using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using MongoDB.Driver;

namespace ShareHubServer.Controllers.Api {

[Route("api/CommunityGateway")]

public class CommunityGatewayController : Controller {

#region singletons

private class Resting {

public TaskCompletionSource<GetMessageResult.Message> Task;

public string Ip;

public string User;

}

private static Dictionary<string, List<Resting>> RESTMessageDictionary = new Dictionary<string, List<Resting>>();

#endregion

#region new community

[HttpPost("New")]

public ActionResult NewResponder(string userKey, string communityId, string communityName) {

if (!Utilities.Sanitized(userKey, communityId, communityName)) {

return StatusCode(405, "one or more arguments were null");

}

return Json(New(userKey, communityId, communityName));

}

internal ResultBase New(string userKey, string communityId, string communityName) {

IMongoCollection<DBEntry.Community> communitiesCollection = Program.Database.GetCollection<DBEntry.Community>("communities");

IMongoCollection<DBEntry.User> usersCollection = Program.Database.GetCollection<DBEntry.User>("users");

List<DBEntry.User> users = usersCollection.Find(x => x.Cookie == userKey).ToList();

if (users.Count > 1) {

return (new Result() { type = "unauthorized", success = false, authorized = false, message = "invalid key" });

}

DBEntry.User user = users[0];

if (user.CookieAvailability < DateTime.Now) {

return (new Result() { type = "unauthorized", success = false, authorized = false, message = "key to old" });

}

DBEntry.Community entry = new DBEntry.Community() {

Users = new string[] { user.Username },

Owner = user.Username,

Name = communityName,

UniqueName = communityId,

\_id = MongoDB.Bson.ObjectId.GenerateNewId(),

};

if (communitiesCollection.Find(x => x.UniqueName == entry.UniqueName).ToList().Count > 0) {

return (new Result() { type = "unsuccessful", success = false, authorized = true, message = "community id already exists" });

}

communitiesCollection.InsertOne(entry);

return (new Result() { type = "success", success = true, authorized = true, message = "all good" });

}

#endregion

#region get my communities

internal class GetMyResult : ResultBase {

public class Community {

public string name;

public string uniqueName;

public string[] users;

public string owner;

public string[] admins;

public string description;

}

public Community[] communities = null;

}

[HttpPost("GetMy")]

public ActionResult GetMyResponder(string userKey) {

if (!Utilities.Sanitized(userKey)) {

return StatusCode(405, "one or more arguments were null");

}

return Json(GetMy(userKey));

}

internal GetMyResult GetMy(string userKey) {

IMongoCollection<DBEntry.Community> communitiesCollection = Program.Database.GetCollection<DBEntry.Community>("communities");

IMongoCollection<DBEntry.User> usersCollection = Program.Database.GetCollection<DBEntry.User>("users");

(UserGatewayController.CheckUserResult userResult, DBEntry.User user) = UserGatewayController.CheckUser(userKey);

if (!userResult.success) {

return new GetMyResult() {

type = userResult.type,

authorized = userResult.authorized,

communities = null,

message = userResult.message,

success = userResult.success

};

}

List<DBEntry.Community> dbCommunities = communitiesCollection.Find(x => x.Users.Contains(user.Username)).ToList();

List<GetMyResult.Community> communities = new List<GetMyResult.Community>();

foreach (DBEntry.Community community in dbCommunities) {

communities.Add(new GetMyResult.Community() {

name = community.Name,

admins = community.Admins,

description = community.Description,

owner = community.Owner,

uniqueName = community.UniqueName,

users = community.Users

});

}

return (new GetMyResult() { type = "successful", authorized = true, success = true, message = "all good", communities = communities.ToArray() });

}

#endregion

#region get community

internal class GetCommunityResult : ResultBase {

public class Community {

public string name;

public string uniqueName;

public string[] users;

public string owner;

public string[] admins;

public string description;

public string[] boxKey;

public string[] boxLocation;

public string[] boxName;

}

public Community community;

}

[HttpPost("GetCommunity")]

public ActionResult GetCommunityResponder(string userKey, string uniqueName) {

if (!Utilities.Sanitized(userKey, uniqueName)) {

return StatusCode(405, "one or more arguments were null");

}

return Json(GetCommunity(userKey, uniqueName));

}

internal GetCommunityResult GetCommunity(string userKey, string uniqueName) {

IMongoCollection<DBEntry.Community> communitiesCollection = Program.Database.GetCollection<DBEntry.Community>("communities");

IMongoCollection<DBEntry.User> usersCollection = Program.Database.GetCollection<DBEntry.User>("users");

(UserGatewayController.CheckUserResult userResult, DBEntry.User user) = UserGatewayController.CheckUser(userKey);

if (!userResult.success) {

return new GetCommunityResult() {

authorized = userResult.authorized,

community = null,

message = userResult.message,

success = userResult.success,

type = userResult.type

};

}

List<DBEntry.Community> communities = communitiesCollection.Find(x => x.UniqueName == uniqueName).ToList();

if (communities.Count != 1) {

return (new GetCommunityResult() { type = "unsuccessful", authorized = true, success = false, message = "community doesnt exist", community = null });

}

DBEntry.Community dbCommunity = communities[0];

if (!dbCommunity.Users.Contains(user.Username)) {

return (new GetCommunityResult() {

type = "unsuccessful",

authorized = true,

success = false,

message = "you are not a part of that community",

community = null

});

}

return (new GetCommunityResult() {

type = "successful",

authorized = true,

success = true,

message = "all good",

community = new GetCommunityResult.Community() {

admins = dbCommunity.Admins,

description = dbCommunity.Description,

name = dbCommunity.Name,

owner = dbCommunity.Owner,

uniqueName = dbCommunity.UniqueName,

users = dbCommunity.Users,

boxKey = dbCommunity.Boxes.Select(x => x.Key).ToArray(),

boxLocation = dbCommunity.Boxes.Select(x => x.Location).ToArray(),

boxName = dbCommunity.Boxes.Select(x => x.Name).ToArray(),

}

});

}

#endregion

#region send message

[HttpPost("SendMessage")]

public ActionResult SendMessageResponder(string userKey, string message, string communityId) {

if (!Utilities.Sanitized(userKey, message, communityId)) {

return StatusCode(405, "one or more arguments were null");

}

return Json(SendMessage(userKey, message, communityId));

}

internal ResultBase SendMessage(string userKey, string message, string communityId) {

IMongoCollection<DBEntry.Community> communitiesCollection = Program.Database.GetCollection<DBEntry.Community>("communities");

IMongoCollection<DBEntry.Message> messageCollection = Program.Database.GetCollection<DBEntry.Message>("messages");

(UserGatewayController.CheckUserResult userResult, DBEntry.User user) = UserGatewayController.CheckUser(userKey);

if (!userResult.success) {

return (userResult);

}

List<DBEntry.Community> communities = communitiesCollection.Find(x => x.UniqueName == communityId).ToList();

if (communities.Count != 1) {

return (new Api.Result() { type = "no community", authorized = true, success = false, message = "community does not exists" });

}

DBEntry.Community community = communities[0];

if (!community.Users.Contains(user.Username)) {

return (new Api.Result() { type = "unauthorized for community", authorized = true, success = false, message = "you are not part of that community" });

}

MongoDB.Bson.ObjectId objectId = MongoDB.Bson.ObjectId.GenerateNewId();

messageCollection.InsertOneAsync(new DBEntry.Message() { Author = user.Username, CommunityId = communityId, Content = message, Ip = Request.HttpContext.Connection.RemoteIpAddress.ToString(), TimeOfSending = DateTime.Now, \_id = objectId });

if (RESTMessageDictionary.ContainsKey(community.UniqueName)) {

GetMessageResult.Message RESTMessage = new GetMessageResult.Message() {

author = user.Username,

content = message,

timeOfSending = ((DateTimeOffset)DateTime.Now).ToUnixTimeSeconds(),

objectId = objectId.ToString()

};

while (RESTMessageDictionary[community.UniqueName].Count != 0) {

if (!RESTMessageDictionary[community.UniqueName][0].Task.Task.IsCompleted) {

RESTMessageDictionary[community.UniqueName][0].Task.SetResult(RESTMessage);

}

try {

RESTMessageDictionary[community.UniqueName].RemoveAt(0);

}

catch (Exception) { };

}

}

return (new Api.Result() { type = "successful", authorized = true, success = true, message = "all good" });

}

#endregion

#region get message

internal class GetMessageResult : ResultBase {

public class Message {

public string author;

public long timeOfSending;

public string content;

public string objectId;

}

public Message[] messages;

}

[HttpPost("GetMessage")]

public ActionResult GetMessageResponder(string userKey, string communityId, int skip, int limit) {

if (!Utilities.Sanitized(userKey, communityId, skip, limit)) {

return StatusCode(405, "one or more arguments were null");

}

return Json(GetMessage(userKey, communityId, skip, limit));

}

internal GetMessageResult GetMessage(string userKey, string communityId, int skip, int limit) {

IMongoCollection<DBEntry.Community> communitiesCollection = Program.Database.GetCollection<DBEntry.Community>("communities");

IMongoCollection<DBEntry.Message> messageCollection = Program.Database.GetCollection<DBEntry.Message>("messages");

(UserGatewayController.CheckUserResult userResult, DBEntry.User user) = UserGatewayController.CheckUser(userKey);

if (!userResult.success) {

return new GetMessageResult() {

authorized = userResult.authorized,

message = userResult.message,

messages = null,

success = userResult.success,

type = userResult.type,

};

}

List<DBEntry.Community> communities = communitiesCollection.Find(x => x.UniqueName == communityId).ToList();

if (communities.Count != 1) {

return (new GetMessageResult() { type = "no community", authorized = true, success = false, message = "community does not exists", messages = new GetMessageResult.Message[0] });

}

DBEntry.Community community = communities[0];

if (!community.Users.Contains(user.Username)) {

return (new GetMessageResult() { type = "unauthorized for community", authorized = true, success = false, message = "you are not part of that community", messages = new GetMessageResult.Message[0] });

}

List<DBEntry.Message> dbMessages = messageCollection.Find(x => x.CommunityId == communityId).SortByDescending(x => x.TimeOfSending).Skip(skip).Limit(limit).ToList();

List<GetMessageResult.Message> messages = new List<GetMessageResult.Message>();

foreach (DBEntry.Message message in dbMessages) {

messages.Add(new GetMessageResult.Message() {

author = message.Author,

content = message.Content,

timeOfSending = ((DateTimeOffset)message.TimeOfSending).ToUnixTimeSeconds(),

objectId = message.\_id.ToString()

});

}

return (new GetMessageResult() { type = "successful", authorized = true, success = true, message = "all good", messages = messages.ToArray() });

}

#endregion

#region REST for new message

internal class RESTMessageResult : ResultBase {

public GetMessageResult.Message restMessage;

}

[HttpPost("RESTMessage")]

public async Task<ActionResult> RESTMessageResponder(string userKey, string communityId) {

if (!Utilities.Sanitized(userKey, communityId)) {

return StatusCode(405, "one or more arguments were null");

}

return Json(await RESTMessage(userKey, communityId));

}

internal async Task<RESTMessageResult> RESTMessage(string userKey, string communityId) {

IMongoCollection<DBEntry.Community> communitiesCollection = Program.Database.GetCollection<DBEntry.Community>("communities");

IMongoCollection<DBEntry.Message> messageCollection = Program.Database.GetCollection<DBEntry.Message>("messages");

(UserGatewayController.CheckUserResult userResult, DBEntry.User user) = UserGatewayController.CheckUser(userKey);

if (!userResult.success) {

return new RESTMessageResult() {

authorized = userResult.authorized,

message = userResult.message,

restMessage = null,

success = userResult.success,

type = userResult.type

};

}

List<DBEntry.Community> communities = communitiesCollection.Find(x => x.UniqueName == communityId).ToList();

if (communities.Count != 1) {

return (new RESTMessageResult() { type = "no community", authorized = true, success = false, message = "community does not exists" });

}

DBEntry.Community community = communities[0];

if (!community.Users.Contains(user.Username)) {

return (new RESTMessageResult() { type = "unauthorized for community", authorized = true, success = false, message = "you are not part of that community" });

}

TaskCompletionSource<GetMessageResult.Message> tcs = new TaskCompletionSource<GetMessageResult.Message>();

if (RESTMessageDictionary.ContainsKey(communityId)) {

RESTMessageDictionary[communityId].Add(new Resting() {

Ip = Request.HttpContext.Connection.RemoteIpAddress.ToString(),

Task = tcs,

User = user.Username

});

}

else {

RESTMessageDictionary.Add(communityId, new List<Resting>(){new Resting() {

Ip = Request.HttpContext.Connection.RemoteIpAddress.ToString(),

Task = tcs

}});

}

GetMessageResult.Message awaitedMessage = await tcs.Task;

return (new RESTMessageResult() {

type = "successful",

authorized = true,

success = true,

message = "all good",

restMessage = awaitedMessage

});

}

#endregion

#region invite

//TODO: user doesnt exist error

//TODO: DBEntry.User.Invite!!!!!!!!!!!!!!!!! use it mads

[HttpPost("Invite")]

public ActionResult InviteResponder(string userKey, string communityId, string userToInvite) {

if (!Utilities.Sanitized(userKey, communityId, userToInvite)) {

return StatusCode(405, "one or more arguments were null");

}

return Json(Invite(userKey, communityId, userToInvite));

}

internal Result Invite(string userKey, string communityId, string userToInvite) {

IMongoCollection<DBEntry.Community> communitiesCollection = Program.Database.GetCollection<DBEntry.Community>("communities");

IMongoCollection<DBEntry.User> usersCollection = Program.Database.GetCollection<DBEntry.User>("users");

List<DBEntry.User> users = usersCollection.Find(x => x.Cookie == userKey).ToList();

if (users.Count > 1) {

return (new Api.Result() {

type = "unauthorized",

success = false,

authorized = false,

message = "invalid key"

});

}

DBEntry.User user = users[0];

if (user.CookieAvailability < DateTime.Now) {

return (new Api.Result() { type = "unauthorized", success = false, authorized = false, message = "key to old" });

}

List<DBEntry.Community> communities = communitiesCollection.Find(x => x.UniqueName == communityId).ToList();

if (communities.Count != 1) {

return (new Api.Result() { type = "no community", authorized = true, success = false, message = "community does not exists" });

}

DBEntry.Community community = communities[0];

if (!community.Users.Contains(user.Username)) {

return (new Api.Result() { type = "unauthorized for community", authorized = true, success = false, message = "you are not part of that community" });

}

if (community.Users.Contains(userToInvite)) {

return (new Api.Result() { type = "unauthorized for target user", authorized = true, success = false, message = "user is already in this community" });

}

List<string> userList = community.Users.ToList();

userList.Add(userToInvite);

community.Users = userList.ToArray();

communitiesCollection.ReplaceOneAsync(x => x.UniqueName == community.UniqueName, community);

return (new Api.Result() { type = "successful", authorized = true, success = true, message = "all good" });

}

#endregion

}

}