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

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту по дисциплине**

**«Технологии программирования»**

Приложение для информационной системы «Виртуальная арт-галерея».

Модуль «Эксплуатация»

Выполнил: студент гр. ПРИ-122

Шутова Т.Е.

Принял: доц. Вершинин В.В.

Владимир, 2024

**АННОТАЦИЯ**

В данном курсовом проекте производилась разработка приложения для ИС «Виртуальная арт-галерея». Модуль «Эксплуатация». Пояснительная записка содержит разделы: введение, постановка задач, характеристики предметной области, проектирование информационной системы, реализация приложения для информационной системы, заключение, список используемой литературы, приложение А, приложение Б, приложение В.

Пояснительная записка содержит 49 листов, в том числе 31 рисунок, 9 использованных источников, 3 приложения.

**ANNOTATION**

In this course project, the development of an application for the IP "Virtual Art Gallery" was carried out. The module "Operation". The explanatory note contains the following sections: introduction, statement of tasks, characteristics of the subject area, information system design, implementation of an application for an information system, conclusion, list of literature used, Appendix A, Appendix B, Appendix B.

The explanatory note contains 49 sheets, including 31 figures, 9 sources used, 3 appendices.

**СОДЕРЖАНИЕ**

[1 ВВЕДЕНИЕ 4](#_Toc185550132)

[2 ПОСТАНОВКА ЗАДАЧ 5](#_Toc185550133)

[2.1 Цель и задачи 5](#_Toc185550134)

[3 ХАРАКТЕРИСТИКИ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc185550135)

[3.1 Описание предметной области 6](#_Toc185550136)

[3.2 Словарь предметной области 6](#_Toc185550137)

[4 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ 8](#_Toc185550138)

[4.1 Диаграмма прецедентов 8](#_Toc185550139)

[4.2. Описание прецедентов 8](#_Toc185550140)

[4.3 Диаграмма классов 10](#_Toc185550141)

[4.4 Диаграмма состояний 11](#_Toc185550142)

[4.5 Диаграмма последовательностей 13](#_Toc185550143)

[4.6 Диаграмма видов деятельности 15](#_Toc185550144)

[5 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ ДЛЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ 17](#_Toc185550145)

[5.1 Разработка базы данных 17](#_Toc185550146)

[5.2 Разработка веб-приложения 18](#_Toc185550147)

[5.3 Разработка приложения для информационной системы 18](#_Toc185550148)

[5.4 Диаграмма компонентов 20](#_Toc185550149)

[5.5 Диаграмма развёртывания 20](#_Toc185550150)

[5.6 Репозиторий проекта 21](#_Toc185550151)

[5.7 Реализация 21](#_Toc185550152)

[5.8 Страницы приложения 21](#_Toc185550153)

[5.9 Тестирование 32](#_Toc185550154)

[6 ЗАКЛЮЧЕНИЕ 33](#_Toc185550155)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 34](#_Toc185550156)

[ПРИЛОЖЕНИЕ А. КОНТРОЛЛЕР ВЫСТАВОК 35](#_Toc185550157)

[ПРИЛОЖЕНИЕ Б. КОНТРОЛЛЕР ПРАЙСОВ 40](#_Toc185550158)

[ПРИЛОЖЕНИЕ В. КОНТРОЛЛЕР ГАЛЕРЕИ 47](#_Toc185550159)

1 ВВЕДЕНИЕ

С развитием цифровых технологий и их активным внедрением в различные сферы жизни, искусство претерпевает значительные изменения. Одной из ключевых тенденций является переход к виртуальным форматам взаимодействия с произведениями искусства, что особенно актуально в условиях глобализации и развития удаленных коммуникаций. Создание виртуальных арт-галерей становится важным инструментом для популяризации искусства, расширения аудитории и предоставления пользователям доступа к культурному наследию, независимо от их географического положения.

В данной работе рассматривается процесс разработки виртуальной арт-галереи, представляющей собой программный продукт, созданный с использованием современных технологий. Главной целью проекта является обеспечение удобного и интерактивного доступа к публикациям. Разработка приложения будет осуществляться с использованием стека технологий, обеспечивающего высокую производительность, надежность и масштабируемость.

В рамках проекта будут проведены анализ требований к системе, проектирование архитектуры приложения, а также реализация его функционала, включающего управление публикациями, профилями художников. Приложение будет ориентировано на создание уникального пользовательского опыта, что сделает виртуальную арт-галерею важным инструментом для популяризации искусства в цифровую эпоху.

2 ПОСТАНОВКА ЗАДАЧ

# 2.1 Цель и задачи

Цель работы: разработать программную информационную систему «Виртуальная арт-галерея» с функционалом, связанным преимущественно с областью эксплуатации.

Для достижения поставленной цели необходимо реализовать следующие функциональные требования:

* Авторизация и регистрация (администратор, художник, посетитель);
* Система поиска публикаций, выставок, художников (администратор, художник, посетитель);
* Добавление прайс-листа и возможность заказа услуги (администратор, художник)
* Отображение публикаций, выставок и прайс-листа в профиле художника (администратор, художник);
* Публикация работ (администратор, художник);
* Организация приватных и открытых выставок и их редактирование (администратор, художник);
* Участие в открытых и закрытых выставках (администратор, художник);
* Безопасное удаление ненужных или некорректных записей (администратор)

А также необходимо решить следующие задачи:

* Анализ предметной области.
* Проектирование системы.
* Проектирование базы данных.
* Реализация системы и интеграция базы данных
* Тестирование и доработка системы
* Составление пояснительной записки.

3 ХАРАКТЕРИСТИКИ ПРЕДМЕТНОЙ ОБЛАСТИ

# 3.1 Описание предметной 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.

# 3.2 Словарь предметной области

*Посетитель* – незарегистрированный или неавторизованный пользователь. Имеет минимальные права, может просматривать работы, художников и выставки и пользоваться поиском. Также он может авторизоваться или зарегистрироваться.

Атрибуты: нет.

*Художник* – зарегистрированный и прошедший авторизацию пользователь. Художнику не обязательно иметь в своём профиле работы, это только роль, присваиваемая при регистрации. Художник имеет те же возможности, что и посетитель, а также он может публиковать работы, устраивать выставки, создавать прайс-листы, заказывать услуги, участвовать в выставках, распоряжаться своим профилем и управлять собственными записями.

Атрибуты: ID, никнейм, пароль, почта.

*Администратор* – зарегистрированный пользователь, имеющий доступ к редактированию содержимого чужих публикаций, чужих выставок и художников. Администратор следит за соблюдением правил платформы, удаляет нарушения, принимает или отклоняет публикации, а также может позиционировать себя как художник и обладает всеми теми же функциями.

Атрибуты: ID, никнейм, пароль, почта.

*Публикация, или Работа, или Картина* – художественная работа определённого автора, располагающаяся либо внутри, либо вне выставки.

Атрибуты: ID, название, описание, изображение, статус публикации, дата создания, категория, автор, привязка к выставке, размеры изображения, количество лайков, количество просмотров.

*Прайс-лист или Прайс* – общедоступное описание возможных услуг, которые художник готов предоставить, и примерная цена за них.

Атрибуты: ID, название, описание, изображение услуги, цена, художник.

*Выставка* – собранная по определенному критерию коллекция работ художника или художников, доступная для редактирования заранее указанному кругу лиц.

Атрибуты: ID, название, описание, приватность, публикации, создатель.

*Профиль художника или Профиль пользователя* – страница отображения информации о художнике, его публикациях, выставках и прайс-листах.

Атрибуты: ID, никнейм, почта, публикации, выставки, прайс-листы.

*Галерея* – главная страница системы, на которой отображается самое необходимое для среднестатистического пользователя: публикации, художники, выставки, а также удобный поиск.

Атрибуты: Публикации, художники, выставки, категория для поиска и ключевые слова для поиска.

4 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

# 4.1 Диаграмма прецедентов

Диаграмма прецедентов представлена на рисунке 1. Прямоугольник с пунктирными краями выделяет функции, реализованные в рамках модуля, с краями пунктира с точкой – в рамках общей информационной системы.

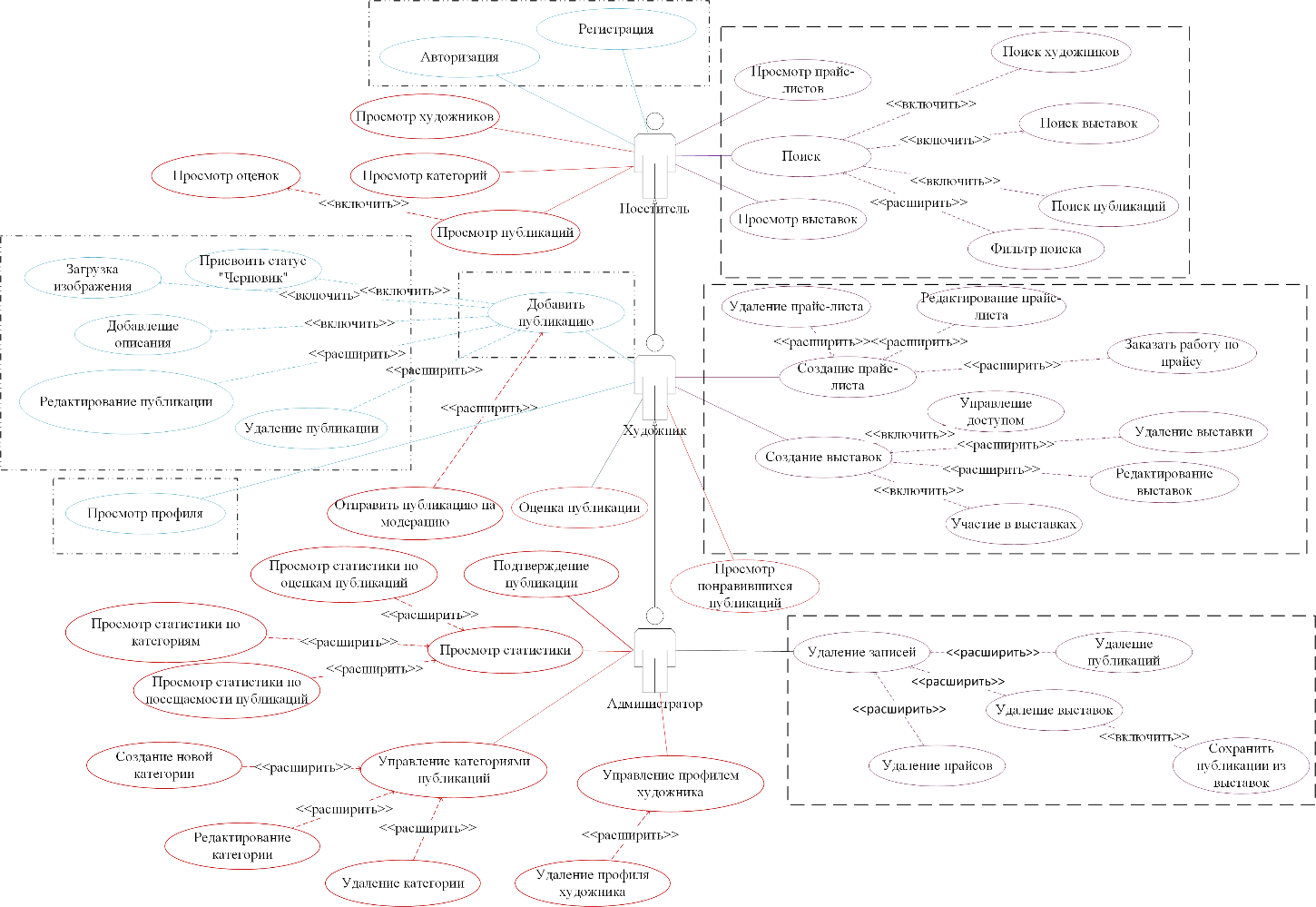


Рисунок 1. Диаграмма прецедентов

# 4.2. Описание прецедентов

|  |
| --- |
| Расширенное описание прецедента «Добавить публикацию» |
| **Название:** «Добавить публикацию»  **Предусловие:** нет  **Действующее лицо:** художник, администратор  **Основной поток:** *Добавить публикацию*  Пользователь нажимает на кнопку «Добавить публикацию» и переходит в форму для публикации.  Обязательно нужно загрузить изображение и выбрать категорию. По желанию можно оформить описание.  Работе присваивается статус «Черновик», она становится доступной для просмотра только создателю публикации.  **Альтернативный поток:** *нет*  **Постусловие:** пользователя автоматически перебрасывает на страницу со всеми публикациями. |

|  |
| --- |
| Расширенное описание прецедента «Поиск» |
| **Название:** Поиск  **Предусловие:** нет  **Действующее лицо:** посетитель, художник, администратор  **Основной поток:** *Поиск*  Пользователь вбивает в строку поиска в навигационной панели или в галерее то, что ему необходимо найти.  На главной странице можно выбрать категорию для поиска, тогда поиск будет осуществляться среди публикаций. Если категории не были выбраны, поиск осуществляется среди публикаций, художников и выставок.  После поиска внутри системы пользователю отображаются все найденные данные.  **Альтернативный поток:** *Ничего не найдено*  В том случае, если среди художников, публикаций и/или выставок ничего не было найдено, выводится «Нет художников», «Нет публикаций» и «Нет выставок» соответственно.  **Постусловие:** галерея обновляется с условием поиска. |

|  |
| --- |
| Расширенное описание прецедента «Создание прайс-листа» |
| **Название:** Создание прайс-листа  **Предусловие:** нет  **Действующее лицо:** художник, администратор  **Основной поток:** *Создание прайс-листа*  Пользователь нажимает на кнопку «Добавить прайс», после чего его перебрасывает на форму заполнения.  После заполнения всех полей, прайс-лист автоматически публикуется в профиле художника.  **Альтернативный поток:** *нет*  **Постусловие:** после создания прайс-листа пользователя перебрасывает в свой профиль, где можно посмотреть прайс. |

|  |
| --- |
| Расширенное описание прецедента «Создание выставок» |
| **Название:** Создание выставок  **Предусловие:** нет  **Действующее лицо:** художник, администратор  **Основной поток:** *Создание выставок*  Пользователь нажимает на кнопку «Создать выставку», после чего его перебрасывает на страницу с заполняемой формой.  В выставке заполняются все необходимые поля и присваивается приватность или открытость.  Пустая выставка автоматически добавляется на сайт и в профиль художника.  **Альтернативный поток:** *нет*  **Постусловие:** пользователя автоматически перебрасывает на страницу со всеми выставками. |

# 4.3 Диаграмма классов

Диаграмма классов [1] представлена на рисунке 2.

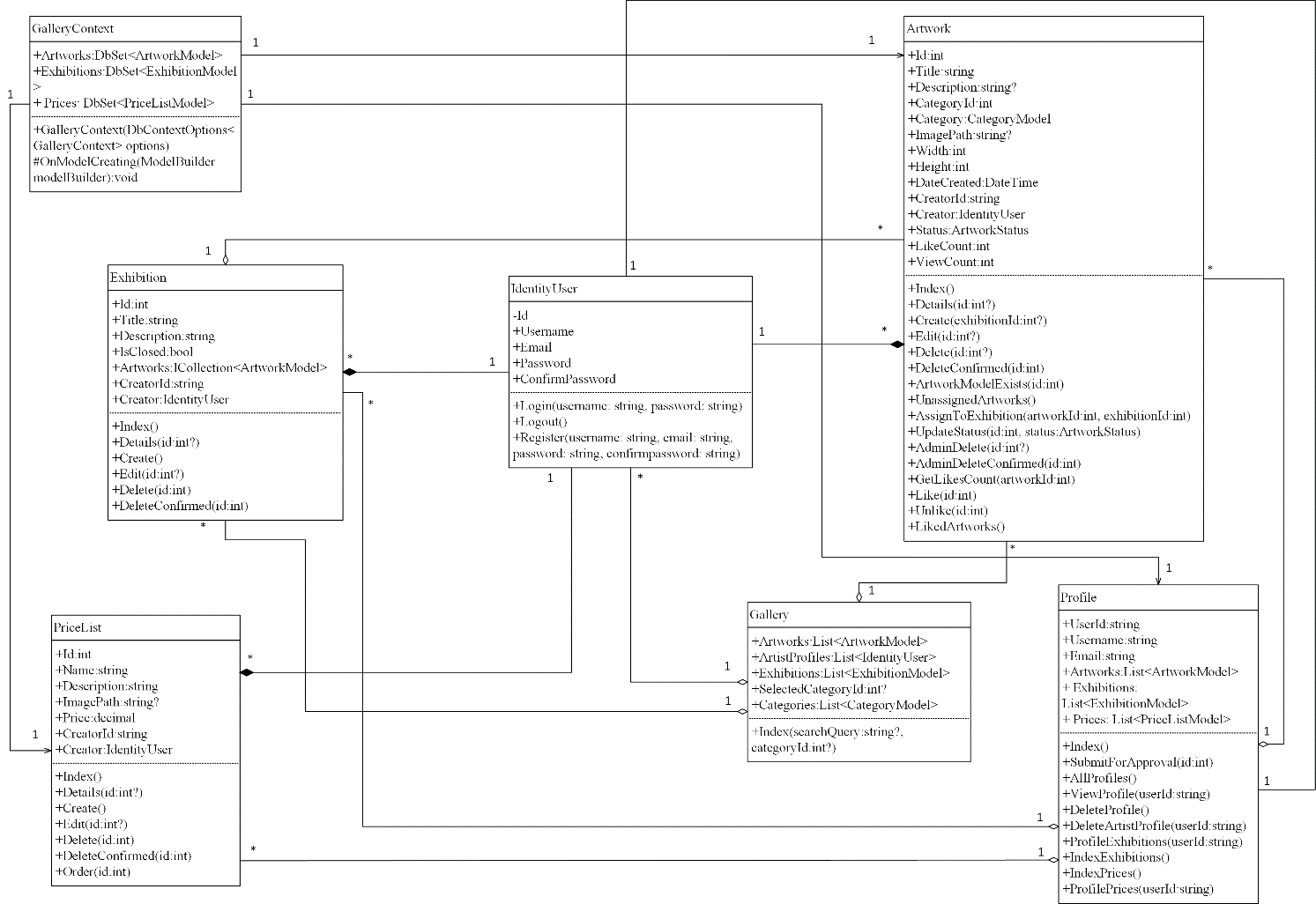


Рисунок 2. Диаграмма классов

# 4.4 Диаграмма состояний

Диаграмма состояний для объекта «Выставка» представлена на рисунке 3.

**Описание диаграммы состояний для объекта «Выставка»**

**1) Данные о выставке заполнены**

Как для создания выставки, так и для редактирования, пользователю открывается форма заполнения. После введения всех нужных данных выставка создаётся/обновляется после нажатия на соответствующую кнопку

**2) Редактирование выставки**

Если необходимо внести изменения в названии, описании или приватности выставки, создатель выставки может сделать это в любой момент.

**3) Публикация создана, получает статус «Черновик»**

Если выставка приватна, публикации в неё может добавлять только создатель выставки. Если открыта – любой авторизованный пользователь. Как и все публикации, прежде чем публикация станет видна всем пользователям, она должна пройти модерацию

**4) Публикация со статусом «Черновик» редактируется**

Пока публикация не отправлена на модерацию, её можно редактировать по своему усмотрению.

**5) Публикация на модерации, получает статус «На модерации»**

Пользователь отправляет публикацию на модерацию, её всё ещё может видеть только создатель публикации.

**6) Работа получает статус «Подтверждена» и отображается в выставке**

Если администратор одобрил работы, выставка обновляется, в ней появляется новая работа, доступная для просмотра всем пользователям. В выставку можно добавлять новые работы неограниченное количество раз.

**7) Работа получает статус «Отклонено» и не отображается в выставке**

Администратор посчитал, что публикация не подходит правилам сайта, поэтому отклоняет их. В таком случае отклонённая работа видна только создателю публикации и её остаётся лишь удалить.

**8) Выставка функционирует (цикл)**

Пока выставка не удалена, её можно редактировать и в неё можно добавлять сколько угодно новых публикаций.

**9) Подтверждение удаления**

Когда создатель выставки или администратор решает, что выставка должна быть удалена, после подтверждения о своём решении выставка удаляется, однако работы остаются в профиле пользователей, но те уже не привязаны к выставке.

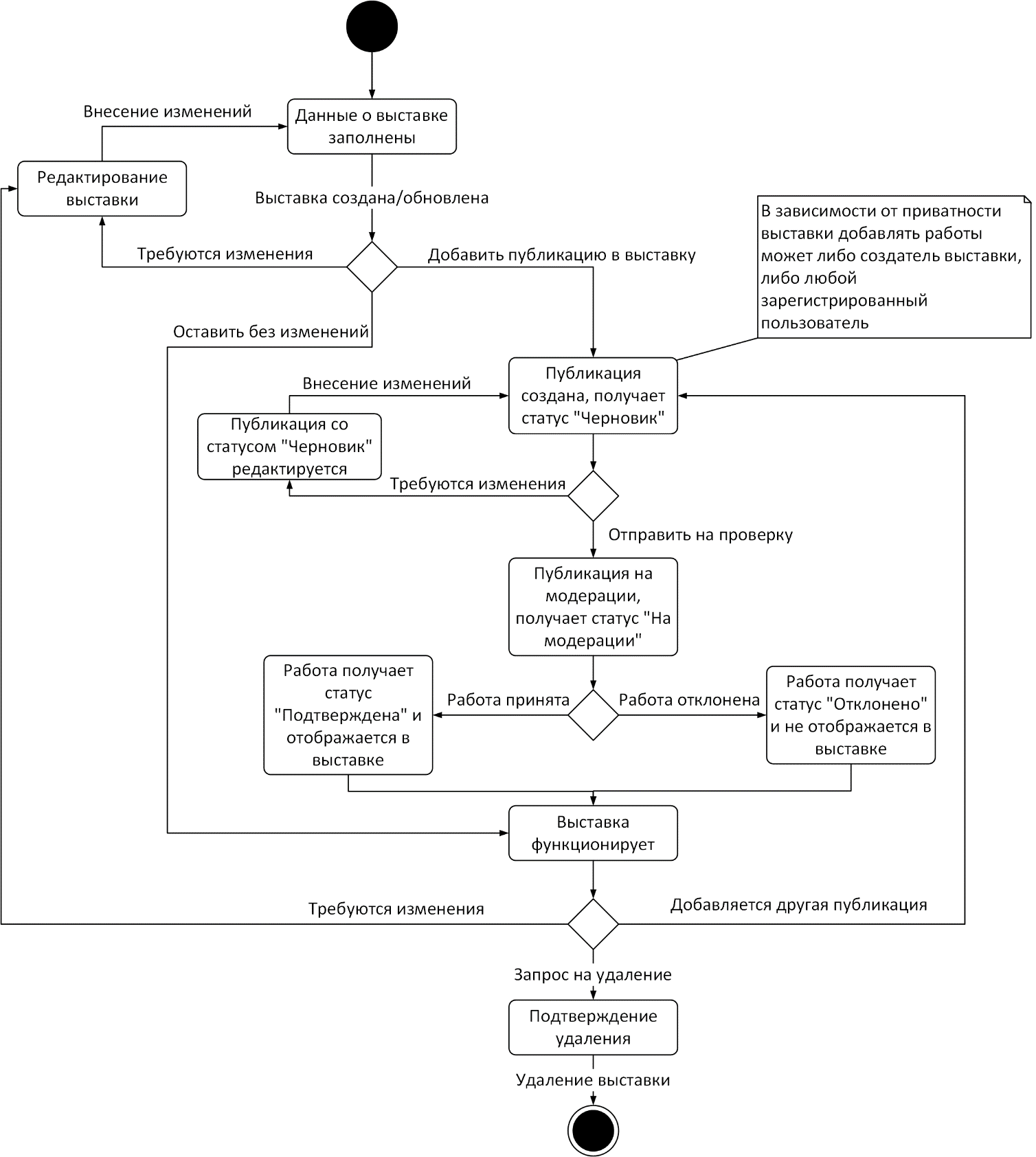


Рисунок 3. Диаграмма состояний

# 4.5 Диаграмма последовательностей

Диаграмма последовательностей для объекта «Выставка» представлена на рисунке 4.

**Описание диаграммы последовательностей объекта «Выставка».**

1. Художник делает запрос на создание выставки.

2. Выставка создаётся и автоматически публикуется.

3. Художник заходит в детали выставки.

4. Если художнику нужно добавить публикацию, выставка обращается к созданию публикации.

5. Публикация отправляется на рассмотрение администратору.

6. Если администратор одобряет публикацию, она становится доступна для просмотра всем пользователям.

7. Публикация добавляется в выставку.

8. Выставка обновлена.

9. Те же операции по добавлению публикаций могут проделывать другие пользователи, если выставка открыта.

10. Если художнику нужно удалить выставку, он делает запрос на удаление.

11. Публикации в выставке безопасно открепляются от неё.

12. Выставка удаляется.

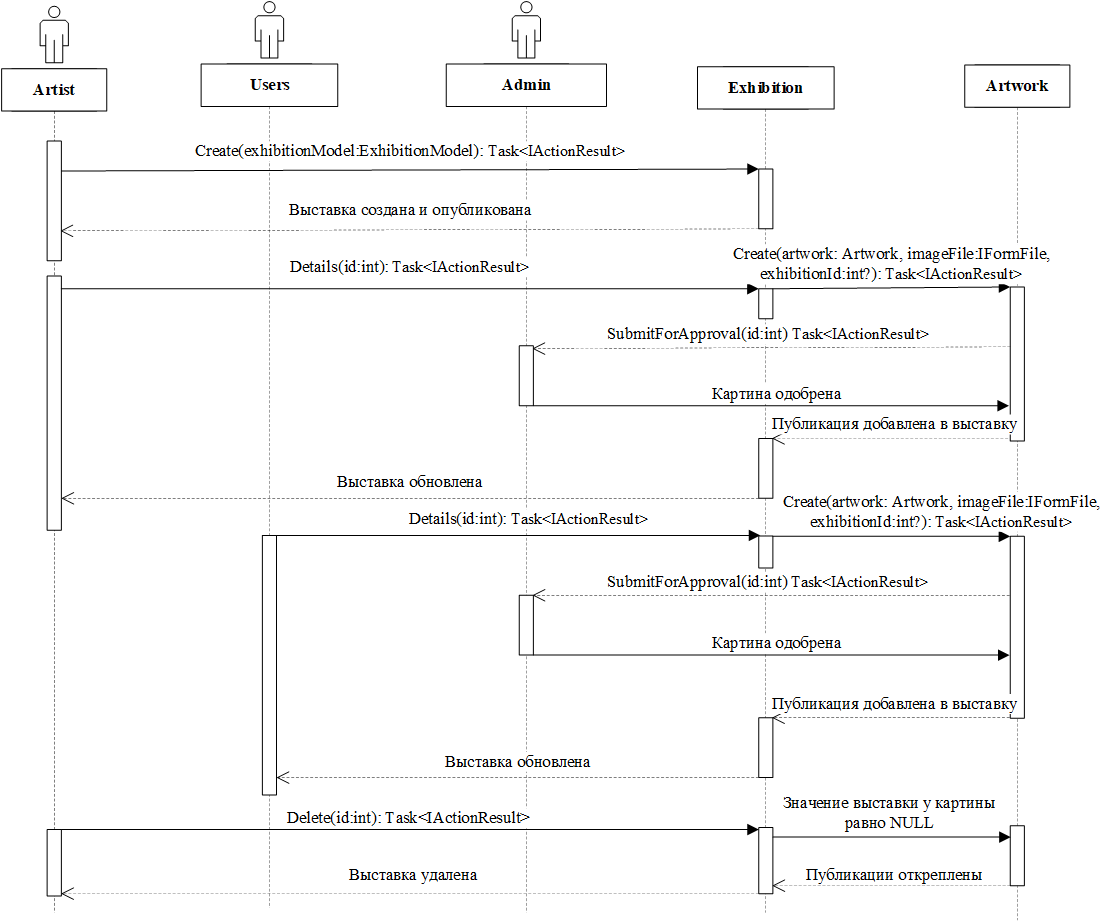


Рисунок 4. Диаграмма последовательностей

# 4.6 Диаграмма видов деятельности

Диаграмма видов деятельности [2] для объекта «Выставка» представлена на рисунке 5.

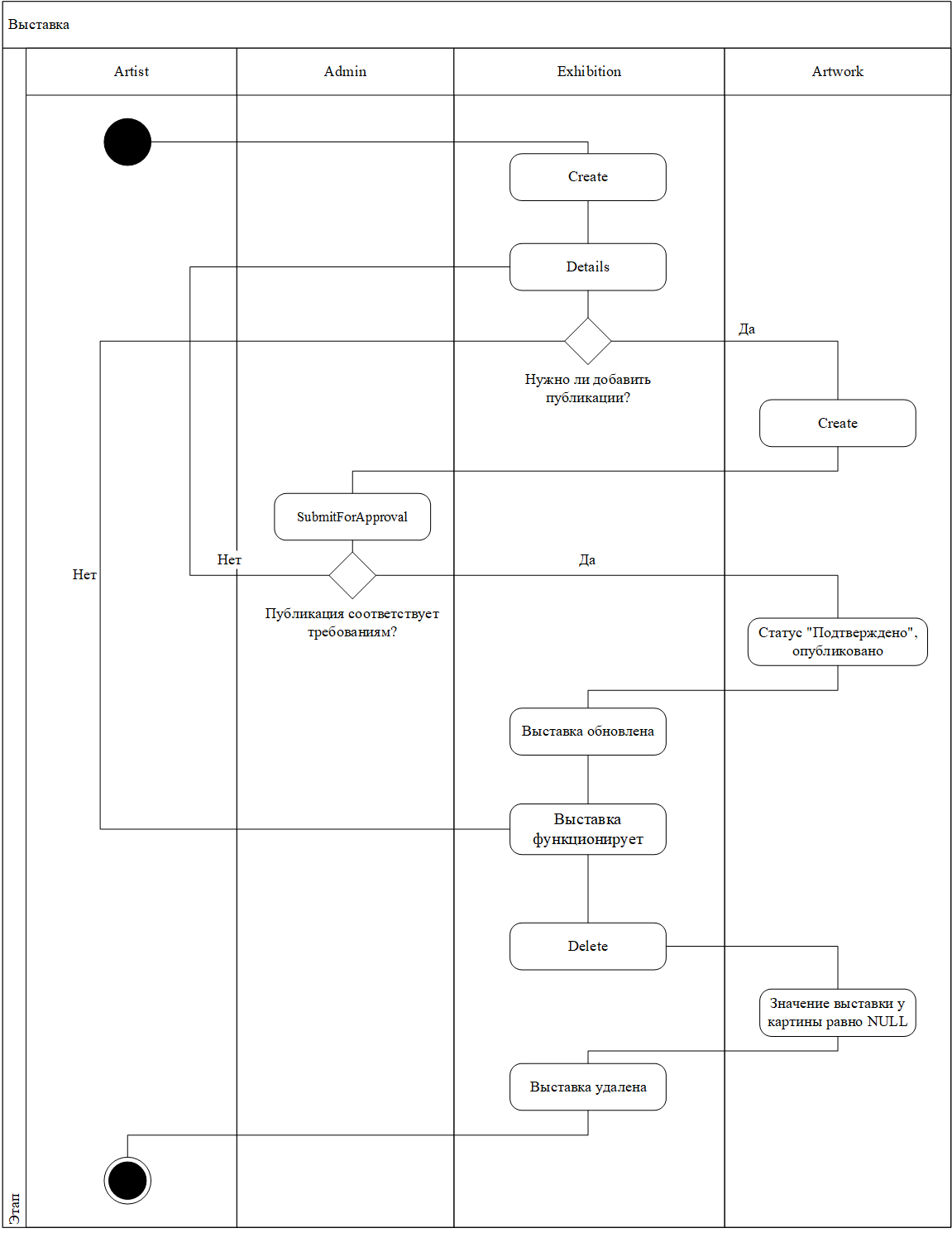


Рисунок 5. Диаграмма видов деятельности

5 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ ДЛЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

# 5.1 Разработка базы данных

Для разработки системы управления базами данных была выбрана технология PostgreSQL [3]. Был выбран подход Database-first. На рисунке 6 предствлена ER-диаграмма.

На диаграмма представленно 13 таблиц из них 7 (AspNetUsers, AspNetRoles, AspNetUserLogins, AspNetUserTokens, AspNetRoleClaims, AspNetUserRoles, AspNetUserClaims) были созданы пакетом Microsoft.AspNetCore.Identity.EntityFrameworkCore для работы с авторизацией и регистрацией. \_\_EFMigrationsHistory – история миграций. Таблицы Likes, Prices, Exhibitions, Artworks и Categories явяляются разработанными функциональными таблицами.

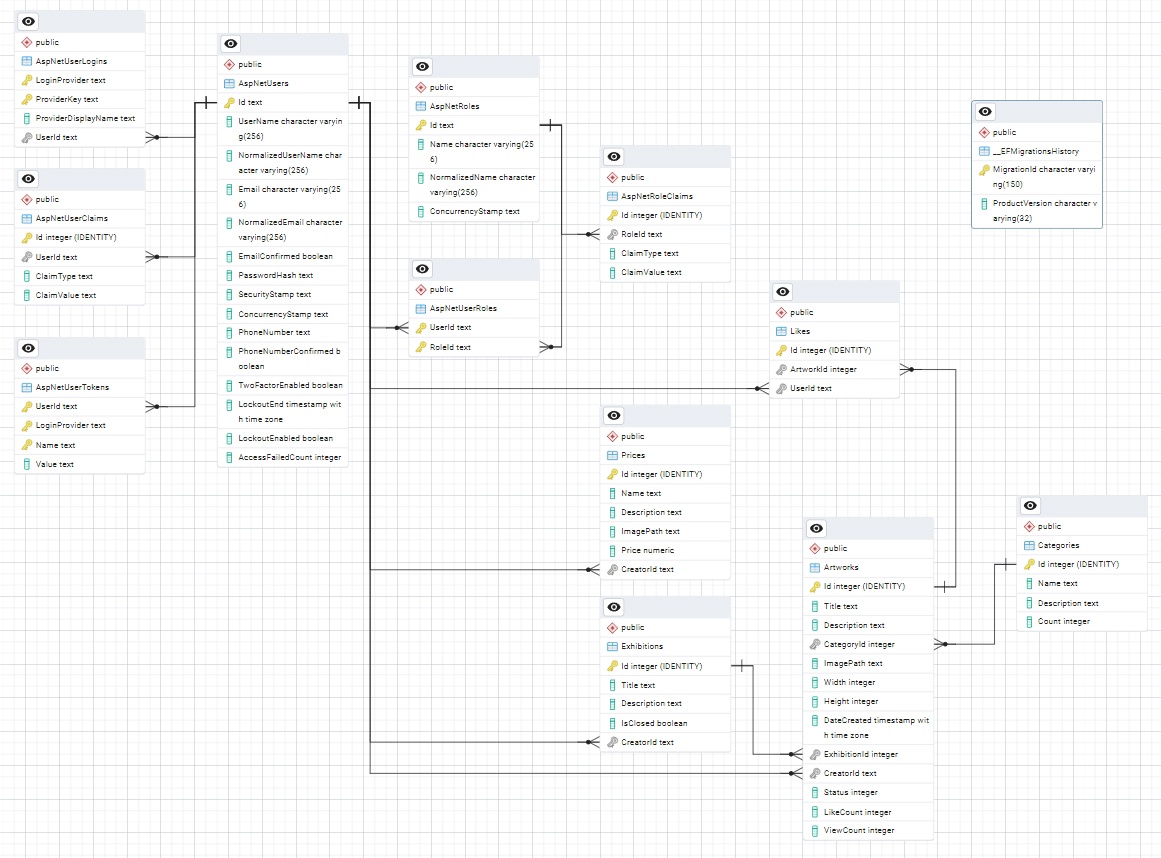


Рисунок 6. ER-диаграмма

# 5.2 Разработка веб-приложения

Веб-приложение было написано на С# .Net. Использовались технологии Core, Core MVC.

**Использованные NuGet-пакеты:**

* *Microsoft.AspNetCore.Identity.EntityFrameworkCore* – позволяет использовать Entity Framework Core в связке с ASP.NET Core Identity для хранения и управления учетными записями пользователей и ролями в базе данных.
* *Microsoft.AspNetCore.Identity.UI* – предоставляет предустановленные пользовательские интерфейсы (UI) для ASP.NET Core Identity, включая страницы регистрации, входа, восстановления пароля и управления профилем.
* *Microsoft.EntityFrameworkCore* – основной пакет для Entity Framework Core. Обеспечивает функциональность ORM (объектно-реляционное отображение) для работы с базами данных.
* *Microsoft.EntityFrameworkCore.Tools* – инструменты для работы с Entity Framework Core, такие как миграции, генерация моделей и управление базами данных через командную строку или интерфейс IDE (например, Visual Studio).
* *Microsoft.EntityFrameworkCore.SqlServer* – провайдер базы данных для Entity Framework Core, позволяющий работать с SQL Server.
* *Npgsql.EntityFrameworkCore.PostgreSQL* – провайдер базы данных для Entity Framework Core, позволяющий работать с PostgreSQL.
* *SixLabors.ImageSharp* – библиотека для работы с изображениями. Предоставляет функционал для обработки, создания и преобразования изображений в .NET приложениях.
* *Selenium.WebDriver.GeckoDriver* – библиотека для тестирования Selenium с помощью браузера Firefox.
* *NUnit3TestAdapter* – адаптер для тестов NUnit в Visual Studio 2012 года и выше.
* *NUnit* – библиотека для тестирования приложений.

# 5.3 Разработка приложения для информационной системы

Приложение было разработано с помощью ASP.NET MVC [4][5]. В приложении для данного модуля были созданы и отредактированы следующие элементы:

**Модели:**

1. ArtworkModel – модель для хранения публикаций.
2. ExhibitionModel – модель для хранения выставок.
3. GalleryIndexViewModel – модель для просмотра галереи.
4. PriceListModel – модель для хранения прайсов.
5. ProfileViewModel – модель для просмотра профилей.

**Представления:**

1. Artwork
   1. Create – создание публикации.
   2. Delete – удаление публикации.
   3. Details – детали публикации.
   4. Edit – редактирование публикации со статусом «Черновик».
   5. Index – просмотр всех публикаций.
2. Exhibition
   1. Create – создание выставки.
   2. Delete – удаление выставки.
   3. Details – детали выставки с возможностью добавить в неё публикацию.
   4. Edit – редактирование названия, описания и приватности выставки.
   5. Index – просмотр всех выставок
3. Gallery
   1. Index – просмотр галереи.
4. PriceList
   1. Create – создание прайс-листа.
   2. Delete – удаление прайс-листа.
   3. Details – детали прайс-листа с возможностью заказать услугу.
   4. Edit – редактирование прайс-листа.
   5. Index – просмотр всех прайс-листов (недоступно по прямой ссылке, т.к. не является объективно обязательной для арт-галереи).
5. Profile
   1. Index – просмотр публикаций пользователя.
   2. IndexExhibitions – просмотр выставок пользователя.
   3. IndexPrices – просмотр прайсов пользователя.
   4. ProfileExhibitions – просмотр выставок художника.
   5. ProfilePrices – просмотр прайсов художника
   6. ViewProfile – просмотр публикаций художника.

**Контроллеры:**

1. ArtworkController – контроллер для работы с публикациями.
2. ExhibitionController – контроллер для работы с выставками.
3. GalleryController – контроллер с реализацией отображения публикаций, художников и выставок, а также их поиск [6].
4. PriceListController – контроллер для работы с прайсами.
5. ProfileController – контроллер для работы с профилями пользователей.

# 5.4 Диаграмма компонентов

Диаграмма компонентов [7] представлена на рисунке 7.

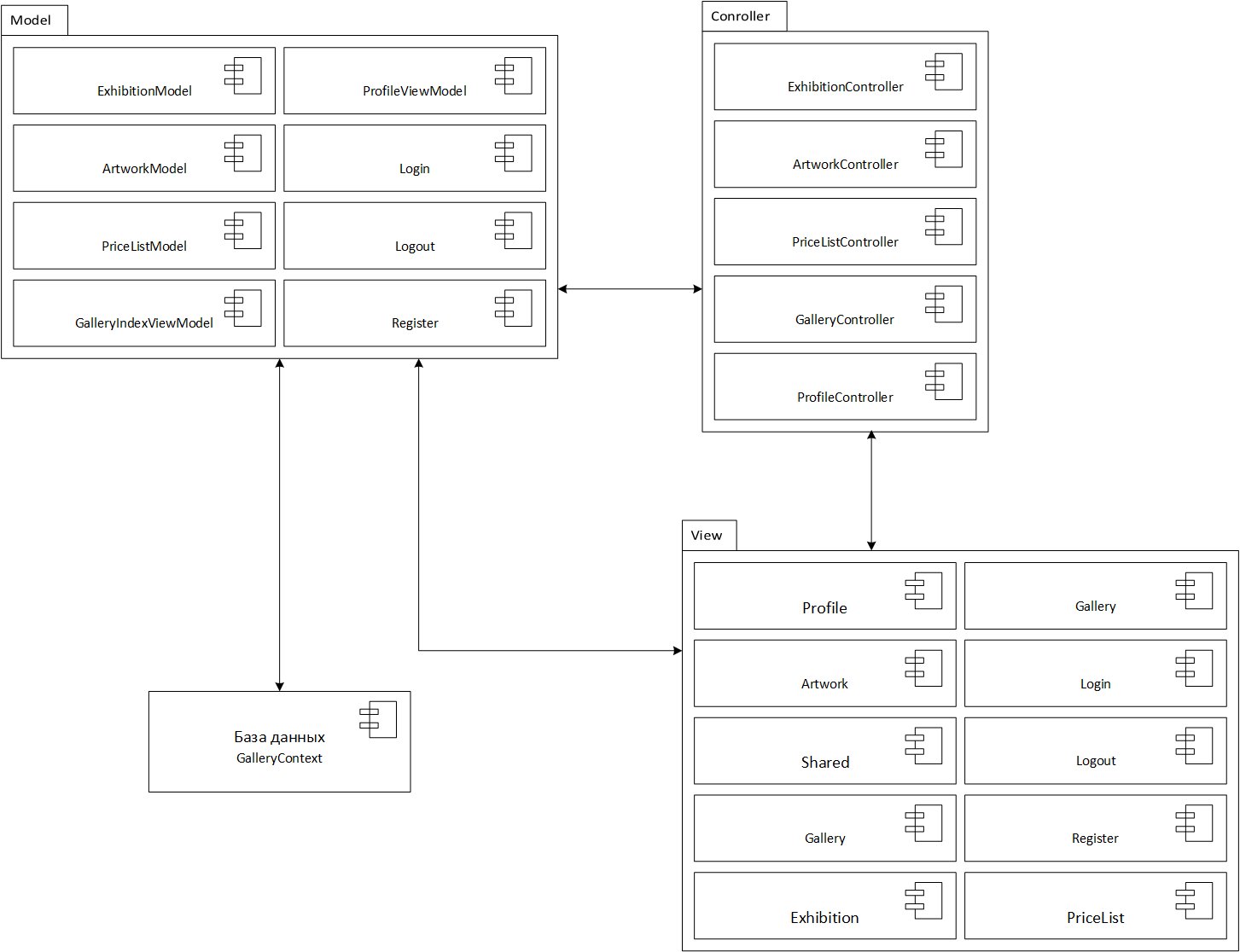


Рисунок 7. Диаграмма компонентов

# 5.5 Диаграмма развёртывания

Диаграмма развёртывания [8] представлена на рисунке 8.

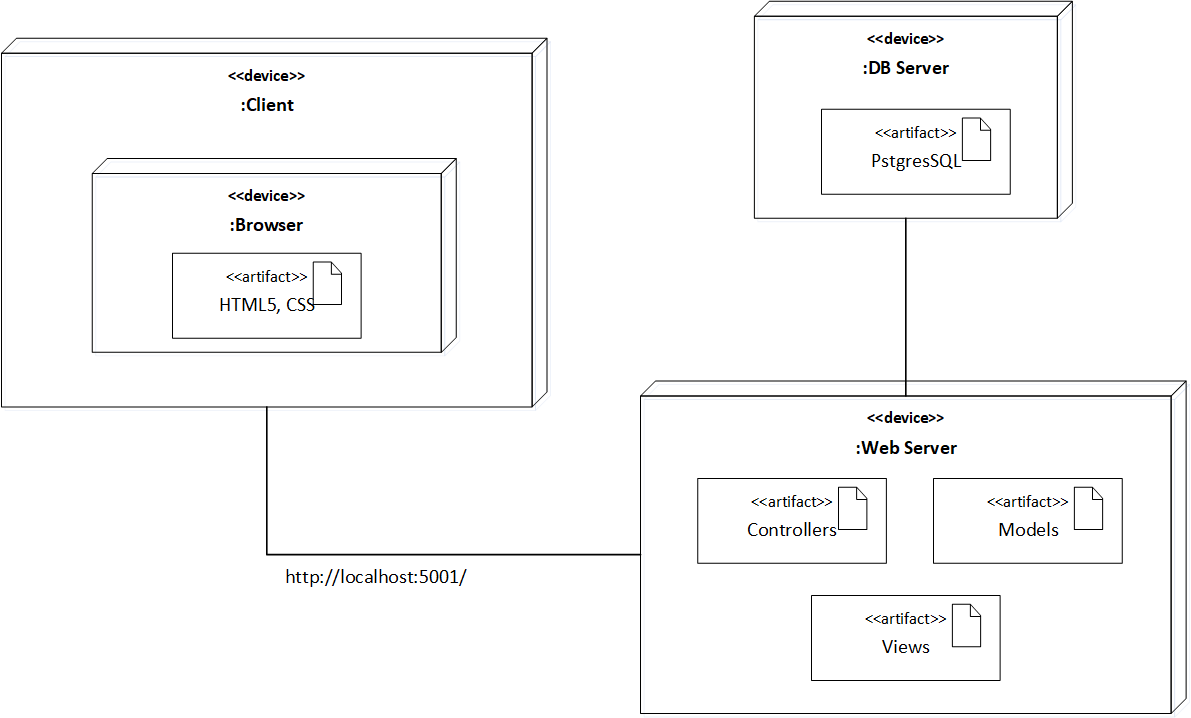


Рисунок 8. Диаграмма развёртывания

# 5.6 Репозиторий проекта

Ссылка на репозиторий проекта: <https://github.com/ShutovaTE/Virtual-Art-Gallery>

# 5.7 Реализация

Контроллеры выставок, прайс-листов и галереи представлены в приложениях А, Б и В соответственно.

# 5.8 Страницы приложения

Страницы готового приложения представлены на рисунках 9-30.

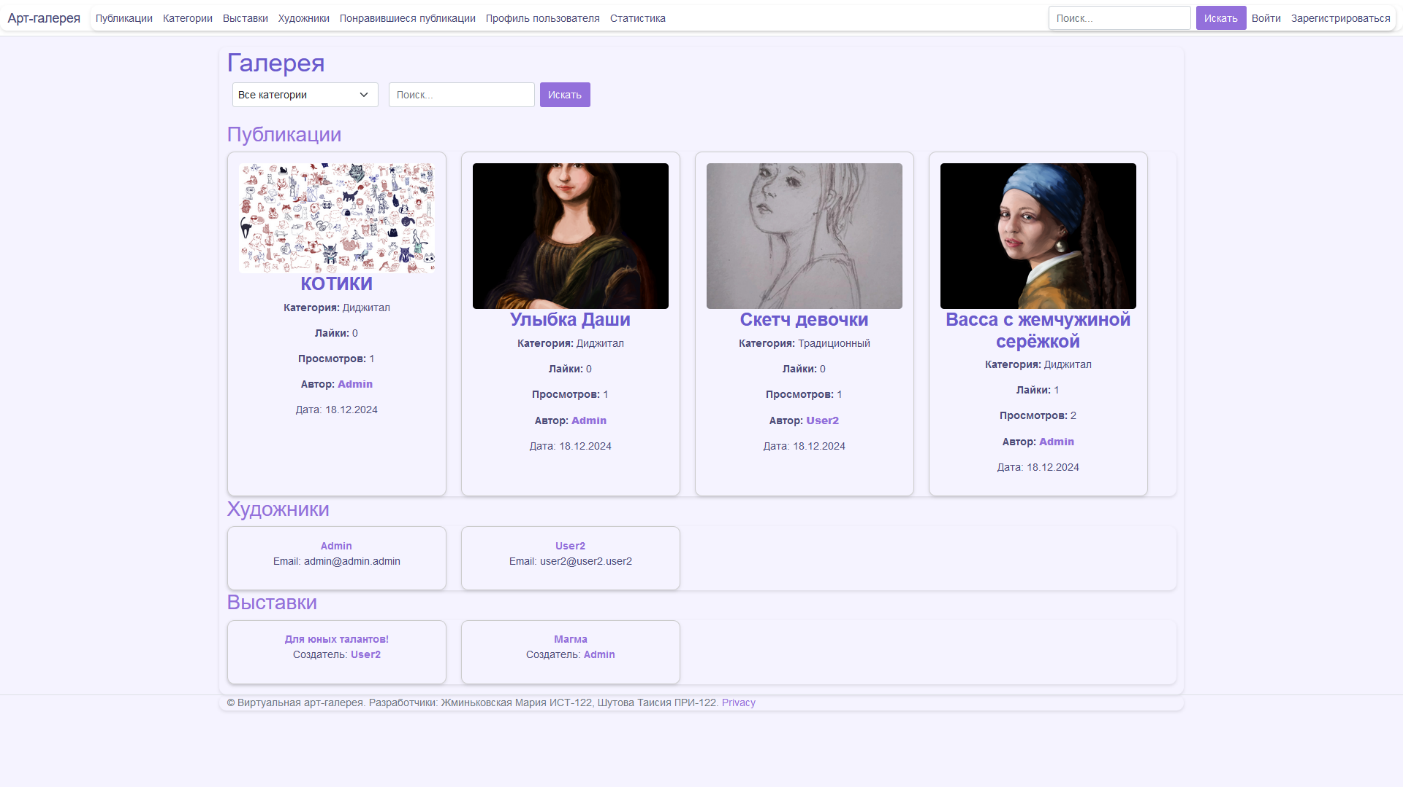


Рисунок 9. Главная страница

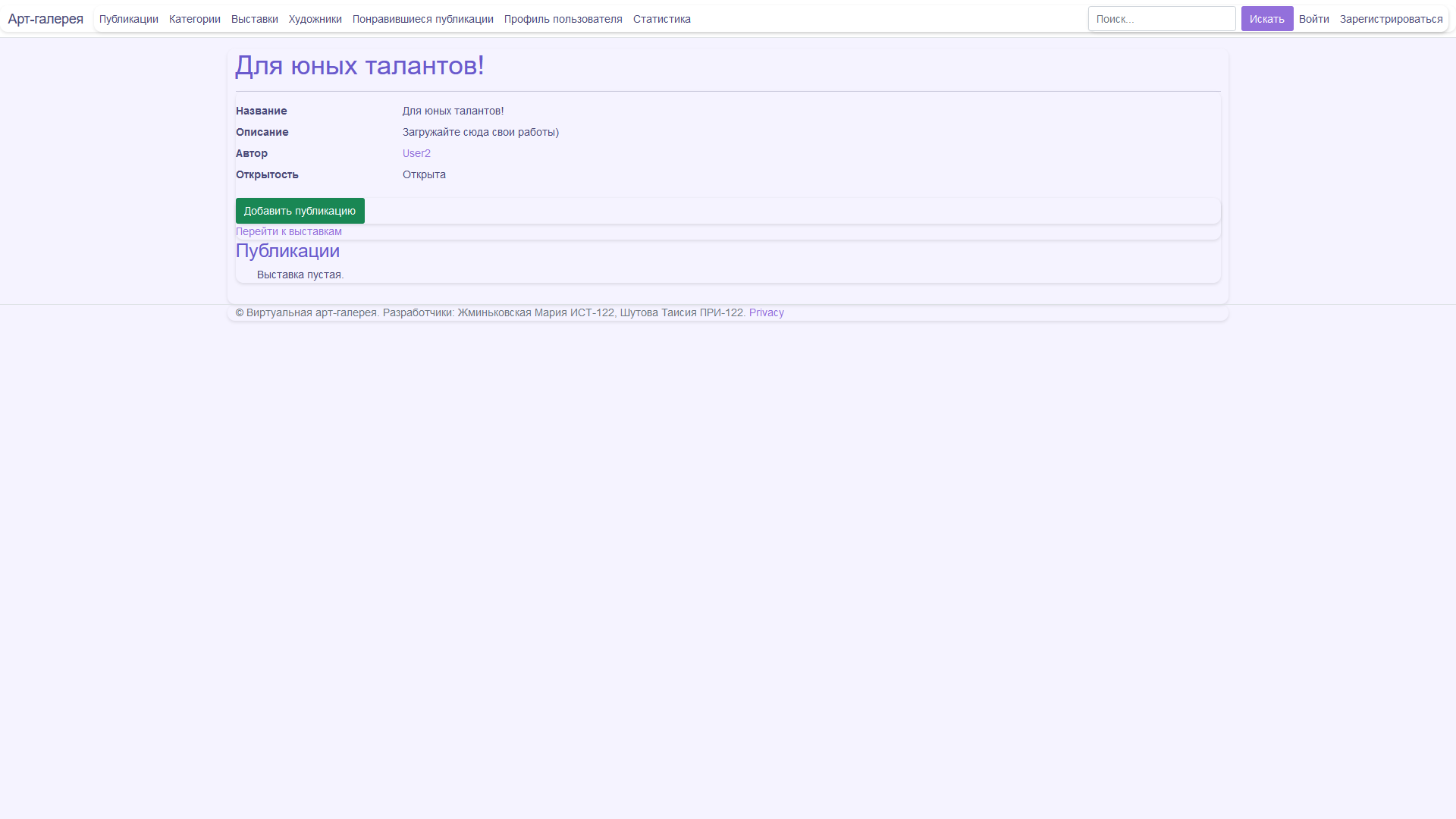


Рисунок 10. Открытая выставка

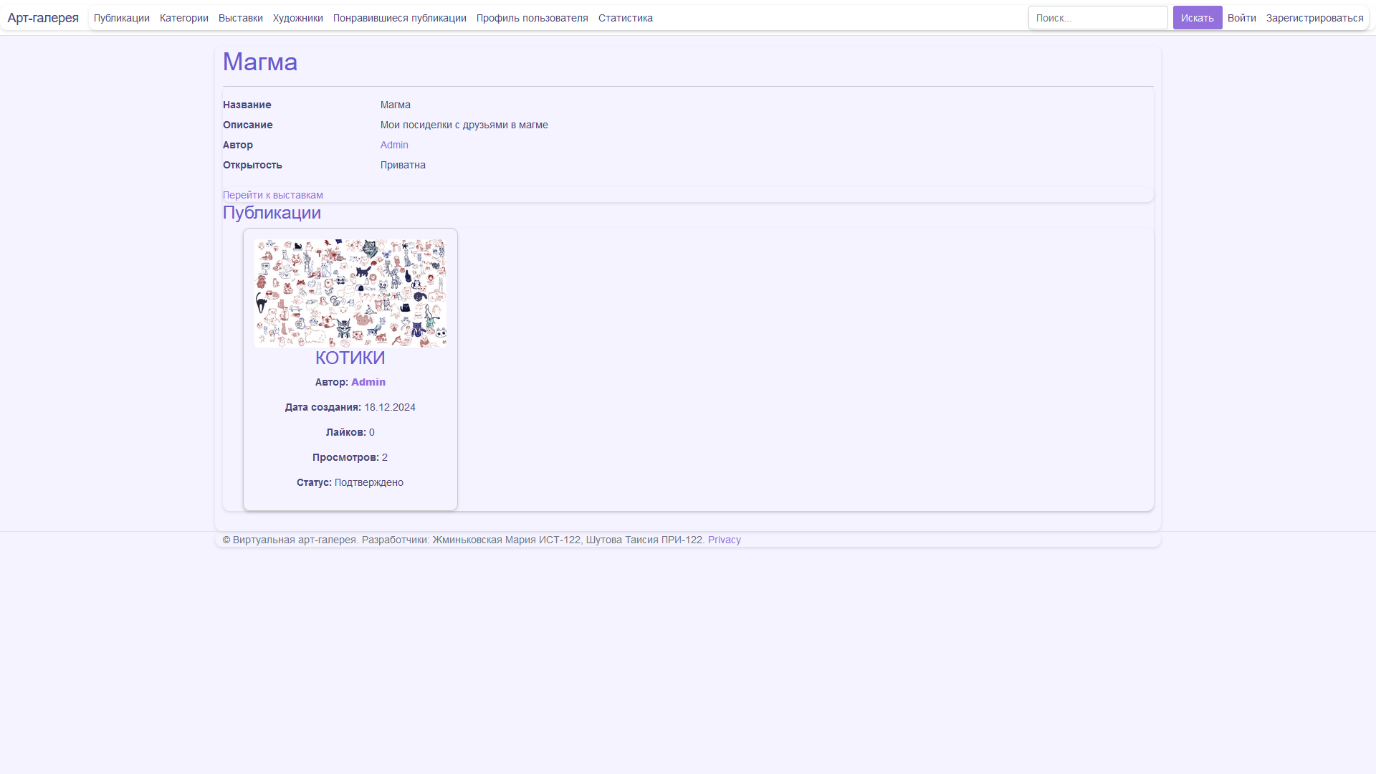


Рисунок 11. Приватная выставка

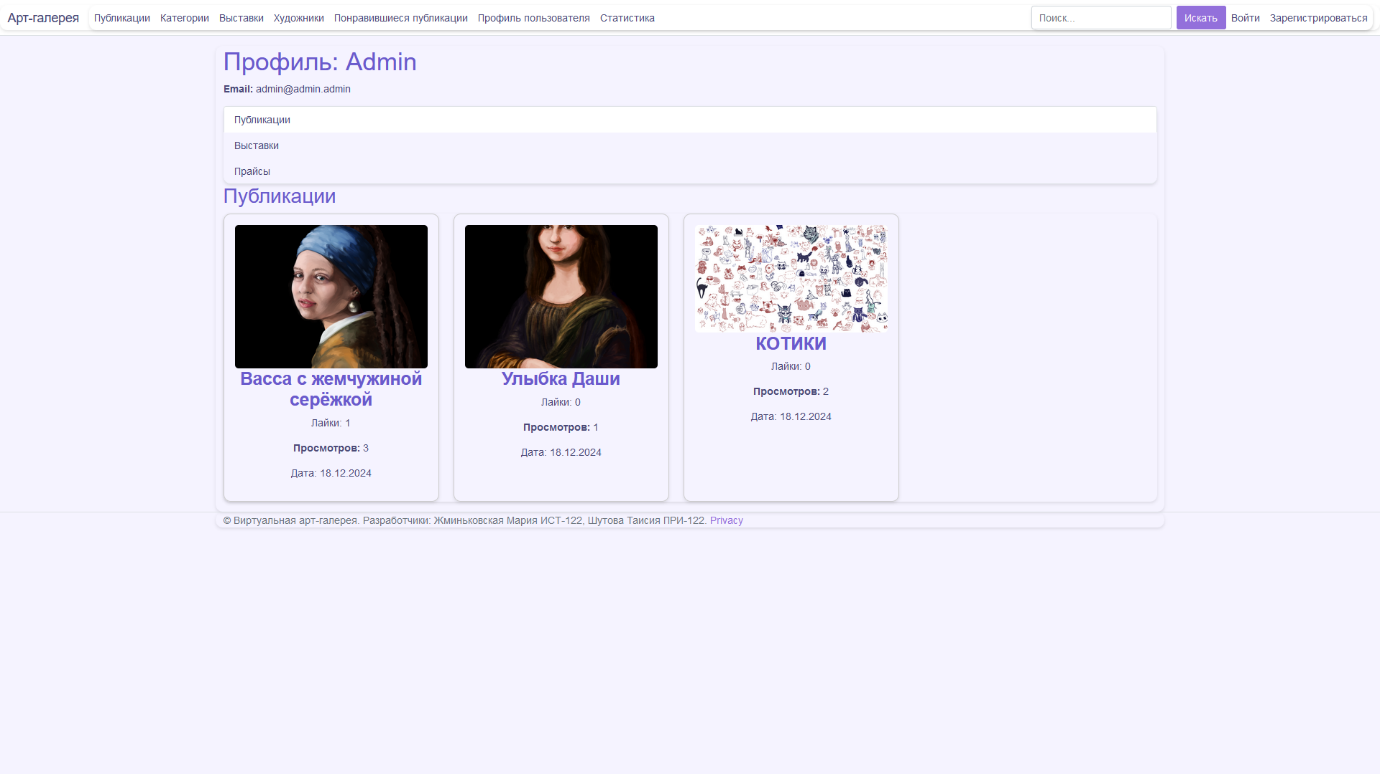


Рисунок 12. Профиль другого пользователя. Публикации

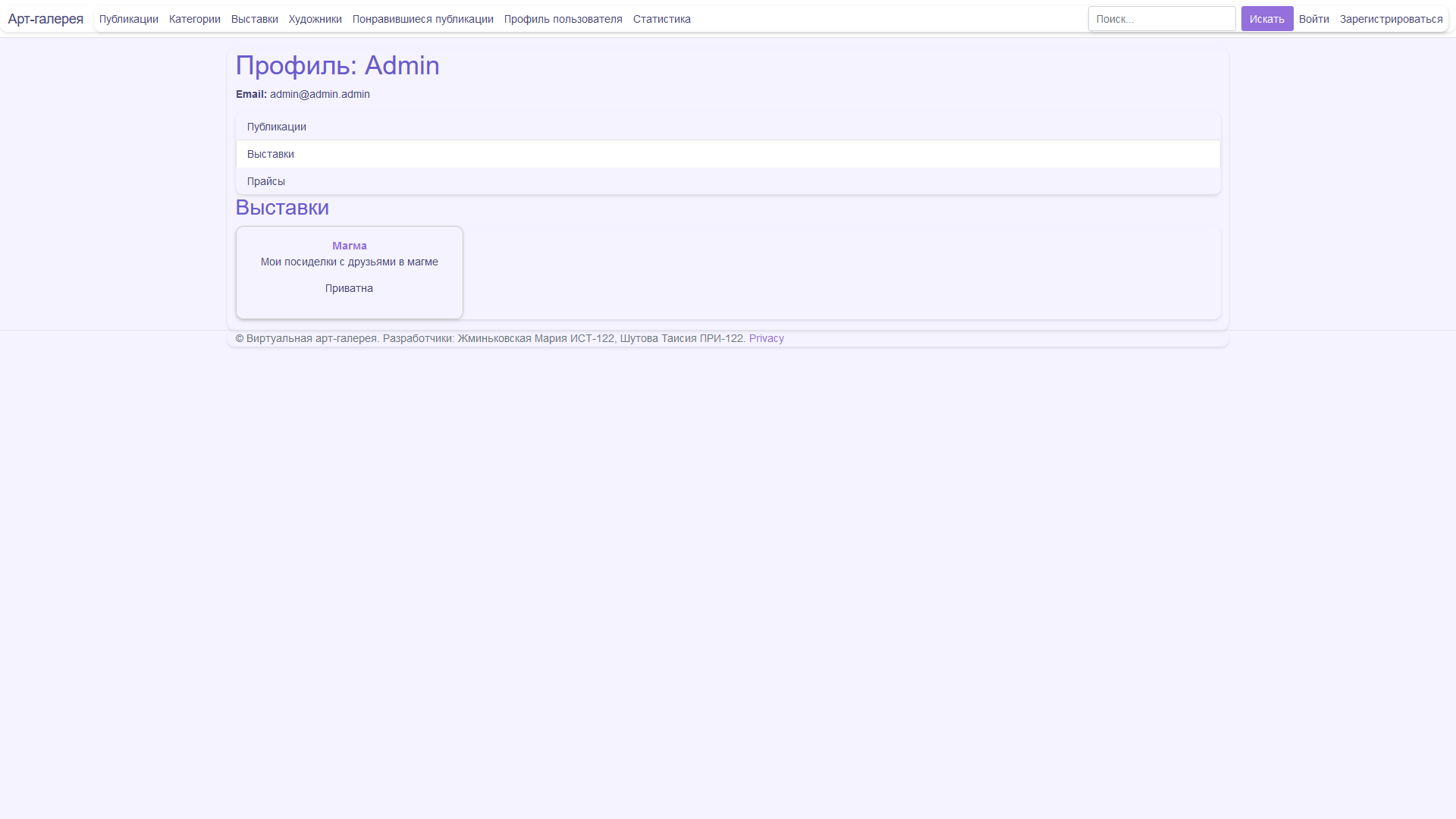


Рисунок 13. Профиль другого пользователя. Выставки

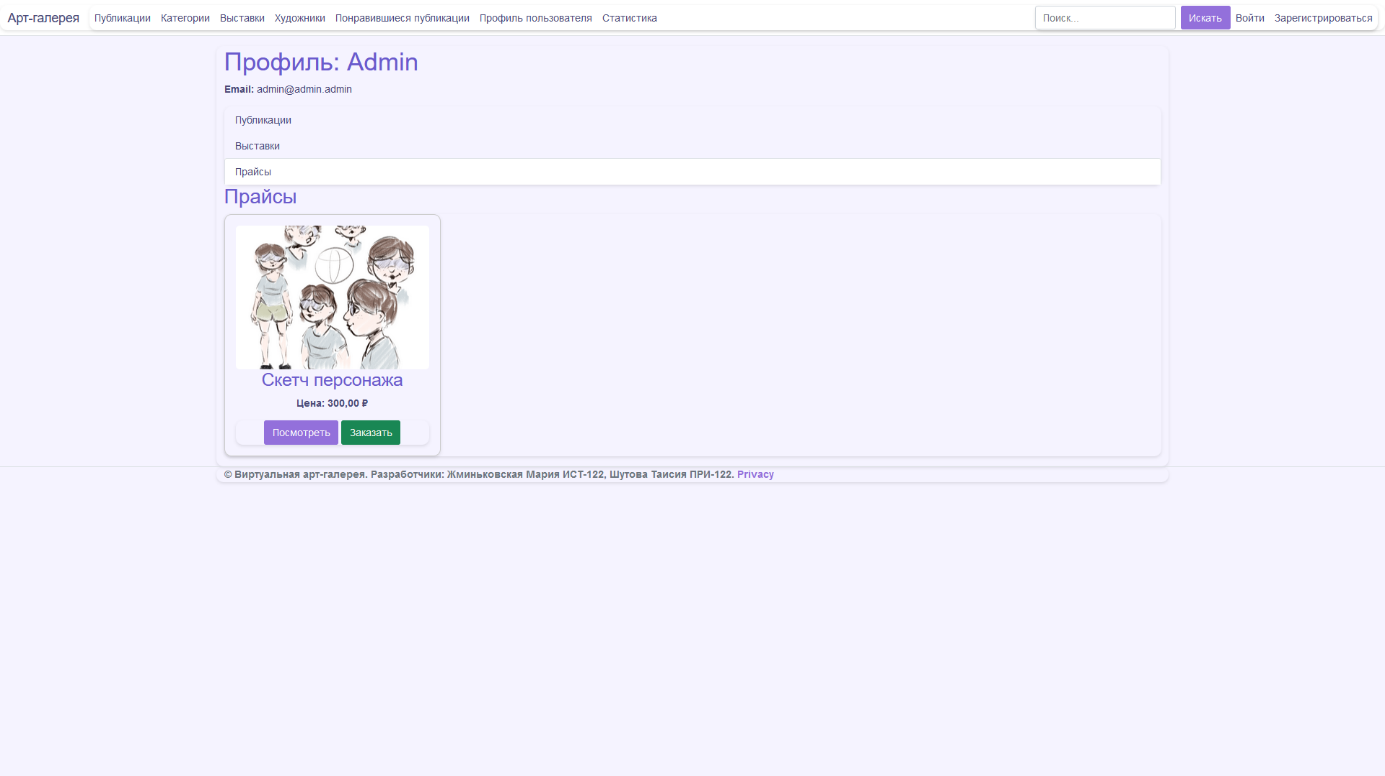


Рисунок 14. Профиль другого пользователя. Прайсы

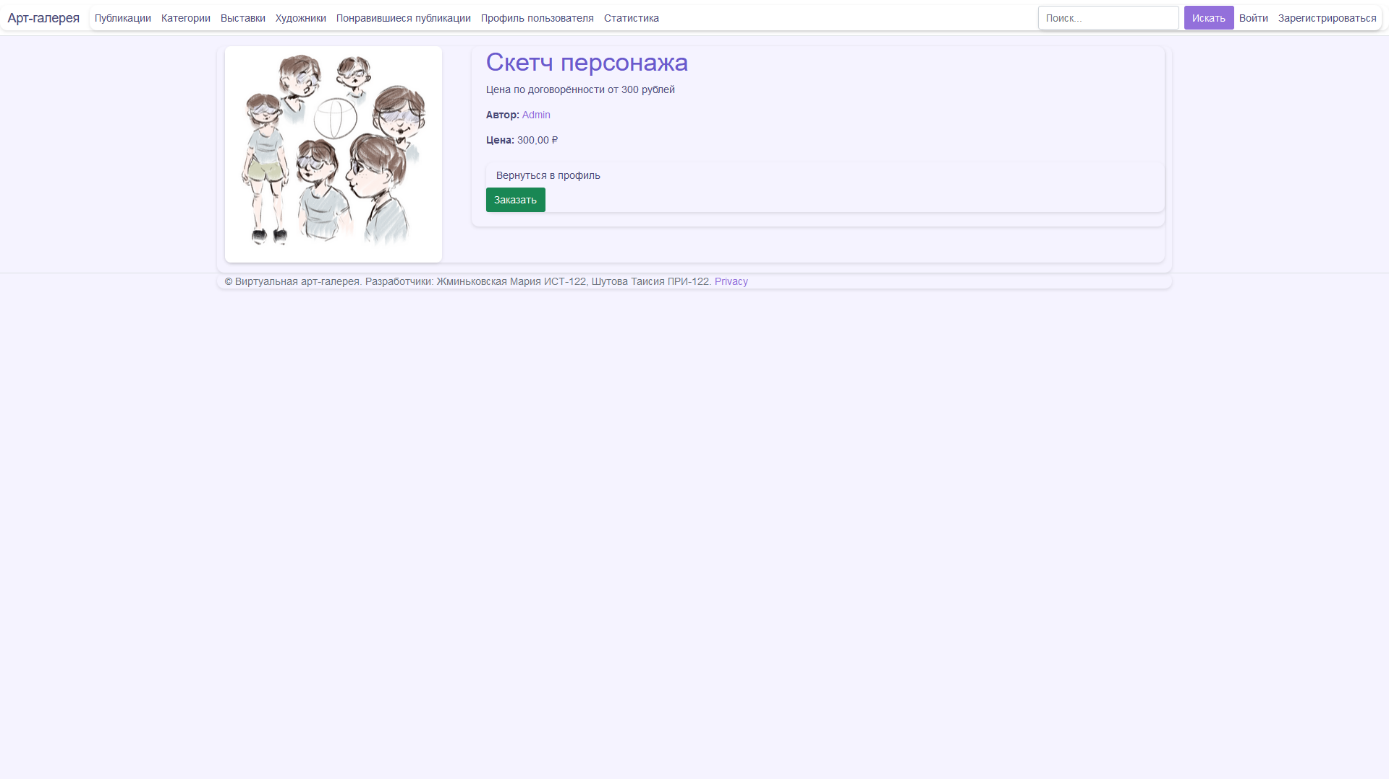


Рисунок 15. Детали чужого прайса

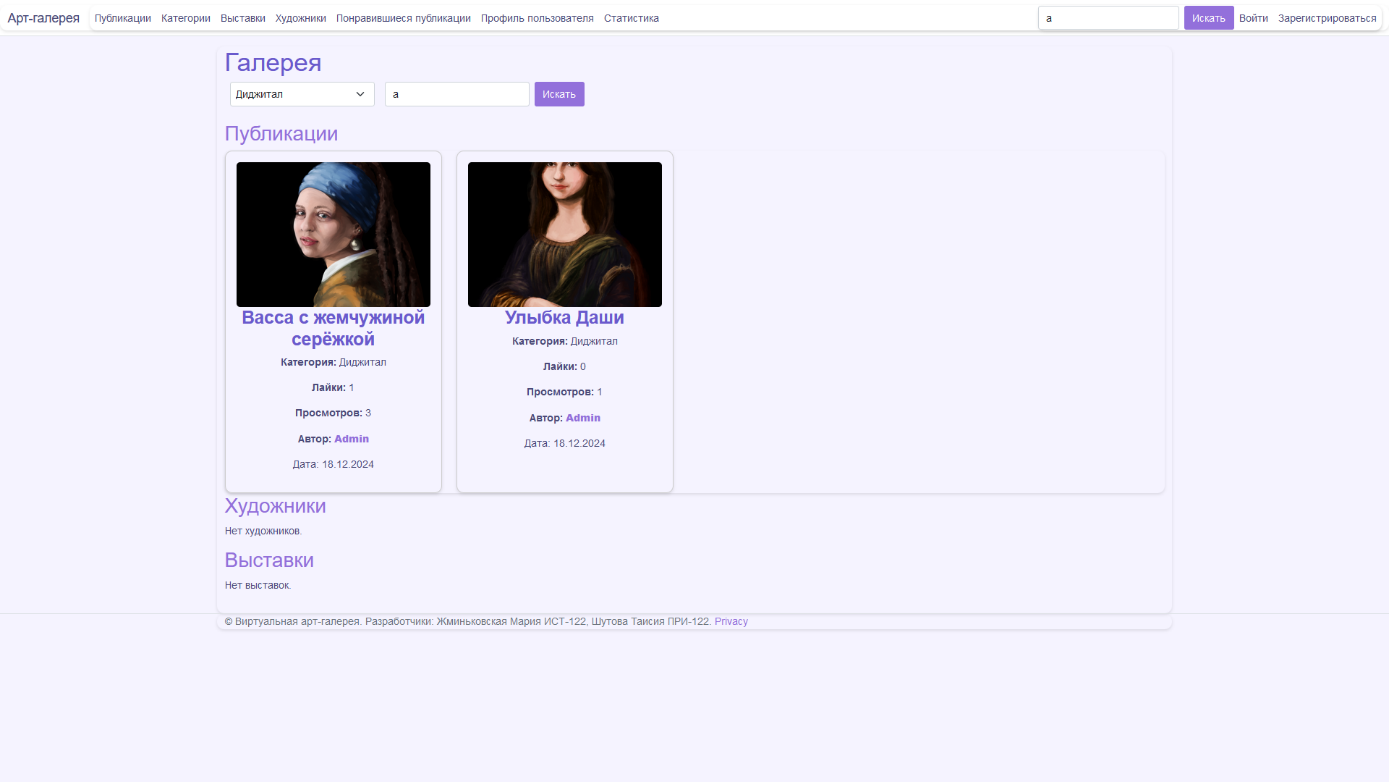


Рисунок 16. Система поиска

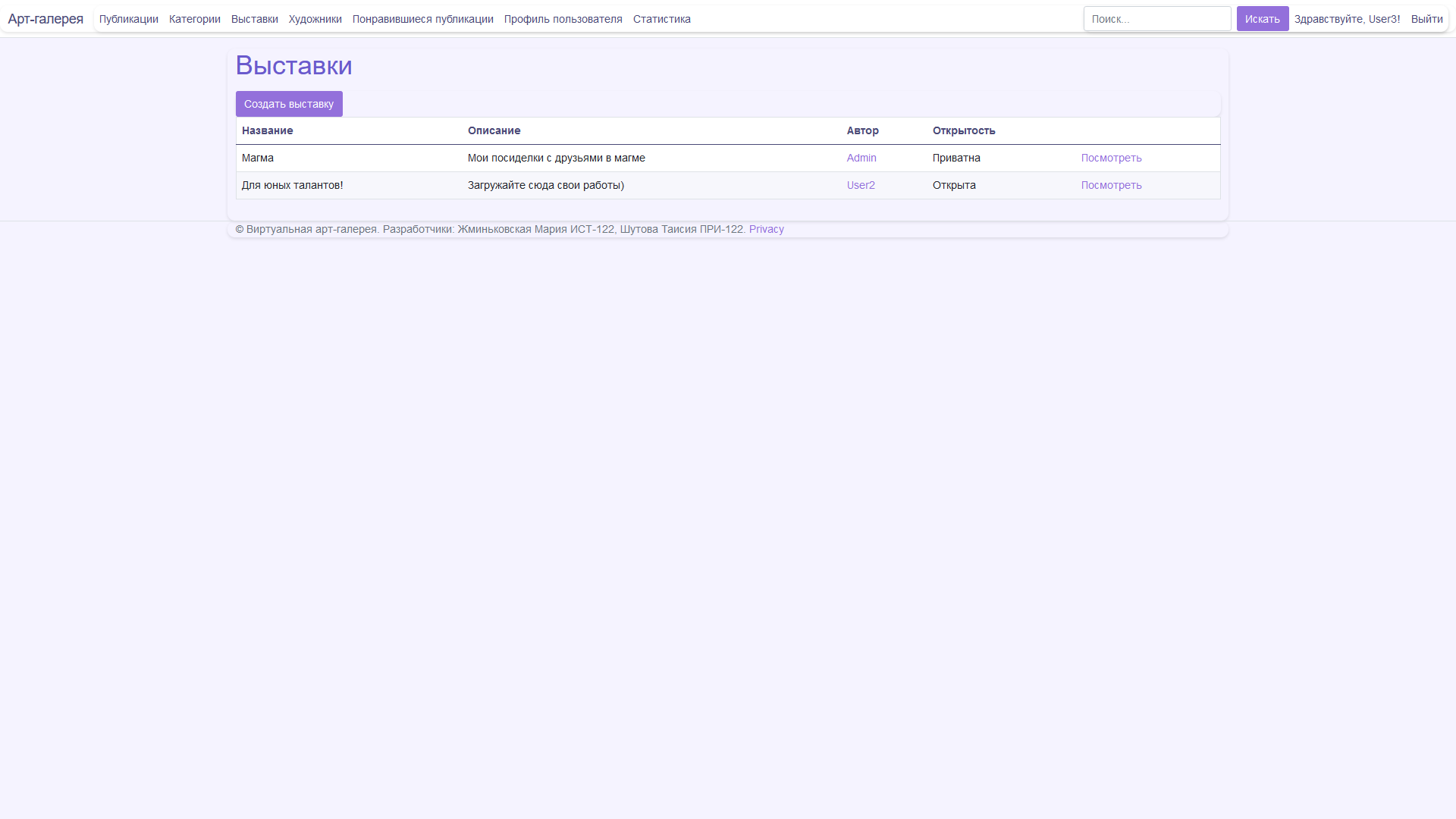


Рисунок 17. Все выставки

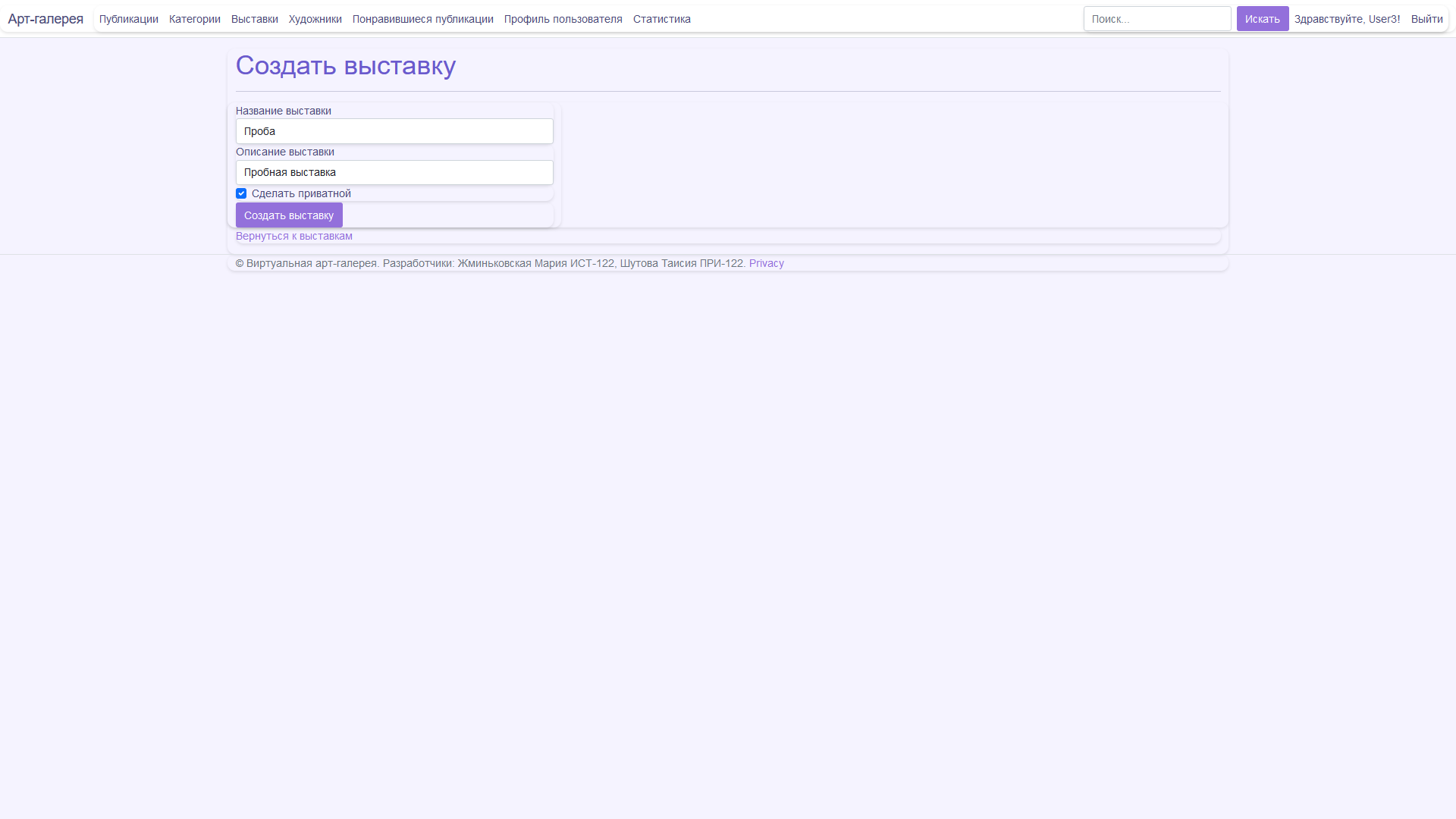


Рисунок 18. Создание выставки

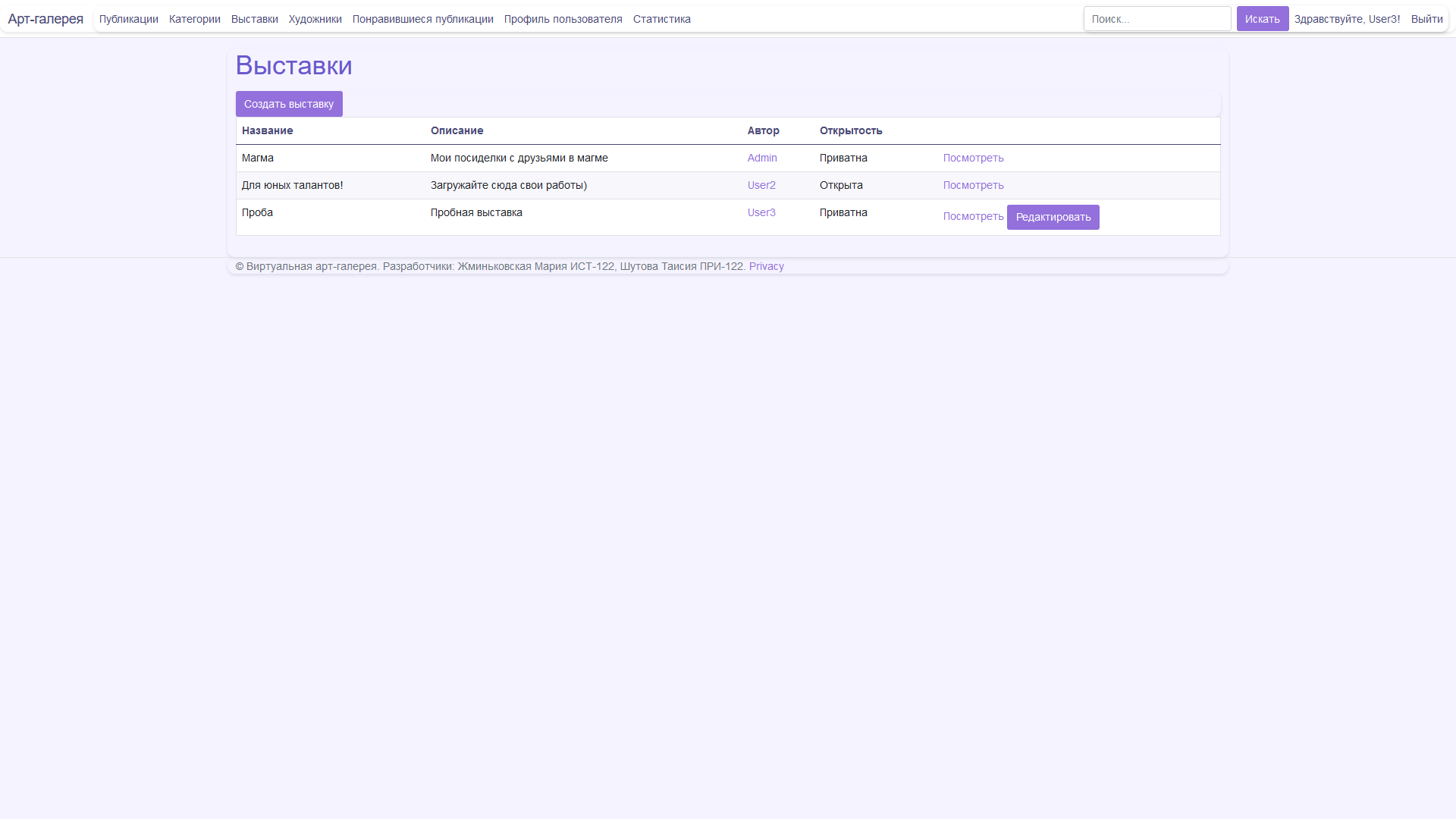


Рисунок 19. Выставка отображается

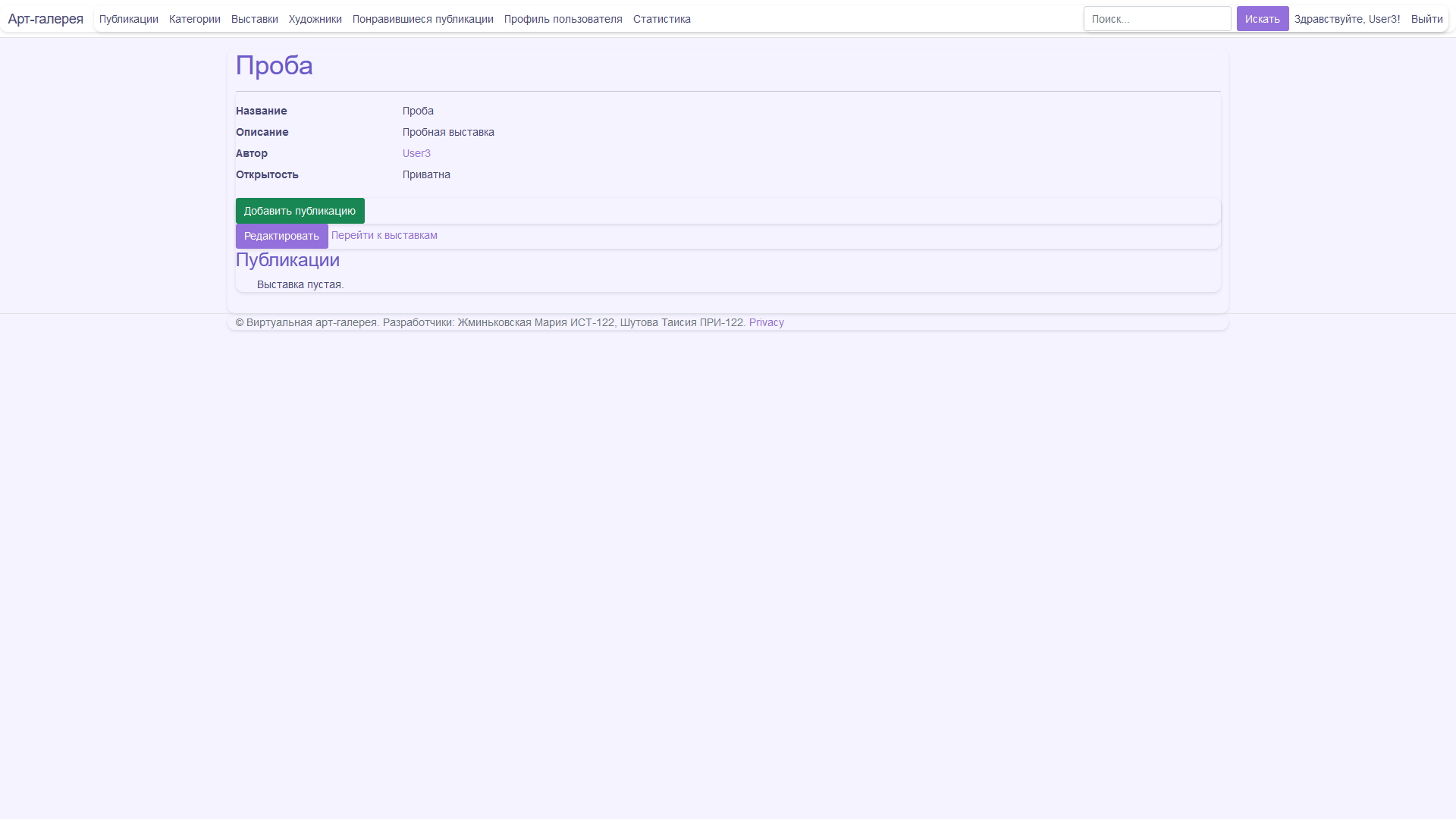


Рисунок 20. Отображение своей выставки

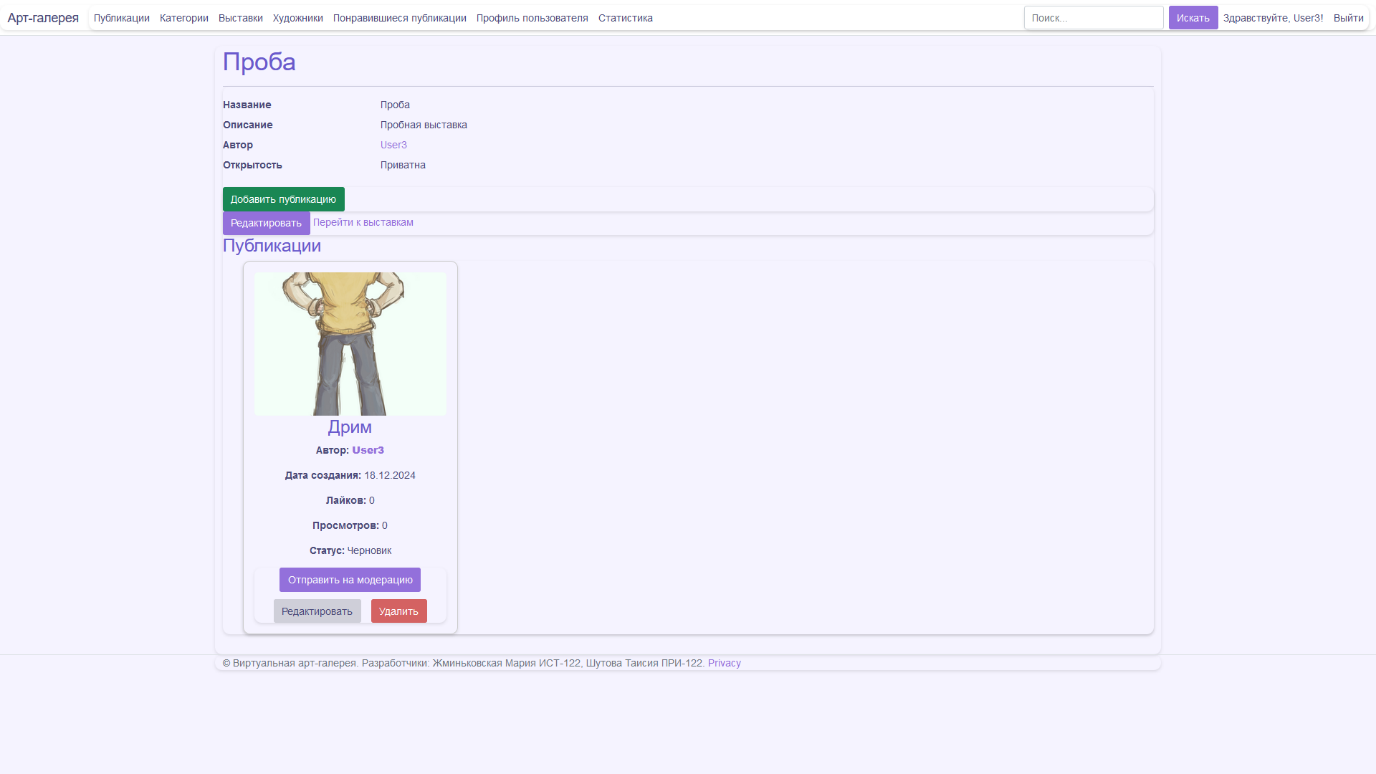


Рисунок 21. Добавленная публикация в выставку, в статусе черновика

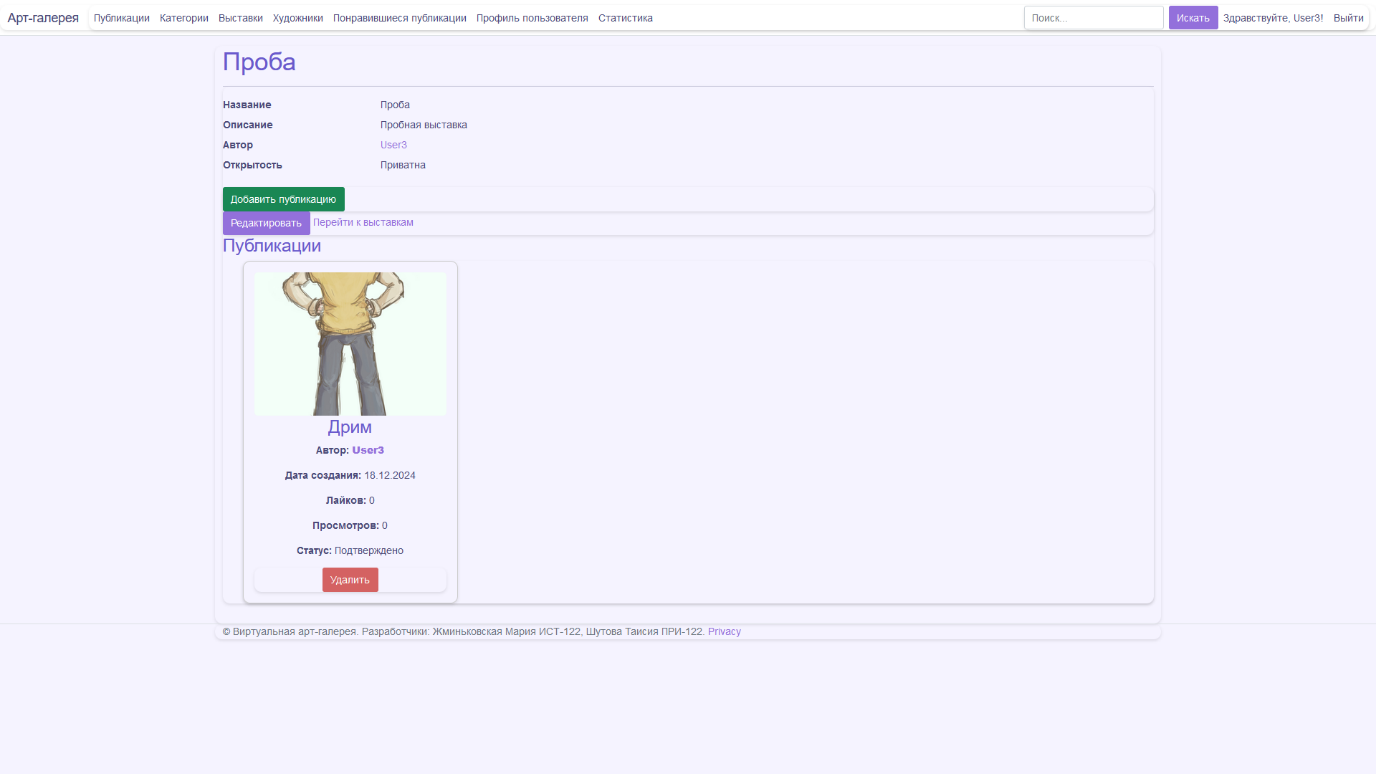


Рисунок 23. Публикация подтверждена

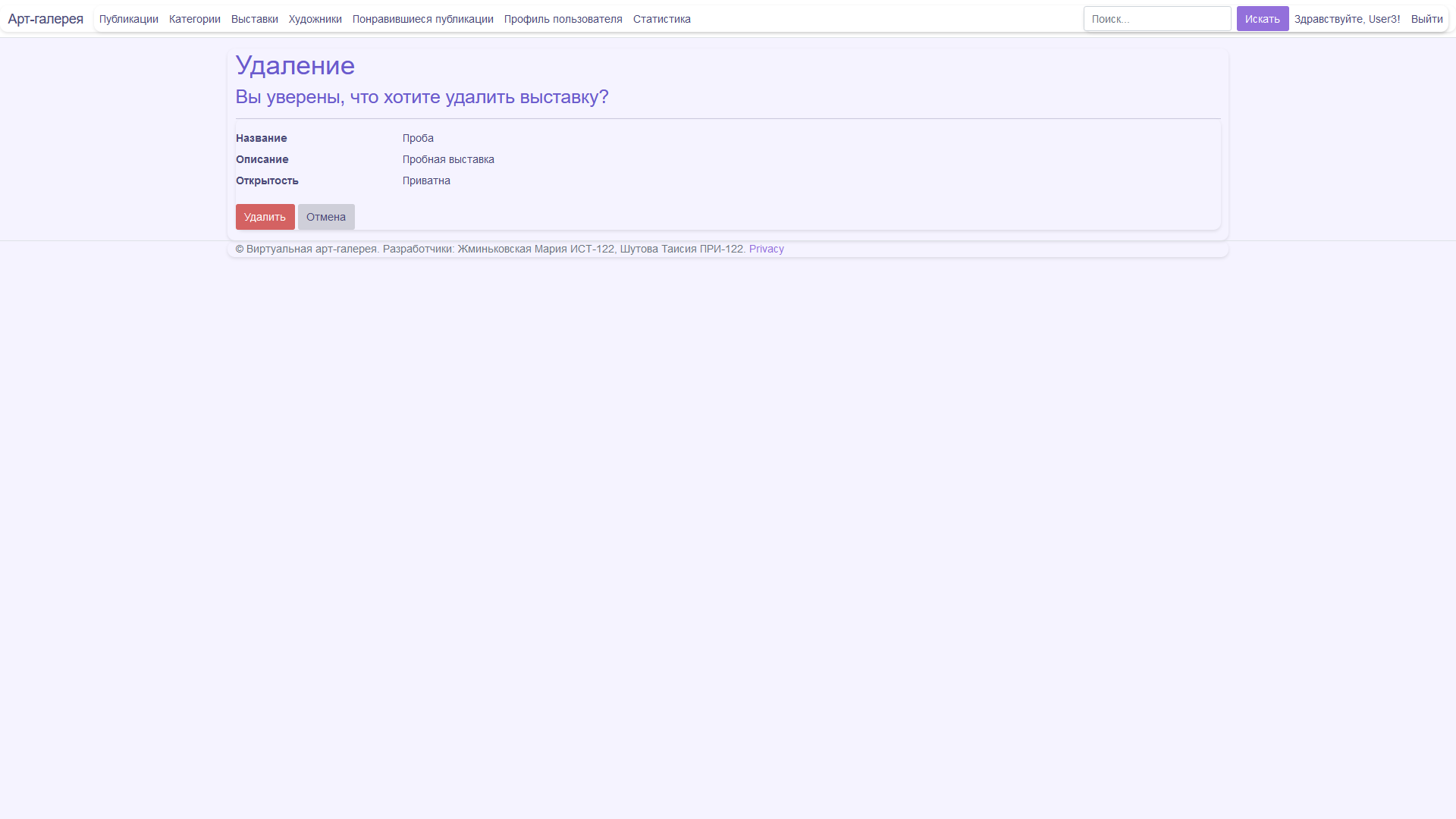


Рисунок 24. Подтверждение удаления выставки

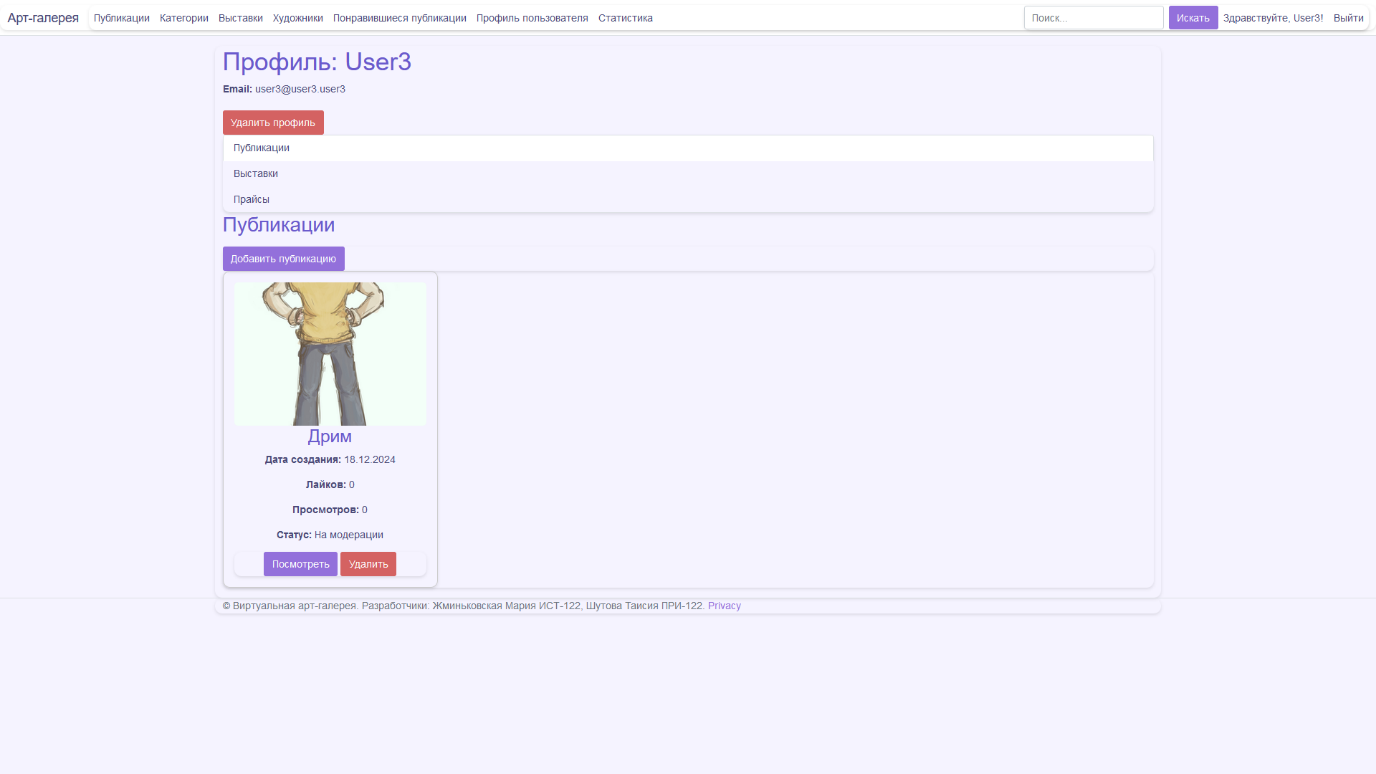


Рисунок 25. После удаления выставки работа остаётся в профиле

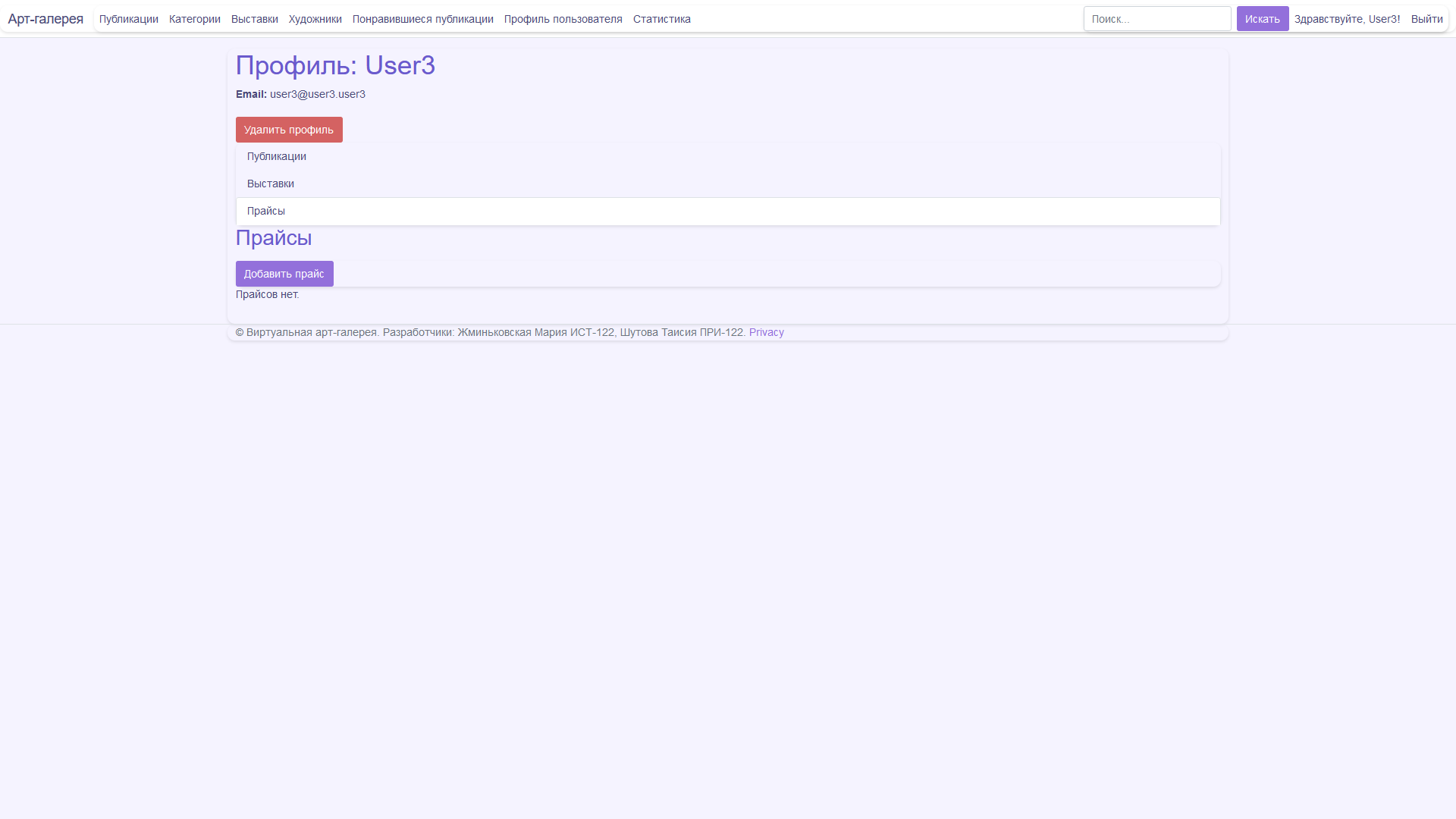


Рисунок 26. Страница своих прайсов

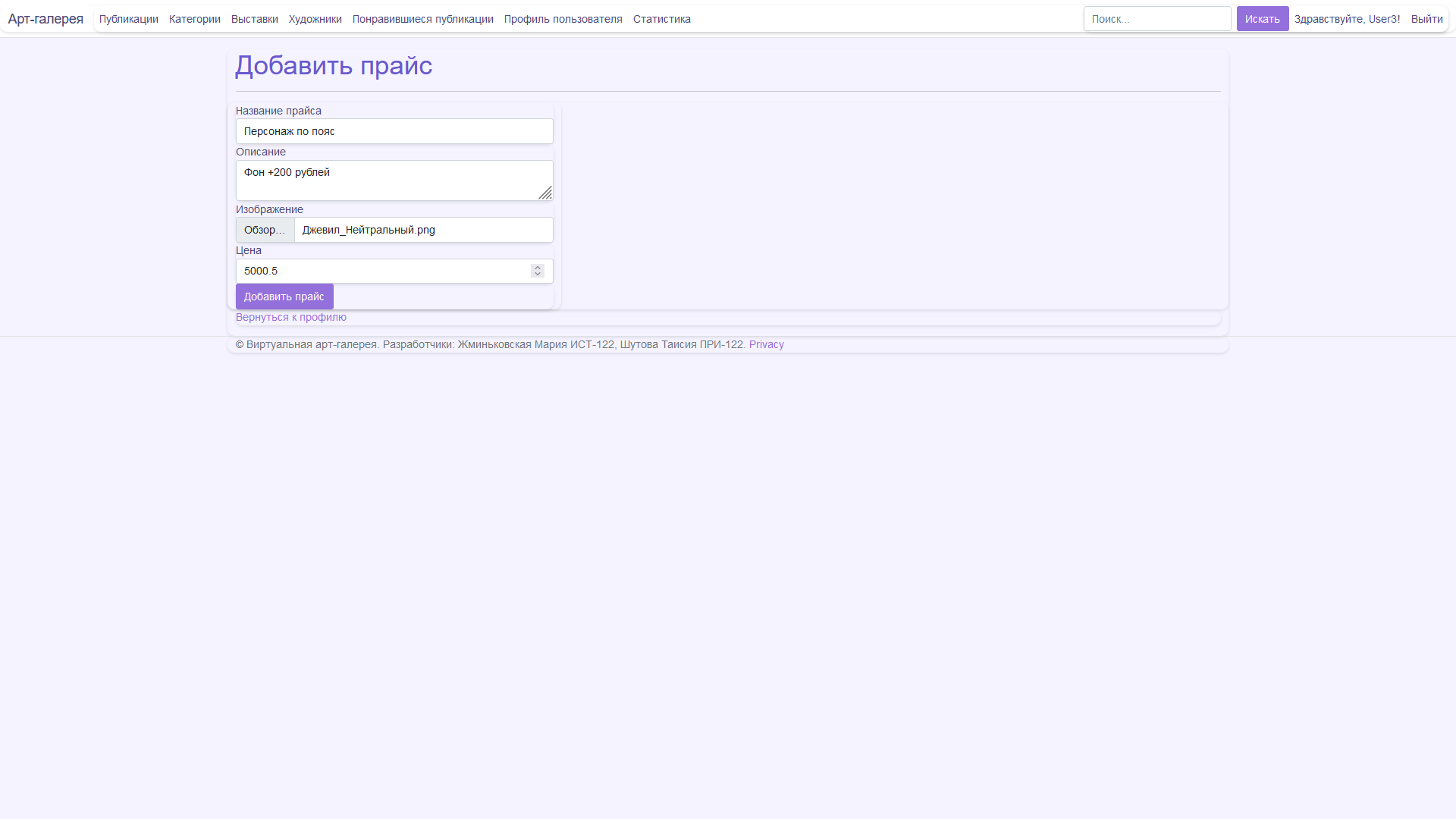


Рисунок 27. Добавление прайса

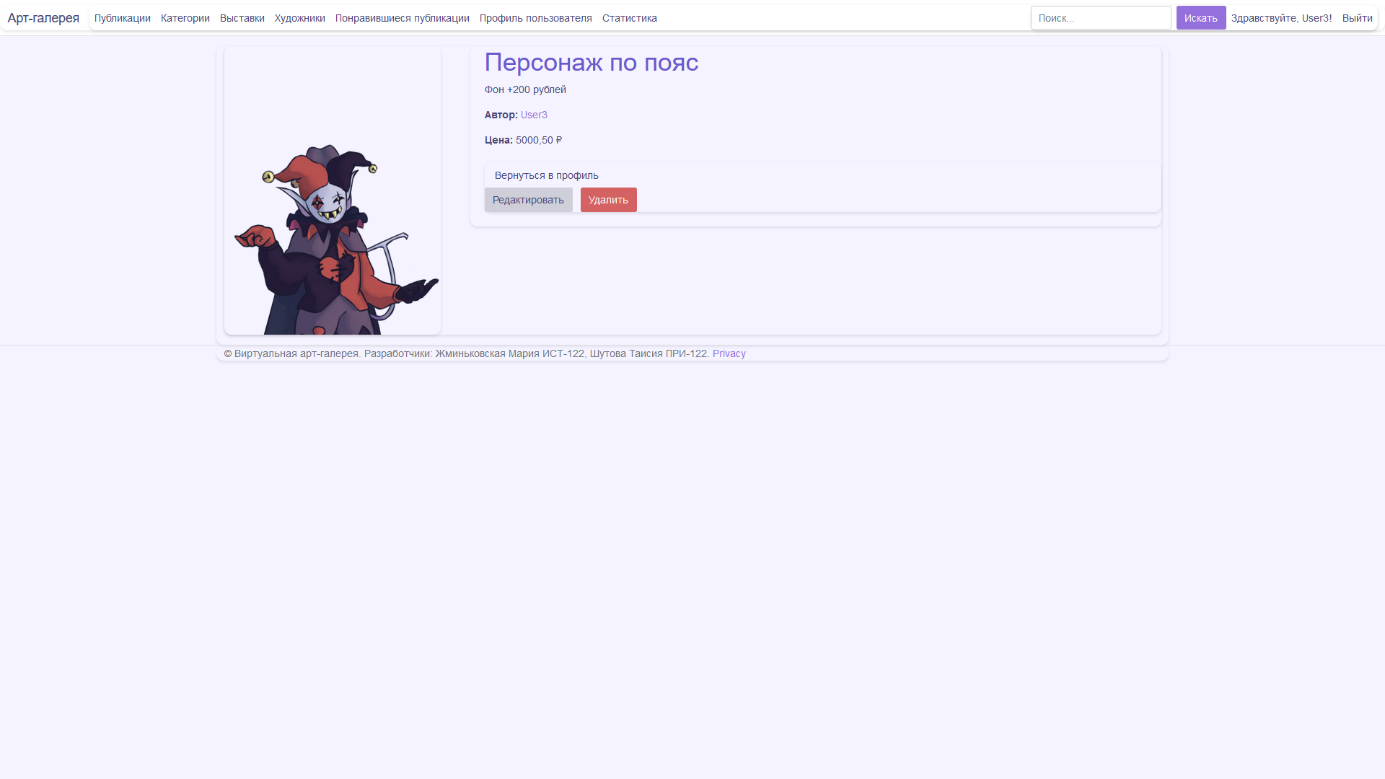


Рисунок 28. Детали прайса

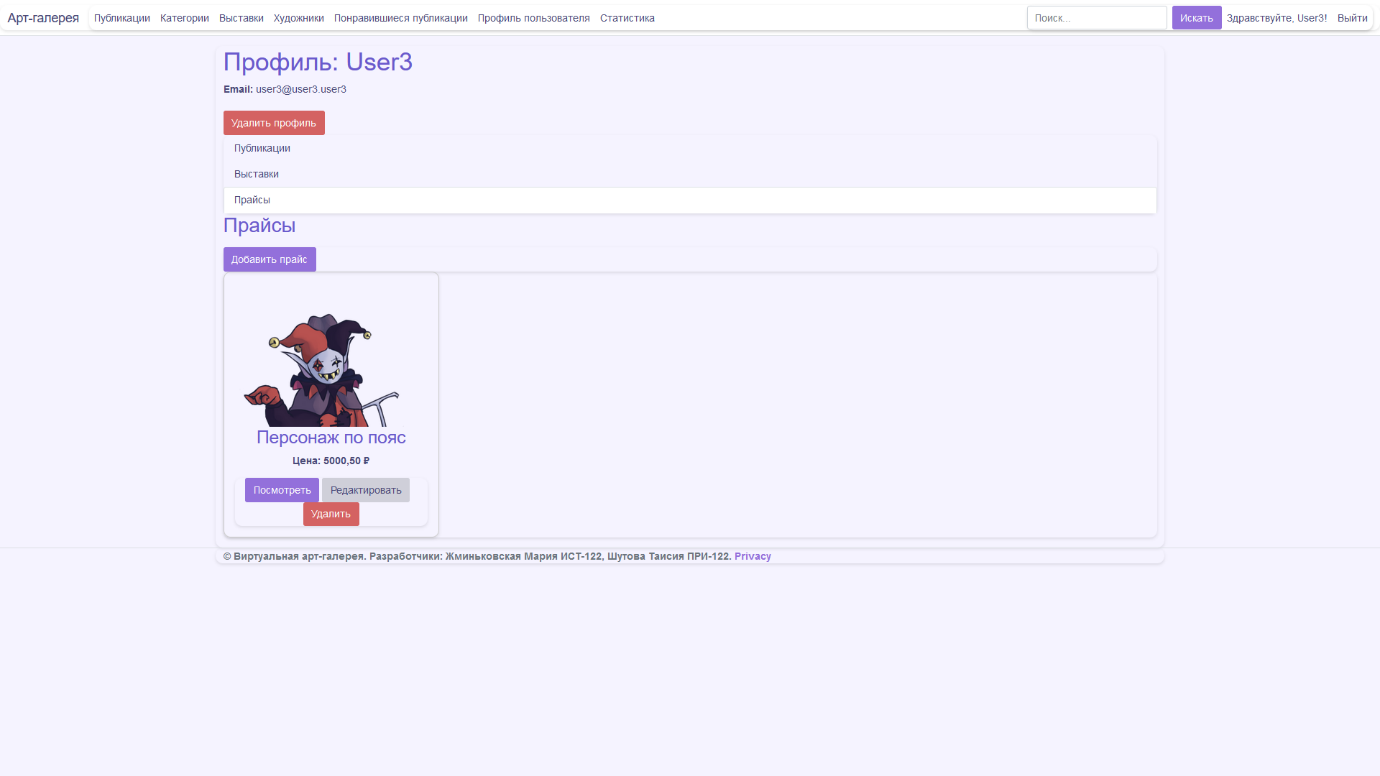


Рисунок 29. Прайс-лист

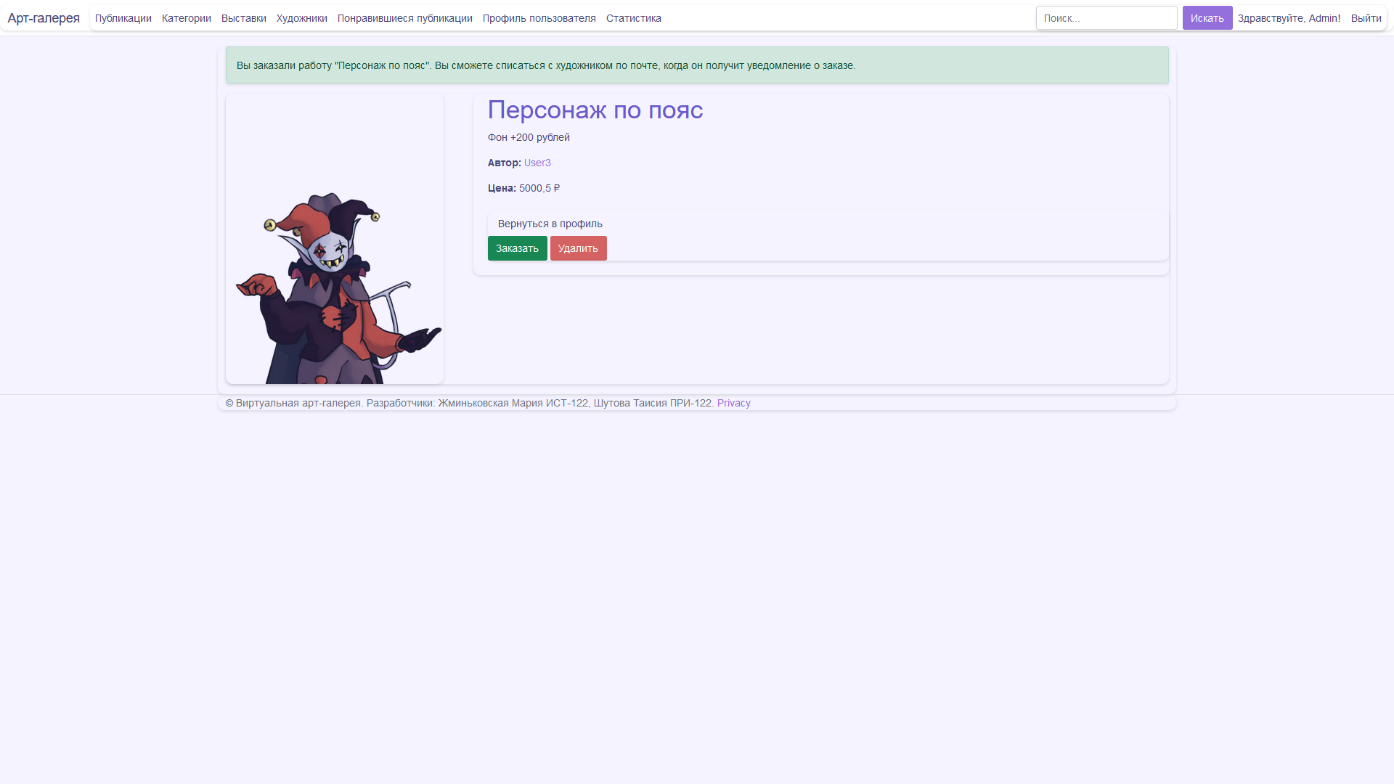


Рисунок 30. Заказать услугу по прайсу.

# 5.9 Тестирование

Были разработаны тесты для проверки входа в систему, просмотра конкретной выставки и всех выставок с помощью Selenium GeckoDriver [9]. Тесты были успешно пройдены, и это представлено на рисунке 31. Остальные функции системы проверялись вручную.

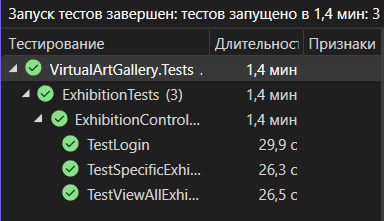


Рисунок 31. Тестирование

6 ЗАКЛЮЧЕНИЕ

В рамках выполнения курсового проекта была реализована информационная система «Виртуальная арт-галерея», модуль «Эксплуатация». Разработка данного модуля позволила создать функциональный инструмент для управления публикациями, выставками и прайсами, взаимодействия с художниками и посетителями, а также поиска данных внутри галереи.

В ходе работы были выполнены все поставленные задачи:

Проведен анализ предметной области, что позволило определить ключевые требования к системе.

Выполнено проектирование архитектуры системы и базы данных, что обеспечило ее высокую производительность, масштабируемость и надежность.

Реализована функциональность системы, включая авторизацию и регистрацию пользователей, управление публикациями, их категориями и статусами, а также внедрены механизмы оценки и сбора статистики.

Проведены тестирование и доработка системы для обеспечения стабильной работы приложения.

Составлена пояснительная записка, в которой детализированы этапы разработки, структура системы и реализованные функциональные возможности.

Созданное приложение обеспечивает удобный доступ пользователей к публикациям, расширяет возможности художников для продвижения их работ, а также предоставляет администраторам мощный инструмент для управления и анализа.

Данный проект демонстрирует, что виртуальные арт-галереи становятся эффективным решением для популяризации искусства в условиях цифровизации. Разработанная система не только решает практические задачи управления контентом, но и способствует созданию уникального пользовательского опыта, что делает ее актуальной и перспективной для дальнейшего развития.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. What Are the Six Types of Relationships in UML Class Diagrams? Blog Visual Paradigm. URL: https://blog.visual-paradigm.com/ru/what-are-the-six-types-of-relationships-in-uml-class-diagrams/ (дата обращения: 08.10.2024).

2. Проектирование: Диаграмма деятельности UML (Activity Diagram). WorldSkills Russia. URL: https://nationalteam.worldskills.ru/skills/proektirovanie-diagrammy-deyatelnosti-uml-activity-diagram/ (дата обращения: 10.10.2024).

3. PostgreSQL. Metanit. URL: https://metanit.com/sharp/mvc5/24.14.php (дата обращения: 10.12.2024).

4. Руководство по ASP.NET Core MVC. Metanit. URL: https://metanit.com/sharp/aspnetmvc/ (дата обращения: 20.09.2024).

5. Начало работы с ASP.NET MVC. Microsoft Learn. URL: https://learn.microsoft.com/ru-ru/aspnet/mvc/overview/getting-started/ (дата обращения: 20.09.2024).

6. Часть 7. Добавление поиска в приложение MVC ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/ru-ru/aspnet/core/tutorials/first-mvc-app/search?view=aspnetcore-9.0 (дата обращения: 02.12.2024).

7. Простое руководство по диаграммы компонентов. Creately. URL: https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-компонентной-диаг/ (дата обращения: 18.12.2024).

8. Простое руководство к диаграммам развертывания UML. Creately. URL: https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-диаграмме-развёрт/ (дата обращения: 18.12.2024).

9. Путь к Инновационному Тестированию с Selenium и C#: Мастерство и Качество. Habr. URL: https://habr.com/ru/articles/757346/ (дата обращения: 19.11.2024).

ПРИЛОЖЕНИЕ А. КОНТРОЛЛЕР ВЫСТАВОК

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

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

{

public class ExhibitionController : Controller

{

private readonly UserManager<IdentityUser> \_userManager;

private readonly GalleryContext \_context;

public ExhibitionController(GalleryContext context, UserManager<IdentityUser> userManager)

{

\_userManager = userManager;

\_context = context;

}

// GET: Exhibition

public async Task<IActionResult> Index()

{

var exhibitions = await \_context.Exhibitions

.Include(e => e.Creator)

.ToListAsync();

return View(exhibitions);

}

// GET: Exhibition/Details/5

public async Task<IActionResult> Details(int id)

{

var userId = \_userManager.GetUserId(User);

var isAdmin = User.IsInRole("Administrator");

var exhibition = \_context.Exhibitions

.Include(e => e.Artworks)

.ThenInclude(a => a.Creator)

.Include(c => c.Creator)

.FirstOrDefault(e => e.Id == id);

if (exhibition == null)

{

return NotFound();

}

exhibition.Artworks = exhibition.Artworks

.Where(a =>

a.Status == ArtworkStatus.Approved ||

(a.CreatorId == userId || isAdmin))

.ToList();

return View(exhibition);

}

// GET: Exhibition/Create

[Authorize]

public IActionResult Create()

{

return View();

}

// POST: Exhibition/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,Title,Description,IsClosed,CreatorId")] ExhibitionModel exhibitionModel)

{

if (ModelState.IsValid)

{

if (!User.Identity.IsAuthenticated)

{

return Unauthorized("Вы должны быть авторизованы для создания выставки.");

}

\_context.Add(exhibitionModel);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(exhibitionModel);

}

// GET: Exhibition/Edit/5

[Authorize]

public async Task<IActionResult> Edit(int id)

{

if (id == null)

{

return NotFound();

}

var exhibitionModel = await \_context.Exhibitions.FindAsync(id);

if (exhibitionModel == null)

{

return NotFound();

}

if (exhibitionModel.CreatorId != \_userManager.GetUserId(User))

{

return Unauthorized("Редактировать выставку может только её создатель.");

}

return View(exhibitionModel);

}

// POST: Exhibition/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,Title,Description,IsClosed,CreatorId")] ExhibitionModel exhibitionModel)

{

if (id != exhibitionModel.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(exhibitionModel);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ExhibitionModelExists(exhibitionModel.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(exhibitionModel);

}

// GET: Exhibition/Delete/5

[Authorize]

[HttpGet]

public async Task<IActionResult> Delete(int id)

{

if (id == null)

{

return NotFound();

}

var exhibitionModel = await \_context.Exhibitions

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

if (exhibitionModel == null)

{

return NotFound();

}

if (exhibitionModel.CreatorId != \_userManager.GetUserId(User) && !User.IsInRole("Administrator"))

{

return Unauthorized("Удалять выставку может только её создатель или администратор.");

}

return View(exhibitionModel);

}

// POST: Exhibition/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var exhibitionModel = await \_context.Exhibitions.FindAsync(id);

if (exhibitionModel != null)

{

\_context.Exhibitions.Remove(exhibitionModel);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ExhibitionModelExists(int id)

{

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

}

}

}

ПРИЛОЖЕНИЕ Б. КОНТРОЛЛЕР ПРАЙСОВ

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using System.Globalization;

using System.Text.RegularExpressions;

using Virtual\_Art\_Gallery.Data;

using Virtual\_Art\_Gallery.Models;

namespace Virtual\_Art\_Gallery.Controllers

{

public class PriceListController : Controller

{

private readonly UserManager<IdentityUser> \_userManager;

private readonly GalleryContext \_context;

public PriceListController(GalleryContext context, UserManager<IdentityUser> userManager)

{

\_userManager = userManager;

\_context = context;

}

// GET: PriceList

public async Task<IActionResult> Index()

{

var galleryContext = \_context.Prices.Include(p => p.Creator);

return View(await galleryContext.ToListAsync());

}

// GET: PriceList/Details/5

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

{

if (id == null)

{

return NotFound();

}

var priceListModel = await \_context.Prices

.Include(p => p.Creator)

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

if (priceListModel == null)

{

return NotFound();

}

return View(priceListModel);

}

// GET: PriceList/Create

[Authorize]

public IActionResult Create()

{

ViewData["CreatorId"] = new SelectList(\_context.Users, "Id", "Id");

return View();

}

// POST: PriceList/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,Description,ImagePath,Price,CreatorId")] PriceListModel priceListModel, IFormFile imageFile)

{

if (ModelState.IsValid)

{

if (!User.Identity.IsAuthenticated)

{

return Unauthorized("Вы должны быть авторизованы для создания прайса.");

}

if (!Regex.IsMatch(priceListModel.Price.ToString(CultureInfo.InvariantCulture), @"^\d+(\.\d{1,2})?$"))

{

ModelState.AddModelError("Price", "Цена должна содержать не более двух цифр после запятой.");

}

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.");

return View(priceListModel);

}

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);

}

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

}

\_context.Add(priceListModel);

await \_context.SaveChangesAsync();

return RedirectToAction("Details", priceListModel);

}

ViewData["CreatorId"] = new SelectList(\_context.Users, "Id", "Id", priceListModel.CreatorId);

return View(priceListModel);

}

// GET: PriceList/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var price = await \_context.Prices.FindAsync(id);

if (price.CreatorId != \_userManager.GetUserId(User))

{

return Unauthorized("Вы не имеете права редактировать этот прайс.");

}

if (price == null)

{

return NotFound();

}

price.Price = Math.Round(price.Price, 2);

ViewData["CreatorId"] = new SelectList(\_context.Users, "Id", "Id", price.CreatorId);

return View(price);

}

// POST: PriceList/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,Description,Price,CreatorId,ImagePath")] PriceListModel priceListModel, IFormFile? imageFile)

{

if (id != priceListModel.Id)

{

return NotFound();

}

if (!Regex.IsMatch(priceListModel.Price.ToString(CultureInfo.InvariantCulture), @"^\d+(\.\d{1,2})?$"))

{

ModelState.AddModelError("Price", "Цена должна содержать не более двух цифр после запятой.");

}

if (ModelState.IsValid)

{

try

{

var existingPriceList = await \_context.Prices.FindAsync(id);

if (existingPriceList == null)

{

return NotFound();

}

existingPriceList.Name = priceListModel.Name;

existingPriceList.Description = priceListModel.Description;

existingPriceList.Price = priceListModel.Price;

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);

}

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

}

\_context.Update(existingPriceList);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!PriceListModelExists(priceListModel.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction("Details", priceListModel);

}

ViewData["CreatorId"] = new SelectList(\_context.Users, "Id", "Id", priceListModel.CreatorId);

return View(priceListModel);

}

// GET: PriceList/Delete/5

[HttpGet]

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

{

if (id == null)

{

return NotFound();

}

var priceListModel = await \_context.Prices

.Include(p => p.Creator)

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

if (priceListModel == null)

{

return NotFound();

}

return View(priceListModel);

}

// POST: PriceList/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var priceListModel = await \_context.Prices.FindAsync(id);

var userId = \_userManager.GetUserId(User);

var creatorId = priceListModel.CreatorId;

if (priceListModel != null)

{

\_context.Prices.Remove(priceListModel);

}

await \_context.SaveChangesAsync();

if (userId == creatorId)

{

return RedirectToAction("IndexPrices", "Profile");

}

else

{

return RedirectToAction("ProfilePrices", "Profile", new {userId = creatorId});

}

}

private bool PriceListModelExists(int id)

{

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

}

[Authorize]

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Order(int id)

{

var price = await \_context.Prices.FindAsync(id);

if (price == null)

{

return NotFound("Прайс не найден.");

}

TempData["OrderMessage"] = $"Вы заказали работу \"{price.Name}\". Вы сможете списаться с художником по почте, когда он получит уведомление о заказе.";

return RedirectToAction("Details", price);

}

}

}

ПРИЛОЖЕНИЕ В. КОНТРОЛЛЕР ГАЛЕРЕИ

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Virtual\_Art\_Gallery.Data;

using Virtual\_Art\_Gallery.Models;

using Microsoft.EntityFrameworkCore;

using System.Linq;

using Microsoft.AspNet.Identity;

using Microsoft.AspNetCore.Mvc.Rendering;

namespace Virtual\_Art\_Gallery.Controllers

{

public class GalleryController : Controller

{

private readonly GalleryContext \_context;

public GalleryController(GalleryContext context)

{

\_context = context;

}

public async Task<IActionResult> Index(string? searchQuery, int? categoryId)

{

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

var categories = await \_context.Categories.ToListAsync();

ViewBag.SearchQuery = searchQuery;

ViewBag.CategoryId = categoryId;

IQueryable<ArtworkModel> artworkQuery = \_context.Artworks

.Where(a => a.Status == ArtworkStatus.Approved)

.Include(a => a.Category)

.Include(a => a.Creator);

IQueryable<IdentityUser> artistQuery = \_context.Users

.Where(u => \_context.Artworks.Any(a => a.CreatorId == u.Id && a.Status == ArtworkStatus.Approved));

IQueryable<ExhibitionModel> exhibitionQuery = \_context.Exhibitions

.Include(e => e.Creator)

.Include(e => e.Artworks)

.ThenInclude(a => a.Creator);

if (categoryId.HasValue)

{

artworkQuery = artworkQuery.Where(a => a.CategoryId == categoryId.Value);

if (!string.IsNullOrEmpty(searchQuery))

{

string lowerSearchQuery = searchQuery.ToLower();

artworkQuery = artworkQuery.Where(a => a.Title.ToLower().Contains(lowerSearchQuery));

}

artistQuery = Enumerable.Empty<IdentityUser>().AsQueryable();

exhibitionQuery = Enumerable.Empty<ExhibitionModel>().AsQueryable();

}

else if (!string.IsNullOrEmpty(searchQuery))

{

string lowerSearchQuery = searchQuery.ToLower();

artworkQuery = artworkQuery.Where(a =>

a.Title.ToLower().Contains(lowerSearchQuery));

exhibitionQuery = exhibitionQuery.Where(e =>

e.Title.ToLower().Contains(lowerSearchQuery));

artistQuery = artistQuery.Where(u => u.UserName.ToLower().Contains(lowerSearchQuery));

}

var artworks = await artworkQuery.OrderBy(a => Guid.NewGuid()).Take(4).ToListAsync();

var artistProfiles = artistQuery.Any()

? await artistQuery.OrderBy(u => Guid.NewGuid()).Take(4).ToListAsync()

: artistQuery.ToList();

var exhibitions = exhibitionQuery.Any()

? await exhibitionQuery.OrderBy(e => Guid.NewGuid()).Take(4).ToListAsync()

: exhibitionQuery.ToList();

if (!artworks.Any())

{

ViewBag.NoArtworksMessage = "Нет публикаций.";

}

if (!artistProfiles.Any())

{

ViewBag.NoProfilesMessage = "Нет художников.";

}

if (!exhibitions.Any())

{

ViewBag.NoExhibitionsMessage = "Нет выставок.";

}

var model = new GalleryIndexViewModel

{

Artworks = artworks,

ArtistProfiles = artistProfiles,

Exhibitions = exhibitions,

Categories = categories,

SelectedCategoryId = categoryId

};

return View(model);

}

}

}