using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Linq.Expressions;

using System.Runtime.InteropServices;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using MongoDB.Driver;

namespace ShareHubServer.Controllers.Api {

[Route("api/UserGateway")]

public class UserGatewayController : Controller {

#region check

internal class CheckResult : ResultBase {

public string cookieKey = null;

}

[HttpPost("Check")]

public ActionResult CheckResponder(string username, string password) {

if (!Utilities.Sanitized(username, password)) {

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

}

return Json(Check(username, password));

}

internal CheckResult Check(string username, string password) {

password = HashPassword(password);

Expression<Func<DBEntry.User, bool>> filter = x => x.Username == username && x.Password == password;

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

List<DBEntry.User> users = usersCollection.Find(filter).ToList();

if (users.Count == 0) {

return (new CheckResult() {

authorized = false,

type = "unauthorized",

message = "username or password didnt match",

success = false

});

}

string key = GenerateKey();

users[0].Cookie = key;

users[0].CookieAvailability = DateTime.Now.AddDays(1);

usersCollection.DeleteOne(filter);

usersCollection.InsertOne(users[0]);

return new CheckResult() {

authorized = true,

type = "authorized",

message = "all good",

cookieKey = key,

success = true,

};

}

#endregion

#region check key

internal class CheckUserResult : ResultBase {

public string username = null;

}

[HttpPost("CheckKey")]

public ActionResult CheckKeyResponder(string key) {

if (!Utilities.Sanitized(key)) {

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

}

return Json(CheckUser(key).Result);

}

internal static (CheckUserResult Result, DBEntry.User User) CheckUser(string key) {

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

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

if (users.Count == 0) {

return (new CheckUserResult() {

type = "unauthorized",

authorized = false,

message = "key not found",

success = false

}, null);

}

DBEntry.User user = users[0];

if (users[0].CookieAvailability < DateTime.Now) {

return (new CheckUserResult() {

type = "unauthorized",

authorized = false,

message = "key to old",

success = false

}, user);

}

return (new CheckUserResult() {

type = "authorized",

authorized = true,

message = "authorized",

username = users[0].Username,

success = true

}, user);

}

#endregion

#region profile data

internal class ProfileDataResult : ResultBase {

public string username = null;

public string[] communitiesName = null;

public string[] communitiesUniqueName = null;

public bool? myself = null;

public string nfcId = null;

public string[] boxes = null;

public string[] boxesKey = null;

}

[HttpPost("ProfileData")]

public ActionResult ProfileDataResponder(string username, string key) {

if (!Utilities.Sanitized(username, key)) {

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

}

return Json(ProfileData(username, key));

}

internal ProfileDataResult ProfileData(string username, string key) {

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

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

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

if (users.Count == 0) {

return (new ProfileDataResult() {

type = "unauthorized",

authorized = false,

message = "key not found",

success = false,

});

}

if (users[0].CookieAvailability < DateTime.Now) {

return (new ProfileDataResult() {

type = "unauthorized",

authorized = false,

message = "key to old",

success = false,

});

}

var user = users[0];

bool myself = user.Username == username;

if (myself) {

List<DBEntry.Community> communities = communitiesCollection.Find(x => x.Users.Contains(username)).ToList();

List<string> communitiesName = new List<string>();

List<string> communitieUniquesName = new List<string>();

communities.ForEach(x => {

communitiesName.Add(x.Name);

communitieUniquesName.Add(x.UniqueName);

});

List<string> boxes = new List<string>();

List<string> boxesKey = new List<string>();

Array.ForEach(user.Boxes, x => {

boxes.Add(x.Name);

boxesKey.Add(x.Key);

});

return (new ProfileDataResult() {

type = "success",

authorized = true,

message = "all good",

success = true,

myself = true,

communitiesName = communitiesName.ToArray(),

communitiesUniqueName = communitieUniquesName.ToArray(),

username = username,

nfcId = user.NFCid,

boxes = boxes.ToArray(),

boxesKey = boxesKey.ToArray()

});

}

else {

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

List<string> communitiesName = new List<string>();

List<string> communitieUniquesName = new List<string>();

communities.ForEach(x => {

communitiesName.Add(x.Name);

communitieUniquesName.Add(x.UniqueName);

});

return (new ProfileDataResult() {

type = "success",

authorized = true,

message = "all good",

success = true,

myself = false,

communitiesName = communitiesName.ToArray(),

communitiesUniqueName = communitieUniquesName.ToArray(),

username = user.Username,

});

}

}

#endregion

#region new

[HttpPost("New")]

public ActionResult NewResponder(string username, string password) {

if (!Utilities.Sanitized(username, password)) {

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

}

return Json(New(username, password));

}

internal CheckResult New(string username, string password) {

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

List<DBEntry.User> users = usersCollection.Find(x => x.Username == username).ToList();

if (users.Count > 0) {

return (new CheckResult() {

authorized = false,

type = "unsuccessful",

message = "username already taken",

success = true

});

}

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

Username = username,

Password = HashPassword(password),

Cookie = GenerateKey(),

CookieAvailability = DateTime.Now.AddDays(1),

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

};

usersCollection.InsertOne(entry);

return (new CheckResult() {

authorized = true,

type = "successful",

message = "success",

cookieKey = entry.Cookie,

success = true

});

}

#endregion

#region register nfc

[HttpPost("RegisterNfc")]

public ActionResult RegisterNfcResponder(string key, string nfcId) {

if (!Utilities.Sanitized(key, nfcId)) {

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

}

return Json(RegisterNfc(key, nfcId));

}

internal ResultBase RegisterNfc(string key, string nfcId) {

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

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

if (!userResult.authorized) {

return (userResult);

}

user.NFCid = nfcId;

usersCollection.ReplaceOne(x => x.Username == user.Username, user);

return (new Api.Result() {

success = true,

authorized = true,

message = "all good",

type = "successful"

});

}

#endregion

#region delete nfc

[HttpPost("DeleteNfc")]

public ActionResult DeleteNfcResponder(string key) {

if (!Utilities.Sanitized(key)) {

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

}

return Json(DeleteNfc(key));

}

internal ResultBase DeleteNfc(string key) {

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

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

if (!userResult.authorized) {

return (userResult);

}

user.NFCid = null;

usersCollection.ReplaceOne(x => x.Username == user.Username, user);

return (new Result() {

success = true,

authorized = true,

message = "all good",

type = "successful"

});

}

#endregion

private string GenerateKey() {

byte[] bytes = new byte[64];

Program.Random.NextBytes(bytes);

string key = Convert.ToBase64String(bytes);

return key;

}

public string HashPassword(string password) {

byte[] data = System.Text.Encoding.ASCII.GetBytes(password);

data = new System.Security.Cryptography.SHA256Managed().ComputeHash(data);

return System.Text.Encoding.ASCII.GetString(data);

}

}

}