Министерство науки и высшего образования Российской Федерации **Федеральное государственное бюджетное образовательное учреждение высшего образования**

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых» (ВлГУ)

Кафедра информационных систем и программной инженерии

**КУРСОВОЙ ПРОЕКТ**

Приложение для информационной системы «Виртуальная арт-галерея».

Модуль «Администрирование»

Выполнил: студент гр. ИСТ-122

Жминьковская М.А.

Принял: доц. Вершинин В.В.

Владимир, 2024

**СОДЕЖАНИЕ**

[Этап 1 3](#_Toc182945145)

[Этап 2 7](#_Toc182945146)

[Этап 3 11](#_Toc182945147)

# Этап 1

**1. Постановка задачи**

Целью данного курсового проекта является разработка программной информационной системы «Виртуальная арт-галерея».

Функциональные требования:

— Авторизация и регистрация (администратор, художник, посетитель)

— Система рейтинга публикаций (художник, посетитель)

— Статистика посещаемости сайта (администратор)

— Управление категориями публикаций (администратор)

— Управление профилями художников (администратор)

— Автоматическое распределение публикаций по категориям (администратор, художник)

**2. Описание предмостной области**

Для художников необходимо иметь платформу, на которой можно размещать собственные работы с целью получения известности или поиска работодателя. Представителям компаний, в чьи ряды требуется графический художник той или иной специальности, важно иметь на руках портфолио кандидата, и именно платформы с графическими публикациями не только упрощают поиск, но и дают доступ к работам и ценам на услуги художника.

Виртуальная арт-галерея – это как раз платформа для художников и их нанимателей. Яркий пример – зарубежный сайт artstation.com. Российского аналога нет, следовательно, требуется заняться его разработкой.

**3. Словарь предметной области**

* Посетитель – незарегистрированный пользователь, который обладает минимальными правами; характеристики: нет; действия: просматривать публикацию, просматривать рейтинг публикации.
* Художник – пользователь, который зарегистрировался и прошёл авторизацию; характеристики: имя, пароль, электронная почта; действия: регистрация, вход, выход, создание/редактирование/удаление публикации, оценка публикации, редактирование категории публикации вручную.
* Администратор – зарегистрированный пользователь с расширенными правами доступа; характеристика: имя, пароль, электронная почта; действия: управление категорией, редактирование/удаление профиля художника, просмотр статистики.
* Публикация – Работа художника в профиле и/или выставке; характеристика: название, описание, фото, рейтинг, дата размещения, статистика; действия: создание черновика, редактирование/удаление публикации
* Категория публикации – тип изображения в зависимости от содержания, тематики и стиля; характеристика: название; действия: редактирование категории.
* Статистика – сбор данных; характеристика: нет, действия: сбор информации о просмотрах публикации, профиля художника.

**4. Диаграмма прецедентов**

Диаграмма прецедентов представлена на рисунке 1.



Рисунок 1. Диаграмма прецедентов

**Описание прецедентов:**

**1)** **Название:** Оценка публикации

**Предусловие**: Пользователь авторизован в системе.

**Действующее лицо:** Художник

**Основной поток:** Оценка публикации

Художник нажимает на публикацию, которую хочет оценить и переходит в окно публикации.

Художник ставит оценку.

Система сохраняет оценку и обновляется рейтинг публикации.

**Альтернативный поток:** нет

**Постусловие:** Рейтинг публикации обновлен с учетом новой оценки.

**2) Название:** Добавить публикацию

**Предусловие:** нет

**Действующее лицо:** Художник

**Основной поток:** Добавление публикации

Художник нажимает на кнопку «Публикация» и его перебрасывает в окно создания публикации.

Художник загружает изображение, добавляет описание и категорию работы.

Художник сохраняет публикацию и ей присваивается статус «Черновик».

Система сохраняет черновик публикации в базе данных и отправляет уведомление администратору о новом черновике.

Художник может редактировать или удалять публикацию пока она находится в состоянии «Черновик».

**Альтернативный поток:** Публикация не добавлена

Если не заполнены необходимые поля, выведется сообщение, что их необходимо заполнить, публикация не будет сохранена в статусе «Черновик».

**Постусловие:** Публикация художника находится в состоянии «Черновик» и ждет подтверждения от администратора.

**3) Название:** Подтверждение публикации

**Предусловие:** Администратор авторизован в системе

**Действующее лицо:** Администратор

**Основной поток:** Подтверждение публикации

Администратор нажимает на кнопку «Подтверждение публикации», после чего его перебрасывает на страницу с отображением публикаций с информацией о них и статусом.

Администратор выбирает статус публикации «Опубликовано».

**Альтернативный поток:** Публикация не подтверждена

Если публикация не удовлетворяет правилам, администратор выбирает статус «Отклонено».

**Постусловие:** Публикация подтверждена и доступна для просмотра

**5. Диаграмма классов**

Диаграмма классов представлена на рисунке 2.



Рисунок 2. Диаграмма классов

# Этап 2

**6. Диаграмма состояний**

**Описание диаграммы состояний для объекта «Публикация»:**

1. **Создание публикации (Черновик):**

Художник создает новую публикацию, добавляя название, описание и загружая свою работу. Объект «Публикация» создается в системе. Публикация находится в состоянии «Черновик». Пользователь может внести изменения в черновик перед отправкой на модерацию.

1. **Отправка на модерацию:**

Художник завершил редактирование публикации и отправил её на модерацию. Публикация помечается как «Ожидает модерации». На этом этапе публикация не доступна для публичного просмотра, ожидается одобрение администратора.

1. **Модерация отклонена:**

Администратор проверил публикацию и отклонил её по причине нарушения правил или недостаточного качества. Художник получает уведомление об отклонении на указанный им e-mail. Публикация возвращается в состояние «Черновик», чтобы художник мог внести необходимые исправления.

1. **Модерация одобрена:**

Администратор проверил публикацию и одобрил её. Публикация переводится в состояние «Опубликовано» и становится доступной для всех пользователей системы.

1. **Редактирование публикации:**

Художник решил внести изменения в уже опубликованную работу. Публикация переводится в состояние «Публикация редактируется», при этом публикация переводится в состояние «Черновик». Изменения находятся в состоянии черновика до повторной отправки на модерацию.

**6.** **Удаление публикации**:

Художник или администратор удаляет публикацию. Публикация помечается как «Удалено» и больше не доступна для пользователей. Художник получает уведомление об удалении публикации.

Диаграмма состояний представлена на рисунке 3.



Рисунок 3. Диаграмма состояний

**7. Диаграмма последовательностей**

**Описание диаграммы последовательностей для объекта «Публикация»:**

Художник вызывает метод ViewArtwork(title, imade, description), чтобы просмотреть публикацию.

Публикацию можно отредактировать при её просмотре. Для редактирования — метод EditArtwork(title, image, description), который изменяет существующую публикацию.

Публикацию можно удалить при её просмотре. Для удаления — метод DeleteArtwork(id).

Каждое действие вызывает ответный сигнал от объекта Artwork, подтверждающий успешное выполнение запроса.

Диаграмма последовательностей представлена на рисунке 4.



Рисунок 4. Диаграмма последовательностей

**8. Диаграмма видов деятельности**

Диаграмма видов деятельностей представлена на рисунке 5.



Рисунок 5. Диаграмма видов деятельностей

# Этап 3

**Models:**  
using Microsoft.AspNetCore.Identity;

using System.ComponentModel.DataAnnotations;

namespace Virtual\_Art\_Gallery.Models

{

public class ArtworkModel

{

[Key]

public int Id { get; set; }

[Required(ErrorMessage = "Введите название публикации")]

public string Title { get; set; }

public string? Description { get; set; }

[Required(ErrorMessage = "Выберите категорию")]

public int CategoryId { get; set; }

public CategoryModel? Category { get; set; }

public string? ImagePath { get; set; }

public int Width { get; set; }

public int Height { get; set; }

[Required]

public DateTime DateCreated

{

get; set;

}

}

}

using System.ComponentModel.DataAnnotations;

namespace Virtual\_Art\_Gallery.Models

{

public class CategoryModel

{

[Key]

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

}

}

**Context:**

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using Virtual\_Art\_Gallery.Models;

namespace Virtual\_Art\_Gallery.Data

{

public class GalleryContext : IdentityDbContext

{

public GalleryContext(DbContextOptions<GalleryContext> options) : base(options) { }

public DbSet<ArtworkModel> Artworks { get; set; }

public DbSet<CategoryModel> Categories { get; set; }

}

}

**Controllers:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Virtual\_Art\_Gallery.Data;

using SixLabors.ImageSharp;

using Virtual\_Art\_Gallery.Models;

using System;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

namespace Virtual\_Art\_Gallery.Controllers

{

public class ArtworkController : Controller

{

private readonly GalleryContext \_context;

public ArtworkController(GalleryContext context)

{

\_context = context;

}

// GET: Artwork

public async Task<IActionResult> Index()

{

var artworks = await \_context.Artworks.Include(a => a.Category).ToListAsync();

return View(artworks);

}

// GET: Artwork/Details/5

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

{

if (id == null)

{

return NotFound();

}

var artwork = await \_context.Artworks.Include(a => a.Category)

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

if (artwork == null)

{

return NotFound();

}

return View(artwork);

}

// GET: Artwork/Create

public IActionResult Create()

{

ViewData["CategoryList"] = new SelectList(\_context.Categories, "Id", "Name");

return View(new ArtworkModel());

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Title,Description,CategoryId")] ArtworkModel artwork, IFormFile imageFile)

{

{

try

{

if (ModelState.IsValid)

{

artwork.DateCreated = DateTime.Now;

if (imageFile != null && imageFile.Length > 0)

{

var allowedExtensions = new[] { ".jpg", ".jpeg", ".png", ".gif" };

var fileExtension = Path.GetExtension(imageFile.FileName).ToLower();

if (!allowedExtensions.Contains(fileExtension))

{

ModelState.AddModelError("ImageFile", "Недопустимый формат файла.");

ViewData["CategoryList"] = new SelectList(\_context.Categories, "Id", "Name", artwork.CategoryId);

return View(artwork);

}

var fileName = Path.GetFileNameWithoutExtension(imageFile.FileName) + "\_" + Guid.NewGuid() + fileExtension;

var imagesFolder = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot/images");

if (!Directory.Exists(imagesFolder))

{

Directory.CreateDirectory(imagesFolder);

}

var filePath = Path.Combine(imagesFolder, fileName);

using (var fileStream = new FileStream(filePath, FileMode.Create))

{

await imageFile.CopyToAsync(fileStream);

}

using (var img = Image.Load(filePath))

{

artwork.Width = img.Width;

artwork.Height = img.Height;

}

artwork.ImagePath = "/images/" + fileName;

}

\_context.Add(artwork);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["CategoryList"] = new SelectList(\_context.Categories, "Id", "Name", artwork.CategoryId);

return View(artwork);

}

catch (Exception ex)

{

Console.WriteLine($"Общая ошибка: {ex.Message}");

ViewData["CategoryList"] = new SelectList(\_context.Categories, "Id", "Name", artwork.CategoryId);

return View(artwork);

}

}

}

// GET: Artwork/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var artwork = await \_context.Artworks.FindAsync(id);

if (artwork == null)

{

return NotFound();

}

ViewData["CategoryList"] = new SelectList(\_context.Categories, "Id", "Name", artwork.CategoryId);

return View(artwork);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Title,Description,CategoryId,ImagePath")] ArtworkModel artwork, IFormFile imageFile)

{

if (id != artwork.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

var existingArtwork = await \_context.Artworks.FindAsync(id);

if (existingArtwork == null)

{

return NotFound();

}

existingArtwork.Title = artwork.Title;

existingArtwork.Description = artwork.Description;

existingArtwork.CategoryId = artwork.CategoryId;

if (imageFile != null && imageFile.Length > 0)

{

var fileName = Path.GetFileNameWithoutExtension(imageFile.FileName) + "\_" + Guid.NewGuid() + Path.GetExtension(imageFile.FileName);

var filePath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot/images", fileName);

using (var fileStream = new FileStream(filePath, FileMode.Create))

{

await imageFile.CopyToAsync(fileStream);

}

existingArtwork.ImagePath = "/images/" + fileName;

using (var img = Image.Load(filePath))

{

existingArtwork.Width = img.Width;

existingArtwork.Height = img.Height;

}

existingArtwork.DateCreated = DateTime.Now;

}

else

{

artwork.ImagePath = existingArtwork.ImagePath;

artwork.Width = existingArtwork.Width;

artwork.Height = existingArtwork.Height;

artwork.DateCreated = existingArtwork.DateCreated;

}

\_context.Update(existingArtwork);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ArtworkModelExists(artwork.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["CategoryList"] = new SelectList(\_context.Categories, "Id", "Name", artwork.CategoryId);

return View(artwork);

}

// GET: Artwork/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var artwork = await \_context.Artworks.Include(a => a.Category)

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

if (artwork == null)

{

return NotFound();

}

return View(artwork);

}

// POST: Artwork/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var artwork = await \_context.Artworks.FindAsync(id);

if (artwork == null)

{

return NotFound();

}

\_context.Artworks.Remove(artwork);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ArtworkModelExists(int id)

{

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

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Virtual\_Art\_Gallery.Data;

using Virtual\_Art\_Gallery.Models;

namespace Virtual\_Art\_Gallery.Controllers

{

[Authorize(Roles = "Administrator")]

public class CategoryController : Controller

{

private readonly GalleryContext \_context;

public CategoryController(GalleryContext context)

{

\_context = context;

}

// GET: Category

public async Task<IActionResult> Index()

{

return View(await \_context.Categories.ToListAsync());

}

// GET: Category/Details/5

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

{

if (id == null)

{

return NotFound();

}

var categoryModel = await \_context.Categories

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

if (categoryModel == null)

{

return NotFound();

}

return View(categoryModel);

}

// GET: Category/Create

public IActionResult Create()

{

ViewData["CategoryList"] = new SelectList(\_context.Categories, "Id", "Name");

return View();

}

// POST: Category/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Name,Description")] CategoryModel categoryModel)

{

if (ModelState.IsValid)

{

\_context.Add(categoryModel);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(categoryModel);

}

// GET: Category/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var categoryModel = await \_context.Categories.FindAsync(id);

if (categoryModel == null)

{

return NotFound();

}

return View(categoryModel);

}

// POST: Category/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Name,Description")] CategoryModel categoryModel)

{

if (id != categoryModel.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(categoryModel);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!CategoryModelExists(categoryModel.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(categoryModel);

}

// GET: Category/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var categoryModel = await \_context.Categories

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

if (categoryModel == null)

{

return NotFound();

}

return View(categoryModel);

}

// POST: Category/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var categoryModel = await \_context.Categories.FindAsync(id);

if (categoryModel != null)

{

\_context.Categories.Remove(categoryModel);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool CategoryModelExists(int id)

{

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

}

}

}

**Program.cs**

using Microsoft.AspNetCore.Identity;

using Virtual\_Art\_Gallery.Data;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<GalleryContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("GalleryContext")));

builder.Services.AddIdentity<IdentityUser, IdentityRole>()

.AddEntityFrameworkStores<GalleryContext>()

.AddDefaultUI()

.AddDefaultTokenProviders();

builder.Services.AddControllersWithViews();

var app = builder.Build();

var scope = app.Services.CreateScope();

var roleManager = scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();

var userManager = scope.ServiceProvider.GetRequiredService<UserManager<IdentityUser>>();

await CreateRoles(roleManager);

await AssignAdminRoleToFirstUser(userManager);

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.MapRazorPages();

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Gallery}/{action=Index}/{id?}");

app.Run();

async Task CreateRoles(RoleManager<IdentityRole> roleManager)

{

string[] roleNames = { "User", "Administrator" };

foreach (var roleName in roleNames)

{

var roleExist = await roleManager.RoleExistsAsync(roleName);

if (!roleExist)

{

await roleManager.CreateAsync(new IdentityRole(roleName));

}

}

}

async Task AssignAdminRoleToFirstUser(UserManager<IdentityUser> userManager)

{

var users = userManager.Users.ToList();

if (users.Count == 0)

{

return;

}

var firstUser = users.FirstOrDefault();

if (firstUser != null)

{

var isUserInRole = await userManager.IsInRoleAsync(firstUser, "Administrator");

if (!isUserInRole)

{

await userManager.AddToRoleAsync(firstUser, "Administrator");

}

}

}

На рисунках 6- представлена демонстрация страниц.

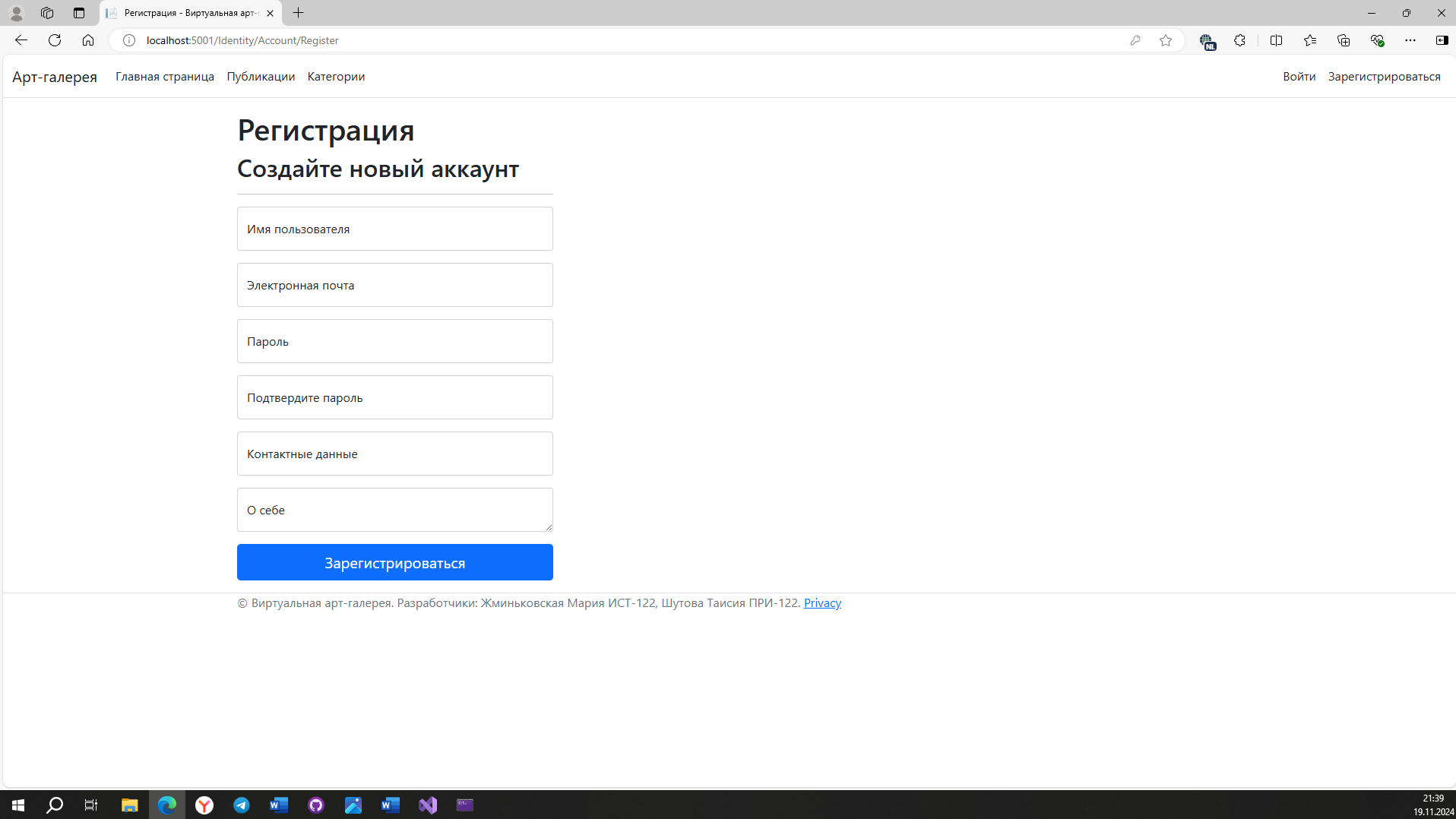


Рисунок 6. Регистрация

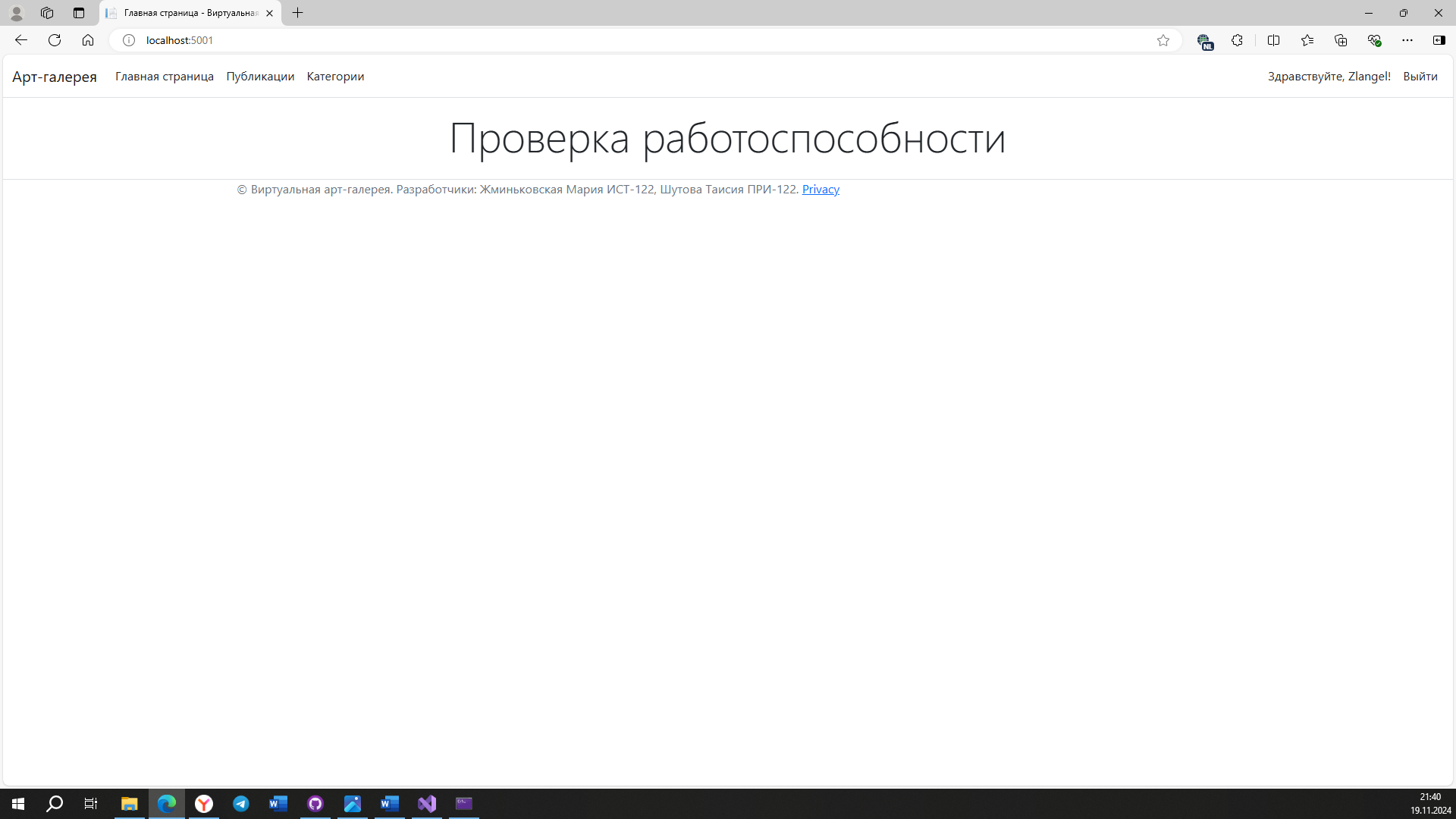


Рисунок 7. После авторизации

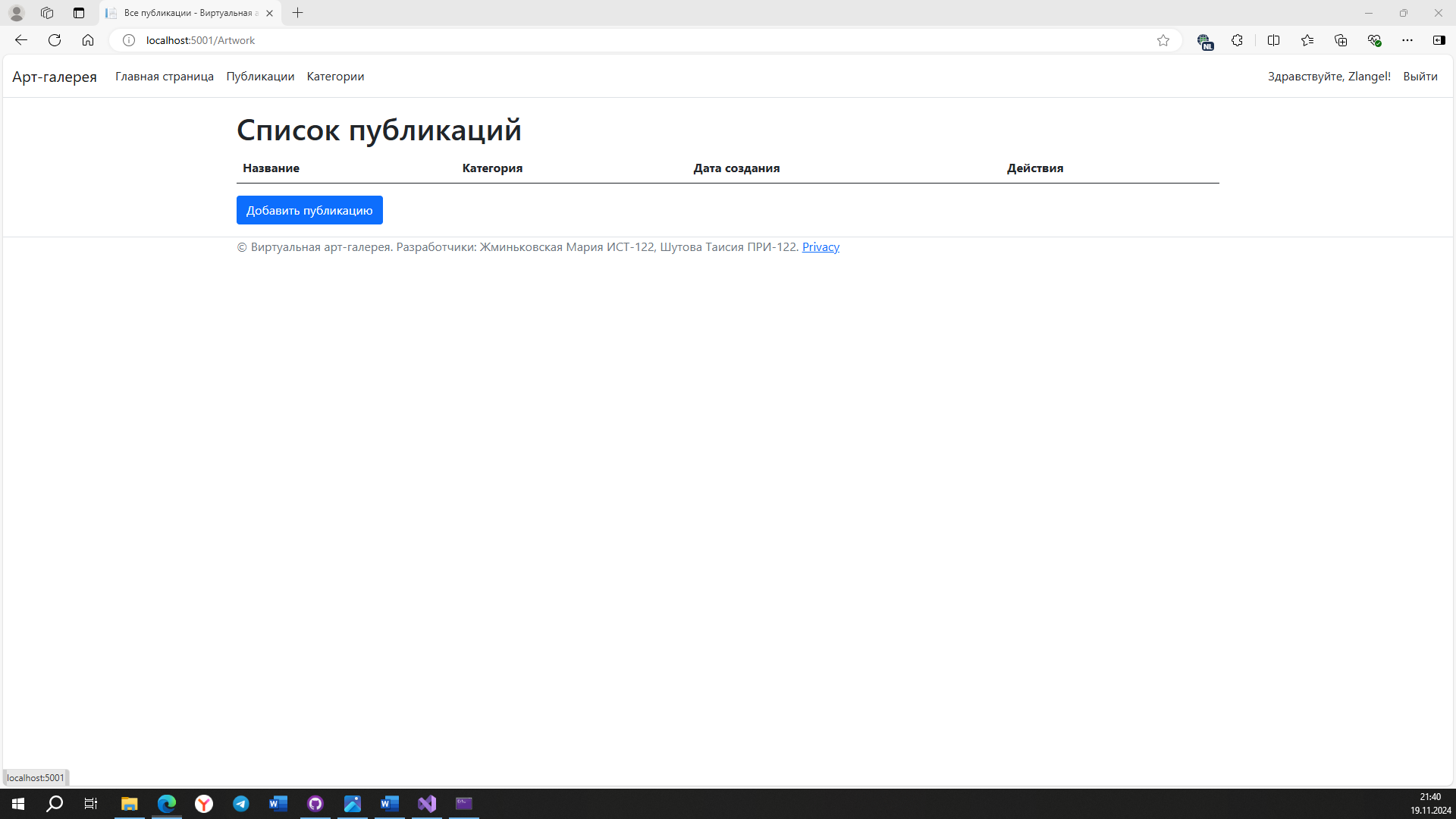


Рисунок 8. Публикации

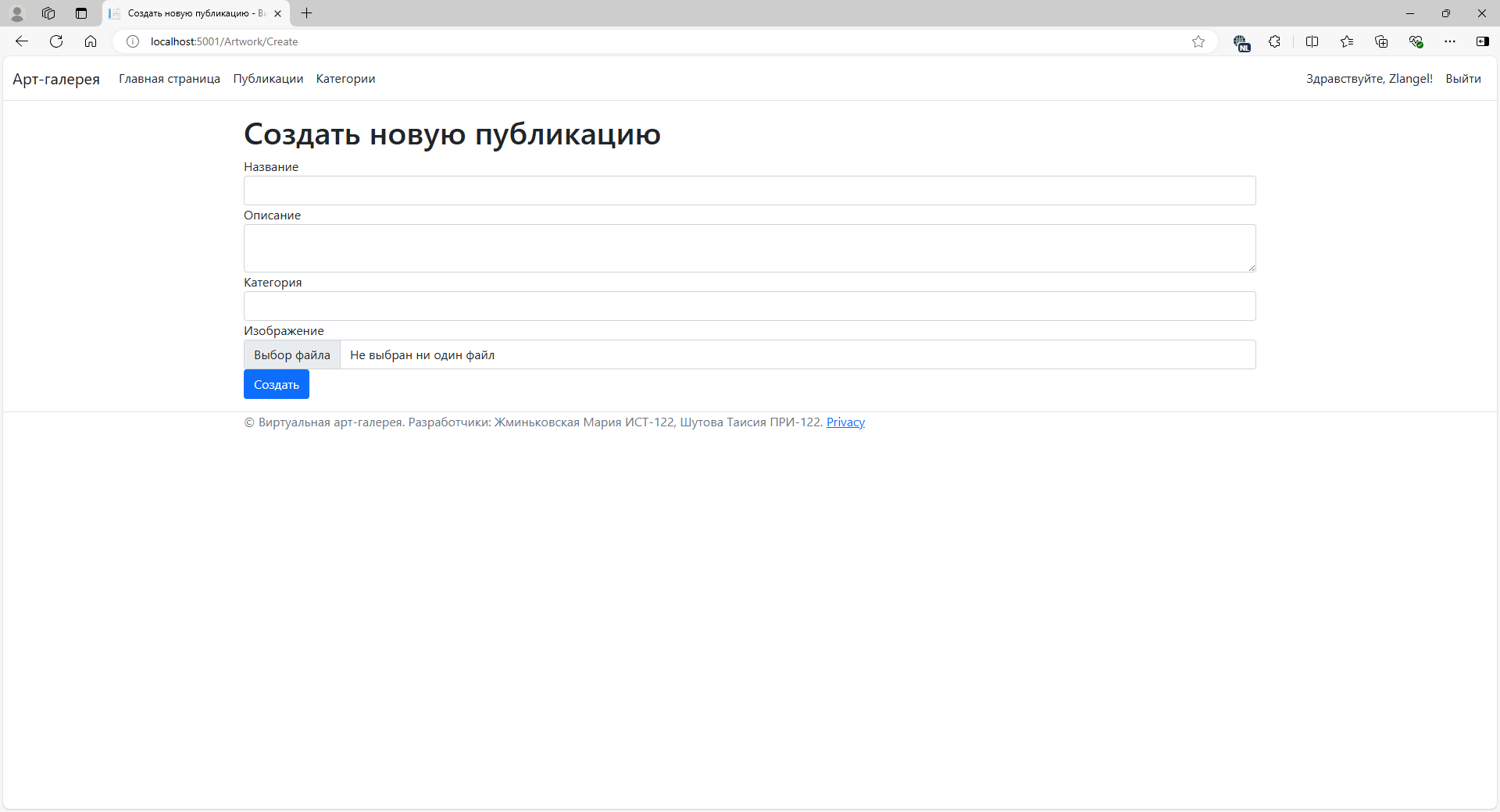


Рисунок 9. Окно создания публикации

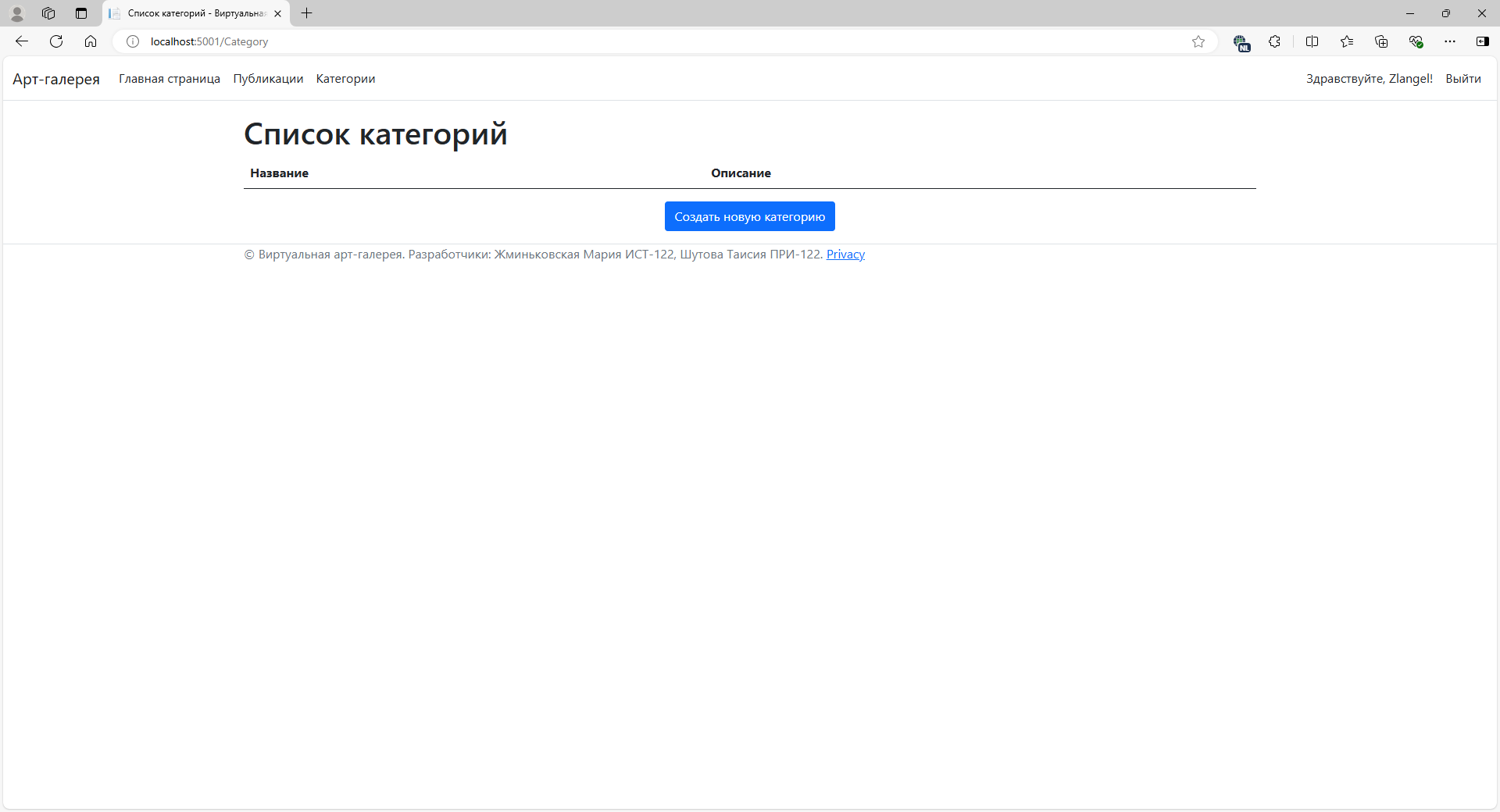


Рисунок 9. Категории

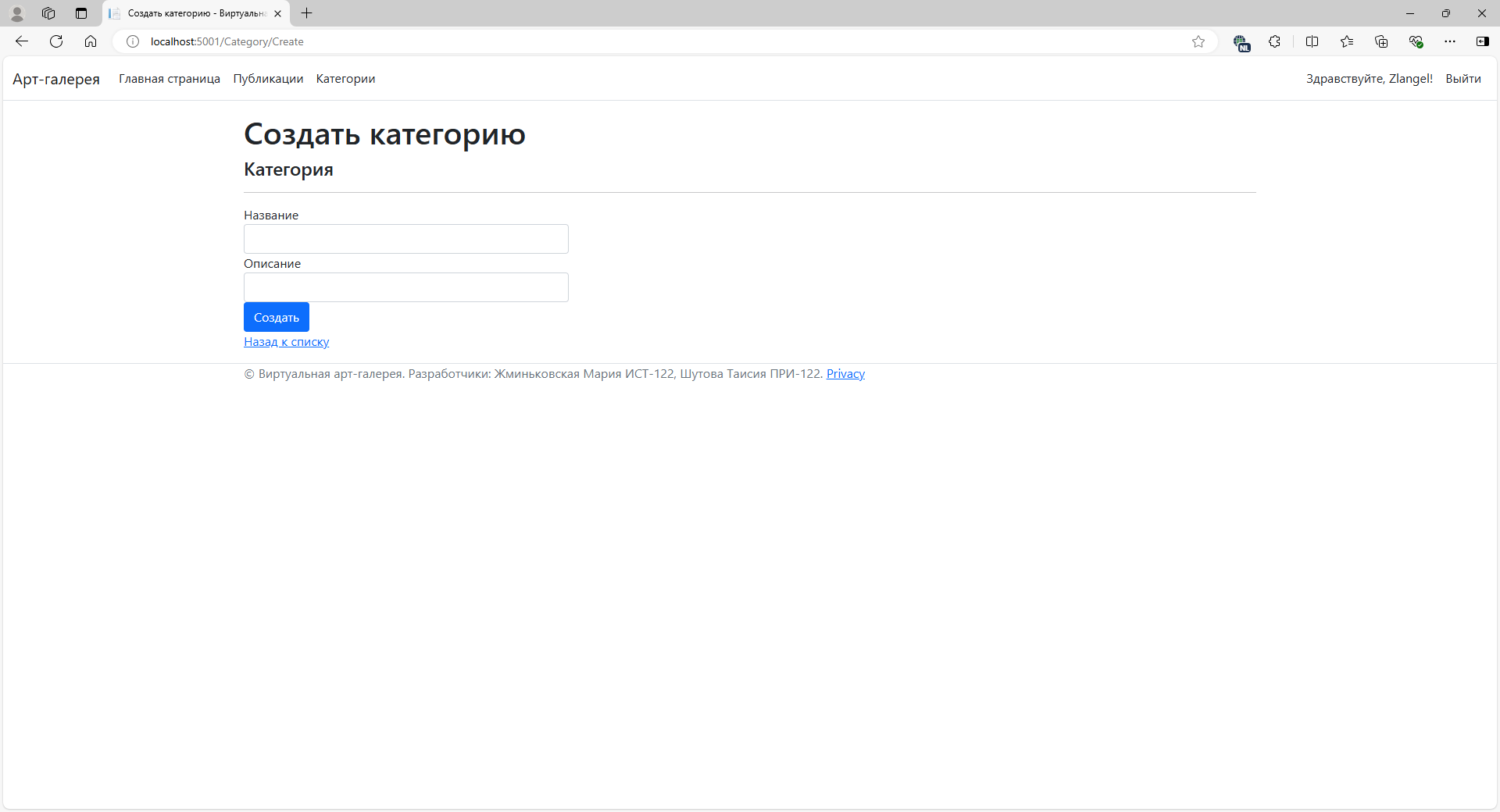


Рисунок 10. Окно создания категории

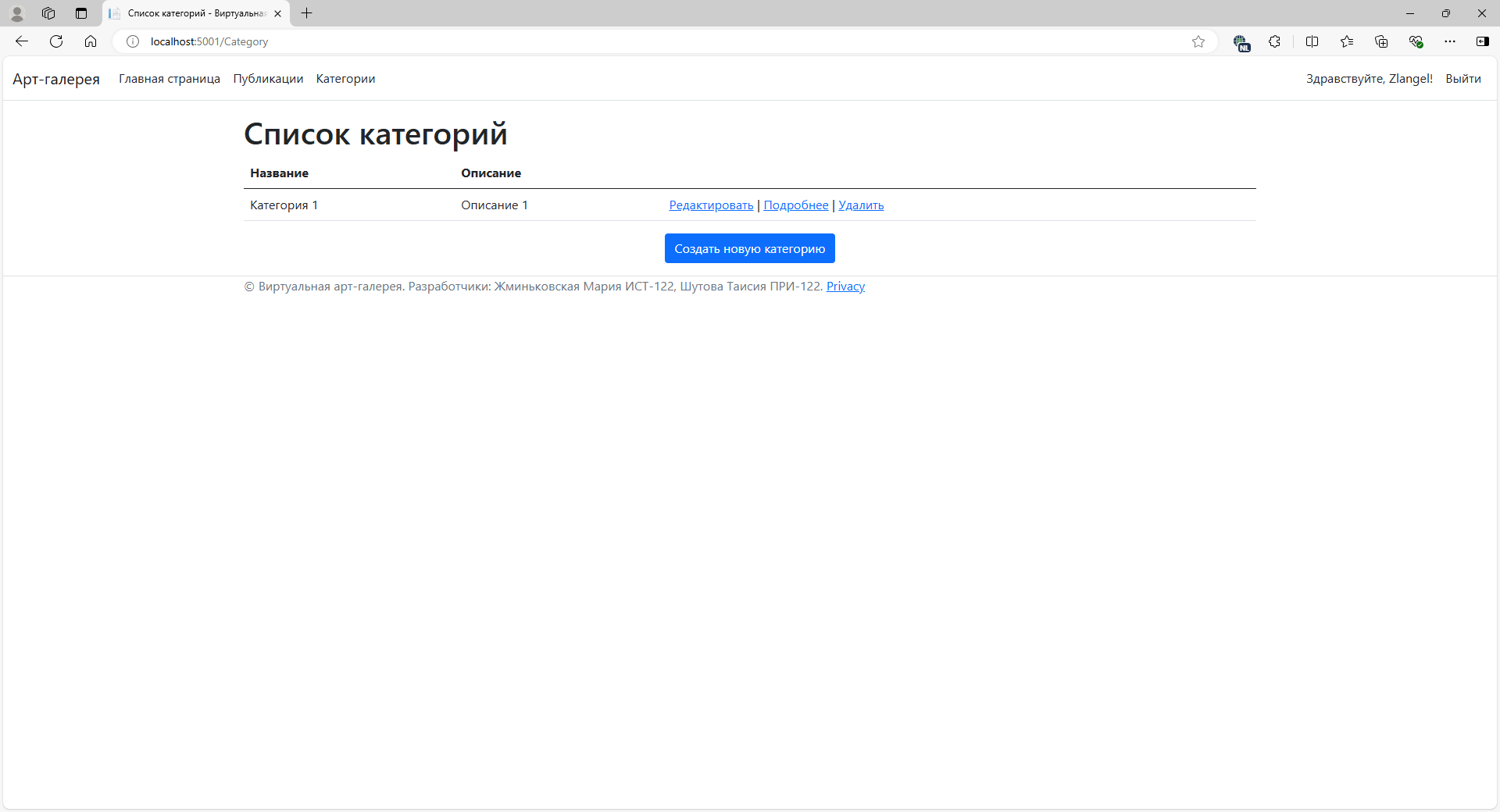


Рисунок 11. Окно списка категорий

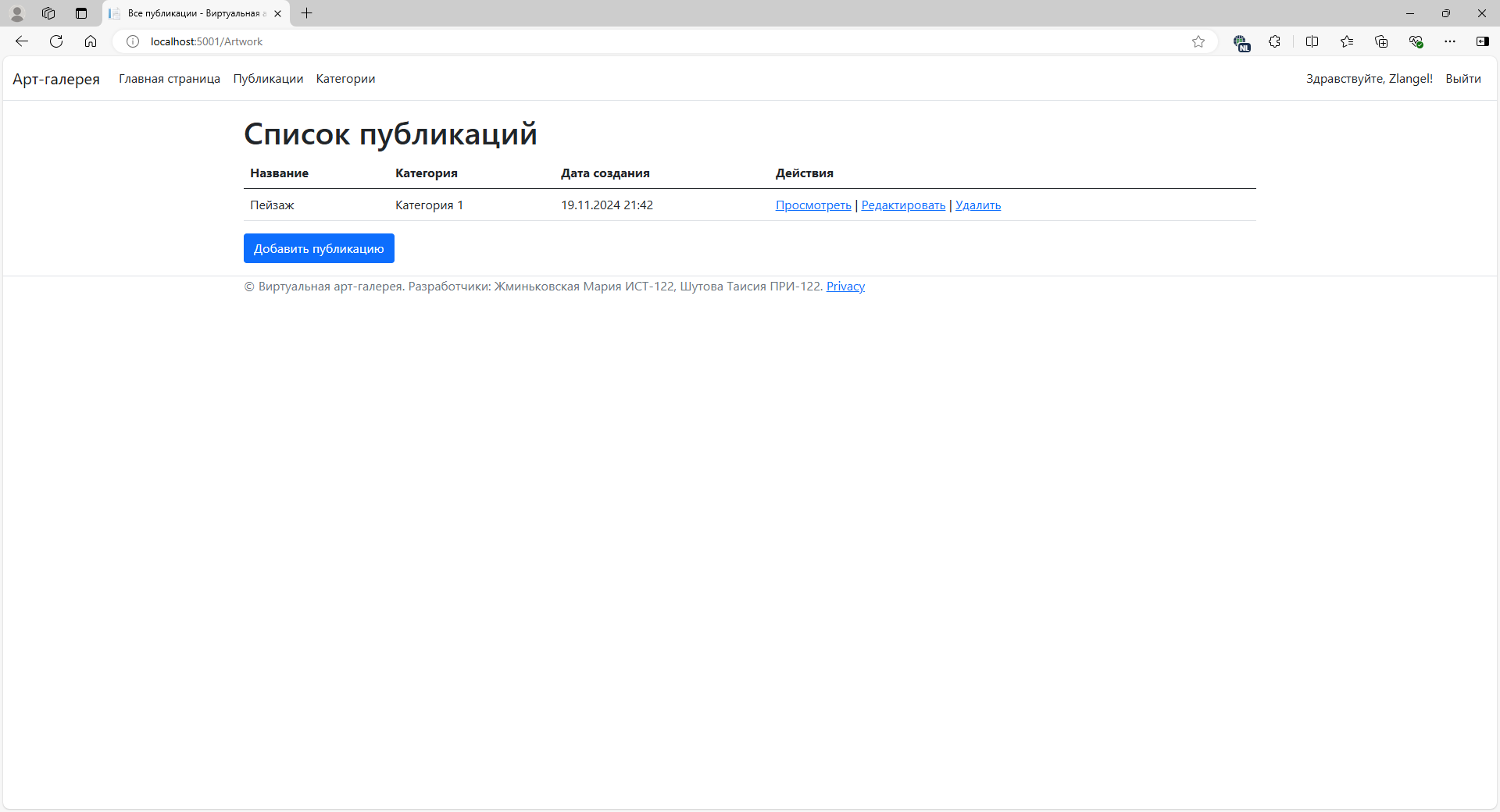


Рисунок 12. Окно списка публикаций

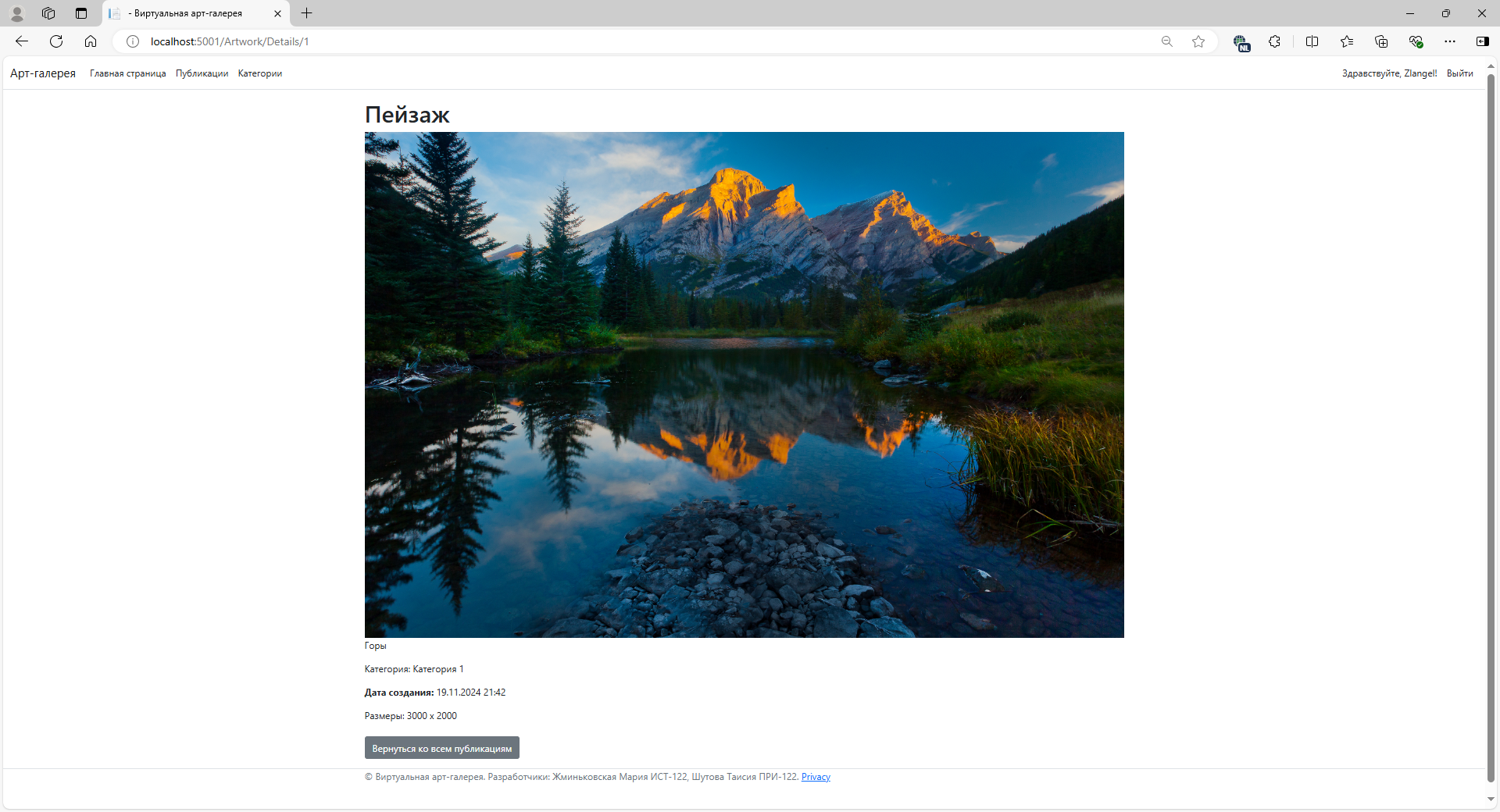


Рисунок 13. Окно просмотра публикации