package com.twitter.tweetypie

package handler

import com.twitter.tweetypie.core.TweetCreateFailure

import com.twitter.tweetypie.thriftscala.CollabControl

import com.twitter.tweetypie.thriftscala.CollabControlOptions

import com.twitter.tweetypie.thriftscala.CollabInvitation

import com.twitter.tweetypie.thriftscala.CollabInvitationOptions

import com.twitter.tweetypie.thriftscala.CollabInvitationStatus

import com.twitter.tweetypie.thriftscala.CollabTweet

import com.twitter.tweetypie.thriftscala.CollabTweetOptions

import com.twitter.tweetypie.thriftscala.Communities

import com.twitter.tweetypie.thriftscala.ExclusiveTweetControl

import com.twitter.tweetypie.thriftscala.InvitedCollaborator

import com.twitter.tweetypie.thriftscala.TrustedFriendsControl

import com.twitter.tweetypie.thriftscala.TweetCreateConversationControl

import com.twitter.tweetypie.thriftscala.TweetCreateState.CollabTweetInvalidParams

import com.twitter.tweetypie.util.CommunityUtil

object CollabControlBuilder {

type Type = Request => Future[Option[CollabControl]]

case class Request(

collabControlOptions: Option[CollabControlOptions],

replyResult: Option[ReplyBuilder.Result],

communities: Option[Communities],

trustedFriendsControl: Option[TrustedFriendsControl],

conversationControl: Option[TweetCreateConversationControl],

exclusiveTweetControl: Option[ExclusiveTweetControl],

userId: UserId)

def apply(): Type = { request =>

val collabControl = convertToCollabControl(request.collabControlOptions, request.userId)

validateCollabControlParams(

collabControl,

request.replyResult,

request.communities,

request.trustedFriendsControl,

request.conversationControl,

request.exclusiveTweetControl,

request.userId

) map { \_ => collabControl }

}

def convertToCollabControl(

collabTweetOptions: Option[CollabControlOptions],

authorId: UserId

): Option[CollabControl] = {

collabTweetOptions flatMap {

case CollabControlOptions.CollabInvitation(

collabInvitationOptions: CollabInvitationOptions) =>

Some(

CollabControl.CollabInvitation(

CollabInvitation(

invitedCollaborators = collabInvitationOptions.collaboratorUserIds.map(userId => {

InvitedCollaborator(

collaboratorUserId = userId,

collabInvitationStatus =

if (userId == authorId)

CollabInvitationStatus.Accepted

else CollabInvitationStatus.Pending

)

})

)

)

)

case CollabControlOptions.CollabTweet(collabTweetOptions: CollabTweetOptions) =>

Some(

CollabControl.CollabTweet(

CollabTweet(

collaboratorUserIds = collabTweetOptions.collaboratorUserIds

)

)

)

case \_ => None

}

}

def validateCollabControlParams(

collabControl: Option[CollabControl],

replyResult: Option[ReplyBuilder.Result],

communities: Option[Communities],

trustedFriendsControl: Option[TrustedFriendsControl],

conversationControl: Option[TweetCreateConversationControl],

exclusiveTweetControl: Option[ExclusiveTweetControl],

userId: UserId

): Future[Unit] = {

val isInReplyToTweet = replyResult.exists(\_.reply.inReplyToStatusId.isDefined)

collabControl match {

case Some(\_: CollabControl)

if (isInReplyToTweet ||

CommunityUtil.hasCommunity(communities) ||

exclusiveTweetControl.isDefined ||

trustedFriendsControl.isDefined ||

conversationControl.isDefined) =>

Future.exception(TweetCreateFailure.State(CollabTweetInvalidParams))

case Some(CollabControl.CollabInvitation(collab\_invitation))

if collab\_invitation.invitedCollaborators.head.collaboratorUserId != userId =>

Future.exception(TweetCreateFailure.State(CollabTweetInvalidParams))

case Some(CollabControl.CollabTweet(collab\_tweet))

if collab\_tweet.collaboratorUserIds.head != userId =>

Future.exception(TweetCreateFailure.State(CollabTweetInvalidParams))

case \_ =>

Future.Unit

}

}

}