package com.twitter.unified\_user\_actions.adapter

import com.twitter.clientapp.thriftscala.\_

import com.twitter.clientapp.thriftscala.SuggestionDetails

import com.twitter.guide.scribing.thriftscala.\_

import com.twitter.guide.scribing.thriftscala.{SemanticCoreInterest => SemanticCoreInterestV1}

import com.twitter.guide.scribing.thriftscala.{SimClusterInterest => SimClusterInterestV1}

import com.twitter.guide.scribing.thriftscala.TopicModuleMetadata.SemanticCoreInterest

import com.twitter.guide.scribing.thriftscala.TopicModuleMetadata.SimClusterInterest

import com.twitter.guide.scribing.thriftscala.TransparentGuideDetails.TopicMetadata

import com.twitter.logbase.thriftscala.LogBase

import com.twitter.scrooge.TFieldBlob

import com.twitter.suggests.controller\_data.home\_hitl\_topic\_annotation\_prompt.thriftscala.HomeHitlTopicAnnotationPromptControllerData

import com.twitter.suggests.controller\_data.home\_hitl\_topic\_annotation\_prompt.v1.thriftscala.{

HomeHitlTopicAnnotationPromptControllerData => HomeHitlTopicAnnotationPromptControllerDataV1

}

import com.twitter.suggests.controller\_data.home\_topic\_annotation\_prompt.thriftscala.HomeTopicAnnotationPromptControllerData

import com.twitter.suggests.controller\_data.home\_topic\_annotation\_prompt.v1.thriftscala.{

HomeTopicAnnotationPromptControllerData => HomeTopicAnnotationPromptControllerDataV1

}

import com.twitter.suggests.controller\_data.home\_topic\_follow\_prompt.thriftscala.HomeTopicFollowPromptControllerData

import com.twitter.suggests.controller\_data.home\_topic\_follow\_prompt.v1.thriftscala.{

HomeTopicFollowPromptControllerData => HomeTopicFollowPromptControllerDataV1

}

import com.twitter.suggests.controller\_data.home\_tweets.thriftscala.HomeTweetsControllerData

import com.twitter.suggests.controller\_data.home\_tweets.v1.thriftscala.{

HomeTweetsControllerData => HomeTweetsControllerDataV1

}

import com.twitter.suggests.controller\_data.search\_response.item\_types.thriftscala.ItemTypesControllerData

import com.twitter.suggests.controller\_data.search\_response.thriftscala.SearchResponseControllerData

import com.twitter.suggests.controller\_data.search\_response.topic\_follow\_prompt.thriftscala.SearchTopicFollowPromptControllerData

import com.twitter.suggests.controller\_data.search\_response.tweet\_types.thriftscala.TweetTypesControllerData

import com.twitter.suggests.controller\_data.search\_response.v1.thriftscala.{

SearchResponseControllerData => SearchResponseControllerDataV1

}

import com.twitter.suggests.controller\_data.thriftscala.ControllerData

import com.twitter.suggests.controller\_data.timelines\_topic.thriftscala.TimelinesTopicControllerData

import com.twitter.suggests.controller\_data.timelines\_topic.v1.thriftscala.{

TimelinesTopicControllerData => TimelinesTopicControllerDataV1

}

import com.twitter.suggests.controller\_data.v2.thriftscala.{ControllerData => ControllerDataV2}

import org.apache.thrift.protocol.TField

import org.junit.runner.RunWith

import org.scalatest.funsuite.AnyFunSuite

import org.scalatest.matchers.should.Matchers

import org.scalatestplus.junit.JUnitRunner

import com.twitter.util.mock.Mockito

import org.mockito.Mockito.when

import org.scalatest.prop.TableDrivenPropertyChecks

@RunWith(classOf[JUnitRunner])

class TopicsIdUtilsSpec

extends AnyFunSuite

with Matchers

with Mockito

with TableDrivenPropertyChecks {

import com.twitter.unified\_user\_actions.adapter.client\_event.TopicIdUtils.\_

trait Fixture {

def buildLogBase(userId: Long): LogBase = {

val logBase = mock[LogBase]

when(logBase.country).thenReturn(Some("US"))

when(logBase.userId).thenReturn(Some(userId))

when(logBase.timestamp).thenReturn(100L)

when(logBase.guestId).thenReturn(Some(1L))

when(logBase.userAgent).thenReturn(None)

when(logBase.language).thenReturn(Some("en"))

logBase

}

def buildItemForTimeline(

itemId: Long,

itemType: ItemType,

topicId: Long,

fn: Long => ControllerData.V2

): Item = {

val item = Item(

id = Some(itemId),

itemType = Some(itemType),

suggestionDetails = Some(SuggestionDetails(decodedControllerData = Some(fn(topicId))))

)

item

}

def buildClientEventForHomeSearchTimeline(

itemId: Long,

itemType: ItemType,

topicId: Long,

fn: Long => ControllerData.V2,

userId: Long = 1L,

eventNamespaceOpt: Option[EventNamespace] = None,

): LogEvent = {

val logEvent = mock[LogEvent]

when(logEvent.eventNamespace).thenReturn(eventNamespaceOpt)

val eventsDetails = mock[EventDetails]

when(eventsDetails.items)

.thenReturn(Some(Seq(buildItemForTimeline(itemId, itemType, topicId, fn))))

val logbase = buildLogBase(userId)

when(logEvent.logBase).thenReturn(Some(logbase))

when(logEvent.eventDetails).thenReturn(Some(eventsDetails))

logEvent

}

def buildClientEventForHomeTweetsTimeline(

itemId: Long,

itemType: ItemType,

topicId: Long,

topicIds: Set[Long],

fn: (Long, Set[Long]) => ControllerData.V2,

userId: Long = 1L,

eventNamespaceOpt: Option[EventNamespace] = None,

): LogEvent = {

val logEvent = mock[LogEvent]

when(logEvent.eventNamespace).thenReturn(eventNamespaceOpt)

val eventsDetails = mock[EventDetails]

when(eventsDetails.items)

.thenReturn(Some(Seq(buildItemForHomeTimeline(itemId, itemType, topicId, topicIds, fn))))

val logbase = buildLogBase(userId)

when(logEvent.logBase).thenReturn(Some(logbase))

when(logEvent.eventDetails).thenReturn(Some(eventsDetails))

logEvent

}

def buildClientEventForGuide(

itemId: Long,

itemType: ItemType,

topicId: Long,

fn: Long => TopicMetadata,

userId: Long = 1L,

eventNamespaceOpt: Option[EventNamespace] = None,

): LogEvent = {

val logEvent = mock[LogEvent]

when(logEvent.eventNamespace).thenReturn(eventNamespaceOpt)

val logbase = buildLogBase(userId)

when(logEvent.logBase).thenReturn(Some(logbase))

val eventDetails = mock[EventDetails]

val item = buildItemForGuide(itemId, itemType, topicId, fn)

when(eventDetails.items).thenReturn(Some(Seq(item)))

when(logEvent.eventDetails).thenReturn(Some(eventDetails))

logEvent

}

def buildClientEventForOnboarding(

itemId: Long,

topicId: Long,

userId: Long = 1L

): LogEvent = {

val logEvent = mock[LogEvent]

val logbase = buildLogBase(userId)

when(logEvent.logBase).thenReturn(Some(logbase))

when(logEvent.eventNamespace).thenReturn(Some(buildNamespaceForOnboarding))

val eventDetails = mock[EventDetails]

val item = buildItemForOnboarding(itemId, topicId)

when(eventDetails.items)

.thenReturn(Some(Seq(item)))

when(logEvent.eventDetails).thenReturn(Some(eventDetails))

logEvent

}

def buildClientEventForOnboardingBackend(

topicId: Long,

userId: Long = 1L

): LogEvent = {

val logEvent = mock[LogEvent]

val logbase = buildLogBase(userId)

when(logEvent.logBase).thenReturn(Some(logbase))

when(logEvent.eventNamespace).thenReturn(Some(buildNamespaceForOnboardingBackend))

val eventDetails = buildEventDetailsForOnboardingBackend(topicId)

when(logEvent.eventDetails).thenReturn(Some(eventDetails))

logEvent

}

def defaultNamespace: EventNamespace = {

EventNamespace(Some("iphone"), None, None, None, None, Some("favorite"))

}

def buildNamespaceForOnboardingBackend: EventNamespace = {

EventNamespace(

Some("iphone"),

Some("onboarding\_backend"),

Some("subtasks"),

Some("topics\_selector"),

Some("removed"),

Some("selected"))

}

def buildNamespaceForOnboarding: EventNamespace = {

EventNamespace(

Some("iphone"),

Some("onboarding"),

Some("topics\_selector"),

None,

Some("topic"),

Some("follow")

)

}

def buildItemForHomeTimeline(

itemId: Long,

itemType: ItemType,

topicId: Long,

topicIds: Set[Long],

fn: (Long, Set[Long]) => ControllerData.V2

): Item = {

val item = Item(

id = Some(itemId),

itemType = Some(itemType),

suggestionDetails =

Some(SuggestionDetails(decodedControllerData = Some(fn(topicId, topicIds))))

)

item

}

def buildItemForGuide(

itemId: Long,

itemType: ItemType,

topicId: Long,

fn: Long => TopicMetadata

): Item = {

val item = mock[Item]

when(item.id).thenReturn(Some(itemId))

when(item.itemType).thenReturn(Some(itemType))

when(item.suggestionDetails)

.thenReturn(Some(SuggestionDetails(suggestionType = Some("ErgTweet"))))

val guideItemDetails = mock[GuideItemDetails]

when(guideItemDetails.transparentGuideDetails).thenReturn(Some(fn(topicId)))

when(item.guideItemDetails).thenReturn(Some(guideItemDetails))

item

}

def buildItemForOnboarding(

itemId: Long,

topicId: Long

): Item = {

val item = Item(

id = Some(itemId),

itemType = None,

description = Some(s"id=$topicId,row=1")

)

item

}

def buildEventDetailsForOnboardingBackend(

topicId: Long

): EventDetails = {

val eventDetails = mock[EventDetails]

val item = Item(

id = Some(topicId)

)

val itemTmp = buildItemForOnboarding(10, topicId)

when(eventDetails.items).thenReturn(Some(Seq(itemTmp)))

when(eventDetails.targets).thenReturn(Some(Seq(item)))

eventDetails

}

def topicMetadataInGuide(topicId: Long): TopicMetadata =

TopicMetadata(

SemanticCoreInterest(

SemanticCoreInterestV1(domainId = "131", entityId = topicId.toString)

)

)

def simClusterMetadataInGuide(simclusterId: Long = 1L): TopicMetadata =

TopicMetadata(

SimClusterInterest(

SimClusterInterestV1(simclusterId.toString)

)

)

def timelineTopicControllerData(topicId: Long): ControllerData.V2 =

ControllerData.V2(

ControllerDataV2.TimelinesTopic(

TimelinesTopicControllerData.V1(

TimelinesTopicControllerDataV1(

topicId = topicId,

topicTypesBitmap = 1

)

)))

def homeTweetControllerData(topicId: Long): ControllerData.V2 =

ControllerData.V2(

ControllerDataV2.HomeTweets(

HomeTweetsControllerData.V1(

HomeTweetsControllerDataV1(

topicId = Some(topicId)

))))

def homeTopicFollowPromptControllerData(topicId: Long): ControllerData.V2 =

ControllerData.V2(

ControllerDataV2.HomeTopicFollowPrompt(HomeTopicFollowPromptControllerData.V1(

HomeTopicFollowPromptControllerDataV1(Some(topicId)))))

def homeTopicAnnotationPromptControllerData(topicId: Long): ControllerData.V2 =

ControllerData.V2(

ControllerDataV2.HomeTopicAnnotationPrompt(HomeTopicAnnotationPromptControllerData.V1(

HomeTopicAnnotationPromptControllerDataV1(tweetId = 1L, topicId = topicId))))

def homeHitlTopicAnnotationPromptControllerData(topicId: Long): ControllerData.V2 =

ControllerData.V2(

ControllerDataV2.HomeHitlTopicAnnotationPrompt(

HomeHitlTopicAnnotationPromptControllerData.V1(

HomeHitlTopicAnnotationPromptControllerDataV1(tweetId = 2L, topicId = topicId))))

def searchTopicFollowPromptControllerData(topicId: Long): ControllerData.V2 =

ControllerData.V2(

ControllerDataV2.SearchResponse(

SearchResponseControllerData.V1(

SearchResponseControllerDataV1(

Some(ItemTypesControllerData.TopicFollowControllerData(

SearchTopicFollowPromptControllerData(Some(topicId))

)),

None

))))

def searchTweetTypesControllerData(topicId: Long): ControllerData.V2 =

ControllerData.V2(

ControllerDataV2.SearchResponse(

SearchResponseControllerData.V1(

SearchResponseControllerDataV1(

Some(ItemTypesControllerData.TweetTypesControllerData(

TweetTypesControllerData(None, Some(topicId))

)),

None

)

)))

//used for creating logged out user client events

def buildLogBaseWithoutUserId(guestId: Long): LogBase =

LogBase(

ipAddress = "120.10.10.20",

guestId = Some(guestId),

userAgent = None,

transactionId = "",

country = Some("US"),

timestamp = 100L,

language = Some("en")

)

}

test("getTopicId should correctly find topic id from item for home timeline and search") {

new Fixture {

val testData = Table(

("ItemType", "topicId", "controllerData"),

(ItemType.Tweet, 1L, timelineTopicControllerData(1L)),

(ItemType.User, 2L, timelineTopicControllerData(2L)),

(ItemType.Topic, 3L, homeTweetControllerData(3L)),

(ItemType.Topic, 4L, homeTopicFollowPromptControllerData(4L)),

(ItemType.Topic, 5L, searchTopicFollowPromptControllerData(5L)),

(ItemType.Topic, 6L, homeHitlTopicAnnotationPromptControllerData(6L))

)

forEvery(testData) {

(itemType: ItemType, topicId: Long, controllerDataV2: ControllerData.V2) =>

getTopicId(

buildItemForTimeline(1, itemType, topicId, \_ => controllerDataV2),

defaultNamespace) shouldEqual Some(topicId)

}

}

}

test("getTopicId should correctly find topic id from item for guide events") {

new Fixture {

getTopicId(

buildItemForGuide(1, ItemType.Tweet, 100, topicMetadataInGuide),

defaultNamespace

) shouldEqual Some(100)

}

}

test("getTopicId should correctly find topic id for onboarding events") {

new Fixture {

getTopicId(

buildItemForOnboarding(1, 100),

buildNamespaceForOnboarding

) shouldEqual Some(100)

}

}

test("should return TopicId From HomeSearch") {

val testData = Table(

("controllerData", "topicId"),

(

ControllerData.V2(

ControllerDataV2.HomeTweets(

HomeTweetsControllerData.V1(HomeTweetsControllerDataV1(topicId = Some(1L))))

),

Some(1L)),

(

ControllerData.V2(

ControllerDataV2.HomeTopicFollowPrompt(HomeTopicFollowPromptControllerData

.V1(HomeTopicFollowPromptControllerDataV1(topicId = Some(2L))))),

Some(2L)),

(

ControllerData.V2(

ControllerDataV2.TimelinesTopic(

TimelinesTopicControllerData.V1(

TimelinesTopicControllerDataV1(topicId = 3L, topicTypesBitmap = 100)

))),

Some(3L)),

(

ControllerData.V2(

ControllerDataV2.SearchResponse(

SearchResponseControllerData.V1(SearchResponseControllerDataV1(itemTypesControllerData =

Some(ItemTypesControllerData.TopicFollowControllerData(

SearchTopicFollowPromptControllerData(topicId = Some(4L)))))))),

Some(4L)),

(

ControllerData.V2(

ControllerDataV2.SearchResponse(

SearchResponseControllerData.V1(

SearchResponseControllerDataV1(itemTypesControllerData = Some(ItemTypesControllerData

.TweetTypesControllerData(TweetTypesControllerData(topicId = Some(5L)))))))),

Some(5L)),

(

ControllerData.V2(

ControllerDataV2

.SearchResponse(SearchResponseControllerData.V1(SearchResponseControllerDataV1()))),

None)

)

forEvery(testData) { (controllerDataV2: ControllerData.V2, topicId: Option[Long]) =>

getTopicIdFromHomeSearch(

Item(suggestionDetails = Some(

SuggestionDetails(decodedControllerData = Some(controllerDataV2))))) shouldEqual topicId

}

}

test("test TopicId From Onboarding") {

val testData = Table(

("Item", "EventNamespace", "topicId"),

(

Item(description = Some("id=11,key=value")),

EventNamespace(

page = Some("onboarding"),

section = Some("section has topic"),

component = Some("component has topic"),

element = Some("element has topic")

),

Some(11L)),

(

Item(description = Some("id=22,key=value")),

EventNamespace(

page = Some("onboarding"),

section = Some("section has topic")

),

Some(22L)),

(

Item(description = Some("id=33,key=value")),

EventNamespace(

page = Some("onboarding"),

component = Some("component has topic")

),

Some(33L)),

(

Item(description = Some("id=44,key=value")),

EventNamespace(

page = Some("onboarding"),

element = Some("element has topic")

),

Some(44L)),

(

Item(description = Some("id=678,key=value")),

EventNamespace(

page = Some("onXYZboarding"),

section = Some("section has topic"),

component = Some("component has topic"),

element = Some("element has topic")

),

None),

(

Item(description = Some("id=678,key=value")),

EventNamespace(

page = Some("page has onboarding"),

section = Some("section has topPic"),

component = Some("component has topPic"),

element = Some("element has topPic")

),

None),

(

Item(description = Some("key=value,id=678")),

EventNamespace(

page = Some("page has onboarding"),

section = Some("section has topic"),

component = Some("component has topic"),

element = Some("element has topic")

),

None)

)

forEvery(testData) { (item: Item, eventNamespace: EventNamespace, topicId: Option[Long]) =>

getTopicFromOnboarding(item, eventNamespace) shouldEqual topicId

}

}

test("test from Guide") {

val testData = Table(

("guideItemDetails", "topicId"),

(

GuideItemDetails(transparentGuideDetails = Some(

TransparentGuideDetails.TopicMetadata(

TopicModuleMetadata.TttInterest(tttInterest = TttInterest.unsafeEmpty)))),

None),

(

GuideItemDetails(transparentGuideDetails = Some(

TransparentGuideDetails.TopicMetadata(

TopicModuleMetadata.SimClusterInterest(simClusterInterest =

com.twitter.guide.scribing.thriftscala.SimClusterInterest.unsafeEmpty)))),

None),

(

GuideItemDetails(transparentGuideDetails = Some(

TransparentGuideDetails.TopicMetadata(TopicModuleMetadata.UnknownUnionField(field =

TFieldBlob(new TField(), Array.empty[Byte]))))),

None),

(

GuideItemDetails(transparentGuideDetails = Some(

TransparentGuideDetails.TopicMetadata(

TopicModuleMetadata.SemanticCoreInterest(

com.twitter.guide.scribing.thriftscala.SemanticCoreInterest.unsafeEmpty

.copy(domainId = "131", entityId = "1"))))),

Some(1L)),

)

forEvery(testData) { (guideItemDetails: GuideItemDetails, topicId: Option[Long]) =>

getTopicFromGuide(Item(guideItemDetails = Some(guideItemDetails))) shouldEqual topicId

}

}

test("getTopicId should return topicIds") {

getTopicId(

item = Item(suggestionDetails = Some(

SuggestionDetails(decodedControllerData = Some(

ControllerData.V2(

ControllerDataV2.HomeTweets(

HomeTweetsControllerData.V1(HomeTweetsControllerDataV1(topicId = Some(1L))))

))))),

namespace = EventNamespace(

page = Some("onboarding"),

section = Some("section has topic"),

component = Some("component has topic"),

element = Some("element has topic")

)

) shouldEqual Some(1L)

}

}