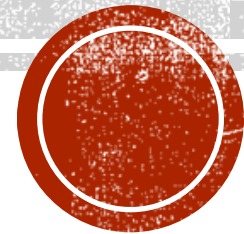


# TP N°3

Gestion des utilisateurs, authentification et autorisation  
avec ASP.NET Identity

**Enseignant :**

Malek Zribi



# AJOUTER L'AUTHENTIFICATION

- Dans le but d'intégrer un module d'authentification et de gestion des utilisateurs dans l'application du TP N°2. On va suivre les étapes suivantes :
- **Installation de l'API Identity** : Installer le Package **Microsoft.AspNetCore.Identity**
- Nous utiliserons Entity Framework Core avec Identity. Par conséquent, nous devons installer Identity EF Core Package.

Installer le Package **Microsoft.AspNetCore.Identity.EntityFrameworkCore**

- Modifier le code de la classe de contexte en ajoutant un héritage de la classe IdentityDbContext :

```
public class AppDbContext : IdentityDbContext
{
    // Reste du code
}
```

- Configurer les services Identity de ASP.NET Core dans le fichier « Program.cs » en ajoutant ce service :

```
builder.Services.AddIdentity<IdentityUser, IdentityRole>().AddEntityFrameworkStores<AppDbContext>();
```



# AJOUTER L'AUTHENTIFICATION

- Ajouter le middleware d'authentification au pipeline de requêtes dans Program.cs :

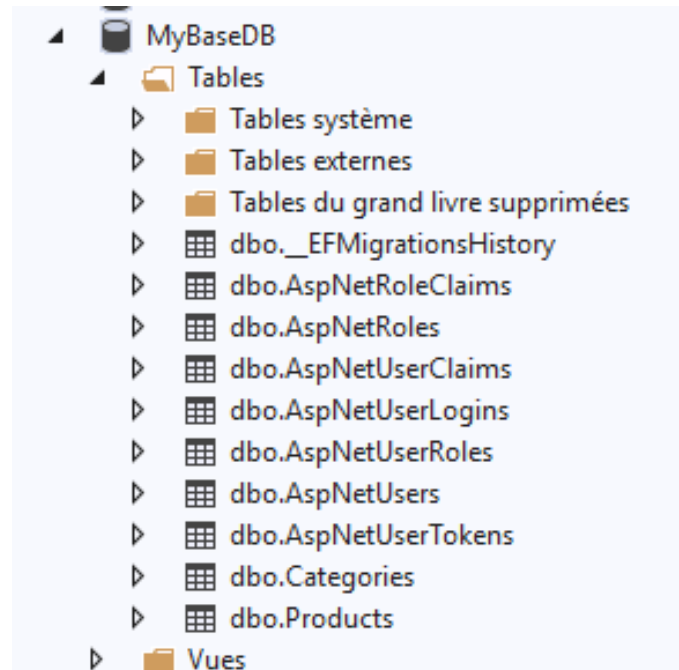
```
app.UseRouting();
```

```
app.UseAuthentication();
```

```
app.UseAuthorization();
```

- Lancer une Migration Entity Framework et générer les tables Identity dans votre base de données de l'application

Add-Migration MajIdentity  
Update-Database



# AJOUTER L'AUTHENTIFICATION

- Ajouter la classe **RegisterViewModel** suivante dans un dossier **ViewModels** :

```
using System.ComponentModel.DataAnnotations;
namespace TP3_Identity.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm password")]
        [Compare("Password",
            ErrorMessage = "Password and confirmation password do not match.")]
        public string ConfirmPassword { get; set; }
    }
}
```



# AJOUTER L'AUTHENTIFICATION

- Ajouter un Contrôleur vide nommé **AccountController** dans lequel vous allez ajouter la méthode d'action suivante :

```
[HttpGet]
public IActionResult Register()
{
    return View();
}
```

- Ajouter la vue Register.cshtml vide correspondante à cette méthode d'action avec ce code :

```
@model TP3_Identity.ViewModels.RegisterViewModel
@{
    ViewBag.Title = "User Registration";
}
<h1>User Registration</h1>
<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
        </form>
    </div>
</div>
```



# AJOUTER L'AUTHENTIFICATION

```
<div class="form-group">
  <label asp-for="Password" class="control-label"></label>
  <input asp-for="Password" class="form-control" />
  <span asp-validation-for="Password" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="ConfirmPassword" class="control-label"></label>
  <input asp-for="ConfirmPassword" class="form-control" />
  <span asp-validation-for="ConfirmPassword" class="text-
danger"></span>
</div>
<div class="form-group">
  <input type="submit" value="Register" class="btn btn-primary" />
</div>
</form>
</div>
</div>
```



# CRÉATION DE L'INTERFACE UTILISATEUR

- Ajouter un lien à la vue Register dans le menu de votre application :

```
<li class="nav-item">  
  <a class="nav-link active" asp-controller="Account" asp-action="Register"> Register </a>  
</li>
```

Gestion des Articles   Home   Liste d'articles ▾   Register

## User Registration

Email

Password

Confirm password

Register



# AJOUT DE NOUVEL UTILISATEUR

- Modifions le contrôleur Account\_Controller pour pouvoir ajouter des utilisateurs :

```
public class AccountController : Controller
{
    // GET: AccountController
    private readonly UserManager<IdentityUser> userManager;
    private readonly SignInManager<IdentityUser>
    signInManager;
    public AccountController(UserManager<IdentityUser>
    userManager, SignInManager<IdentityUser> signInManager)
    {
        this.userManager = userManager;
        this.signInManager = signInManager;
    }
    [HttpGet]
    public IActionResult Register()
    {
        return View();
    }
}
```

```
[HttpPost]
public async Task<IActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        // Copy data from RegisterViewModel to IdentityUser
        var user = new IdentityUser
        {
            UserName = model.Email,
            Email = model.Email
        };
        // Store user data inAspNetUsers database table
        var result = await userManager.CreateAsync(user, model.Password);
        // If user is successfully created, sign-in the user using
        // SignInManager and redirect to index action of HomeController
        if (result.Succeeded)
        {
            await signInManager.SignInAsync(user, isPersistent: false);
            return RedirectToAction("index", "home");
        }
        // If there are any errors, add them to the ModelState object
        // which will be displayed by the validation summary tag helper
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError(string.Empty, error.Description);
        }
    }
    return View(model);
}
```





# AJOUT DE NOUVEL UTILISATEUR

- Les paramètres par défaut sont gérés dans une classe PasswordOptions avec :

Propriété	Description	Faire défaut
RequireDigit	Nécessite un nombre compris entre 0 et 9 dans le mot de passe.	true
RequiredLength	Longueur minimale du mot de passe.	6
RequireLowercase	Nécessite un caractère minuscule dans le mot de passe.	true
RequireNonAlphanumeric	Nécessite un caractère non alphanumérique dans le mot de passe.	true
RequiredUniqueChars	S'applique uniquement à ASP.NET Core 2.0 ou version ultérieure.	1
	Nécessite le nombre de caractères distincts dans le mot de passe.	
RequireUppercase	Nécessite un caractère majuscule dans le mot de passe.	true

- Pour changer la configuration par défaut des mots de passe, appliquer les modifications suivantes :

```
builder.Services.Configure<IdentityOptions>(options =>
{
    // Default Password settings.
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = false;
});
```



# AFFICHER ET MASQUER LES LIENS LOGIN ET LOGOUT

- Selon est ce que l'utilisateur est connecté ou non on va voir comment afficher ou masquer les liens Login, Logout et Register.
- Si l'utilisateur n'est pas connecté on affiche les liens Login et Register dans le Navbar.



- Une fois l'utilisateur est connecté, on affiche le lien Logout dans le Navbar.



# AFFICHER ET MASQUER LES LIENS LOGIN ET LOGOUT

```
@using Microsoft.AspNetCore.Identity
@inject SignInManager<IdentityUser> SignInManager
```

```
<div class="collapse navbar-collapse" id="collapsibleNavbar">
  <ul class="navbar-nav ml-auto">
    @*If the user is signed-in display Logout link*@
    @if(SignInManager.IsSignedIn(User))
    {
      <li class="nav-item">
        <form method="post" asp-controller="account" asp-action="logout">
          <button type="submit" style="width:auto"
            class="nav-link btn btn-link py-0">
            Logout @User.Identity.Name
          </button>
        </form>
      </li>
    }
    else
    {
      <li class="nav-item">
        <a class="nav-link" asp-controller="Account" asp-action="Register">
          Register
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link" asp-controller="Account" asp-action="Login">
          Login
        </a>
      </li>
    }
  </ul>
</div>
```

```
[HttpPost]
public async Task<IActionResult> Logout()
{
    await signInManager.SignOutAsync();
    return RedirectToAction("Index", "Home");
}
```

**Fichier Account\_Controller**

**Fichier\_layout.cshtml**



# IMPLÉMENTATION DE LA FONCTIONNALITÉ LOGIN

- Ajouter la classe suivante au dossier **ViewModels**. Pour connecter un utilisateur, nous avons besoin de son E-Mail qui est le nom d'utilisateur, le mot de passe.

```
public class LoginViewModel
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [Display(Name = "Remember me")]
    public bool RememberMe { get; set; }
}
```



# IMPLÉMENTATION DE LA FONCTIONNALITÉ LOGIN

➤ Dans AccountController Ajouter les méthodes d'actions suivantes :

```
[HttpGet]
public IActionResult Login()
{
    return View();
}
[HttpPost]
public async Task<IActionResult> Login(LoginViewModel model, string? returnUrl)
{
    if (ModelState.IsValid)
    {
        var result = await signInManager.PasswordSignInAsync(model.Email,
            model.Password, model.RememberMe, false);

        if (result.Succeeded)
        {
            if (!string.IsNullOrEmpty(returnUrl))
            {
                return LocalRedirect(returnUrl);
            }
            else
            {
                return RedirectToAction("Index", "Home");
            }
        }

        ModelState.AddModelError(string.Empty, "Invalid Login Attempt");
    }
    return View(model);
}
```



# IMPLÉMENTATION DE LA FONCTIONNALITÉ LOGIN

- Ajouter la vue Login.cshtml vide correspondante à la méthode d'action login avec ce code :

```
@model TP3_Identity.ViewModels.LoginViewModel
@{
    ViewBag.Title = "User Login";
}
<h1>User Login</h1>
@{
    var returnUrl = Context.Request.Query["ReturnUrl"];
}
<div class="row">
    <div class="col-md-12">
        <form asp-route-returnurl = "@returnUrl" method="post">
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Email"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <div class="checkbox">
                    <label asp-for="RememberMe">
                        <input asp-for="RememberMe" />
                        @Html.DisplayNameFor(m => m.RememberMe)
                    </label>
                </div>
            </div>
            <button type="submit" class="btn btn-primary" >Login</button>
        </form>
    </div>
</div>
```



# AJOUTER UNE AUTORISATION D'ACCÈS AU CONTRÔLEUR

- Pour rendre nos contrôleurs ProductController et CategoryController non accessibles que par une authentification, on va ajouter cette annotation :

`[Authorize]`

```
public class CategoryController : Controller
{
    ...
}
```

`[Authorize]`

```
public class ProductController : Controller
{
    ...
}
```

- Maintenant l'accès à Product et Category nécessitent obligatoirement une authentification.  
**Testez Votre Application !!!**
- Pour autoriser l'accès à la méthode d'action Index des contrôleurs, on va ajouter l'annotation suivante :

`// GET: CategoryController`

`[AllowAnonymous]`

```
public ActionResult Index()
{
    var Categories = CategRepository.GetAll();
    return View(Categories);
}
```

`// GET: ProductController`

`[AllowAnonymous]`

```
public ActionResult Index()
{
    var Products = ProductRepository.GetAll();
    return View(Products);
}
```



# CRÉATION DE RÔLES

- Pour mieux gérer notre application, nous allons ajouter un administrateur dont le rôle est l'ajout d'utilisateurs et l'attribution des rôles pour chacun. Pour cela, on va commencer par créer une nouvelle classe **CreateRoleViewModel** dans le dossier **ViewModels** :

```
using System.ComponentModel.DataAnnotations;
namespace TP3_Identity.ViewModels
{
    public class CreateRoleViewModel
    {
        [Required]
        [Display(Name = "Role")]
        public string RoleName { get; set; }
    }
}
```





# CRÉATION DE RÔLES

- Créer le contrôleur **AdminController** suivant :

```
public class AdminController : Controller {
    private readonly RoleManager<IdentityRole> roleManager;
    public AdminController(RoleManager<IdentityRole> roleManager)
    {
        this.roleManager = roleManager;
    }
    [HttpGet]
    public IActionResult CreateRole()
    {
        return View();
    }
    [HttpPost]
    public async Task<IActionResult> CreateRole(CreateRoleViewModel model)
    {
        if (ModelState.IsValid)
        {
            IdentityRole role = new IdentityRole { Name = model.RoleName };
            IdentityResult result = await roleManager.CreateAsync(role);
            if (result.Succeeded)
            {
                return RedirectToAction("ListRoles");
            }
            foreach (IdentityError error in result.Errors)
            {
                ModelState.AddModelError(string.Empty, error.Description);
            }
        }
        return View(model);
    }
    // Reste du code
}
```



# CRÉATION DE RÔLES

- Ajouter la vue correspondante à l'action **CreateRole** suivante :

```
@model TP3_Identity.ViewModels.CreateRoleViewModel
@{
    ViewBag.Title = "Create New Role";
}
<h1>Create New Role</h1>

<form asp-action="CreateRole" method="post" class="mt-3">
    <div asp-validation-summary="All" class="text-danger"></div>
    <div class="form-group row">
        <label asp-for="RoleName" class="col-sm-2-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="RoleName" class="form-control" placeholder="Name">
            <span asp-validation-for="RoleName" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group row">
        <div class="col-sm-10">
            <button type="submit" class="btn btn-primary" style="width : auto">
                Create Role
            </button>
        </div>
    </div>
</form>
```

# AFFICHAGE LISTE DES RÔLES

- Pour gérer tous les rôles de l'application, on va ajouter la méthode d'action **ListRoles** suivante :

```
[HttpGet]
public IActionResult ListRoles()
{
    var roles = roleManager.Roles;
    return View(roles);
}
```

- Et la vue correspondante à cette méthode d'action :

```
@using Microsoft.AspNetCore.Identity
@model IEnumerable<IdentityRole>
@{
    ViewBag.Title = "All Roles";
}
<h1>All Roles</h1>
@if (Model.Any())
{
    <a class="btn btn-primary mb-3" style="width:auto" asp-action="CreateRole"
      asp-controller="Admin">Add new role</a>
    foreach (var role in Model)
    {
        <div class="card mb-3">
            <div class="card-header"> Role Id : @role.Id </div>
            <div class="card-body"> <h5 class="card-title">@role.Name</h5> </div>
            <div class="card-footer">
                <a href="#" class="btn btn-primary">Edit</a>
                <a href="#" class="btn btn-danger">Delete</a>
            </div>
        </div>
    }
}
```



# AFFICHAGE LISTE DE RÔLES

```
else
{
    <div class="card">
        <div class="card-header">
            No roles created yet
        </div>
        <div class="card-body">
            <h5 class="card-title">
                Use the button below to create a role
            </h5>
            <a class="btn btn-primary" style="width:auto"
                asp-controller="Admin" asp-action="CreateRole">
                Create Role
            </a>
        </div>
    </div>
}
```



# MODIFICATION D'UN RÔLE

- Pour pouvoir modifier les rôles déjà créés, on a besoin d'ajouter la classe **EditRoleViewModel** suivante dans le dossier **ViewModels** :

```
public class EditRoleViewModel
{
    public EditRoleViewModel()
    {
        Users = new List<string>();
    }
    public string Id { get; set; }
    [Required(ErrorMessage = "Role Name is required")]
    public string RoleName { get; set; }
    public List<string> Users { get; set; }
}
```

- Au niveau du contrôleur **AdminController**, nous allons ajouter cette déclaration et nous allons modifier le constructeur :

```
public class AdminController : Controller
{
    private readonly RoleManager<IdentityRole> roleManager;
    private readonly UserManager<IdentityUser> userManager;
    public AdminController(RoleManager<IdentityRole> roleManager, UserManager<IdentityUser> userManager)
    {
        this.roleManager = roleManager;
        this.userManager = userManager;
    }
}
```



# MODIFICATION D'UN RÔLE

- Et nous allons aussi ajouter cette méthode d'action :

```
// Role ID is passed from the URL to the action
[HttpGet]
public async Task<IActionResult> EditRole(string id)
{
    // Find the role by Role ID
    var role = await roleManager.FindByIdAsync(id);
    if (role == null)
    {
        ViewBag.ErrorMessage = $"Role with Id = {id} cannot be found";
        return View("NotFound");
    }
    var model = new EditRoleViewModel
    {
        Id = role.Id,
        RoleName = role.Name
    };
    // Retrieve all the Users
    foreach (var user in userManager.Users.ToList())
    {
        // If the user is in this role, add the username to
        // Users property of EditRoleViewModel. This model
        // object is then passed to the view for display
        if (await userManager.IsInRoleAsync(user, role.Name))
        {
            model.Users.Add(user.UserName);
        }
    }
    return View(model);
}
```



# MODIFICATION D'UN RÔLE

- Et nous allons aussi ajouter cette méthode d'action :

```
// This action responds to HttpPost and receives EditRoleViewModel
[HttpPost]
public async Task<IActionResult> EditRole(EditRoleViewModel model)
{
    var role = await roleManager.FindByIdAsync(model.Id);
    if (role == null)
    {
        ViewBag.ErrorMessage = $"Role with Id = {model.Id} cannot be found";
        return View("NotFound");
    }
    else
    {
        role.Name = model.RoleName;
        // Update the Role using UpdateAsync
        var result = await roleManager.UpdateAsync(role);
        if (result.Succeeded)
        {
            return RedirectToAction("ListRoles");
        }
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError("", error.Description);
        }
        return View(model);
    }
}
```



# SUPPRESSION D'UN RÔLE

- Pour pouvoir supprimer un rôle, nous allons commencer par ajouter cette méthode d'action :

```
[HttpPost]
public async Task<IActionResult> DeleteRole(string id)
{
    var role = await roleManager.FindByIdAsync(id);

    if (role == null)
    {
        ViewBag.ErrorMessage = $"Role with Id = {id} cannot be found";
        return View("NotFound");
    }
    else
    {
        var result = await roleManager.DeleteAsync(role);

        if (result.Succeeded)
        {
            return RedirectToAction("ListRoles");
        }

        foreach (var error in result.Errors)
        {
            ModelState.AddModelError("", error.Description);
        }

        return View("ListRoles");
    }
}
```





# MODIFICATION/ SUPPRESSION D'UN RÔLE

- Au niveau de la vue **ListRoles** modifier le code du bouton **Edit** et **Delete** comme suit :

```
<div class="card-footer">
    <form method="post" asp-action="DeleteRole" asp-route-id="@role.Id">
        <a asp-controller="Admin" asp-action="EditRole"
        asp-route-id="@role.Id" class="btn btn-primary">Edit</a>
        <span id="confirmDeleteSpan_@role.Id" style="display:none">
            <span>Are you sure you want to delete?</span>
            <button type="submit" class="btn btn-danger">Yes</button>
            <a href="#" class="btn btn-primary" onclick="confirmDelete('@role.Id', false)">No</a>
        </span>
        <span id="deleteSpan_@role.Id">
            <a href="#" class="btn btn-danger" onclick="confirmDelete('@role.Id', true)">Delete</a>
        </span>
    </form>
</div>
```

- Dans le sous dossier **Js** du dossier **wwwroot**, ajouter un fichier nommé **CustomScript.js** contenant le code suivant :

```
function confirmDelete(uniqueId, isDeleteClicked) {
    var deleteSpan = 'deleteSpan_' + uniqueId;
    var confirmDeleteSpan = 'confirmDeleteSpan_' + uniqueId;

    if (isDeleteClicked) {
        $('#' + deleteSpan).hide();
        $('#' + confirmDeleteSpan).show();
    } else {
        $('#' + deleteSpan).show();
        $('#' + confirmDeleteSpan).hide();
    }
}
```

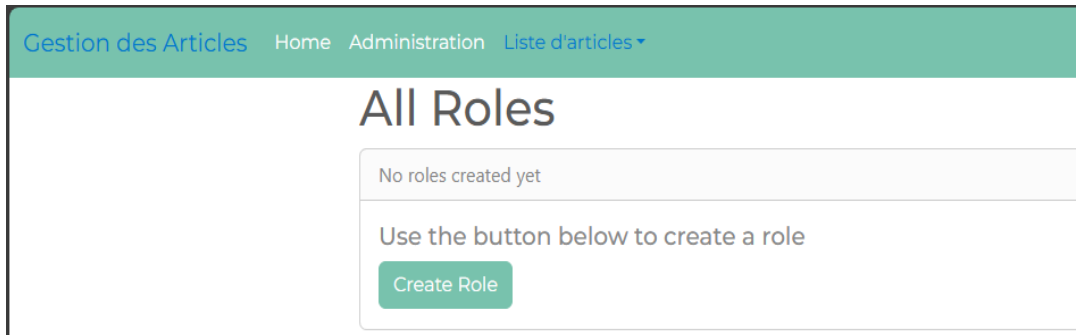


# MODIFICATION D'UN RÔLE

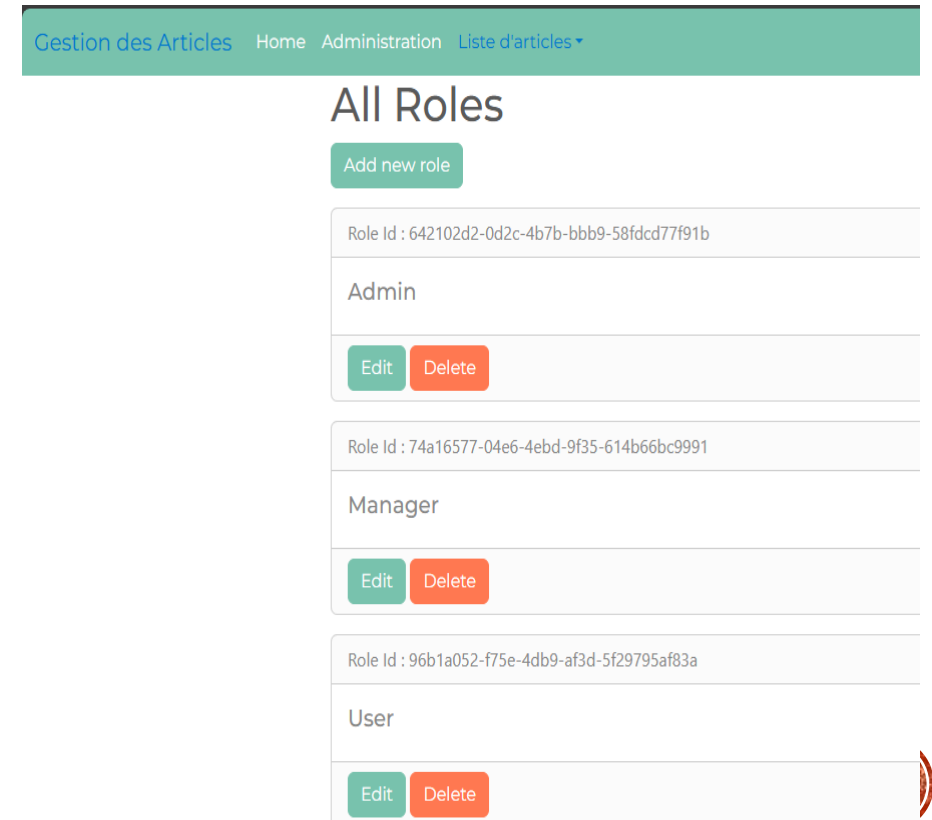
- Au niveau du fichier `_layout.cshtml` ajouter cette instruction pour faire appel à la fonction précédemment créée :

```
<script src="~/js/CustomScript.js"></script>
```

- Ajouter le bouton **Administration** dans le NavBar qui permet d'afficher la vue **ListRoles** :

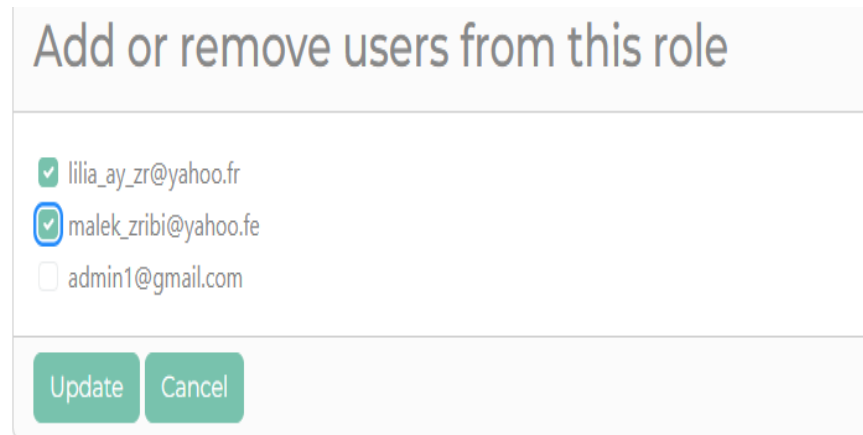


- Exécutez votre application et ajoutez les rôles **Admin**, **User** et **Manager**.



# AJOUT ET SUPPRESSION D'UTILISATEURS D'UN RÔLE DONNÉ

- Dans cette partie, nous allons essayer de programmer les actions **Add Users** et **Remove Users** pour un rôle donné.
- Pour ajouter ou supprimer un utilisateur, il suffit de cocher ou décocher la case correspondante dans la liste des utilisateurs.



The screenshot shows a web interface for managing users. At the top, there is a header "Add or remove users from this role". Below this, there is a list of three users, each with a checkbox and their email address:

- ☒ lilia\_ay\_zr@yahoo.fr
- ☒ malek\_zribi@yahoo.fe
- ☐ admin1@gmail.com

At the bottom of the list, there are two buttons: "Update" and "Cancel".

- On va alors ajouter la classe **UserRoleViewModel** dans le dossier **ViewModels**.

```
public class UserRoleViewModel
{
    public string UserId { get; set; }
    public string UserName { get; set; }
    public bool IsSelected { get; set; }
}
```



# AJOUT ET SUPPRESSION D'UTILISATEURS D'UN RÔLE DONNÉ

- Pour ce faire, commençons par ajouter la vue **EditRole** suivante :

```
@model TP3_Identity.ViewModels.EditRoleViewModel

@{
    ViewBag.Title = "Edit Role";
}

<h1>Edit Role</h1>
<form method="post" class="mt-3">
    <div class="form-group row">
        <label asp-for="Id" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="Id" disabled class="form-control">
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="RoleName" class="col-sm-2 col-form-label"></label>
        <div class="col-sm-10">
            <input asp-for="RoleName" class="form-control">
            <span asp-validation-for="RoleName" class="text-danger"></span>
        </div>
    </div>
    <div asp-validation-summary="All" class="text-danger"></div>
    <div class="form-group row">
        <div class="col-sm-10">
            <button type="submit" class="btn btn-primary">Update</button>
            <a asp-action="ListRoles" class="btn btn-primary">Cancel</a>
        </div>
    </div>
</div>
```



# AJOUT ET SUPPRESSION D'UTILISATEURS D'UN RÔLE DONNÉ

```
<div class="card">
  <div class="card-header">
    <h3>Users in this role</h3>
  </div>
  <div class="card-body">
    @if (Model.Users.Any())
    {
      foreach (var user in Model.Users)
      {
        <h5 class="card-title">@user</h5>
      }
    }
    else
    {
      <h5 class="card-title">None at the moment</h5>
    }
  </div>
  <div class="card-footer">
    <a asp-controller="Admin" asp-action="EditUsersInRole" asp-route-roleId="@Model.Id" class="btn btn-primary"> Add or Remove
    Users from this Role
    </a>
  </div>
</div>
</form>
```



# AJOUT ET SUPPRESSION D'UTILISATEURS D'UN RÔLE DONNÉ

- Au niveau du contrôleur **AdminController** nous allons ajouter les méthodes d'action suivantes :

```
[HttpGet]
public async Task<IActionResult> EditUsersInRole(string roleId)
{
    ViewBag.roleId = roleId;
    var role = await roleManager.FindByIdAsync(roleId);
    if (role == null)
    {
        ViewBag.ErrorMessage = $"Role with Id = {roleId} cannot be found";
        return View("NotFound");
    }
    var model = new List<UserRoleViewModel>();
    foreach (var user in userManager.Users.ToList())
    {
        var userRoleViewModel = new UserRoleViewModel
        {
            UserId = user.Id,
            UserName = user.UserName
        };
        if (await userManager.IsInRoleAsync(user, role.Name))
        {
            userRoleViewModel.IsSelected = true;
        }
        else
        {
            userRoleViewModel.IsSelected = false;
        }
        model.Add(userRoleViewModel);
    }
    return View(model);
}
```



# AJOUT ET SUPPRESSION D'UTILISATEURS D'UN RÔLE DONNÉ

[HttpPost]

```
public async Task<IActionResult> EditUsersInRole(List<UserRoleViewModel> model, string roleId)
{
    var role = await roleManager.FindByIdAsync(roleId);
    if (role == null)
    {
        ViewBag.ErrorMessage = $"Role with Id = {roleId} cannot be found";
        return View("NotFound");
    }
    for (int i = 0; i < model.Count; i++)
    {
        var user = await userManager.FindByIdAsync(model[i].UserId);
        IdentityResult result = null;
        if (model[i].IsSelected && !(await userManager.IsInRoleAsync(user, role.Name)))
        {
            result = await userManager.AddToRoleAsync(user, role.Name);
        }
        else if (!model[i].IsSelected && await userManager.IsInRoleAsync(user, role.Name))
        {
            result = await userManager.RemoveFromRoleAsync(user, role.Name);
        }
        else
        {
            continue;
        }
        if (result.Succeeded)
        {
            if (i < (model.Count - 1))
                continue;
            else
                return RedirectToAction("EditRole", new { Id = roleId });
        }
    }
    return RedirectToAction("EditRole", new { Id = roleId });
}
```



# AJOUT ET SUPPRESSION D'UTILISATEURS D'UN RÔLE DONNÉ

- Le code de la vue **EditUsersInRole** sera le suivant :

```
@using TP3_Identity.ViewModels
@model List<UserRoleViewModel>

@{
    var roleId = ViewBag.roleId;
}

<form method="post">
    <div class="card">
        <div class="card-header">
            <h2>Add or remove users from this role</h2>
        </div>
        <div class="card-body">
            @for (int i = 0; i < Model.Count; i++)
            {
                <div class="form-check m-1">
                    <input type="hidden" asp-for="@Model[i].UserId" />
                    <input type="hidden" asp-for="@Model[i].UserName" />
                    <input asp-for="@Model[i].IsSelected" class="form-check-input" />
                    <label class="form-check-label" asp-for="@Model[i].IsSelected">
                        @Model[i].UserName
                    </label>
                </div>
            }
        </div>
        <div class="card-footer">
            <input type="submit" value="Update" class="btn btn-primary"
                style="width:auto" />
            <a asp-action="EditRole" asp-route-id="@roleId"
                class="btn btn-primary" style="width:auto">Cancel</a>
        </div>
    </div>
</form>
```





# AJOUT ET SUPPRESSION D'UTILISATEURS D'UN RÔLE DONNÉ

- Exécutez votre application.
- Créer des utilisateurs et précisez le rôle de chacun.

[Gestion des Articles](#) [Home](#) [Administration](#) [Liste d'articles ▾](#)

## Edit Role

Id

642102d2-0d2c-4b7b-bbb9-58fcd77f91b

RoleName

Admin

Update

Cancel

Users in this role

None at the moment

Add or Remove Users from this Role

[Gestion des Articles](#) [Home](#) [Administration](#) [Liste d'articles ▾](#)

## Add or remove users from this role

☒ lilia\_ay\_zr@yahoo.fr

☒ malek\_zribi@yahoo.fe

☐ admin1@gmail.com

Update

Cancel



# ATTRIBUTION D'AUTORISATION POUR UN RÔLE DONNÉ

- On va interdire l'accès au contrôleur **Admin** pour les non-membres du rôle **Administrateur** en ajoutant cette instruction dans le code de **AdminController** :

```
[Authorize(Roles = "Admin")]  
public class AdminController : Controller  
{...}
```

- Au niveau du contrôleur **AccountController** nous allons ajouter cette méthode d'action :

```
[AllowAnonymous]  
    public IActionResult AccessDenied()  
    {  
        return View();  
    }
```

- Et créer la vue **AccessDenied** correspondante avec le code suivant :

```
<div class="text-center">  
    <h1 class="text-danger">Access Denied</h1>  
    <h6 class="text-danger">You do not have persmission to view this resource</h6>  
      
</div>
```

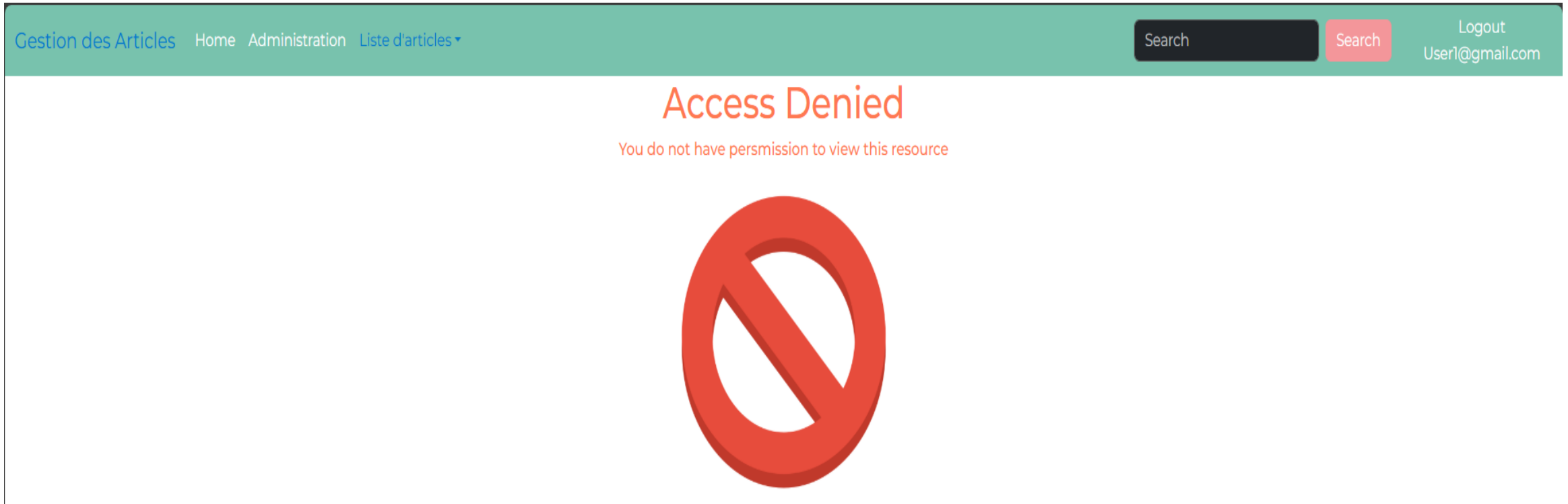


**RQ :** Pour l'image, ajouter un dossier **images** sous le dossier **root** de votre application et ajouter une image nommée **noaccess.png**



# ATTRIBUTION D'AUTORISATION POUR UN RÔLE DONNÉ

- Un utilisateur dont le rôle est différent de **Admin** n'aura plus l'autorisation pour accéder à l'administration et la fenêtre suivante sera affichée :



# AFFICHER OU MASQUER LE MENU DE NAVIGATION EN FONCTION DU RÔLE DE L'UTILISATEUR CONNECTÉ

- Le menu de navigation **Administration** ne sera affiché seulement si l'utilisateur connecté dispose du rôle Admin. Pour cela, au niveau du fichier \_layout.cshtml on ajoutera le code suivant :

```
@if (SignInManager.IsSignedIn(User) && User.IsInRole("Admin"))
{
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" data-bs-toggle="dropdown" href="#" role="button" aria-
  haspopup="true" aria-expanded="false">Administration</a>
  <div class="dropdown-menu">
    <a class="dropdown-item" asp-area="" asp-controller="Admin" asp-action="CreateRole">Create
    Role</a>
    <div class="dropdown-divider"></div>
    <a class="dropdown-item" asp-area="" asp-controller="Admin" asp-action="ListRoles">List
    Roles</a>
  </div>
</li>
}
```

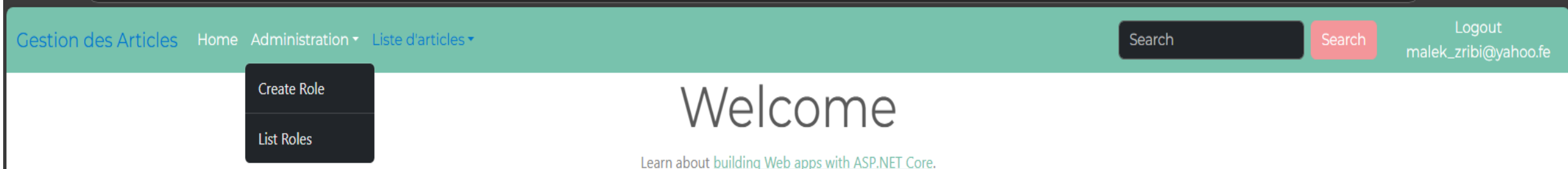


# AFFICHER OU MASQUER LE MENU DE NAVIGATION EN FONCTION DU RÔLE DE L'UTILISATEUR CONNECTÉ

- On aura alors l'affichage suivant :



Menu pour un utilisateur  
dont le rôle est **User**



Menu pour un utilisateur  
dont le rôle est **Admin**



# AJOUT DE MANAGER PRODUCT ET CATEGORY

- On vous demande d'ajouter un nouveau rôle **Manager**. Ce rôle aura la possibilité avec l'Admin de créer, modifier, supprimer et afficher la liste des **Products** ou **Categories**. Pour le rôle **User**, il n'aura la possibilité que de consulter les listes. Pour cela, au niveau des contrôleurs **ProductController** et **CategoryController**, ajouter l'autorisation suivante :

```
[Authorize(Roles = "Admin,Manager")]
public class ProductController : Controller
{
    readonly IProductRepository ProductRepository;
    readonly ICategorieRepository CategRepository;
    private readonly IWebHostEnvironment hostingEnvironment;

    ...
    // GET: ProductController
    [AllowAnonymous]
    public ActionResult Index()
    {
        var Products = ProductRepository.GetAll();
        return View(Products);
    }
}
```

```
[Authorize(Roles = "Admin,Manager")]
public class CategoryController : Controller
{
    readonly ICategorieRepository CategRepository;
    public CategoryController(ICategorieRepository
                             categRepository)

    ...
    // GET: CategoryController
    [AllowAnonymous]
    public ActionResult Index()
    {
        var Categories = CategRepository.GetAll();
        return View(Categories);
    }
}
```

