|  |  |  |
| --- | --- | --- |
| **МИНИСТЕРСТВО НА ОБРАЗОВАНИЕТО И НАУКАТА** | | |
| **ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО ЕЛЕКТРОТЕХНИКА И ЕЛЕКТРОНИКА**  бул. Пещерско шосе № 26  4002 гр. Пловдив, България  тел. 032 / 643-657  info-1690174@edu.mon.bg |  | **VOCATIONAL SCHOOL OF ELECTRICAL ENGINEERING AND ELECTRONICS**  26 Peshtersko chaussee blvd.,  4002 Plovdiv, Bulgaria  Phone: 032 / 643-657  info-1690174@edu.mon.bg |

**професия код 481030 „Приложен програмист“**

**специалност код 4810301 „Приложно програмиране“**

**Курсов Проект**

**Тема:Изграждане на приложение „Каталог за игри“**

**Разработили**: …………… **Ръководител-консултант**: …………

/Илиян Куршумов, /инж. Ваня Менгова/

Ангел Масарлиев,

Божидар Иванов/

Клас: 12а

e-mail: ilijan.kurshumov@gmail.com

Пловдив

2025 година

# Съдържание

# Увод

Проектът представлява платформа за управление на каталог за игри, създадена с цел да бъде удобна и функционална за потребителите. Основната цел е да се осигури интуитивен и ефективен начин за работа с информация за игри.

Приложението предоставя възможност на потребителите да разглеждат наличните игри, да добавят нови записи, да редактират информацията за вече съществуващи игри, както и да изтриват записи, ако е необходимо. В допълнение към тези основни функционалности е добавена опция за бързо и лесно търсене чрез **търсачка (search bar)**, която улеснява намирането на конкретни игри в каталога.

Тази платформа е разработена, за да предложи както удобство за крайния потребител, така и възможност за разширяване с допълнителни функционалности в бъдеще.

# Глава 1 - Технологии и Концепции

## 1.1. Asp.Net Core

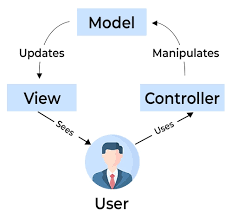
**ASP.NET Core** е модерна, платформа-независима и с отворен код уеб рамка, разработена от Microsoft. Тя позволява създаването на уеб приложения, услуги и API-та с висока производителност. Благодарение на своята модулна архитектура и платформена независимост, приложенията могат да работят на Windows, macOS и Linux.

Рамката е оптимизирана за облачни услуги, като Azure, и предлага мощни инструменти като Razor Pages и Model-View-Controller (MVC). Със своите висока скорост и гъвкавост, ASP.NET Core е идеален избор за съвременни, мащабируеми и сигурни уеб приложения.

## 1.2. MVC

MVC (Model-View-Controller) е архитектурен шаблон в програмирането, който разделя бизнес логиката на три свързани компонента: Model, View и Controller. Model представлява "ядрото" на приложението и съдържа всички данни, с които се работи. View е изходната част на софтуера, която визуализира обработените данни за потребителя. Controller извиква методи върху данните в модела, за да ги обработи и подготви за визуализация.

Тази структура осигурява ясно разделение на отговорностите, прави приложенията по-лесни за разработка и поддръжка и е основна част от работната рамка ASP.NET Core(Фигура 1).



Фигура 1. Визуализация на MVC

## 1.3. EntityFrameworkCore

С **Entity Framework Core (EF Core)** е лек и разширяем ORM (Object-Relational Mapping) инструмент, който позволява на разработчиците да работят с бази данни чрез .NET обекти, вместо да пишат SQL код. EF Core автоматично преобразува операциите като създаване, четене, актуализиране и изтриване на данни в SQL команди.

Той поддържа различни бази данни (SQL Server, MySQL, PostgreSQL и други) и позволява използването на LINQ за създаване на заявки. EF Core включва функции за миграции, които улесняват управлението на промените в базата данни, и предлага поддръжка за различни типове взаимоотношения между таблици. Това прави работата с бази данни по-бърза, лесна и ефективна.

## 1.4. База данни

## База данни е организирана колекция от данни, които се съхраняват и управляват по структуриран начин, позволявайки лесен достъп, манипулация и обновяване. Базите данни се използват за много видове проекти, в нашия случай – за управление на каталог за игри. Например, в базата данни на каталога се съхраняват информация за игрите като имена, жанрове, платформи, дати на издаване и оценки. Това позволява на потребителите лесно да разглеждат, добавят, редактират или изтриват игри, както и да търсят конкретни заглавия в каталога.

## SQL (Structured Query Language) е език за управление на релационни бази данни, който се използва за създаване, четене, актуализиране и изтриване на данни. В контекста на каталог за игри, SQL може да се използва за извличане на информация за конкретна игра, добавяне на нови записи или актуализиране на съществуващи.

## 1.5. Езици за програмиране

* + 1. **C#**: Езикът за програмиране, използван в сърцето на проекта, чрез който се реализира логиката на приложението, като работа с базата данни и обработка на потребителски заявки.
    2. **HTML**: Използва се за структурата на уеб страниците, създавайки основните елементи като заглавия, параграфи и формуляри, които потребителите взаимодействат.
    3. **CSS**: Отговаря за стилизирането и оформлението на уеб страниците, осигурявайки визуално привлекателен и подреден изглед на интерфейса.
    4. **JavaScript**: Използва се за добавяне на динамичност и интерактивност към сайта, като например обработка на събития от потребителския интерфейс и взаимодействие с данни в реално време.

# Глава 2 – Структура на базата данни

## 2.1. Структура на таблица и данни

В нашия проект се използва релационна SQL база от данни, във вид на таблица. Присъстващи колони в нея са: Name, Manufacturer, Description, Instruction, Rating, ImageUrl (Фигура 2).

|  |  |
| --- | --- |
| **Column** | **Type** |
| Name | [string] |
| Manufacturer | [string] |
| Description | [string] |
| Instruction | [string] |
| Rating | [number] |
| ImageUrl | [string] |

Фигура 2. Колоните и техните типове

## 

# Глава 3 – Реализация на Програмата

## 3.1. Създаване на база от данни

За създаване на базата от данни първо трябва да направим **model** с колоните, които искаме да има нашата таблица и техните типове на данните.

Модел Game:

public int Id { get; set; }

[Required]

[StringLength(100)]

public string Name { get; set; } = string.Empty;

[Required]

[StringLength(100)]

public string Manufacturer { get; set; } = string.Empty;

public string Description { get; set; } = string.Empty;

public string Instructions { get; set; } = string.Empty;

[Range(0, 5)]

public double Rating { get; set; }

public string ImageUrl { get; set; } = string.Empty;

След това дефинираме ApplicationDbContext, който служи като контекст за базата данни в приложението. Той наследява DbContext от Entity Framework Core и конфигурира връзката към базата данни чрез конструктора. Чрез свойството DbSet<Game> контекстът свързва модела Game.

ApplicationDbContext:

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options) { }

public DbSet<Game> Games { get; set; }

}

След това в appsetting.json файла променяме връзката, така че да се свърже с базата данни. В program.cs конфигурираме новия контекст на базата и в в конзолата създаваме миграция и обновяваме базата.

## 3.2. Game контролер

GameController е контролер в ASP.NET Core MVC приложение, който управлява CRUD операциите за игри в каталога, използвайки базата данни чрез ApplicationDbContext. Той съдържа следните основни действия: Преглед на игрите, създаване на нова игра, редактиране на вече създадена игра, преглед на игра, изтриване на игра и търсене на игра чрез търсачка.

Method за търсачка:

[HttpGet]

public IActionResult Search(string searchQuery)

{

var games = \_context.Games.AsQueryable();

if (!string.IsNullOrEmpty(searchQuery))

{

games = games.Where(g =>

g.Name.Contains(searchQuery) ||

g.Manufacturer.Contains(searchQuery));

}

return PartialView("\_GameList", games.ToList());

}

## 3.3. Javascript метод за търсачка

Този JavaScript код добавя динамично търсене на игри. Когато потребителят въвежда текст в полето с ID searchInput, се изпраща заявка към /Game/Search с текущия текст като параметър searchQuery. След като сървърът върне новия HTML с резултатите от търсенето, съдържанието на елемента с ID gameList се обновява, показвайки актуализирания списък с игри. В случай на грешка се извежда съобщение в конзолата. Това позволява на потребителите да виждат резултатите в реално време без презареждане на страницата.

Код:

document.getElementById('searchInput').addEventListener('input', function () {

const query = this.value;

fetch(`/Game/Search?searchQuery=${query}`)

.then(response => response.text())

.then(html => {

document.getElementById('gameList').innerHTML = html;

})

.catch(error => console.error('Error:', error));

});

## 3.4. Заглавие на параграф

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

## 3.5. Заглавие на параграф

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

# Глава 4 - ръководство за потребителя

## 4.1. Заглавие на параграф

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

## 4.2. Заглавие на параграф

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

## 4.3. Заглавие на параграф

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

## 4.4. Заглавие на параграф

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

## 4.5. Заглавие на параграф

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

Съдържание …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

# Заключение

Проектът "Каталог за игри" представлява ефективно решение за управление на игри, използващо ASP.NET Core MVC и SQL база данни. Чрез предоставените функционалности за добавяне, редактиране, преглеждане на детайли, изтриване, преглеждане на всички игри и търсене, потребителите и администраторите могат лесно да взаимодействат с каталога, като добавят нови записи, актуализират информация, или търсят игри.

Бъдещо развитие на проекта включва добавяне на нови функции като система за потребителски рецензии и оценки на игрите и възможност за създаване на списъци с любими игри.

# Използвани Източници

[Бек, Д. & Бек](https://www.orangecenter.bg/author/dzhesika-nyuman-bek-mat-bek.html), М. (2024). Бързо ръководство WordPress. София. [АлексСофт](https://www.orangecenter.bg/knizharnitsa/nehudozhestvena-literatura/informatsionni-tehnologii?publisher=%D0%90%D0%BB%D0%B5%D0%BA%D1%81%D0%A1%D0%BE%D1%84%D1%82)

Freeman, А. (2024). Pro ASP.NET Core 6. Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. София. APress

Коцев, Н., Коцев, Е. & Рускова, С. (2013). Основи на управлението. Русе: Авангард принт

MongoDB Inc. (2024). How to use Mern stack: a complete guide. Достъпно към 25.11.2024г. от <https://www.mongodb.com/resources/languages/mern-stack-tutorial>

Microsoft Learn Challenge. Първи стъпки с ASP.NET MVC. Достъпно към 25.11.2024г. от <https://learn.microsoft.com/bg-bg/aspnet/mvc/overview/getting-started/introduction/>

# Приложение 1 - GameController

public class GameController : Controller

{

private readonly ApplicationDbContext \_context;

public GameController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Game

public IActionResult Index(string searchQuery)

{

var games = \_context.Games.AsQueryable();

if (!string.IsNullOrEmpty(searchQuery))

{

games = games.Where(g =>

g.Name.Contains(searchQuery) ||

g.Manufacturer.Contains(searchQuery));

}

return View(games.ToList());

}

// GET: Game/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var game = await \_context.Games

.FirstOrDefaultAsync(m => m.Id == id);

if (game == null)

{

return NotFound();

}

return View(game);

}

// GET: Game/Create

public IActionResult Create()

{

return View();

}

// POST: Game/Create

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Name,Manufacturer,Description,Instructions,Rating,ImageUrl")] Game game)

{

if (ModelState.IsValid)

{

\_context.Add(game);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(game);

}

// GET: Game/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var game = await \_context.Games.FindAsync(id);

if (game == null)

{

return NotFound();

}

return View(game);

}

// POST: Game/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Name,Manufacturer,Description,Instructions,Rating,ImageUrl")] Game game)

{

if (id != game.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(game);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!GameExists(game.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(game);

}

// GET: Game/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var game = await \_context.Games

.FirstOrDefaultAsync(m => m.Id == id);

if (game == null)

{

return NotFound();

}

return View(game);

}

// POST: Game/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var game = await \_context.Games.FindAsync(id);

if (game != null)

{

\_context.Games.Remove(game);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

//Searching loggic

[HttpGet]

public IActionResult Search(string searchQuery)

{

var games = \_context.Games.AsQueryable();

if (!string.IsNullOrEmpty(searchQuery))

{

games = games.Where(g =>

g.Name.Contains(searchQuery) ||

g.Manufacturer.Contains(searchQuery));

}

return PartialView("\_GameList", games.ToList());

}

private bool GameExists(int id)

{

return \_context.Games.Any(e => e.Id == id);

}

}

}

# Приложение 2

Съдържание на източници. …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

# Приложение 3

Съдържание на източници. …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………

# Приложение 4

Съдържание на източници. …………………… ………… ……… ………….. ……………… ………………….. ……………………………. …………… ………………… ………