package com.twitter.tweetypie

package handler

import com.twitter.scrooge.schema.scrooge.scala.CompiledScroogeDefBuilder

import com.twitter.scrooge.schema.scrooge.scala.CompiledScroogeValueExtractor

import com.twitter.scrooge.schema.tree.DefinitionTraversal

import com.twitter.scrooge.schema.tree.FieldPath

import com.twitter.scrooge.schema.{ThriftDefinitions => DEF}

import com.twitter.scrooge\_internal.linter.known\_annotations.AllowedAnnotationKeys.TweetEditAllowed

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.TweetCreateFailure

import com.twitter.tweetypie.repository.TweetQuery.Options

import com.twitter.tweetypie.repository.TweetQuery

import com.twitter.tweetypie.repository.TweetRepository

import com.twitter.tweetypie.thriftscala.ConversationControl

import com.twitter.tweetypie.thriftscala.TweetCreateState.FieldEditNotAllowed

import com.twitter.tweetypie.thriftscala.TweetCreateState.InitialTweetNotFound

import com.twitter.tweetypie.thriftscala.EditOptions

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.util.Future

import com.twitter.util.logging.Logger

/\*\*

\* This class constructs a validator `Tweet => Future[Unit]` which

\* takes a new edit tweet and performs some validations. Specifically, it

\*

\* 1) ensures that no uneditable fields were edited. Uneditable fields are marked

\* on the tweet.thrift using the thrift annotation "tweetEditAllowed=false".

\* By default, fields with no annotation are treated as editable.

\*

\* 2) ensures that the conversationControl field (which is editable) remains the

\* same type, e.g. a ConversationControl.ByInvitation doesn't change to a

\* ConversationControl.Community.

\*

\* If either of these validations fail, the validator fails with a `FieldEditNotAllowed`

\* tweet create state.

\*/

object EditValidator {

type Type = (Tweet, Option[EditOptions]) => Future[Unit]

val log: Logger = Logger(getClass)

// An object that describes the tweet thrift, used to walk a tweet object looking

// for annotated fields.

val TweetDef = CompiledScroogeDefBuilder.build[Tweet].asInstanceOf[DEF.StructDef]

// Collect the `FieldPath` for any nested tweet field with a uneditable field annotation

// that is set to false. These are the fields that this validator ensures cannot be edited.

val uneditableFieldPaths: Seq[FieldPath] = {

DefinitionTraversal().collect(TweetDef) {

case (d: DEF.FieldDef, path) if (d.annotations.get(TweetEditAllowed).contains("false")) =>

path

}

}

// A tweet query options which includes

// - any top level tweet field which either is an uneditable field, or contains an uneditable

// subfield.

// - the conversationControl field

// These fields must be present on the initial tweet in order for us to compare them against the

// edit tweet.

val previousTweetQueryOptions = {

// A set of the top level field ids for each (potentially nested) uneditable field.

val topLevelUneditableTweetFields = uneditableFieldPaths.map(\_.ids.head).toSet

Options(

TweetQuery.Include(

tweetFields = topLevelUneditableTweetFields + Tweet.ConversationControlField.id

))

}

def validateUneditableFields(previousTweet: Tweet, editTweet: Tweet): Unit = {

// Collect uneditable fields that were edited

val invalidEditedFields = uneditableFieldPaths.flatMap { fieldPath =>

val previousValue =

FieldPath.lensGet(CompiledScroogeValueExtractor, previousTweet, fieldPath)

val editValue = FieldPath.lensGet(CompiledScroogeValueExtractor, editTweet, fieldPath)

if (previousValue != editValue) {

Some(fieldPath.toString)

} else {

None

}

}

if (invalidEditedFields.nonEmpty) {

// If any inequalities are found, log them and return an exception.

val msg = "uneditable fields were edited: " + invalidEditedFields.mkString(",")

log.error(msg)

throw TweetCreateFailure.State(FieldEditNotAllowed, Some(msg))

}

}

def validateConversationControl(

previous: Option[ConversationControl],

edit: Option[ConversationControl]

): Unit = {

import ConversationControl.ByInvitation

import ConversationControl.Community

import ConversationControl.Followers

(previous, edit) match {

case (None, None) => ()

case (Some(ByInvitation(\_)), Some(ByInvitation(\_))) => ()

case (Some(Community(\_)), Some(Community(\_))) => ()

case (Some(Followers(\_)), Some(Followers(\_))) => ()

case (\_, \_) =>

val msg = "conversationControl type was edited"

log.error(msg)

throw TweetCreateFailure.State(FieldEditNotAllowed, Some(msg))

}

}

def apply(tweetRepo: TweetRepository.Optional): Type = { (tweet, editOptions) =>

Stitch.run(

editOptions match {

case Some(EditOptions(previousTweetId)) => {

// Query for the previous tweet so that we can compare the

// fields between the two tweets.

tweetRepo(previousTweetId, previousTweetQueryOptions).map {

case Some(previousTweet) =>

validateUneditableFields(previousTweet, tweet)

validateConversationControl(

previousTweet.conversationControl,

tweet.conversationControl)

case \_ =>

// If the previous tweet is not found we cannot perform validations that

// compare tweet fields and we have to fail tweet creation.

throw TweetCreateFailure.State(InitialTweetNotFound)

}

}

// This is the case where this isn't an edit tweet (since editOptions = None)

// Since this tweet is not an edit there are no fields to validate.

case \_ => Stitch.Unit

}

)

}

}