ДИПЛОМЕН ПРОЕКТ ЗА ДЪРЖАВЕН ЗРЕЛОСТЕН ИЗПИТ

по професия код 481030 „Приложен програмист“

специалност код 4810301 Приложно програмиране“

Тема: „Платформа за препродажба на стоки изпит по теория“

Автор:

Валентин Петров Петров, клас XII V

Ръководител:

Красимир Ватев

БургасСЪДЪРЖАНИЕ

[1 Увод 3](#_Toc119855807)

[2 Цели и обхват на софтуерното приложение 3](#_Toc119855808)

[3 Анализ на решението 3](#_Toc119855809)

[3.1 Потребителски изисквания и работен процес 3](#_Toc119855810)

[3.2 Примерен потребителски интерфейс 5](#_Toc119855811)

[3.3 Диаграми на анализа 5](#_Toc119855812)

[3.4 Модел на съдържанието / данните 5](#_Toc119855813)

[4 Дизайн 6](#_Toc119855814)

[4.1 Реализация на архитектурата на приложението 6](#_Toc119855815)

[4.2 Описание на слоевете, предназначението им, библиотеки и методи включени в съответния слой. 6](#_Toc119855816)

[4.3 Организация и код на заявките към база от данни 6](#_Toc119855817)

[4.4 Наличие на потребителски интерфейс (конзолен, графичен, уеб) 6](#_Toc119855818)

[5 Ефективност и бързодействие на решението 6](#_Toc119855819)

[6 Тестване 6](#_Toc119855820)

[7 Заключение и възможно бъдещо развитие 6](#_Toc119855821)

[8 Използвани литературни източници и Уеб сайтове 7](#_Toc119855822)

[9 Приложения 7](#_Toc119855823)

[10 Критерии и показатели за оценяване 8](#_Toc119855824)

# Увод

В днешния цифров свят електронната търговия е станала неотменима част от нашия ежедневен живот. Онлайн пазаруването не само прави пазаруването по-удобно за хората, които могат да пазаруват от удобството на своите домове, но и отваря нови възможности за бизнеса да достигне до по-голяма аудитория. Проектът ми, C# MVC .NET архитектурен електронен магазин за маркови дрехи, се фокусира върху осигуряването на изключително онлайн пазаруване за клиентите, които търсят да купуват дрехи.

Актуалността на темата за проекта се крие във факта, че онлайн пазарът за облекло бързо нараства и има нарастваща нужда от онлайн пазаруване, което е удобно за потребителите, ефективно и сигурно. Уебсайтът има за цел да се справи с проблемите, с които потребителите се сблъскват при пазаруване на дрехи онлайн.

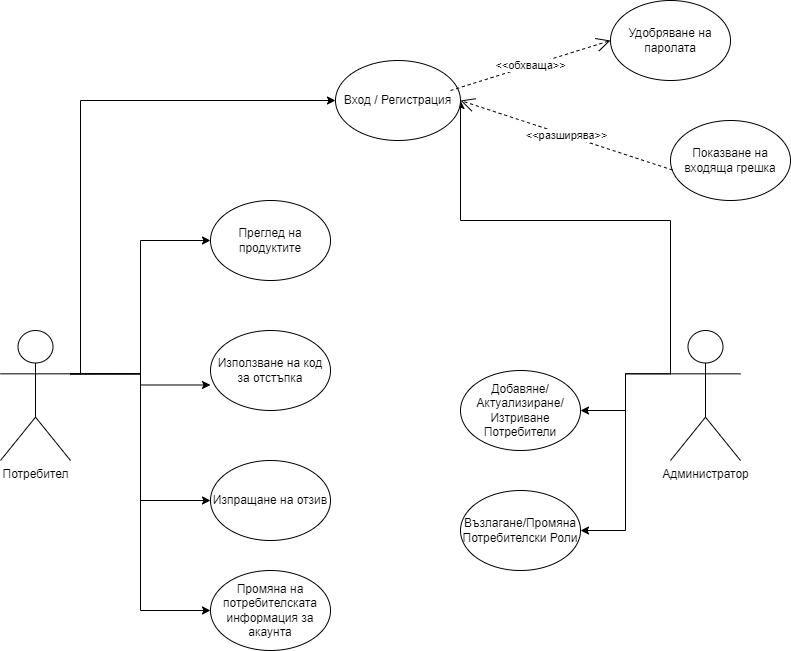
Проектът решава проблема с неудобството и неефективността, с които често се сблъскват клиентите при пазаруване на дрехи онлайн. В днешния бърз начин живот хората имат все по-малко време за пазаруване на място и искат да избегнат проблемите при посещаване на магазините. Уебсайтът позволява на потребителите да разглеждат и да купуват дрехи от удобството на своите домове. Освен това, уебсайтът е проектиран да осигури безпроблемно потребителско изживяване, като позволява на клиентите да навигират лесно през уебсайта, да преглеждат продукти и да извършват покупки на сигурен начин.

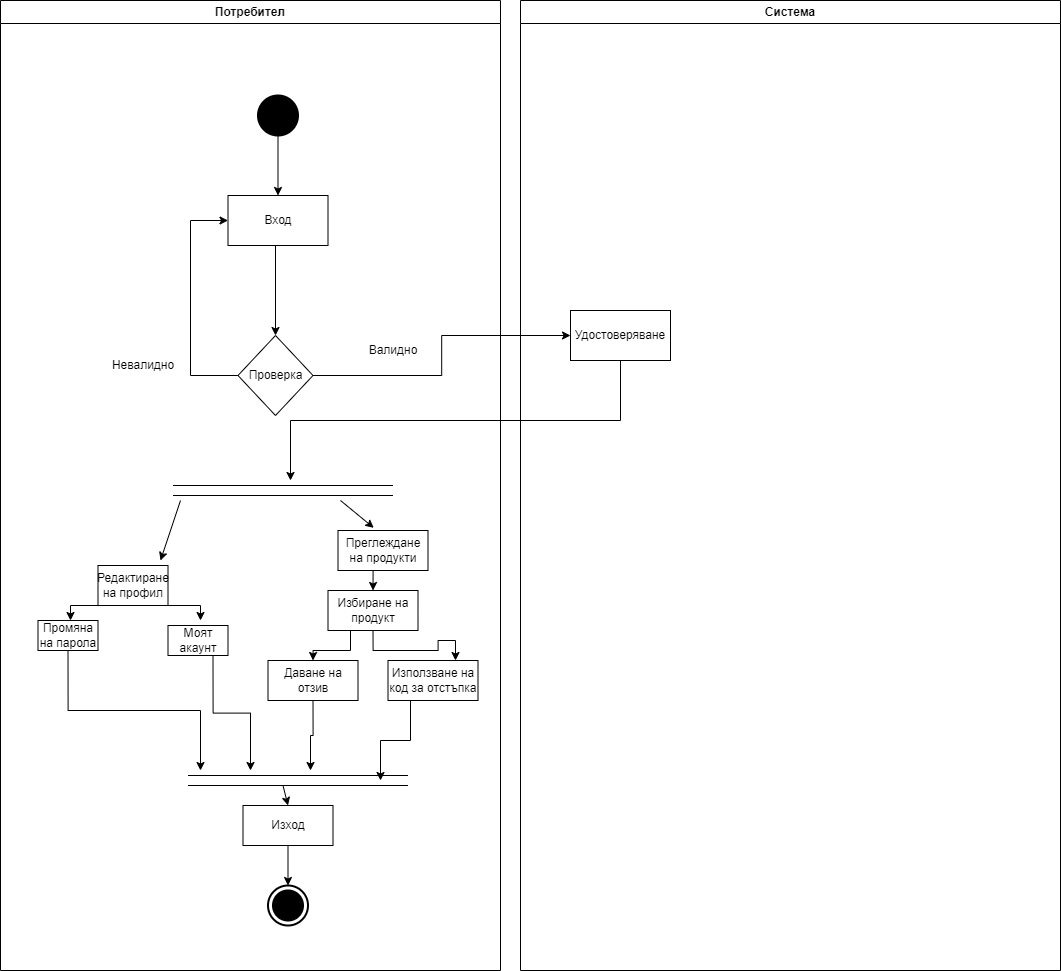
# Цели и обхват на софтуерното приложение

Целта на проекта е да има достъп до платформа за продажби на стоки, които могат да бъдат достъпвани от потребители с и без акаунти. Но по този начин ще бъдат ограничени от към това да дават ревю спрямо определен продукт или да ползват код за отстъпка. Продуктите биват публикувани от администратора. И той има достъп до други акаунти да променя определени данни и да задава роли за определен потребител. Като потребителят също може да променя данни за акаунта си. Бизнес цел е да се увеличи трафика на проекта. Като това, че анонимният потребител е ограничен от към отстъпки и подаване на ревю е начин за постигането на тази бизнес цел.

# Анализ на решението

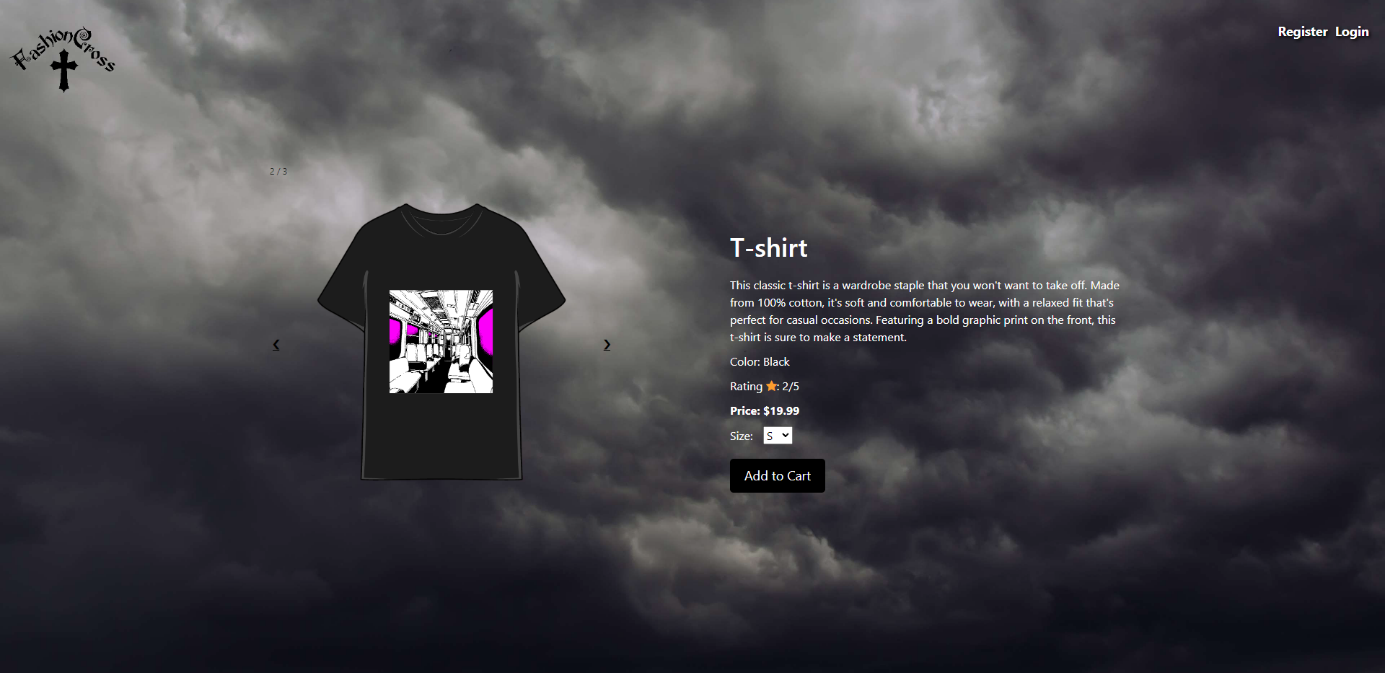
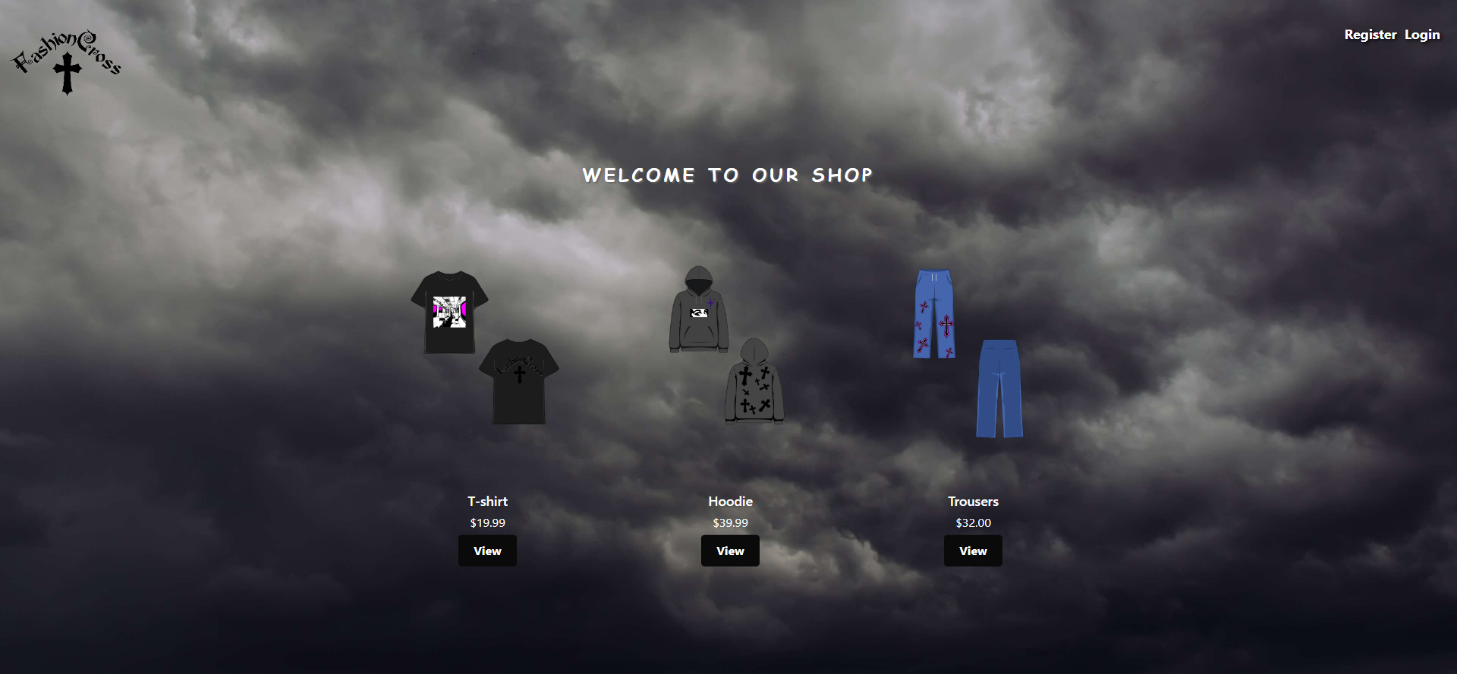
## Потребителски изисквания и работен процес





Потребителят влиза в сайта и има две опции да си направи акаунт/да се впише в съществуващ акаунт или да разгледа продуктите. Ако вписаните данни за акаунта са грешни се показва на екрана, че има входяща грешка. Ако не са грешни и всичко е наред потребителя се вписва успешно в акаунта си след като е удобрена паролата му автоматично за конкретния акаунт. След когато един потребител се е вписал в акаунта си той може да разглежда пак продуктите, но този път да използва функции, които анонимните потребители не могат. А те са ползване на код за отстъпка, промяна на потребителската информация на конкретния акаунт и изпращане на отзиви за определени продукти като отзивите се съхрянават в база данни и може и оценката, която прати потребител за определен продукт се добавя към другите оценки от 1-5 от другите потребители и накрая се прави средноаритметично на тези оценки и се записват в базата данни за конкретния продукт. Администратора за да използва функционалстите си като такъв трябва да се впише в акаунта си с правилната парола и имейл. За него ще има управляващ панел, който се показва само на администратора. От там той ще може да добавя, актуализира, изтрива потребители, както и да променя или вълга потребителски роли ако е нужно.

## Примерен потребителски интерфейс

Итерфейс: 

## Диаграми на анализа

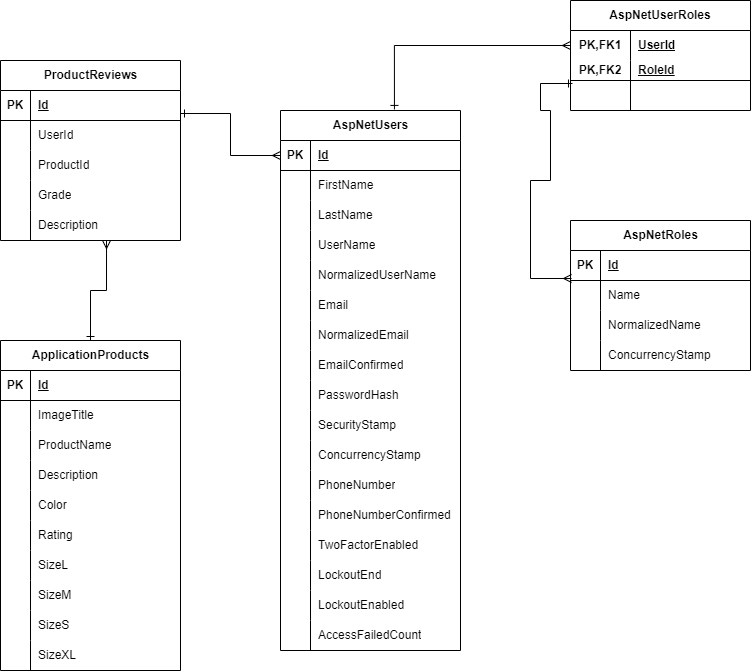
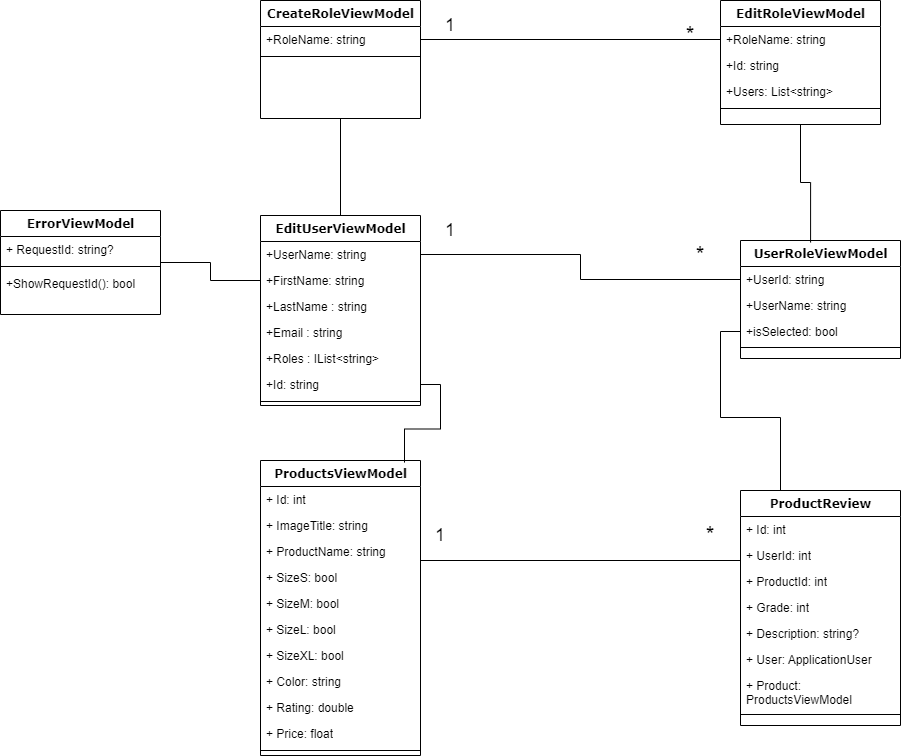


Таблица “Users” е с връзка едно към много с таблица “Roles”, защото всеки потребител може да има повече от една роля, докато една роля предналижи само на един потребител като са свързани с междинна таблица, където се взимат индентификационните им номера. Таблица “ApplicationProducts” има връзка едно към много с таблица “ProductReviews”, защото един продукт може да има много ревюта, но ревю пренадлежи само към един продукт. Връзката е едно към много също и с таблиците “ProductReviews” и “Users”, защото един потребител може да напише много ревюта, но едно ревю може да бъде написано само от един потребител.



Класовете - “CreateRoleViewModel” с “EditRoleViewModel”, “EditUserViewModel” с “UserRoleViewModel” и “ProductsViewModel” с “ProductReview” са с връзка едно към много, защото единичен “CreateRoleViewModel” клас може да бъде асоцииран с няколко “EditRoleViewModel” класа. Единичен “EditUserViewModel” клас може да бъде също асоцииран с няколко “UserRoleViewModel” класа. И същото се отнася за “ProductsViewModel” клас може да бъде асоцииран с няколко “ProductReview” класа.



Първа стъпка на потребителя е да се впише. Втора спъка е да отвори страницата за продукти и ако си хареса някой продукт да кликне на него. След това ще се отвори страница с избрания продукт. И от там вече може да прилага код за отстъпка ако той е валиден и написан правилно и да изпрати обратна връзка за продукта.

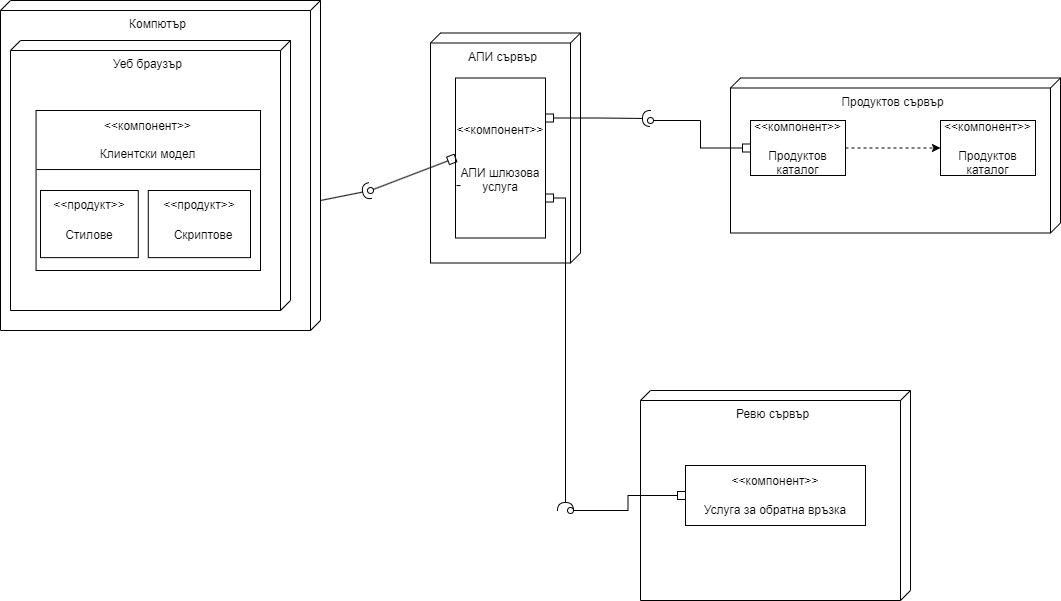
## Модел на съдържанието / данните

Проектът включва визуални медии под формата на снимки и видеоклипове. По-конкретно, видеоклипът се използва като фон за главната страница, докато снимките служат като фон за други страници или като визуални помощни средства за представяне на продуктите. Тази интеграция на мултимедийни елементи подобрява цялостната естетическа привлекателност и потребителското изживяване на проекта. Снимките са с разширение “.png”, а видеото е “.mp4” формат. За логото има използван така наречения “gif” за да може да се добие този въртящ 360 ефект.

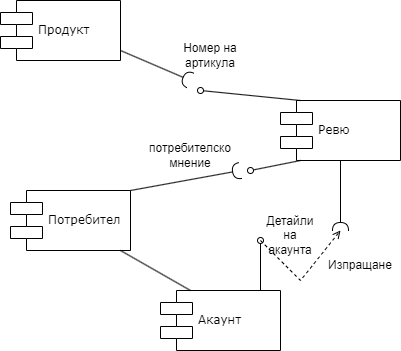
# Дизайн

Избрал съм да работя със софтуерната платформа .NET, защото тя е широко използвана, здрава и ми позволява да разработвам мащабируеми и надеждни софтуерни приложения. Платформата .NET предлага богат набор от библиотеки, инструменти и API-та, които ми помагат да увелича продуктивността си и да намаля времето за разработка. Като разработчик оценявам гъвкавостта и леснотата на употреба на .NET, което ми позволява лесно да изградя уеб, десктоп и мобилни приложения, използвайки общ език и среда за разработка. Освен това, платформата .NET има голяма и активна общност от разработчици, което означава, че лесно мога да намеря поддръжка, ресурси и най-добри практики, за да постигна успех в моите разработчески проекти.

Диаграма на разгръщането:



Диаграма на компонентите:



Сорс код за продуктовата страница:

@model ProductsViewModel

<head>

<link rel="stylesheet" type="text/css" href="~/css/ProductPage.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">

</head>

<div class="product-container">

@{

var imgPath1 = "/Images/Products/" + (@Model.ImageTitle + ".png");

var imgPath2 = "/Images/Products/" + (@Model.ImageTitle + "-1.png");

var imgPath3 = "/Images/Products/" + (@Model.ImageTitle + "-2.png");

}

<div class="slideshow-container">

<div class="mySlides">

<div class="numbertext">1 / 3</div>

<img src="@imgPath1" style="width:100%">

</div>

<div class="mySlides">

<div class="numbertext">2 / 3</div>

<img src="@imgPath2" style="width:100%">

</div>

<div class="mySlides">

<div class="numbertext">3 / 3</div>

<img src="@imgPath3" style="width:100%">

</div>

<a class="prev" onclick="plusSlides(-1)">❮</a>

<a class="next" onclick="plusSlides(1)">❯</a>

</div>

<br>

<div class="product-details">

<h1>@Model.ProductName</h1>

<p>@Model.Description</p>

<p>Цвят: @Model.Color</p>

<p>Оценка <span class="fa fa-star checked"></span>: @Model.Rating/5

&nbsp;

@if (User.Identity.IsAuthenticated)

{

<b><a style="cursor:pointer; color:white;" onclick="openForm()">Дайте отзив</a></b>

}

</p>

<div class="form-group row">

<div class="col-sm-10">

<div id="myForm">

<div id="myFormContent">

<span class="close" onclick="closeForm()">&times;</span>

<h2>Дайте своето мнение за продукта</h2>

<form onsubmit="submitForm(event)">

<input type="hidden" name="productId" value="@Model.Id" />

<div class="form-group">

<label for="grade">Оценка:</label>

<br/>

<div class="rate">

<input type="radio" id="star5" name="rate" value="5" />

<label for="star5" title="text">5 stars</label>

<input type="radio" id="star4" name="rate" value="4" />

<label for="star4" title="text">4 stars</label>

<input type="radio" id="star3" name="rate" value="3" />

<label for="star3" title="text">3 stars</label>

<input type="radio" id="star2" name="rate" value="2" />

<label for="star2" title="text">2 stars</label>

<input type="radio" id="star1" name="rate" value="1" />

<label for="star1" title="text">1 star</label>

</div>

</div>

<br/>

<br />

<div class="form-group">

<label for="description">Описание:</label>

<textarea class="form-control" id="description" name="description"></textarea>

</div>

<br/><br/>

<div class="myButton">

<button type="submit" onclick="submitForm(event)">

Изпрати

<div class="arrow-wrapper">

<div class="arrow"></div>

</div>

</button>

</div>

</form>

</div>

</div>

</div>

</div>

<b><p>Цена: @Model.Price.ToString("C")</p></b>

<div class="product-sizes">

<form>

<label for="size">Размер:</label>

<select id="size" name="size">

@if (Model.SizeS)

{

<option value="S">S</option>

}

@if (Model.SizeM)

{

<option value="M">M</option>

}

@if (Model.SizeL)

{

<option value="L">L</option>

}

@if (Model.SizeXL)

{

<option value="XL">XL</option>

}

</select>

</form>

</div>

@if (User.Identity.IsAuthenticated)

{

<div class="discount-container">

<label for="discount-code">Код за отстъпка:</label>

<input type="text" id="discount-code" name="discount-code">

<button class="apply-discount">Приложи</button>

<p id="discount-message"></p>

</div>

}

<button class="add-to-cart">Добавяне в кошницата</button>

</div>

</div>

@section Scripts {

<script>

//For the discount code

$(function () {

$('.apply-discount').on('click', function (e) {

e.preventDefault();

var discountCode = $('#discount-code').val().toUpperCase();

var discountMessage = $('#discount-message');

// check if the discount code is valid

if (discountCode === 'WELCOME') {

// apply 20% discount to the price

var newPrice = parseFloat('@Model.Price') \* 0.8;

$('.product-details p:contains("Цена:")').html('Цена: $' + newPrice.toFixed(2).toString());

discountMessage.text('The code is successfully applied!');

discountMessage.css('color', 'green');

}

else {

discountMessage.text('Invalid code!');

discountMessage.css('color', 'red');

}

});

});

//For the slideshow

let slideIndex = 1;

showSlides(slideIndex);

function plusSlides(n) {

showSlides(slideIndex += n);

}

function currentSlide(n) {

showSlides(slideIndex = n);

}

function showSlides(n) {

let i;

let slides = document.getElementsByClassName("mySlides");

let dots = document.getElementsByClassName("dot");

if (n > slides.length) { slideIndex = 1 }

if (n < 1) { slideIndex = slides.length }

for (i = 0; i < slides.length; i++) {

slides[i].style.display = "none";

}

for (i = 0; i < dots.length; i++) {

dots[i].className = dots[i].className.replace(" active", "");

}

slides[slideIndex - 1].style.display = "block";

dots[slideIndex - 1].className += " active";

}

// Function to open the pop-up window for the review

function openForm() {

document.getElementById("myForm").style.display = "block";

}

// Function to close the pop-up window

function closeForm() {

document.getElementById("myForm").style.display = "none";

}

//Submitting the info from the review and sending it to the controller

function submitForm(event) {

event.preventDefault(); // prevent the default form submission

// create a new FormData object

const formData = new FormData();

// get the form data and add it to the FormData object

const productId = document.querySelector('input[name="productId"]').value;

const grade = document.querySelector('input[name="rate"]:checked').value;

const description = document.querySelector('textarea[name="description"]').value;

formData.append("productId", productId);

formData.append("grade", grade);

formData.append("description", description);

// send an AJAX request to the server

fetch('@Url.Action("SubmitProductReview", "User")', {

method: 'POST',

body: formData

})

.then(response => {

if (response.ok) {

// redirect to the product page

window.location.href = '@Url.Action("ProductPage", "User")';

} else {

// handle error

alert('You already submit a review for this product');

console.error(response.statusText);

}

})

.catch(error => {

console.error(error);

});

}

</script>

}

## Реализация на архитектурата на приложението

Използвам “MVC” (Model View Controler) архитерктура за приложението за продажби. Като “MVC” е модел за проектиране, който разделя логиката на приложението на три компонента: модел, изглед и контролер. Слоят за данни в “ASP.NET Core MVC” приложение обикновено използва “ORM” рамка като” Entity Framework Core” за взаимодействие с базата данни. Фронт-ендът и бек-ендът на “ASP.NET Core MVC” приложението взаимодействат чрез “HTTP” заявки и отговори. Контролерът обработва заявките от фронт-ендът, използва модела, за да извлича и манипулира данни, и изпраща отговорите обратно към фронт-ендът, за да актуализира изгледа. Част от библиотеките, които използвам в моя проект са “Entity Framework” за базата данни и “xUnit” за тестването на апликацията. Избрах “.NET” заради поддръжката на тези библиотеки, както и заради неговата надеждност, мащабируемост и функции за сигурност.

За целта за създаването на приложението да взаймодествия с базата данни правилно се използва за по лесно пакета за идентичност по подразбиране на “.NET” (the dafault ASP .NET identity). И са създадени още две таблици в базата данни – за продуктите и за ревютата на потребителите. За продукти модела на базата данни е прост и оптимален като всеки друг – номер, име на снимката, име на продукт, описание, размерите (S,M,L,XL), цвят, оценка и цена. А за ревютата – номер на потребител, оценка и описанието пратено от тях. За цените на продуктите те биват нормално избрани спрямо другите приложения предлагащи продукти за продажба. Като за предпазване от злонамеронст е сложена двойна защита при промяна на данни за потребител от администратор като се изискват повторно данните за вход.

## Описание на слоевете, предназначението им, библиотеки и методи включени в съответния слой.

### Слоеве

Приложението е организирано с помощта на архитектурата Модел-Изглед-Контролер (Model-View-Controller-MVC), която се състои от следните слоеве:

Моделен слой: отговаря за определянето на структурата на данните и бизнес логиката на приложението. Този слой включва моделите на базата данни, обектите за достъп до данни и услугите, които взаимодействат с базата данни.

Слой Изглед: отговаря за представянето на данните на потребителя във формат, който е лесен за разбиране и взаимодействие. Този слой включва изгледите на “Razor” и шаблоните на “HTML”, които определят потребителския интерфейс.

Слой Контролер: отговаря за обработката на потребителските заявки и за взаимодействието с двата слоя Модел и Изглед. Този слой включва контролерите на “ASP.NET Core”, които обработват “HTTP” заявките и отговорите.

### Предназначение

Предназначението на всеки слой е следното:

Слой Модел гарантира, че данните на приложението са добре дефинирани, последователни и правилно управлявани. Той осигурява слой на абстракция между базата данни и останалата част от приложението, като улеснява актуализирането на схемата на базата данни или преминаването към друг механизъм за бази данни.

Слоят Изглед осигурява ясен и интуитивен потребителски интерфейс, който е лесен за използване и навигация. Той гарантира, че данните, представени на потребителя, са точни, актуални и визуално привлекателни.

Слоят Контролер (Контролер) управлява потока от данни между слоевете Модел и Изглед, като гарантира, че потребителските заявки се обработват бързо и ефективно. Той осигурява слой на абстракция между потребителя и вътрешната работа на приложението, като улеснява добавянето на нови функции или промяната на съществуващите.

### Библиотеки и методи

Всеки слой разчита на набор от библиотеки и методи, за да постигне целта си:

Слой Модел използва “Entity Framework Core” за взаимодействие с базата данни, предоставяйки рамка за обектно-релационно съпоставяне (ORM), която абстрахира детайлите на основния двигател на базата данни. Освен това той използва “LINQ” (Language Integrated Query) за писане на ефективни и кратки заявки.

Слоят Изглед използва шаблони “Razor” за генериране на “HTML”, “CSS” и “JavaScript” код, който се визуализира в уеб браузъра на потребителя. Той също така използва библиотеки от страна на клиента като “jQuery” и “Bootstrap”, за да подобри потребителското изживяване.

Слоят Контролер използва “ASP.NET Core” за обработване на “HTTP” заявки и отговори, като осигурява надеждна и разширяема рамка за изграждане на уеб приложения. Той също така използва междинен софтуер за добавяне на допълнителна функционалност, като например удостоверяване, кеширане или регистриране.

## Организация и код на заявките към база от данни

Потребителски контролер:

using EcommerceApp.Areas.Identity.Data;

using EcommerceApp.Models;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Security.Claims;

namespace EcommerceApp.Controllers

{

public class UserController : Controller

{

private readonly ApplicationDbContext \_context;

public UserController(ApplicationDbContext context)

{

\_context = context;

}

public ActionResult Products()

{

List<ProductsViewModel> products;

// The 'using' statement ensures proper disposal of the database context, including closing open connections and

// returning them to the connection pool. This helps prevent exhaustion of the pool and ensures proper resource management.

using (var context = \_context)

{

products = \_context.ApplicationProducts.Select(p => new ProductsViewModel

{

Id = p.Id,

ImageTitle = p.ImageTitle,

ProductName = p.ProductName,

Description = p.Description,

SizeS = p.SizeS,

SizeM = p.SizeM,

SizeL = p.SizeL,

SizeXL = p.SizeXL,

Color = p.Color,

Rating = p.Rating,

Price = p.Price

}).ToList();

return View(products);

}

}

public IActionResult ProductPage(int id)

{

var product = \_context.ApplicationProducts.FirstOrDefault(p => p.Id == id);

// check if product was found

if (product == null)

{

return NotFound();

}

// create a view model to pass to the view

var viewModel = new ProductsViewModel

{

Id = product.Id,

ImageTitle = product.ImageTitle,

ProductName = product.ProductName,

Description = product.Description,

SizeS = product.SizeS,

SizeM = product.SizeM,

SizeL = product.SizeL,

SizeXL = product.SizeXL,

Color = product.Color,

Rating = product.Rating,

Price = product.Price

};

// pass the view model to the view

return View(viewModel);

}

[HttpPost]

public IActionResult SubmitProductReview(int productId, int grade, string description)

{

// get the current user's id

var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;

// check if the user has already reviewed this product

var existingReview = \_context.ProductReviews.FirstOrDefault(r => r.ProductId == productId && r.UserId == userId);

if (existingReview != null)

{

// user has already reviewed this product, return an error message or redirect to an error page

return BadRequest(new { message = "You have already reviewed this product." });

}

// create a new ProductReview instance with the data submitted by the user

var review = new ProductReview

{

UserId = userId,

ProductId = productId,

Grade = grade,

Description = description

};

// add the new review to the database and save changes

\_context.ProductReviews.Add(review);

\_context.SaveChanges();

// Retrieve all reviews for the current product

List<ProductReview> reviews = \_context.ProductReviews

.Where(r => r.ProductId == productId)

.ToList();

// Calculate the average of the grades

double averageGrade = reviews.Average(r => r.Grade);

// Update the rating field in the ApplicationProducts table

var product = \_context.ApplicationProducts

.Where(p => p.Id == productId)

.FirstOrDefault();

if (product != null)

{

product.Rating = averageGrade;

\_context.SaveChanges();

}

return RedirectToAction("ProductPage", new { id = productId });

}

public ActionResult Sizing()

{

return View();

}

public ActionResult Policies()

{

return View();

}

}

}

Административен контролер:

using EcommerceApp.Areas.Identity.Data;

using EcommerceApp.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Runtime.ConstrainedExecution;

namespace EcommerceApp.Controllers

{

[Authorize(Roles="Admin")]

public class AdministrationController : Controller

{

private readonly RoleManager<IdentityRole> roleManager;

private readonly UserManager<ApplicationUser> userManager;

private readonly SignInManager<ApplicationUser> signInManager;

public AdministrationController(RoleManager<IdentityRole> roleManager,

UserManager<ApplicationUser> userManager,

SignInManager<ApplicationUser> signInManager)

{

this.roleManager = roleManager;

this.userManager = userManager;

this.signInManager = signInManager;

}

[HttpPost]

public async Task<IActionResult> DeleteUser(string id)

{

var user = await userManager.FindByIdAsync(id);

if (user == null)

{

ViewBag.ErrorMessage = $"User with Id = {id} cannot be found";

return View("NotFound");

}

else

{

var result = await userManager.DeleteAsync(user);

if (result.Succeeded)

{

return RedirectToAction("ListUsers");

}

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error.Description);

}

return View("ListUsers");

}

}

[HttpPost]

public async Task<IActionResult> DeleteRole(string id)

{

var role = await roleManager.FindByIdAsync(id);

if (role == null)

{

ViewBag.ErrorMessage = $"Role with Id = {id} cannot be found";

return View("NotFound");

}

else

{

var result = await roleManager.DeleteAsync(role);

if (result.Succeeded)

{

return RedirectToAction("ListRoles");

}

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error.Description);

}

return View("ListRoles");

}

}

[HttpGet]

public IActionResult ListUsers()

{

var users = userManager.Users;

return View(users);

}

[HttpGet]

public async Task<IActionResult> EditUser(string id)

{

var user = await userManager.FindByIdAsync(id);

if (user == null)

{

ViewBag.ErrorMessage = $"User with Id = {id} cannot be found";

return View("NotFound");

}

var userClaims = await userManager.GetClaimsAsync(user);

var userRoles = await userManager.GetRolesAsync(user);

var model = new EditUserViewModel

{

Id = user.Id,

Email = user.Email,

UserName = user.Email,

FirstName = user.FirstName,

LastName = user.LastName,

Roles = userRoles

};

return View(model);

}

[HttpPost]

public async Task<IActionResult> EditUser(EditUserViewModel model)

{

var adminEmail = Request.Form["adminEmail"];

var adminPassword = Request.Form["adminPassword"];

var userId = Request.Form["userId"];

var userFirstName = Request.Form["userFirstName"];

var userLastName = Request.Form["userLastName"];

var userEmail = Request.Form["userEmail"];

//The code for checking if the email and password are from the admin

// Get the current user

var currentUser = await userManager.GetUserAsync(User);

// Check if the current user is an admin

if (!await userManager.IsInRoleAsync(currentUser, "Admin"))

{

return Forbid(); // Return 403 Forbidden if the current user is not an admin

}

// Check if the provided admin email and password are correct

var resultAdmin = await signInManager.PasswordSignInAsync(adminEmail, adminPassword, false, lockoutOnFailure: false);

if (!resultAdmin.Succeeded)

{

return BadRequest(new { message = "Invalid admin email or password" }); // Return 400 Bad Request if the provided admin email or password are incorrect

}

//The code for the user that we wanna edit

var user = await userManager.FindByIdAsync(model.Id);

if (user == null)

{

ViewBag.ErrorMessage = $"User with Id = {model.Id} cannot be found";

return View("NotFound");

}

else

{

user.Email = userEmail;

user.UserName = userEmail;

user.FirstName = userFirstName;

user.LastName = userLastName;

var result = await userManager.UpdateAsync(user);

if (result.Succeeded)

{

return RedirectToAction("ListUsers");

}

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error.Description);

}

return View(model);

}

}

[HttpGet]

public IActionResult CreateRole()

{

return View();

}

[HttpPost]

public async Task<IActionResult> CreateRole(CreateRoleViewModel model)

{

if (ModelState.IsValid)

{

IdentityRole identityRole = new IdentityRole

{

Name = model.RoleName

};

IdentityResult result = await roleManager.CreateAsync(identityRole);

if (result.Succeeded)

{

return RedirectToAction("ListRoles", "Administration");

}

foreach(IdentityError error in result.Errors)

{

ModelState.AddModelError("", error.Description);

}

}

return View(model);

}

[HttpGet]

public IActionResult ListRoles()

{

var roles = roleManager.Roles;

return View(roles);

}

[HttpGet]

public async Task<IActionResult> EditRole(string id)

{

var role = await roleManager.FindByIdAsync(id);

if(role == null)

{

ViewBag.ErrorMessage = $"Role with Id= {id} cannot be found";

return View("NotFound");

}

var model = new EditRoleViewModel

{

Id = role.Id,

RoleName = role.Name

};

foreach(var user in userManager.Users)

{

if(await userManager.IsInRoleAsync(user, role.Name))

{

model.Users.Add(user.UserName);

}

}

return View(model);

}

[HttpPost]

public async Task<IActionResult> EditRole(EditRoleViewModel model)

{

var role = await roleManager.FindByIdAsync(model.Id);

if (role == null)

{

ViewBag.ErrorMessage = $"Role with Id= {model.Id} cannot be found";

return View("NotFound");

}

else

{

role.Name = model.RoleName;

var result = await roleManager.UpdateAsync(role);

if(result.Succeeded)

{

return RedirectToAction("ListRoles");

}

foreach(var error in result.Errors)

{

ModelState.AddModelError("", error.Description);

}

return View(model);

}

}

[HttpGet]

public async Task<IActionResult> EditUsersInRole(string roleId)

{

ViewBag.roleId = roleId;

var role = await roleManager.FindByIdAsync(roleId);

if (role == null)

{

ViewBag.ErrorMessage = $"Role with Id = {roleId} cannot be found";

return View("Not Found");

}

var model = new List<UserRoleViewModel>();

foreach(var user in userManager.Users)

{

var userRoleViewModel = new UserRoleViewModel

{

UserId = user.Id,

UserName = user.UserName

};

if(await userManager.IsInRoleAsync(user, role.Name))

{

userRoleViewModel.isSelected = true;

}

else

{

userRoleViewModel.isSelected = false;

}

model.Add(userRoleViewModel);

}

return View(model);

}

[HttpPost]

public async Task<IActionResult> EditUsersInRole(List<UserRoleViewModel> model, string roleId)

{

var role = await roleManager.FindByIdAsync(roleId);

if(role == null)

{

ViewBag.ErrorMessage = $"Role with Id = {roleId} cannot be found";

return View("NotFound");

}

for (int i = 0; i < model.Count; i++)

{

var user = await userManager.FindByIdAsync(model[i].UserId);

IdentityResult result = null;

if (model[i].isSelected && !(await userManager.IsInRoleAsync(user, role.Name)))

{

result = await userManager.AddToRoleAsync(user, role.Name);

}

else if (!model[i].isSelected && await userManager.IsInRoleAsync(user, role.Name))

{

result = await userManager.RemoveFromRoleAsync(user, role.Name);

}

else

{

continue;

}

if(result.Succeeded)

{

if (i < (model.Count - 1))

continue;

else

return RedirectToAction("EditRole", new { Id = roleId });

}

}

return RedirectToAction("EditRole", new {Id=roleId});

}

}

}

## Наличие на потребителски интерфейс (конзолен, графичен, уеб)

Началната страница на уеб приложението има видеоклип за заден фон и въртящо се лого с разширение .gif. Това е за цел да бъде по различната страницата и да направи впечатление на потребителя. Бутоните в проекта също така са стилизирани по подобаващ начин за да изглеждат по-добре и да правят впечатление. Стилизарана е и много добре ревю формата като потребителя вместо да избира от скучните цифри от 1-5, той може да плъзне мишката от звезда първа до пета като те се запълват с жълто. Също така задния фон за останалите страници е съвместен с този на началната страница, за да може да изглежда всичко свързано.

# Ефективност и бързодействие на решението

Едно от ключовите съображения при изграждането на високоефективно приложение за електронна търговия е ефективността и скоростта на внедрените решения. В този проект най-трудна беше задачата за потвърждаването на администратора преди да актуализира информацията на потребителя. Трябваше да се подходи с техника “Ajax”, която до сега не бях използвал. След имплементирането веднъж вече беше по лесно за създаването на останалите форми, където е било нужно.

Времевата сложност на админстративния контролер е “O(n)”, има два метода, коита са “O(1)”, а останалите са “O(n)”. Във потребителския контролер също има както и “O(1)” така и ”O(n)”, цялостно контролера и той е “O(n)”. Примeр за “O(n)”:

public ActionResult Products()

{

List<ProductsViewModel> products;

using (var context = \_context)

{

products = \_context.ApplicationProducts.Select(p => new ProductsViewModel

{

Id = p.Id,

ImageTitle = p.ImageTitle,

ProductName = p.ProductName,

Description = p.Description,

SizeS = p.SizeS,

SizeM = p.SizeM,

SizeL = p.SizeL,

SizeXL = p.SizeXL,

Color = p.Color,

Rating = p.Rating,

Price = p.Price

}).ToList();

return View(products);

}

}

Използвам xUnit тестовете. Избрал съм тях, защото предлагат редица предимства пред “нормалните” тестови пакети, включително по-мощно и гъвкаво. Такава е времевата сложност, защото се гледа сложността на времето за избор на всички продукти и създаване на модел на изглед за всеки продукт, което е “O(n)” , където n е броят на продуктите в базата данни. Докато когато времевата сложност е O(1), това означава, че времето за изпълнение на алгоритъма е постоянно и не зависи от размера на входа. С други думи, изпълнението на алгоритъма винаги ще отнема едно и също време, независимо от това колко голям или малък е входът. Ако проектът трябваше да се определи с една времева сложност, то тя щеше да е “O(n)”.

# Тестване

Използвам xUnit тестовете. Избрал съм тях, защото предлагат редица предимства пред “нормалните” тестови пакети, включително по-мощно и гъвкаво откриване на тестове, по-добра поддръжка за паралелно изпълнение и възможност за лесно интегриране с други инструменти за разработка. Използвам и “FakeItEasy” могат лесно да създават фалшиви имплементации на интерфейси или класове, да определят поведението на методите или свойствата и да проверяват дали определени методи са били извикани с определени аргументи. Това може да помогне за опростяване на тестването на единици, като позволи на разработчиците да изолират части от кода си и да ги тестват независимо.

Имам тестове за двата основни контрелера – потребителски и административен. В тестовия потребителския контролер файл имам два теста. Първия проверява дали метода “Products” от контролера връща изглед с празен лист, когато никакви продукти не съществуват. И другия тест проверява метода “ProductPage” дали връща “NotFound”, когато продукти не съществуват. В административния котролер имам също два теста. Първия е за метода “DeleteUser” проверявя дали ако админа изтрие потребител, той ще бъде изтрит и препратен на страницата с всички потребители. Втория тест е аналогичен на първия като само, че вместо за потребител става въпрос за роли. Метода е “DeleteRole” и проверява дали бива изпълнено и препратен администратора към листа с ролите. Ето един тест за пример от потребителския контролер (втория):

[Fact]

public void ProductPage\_Returns\_NotFound\_When\_Product\_Does\_Not\_Exist()

{

// Arrange

var dbContextOptions = new DbContextOptionsBuilder<ApplicationDbContext>()

.UseInMemoryDatabase(databaseName: "TestDatabase")

.Options;

var fakeDbContext = new ApplicationDbContext(dbContextOptions);

var controller = new UserController(fakeDbContext);

// Act

var result = controller.ProductPage(-1);

// Assert

result.Should().BeOfType<NotFoundResult>();

}

# Заключение и възможно бъдещо развитие

Обобщено, изработката на такъв проект не е лесна и бих казал, че съм научил много нови неща в програмиране, както и затвърдил стара информация и знания. Като бъдеща реализация е добе да се добавят повече различни продукти. Да се добави функцията “добавяне в кошница” и да може да се направи крайната форма за покупка. Добре ще е да се добави страница с изоложените ревюта от потребителите. От към административна гледна точка ще е по-добре да има упростен интерфейс - метод за качване на нови продукти в уеб сайта.

# Използвани литературни източници и Уеб сайтове

Използвайте вградената функционалност на Word: References > Citations & Bibliography

1. Уеб сайт на ….., адрес ….
2. Уеб сайт на ….., адрес ….
3. Уеб сайт на ….., адрес ….
4. (Microsoft, 2023)
5. Литературен източник 2
6. Литературен източник 3
7. Литературен източник 4
8. Литературен източник 5

# Приложения

Проектът на C# .NET MVC Core е уеб приложение за електронна търговия, което е предназначено да помага на бизнеса да продава своите продукти онлайн. Уебстраницата предоставя платформа, в която клиентите могат лесно да разглеждат, търсят и купуват продукти от широк набор от категории. То също така позволява на клиентите да оставят отзиви за продуктите, което може да помогне на бизнеса да подобри своите продукти и да увеличи удовлетвореността на клиентите.

Проектът разполага със система от кодове за отстъпки, която стимулира клиентите да правят покупки и помага на предприятията да стимулират продажбите. За да използват код за отстъпка, клиентите трябва да създадат акаунт, което помага на предприятията да изградят база от лоялни клиенти.

Проектът включва също потребителски роли и разрешения, които помагат на предприятията да управляват данните на своите клиенти и служители. Например предприятията могат да използват ролята администратор, за да управляват данните на потребителите в базата данни, включително ролите и разрешенията.

Като цяло уеб приложението за електронна търговия помага на бизнеса да увеличи продажбите си, да достигне до повече клиенти и да подобри продуктите си и удовлетвореността на клиентите.

# Критерии и показатели за оценяване

|  |  |  |
| --- | --- | --- |
| Критерии и показатели за оценяване | Максимален брой точки за показателите | Максимален брой точки за критерия |
| 1. Съответствие с изискванията за съдържание и структура на дипломния проект |  | 20 |
| 1. 1. логическа последователност и структура на изложението, балансиране на отделните части | 4 |  |
| 1.2. задълбоченост и пълнота при формулиране на обекта, предмета, целта и задачите в разработването на темата | 7 |  |
| 1.3. използване на подходящи изследователски методи | 4 |  |
| 1.4. стил и оформяне на дипломната работа (терминология, стил на писане, текстообработка и оформяне на фигури и таблици) | 5 |  |
| 2. Съответствие между поставените цели на дипломния проект и получените резултати |  | 20 |
| 2.1. изводите следват пряко от изложението, формулирани са ясно, решават поставените в началото на изследването цели и задачи и водят до убедителна защита на поставената теза | 10 |  |
| 2.2. оригиналност, значимост и актуалност на темата | 6 |  |
| 2.3. задълбоченост и обоснованост на предложенията и насоките | 4 |  |
| 3. Представяне на дипломния проект |  | 20 |
| 3.1. представянето на разработката по темата e ясно и точно | 5 |  |
| 3.2. онагледяване на експозето с:  а) презентация;  б) графични материали;  в) практически резултати;  г) компютърна мултимедийна симулация и анимация | 10 |  |
| 3.3. умения за презентиране | 5 |  |
| 4. Отговори на зададените въпроси от рецензента и/или членовете на комисията за защита на дипломен проект |  | 30 |
| 4.1. разбира същността на зададените въпроси и отговаря пълно, точно и убедително | 10 |  |
| 4.2. логически построени и точни отговори на зададените въпроси | 10 |  |
| 4.3. съдържателни и обосновани отговори на въпросите | 10 |  |
| 5. Използване на професионалната терминология, добър и ясен стил, обща езикова грамотност |  | 10 |
| 5.1. Правилно използване на професионалната терминология | 5 |  |
| 5.2. Ясен изказ и обща езикова грамотност | 5 |  |
| Общ брой точки: | Максимален бр. точки 100 | Максимален бр. точки 100 |