AdminAuthorizeAttribute : AuthorizeAttribute

Store user:

FormsAuthentication.Initialize();

FormsAuthentication.SetAuthCookie(userId.ToString(), false);

Read User cookie:

userFromCookie = HttpContext.Current.User.Identity.Name;

Recorder Authorization:

TokenAuthorizeAttribute:

actionContext.Request.GetDependencyScope().Resolve<IUserSession>().SetCurrentUser(user);

Fronted:

RecordingUploadedController

UnzipRecording Queue [

RecordingUploadedMessage (zipFile name , s3 folder)

]

BackEnd:

RecordingUploadedMessageHandler

-Download recording from bucket of temp recordings to EC2 local folder (needed role\token)

-unzip recording

-upload unzipped recording folder to bucket of constant recordings (need role\token)

-update recordings collection s3Path=[recording path],

Status = “unzipped”

Copy\Move

Fronted:

RecordingController

TransferRecordingFiles Queue [

BeginCopyRecordingMessage(

recordingId, destination course id, user id

]

BeginCopyRecordingMessageHandler:

(Recording)

entity.Status.ProcessingTasks.CopyTasks.Add(sourceTask);

foreach(var destination in message.Destination)

-can be copied to one or more courses

CopyRecordingMessage

-Copy recording files from s3 [source course] to s3 [target course]

-modify recording xml files

EndCopyRecordingMessage

Transcoding if required

Removing copy tasks

Push notification

-

{

.when('/courses', {

templateUrl: '/content/templates/courses/coursesList.html',

controller: 'coursesListCtrl',

title: 'Tegrity - Courses',

resolve: {

security: ["routeChangeSvc", function (routeChangeSvc) {

return routeChangeSvc.checkRegularUserOrGuestPermissions(Constants.Protocols.HTTPS);

}]

}

})

courseSvc.get

coursesController.get

Frontend:

PlaybackController\CreatePlayback

Fronted Queue [

AddWatchlogMessage (insert viewing to mongo db)

]

LogReportingEvents Queue [

AddWatchlogForReportsMessage (userId , CourseId,recordingId , viewingType (regular,webcast))

]

Backend:

AddWatchlogForReportsMessageHandler:

ReporingManager.AddViewing

using (var connection = new SqlConnection(\_reportSettings.ConnectionString))

{

using (var command = new SqlCommand("usp\_AddViewing", connection) { CommandType = CommandType.StoredProcedure })

{

command.Parameters.AddWithValue("@ClientID", clientId);

command.Parameters.AddWithValue("@ViewingID", viewingId);

command.Parameters.AddWithValue("@DateView", viewedDate);

command.Parameters.AddWithValue("@UserID", userId);

command.Parameters.AddWithValue("@InstructorID", instructorId);

command.Parameters.AddWithValue("@SessionID", sessionId);

command.Parameters.AddWithValue("@CourseID", courseId);

command.Parameters.AddWithValue("@PlaybackType", (int)playbackHostType);

command.Parameters.AddWithValue("@Type", (int)viewingType);

connection.Open();

command.ExecuteNonQuery();

Logger.Info(

"AddViewing finished execution. UniversityId:{0}, UserId:{1}, InstructorId:{2}, RecordingId:{3}, CourseId:{4}",

clientId, userId, instructorId, sessionId, courseId);

}

CaptionREquestController:

Fronted:

ApproveCaptionRequestsMessage (executes in fronted withoutquee)

ApproveCaptionRequestsMessageHandler

EditRecordingTask Queue [

RequestCaptionsFromProviderMessage (captionRequestIDs)

]

RequestCaptionsFromProviderMessageHandler

Email is being sent with link to mp3 recording:

@"http://{0}/api/mediadownload?recordingid={1}&userid={2}&format=mp3";

Webcast controller\Create:

Frontend Queue [

CreateWebCastMessage (WebcastProperties,mediaStreams(instructor,screen) )

]

CreateWebCastMessageHandler:

- create new folder in the bucket for the recording

-sets media steams url (ends with .m3u8)

foreach(var mediaStream in recording.Webcast.MediaStreams)

{

mediaStream.Url = new BlobStorageIdentifier(\_awsSettings.RegionName, \_awsSettings.RecordingsS3BucketName, recordingIdentifier.BlobName + "Class/Projector/" + mediaStream.StreamId + ".m3u8").SecureAbsoluteUrl;

CopySkeleton from from \_componentsManager.GetSkeletonFolder()

WebcastState.ready

Webcast controller\Join

Webcast controller\JoinLatestOfCourse

NUnit Test:

[TestFixture]

public class EndCopyRecordingsMessageHandlerTests{

IEndCopyRecordingMessage \_message

IEndCopyREcordingMessageHandler \_handler

}

[Test]

public void HandleEndCopyRecordingEntityNotFound()

{

\_message.RecordingId = Guid.Empty;

Assert.Throws<EntityNotFoundException>(() => \_handler.Handle(\_message, \_taskContext));

}

dependency injection

IOContainerConfiguration.cs

.Register(Component.For<IFileSystemProxy>().ImplementedBy<FileSystemProxy>())

CourseCtrl.js

$scope.startRecording

launchRecorder

recorderSvc.getRecorderLink

addContentFile.js

courseSvc.getFileCredentials

proxy.uploadFile (done from the client side)

courseSvc.addAdditionalContentFile

AddContentFileMessage[Title,Path,AddedBy,Extension]

AddContentFileMessageHandler -

inserts uploaded file message handler to db