# MailHelper

using System.Collections.Generic;

using System.Net;

using System.Net.Mail;

using System.Threading.Tasks;

using System.Web.Configuration;

namespace Backend.Classes

{

public class MailHelper

{

public static async Task SendMail(string to, string subject, string body)

{

var message = new MailMessage();

message.To.Add(new MailAddress(to));

message.From = new MailAddress(WebConfigurationManager.AppSettings["AdminUser"]);

message.Subject = subject;

message.Body = body;

message.IsBodyHtml = true;

using (var smtp = new SmtpClient())

{

var credential = new NetworkCredential

{

UserName = WebConfigurationManager.AppSettings["AdminUser"],

Password = WebConfigurationManager.AppSettings["AdminPassWord"]

};

smtp.Credentials = credential;

smtp.Host = WebConfigurationManager.AppSettings["SMTPName"];

smtp.Port = int.Parse(WebConfigurationManager.AppSettings["SMTPPort"]);

smtp.EnableSsl = true;

await smtp.SendMailAsync(message);

}

}

public static async Task SendMail(List<string> mails, string subject, string body)

{

var message = new MailMessage();

foreach (var to in mails)

{

message.To.Add(new MailAddress(to));

}

message.From = new MailAddress(WebConfigurationManager.AppSettings["AdminUser"]);

message.Subject = subject;

message.Body = body;

message.IsBodyHtml = true;

using (var smtp = new SmtpClient())

{

var credential = new NetworkCredential

{

UserName = WebConfigurationManager.AppSettings["AdminUser"],

Password = WebConfigurationManager.AppSettings["AdminPassWord"]

};

smtp.Credentials = credential;

smtp.Host = WebConfigurationManager.AppSettings["SMTPName"];

smtp.Port = int.Parse(WebConfigurationManager.AppSettings["SMTPPort"]);

smtp.EnableSsl = true;

await smtp.SendMailAsync(message);

}

}

}

}

# UsersHelper

using Backend.Models;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.EntityFramework;

using System;

using System.Threading.Tasks;

using System.Web.Configuration;

namespace Backend.Classes

{

public class UsersHelper : IDisposable

{

private static ApplicationDbContext userContext = new ApplicationDbContext();

private static DataContextLocal db = new DataContextLocal();

public static bool DeleteUser(string userName, string roleName)

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var userASP = userManager.FindByEmail(userName);

if (userASP == null)

{

return false;

}

var response = userManager.RemoveFromRole(userASP.Id, roleName);

return response.Succeeded;

}

public static bool UpdateUserName(string currentUserName, string newUserName)

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var userASP = userManager.FindByEmail(currentUserName);

if (userASP == null)

{

return false;

}

userASP.UserName = newUserName;

userASP.Email = newUserName;

var response = userManager.Update(userASP);

return response.Succeeded;

}

public static void CheckRole(string roleName)

{

var roleManager = new RoleManager<IdentityRole>(new RoleStore<IdentityRole>(userContext));

// Check to see if Role Exists, if not create it

if (!roleManager.RoleExists(roleName))

{

roleManager.Create(new IdentityRole(roleName));

}

}

public static void CheckSuperUser()

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var email = WebConfigurationManager.AppSettings["AdminUser"];

var password = WebConfigurationManager.AppSettings["AdminPassWord"];

var userASP = userManager.FindByName(email);

if (userASP == null)

{

CreateUserASP(email, "Admin", password);

return;

}

userManager.AddToRole(userASP.Id, "Admin");

}

public static void CreateUserASP(string email, string roleName)

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var userASP = userManager.FindByEmail(email);

if (userASP == null)

{

userASP = new ApplicationUser

{

Email = email,

UserName = email,

};

userManager.Create(userASP, email);

}

userManager.AddToRole(userASP.Id, roleName);

}

public static void CreateUserASP(string email, string roleName, string password)

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var userASP = new ApplicationUser

{

Email = email,

UserName = email,

};

var result = userManager.Create(userASP, password);

if (result.Succeeded)

{

userManager.AddToRole(userASP.Id, roleName);

}

}

public static async Task PasswordRecovery(string email)

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var userASP = userManager.FindByEmail(email);

if (userASP == null)

{

return;

}

var random = new Random();

var newPassword = string.Format("{0}", random.Next(100000, 999999));

var response = await userManager.AddPasswordAsync(userASP.Id, newPassword);

if (response.Succeeded)

{

var subject = "Soccer App - Recuperación de contraseña";

var body = string.Format(@"

<h1>Soccer App - Recuperación de contraseña</h1>

<p>Su nueva contraseña es: <strong>{0}</strong></p>

<p>Por favor no olvide cambiarla por una de fácil recordación",

newPassword);

await MailHelper.SendMail(email, subject, body);

}

}

public void Dispose()

{

userContext.Dispose();

db.Dispose();

}

}

}

# Web.config

<appSettings>

<add key="webpages:Version" value="3.0.0.0" />

<add key="webpages:Enabled" value="false" />

<add key="ClientValidationEnabled" value="true" />

<add key="UnobtrusiveJavaScriptEnabled" value="true" />

<add key="AdminUser" value="jzuluaga55@gmail.com" />

<add key="AdminPassWord" value="123456" />

<add key="SMTPName" value="smtp.gmail.com" />

<add key="SMTPPort" value="587" />

</appSettings>

# Global.asax

using Backend.Classes;

using System.Data.Entity;

using System.Web.Mvc;

using System.Web.Optimization;

using System.Web.Routing;

namespace Backend

{

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

Database.SetInitializer(

new MigrateDatabaseToLatestVersion<Models.DataContextLocal,

Migrations.Configuration>());

CheckRolesAndSuperUser();

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

private void CheckRolesAndSuperUser()

{

UsersHelper.CheckRole("Admin");

UsersHelper.CheckRole("User");

UsersHelper.CheckSuperUser();

}

}

}

# UsersController (Backend)

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Create(UserView view)

{

if (ModelState.IsValid)

{

var pic = string.Empty;

var folder = "~/Content/Users";

if (view.PictureFile != null)

{

pic = FilesHelper.UploadPhoto(view.PictureFile, folder);

pic = string.Format("{0}/{1}", folder, pic);

}

var user = ToUser(view);

user.Picture = pic;

db.Users.Add(user);

await db.SaveChangesAsync();

UsersHelper.CreateUserASP(view.Email, "User", view.Password);

return RedirectToAction("Index");

}

ViewBag.FavoriteLeagueId = new SelectList(db.Leagues.OrderBy(l => l.Name), "LeagueId", "Name", view.FavoriteLeagueId);

ViewBag.FavoriteTeamId = new SelectList(db.Teams.Where(t => t.LeagueId == view.FavoriteLeagueId).OrderBy(t => t.Name), "TeamId", "Name", view.FavoriteTeamId);

ViewBag.UserTypeId = new SelectList(db.UserTypes.OrderBy(ut => ut.Name), "UserTypeId", "Name", view.UserTypeId);

return View(view);

}

Los autorice del backend van con:

[Authorize(Roles = "Admin")]

# UserResponse

using Domain;

using System.Collections.Generic;

namespace API.Classes

{

public class UserResponse

{

public int UserId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public int UserTypeId { get; set; }

public string Picture { get; set; }

public string Email { get; set; }

public string NickName { get; set; }

public int FavoriteTeamId { get; set; }

public int Points { get; set; }

public UserType UserType { get; set; }

public Team FavoriteTeam { get; set; }

public List<Group> Groups { get; set; }

public List<GroupUser> GroupUsers { get; set; }

public List<Prediction> Predictions { get; set; }

}

}

# UsersController (API)

[RoutePrefix("api/Users")]

[Authorize(Roles = "User")]

public class UsersController : ApiController

{

private DataContext db = new DataContext();

[HttpPost]

[Route("GetUserByEmail")]

public async Task<IHttpActionResult> GetUserByEmail(JObject form)

{

var email = string.Empty;

dynamic jsonObject = form;

try

{

email = jsonObject.Email.Value;

}

catch

{

return BadRequest("Incorrect call");

}

var user = await db.Users.Where(u => u.Email.ToLower() == email.ToLower()).FirstOrDefaultAsync();

if (user == null)

{

return NotFound();

}

var userResponse = ToUserResponse(user);

return Ok(userResponse);

}

# LeagueResponse

using Domain;

using System.Collections.Generic;

namespace API.Classes

{

public class LeagueResponse

{

public int LeagueId { get; set; }

public string Name { get; set; }

public string Logo { get; set; }

public List<Team> Teams { get; set; }

}

}

# LeaguesController

[Authorize(Roles = "User")]

public class LeaguesController : ApiController

{

private DataContext db = new DataContext();

public async Task<IHttpActionResult> GetLeagues()

{

var leagues = await db.Leagues.ToListAsync();

var leaugesResponse = new List<LeagueResponse>();

foreach (var league in leagues)

{

leaugesResponse.Add(new LeagueResponse

{

LeagueId = league.LeagueId,

Logo = league.Logo,

Name = league.Name,

Teams = league.Teams.ToList(),

});

}

return Ok(leaugesResponse);

}

# TournamentResponse

using Domain;

using System.Collections.Generic;

namespace API.Classes

{

public class TournamentResponse

{

public int TournamentId { get; set; }

public string Name { get; set; }

public string Logo { get; set; }

public List<TournamentGroup> Groups { get; set; }

public List<Date> Dates { get; set; }

}

}

# MatchResponse

using Domain;

using System;

namespace API.Classes

{

public class MatchResponse

{

public int MatchId { get; set; }

public int DateId { get; set; }

public DateTime DateTime { get; set; }

public int LocalId { get; set; }

public int VisitorId { get; set; }

public int? LocalGoals { get; set; }

public int? VisitorGoals { get; set; }

public int StatusId { get; set; }

public int TournamentGroupId { get; set; }

public Team Local { get; set; }

public Team Visitor { get; set; }

}

}

# TournamentsController

[RoutePrefix("api/Tournaments")]

[Authorize(Roles = "User")]

public class TournamentsController : ApiController

{

private DataContext db = new DataContext();

[HttpPost]

[Route("GetMatchesToPredict")]

public IHttpActionResult GetMatchesToPredict(JObject form)

{

var tournamentIdstring = string.Empty;

var userIdstring = string.Empty;

dynamic jsonObject = form;

try

{

userIdstring = jsonObject.UserId.Value;

tournamentIdstring = jsonObject.TournamentId.Value;

}

catch

{

return BadRequest("Incorrect call");

}

int tournamentId = 0;

int userId = 0;

int.TryParse(tournamentIdstring, out tournamentId);

int.TryParse(userIdstring, out userId);

if (userId == 0 || tournamentId == 0)

{

return BadRequest("Incorrect call");

}

var qry = (from t in db.Tournaments

join d in db.Dates on t.TournamentId equals d.TournamentId

join m in db.Matches on d.DateId equals m.DateId

where t.TournamentId == tournamentId && m.StatusId != 3 && m.DateTime > DateTime.Now

select new { m }).ToList();

var predictions = db.Predictions.Where(p => p.UserId == userId).ToList();

var matches = new List<MatchResponse>();

foreach (var item in qry)

{

var prediction = predictions.Where(p => p.MatchId == item.m.MatchId).FirstOrDefault();

if (prediction == null)

{

matches.Add(new MatchResponse

{

DateId = item.m.DateId,

DateTime = item.m.DateTime,

Local = item.m.Local,

LocalGoals = item.m.LocalGoals,

LocalId = item.m.LocalId,

MatchId = item.m.MatchId,

StatusId = item.m.StatusId,

TournamentGroupId = item.m.TournamentGroupId,

Visitor = item.m.Visitor,

VisitorGoals = item.m.VisitorGoals,

VisitorId = item.m.VisitorId,

});

}

}

return Ok(matches);

}

public async Task<IHttpActionResult> GetTournaments()

{

var tournaments = await db.Tournaments.Where(t => t.IsActive).OrderBy(t => t.Order).ToListAsync();

var tournamentsResponse = new List<TournamentResponse>();

foreach (var tournament in tournaments)

{

tournamentsResponse.Add(new TournamentResponse

{

Dates = tournament.Dates.ToList(),

Groups = tournament.Groups.ToList(),

Logo = tournament.Logo,

Name = tournament.Name,

TournamentId = tournament.TournamentId,

});

}

return Ok(tournamentsResponse);

}