UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

PIXELFORGE
Aplicație Web pentru distribuirea digitală a jocurilor video

Absolvent

Rădoi Dragoș-Cosmin

Coordonator științific

Conf. Univ. Dr. Marius Iulian Mihăilescu

București, iunie 2025

Rezumat

Aplicația Web PixelForge are obiectivul de a promova imbunătățirea interacțiunilor dintre dezvoltatorii de jocuri PC si pasionații de jocuri. Platforma le permite publisher-ilor să adauge, actualizeze sau să șteargă jocuri video din magazinul online si utilizatorilor să cumpere jocuri si să trimită mesaje despre probleme tehnice sau sugestii de funcționalități publisher-ilor.

O contribuție cheie a acestui proiect este distribuția digitală a jocurilor si deținerea acestora. Funcționalitatea de deținere a jocurilor este implementată in așa fel încât, dacă publisher-ul ar fi să elimine un joc din magazin sau să îl actualizeze, utilizatorul o să păstreze jocul pentru totdeauna în toate versiunile sale existente, rămânând accesibil în mod permanent. O altă funcționalitate importantă este sistemul de recenzii în care utilizatorii care dețin un joc pot crea recenzii care pot primi like sau dislike.

Aplicația folosește arhitectura Model-View-Controller, baze de date care se ocupă de conținut și un sistem de autentificare bazat pe roluri. Cu toate aceste funcționalități, am reușit să cream un site web funcțional cu un magazin online, librărie de jocuri personală, recenzii făcute de utilizatori, un sistem de suport bazat pe mesaje și o gestiune a versiunilor jocurilor.

Abstract

The Web Application PixelForge has the objective of promoting better interactions between PC gaming developers and passionate gamers. The platform allows publishers to add, update or delete video games from the store and the users to buy games and send messages about technical issues or feature suggestions to the publishers.

One of the key contributions of this project in the distribution of digital games is game ownership. The game ownership feature is handled in such a way that, if a publisher were to remove any published game from the store or update the game, the user will still keep the game forever, with all its existing versions, as it remains permanently accessible. Another important feature is the review system in which the users who own the games can create reviews that can be liked or disliked.

The application uses the Model-View-Controller architecture, databases for handling the content and a role-based authentication system. With all these features, we have managed to create a fully functional web app with an online store, personal game library, user reviews, a message-based support system and version management for the games.

Cuprins

Capitolul 1

Introducere

1.1 Tipul lucrării și subdomeniul

Lucrarea aceasta este de tip aplicație web practică si aparține domeniului de dezvoltare a aplicațiilor web moderne. Mai bine zis, lucrarea se concentrează pe realizarea unei platforme web de distribuire digitală a jocurilor video, folosind tehnologia ASP.NET Core si arhitectura Model-View-Controller. Proiectul implică și lucrul cu baze de date SQL, gestionarea și descărcarea fișierelor si un sistem de autentificare si autorizare.

1.2 Prezentarea generală a temei

PixelForge este o aplicație web care oferă o platformă comună pentru două tipuri de utilizatori: publisheri si utilizatori. Publisherii pot publica și actualiza jocuri într-un magazin online in timp ce utilizatorii obișnuiți pot cumpăra și descărca jocuri si pot scrie recenzii. Pe lângă aceste funcționalități, aplicația include și alte funcții precum scrierea unor postări legate de probleme ale unui joc adresate publisherului, păstrarea în permanență a jocurilor cumpărate, chiar și în cazul în care acestea au fost șterse sau actualizate și un sistem de selectare a versiunii jocului. Obiectivul aplicației este să ofere o experiență cât mai plăcută utilizatorilor și pasionaților de jocuri pe PC, inspirată din platforme precum Steam sau GOG.

1.3 Motivație

Motivația principală pentru această temă a fost, în primul rând, pasiunea mea pentru jocurile video pe PC și, în al doilea rând, interesul meu fată de modul în care acestea sunt distribuite și întreținute în mediul digital. Mi-am dorit să creez o aplicație care, nu doar că este funcțională, dar si prioretizează distribuția digitală a jocurilor și păstrarea drepturilor de proprietar. În același timp, am vrut să valorific cunoștințele acumulate de-a lungul facultății, de la autentificarea cu roluri și gestionarea acestora, până la arhitectura MVC si interacțiunea web in cadrul unui proiect cu aplicabilitate practica.

1.4 Contribuția proprie

Proiectul a fost realizat în întregime pe baza cunoștințelor acumulate în timpul studiilor,

precum și documentației oficiale și materiale de specialitate. Am dezvoltat atât interfața utilizatorului și a publisherului, cât și logica aplicației, acoperind funcționalități esențiale precum: adăugarea și editarea jocurilor de către publisheri, sistemul de cumpărare, biblioteca permanentă pentru utilizatori, recenziile cu sistem de like/dislike, suportul prin mesaje și gestiunea versiunilor. De asemenea, am avut în vedere protejarea datelor și asigurarea accesului permanent la jocurile cumpărate, inclusiv în cazul în care acestea sunt eliminate din magazin.

1.5 Structura lucrării

Capitolul II introduce noțiunile teoretice și tehnologiile esențiale utilizate în proiect, precum arhitectura Model-View-Controller, ASP.NET Core, Entity Framework Core și sistemul de autentificare bazat pe roluri.

Capitolul III reprezintă partea centrală a lucrării și conține descrierea completă a aplicației dezvoltate. Sunt detaliate etapele de proiectare, componentele arhitecturale și implementarea principalelor funcționalități: autentificarea, publicarea jocurilor, cumpărarea, recenziile, sistemul de suport și gestiunea versiunilor.

Capitolul IV oferă o evaluare finală a proiectului, evidențiind rezultatele obținute, aspectele ce pot fi îmbunătățite și posibile direcții de dezvoltare ulterioară.

Capitolul 2

Preliminarii

2.1 Noțiuni tehnologice relevante

Aplicația web PixelForge a fost dezvoltată folosind ASP.NET Core, un framework modern și open-source dezvoltat de Microsoft pentru proiectarea de aplicații web. Arhitectura utilizată este Model-View-Controller (MVC) care separă aplicația în trei părți: Model, care reprezintă informațiile interne, View, care este interfata cu informații și care primește date de la utilizator și Controller, care leagă împreună Model și View.

Pentru lucrul cu baze de date, am folosit Entity Framework Core, un Object-Relational Mapper care permite asocierea între clasele definite în C# și tabelele din baza de date relațională, fără a scrie direct cod SQL. EF Core oferă și suport pentru migrații automate, relații între entități și validări de date. Baza de date utilizată este Microsoft SQL Server, gestionată în mod automat prin migrațiile generate din cod.

Pentru autentificare și autorizare, am folosit ASP.NET Core Identity, care gestionează utilizatorii, parolele, rolurile și autentificarea pe baza de cookie-uri. În cadrul aplicației,

utilizatorii sunt împărțiți în două categorii: utilizatori obișnuiți și publisheri, fiecare cu drepturi și acces diferit în platformă. Această diferențiere este esențială experienței pe care încearcă să o ofere platforma, deoarece publisherii au posibilitatea de a adăuga și edita jocuri, în timp ce utilizatorii pot doar să vizualizeze, să cumpere, să descarce și să scrie recenzi.

2.2 Stadiul actual al domeniului

Distribuția digitală a jocurilor a devenit în ultimii ani standardul în industrie. Platforme precum Steam, Epic Games Store sau GOG oferă dezvoltatorilor de jocuri o modalitate prin care pot publica și vinde produse către un public larg, fără a fi necesară o lansare fizică. Aceste platforme includ sisteme de plată, biblioteci digitale personale, recenzii, forumuri și actualizări automate. Totuși, accesul dezvoltatorilor independenți la aceste platforme este adesea limitat de costuri sau cerințe contractuale.

Platformele comerciale mari oferă funcționalități avansate, dar sunt închise și controlate de entități centrale. Acest lucru poate duce la o lipsă de transparență în ceea ce privește gestionarea conținutului digital și accesul pe termen lung la jocurile achiziționate. Deși platforme precum Steam sau GOG permit în general păstrarea accesului la jocurile cumpărate chiar și după ce acestea sunt eliminate din magazin, există și cazuri, precum platforma Ubisoft Connect, în care publisherii pot restricționa sau revoca complet accesul la jocuri sau DLC-uri, chiar dacă utilizatorul le-a cumpărat. În contrast, aplicația dezvoltată în acest proiect oferă o soluție simplă: odată cumpărat, un joc rămâne permanent în biblioteca utilizatorului.

2.3 Obiectivele lucrării

Obiectivul principal al lucrării este dezvoltarea unei aplicații web funcționale care simulează un magazin digital de jocuri și se concentrează pe experiența utilizatorilor. Aplicația trebuie să permită publicarea și actualizarea jocurilor de către publisheri, cumpărarea și descărcarea acestora de către utilizatori, precum și păstrarea lor permanentă în biblioteca personală, indiferent de acțiunile publisherului.

Pe partea de suport, aplicația urmărește să faciliteze comunicarea între utilizatori și publisheri, printr-un sistem simplu de postări. Astfel, utilizatorii pot scrie despre probleme sau pot veni cu idei de îmbunătățiri, iar publisherii pot răspunde la fiecare postare în parte.

Un alt obiectiv este implementarea sistemului de versiuni, care permite publisherilor să încarce versiuni noi ale unui joc fără a pierde versiunile mai vechi. Utilizatorii care au cumpărat jocul pot alege oricând ce versiune să descarce, având control total asupra jocurilor deținute. Această funcționalitate este adresată unei probleme frecvente a platformelor reale, în care actualizările înlocuiesc versiunile anterioare în mod definitiv.

Capitolul 3

Contribuția Propriu Zisă

3.1 Autentificare și roluri:

Pentru autentificare, am utilizat ASP.NET Core Identity astfel încât utilizatorul, dacă dorește să acceseze serviciile aplicației, trebuie să creeze un cont si să se conecteze la acesta. Utilizatorul trebuie să își înregistreze email-ul și să selecteze un nume, unul dintre cele două roluri (User/Publisher) și o parolă care îndeplineste anumite cerințe de securitate (numărul de caractere, litere mari si mici si caractere speciale).

```
public class InputModel
{
    [Required]
    [DataType(DataType.Text)]
    [Display(Name = "First Name")]
    4 references
    public string FirstName { get; set; }

    [Required]
    [DataType(DataType.Text)]
    [Display(Name = "Last Name")]
    5 references
    public string LastName { get; set; }


    /// <summary>
    ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
    ///     directly from your code. This API may change or be removed in future releases.
    /// </summary>
    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    6 references
    public string Email { get; set; }

    [Required]
    [Display(Name = "Role")]
    [RegularExpression("User|Publisher", ErrorMessage = "Invalid role selected.")]
    7 references
    public string Role { get; set; }


    /// <summary>
    ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
    ///     directly from your code. This API may change or be removed in future releases.
    /// </summary>
    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    4 references
    public string Password { get; set; }

    /// <summary>
    ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
    ///     directly from your code. This API may change or be removed in future releases.
    /// </summary>
    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
    3 references
    public string ConfirmPassword { get; set; }
}
```

Figura 3.1 – Clasa utilizată pentru validarea datelor de înregistrare

Rolurile, în funcție de selectarea la autentificare, au acces la diverse funcții și interfete. Utilizatorul are acces la magazinul online, librăria sa de jocuri detinute și la sistemul de postare a recenzilor, în timp ce Publisher-ul are acces la catalogul său de jocuri incărcate de acesta, magazinul online și la sistemul de interacțiune cu recenziile.

3.2 Magazinul online

Pagina principală a aplicației prezintă o listă ordonată cu toate jocurile disponibile. Pentru fiecare joc sunt afișate genul jocului, vârsta minimă recomandata, prețul, numele Publisher-ului, un link către pagina de detalii a jocului si opțiunea de a adăuga jocul in coșul de cumpărături. Utilizatorul are și opțiunea de a căuta un anumit joc, de a sorta in funcție de categoriile menționate mai sus și de a filtra pagina pentru a afișa anumite jocuri in funcție de parametri precizați.

```
public async Task<IActionResult> Store(string searchString, string sortOrder, string genreFilter, double? minPrice, double? maxPrice, string ageFilter)
{
    var game = await _context.Games
        .Include(g => g.Publisher)
        .Where(g => !g.IsDeleted)
        .ToListAsync();

    if (!string.IsNullOrEmpty(searchString))
    {
        game = game.Where(g => g.Title.Contains(searchString, StringComparison.OrdinalIgnoreCase)).ToList();
    }

    if (!string.IsNullOrEmpty(genreFilter))
    {
        game = game.Where(g => g.Genre.Equals(genreFilter, StringComparison.OrdinalIgnoreCase)).ToList();
    }

    if (minPrice.HasValue)
    {
        game = game.Where(g => g.Price >= minPrice.Value).ToList();
    }
    if (maxPrice.HasValue)
    {
        game = game.Where(g => g.Price <= maxPrice.Value).ToList();
    }

    if (!string.IsNullOrEmpty(ageFilter))
    {
        game = game.Where(g => g.AgeRating.ToString() == ageFilter).ToList();
    }


    ViewData["NameSortParam"] = sortOrder == "name_asc" ? "name_desc" : "name_asc";
    ViewData["PriceSortParam"] = sortOrder == "price_asc" ? "price_desc" : "price_asc";
    ViewData["PublisherSortParam"] = sortOrder == "publisher_asc" ? "publisher_desc" : "publisher_asc";
    ViewData["GenreSortParam"] = sortOrder == "genre_asc" ? "genre_desc" : "genre_asc";
    ViewData["AgeSortParam"] = sortOrder == "age_asc" ? "age_desc" : "age_asc";
    ViewData["PopularitySortParam"] = sortOrder == "popularity_asc" ? "popularity_desc" : "popularity_asc";
```

Figura 3.2 - Sistemul de filtrare al jocurilor

```
var gameWithCounts = game.Select(g => new
{
    Game = g,
    OwnersCount = _context.UserGames.Count(ug => ug.GameId == g.Id)
}).ToList();

switch (sortOrder)
{
    case "name_desc":
        gameWithCounts = gameWithCounts.OrderByDescending(g => g.Game.Title).ToList();
        break;
    case "name_asc":
        gameWithCounts = gameWithCounts.OrderBy(g => g.Game.Title).ToList();
        break;
    case "price_desc":
        gameWithCounts = gameWithCounts.OrderByDescending(g => g.Game.Price).ToList();
        break;
    case "price_asc":
        gameWithCounts = gameWithCounts.OrderBy(g => g.Game.Price).ToList();
        break;
    case "publisher_desc":
        gameWithCounts = gameWithCounts.OrderByDescending(g => g.Game.Publisher.FirstName + " " + g.Game.Publisher.SecondName).ToList();
        break;
    case "publisher_asc":
        gameWithCounts = gameWithCounts.OrderBy(g => g.Game.Publisher.FirstName + " " + g.Game.Publisher.SecondName).ToList();
        break;
    case "genre_desc":
        gameWithCounts = gameWithCounts.OrderByDescending(g => g.Game.Genre).ToList();
        break;
    case "genre_asc":
        gameWithCounts = gameWithCounts.OrderBy(g => g.Game.Genre).ToList();
        break;
    case "age_desc":
        gameWithCounts = gameWithCounts.OrderByDescending(g => g.Game.AgeRating).ToList();
        break;
    case "age_asc":
        gameWithCounts = gameWithCounts.OrderBy(g => g.Game.AgeRating).ToList();
        break;
    case "popularity_desc":
        gameWithCounts = gameWithCounts.OrderByDescending(g => g.OwnersCount).ToList();
        break;
    case "popularity_asc":
        gameWithCounts = gameWithCounts.OrderBy(g => g.OwnersCount).ToList();
        break;
    default:
        gameWithCounts = gameWithCounts.OrderBy(g => g.Game.Title).ToList();
        break;
}

game = gameWithCounts.Select(g => g.Game).ToList();


return View(game);
```

Figura 3.3 - Sistemul de filtrare al jocurilor

3.3 Coșul de cumpărături și biblioteca utilizatorului

Coșul de cumpărături este implementat folosind sesiunea (HttpContext.Session) și nu este salvat în baza de date. Utilizatorii pot adăuga jocuri în coș, iar la finalizarea comenzii, pentru fiecare joc selectat, se adaugă o relație UserGame în baza de date. Astfel, se marchează achiziția, iar utilizatorul obține accesul permanent la jocul respectiv. Pentru a preveni cumpărarea multiplă a unui joc de către același utilizator, aplicația verifică de fiecare dată dacă utilizatorul curent deține jocul înainte de a-i permite descărcarea sau scrierea unei recenzii. Această verificare este realizată printr-o interogare directă în UserGame și controlează atât interfața, cât și acțiunile disponibile în backend.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Buy()
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (userId == null)
        return Challenge();

    var cart = HttpContext.Session.GetObjectFromJson<List<CartItem>>("Cart");
    if (cart == null || !cart.Any())
    {
        TempData["Error"] = "Your cart is empty.";
        return RedirectToAction("Index");
    }

    var ownedGameIds = await _context.UserGames
        .Where(ug => ug.UserId == userId)
        .Select(ug => ug.GameId)
        .ToListAsync();

    foreach (var item in cart)
    {
        if (!ownedGameIds.Contains(item.GameId))
        {
            _context.UserGames.Add(new UserGame
            {
                UserId = userId,
                GameId = item.GameId
            });
        }
    }

    await _context.SaveChangesAsync();

    HttpContext.Session.Remove("Cart");

    TempData["Success"] = "Purchase successful! Your games are now in your library.";
    return RedirectToAction("Library", "Game");
}
```

Figura 3.4 - Metoda prin care cumpărăm jocurile

3.4 Sistemul de recenzii si de rating

Recenziile reprezintă una dintre cele mai importante funcționalități ale aplicației PixelForge. Aceastea oferă utilizatorilor posibilitatea de a își exprima opiniile despre jocurile pe care le dețin. Existenta recenziilor contribuie la răsplatirea publisher-ilor care distribuie jocuri de calitate și aduce critici constructive celor cu jocuri de o calitate mai slabă.

Pentru a scrie o recenzie, utilizatorul trebuie să acceseze librăria proprie și să selecteze opțiunea “Review” din dreapta jocului. Fiecare utilizator poate scrie o singură recenzie pentru fiecare joc detinut in parte si sa îi acorde o notă de la 1 la 5.

După validare, recenzia este salvată în baza de date a aplicației. Acest proces asigură integritatea sistemului și previne spamul sau publicarea mai multor recenzii care ar putea distorsiona evaluarea generală.

Odată ce a fost publicată, recenzia va fi afisată pe pagina de detalii a jocului și va fi vizibilă tututor utilizatorilor, indiferent dacă acestia dețin sau nu jocul. Astfel, utilizatorii care incă nu s-au decis să cumpere jocul pot lua în considerare scorul total al jocului, calculat ca media aritmetică a tuturor recenziilor.

```
[HttpGet]
0 references
public async Task<IActionResult> Create(int gameId)
{
    var user = await _userManager.GetUserAsync(User);

    var alreadyReviewed = await _context.Reviews
        .AnyAsync(r => r.GameId == gameId && r.UserId == user.Id);

    if (alreadyReviewed)
    {
        TempData["Error"] = "You already reviewed this game.";
        return RedirectToAction("Library", "Game");
    }

    var game = await _context.Games.FirstOrDefaultAsync(g => g.Id == gameId);
    if (game == null)
        return NotFound();

    ViewBag.GameTitle = game.Title;
    ViewBag.GameId = gameId;
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create(int gameId, int rating, string comment)
{
    var user = await _userManager.GetUserAsync(User);

    var alreadyReviewed = await _context.Reviews
        .AnyAsync(r => r.GameId == gameId && r.UserId == user.Id);

    if (alreadyReviewed)
    {
        TempData["Error"] = "You already reviewed this game.";
        return RedirectToAction("Library", "Game");
    }

    var review = new Review
    {
        GameId = gameId,
        UserId = user.Id,
        Rating = rating,
        Comment = comment
    };

    _context.Reviews.Add(review);
    await _context.SaveChangesAsync();

    TempData["Success"] = "Review submitted successfully!";
    return RedirectToAction("Library", "Game");
}
```

Figura 3.5 - Controller-ul pentru Crearea si Validarea Recenziilor

Pentru a permite evaluarea recenziilor publicate, aplicația PixelForge integrează un sistem de reacții simple, like și dislike. Acesta ajută la identificarea recenziilor utile și a celor nepotrivite. Reacțiile sunt disponibile în pagina de detalii a fiecărui joc, lângă fiecare recenzie, și pot fi acordate sau anulate de orice utilizator autentificat, indiferent dacă deține jocul sau nu.

```
if (vote != null)
{
    if (vote.IsLike == isLike)
    {
        _context.ReviewVotes.Remove(vote);
    }
    else
    {
        vote.IsLike = isLike;
        _context.ReviewVotes.Update(vote);
    }
}
else
{
    var newVote = new ReviewVote
    {
        UserId = userId,
        ReviewId = reviewId,
        IsLike = isLike
    };
    _context.ReviewVotes.Add(newVote);
}
```

Figura 3.6 - Logica de gestionare a reacțiilor de like/dislike pentru recenzii

3.5 Suport și comunicare

Pentru a încuraja comunicarea directă între utilizatori și publisheri, aplicația oferă un sistem de suport bazat pe mesaje publice către publisher. Fiecare joc publicat are o pagină dedicată de suport, accesibilă din pagina sa de detalii. Astfel:

- Toți utilizatorii pot să vadă mesajele de suport existente;
- Doar utilizatorii care au cumpărat jocul pot să trimită mesaje noi;
- Publisherul poate să răspundă fiecărui mesaj, public, printr-un câmp de răspuns.

Sistemul este implementat în Controller-ul pentru suport, unde se diferențiază clar drepturile în funcție de rol:

```
if (User.IsInRole("Publisher") && game.PublisherId == currentUserId)
{
    messages = await _context.SupportMessages
        .Include(m => m.Author)
        .Where(m => m.GameId == gameId)
        .OrderByDescending(m => m.CreatedAt)
        .ToListAsync();
}
else if (User.IsInRole("User"))
{
    messages = await _context.SupportMessages
        .Include(m => m.Author)
        .Where(m => m.GameId == gameId && m.AuthorId == currentUserId)
        .OrderByDescending(m => m.CreatedAt)
        .ToListAsync();
}
else
{
    return Unauthorized();
}
```

Figura 3.7 - Filtrarea mesajelor în funcție de rolul utilizatorului

Răspunsul Utilizatorului este salvat în tabelul asociat entității SupportMessage:

```
var userId = _userManager.GetUserId(User);
var game = await _context.Games.FindAsync(message.GameId);

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

message.AuthorId = userId;
message.CreatedAt = DateTime.Now;
_context.SupportMessages.Add(message);
await _context.SaveChangesAsync();
```

Figura 3.8 - Trimiterea și salvarea mesajului

Iar răspunsul Publisher-ului este salvat în același tabel:

```
var message = await _context.SupportMessages
    .Include(m => m.Game)
    .Include(m => m.Author)
    .FirstOrDefaultAsync(m => m.Id == id);

var currentUserId = _userManager.GetUserId(User);

if (message == null || message.Game.PublisherId != currentUserId)
    return Unauthorized();

message.PublisherReply = reply;
message.ReplyDate = DateTime.Now;

await _context.SaveChangesAsync();
```

Figura 3.9 - Salvarea răspunsului într-un mesaj de suport existent

Acesta funcționează similar unui forum public de discuții asociat jocului, dar fără răspunsuri ierarhizate. Astfel, simplitatea sistemului facilitează interacțiunea informală și eficientă intre utilizator si publisher prin care aceștia pot discuta despre probleme mai mici sau mai mari ale jocului respectiv, fără nevoia unei infrastructuri mai complexe.

Avantaje:

- Centralizare: toate mesajele sunt grupate într-o singură pagină dedicată jocului respectiv.
- Vizibilitate: toți utilizatorii văd întrebările și răspunsurile.
- Control editorial: doar publisherul respectiv poate sa răspundă.

3.6 Catalogul online

Catalogul online este platforma unde Publisher-ul poate să încarce jocurile pe care acesta dorește să le expună în magazinul online al aplicației și de unde poate să actualizeze numele, prețul, genul, vârsta recomandată și conținutul jocului respectiv. O altă funcționalitate pe care o are la indemână este ștergerea jocului. Odată șters, jocul este eliminat din magazinul online și din catalogul Publisher-ului, dar ramâne în continuare în librăria utilizatorilor care au cumpărat jocul respectiv, cu toate versiunile sale cu tot.

Pentru a încărca conținutul jocului, Publisher-ul trebuie să atașeze un fișier de tip zip, rar, exe sau msi și în cazul în care este valid, fișierul e salvat in folderul ”uploads”, iar calea e păstrata în baza de date a aplicației.

```
var allowedExtensions = new[] { ".zip", ".rar", ".exe", ".msi" };
var errors = new List<string>();

if (gameFile == null || gameFile.Length == 0)
{
    errors.Add("Please upload a game file.");
}
else
{
    var extension = Path.GetExtension(gameFile.FileName).ToLowerInvariant();
    if (!allowedExtensions.Contains(extension))
    {
        errors.Add("Only .zip, .rar, .exe, and .msi files are allowed.");
    }
}

if (!ModelState.IsValid || errors.Count > 0)
{
    ViewBag.Errors = errors;
    return View(game);
}

var uploadsPath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "uploads");
if (!Directory.Exists(uploadsPath))
{
    Directory.CreateDirectory(uploadsPath);
}

var uniqueFileName = $"{Guid.NewGuid()}_{Path.GetFileName(gameFile.FileName)}";
var fullPath = Path.Combine(uploadsPath, uniqueFileName);

using (var stream = new FileStream(fullPath, FileMode.Create))
{
    await gameFile.CopyToAsync(stream);
}

game.GameFilePath = $"/uploads/{uniqueFileName}";

game.PublisherId = _userManager.GetUserId(User);
_context.Games.Add(game);
await _context.SaveChangesAsync();

return RedirectToAction("Index");
```

Figura 3.10 - Încărcarea unui joc in aplicație

Editarea unui jocul, pe de altă parte, constă în actualizarea datelor sale existente(titlul, genul, prețul si vârsta recomandata) și îi permite Publisher-ului să încarce o versiune nouă a jocului. În cazul în care fișierul îndeplinește condițiile enumerate mai sus, versiunea nouă este încărcată printr-un fișier suplimentar și primește un număr de versiune, iar fișierul e salvat tot în "uploads", împreună cu versiunea originală a jocului.

```
var existingGame = await _context.Games
    .Include(g => g.Versions)
    .FirstOrDefaultAsync(g => g.Id == id);

if (existingGame == null || existingGame.PublisherId != game.PublisherId)
    return Unauthorized();

existingGame.Title = game.Title;
existingGame.Price = game.Price;
existingGame.Genre = game.Genre;
existingGame.AgeRating = game.AgeRating;

if (newVersionFile != null && newVersionFile.Length > 0 && !string.IsNullOrWhiteSpace(versionNumber))
{
    var allowedExtensions = new[] { ".zip", ".rar", ".exe", ".msi" };
    var extension = Path.GetExtension(newVersionFile.FileName).ToLowerInvariant();

    if (!allowedExtensions.Contains(extension))
    {
        ModelState.AddModelError("", "File type not allowed.");
        return View(game);
    }

    var uploadsPath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "uploads");
    if (!Directory.Exists(uploadsPath))
        Directory.CreateDirectory(uploadsPath);

    var uniqueFileName = $"{Guid.NewGuid()}_{Path.GetFileName(newVersionFile.FileName)}";
    var fullPath = Path.Combine(uploadsPath, uniqueFileName);

    using (var stream = new FileStream(fullPath, FileMode.Create))
    {
        await newVersionFile.CopyToAsync(stream);
    }

    var version = new GameVersion
    {
        GameId = existingGame.Id,
        VersionNumber = versionNumber,
        FilePath = $"/uploads/{uniqueFileName}",
        UploadDate = DateTime.Now
    };

    _context.GameVersions.Add(version);
```

Figura 3.11 - Editarea unui joc existent

Ștergerea unui joc este implementată printr-un mecanism de soft delete, pentru a evita pierderea jocului de către utilizatorii care dețin jocul respectiv. Odată ce Publisher-ul a șters jocul, aplicația nu elimină jocul din baza sa de date, ci setează proprietatea IsDeleted la valoarea True. Astfel, jocul este eliminat din catalogul Publisher-ului și din magazinul online, dar rămâne disponibil în librăria utilizatorului.

```
0 references
public async Task<IActionResult> Delete(int id)
{
    var game = await _context.Games.FirstOrDefaultAsync(x => x.Id == id);
    var userId = _userManager.GetUserId(User);

    if (game == null || game.PublisherId != userId)
        return Unauthorized();

    return View(game);
}

[HttpPost, ActionName("Delete")]
0 references
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var game = await _context.Games.FindAsync(id);
    if (game != null) {
        game.IsDeleted = true;
        _context.Games.Update(game);
        await _context.SaveChangesAsync();
    }
    return RedirectToAction("Index");
}
```

Figura 3.12 - Metoda prin care stergem un joc

3.7 Descărcarea jocurilor

Descărcarea jocurilor este una dintre cele mai importante funcționalități ale aplicației PixelForge. Aceasta a fost concepută astfel încât doar utilizatorii care dețin deja jocul în librărie pot să îl descarce, în toate versiunile sale disponibile.

Descărcarea este posibilă doar pentru utilizatorii autentificați care au înregistrată relația de achiziție pentru jocul respectiv. În controllerul Download, sistemul verifică:

- Dacă utilizatorul este logat;
- Dacă deține jocul respectiv;
- Dacă versiunea cerută există și aparține jocului.

```
public async Task<IActionResult> Download(int id)
{
    var userId = _userManager.GetUserId(User);
    var hasGame = await _context.UserGames.AnyAsync(ug => ug.UserId == userId && ug.GameId == id);

    if (!hasGame)
        return Unauthorized();

    var game = await _context.Games.FirstOrDefaultAsync(g => g.Id == id);

    if (game == null || string.IsNullOrEmpty(game.GameFilePath))
        return NotFound();

    var fullPath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", game.GameFilePath.TrimStart('/'));

    if (!System.IO.File.Exists(fullPath))
        return NotFound("File not found on server.");

    var fileName = Path.GetFileName(fullPath);
    var contentType = "application/octet-stream";

    var memory = new MemoryStream();
    using (var stream = new FileStream(fullPath, FileMode.Open))
    {
        await stream.CopyToAsync(memory);
    }
    memory.Position = 0;

    return File(memory, contentType, fileName);
}
```

Figura 3.13 - Controller-ul pentru descărcarea jocurilor

3.8 Versiunea jocurilor

Sistemul de gestionare a versiunilor unui joc este una dintre funcționalitățile esențiale ale aplicației PixelForge, aceasta stând la baza proiectului. Spre deosebire de unele platforme online care pur si simplu nu permit accesarea unei versiuni mai vechi, PixelForge permite utilizatorilor să acceseze orice versiune mai veche a jocului deținut de aceștia.

Fiecare joc poate avea mai multe versiuni, fiind reprezentate de entitatea GameVersion, care conține fișierul asociat, numărul versiunii(VersionNumber) și data încărcării. Utilizatorii care dețin jocul pot selecta versiunea dorită și o pot descărca în orice moment. Astfel, versiunea veche a jocului nu este înlocuită de versiunea noua, ci este stocata in folderul “uploads” împreună cu toate celelalte jocuri si versiunile lor, iar calea lor este păstrată in baza de date a aplicației.

```
public async Task<IActionResult> DownloadVersion(string versionId, int gameId)
{
    var userId = _userManager.GetUserId(User);
    var ownsGame = await _context.UserGames.AnyAsync(ug => ug.UserId == userId && ug.GameId == gameId);
    if (!ownsGame)
        return Unauthorized();

    if (versionId == "legacy")
    {
        var game = await _context.Games.FirstOrDefaultAsync(g => g.Id == gameId);
        if (game == null || string.IsNullOrEmpty(game.GameFilePath))
            return NotFound();

        var legacyPath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", game.GameFilePath.TrimStart('/'));
        if (!System.IO.File.Exists(legacyPath))
            return NotFound("Legacy file not found.");

        var fileName = Path.GetFileName(legacyPath);
        var memory = new MemoryStream();
        using (var stream = new FileStream(legacyPath, FileMode.Open))
        {
            await stream.CopyToAsync(memory);
        }
        memory.Position = 0;

        return File(memory, "application/octet-stream", fileName);
    }

    if (!int.TryParse(versionId, out int id))
        return BadRequest("Invalid version ID.");

    var version = await _context.GameVersions
        .Include(v => v.Game)
        .FirstOrDefaultAsync(v => v.Id == id && v.GameId == gameId);

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

    var path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", version.FilePath.TrimStart('/'));
    if (!System.IO.File.Exists(path))
        return NotFound("Version file not found.");

    var versionFileName = Path.GetFileName(path);
    var versionMemory = new MemoryStream();
    using (var stream = new FileStream(path, FileMode.Open))
    {
        await stream.CopyToAsync(versionMemory);
    }
    versionMemory.Position = 0;

    return File(versionMemory, "application/octet-stream", versionFileName);
}
```

Figura 3.14 - Controller-ul pentru descărcarea unei versiuni specifice

3.9 Interfața Aplicației

Interfața aplicației a fost concepută pentru a fi funcțională, minimalistică și usor de navigat pentru utilizatori. S-au folosit elemente de HTML și CSS simple, precum și framework-uri externe precum Bootstrap, pentru a avea un UI cât mai acceptabil.

Există două interfețe principale:

- Interfața Utilizatorului: magazinul online, coșul de cumpărături, librăria personală, sistemul de recenzii și sistemul de suport;
- Interfața Publisherului: catalogul personal de jocuri, sistemul de suport.

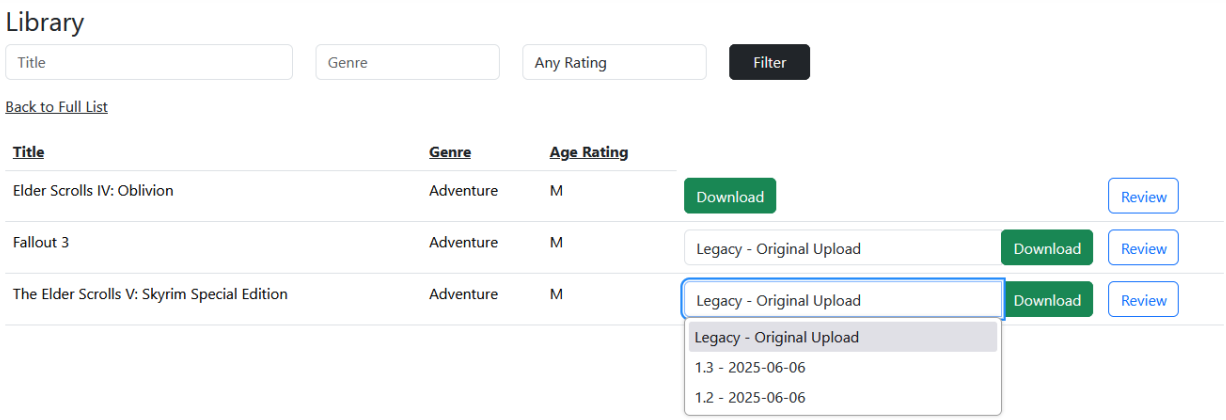

Figura 3.15 - Librăria

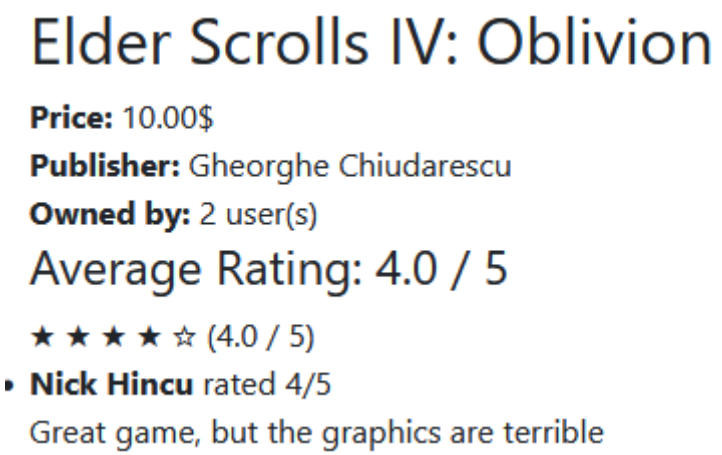

Figura 3.16 - Pagina de detalii

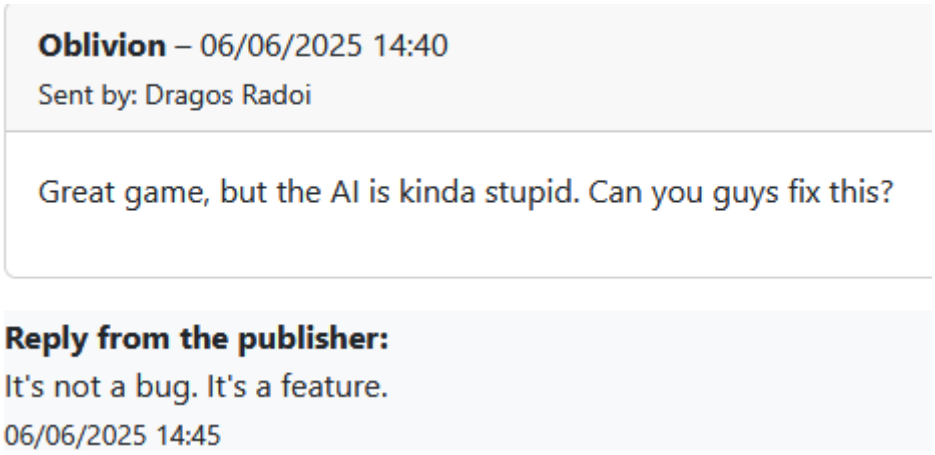

Figura 3.17 - Mesaj de la utilizator și răspunsul publisher-ului

Capitolul 4

Concluzia

Pixelforge reprezintă un proiect prin care am demonstrat că este posibil pentru un singur

student să creeze o aplicație web care are funcționalitățile unui magazin online de jocuri de PC. Întregul proces a fost posibil cu ajutorul tehnologiilor ca ASP.NET Core și Entity Framework și prin valorificarea cunoștințelor dobândite de-a lungul universității. Aplicația integrează funcționalități fundamentale unui magazin online precum înregistrarea și autentificarea unui cont de utilizator, publicarea, cumpărarea și descărcarea jocurilor video, scrierea și publicarea recenziilor pentru jocuri și un sistem de mesaje între utilizator și publisher.

De-a lungul dezvoltării aplicației au existat și funcționalități care nu au putut fi implementate în această etapă, însă în viitor pot fi adăugate sau extinse, cum ar fi:

- Implementarea unui sistem de plată online
- Îmbunatațirea interfeței aplicației, pentru o experientă mult mai placută pentru utilizator
- Optimizarea sistemului de gestionare a jocurilor, pentru a permite încărcarea jocurilor de dimensiuni mai mari(80-100GB)
- Implementarea unui sistem de prietenie si comunicare între utilizatori pentru a încuraja crearea unei comunități

În concluzie, aplicația PixelForge dovedește că este posibilă crearea unui magazin online care este în avantajul pasionaților de jocuri de PC și a creatorilor de jocuri, oferind un mediu transparent, flexibil și sigur pentru distribuția digitală.

Bibliografia

[1] Lock, Andrew. *ASP. NET core in Action*. Simon and Schuster, 2023.

[2] Freeman, Adam. *Pro ASP. NET Core 7*. Simon and Schuster, 2023.

[3] Yan, Xiaodi. *Web API Development with ASP. NET Core 8: Learn techniques, patterns, and tools for building high-performance, robust, and scalable web APIs*. Packt Publishing Ltd, 2024.

[4] ASP.NET Core Documentation, See also: https://learn.microsoft.com/en-us/aspnet/core/overview?view=aspnetcore-9.0

[5] SQL Server Management Studio Documentation, See also: https://learn.microsoft.com/en-us/ssms/sql-server-management-studio-ssms

[6] Entity Framework Core Documentation, See also: https://learn.microsoft.com/en-us/ef/core/

[7] Brind, Mike. *ASP. NET Core Razor Pages in Action*. Simon and Schuster, 2023.

[8] Mezei, Razvan Alexandru. *Introduction to the Development of Web Applications Using ASP. Net (Core) MVC*. Springer Nature, 2023.

[9] Jayaraman, Kumaresan Durvas, and Er Siddharth. "Harnessing the Power of Entity Framework Core for Scalable Database Solutions." *Journal of Quantum Science and Technology (JQST)* 2.1 (2025): 151-171.

[10] Bootstrap Team, Bootstrap v5.3 Documentation, See also: https://getbootstrap.com/docs/5.3/getting-started/introduction/

[11] Mushica, Florinë, and Agon Memeti. "A COMPREHENSIVE ANALYSIS OF ARCHITECTURAL PATTERNS IN ASP. NET CORE WEB APPLICATION DEVELOPMENT." *JNSM Journal of Natural Sciences and Mathematics of UT* 9.17-18 (2024): 207-218.

[12] Pekh, P., and B. Yankovsky. "Features of creating web applications using C# ASP. NET CORE MVC." *COMPUTER-INTEGRATED TECHNOLOGIES: EDUCATION, SCIENCE, PRODUCTION* 59 (2025): 253-257.