# Custom Authorization

Η υλοποίηση ενός custom authorization attribute θα γίνει με την χρήση ενός δικού μας attribute που θα αντικαθιστά το [Authorize] attribute.

Αυτό θα γίνει με μια κλάση που θα υλοποιεί το IAuthorizationFilter interface μέσω ενός TypeFilterAttribute ώστε να έχουμε την δυνατότητα να εισάγουμε services (inject) με την βοήθεια της DI (Dependency Injection). Δείτε αναλυτική εξήγηση στα συνοδευτικά έγγραφα   
**Custom Authorizatio.docx, Custom** **Authorization Attribute.docx.**

Το custom Attribute υλοποιεί την OnAuthorization μέθοδο που εκτελείται πριν από κάθε Action.

Ελέγχουμε αν υπάρχει πιστοποιημένος χρήστης με την βοήθεια μιας νέας κλάσης της WorkContext που μας παρέχει όποτε χρειαστούμε, εκτός των άλλων, και τον πιστοποιημένο χρήστη, εάν υπάρχει.

Εάν υπάρχει καλείται η PermissionService.AuthorizeAsync("AdminPanel") που ελέγχει αν στους ρόλους που ανήκει ο χρήστης έχουν εκχωρημένο το ζητούμενο δικαίωμα (permission).

Εάν δεν υπάρχει πιστοποιημένος χρήστης, η ο πιστοποιημένος χρήστης δεν έχει το ζητούμενο δικαίωμα, η OnAuthorization καλεί την ChallengeResult() του AspNetCore.Mvc που θα οδηγήσει τον χρήστη σε ένα **route** που καθορίζεται στην αρχική δήλωση του **authentication cookie** που θα δούμε πιο κάτω.

Στο program.cs αρχείο προσθέτουμε τις ακόλουθες δηλώσεις στην αρχή των δηλώσεων για τα service της εφαρμογής.

//set default authentication schemes

var authenticationBuilder = builder.Services.AddAuthentication(options => {

options.DefaultChallengeScheme =

SportStoreAuthenticationDefaults.AuthenticationScheme;

options.DefaultScheme =

SportStoreAuthenticationDefaults.AuthenticationScheme;

});

//add main cookie authentication

authenticationBuilder.AddCookie(SportStoreAuthenticationDefaults.AuthenticationScheme,

options => {

options.Cookie.Name =

$"{SportsCookieDefaults.Prefix}{SportsCookieDefaults.AuthenticationCookie}";

options.Cookie.HttpOnly = true;

options.Cookie.SecurePolicy = CookieSecurePolicy.SameAsRequest;

options.LoginPath = SportStoreAuthenticationDefaults.LoginPath;

options.AccessDeniedPath = SportStoreAuthenticationDefaults.AccessDeniedPath;

});

Στα options του AddAuthentication service καθορίζουμε το όνομα (string) του σχήματος που θα χρησιμοποιήσουμε στο cookie.

Στα options του cookie που θα χρησιμοποιηθεί για τον πιστοποιημένο χρήστη καθορίζουμε το όνομά του, και εκτός των άλλων το LoginPathκαθώς και τοAccessDeniedPath που θα χρησιμοποιηθούν για να ανακατευθύνουν τον χρήστη αναλόγως.

Τις τιμές των options τις αποθηκεύουμε για εύκολη πρόσβαση στις βοηθητικές κλάσεις SportStoreAuthenticationDefaults και SportsCookieDefaults.

Μετά την εντολή app.UseRouting(); προσθέτουμε την εντολή app.UseAuthorization(); που προσθέτει το κατάλληλο Authorization middleware.

Επίσης ορίζουμε και τα νέα endpoints που χρειαζόμαστε πλέον ως εξής:

app.UseEndpoints(endpoints => {

app.MapControllerRoute("Default", "{controller=Home}/{action=Index}/{id?}");

//home page

app.MapControllerRoute("Homepage",

"/",

new { controller = "Home", action = "Index" });

app.MapControllerRoute("Login",

"login/",

new { controller = "Customer", action = "Login" });

});

Καταργούμε την απλή δήλωση app.MapDefaultControllerRoute(); μια και την ορίσαμε πιο πάνω με κυριολεκτικό τρόπο.

Η Login ActionResult μέθοδος του CustomerController θα χρησιμοποιήσει την CustomerRegistrationService.ValidateCustomerAsync για επαληθεύσει τον χρήστη που επιχειρεί να πιστοποιηθεί.

Η κλάση CustomerRegistrationService έχει δύο μεθόδους τις:

* ValidateCustomerAsync
* SignInCustomerAsync

Η SignInCustomerAsync χρησιμοποιεί την SignInAsync της κλάσης CookieAuthenticationService για να δημιουργήσει τα απαιτούμενα claims, principals κτλ, και στο τέλος να χρησιμοποιήσει το σχήμα που δηλώσαμε στο program.cs, δηλαδή την δημιουργία του κατάλληλου cookie που θα πιστοποιεί τον χρήστη.

Επίσης διαθέτει και την μέθοδο SignOutAsync, που εξασφαλίζει την ασφαλή έξοδο του χρήστη από την εφαρμογή.

**Σημ**: Τα αρχεία με τις κλάσεις που υλοποιούν την πιο πάνω λειτουργικότητα επισυνάπτονται.