NOTES

Contenido

[Diagrama entidad relación 1](#_Toc454284252)

[Modelos 2](#_Toc454284253)

[User 2](#_Toc454284254)

[Group 2](#_Toc454284255)

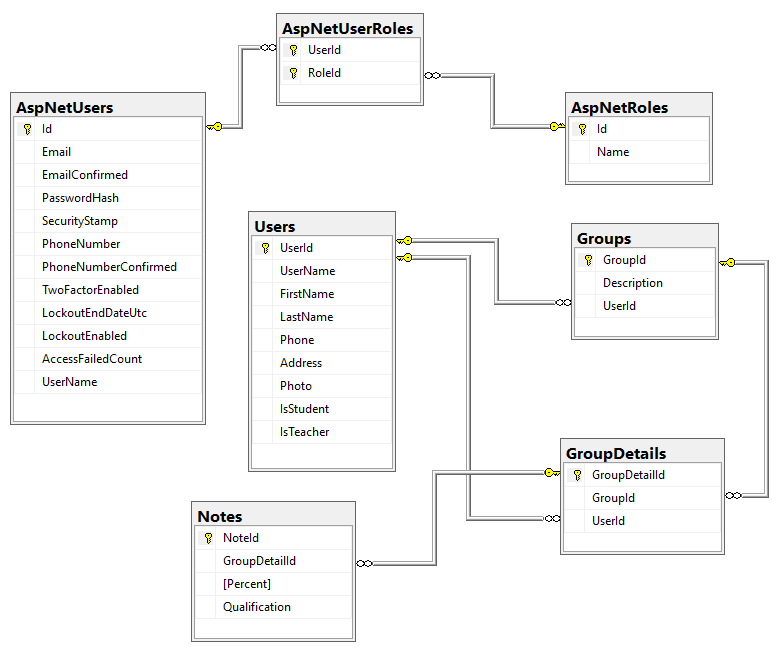
[Group Detail 3](#_Toc454284256)

[Note 3](#_Toc454284257)

[Utilidades 3](#_Toc454284258)

[API Login 7](#_Toc454284259)

# Diagrama entidad relación



# Modelos

## User

public class User

{

[Key]

public int UserId { get; set; }

[Display(Name = "E-Mail")]

[Required(ErrorMessage = "The field {0} is required")]

[StringLength(100, ErrorMessage = "The field {0} can contain maximun {1} and minimum {2} characters", MinimumLength = 7)]

[DataType(DataType.EmailAddress)]

[Index("UserNameIndex", IsUnique = true)]

public string UserName { get; set; }

[Display(Name = "First name")]

[Required(ErrorMessage = "The field {0} is required")]

[StringLength(50, ErrorMessage = "The field {0} can contain maximun {1} and minimum {2} characters", MinimumLength = 2)]

public string FirstName { get; set; }

[Display(Name = "Last name")]

[Required(ErrorMessage = "The field {0} is required")]

[StringLength(50, ErrorMessage = "The field {0} can contain maximun {1} and minimum {2} characters", MinimumLength = 2)]

public string LastName { get; set; }

[Display(Name = "User")]

public string FullName { get { return string.Format("{0} {1}", this.FirstName, this.LastName); } }

[Required(ErrorMessage = "The field {0} is required")]

[StringLength(20, ErrorMessage = "The field {0} can contain maximun {1} and minimum {2} characters", MinimumLength = 7)]

public string Phone { get; set; }

[Required(ErrorMessage = "The field {0} is required")]

[StringLength(100, ErrorMessage = "The field {0} can contain maximun {1} and minimum {2} characters", MinimumLength = 10)]

public string Address { get; set; }

[DataType(DataType.ImageUrl)]

public string Photo { get; set; }

[Display(Name = "Is Student")]

public bool IsStudent { get; set; }

[Display(Name = "Is Teacher")]

public bool IsTeacher { get; set; }

public virtual ICollection<Group> Groups { get; set; }

public virtual ICollection<GroupDetail> GroupDetails { get; set; }

}

## Group

public class Group

{

[Key]

public int GroupId { get; set; }

[Required(ErrorMessage = "The field {0} is required")]

[StringLength(50, ErrorMessage = "The field {0} can contain maximun {1} and minimum {2} characters", MinimumLength = 3)]

[Index("GroupDescriptionIndex", IsUnique = true)]

public string Description { get; set; }

public int UserId { get; set; }

public virtual User Teacher { get; set; }

public virtual ICollection<GroupDetail> GroupDetails { get; set; }

}

## Group Detail

public class GroupDetail

{

[Key]

public int GroupDetailId { get; set; }

public int GroupId { get; set; }

public int UserId { get; set; }

public virtual Group Group { get; set; }

public virtual User Student { get; set; }

public string GroupStudent { get { return string.Format("{0} / {1}", Group.Description, Student.FullName); } }

public virtual ICollection<Note> Notes { get; set; }

}

## Note

public class Note

{

[Key]

public int NoteId { get; set; }

public int GroupDetailId { get; set; }

[Required(ErrorMessage = "You must enter a {0}")]

[Range(0, 1, ErrorMessage = "The field {0} must be contain values between {1} and {2}")]

[DisplayFormat(DataFormatString = "{0:P2}", ApplyFormatInEditMode = false)]

public float Percent { get; set; }

[Required(ErrorMessage = "You must enter a {0}")]

[Range(0, 5, ErrorMessage = "The field {0} must be contain values between {1} and {2}")]

[DisplayFormat(DataFormatString = "{0:N2}", ApplyFormatInEditMode = false)]

public float Qualification { get; set; }

public virtual GroupDetail GroupDetail { get; set; }

}

# Utilidades

public class Utilities : IDisposable

{

private static ApplicationDbContext userContext = new ApplicationDbContext();

private static NotesContext db = new NotesContext();

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(string role)

{

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, role, password);

return;

}

}

public static void CreateUserASP(string email)

{

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

var userASP = new ApplicationUser

{

Email = email,

UserName = email,

};

userManager.Create(userASP, email);

}

public static void CreateUserASP(string email, string roleName)

{

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

var userASP = new ApplicationUser

{

Email = email,

UserName = email,

};

userManager.Create(userASP, email);

userManager.AddToRole(userASP.Id, roleName);

}

public static void AddRoleToUser(string email, string roleName)

{

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

var userASP = userManager.FindByEmail(email);

if (userASP == null)

{

return;

}

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,

};

userManager.Create(userASP, password);

userManager.AddToRole(userASP.Id, roleName);

}

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);

}

}

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 user = db.Users.Where(tp => tp.UserName == email).FirstOrDefault();

if (user == null)

{

return;

}

var random = new Random();

var newPassword = string.Format("{0}{1}{2:04}\*",

user.FirstName.Trim().ToUpper().Substring(0, 1),

user.LastName.Trim().ToLower(),

random.Next(10000));

userManager.RemovePassword(userASP.Id);

userManager.AddPassword(userASP.Id, newPassword);

var subject = "Notes Password Recovery";

var body = string.Format(@"

<h1>Taxes Password Recovery</h1>

<p>Yor new password is: <strong>{0}</strong></p>

<p>Please change it for one, that you remember easyly",

newPassword);

await SendMail(email, subject, body);

}

public static string UploadPhoto(HttpPostedFileBase file)

{

// Upload image

string path = string.Empty;

string pic = string.Empty;

if (file != null)

{

pic = Path.GetFileName(file.FileName);

path = Path.Combine(HttpContext.Current.Server.MapPath("~/Content/Photos"), pic);

file.SaveAs(path);

using (MemoryStream ms = new MemoryStream())

{

file.InputStream.CopyTo(ms);

byte[] array = ms.GetBuffer();

}

}

return pic;

}

public void Dispose()

{

userContext.Dispose();

db.Dispose();

}

}

# API Login

En la clase:

[RoutePrefix("API/Users")]

El método:

[HttpPost]

[Route("Login")]

public IHttpActionResult Longin(JObject form)

{

string email = string.Empty;

string password = string.Empty;

dynamic jsonObject = form;

try

{

email = jsonObject.Email.Value;

password = jsonObject.Password.Value;

}

catch

{

return this.BadRequest("Incorrect call");

}

var userContext = new ApplicationDbContext();

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

var userASP = userManager.Find(email, password);

if (userASP == null)

{

return this.BadRequest("User or password wrong");

}

var user = db.Users

.Where(u => u.UserName == email)

.FirstOrDefault();

if (user == null)

{

return this.BadRequest("User or password wrong");

}

return this.Ok(user);

}