2026

«Διαχείριση και Σύνθεση ομάδων εργαστηρίων» & Κατασκευή REST API

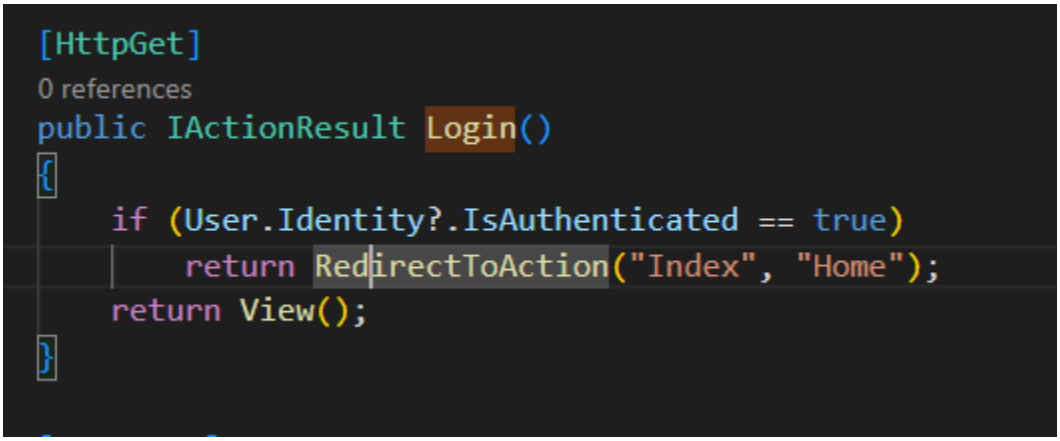
ΒΛΑΧΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ Π22024
VLACHOSKONSTANTINOS02@GMAIL.COM

Θέμα 1: Διαχείριση portal εργαστηριακών ασκήσεων

Controllers:

- **AccountController:** Ο συγκεκριμένος controller ασχολείται με την εγγραφή, τη σύνδεση και την αποσύνδεση του χρήστη.

    - **Η διαδικασία Login (Get):** χρησιμοποιείται για την εμφάνιση της φόρμας σύνδεσης. Όταν ένας χρήστης επιχειρεί να μεταβεί στη σελίδα σύνδεσης, το σύστημα ελέγχει αν υπάρχει ήδη ενεργή αυθεντικοποιήση μέσω cookies. Αν ο χρήστης είναι ήδη συνδεδεμένος, αποφεύγεται η επανάληψη της διαδικασίας Login και αποφεύγεται ανακατεύθυνση στην αρχική σελίδα Home. Αν δεν υπάρχει ενεργή συνεδρία, επιστρέφεται το αντίστοιχο View ώστε ο χρήστης να εισάγει τα στοιχεία του. Η συγκεκριμένη διαδικασία διασφαλίζει σωστή ροή πλοήγησης και αποτρέπει περιττές προσπάθειες σύνδεσης από ήδη αυθεντικοποιημένους χρήστες.



    - **Η διαδικασία Login (Post):** υλοποιεί τον μηχανισμό αυθεντικοποίησης με χρήση username και password. Αρχικά γίνεται έλεγχος ότι τα πεδία δεν είναι κενά. Αν κάποιο πεδίο λείπει, επιστρέφει μήνυμα σφάλματος. Στη συνέχεια πραγματοποιείται αναζήτηση του χρήστη στη βάση δεδομένων βάσει του username. Αν ο χρήστης δεν βρεθεί ή αν το has του εισαγόμενου κωδικού, με βάση το αποθηκευμένο salt, δεν ταυτίζεται με το αποθηκευμένο hash, η διαδικασία αποτυγχάνει και εμφανίζει μήνυμα λανθασμένων στοιχειών.

```
if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))
{
    TempData["Error"] = "Παρακαλώ συμπληρώστε username και password.";
    return View();
}

var user = _context.Users.FirstOrDefault(u => u.Username == username);
if (user == null || Password.HashPassword(password, user.Salt) != user.Hash)
{
    TempData["Error"] = "Λάθος username ή password.";
    return View();
}
```

Σε περίπτωση επιτυχούς ταυτοποίησης, δημιουργείται λίστα Claims που περιλαμβάνει τον μοναδικό αναγνωριστικό χρήστη, το username και το email. Με βάση αυτά τα Claims δημιουργείται το ClaimsIdentity και στη συνέχεια ClaimsPrincipal. Το σύστημα δημιουργεί authentication cookie με persistent ιδιότητα, επιτρέποντας τη διατήρηση της συνεδρίας. Τέλος, ο χρήστης ανακατευθύνεται στην αρχική σελίδα. Η διαδικασία διασφαλίζει ότι οι κωδικοί δεν αποθηκεύονται σε απλή μορφή αλλά γίνεται χρήση hash και salt για αύξηση ασφάλειας.

```
    var claims = new List<Claim>
    {
        new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
        new Claim(ClaimTypes.Name, user.Username),
        new Claim(ClaimTypes.Email, user.Email)
    };

    var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationSc
    var principal = new ClaimsPrincipal(identity);

    await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, princi
        new AuthenticationProperties { IsPersistent = true });

    TempData["Success"] = $"Καλώς ήρθες, {user.Username}!";
    return RedirectToAction("Index", "Home");
}
```

- **Η διαδικασία Register (Get):** εμφανίζει τη φόρμα εγγραφής νέου χρήστη. Πριν από την εμφάνιση, γίνεται έλεγχος αν ο χρήστης είναι

ήδη συνδεδεμένος. Αν υπάρχει ενεργή συνεδρία, πραγματοποιείται ανακατεύθυνση στην αρχική σελίδα. Αν όχι, επιστρέφεται η φόρμα εγγραφής. Με αυτόν τον τρόπο αποφεύγεται η δημιουργία νέου λογαριασμού από ήδη συνδεδεμένο χρήστη.

```
[HttpGet]
0 references
public IActionResult Register()
{
    if (User.Identity?.IsAuthenticated == true)
        return RedirectToAction("Index", "Home");
    return View();
}
```

- **Η διαδικασία Register (Post):** υλοποιεί τη δημιουργία νέου λογαριασμού. Αρχικά, γίνεται έλεγχος ότι όλα τα υποχρεωτικά πεδία (username, email, password) είναι συμπληρωμένα. Στη συνέχεια ελέγχεται αν ο κωδικός και η επιβεβαίωση κωδικού ταυτίζονται. Αν υπάρχει ασυμφωνία, η διαδικασία διακόπτεται. Ακολουθεί ο έλεγχος μοναδικότητάς του username στη βάση δεδομένων ώστε να αποφευχθούν διπλότυποι λογαριασμοί. Αν το username υπάρχει ήδη, θα εμφανιστεί μήνυμα σφάλματος.

```
if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(email) || string.IsNullOrEmpt
{
    TempData["Error"] = "Όλα τα πεδία είναι υποχρεωτικά.";
    return View();
}

if (password != confirmPassword)
{
    TempData["Error"] = "Οι κωδικοί δεν ταιριάζουν.";
    return View();
}

if (_context.Users.Any(u => u.Username == username))
{
    TempData["Error"] = "Το username υπάρχει ήδη.";
    return View();
}
```

Σε περίπτωση επιτυχούς ελέγχου, δημιουργείται μοναδικό salt και υπολογίζεται το hash του κωδικού. Δημιουργείται νέο αντικείμενο χρήστη με αποθηκευμένα salt και hash και καταχωρείται στη βάση δεδομένων. Μετά την επιτυχή εγγραφή, πραγματοποιείται αυτόματη σύνδεση του χρήστη μέσω δημιουργίας claims και authentication cookie, χωρίς να απαιτείται ξεχωριστό login.

```
_context.Users.Add(user);
await _context.SaveChangesAsync();

// Auto sign-in after registration
var claims = new List<Claim>
{
    new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
    new Claim(ClaimTypes.Name, user.Username),
    new Claim(ClaimTypes.Email, user.Email)
};

var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationSch
var principal = new ClaimsPrincipal(identity);

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, princip
    new AuthenticationProperties { IsPersistent = true });

TempData["Success"] = "Η εγγραφή ολοκληρώθηκε επιτυχώς!";
return RedirectToAction("Index", "Home");
```

- **Η διαδικασία GoogleLogin (Get):** ξεκινά τη διαδικασία αυθεντικοποίησης μέσω Google. Δημιουργούνται authentication properties με καθορισμένο RedirectUri, το οποίο δείχνει στη μέθοδο GoogleResponse. Στη συνέχεια ενεργοποιείται το authentication challenge προς το Google Authentication Scheme. Ο χρήστης μεταφέρεται στη σελίδα σύνδεσης της Google και μετά την επιτυχή αυθεντικοποίηση επιστρέφει στην εφαρμογή.

```
[HttpGet]
0 references
public IActionResult GoogleLogin()
{
    var properties = new AuthenticationProperties
    {
        RedirectUri = Url.Action("GoogleResponse")
    };
    return Challenge(properties, GoogleDefaults.AuthenticationScheme);
}
```

- **Η διαδικασία GoogleResponse:** διαχειρίζεται την απάντηση από την Google μετά την επιτυχή σύνδεση. Αρχικά ανακτάται το authentication result από το cookie scheme. Αν δεν υπάρχουν διαθέσιμα στοιχεία, η διαδικασία αποτυγχάνει. Στη συνέχεια εξάγονται τα Claims που περιλαμβάνουν το email και το όνομα του χρήστη. Αν κάποιο από αυτά λείπει, εμφανίζεται μήνυμα σφάλματος. Ακολουθεί ο έλεγχος στη βάση δεδομένων για ύπαρξη χρήστη με το συγκεκριμένο email. Αν δεν υπάρχει, δημιουργείται νέος χρήστης με τα στοιχεία που επιστράφηκαν από την Google. Στην περίπτωση αυτή δεν αποθηκεύονται salt και hash, καθώς η αυθεντικοποιήση γίνεται από την Google.

```
{
    var result = await HttpContext.AuthenticateAsync(CookieAuthenticationDefaults.Authentica
    if (result?.Principal == null)
    {
        TempData["Error"] = "Αποτυχία σύνδεσης με Google.";
        return RedirectToAction("Login");
    }

    var claims = result.Principal.Identities.FirstOrDefault()?.Claims;
    var email = claims?.FirstOrDefault(c => c.Type == ClaimTypes.Email)?.Value;
    var username = claims?.FirstOrDefault(c => c.Type == ClaimTypes.Name)?.Value;

    if (email == null || username == null)
    {
        TempData["Error"] = "Δεν ήταν δυνατή η ανάκτηση στοιχείων από το Google.";
        return RedirectToAction("Login");
    }

    var user = _context.Users.FirstOrDefault(u => u.Email == email);
```

```
// Sign in with cookie
var userClaims = new List<Claim>
{
    new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
    new Claim(ClaimTypes.Name, user.Username),
    new Claim(ClaimTypes.Email, user.Email)
};

var identity = new ClaimsIdentity(userClaims, CookieAuthenticationDefaults.Authenticat
var principal = new ClaimsPrincipal(identity);

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, princ
    new AuthenticationProperties { IsPersistent = true });

TempData["Success"] = $"Καλώς ήρθες, {user.Username}!";
return RedirectToAction("Index", "Home");
```

Τέλος, δημιουργείται authentication cookie με Claims και ο χρήστης ανακατευθύνεται στην αρχική σελίδα.

- Η διαδικασία Logout τερματίζει την ενεργή συνεδρία του χρήστη. Καλείται η μέθοδος SignOutAsync για το cookie authentication scheme, με αποτέλεσμα τη διαγραφή του authentication cookie. Μετά την αποσύνδεση ο χρήστης ανακατευθύνεται στην αρχική σελίδα Login.

```
[HttpGet]
0 references
public async Task<IActionResult> Logout()
{
    await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
    return RedirectToAction("Login");
}
```

- **ContactController:** διαχειρίζεται τη δημιουργία και την προβολή προσωπικών επαφών μεταξύ χρηστών του συστήματος. Ο controller προστατεύεται με το attribute Authorize, πράγμα που σημαίνει ότι όλες οι λειτουργίες του είναι διαθέσιμες μόνο σε αυθεντικοποιημένους χρήστες.
    - **Η διαδικασία Index:** χρησιμοποιείται για την προβολή της λίστας προσωπικών επαφών του συνδεδεμένου χρήστη. Αρχικά, ανακτάται το μοναδικό αναγνωριστικό του χρήστη από τα authentication Claims, το οποίο έχει αποθηκευτεί κατά τη διαδικασία Login. Στη συνέχεια πραγματοποιείται αναζήτηση στη βάση δεδομένων για όλες τις εγγραφές στον πίνακα Contract, όπου το UserId αντιστοιχεί στον τρέχων χρήστη. Με τη βοήθεια του Include γίνεται load το αντικείμενο ContactUser, ώστε να είναι διαθέσιμα τα πλήρη στοιχεία του χρήστη που έχει προστεθεί ως επαφή.
    -

```
int userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0");

var contacts = await _context.Contact
    .Where(c => c.UserId == userId)
    .Include(c => c.ContactUser)
    .ToListAsync();
```

Το ViewBag.AllUsers αποθηκεύει λίστα με όλους τους υπόλοιπους χρήστες του συστήματος (εκτός του τρέχοντος), ώστε να μπορούν να εμφανιστούν στη Διεπαφή και να επιλέγουν για προσθήκη ως επαφή.

```
ViewBag.AllUsers = await _context.Users
    .Where(u => u.Id != userId)
    .ToListAsync();
```

    - **Η διαδικασία Create (Post):** υλοποιεί την προθήκη νέας επαφής. Αρχικά ανακτάται το αναγνωριστικό του τρέχοντος χρήστη από τα Claims. Στη συνέχεια πραγματοποιούνται διαδοχικοί έλεγχοι εγκυρότητας :
        - Έλεγχος αν ο χρήστης προσπαθεί να προσθέσει τον εαυτό του.
        - Έλεγχος αν υπάρχει ήδη καταχωρημένη επαφή με τα ίδια στοιχεία ίδια user id των χρηστών.
        - Έλεγχος αν υπάρχει ο χρήστης στη βάση.

```
{
    TempData["Error"] = "Δεν μπορείτε να προσθέσετε τον εαυτό σας.";
    return RedirectToAction("Index");
}

var existing = await _context.Contact
    .AnyAsync(c => c.UserId == userId && c.ContactUserId == contactUserId);

if (existing)
{
    TempData["Error"] = "Η επαφή υπάρχει ήδη.";
    return RedirectToAction("Index");
}

var                                          (contactUserId);
if (contactUser == null)
{
    TempData["Error"] = "Ο χρήστης δεν βρέθηκε.";
    return RedirectToAction("Index");
```

(local variable) Models.User? contactUser

'contactUser' may be null here.

- **Ο MessageController** διαχειρίζεται την αποστολή, τη λήψη και την προβολή προσωπικών ή ομαδικών μηνυμάτων μεταξύ χρηστών. Ο controller προστατεύεται με το Authorize, με αποτέλεσμα να λειτουργεί μόνο για αυθεντικοποιημένους χρήστες.
    - **Η διαδικασία Index (Get)** εμφανίζει τη λίστα των συνομιλιών του συνδεδεμένου χρήστη. Αρχικά ανακτάται το αναγνωριστικό του χρήστη από το authentication claims. Στη συνέχεια εντοπίζονται όλοι οι χρήστες με τους οποίους έχει υπάρξει επικοινωνία, είτε αποστολέας είτε παραλήπτης.

```
int userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0");

var sentToUserIds = _context.Messages
    .Where(m => m.SenderId == userId)
    .SelectMany(m => m.Receivers.Select(r => r.ReceiverId))
    .ToList();

var receivedFromUserIds = _context.Messages
    .Where(m => m.Receivers.Any(r => r.ReceiverId == userId))
    .Select(m => m.SenderId)
    .ToList();

var allUserIds = sentToUserIds.Concat(receivedFromUserIds)
    .Distinct()
    .Where(id => id != userId)
    .ToList();
```

Για κάθε συνομιλία υπολογίζονται

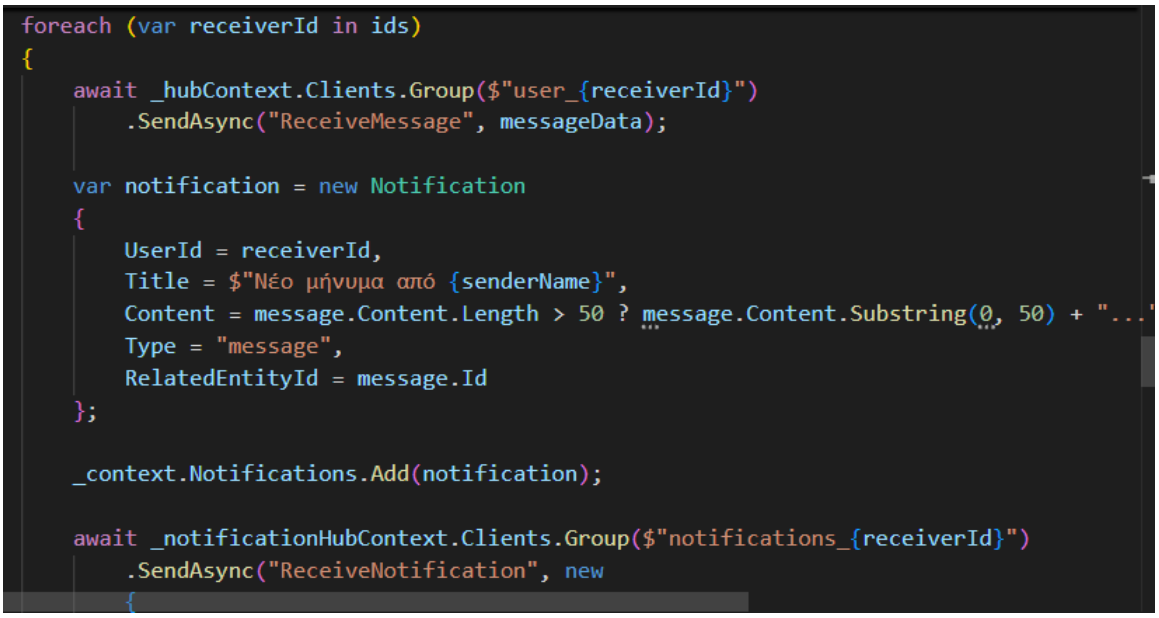
- Ο αριθμός μη αναγνωσμένων μηνυμάτων.
- Το τελευταίο μήνυμα της συνομιλίας.
- Ο χρόνος αποστολής του τελευταίου μηνύματος.

```
var unreadCount = _context.Receivers
    .Include(r => r.Message)
    .Count(r => r.ReceiverId == userId &&
                r.Message.SenderId == otherUserId &&
                !r.IsRead);

var lastMsg = _context.Messages
    .Include(m => m.Receivers)
    .Where(m =>
        (m.SenderId == userId && m.Receivers.Any(r => r.ReceiverId == otherUserId))
        (m.SenderId == otherUserId && m.Receivers.Any(r => r.ReceiverId == userId))
    )
    .OrderByDescending(m => m.CreatedAt)
    .Select(m => new { m.Content, m.CreatedAt })
    .FirstOrDefault();
```

- **Η διαδικασία Conversation (Get)** εμφανίζει το ιστορικό συνομιλίας μεταξύ του τρέχοντος χρήστη και ενός συγκεκριμένου άλλου χρήστη. Αρχικά ελέγχεται αν ο άλλος ο χρήστης υπάρχει. Στη συνέχεια ανακτώνται όλα τα μηνύματα που έχουν ανταλλαχθεί μεταξύ των δύο χρηστών, ταξινομημένα χρονολογικά. Για κάθε μήνυμα επιστρέφονται
  - Το περιεχόμενο
  - Ο αποστολέας
  - Η ημερομηνία αποστολής
  - Ένδειξη αν το μήνυμα στάλθηκε από τον τρέχοντα χρήστη.

```
    .Include(m => m.Sender)
    .Where(m =>
        (m.SenderId == currentUserId && m.Receivers.Any(r => r.ReceiverId == userId))
        (m.SenderId == userId && m.Receivers.Any(r => r.ReceiverId == currentUserId))
    )
    .OrderBy(m => m.CreatedAt)
    .Select(m => new
    {
        m.Id,
        m.Content,
        m.SenderId,
        SenderName = m.Sender.Username,
        m.CreatedAt,
        IsSentByMe = m.SenderId == currentUserId
    })
    .ToList();
```

- **Η διαδικασία Send(Post)** υλοποιεί την αποστολή μηνύματος και υποστηρίζει τόσο κανονικό form όσο και Ajax κλήσεις. Ανακτάται το αναγνωριστικό του αποστολέα, γίνεται έλεγχος αν υπάρχει περιεχόμενο και παραλήπτης, διαφορετικά επιστρέφεται κατάλληλο μήνυμα σφάλματος. Οι παραλήπτες μετατρέπονται από συμβολοσειρά σε λίστα ακέραιων Ids. Για κάθε παραλήπτη δημιουργείται ένα real time ενημέρωση μέσω SignalR.



```
foreach (var receiverId in ids)
{
    await _hubContext.Clients.Group($"user_{receiverId}")
        .SendAsync("ReceiveMessage", messageData);

    var notification = new Notification
    {
        UserId = receiverId,
        Title = $"Νέο μήνυμα από {senderName}",
        Content = message.Content.Length > 50 ? message.Content.Substring(0, 50) + "...
        Type = "message",
        RelatedEntityId = message.Id
    };

    _context.Notifications.Add(notification);

    await _notificationHubContext.Clients.Group($"notifications_{receiverId}")
        .SendAsync("ReceiveNotification", new
        {
```

- **Ο PostController:** διαχειρίζεται τη δημιουργία, την αναζήτηση, την προβολή και την κατηγοριοποίηση δημοσιεύσεων στο σύστημα.
    - **Η διαδικασία Index** εμφανίζει τη λίστα όλων των posts του συστήματος. Αν δοθεί παράμετρος αναζήτησης (search), το query φιλτράρει τα posts ώστε να επιστρέφονται μόνο όσα περιέχουν τη συγκεκριμένη λέξη στον τίτλο ή στο περιεχόμενο. Τα posts ταξινομούνται κατά φθίνουσα ημερομηνία δημιουργίας (νεότερα πρώτα) και μετατρέπονται σε λίστα.

```
0 references
public async Task<IActionResult> Index(string? search)
{
    var query = _context.Posts.Include(p => p.User).AsQueryable();

    if (!string.IsNullOrEmpty(search))
    {
        query = query.Where(p => p.Title.Contains(search) || p.Content.Contains(search));
        ViewBag.Search = search;
    }

    var posts = await query.OrderByDescending(p => p.CreatedAt).ToListAsync();
    ViewBag.Categories = await _context.Categories.ToListAsync();
    return View(posts);
}
```

    - Η διαδικασία Create (POST) υλοποιεί τη δημιουργία νέας δημοσίευσης. Αρχικά ανακτάται το αναγνωριστικό του συνδεδεμένου χρήστη από τα authentication claims. Γίνεται έλεγχος ότι ο τίτλος και το περιεχόμενο δεν είναι κενά. Αν κάποιο πεδίο λείπει, εμφανίζεται μήνυμα σφάλματος και η διαδικασία διακόπτεται.

```
int userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0");

if (string.IsNullOrEmpty(title) || string.IsNullOrEmpty(content))
{
    TempData["Error"] = "Ο τίτλος και το περιεχόμενο είναι υποχρεωτικά.";
    return View();
}
```

    Αν τα δεδομένα είναι έγκυρα, δημιουργείται νέο αντικείμενο Post.

    - Η διαδικασία Details εμφανίζει αναλυτικά στοιχεία συγκεκριμένου post. Αναζητείται το post βάσει του αναγνωριστικού του και φορτώνονται τα στοιχεία του δημιουργού. Αν το post δεν βρεθεί,

επιστρέφεται NotFound. Στη συνέχεια ανακτώνται όλες οι κατηγορίες που σχετίζονται με το συγκεκριμένο post μέσω του πίνακα συσχέτισης PostCategories. Οι κατηγορίες αποθηκεύονται στο ViewBag για εμφάνιση.

```
public async Task<IActionResult> Details(int id)
{
    var post = await _context.Posts
        .Include(p => p.User)
        .FirstOrDefaultAsync(p => p.Id == id);

    if (post == null)
        return NotFound();

    var categories = await _context.PostCategories
        .Where(pc => pc.PostId == id)
        .Include(pc => pc.Category)
        .Select(pc => pc.Category)
        .ToListAsync();

    ViewBag.PostCategories = categories;
    ViewBag.AllCategories = await _context.Categories.ToListAsync();
    return View(post);
}
```

- Η διαδικασία AddCategory συσχετίζει ένα post με μία κατηγορία. Αρχικά ελέγχεται αν υπάρχει ήδη η συγκεκριμένη συσχέτιση μεταξύ post και κατηγορίας. Αν δεν υπάρχει, δημιουργείται νέα εγγραφή στον πίνακα PostCategory και αποθηκεύεται στη βάση. Εμφανίζεται μήνυμα επιτυχίας. Αν η κατηγορία έχει ήδη συσχετιστεί με το post, εμφανίζεται μήνυμα ότι η κατηγορία υπάρχει ήδη.

```
public async Task<IActionResult> AddCategory(int postId, int categoryId)
{
    var exists = await _context.PostCategories
        .AnyAsync(pc => pc.PostId == postId && pc.CategoryId == categoryId);

    if (!exists)
    {
        _context.PostCategories.Add(new PostCategory
        {
            PostId = postId,
            CategoryId = categoryId
        });
        await _context.SaveChangesAsync();
        TempData["Success"] = "Η κατηγορία προστέθηκε!";
    }
    else
    {
        TempData["Error"] = "Η κατηγορία υπάρχει ήδη σε αυτό το post.";
    }
```

- Ο NotificationController διαχειρίζεται την προβολή, την ενημέρωση και τη διαγραφή ειδοποιήσεων του χρήστη.
    - Η διαδικασία Index εμφανίζει όλες τις ειδοποιήσεις του συνδεδεμένου χρήστη. Αρχικά ανακτάται το αναγνωριστικό του χρήστη από τα authentication claims. Στη συνέχεια πραγματοποιείται αναζήτηση στον πίνακα Notifications για όλες τις εγγραφές που αντιστοιχούν στον συγκεκριμένο χρήστη. Οι ειδοποιήσεις ταξινομούνται κατά φθίνουσα ημερομηνία δημιουργίας, ώστε οι πιο πρόσφατες να εμφανίζονται πρώτες.

```
// GET: Notification
0 references
public IActionResult Index()
{
    int userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0");

    var notifications = _context.Notifications
        .Where(n => n.UserId == userId)
        .OrderByDescending(n => n.CreatedAt)
        .ToList();

    return View(notifications);
}
```

- Η διαδικασία MarkRead χρησιμοποιείται για τη σήμανση συγκεκριμένης ειδοποίησης ως αναγνωσμένης. Αρχικά ανακτάται το ID του τρέχοντος χρήστη από τα claims. Στη συνέχεια αναζητείται ειδοποίηση με το συγκεκριμένο ID, με επιπλέον έλεγχο ότι ανήκει στον ίδιο τον χρήστη. Αυτό αποτρέπει την τροποποίηση ειδοποιήσεων άλλων χρηστών. Αν βρεθεί η ειδοποίηση, το πεδίο IsRead τίθεται σε true.

```
0 references
public IActionResult MarkRead(int id)
{
    int userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0");

    var notification = _context.Notifications
        .FirstOrDefault(n => n.Id == id && n.UserId == userId);

    if (notification != null)
    {
        notification.IsRead = true;
        _context.SaveChanges();
    }

    if (Request.Headers["X-Requested-With"] == "XMLHttpRequest")
        return Json(new { success = true });

    return RedirectToAction("Index");
}
```

- Η διαδικασία MarkAllRead χρησιμοποιείται για τη μαζική σήμανση όλων των μη αναγνωσμένων ειδοποιήσεων ως αναγνωσμένων. Αρχικά εντοπίζονται όλες οι ειδοποιήσεις του χρήστη που έχουν IsRead = false. Στη συνέχεια, για κάθε μία από αυτές το πεδίο IsRead ενημερώνεται σε true. Μετά την αποθήκευση των αλλαγών, εμφανίζεται μήνυμα επιτυχίας που περιλαμβάνει τον αριθμό ειδοποιήσεων που ενημερώθηκαν. Τέλος, γίνεται ανακατεύθυνση στη σελίδα Index.

```
public IActionResult MarkAllRead()
{
    int userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0");

    var notifications = _context.Notifications
        .Where(n => n.UserId == userId && !n.IsRead)
        .ToList();

    foreach (var n in notifications)
        n.IsRead = true;

    _context.SaveChanges();

    TempData["Success"] = $"{notifications.Count} ειδοποιήσεις σημειώθηκαν ως αναγνωσμένες
    return RedirectToAction("Index");
}
```

- Η διαδικασία Delete διαγράφει συγκεκριμένη ειδοποίηση. Αρχικά ανακτάται το ID του χρήστη από τα claims. Στη συνέχεια αναζητείται ειδοποίηση με το συγκεκριμένο ID, με επιπλέον έλεγχο ότι ανήκει στον ίδιο τον χρήστη. Αν η ειδοποίηση βρεθεί, αφαιρείται από τον πίνακα Notifications και οι αλλαγές αποθηκεύονται στη βάση δεδομένων. Τέλος, γίνεται ανακατεύθυνση στη σελίδα Index.

```
0 references
public IActionResult Delete(int id)
{
    int userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0");

    var notification = _context.Notifications
        .FirstOrDefault(n => n.Id == id && n.UserId == userId);

    if (notification != null)
    {
        _context.Notifications.Remove(notification);
        _context.SaveChanges();
    }

    return RedirectToAction("Index");
}
```

Θέμα 2: Δημιουργία REST API

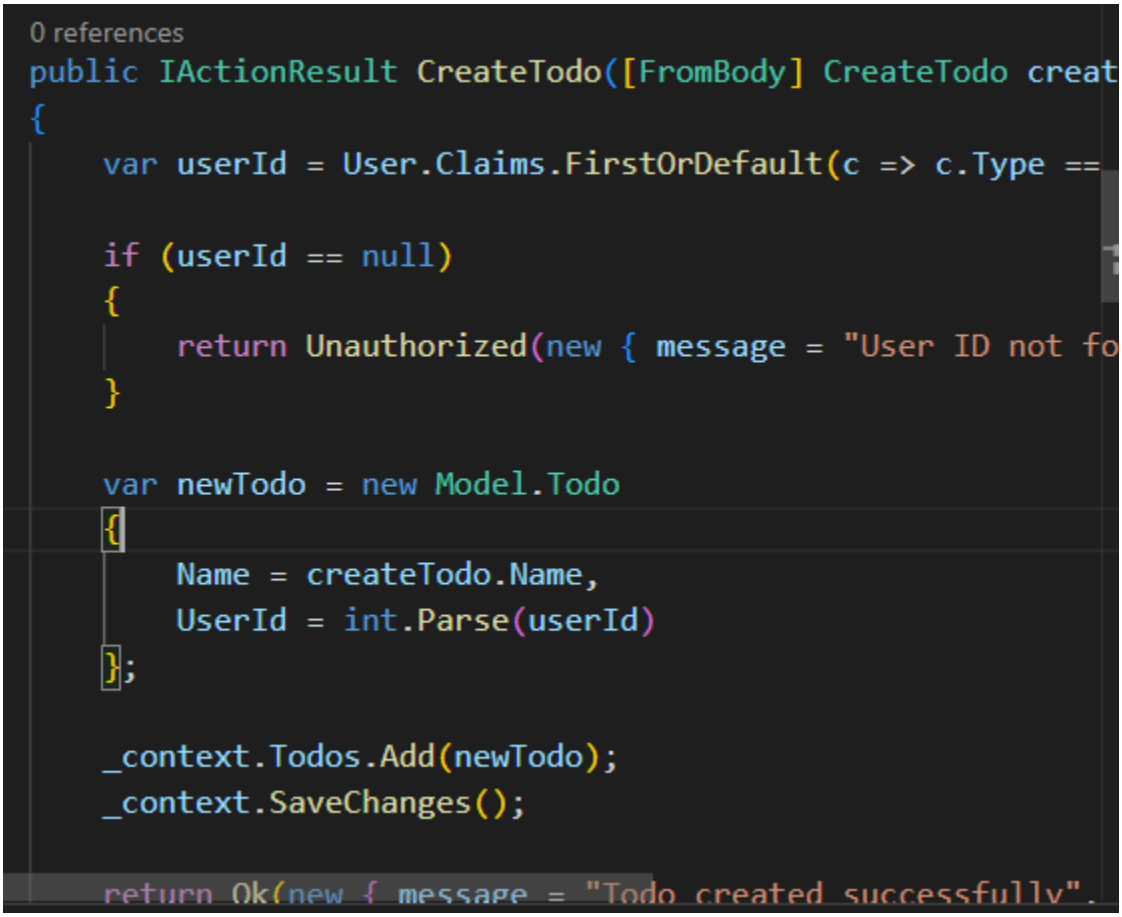
Controllers:

Todo: Όλα τα endpoints απαιτούν authenticated χρήστη μέσω token (Bearer JWT). Ο χρήστης αναγνωρίζεται από το claim NameIdentifier**.**

- **Get /todos :** Επιστρέφει όλα τα todos του authenticated χρήστη μαζί με τα tags τους**.**
  1. **Response 200 ok - επιστροφή του object**
  2. **Response 401 unauthorized - User ID not found in token**

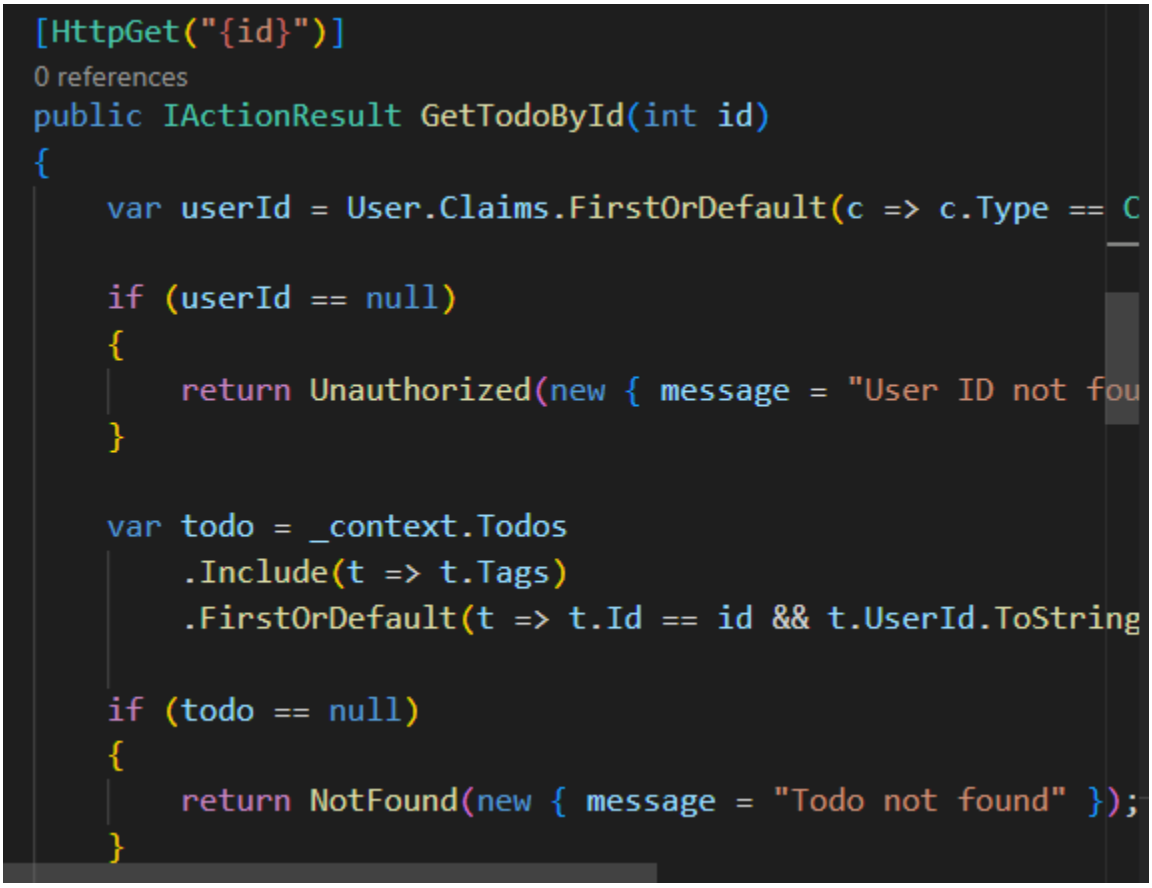
```
public IActionResult GetTodos()
{
    var userId = User.Claims.FirstOrDefault(c => c.Type == (

    if (userId == null)
    {
        return Unauthorized(new { message = "User ID not fou
    }

    var todos = _context.Todos
        .Include(t => t.Tags)
        .Where(t => t.UserId.ToString() == userId)
        .ToList();
    return Ok(todos);
}
```

- **Post /todos : Δημιουργεί νέο todo για τον authenticated χρήστη.**
- **Request Body:** name (string)
- **Response 200 OK - Todo created successfully**
  1. Επιστρέφει μήνυμα επιτυχίας
  2. Επιστρέφει το νέο todo object
- **Response 401 Unauthorized - User ID not found in token**
  1. Δεν βρέθηκε αναγνωριστικό χρήστη στο authentication token



```
0 references
public IActionResult CreateTodo([FromBody] CreateTodo creat
{
    var userId = User.Claims.FirstOrDefault(c => c.Type ==

    if (userId == null)
    {
        return Unauthorized(new { message = "User ID not fo
    }

    var newTodo = new Model.Todo
    {
        Name = createTodo.Name,
        UserId = int.Parse(userId)
    };

    _context.Todos.Add(newTodo);
    _context.SaveChanges();

    return Ok(new { message = "Todo created successfully".
```

- **Get /todos/{id} : Επιστρέφει συγκεκριμένο todo του authenticated χρήστη.**
- **Path Parameter :id (int) - Αναγνωριστικό todo**
- **Response 200 OK - Επιστροφή object**
    1. Επιστρέφει το todo μαζί με τα tags του
    2. Response 404 Not Found - Todo not found
    3. Το todo δεν υπάρχει ή δεν ανήκει στον χρήστη
- **Response 401 Unauthorized - User ID not found in token**
    1. Δεν βρέθηκε αναγνωριστικό χρήστη στο authentication token



```
[HttpGet("{id}")]
0 references
public IActionResult GetTodoById(int id)
{
    var userId = User.Claims.FirstOrDefault(c => c.Type == C

    if (userId == null)
    {
        return Unauthorized(new { message = "User ID not fou
    }

    var todo = _context.Todos
        .Include(t => t.Tags)
        .FirstOrDefault(t => t.Id == id && t.UserId.ToString

    if (todo == null)
    {
        return NotFound(new { message = "Todo not found" });
    }
```

- **Put /todos/{id} : Ενημερώνει υπάρχον todo του authenticated χρήστη.**
- **Path Parameter:** id (int)
- **Request Body:** name (string)
- **Response 200 OK - Todo updated successfully**
    1. Επιστρέφει μήνυμα επιτυχίας
    2. Επιστρέφει το ενημερωμένο todo object
- **Response 404 Not Found - Todo not found**
    1. Το todo δεν υπάρχει ή δεν ανήκει στον χρήστη
- **Response 401 Unauthorized - User ID not found in token**
    1. Δεν βρέθηκε αναγνωριστικό χρήστη στο authentication token

```
var userId = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier)?.Value

if (userId == null)
{
    return Unauthorized(new { message = "User ID not found in token" });
}

var todo = _context.Todos
    .Include(t => t.Tags)
    .FirstOrDefault(t => t.Id == id && t.UserId.ToString() == userId);

if (todo == null)
{
    return NotFound(new { message = "Todo not found" });
}
```

- **Delete /todos/{id} : Διαγράφει todo του authenticated χρήστη.**
  Path Parameter: id (int)
- **Response 200 OK - Todo deleted successfully**
  1. Επιστρέφει μήνυμα επιτυχίας
- **Response 404 Not Found - Todo not found**
  1. Το todo δεν υπάρχει ή δεν ανήκει στον χρήστη
- **Response 401 Unauthorized - User ID not found in token**
  1. Δεν βρέθηκε αναγνωριστικό χρήστη στο authentication token

```
if (userId == null)
{
    return Unauthorized(new { message = "User ID not found in token" });
}

var todo = _context.Todos
    .FirstOrDefault(t => t.Id == id && t.UserId.ToString() == userId);

if (todo == null)
{
    return NotFound(new { message = "Todo not found" });
}

_context.Todos.Remove(todo);
_context.SaveChanges();
```

TodoListItemController:

- **Get /todos/{id}/items/{itemId}: Επιστρέφει συγκεκριμένο todo item που ανήκει σε todo του authenticated χρήστη.**
  Path Parameters id (int) - Αναγνωριστικό Todo
  itemId (int) - Αναγνωριστικό Todo Item
- **Response 200 OK - Επιστροφή object**
  1. Επιστρέφει το συγκεκριμένο todo list item
  2. Περιλαμβάνει τα στοιχεία του item
- **Response 404 Not Found - Todo list item not found**
  1. Το item δεν υπάρχει
  2. Ή δεν ανήκει στο συγκεκριμένο todo
  3. Ή το todo δεν ανήκει στον χρήστη
- **Response 401 Unauthorized - User ID not found in token**
  1. Δεν βρέθηκε αναγνωριστικό χρήστη στο authentication token

```
ublic IActionResult GetTodoListItem(int id, int itemId)

    var userId = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier)?.Val

    if (userId == null)
    {
        return Unauthorized(new { message = "User ID not found in token" });
    }

    var todoListItem = _context.TodoListItems
        .Include(t => t.Todo)
        .FirstOrDefault(t => t.Id == itemId && t.TodoId == id && t.Todo!.UserId.ToString()
    if (todoListItem == null)
    {
        return NotFound(new { message = "Todo list item not found" });
    }
    return Ok(todoListItem);
```

- **Post /todos/{id}/items:** Δημιουργεί νέο todo item σε συγκεκριμένο todo του authenticated χρήστη.
- **Path Parameter** id (int) - Αναγνωριστικό Todo
- **Request Body** content (string)
- **Response 200 OK - Todo item created successfully**
  1. Επιστρέφει μήνυμα επιτυχίας
  2. Επιστρέφει το νέο todoListItem object
  3. Επιστρέφει το ενημερωμένο todo object
- **Response 404 Not Found - Todo not found**

1. Το todo δεν υπάρχει
2. Ή δεν ανήκει στον χρήστη

- **Response 401 Unauthorized - User ID not found in token**
  1. Δεν βρέθηκε αναγνωριστικό χρήστη στο authentication token

```
var userId = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier)?.Val

if (userId == null)
{
    return Unauthorized(new { message = "User ID not found in token" });
}

var todo = _context.Todos.FirstOrDefault(t => t.Id == id && t.UserId.ToString() == use
if (todo == null)
{
    return NotFound(new { message = "Todo not found" });
}

var newTodoListItem = new Model.TodoListItem
{
    Content = createTodoListItem.Content,
    TodoId = id
```

- **Put /todos/{id}/items/{itemId}: Ενημερώνει υπάρχον todo item.**
- **Path Parameters**
  1. id (int) - Αναγνωριστικό Todo
  2. ItemId (int) - Αναγνωριστικό Todo Item
- **Request Body**
  1. content (string)
  2. isCompleted (boolean)
- **Response 200 OK - Todo list item updated successfully**
  1. Επιστρέφει μήνυμα επιτυχίας
  2. Επιστρέφει το ενημερωμένο todoListItem object
- **Response 404 Not Found - Todo list item not found**
  1. Το item δεν υπάρχει
  2. Ή δεν ανήκει στο todo του χρήστη
- **Response 401 Unauthorized - User ID not found in token**
  1. Δεν βρέθηκε αναγνωριστικό χρήστη στο authentication token

```
public IActionResult UpdateTodoListItem(int id, int itemId, [FromBody] TodoItemUpdate todo
{
    var userId = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier)?.Val

    if (userId == null)
    {
        return Unauthorized(new { message = "User ID not found in token" });
    }

    var todoListItem = _context.TodoListItems
        .Include(t => t.Todo)
        .FirstOrDefault(t => t.Id == itemId && t.TodoId == id && t.Todo!.UserId.ToString()

    if (todoListItem == null)
    {
        return NotFound(new { message = "Todo list item not found" });
```

- **Delete /todos/{id}/items/{itemId}: Διαγράφει todo item από συγκεκριμένο todo.**
- **Path Parameters**
    1. id (int) - Αναγνωριστικό Todo
    2. itemId (int) - Αναγνωριστικό Todo Item
- **Response 200 OK - Todo item deleted successfully**
    1. Επιστρέφει μήνυμα επιτυχίας
- **Response 404 Not Found - Todo item not found**
    1. Το item δεν υπάρχει
    2. Ή δεν ανήκει στο todo του χρήστη
- **Response 401 Unauthorized - User ID not found in token**
    1. Δεν βρέθηκε αναγνωριστικό χρήστη στο authentication token

```
var userId = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier)?.Va

if (userId == null)
{
    return Unauthorized(new { message = "User ID not found in token" });
}

var todoListItem = _context.TodoListItems
    .Include(t => t.Todo)
    .FirstOrDefault(t => t.Id == itemId && t.TodoId == id && t.Todo!.UserId.ToString(

if (todoListItem == null)
{
    return NotFound(new { message = "Todo item not found" });
}

_context.TodoListItems.Remove(todoListItem);
```

UserController:

- **Post /auth/signup: Δημιουργεί νέο χρήστη στο σύστημα.**
- **Request Body**
  1. username (string)
  2. email (string)
  3. password (string)
- **Response 200 OK - User created successfully**
  1. Δημιουργείται νέος χρήστης
  2. Ο κωδικός αποθηκεύεται hashed με salt
- **Response 400 Bad Request - Username already exists**
  1. Υπάρχει ήδη χρήστης με τα ίδια στοιχεία

```
var user=_context.Users.FirstOrDefault(u=>u.Username==model.Username && u.Email==model
if(user==null)
{
    var salt = Password.GenerateSalt();
    var newUser=new User
    {
        Username=model.Username,
        Email=model.Email,
        Salt=salt,
        Hash=Password.HashPassword(model.Password,salt)
    };
    _context.Users.Add(newUser);
    _context.SaveChanges();
    return Ok(new {message="User created successfully"});
}
else
{
    return BadRequest(new {message="Username already exists"});
```

- **Post /auth/login: Πραγματοποιεί login και επιστρέφει JWT token.**
- **Request Body**
  1. username (string)
  2. password (string)
- **Response 200 OK - Επιστροφή JWT token**
  1. Επιστρέφεται token για χρήση σε επόμενα requests
  2. Το token χρησιμοποιείται ως Bearer Authentication
  3. Παράδειγμα:

```
{
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

- **Response 401 Unauthorized - Invalid username**
    1. Δεν βρέθηκε χρήστης με το συγκεκριμένο username
- **Response 401 Unauthorized - Invalid password**
    1. Ο κωδικός είναι λανθασμένος

```
[HttpPost("login")]
0 references
public IActionResult Login([FromBody] LoginModel model)
{
    var user = _context.Users.FirstOrDefault(u => u.Username == model.Username);
    if (user == null)
    {
        return Unauthorized(new {message="Invalid username"});
    }

    var hashedPassword = Password.HashPassword(model.Password, user.Salt);
    if (hashedPassword != user.Hash)
    {
            return Unauthorized(new {message="Invalid password"});
    }

    var token = _userService.CreateTokenForUser(user);
    return Ok(new { token });
}
```