData match {

case Some(coreData) =>

Return(

makeEditControlInitial(

tweetId = initialTweet.id,

createdAt = Time.fromMilliseconds(coreData.createdAtSecs \* 1000),

setEditWindowToSixtyMinutes = Gate(\_ => true)

)

)

case None => Throw(new Exception("Tweet Missing Required CoreData"))

}

}

}

def updateEditControl(tweet: Tweet, newEditId: TweetId): Try[Tweet] =

editControlForInitialTweet(tweet, newEditId).map { editControl =>

tweet.copy(editControl = Some(editControl))

}

def plusEdit(initial: EditControlInitial, newEditId: TweetId): EditControlInitial = {

val newEditTweetIds = (initial.editTweetIds :+ newEditId).distinct.sorted

val editsCount = newEditTweetIds.size - 1 // as there is the original tweet ID there too.

initial.copy(

editTweetIds = newEditTweetIds,

editsRemaining = Some(maxTweetEditsAllowed - editsCount),

)

}

// The ID of the initial Tweet if this is an edit

def getInitialTweetIdIfEdit(tweet: Tweet): Option[TweetId] = tweet.editControl match {

case Some(EditControl.Edit(edit)) => Some(edit.initialTweetId)

case \_ => None

}

// If this is the first tweet in an edit chain, return the same tweet id

// otherwise return the result of getInitialTweetId

def getInitialTweetId(tweet: Tweet): TweetId =

getInitialTweetIdIfEdit(tweet).getOrElse(tweet.id)

def isInitialTweet(tweet: Tweet): Boolean =

getInitialTweetId(tweet) == tweet.id

// Extracted just so that we can easily track where the values of isEditEligible is coming from.

private def defaultIsEditEligible: Option[Boolean] = Some(true)

// returns true if it's an edit of a Tweet or an initial Tweet that's been edited

def isEditTweet(tweet: Tweet): Boolean =

tweet.editControl match {

case Some(eci: EditControl.Initial) if eci.initial.editTweetIds.size <= 1 => false

case Some(\_: EditControl.Initial) | Some(\_: EditControl.Edit) | Some(

EditControl.UnknownUnionField(\_)) =>

true

case None => false

}

// returns true if editControl is from an edit of a Tweet

// returns false for any other state, including edit intial.

def isEditControlEdit(editControl: EditControl): Boolean = {

editControl match {

case \_: EditControl.Edit | EditControl.UnknownUnionField(\_) => true

case \_ => false

}

}

def getEditTweetIds(editControl: Option[EditControl]): Try[Seq[TweetId]] = {

editControl match {

case Some(EditControl.Edit(EditControlEdit(\_, Some(eci)))) =>

Return(eci.editTweetIds)

case Some(EditControl.Initial(initial)) =>

Return(initial.editTweetIds)

case \_ =>

Throw(new Exception(s"EditControlInitial not found in $editControl"))

}

}

}

object TweetEditFailure {

abstract class TweetEditException(msg: String) extends Exception(msg)

case object TweetEditGetInitialEditControlException

extends TweetEditException("Initial EditControl not found")

case object TweetEditInvalidEditControlException

extends TweetEditException("Invalid EditControl for initial\_tweet")

case object TweetEditUpdateEditControlException

extends TweetEditException("Invalid Edit Control Update")

}