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**

по дисциплине

«Технологии разработки мобильных приложений»

на тему:

«Разработка мобильного приложения «Виртуальная арт-галерея». Модуль эксплуатации»

Выполнил: студент гр. ПРИ-122

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шутова Т.Е.

(подпись)

Принял: ст. преп. каф. ИСПИ

\_\_\_\_\_\_\_\_\_\_\_\_\_ Данилов В.В.

(подпись)

Дата сдачи: \_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_

Владимир, 2025

**АННОТАЦИЯ**

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

Пояснительная записка содержит 52 листа, в том числе 25 рисунков, 14 использованных источника, 3 приложения.

**ANNOTATION**

In this course project, an application was developed for the Virtual Art Gallery software system. The "Operation" module. The explanatory note contains the following sections: introduction, description of the subject area, design of the PS application, implementation of the PS application, load testing and optimization, conclusion, list of references, Appendix A, Appendix B, Appendix B.

The explanatory note contains 52 sheets, including 25 figures, 14 sources used, 3 appendices.

**СОДЕРЖАНИЕ**

[1 ВВЕДЕНИЕ 4](#_Toc198668616)

[2 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc198668617)

[2.1 Цель и задачи 5](#_Toc198668618)

[2.2 Описание предметной области 5](#_Toc198668619)

[2.3 Словарь предметной области 6](#_Toc198668620)

[3 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ ДЛЯ ИС 7](#_Toc198668621)

[3.1 Диаграмма прецедентов 7](#_Toc198668622)

[3.2 Описание прецедентов 8](#_Toc198668623)

[3.3 Диаграмма классов 8](#_Toc198668624)

[3.4 Диаграмма состояний 9](#_Toc198668625)

[3.5 Диаграмма последовательностей 10](#_Toc198668626)

[3.6 Схема бизнес-процесса (BPMN) 11](#_Toc198668627)

[3.7 Схема взаимодействия программ 12](#_Toc198668628)

[4 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ ДЛЯ ИС 14](#_Toc198668629)

[4.1 Разработка базы данных 14](#_Toc198668630)

[4.2 Разработка веб-приложения 14](#_Toc198668631)

[4.3 Разработка приложения для информационной системы 16](#_Toc198668632)

[4.4 Диаграмма компонентов 17](#_Toc198668633)

[4.5 Диаграмма развёртывания 18](#_Toc198668634)

[4.6 Репозиторий проекта 18](#_Toc198668635)

[4.7 Реализация 18](#_Toc198668636)

[4.8 Страницы приложения 19](#_Toc198668637)

[5 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ 26](#_Toc198668638)

[6 ЗАКЛЮЧЕНИЕ 29](#_Toc198668639)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 30](#_Toc198668640)

[ПРИЛОЖЕНИЕ А. ПУБЛИКАЦИИ 31](#_Toc198668641)

[ПРИЛОЖЕНИЕ Б. ВЫСТАВКИ 40](#_Toc198668642)

[ПРИЛОЖЕНИЕ В. ПОЛЬЗОВАТЕЛИ 48](#_Toc198668643)

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 фотохостингов для публикации художественных изображений 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 мобильного приложения для виртуальной арт-галереи, представляющей собой программный продукт, созданный с использованием современных технологий. Главной целью проекта является обеспечение удобного и интерактивного доступа к публикациям с мобильного устройства. 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, выставками и профилями художников. Приложение будет ориентировано на создание уникального пользовательского опыта, что сделает виртуальную арт-галерею важным инструментом для популяризации искусства в цифровую эпоху.

2 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

# 2.1 Цель и задачи

Цель работы: разработать мобильное приложение для информационной системы «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 просмотра публикаций, выставок, художников (администратор, художник, посетитель);
* Отображение публикаций и выставок в профиле художника (администратор, художник);
* Публикация работ (администратор, художник);
* Создание приватных и открытых выставок и их редактирование (администратор, художник);
* Участие в публичных и приватных выставках (администратор, художник).

А также необходимо решить следующие задачи:

1. Анализ предметной области, функциональная декомпозиция системы, требования к системе, обзор аналогов;
2. Проектирование мобильного приложения, разработка модели данных;
3. Реализация программной системы с использованием заданных технологий, разработка своего API и использование стороннего API;
4. Разработать сценарии тестирования приложения;
5. Развернуть мобильное приложение в магазине приложений;
6. Оформить документацию по разработанному приложению.

# 2.2 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.

# 2.3 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, количество лайков.

*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 и выставках.

Атрибуты: ID, никнейм, почта, публикации, выставки.

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

Атрибуты: публикации, художники, выставки.

# 2.4 Обзор аналогов

Виртуальная арт-галерея — информационная система, нацеленная на художников и ценителей искусства. Поэтому для сравнения с возможностями и направленностями разрабатываемого приложения будут выбраны мобильные приложения с похожей тематикой: Pinterest, DeviantArt, Artsy.

Таблица 1. Сравнительный анализ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Критерий** | ***VAGMobile*** | **Pinterest** | **DeviantArt** | **Artsy** |
| Целевая аудитория | *Художники и ценители искусства с контролем качества* | Широкая публика для визуального поиска идей | Художники и арт-сообщество | Коллекционеры и профессионалы арт-рынка |
| Основная функция | *Публикация художественных работ с контролем качества* | Визуальный поиск и сохранение контента | Сообщество художников, портфолио | Коммерческая платформа для искусства |
| Модерация контента | *Строгая система одобрения/отклонения публикаций* | Автоматическая + пользовательские жалобы | Сообщество + модераторы | Строгая верификация галерей |
| Социальные функции | *Лайки, комментарии, профили художников, выставки* | Доски, следование, репосты | Критика, группы, конкурсы | Следование галереям, коллекции |
| Коммерческая ориентация | *Минимальная (демонстрация искусства)* | Реклама, партнерства | Платные подписки, комиссионные | Аукционы, продажи искусства |
| Категоризация | *Категории искусства, кураторские выставки* | Широкие категории интересов | Типы искусства, стили | Галереи, художники, движения |
| UI/UX подход | *Элегантный Material Design 3, художественная атмосфера* | Яркий, интуитивный | Темный интерфейс для художников | Минималистичный премиум |
| Специализация | *Традиционное искусство с модерацией* | Широкий визуальный контент | Художественное сообщество | Коммерческий арт-рынок |

Подробный анализ по ключевым критериям

1. Модерация и качество контента

* **VAGMobile**: Имеет строгую систему модерации с тремя статусами (одобрено/отклонено/ожидание), что гарантирует высокое качество публикуемого контента
* Pinterest: Полагается на алгоритмы и пользовательские жалобы, что приводит к смешанному качеству контента
* DeviantArt: Комбинирует модерацию сообщества с администраторами, но допускает более широкий спектр контента
* Artsy: Строгая верификация участников, но фокус на коммерческой ценности

2. Организация контента

* **VAGMobile**: Уникальная система выставок позволяет создавать тематические коллекции, что является преимуществом для художественного контекста
* Pinterest: Доски пользователей для персонального хранения идей
* DeviantArt: Группы и сообщества по интересам
* Artsy: Организация по галереям и движениям

3. UI/UX подход

* **VAGMobile**: Элегантный интерфейс с атмосферой художественной галереи, Material Design 3 обеспечивает комфортное восприятие искусства
* Pinterest: Оптимизирован для быстрого скроллинга и визуального поиска
* DeviantArt: Темный интерфейс для продолжительной работы художников
* Artsy: Премиум-дизайн для серьезных коллекционеров

Преимущества VAGMobile перед аналогами

1. Специализация на художественном искусстве с гарантией качества

В отличие от широких платформ вроде Pinterest, где контент может быть любого качества и тематики, VAGMobile фокусируется исключительно на художественном искусстве с обязательной модерацией. Это создает доверительную среду для художников и зрителей, где каждая работа проходит проверку качества.

2. Уникальная система выставок

Ни одна из анализируемых платформ не имеет столь развитой системы выставок. Это позволяет:

* Создавать тематические коллекции произведений
* Проводить виртуальные выставки
* Обеспечивать контекст для восприятия искусства

3. Баланс между социальными функциями и профессионализмом

VAGMobile сочетает социальные возможности (лайки, комментарии, профили) с профессиональным подходом к модерации, что отличает его от:

* DeviantArt (слишком ориентирован на сообщество, меньший контроль качества)
* Pinterest (социальные функции доминируют над качеством контента)
* Artsy (слишком коммерчески ориентирован)

4. Элегантный интерфейс без отвлекающих элементов

В отличие от яркого и насыщенного интерфейса Pinterest, VAGMobile создает атмосферу художественной галереи с помощью:

* Спокойной цветовой схемы (фиолетово-розовые тона)
* Четкой типографики
* Фокуса на самом искусстве, а не на интерфейсе

5. Отсутствие коммерческого давления

VAGMobile не навязывает монетизацию, фокусируясь на демонстрации искусства как такового. Это отличает его от:

* Artsy (явно коммерческая платформа)
* Pinterest (реклама и партнерства)

# 2.5 Функциональные требования

**2.5.1 Авторизация и регистрация**

2.5.1.1 «Авторизация и регистрация»

2.5.1.2 Система должна предоставлять возможность регистрации нового пользователя (с ролью "Художник") и авторизации существующих пользователей (включая "Администратора").

2.5.1.3 Обеспечить безопасный доступ к персональным функциям системы.

2.5.1.4 Требуется подключение к серверу для проверки учетных данных.

**2.5.2 Просмотр галереи**

2.5.2.1 «Просмотр галереи»

2.5.2.2 На главном экране должна отображаться лента одобренных публикаций, а также разделы с популярными художниками и выставками. Реализована пагинация для подгрузки контента.

2.5.2.3 Обеспечить пользователю удобный доступ к основному контенту.

2.5.2.4 Требуется доступ к сети для загрузки данных.

**2.5.3 Управление публикациями**

2.5.3.1 «Управление публикациями»

2.5.3.2 Пользователь может создавать новую публикацию, загружая изображение, указывая название, описание и выбирая категории. Может редактировать (доступно только для отклонённых публикаций) или удалять свои публикации.

2.5.3.3 Предоставить инструмент для демонстрации художественных работ.

2.5.3.4 Требуется авторизация.

**2.5.4 Управление выставками и участие в них**

2.5.4.1 «Управление выставками и участие в них»

2.5.4.2 Пользователь может создавать выставку (публичную или приватную), редактировать ее данные (название и описание), добавлять в нее свои работы с любым статусом, удалять любые работы из своих выставок, а также удалять свои выставки. Пользователь может добавлять и удалять свои работы из чужих публичных выставок.

2.5.4.3 Позволить пользователям тематически группировать работы.

2.5.4.4 Требуется авторизация.

**2.5.5 Управление профилем**

2.5.5.1 «Управление профилем»

2.5.5.2 Пользователь может просматривать и редактировать свой профиль (изменять никнейм, почту, пароль), из раздела с профилем просматривать свои публикации и их статусы, свои выставки, а также возможно управление своими публикациями и выставками из текущего вида.

2.5.5.3 Предоставить пользователям возможность настраивать свой профиль для публичного просмотра.

2.5.5.4 Требуется авторизация.

# 2.6 Нефункциональные требования

**2.6.1 Производительность**

Время отклика интерфейса на действия пользователя не должно превышать 100 мс.

Загрузка основной ленты Галереи должна занимать не более 2 секунд при стабильном подключении к Интернету.

Оптимизация размера и кэширование изображений для минимизации трафика.

Приложение должно поддерживать одновременную работу до 100 активных пользователей.

Потребление оперативной памяти приложением не должно превышать 250 МБ.

**2.6.2 Надежность**

Уровень доступности приложения должен составлять не менее 99%.

Время воостановления после сбоя не должно превышать 15 минут.

Потеря данных при восстановлении не допускается.

Частота критических сбоев не должна превышать 1 раза в 3 месяца.

**2.6.3 Доступность**

Приложение должно поддерживать масштабирование шрифтов, заданное в системе.

**2.6.4 Безопасность**

Все передаваемые данные должны шифроваться с использованием протокола HTTPS.

Пароли пользователей должны храниться на сервере в хешированном виде (приложение не занимается хешированием).

Сессии пользователей должны иметь ограниченное время жизни (30 минут).

**2.6.5 Сопровождаемость**

Код должен быть написан в соответствии с принципами Clean Code.

Архитектура приложения должна быть модульной (предпочтительно использование MVVM).

**2.6.6 Переносимость**

Приложение должно корректно работать на различных устройствах под управлением Android 14.0+.

3 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ ДЛЯ ИС

# 3.1 Диаграмма прецедентов

Диаграмма прецедентов [1] представлена на рисунке 1.



Рисунок 1. Диаграмма прецедентов

# 3.2 Описание прецедентов

|  |
| --- |
| Расширенное описание прецедента «Добавление публикации» |
| **Название:** «Добавление публикации»  **Предусловие:** нет  **Действующее лицо:** художник, администратор  **Основной поток:** *Добавление публикации*  Пользователь нажимает на кнопку «Создать публикацию» и переходит в активность для публикации.  Обязательно нужно загрузить изображение, выбрать категории, ввести название и описание.  Работе присваивается статус «PENDING», она становится доступной для просмотра только создателю публикации и администраторам в админ-панели.  **Альтернативный поток:** *нет*  **Постусловие:** пользователя автоматически перебрасывает на активность со всеми публикациями. |

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

# 3.3 Диаграмма состояний

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

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

**1) Выставка создана/обновлена**

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) Выставка обновлена с учётом новых/удалённых публикаций**

Если выставка приватна, публикации в неё может добавлять только создатель выставки. Если публична – любой авторизованный пользователь. При добавлении новой или существующей публикации к выставке она прикрепляется к выставке вне зависимости от её статуса. При удалении публикации из выставки удаляется только запись из связующей таблицы БД.

**4) Выставка функционирует (цикл)**

Пока выставка не удалена, её можно редактировать и в неё можно добавлять любое количество новых публикаций.

**5) Подтверждение удаления**

Когда создатель выставки 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.

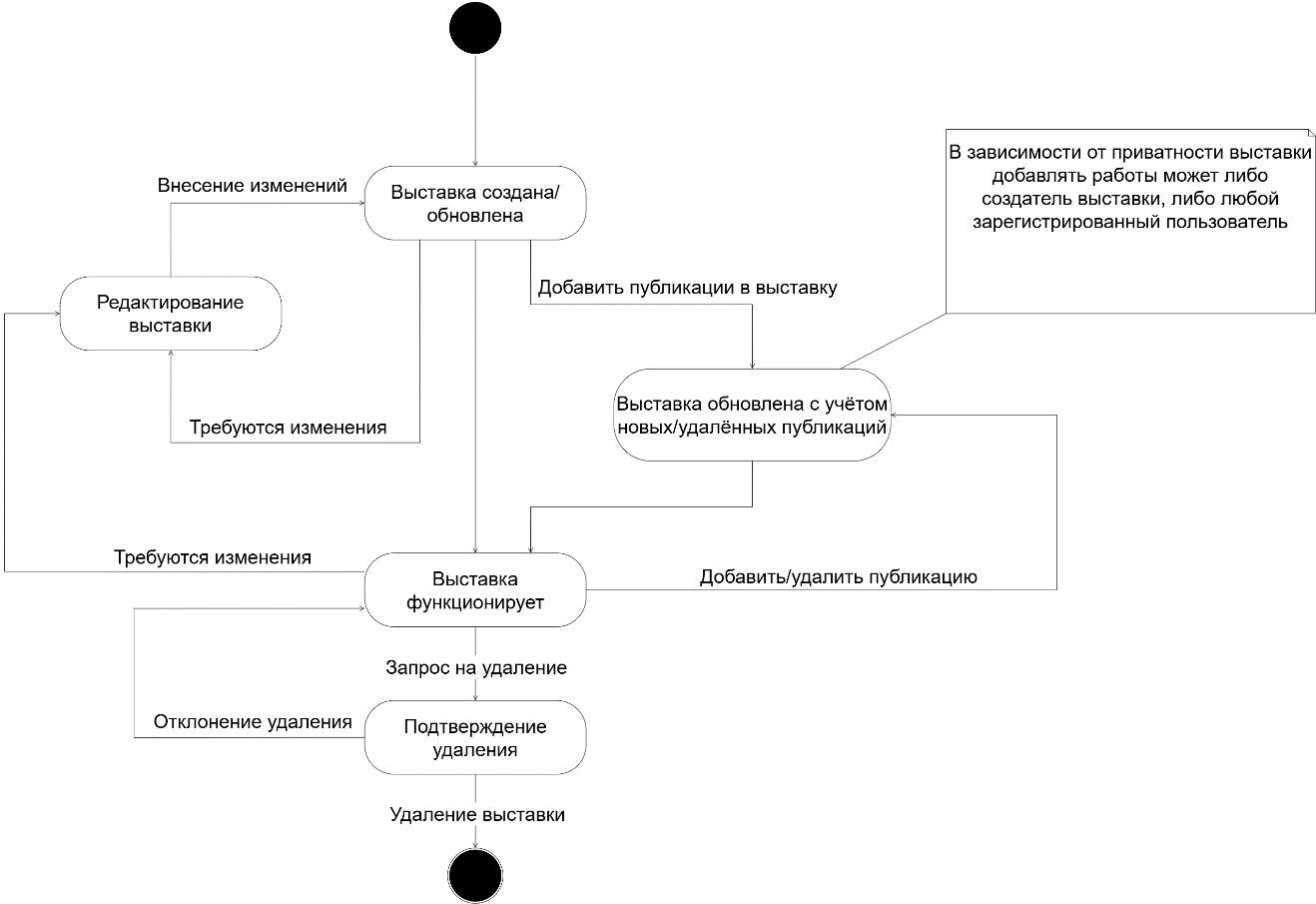


Рисунок 3. Диаграмма состояний

# 3.4 Диаграмма последовательностей

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

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

1. Художник создаёт выставку на форме создания выставки.

2. Запрос передаётся в контроллер.

3. Объект выставки создаётся.

4. Данные выставки заполнены.

5. Выставка отображается в приложении.

6. Художник заходит в детали выставки и хочет добавить выставку.

7. В таблицу связей публикации и выставки добавляется новая запись.

8. Публикация добавляется в выставку.

9. Выставка обновлена.

10. Те же операции по добавлению публикаций могут проделывать другие пользователи, если выставка публична.

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

12. Публикации в выставке безопасно открепляются от неё.

13. Выставка удаляется.

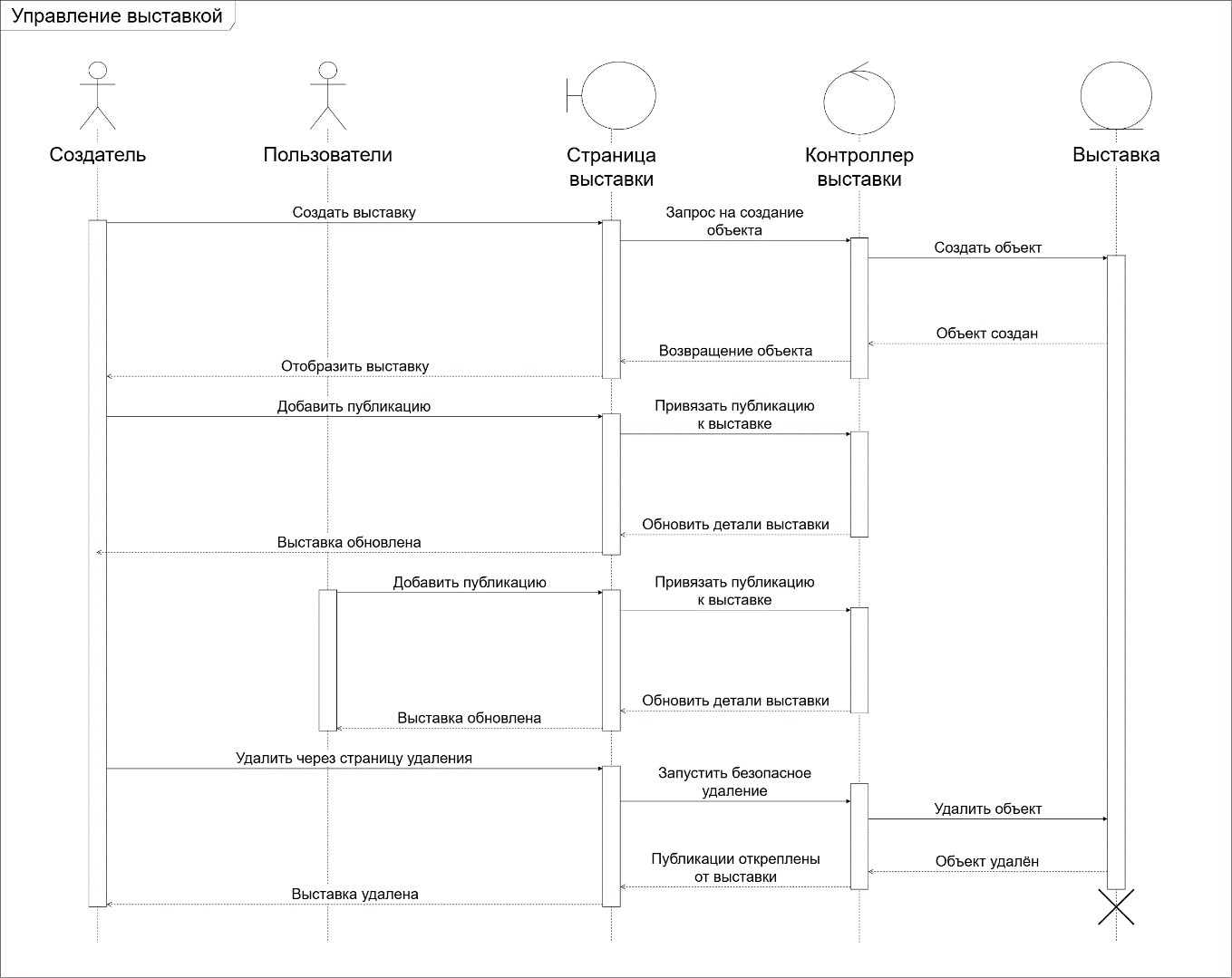


Рисунок 4. Диаграмма последовательностей

# 3.5 Схема бизнес-процесса (BPMN)

Схема бизнес-процесса [5] «Управление выставкой» представлена на рисунке 5.

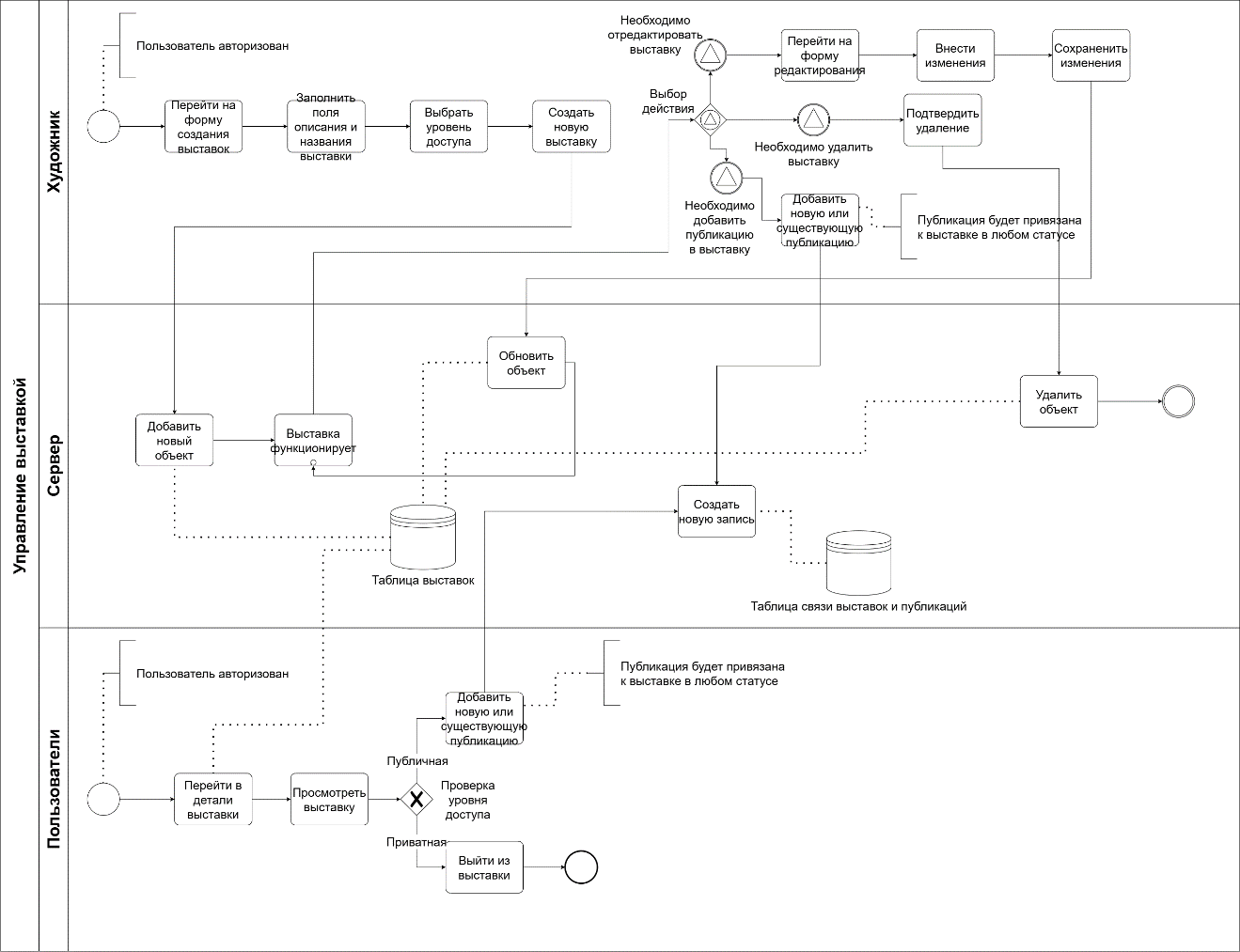


Рисунок 5. Схема бизнес-процесса

# 3.6 Схема взаимодействия программ

Схема взаимодействия программ для объекта «Выставка» представлена на рисунке 6.

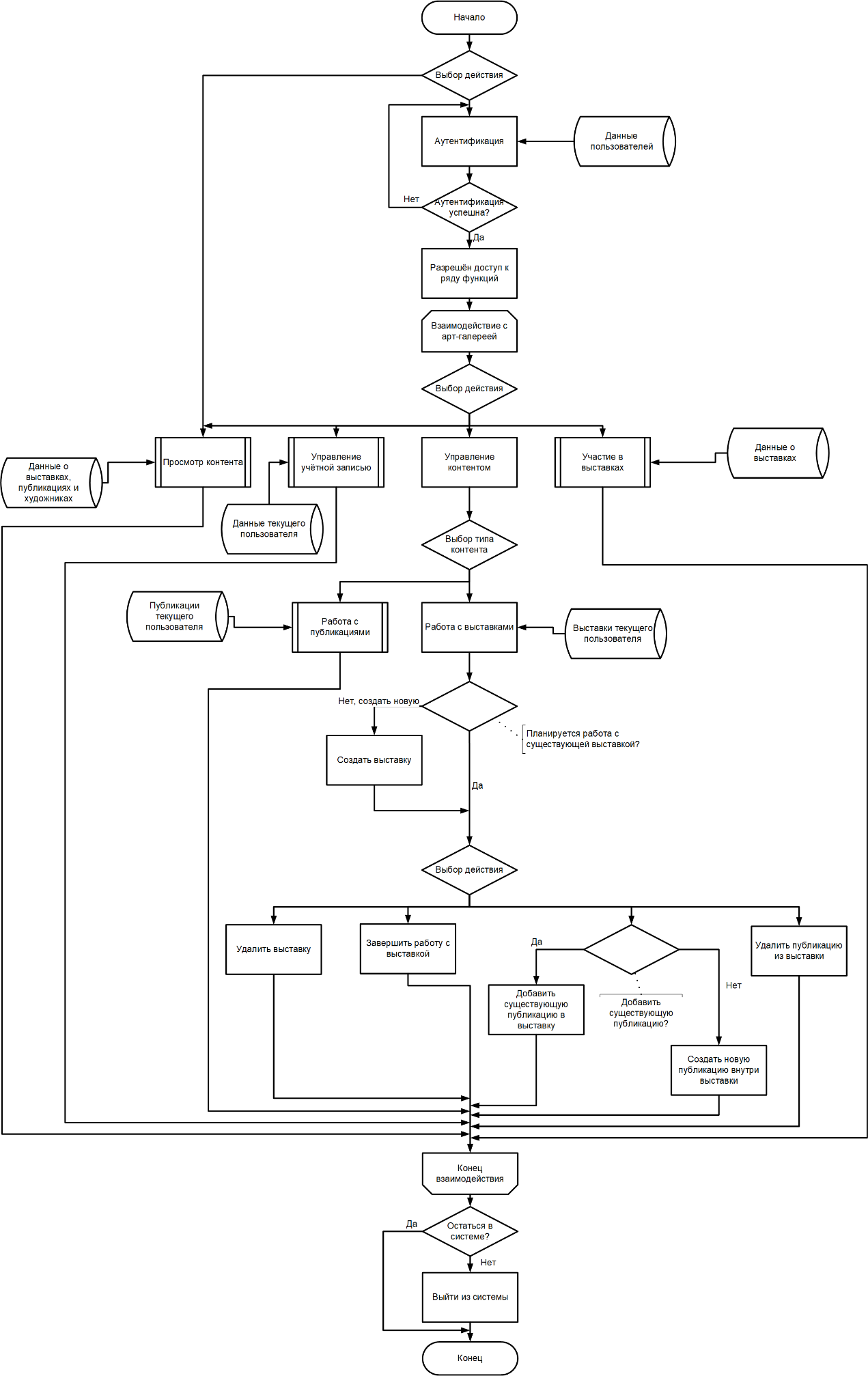


Рисунок 6. Схема взаимодействия программ

4 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ ДЛЯ ИС

# 4.1 Описание базы данных

В мобильном приложении используются серверная и локальная база данных. Серверная база данных была реализована в веб-приложении виртуальной арт-галереи (рис. 7). Серверная база данных содержит таблицы для всех необходимых для работы приложения сущностей.

На диаграмме представлено 9 таблиц:

* artworks – для записей публикаций;
* exhibitions – для записей выставок;
* exhibition\_artworks – для записей публикаций, прикреплённых к выставкам;
* comments – для записей комментариев;
* users – для записей пользователей;
* roles – для записей ролей пользователей;
* likes – для записей лайков на публикациях;
* categories – для записей категорий;
* artwork\_category – для записей категорий, указанных в публикациях.

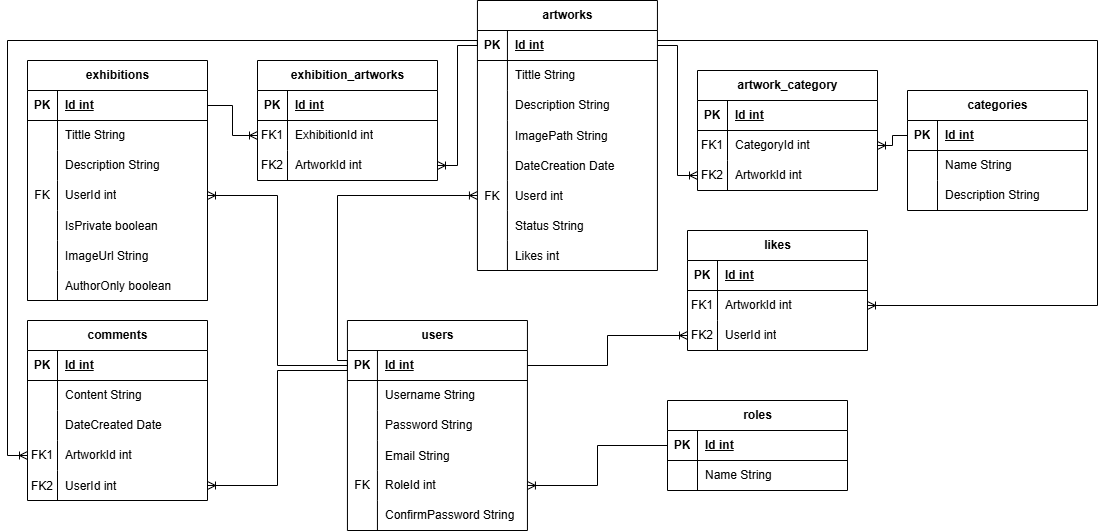


Рисунок 7. ER-диаграмма

# 4.2 Разработка веб-приложения

Для реализации веб-приложения были использованы зависимости.

**Spring Framework [8]**

* **spring-web**: Обеспечивает базовую интеграцию Spring с веб-приложениями (HTTP-клиенты, REST-поддержка).
* **spring-webmvc**: Реализует MVC-паттерн для веб-приложений (контроллеры, представления).
* **spring-orm**: Интеграция с ORM-библиотеками (Hibernate, JPA).
* **spring-tx**: Управление транзакциями (аннотации @Transactional).
* **spring-context**: Ядро Spring (DI, IoC-контейнер, аннотации @Component, @Autowired).
* **spring-data-jpa**: Упрощает работу с JPA (репозитории, автоматические запросы).
* **spring-data-commons**: Общие компоненты Spring Data (Pagination, CrudRepository).

**Spring Security [9]**

* **spring-security-web**: Защита веб-приложений (фильтры, аутентификация, авторизация).
* **spring-security-config**: Конфигурация через аннотации и XML (класс WebSecurityConfigurerAdapter).
* **spring-security-core**: Базовые компоненты безопасности (роли, права, шифрование).

**Thymeleaf [10]**

* **thymeleaf**: Шаблонизатор для HTML (динамические шаблоны на стороне сервера).
* **thymeleaf-spring5**: Интеграция с Spring 5 (поддержка Spring-выражений, MVC).
* **thymeleaf-extras-springsecurity5**: Дополнения для Spring Security (отображение данных пользователя).
* **thymeleaf-extras-java8time**: Поддержка Java 8 Time API в шаблонах (форматирование дат).

**Базы данных**

* **hibernate-core**: ORM-фреймворк (маппинг сущностей, HQL).
* **hibernate-validator**: Валидация данных через аннотации (например, @NotNull).
* **mysql-connector-java**: Драйвер для подключения к MySQL.
* **HikariCP**: Пул соединений для БД (высокая производительность).

**JPA [11]**

* **javax.persistence-api**: Спецификация JPA (аннотации @Entity, @Table).

**Веб и Servlet**

* **javax.servlet-api**: API для работы с сервлетами (классы HttpServlet, Filter).
* **javax.servlet.jsp-api**: API для JSP-страниц.
* **jstl**: Теги для JSP (упрощение логики в шаблонах).

**Утилиты**

* **commons-fileupload**: Загрузка файлов на сервер (обработка multipart/form-data).
* **lombok**: Генерация кода (геттеры, сеттеры через аннотации @Data).
* **javax.annotation-api**: Аннотации общего назначения (например, @PostConstruct).

**Логирование**

* **slf4j-api**: Интерфейс для логирования (абстракция над реализациями).
* **slf4j-simple**: Простая реализация SLF4J (логи в консоль).

**Tomcat**

* **tomcat-jdbc**: Реализация DataSource для Tomcat (пул соединений).

**Прочее**

* **maven-antrun-plugin**: Плагин Maven для выполнения Ant-задач (например, копирование WAR-файла).
* **maven-war-plugin**: Сборка WAR-архива.
* **maven-compiler-plugin**: Настройка компиляции (Java 16).

# 4.3 Разработка приложения для информационной системы

Для разработки использовался язык программирования Java. Были разработаны модели, контролеры, репозитории, сервисы и представления.

**Модели:**

1. Artwork – модель для хранения публикаций.
2. Exhibition – модель для хранения выставок.
3. Role – модель для хранения ролей пользователей.
4. User – модель профилей пользователей.

**Представления:**

1. index – просмотр галереи.
2. artwork
   1. create – создание публикации;
   2. details – детали публикации;
   3. edit – редактирование публикации со статусом «REJECTED»;
   4. list – просмотр всех публикаций.
3. exhibition
   1. add-exist-artwork – добавление существующей публикации к выставке;
   2. create – создание выставки;
   3. details – детали выставки с возможностью добавить в неё публикацию;
   4. edit – редактирование названия, описания и приватности выставки;
   5. list – просмотр всех выставок.
4. user
   1. liked – просмотр понравившихся публикаций;
   2. list – просмотр всех пользователей;
   3. profile – просмотр профиля пользователя;
   4. settings – редактирование профиля пользователя.
5. auth
   1. access-denied – страница запрещённого доступа;
   2. login – страница авторизации;
   3. register – страница регистрации.

**Контроллеры:**

1. ArtworkController – контроллер для работы с публикациями;
2. AuthController – контроллер аутентификации пользователей;
3. ExhibitionController – контроллер для работы с выставками;
4. HomeController – контроллер с реализацией отображения публикаций, художников и выставок;
5. ProfileController – контроллер для работы с профилями пользователей.

**Репозитории:**

1. ArtworkRepository – работа с публикациями;
2. ExhibitionRepository – работа с выставками;
3. RoleRepository – работа с ролями;
4. UserRepository – работа с пользователями.

**Сервисы:**

1. ArtworkService – бизнес-логика для публикаций;
2. CategoryService – бизнес-логика для категорий;
3. CustomUserDetailsService – бизнес-логика профилей пользователей;
4. ExhibitionService – бизнес-логика для выставок;
5. UserService – бизнес-логика для управления пользователями.

# 4.4 Диаграмма компонентов

Диаграмма компонентов [12] представлена на рисунке 8, на которой показана связь всех компонентов системы.

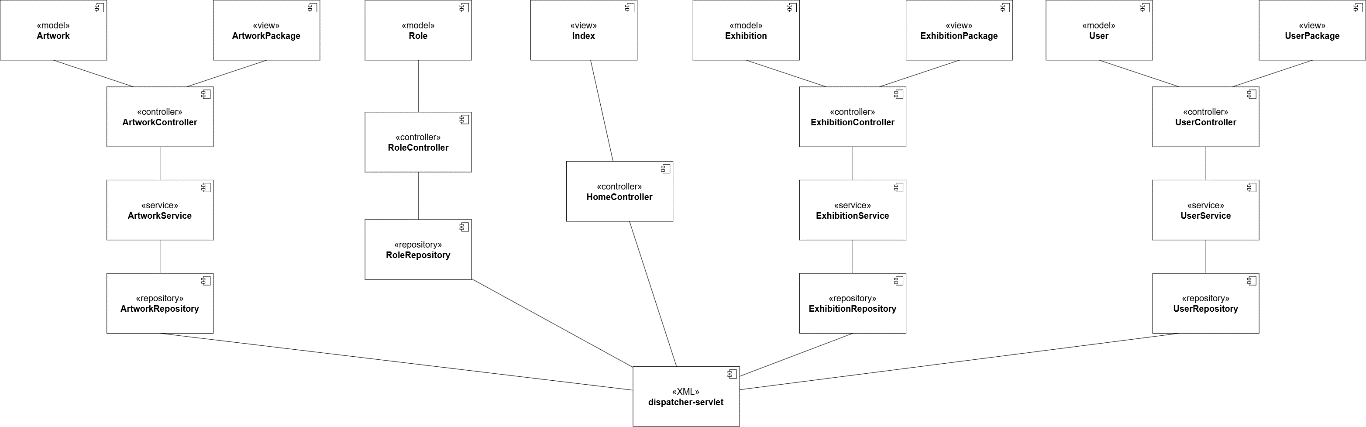


Рисунок 8. Диаграмма компонентов

# 4.5 Диаграмма развёртывания

Диаграмма развёртывания [13] представлена на рисунке 9.

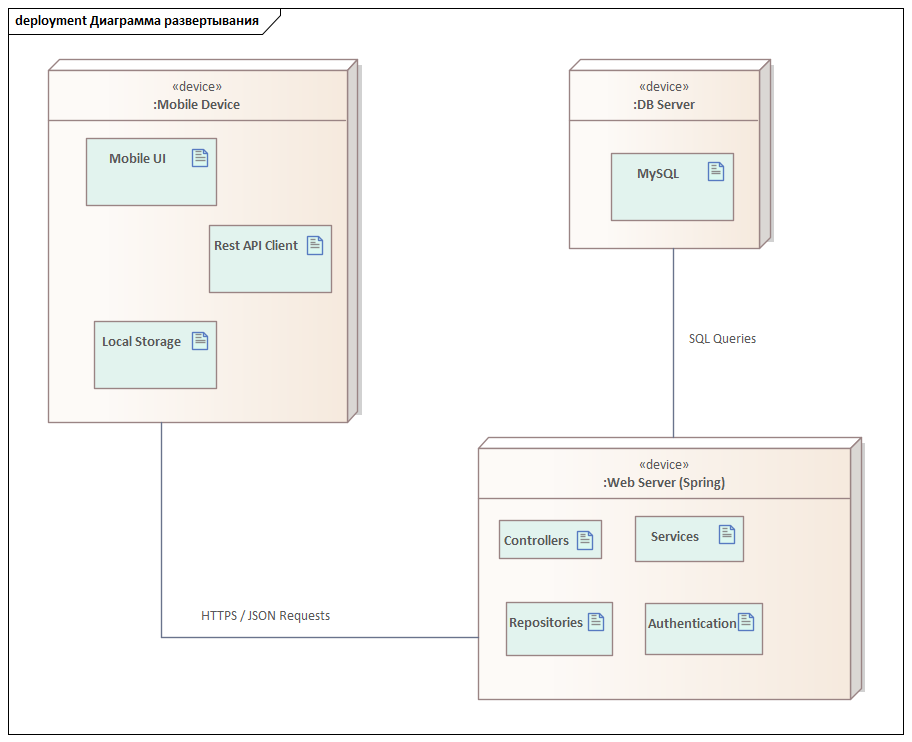


Рисунок 9. Диаграмма развёртывания

# 4.6 Репозиторий проекта

Ссылка на репозиторий: <https://github.com/ShutovaTE/VAG-Java>

# 4.7 Реализация

Основной код публикаций, выставок и пользователей представлен в приложениях А, Б и В соответственно.

# 4.8 Страницы приложения

Страницы готового приложения представлены на рисунках 10-23.

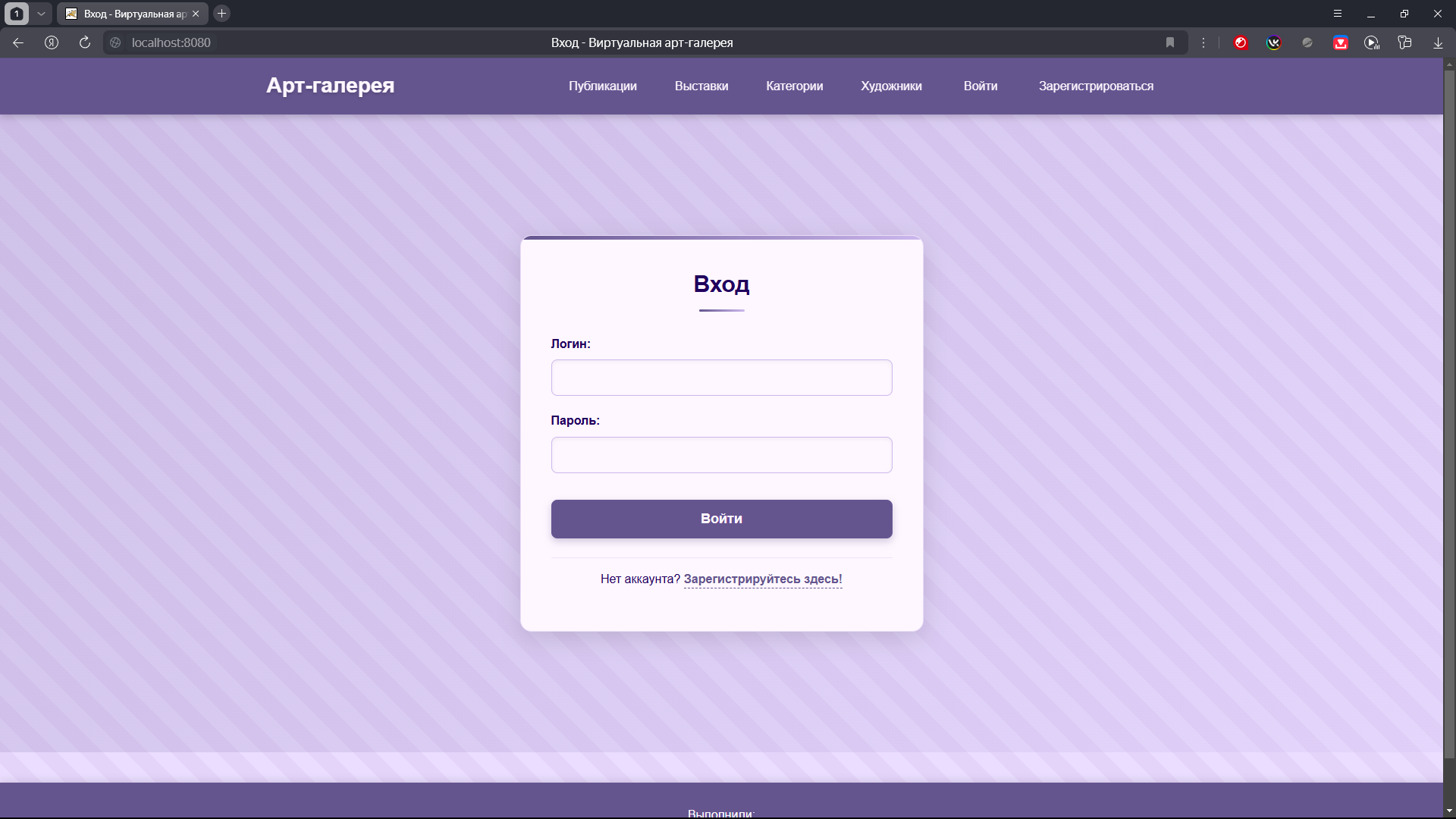


Рисунок 10. Форма авторизации

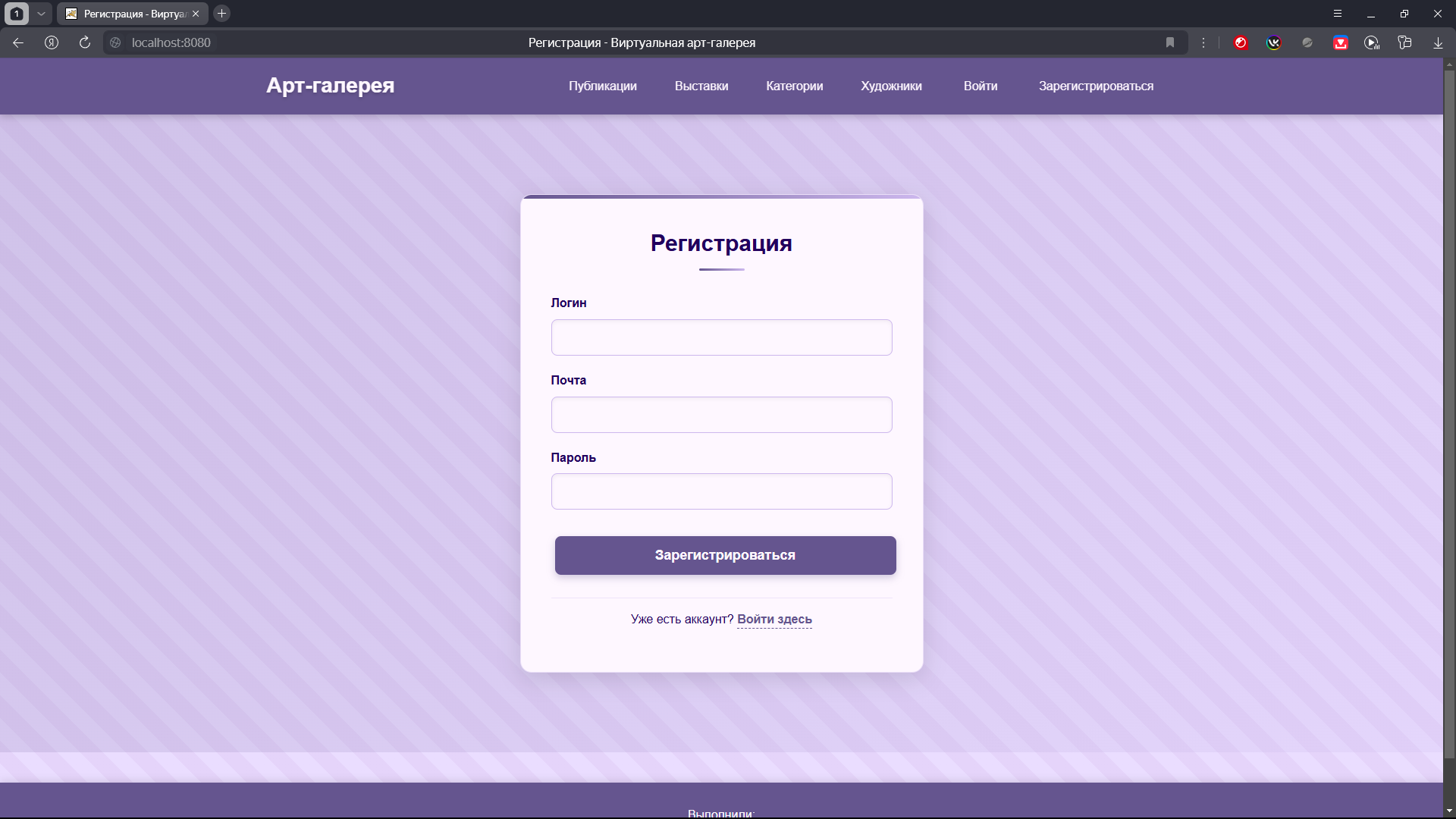


Рисунок 11. Форма регистрации

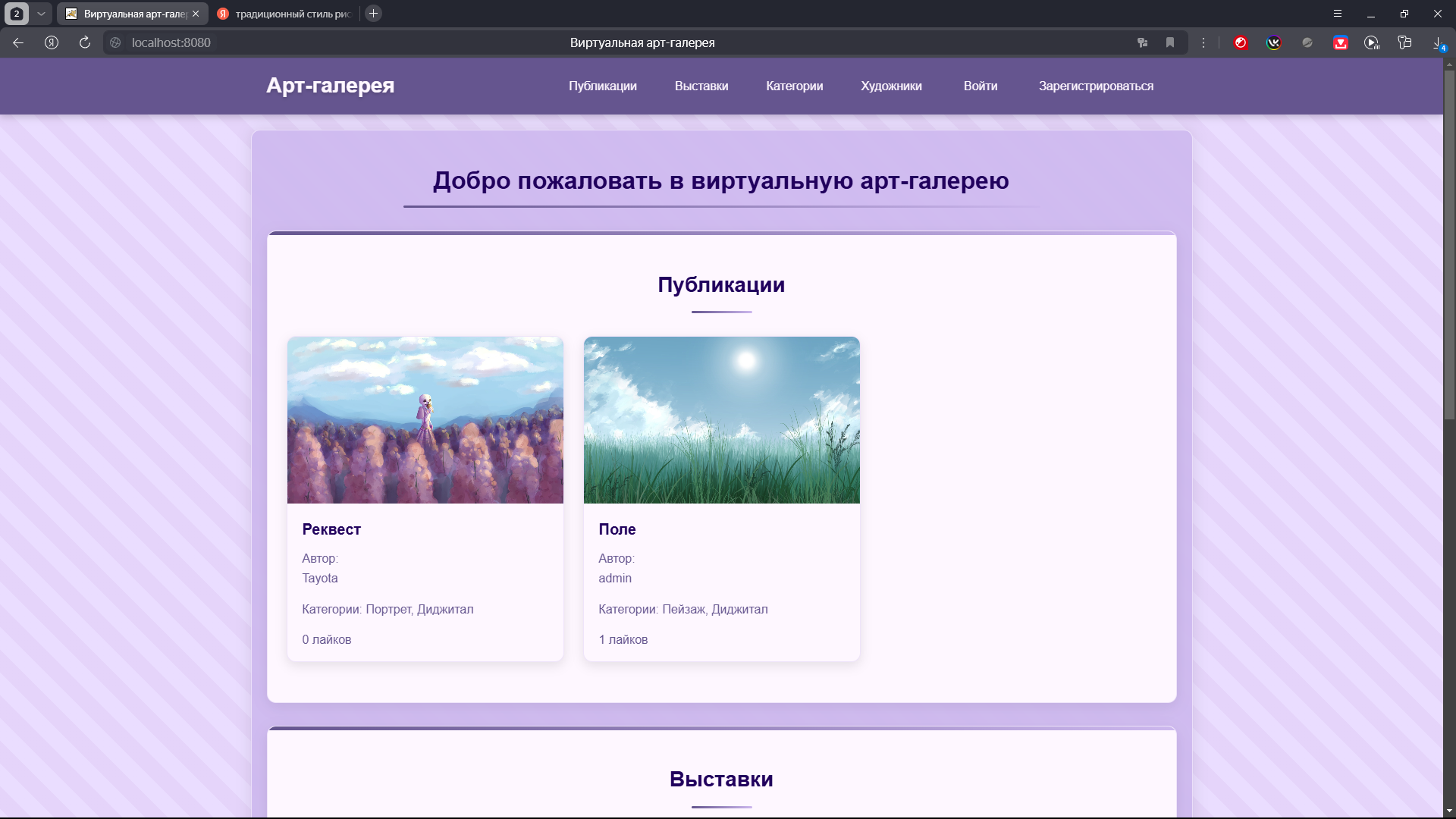


Рисунок 12. Часть главной страницы

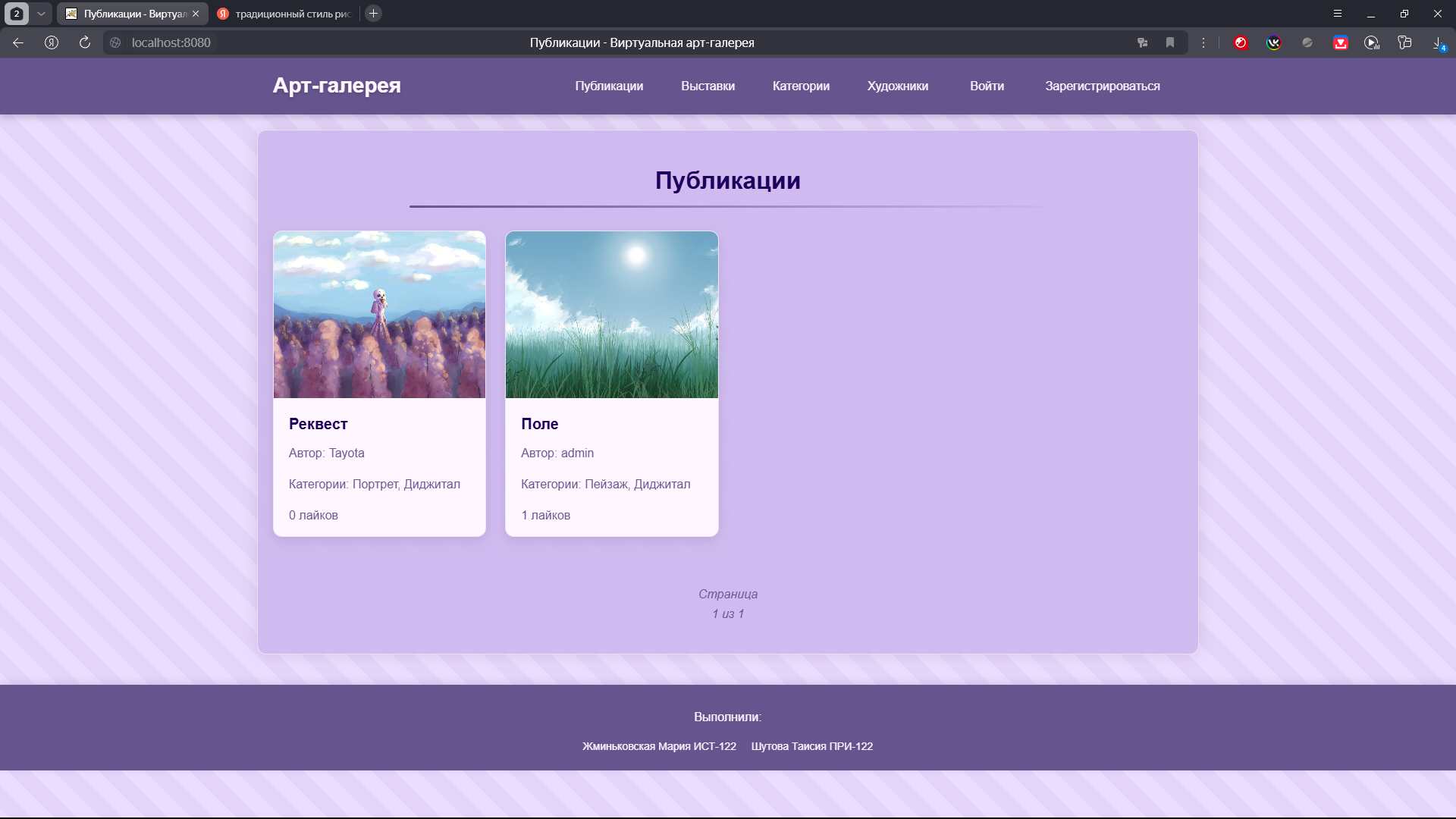


Рисунок 13. Отображение публикаций

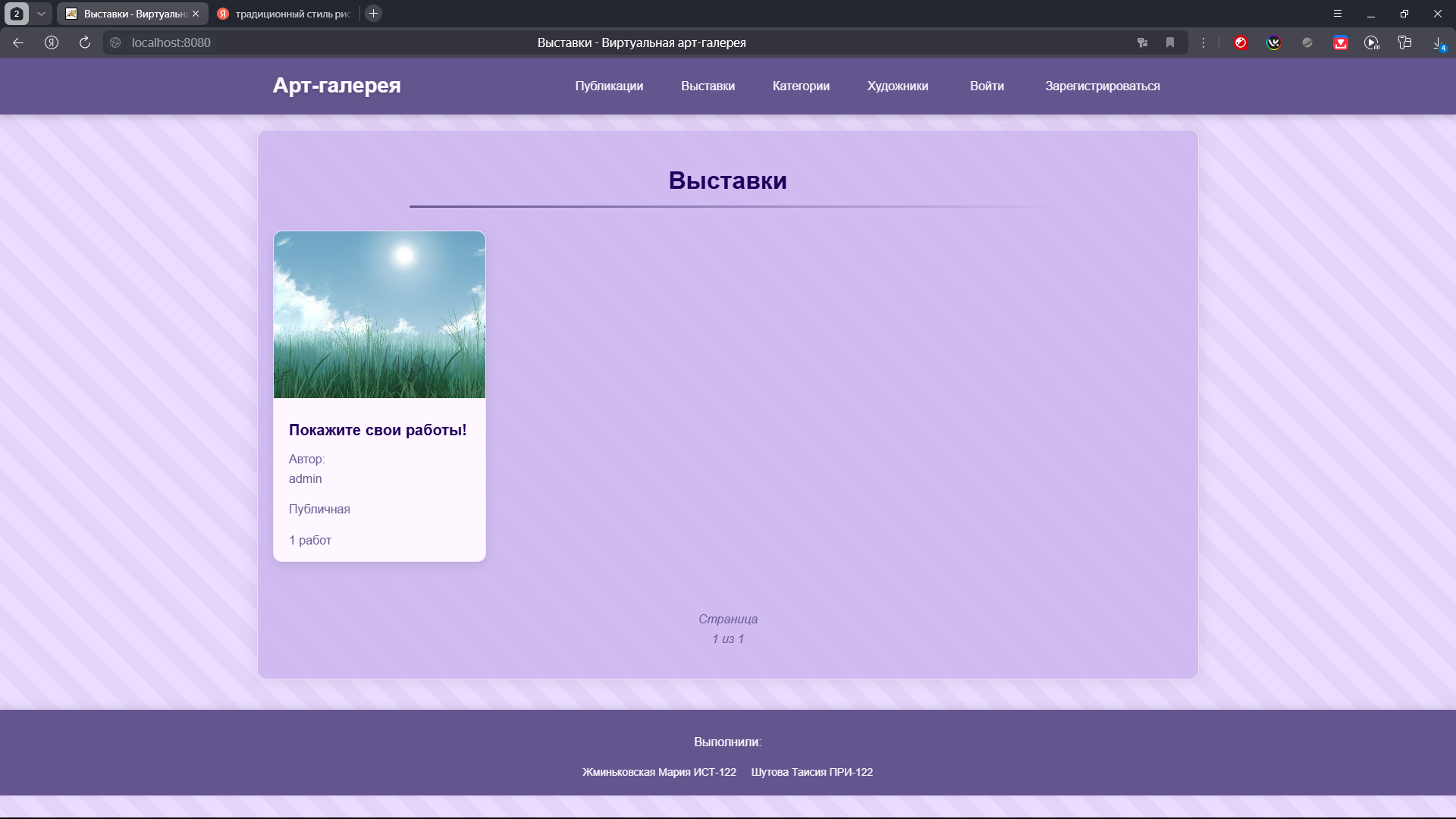


Рисунок 14. Отображение выставок

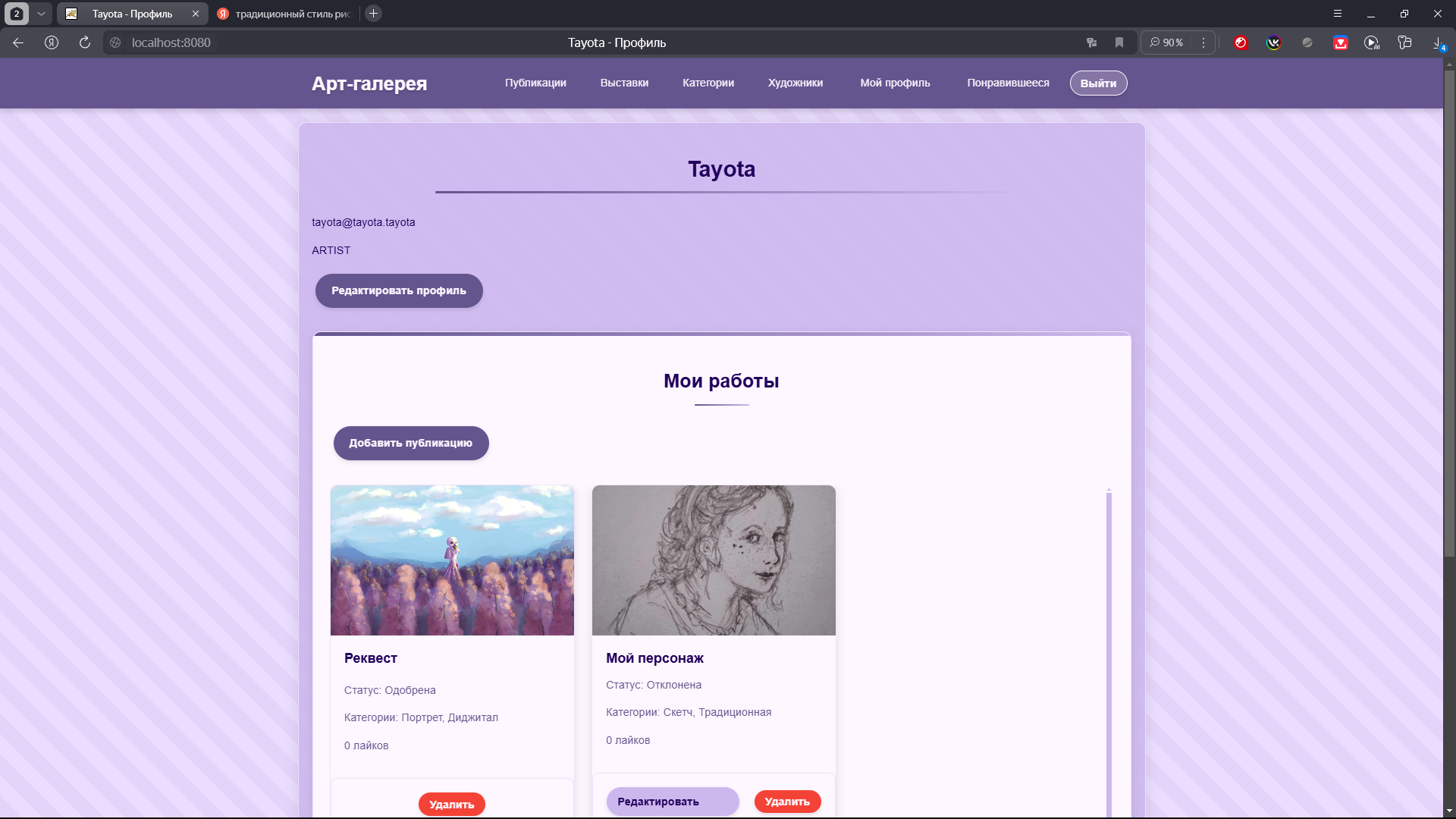


Рисунок 15. Частичный профиль пользователя

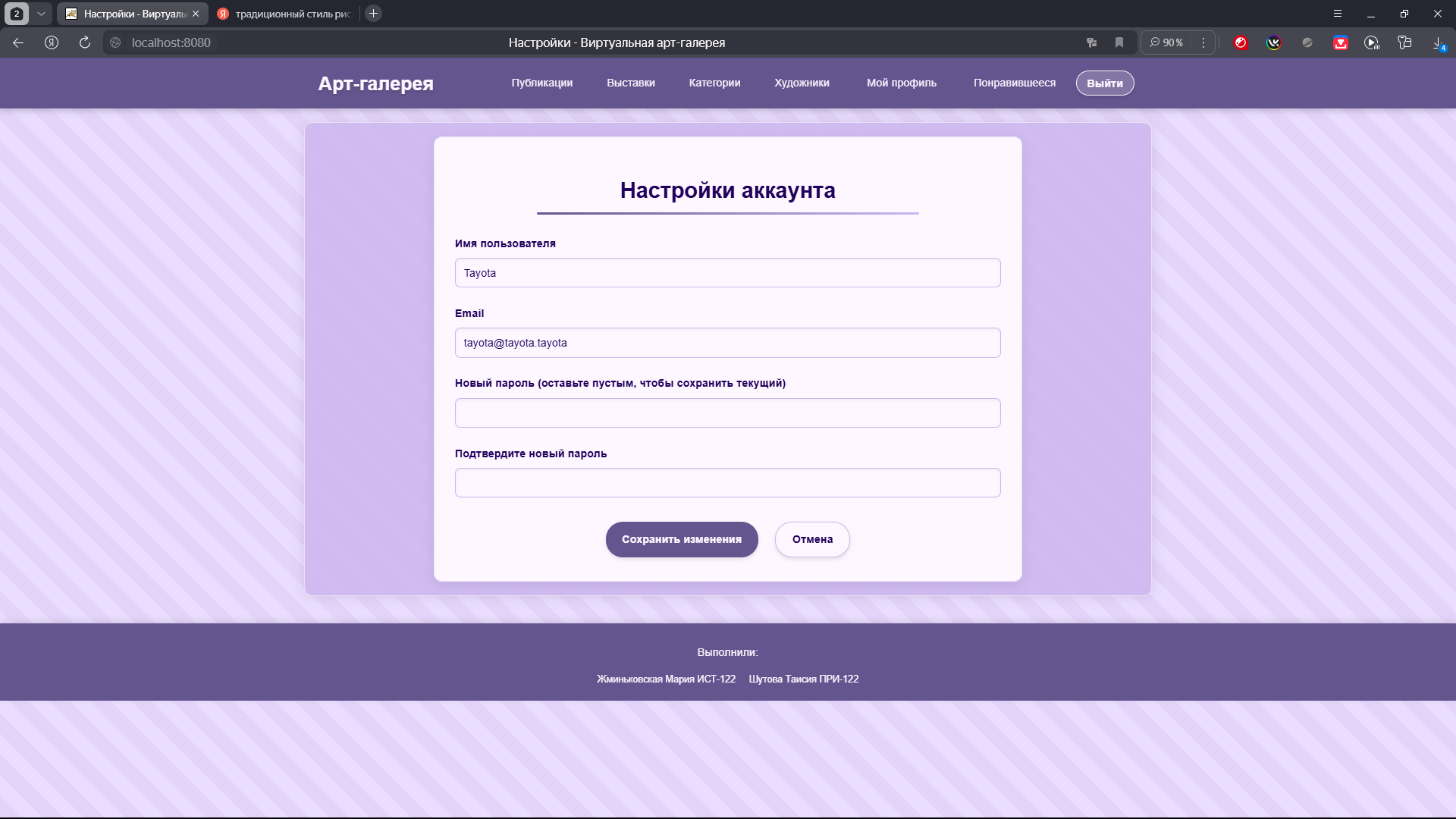


Рисунок 16. Настройки аккаунта

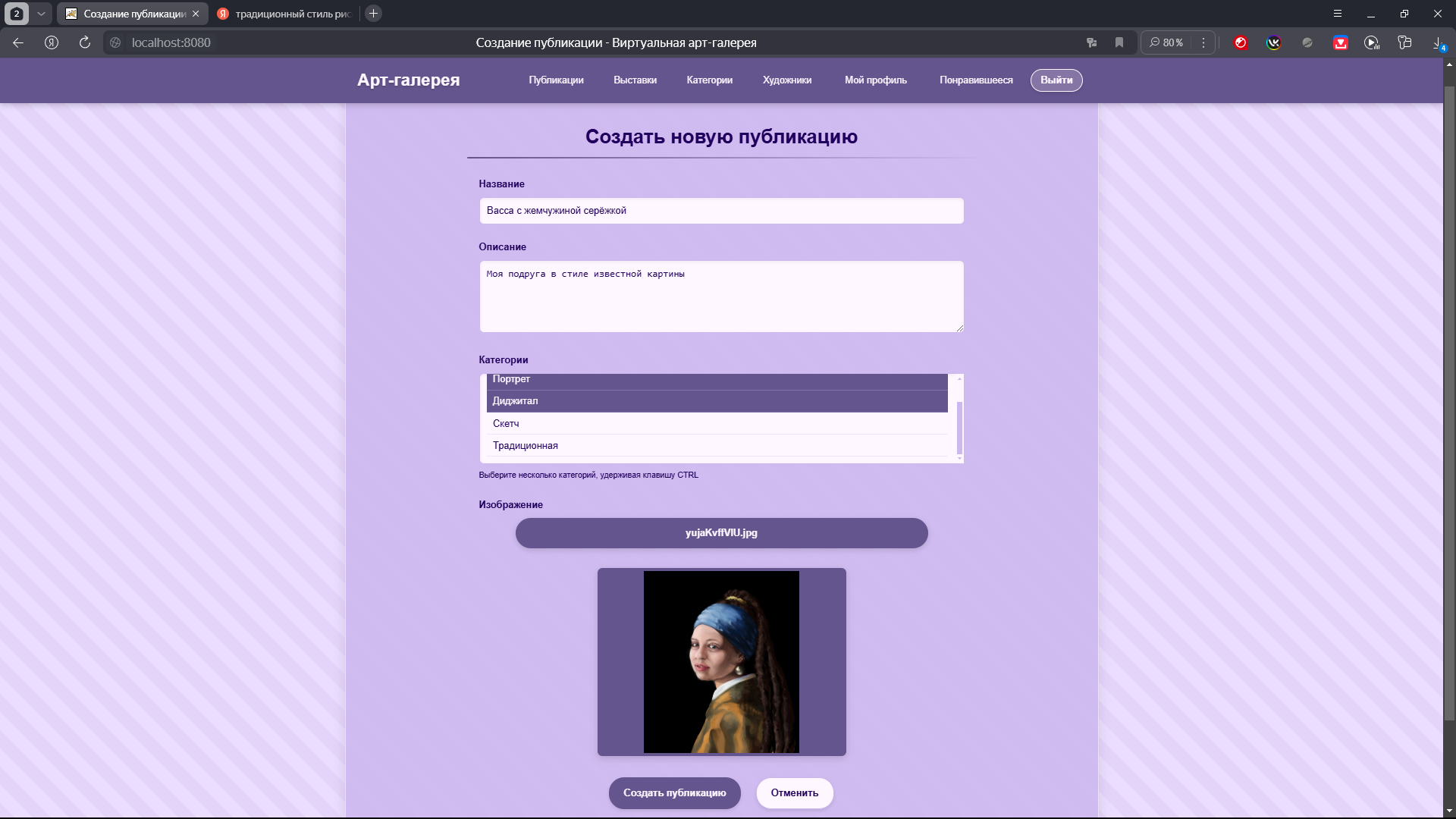


Рисунок 17. Создание публикации

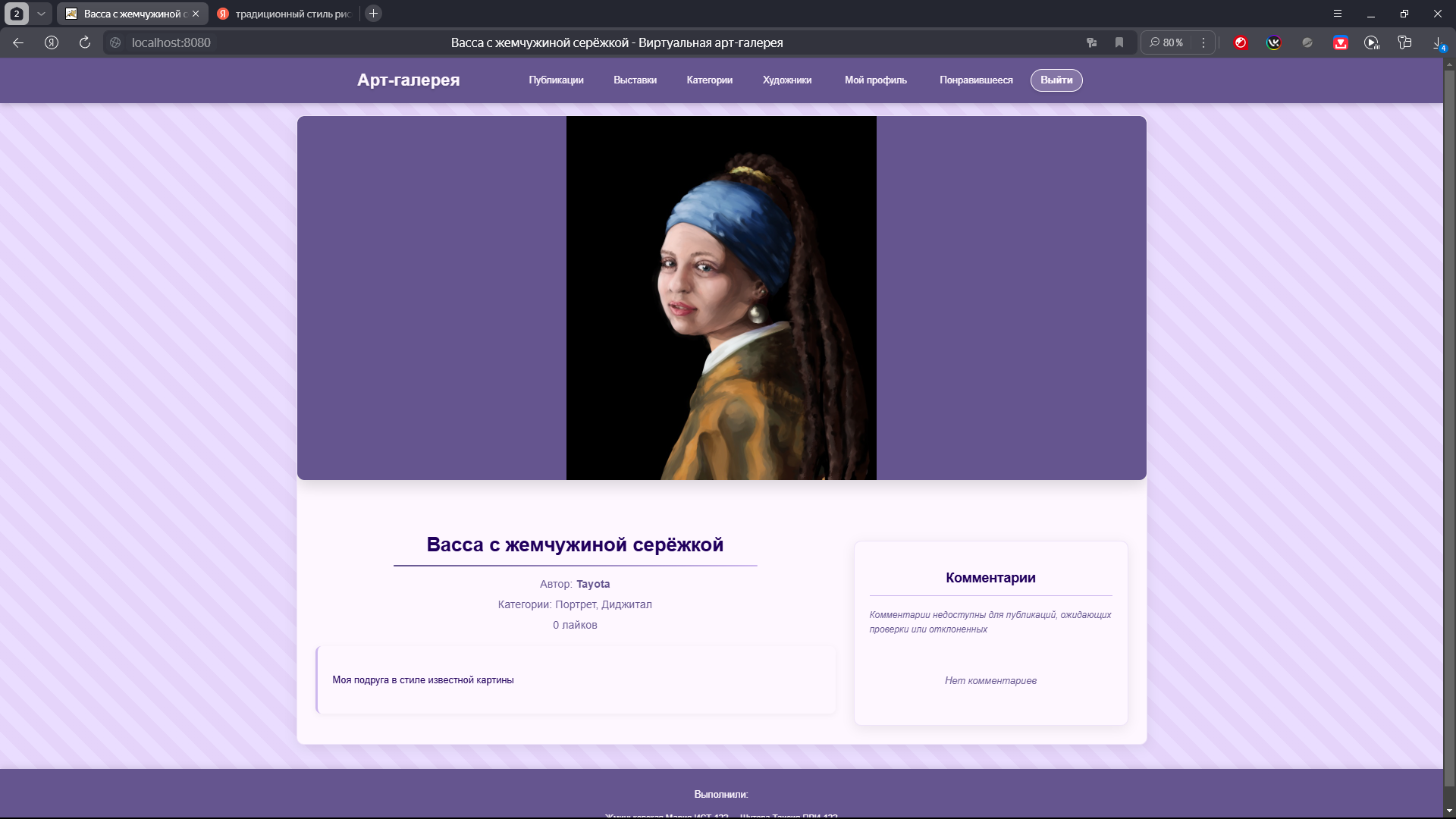


Рисунок 18. Детали неодобренной публикации

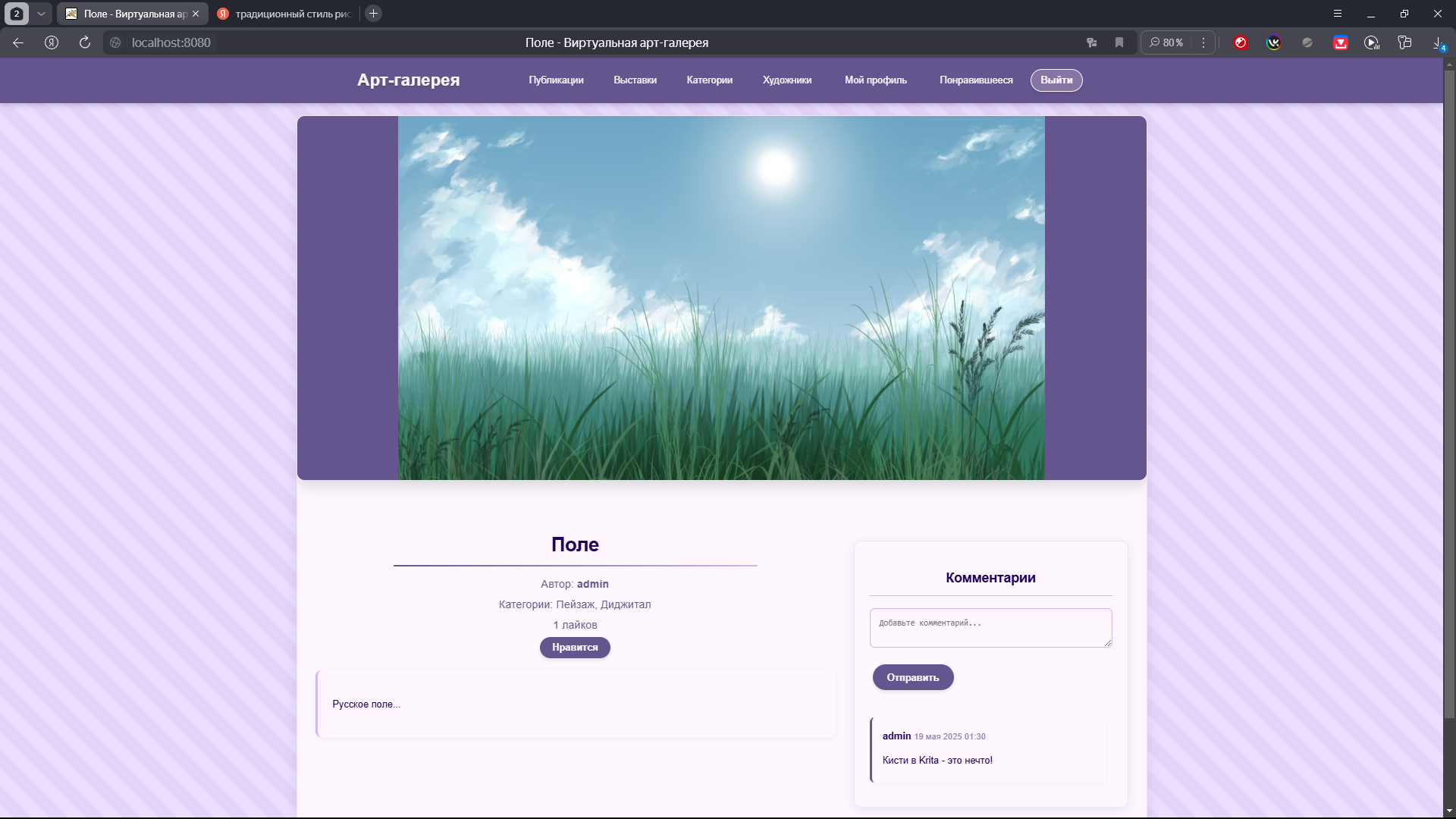


Рисунок 19. Детали одобренной публикации

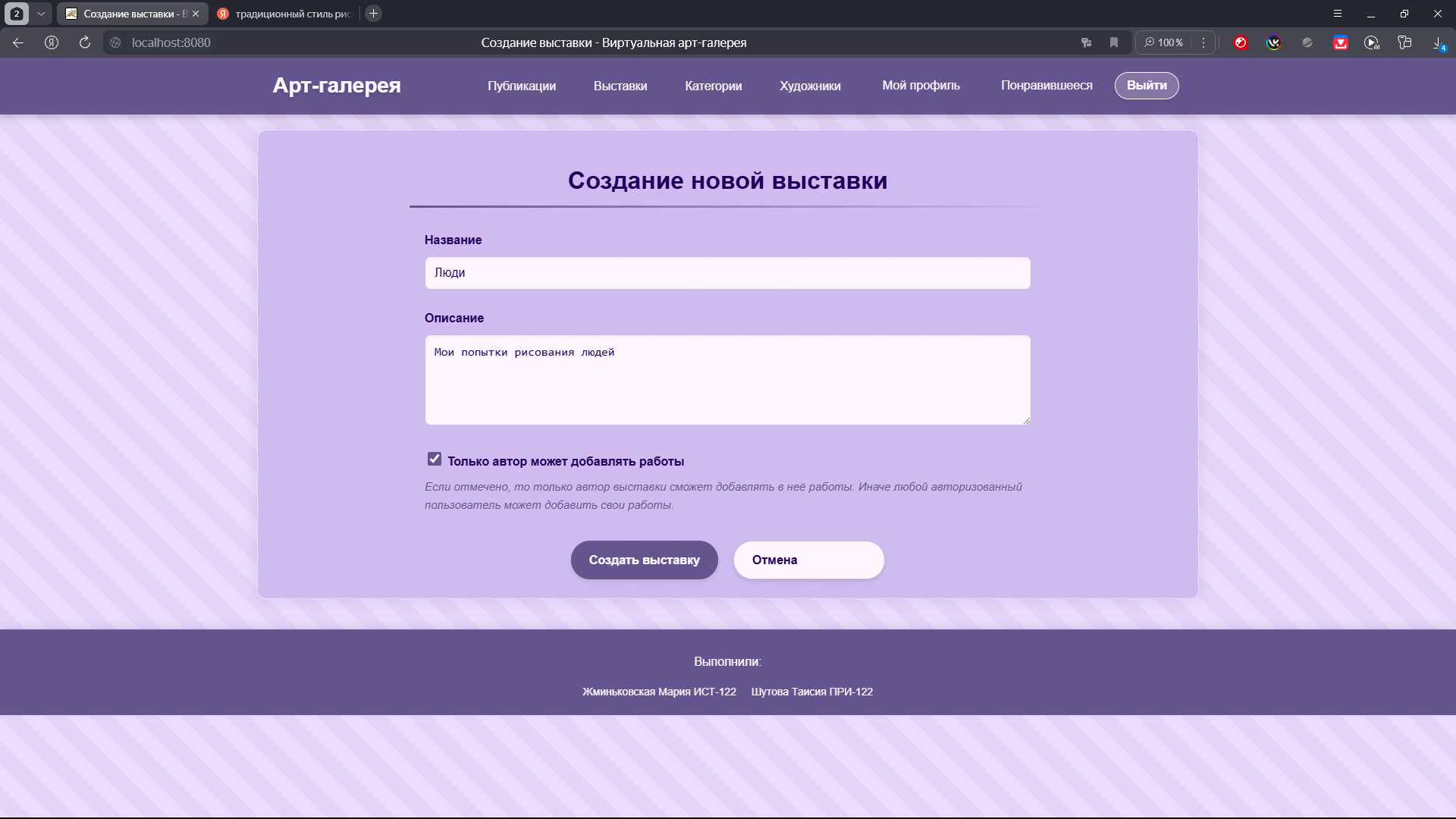


Рисунок 20. Создание выставки

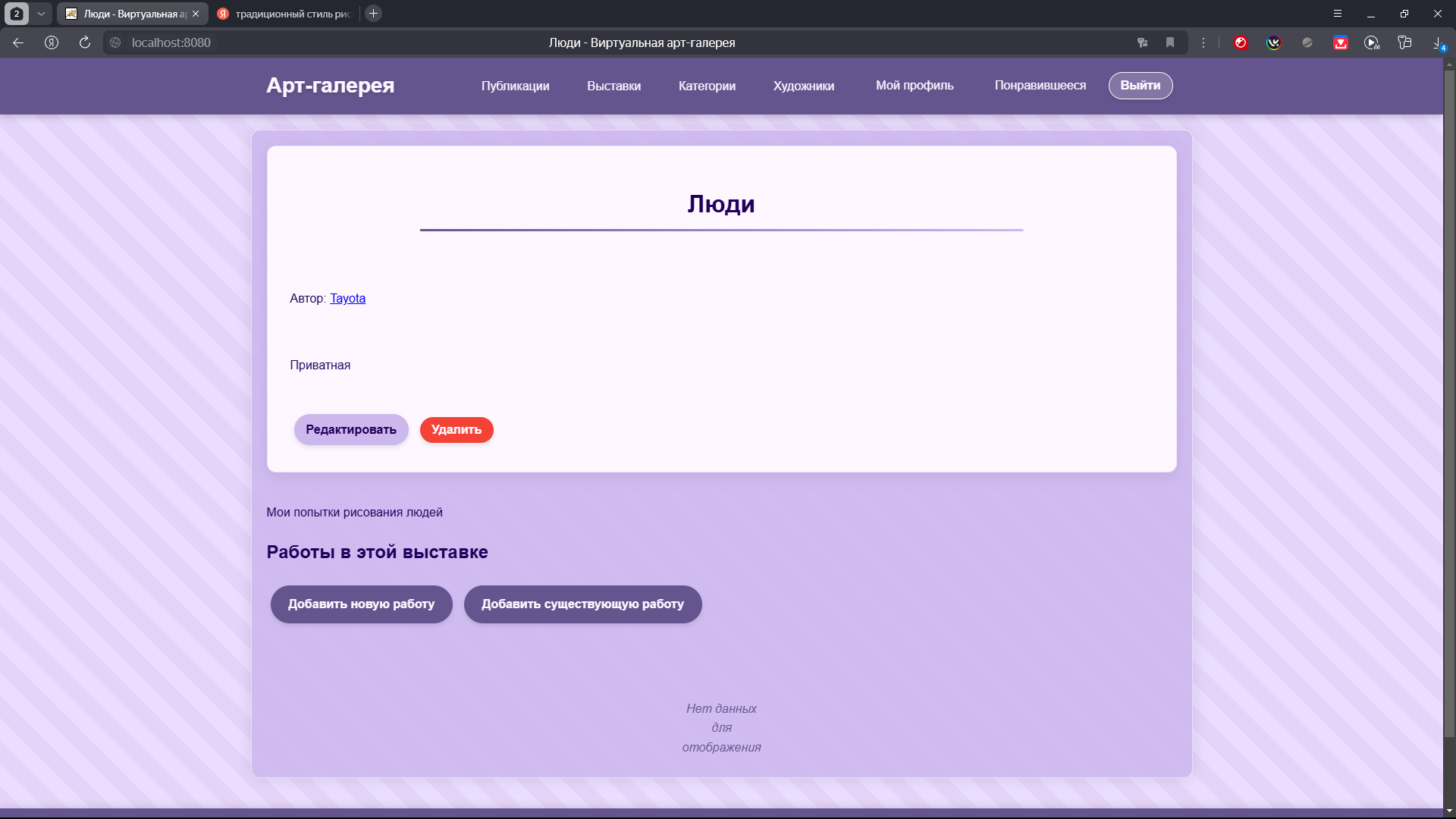


Рисунок 21. Детали выставки

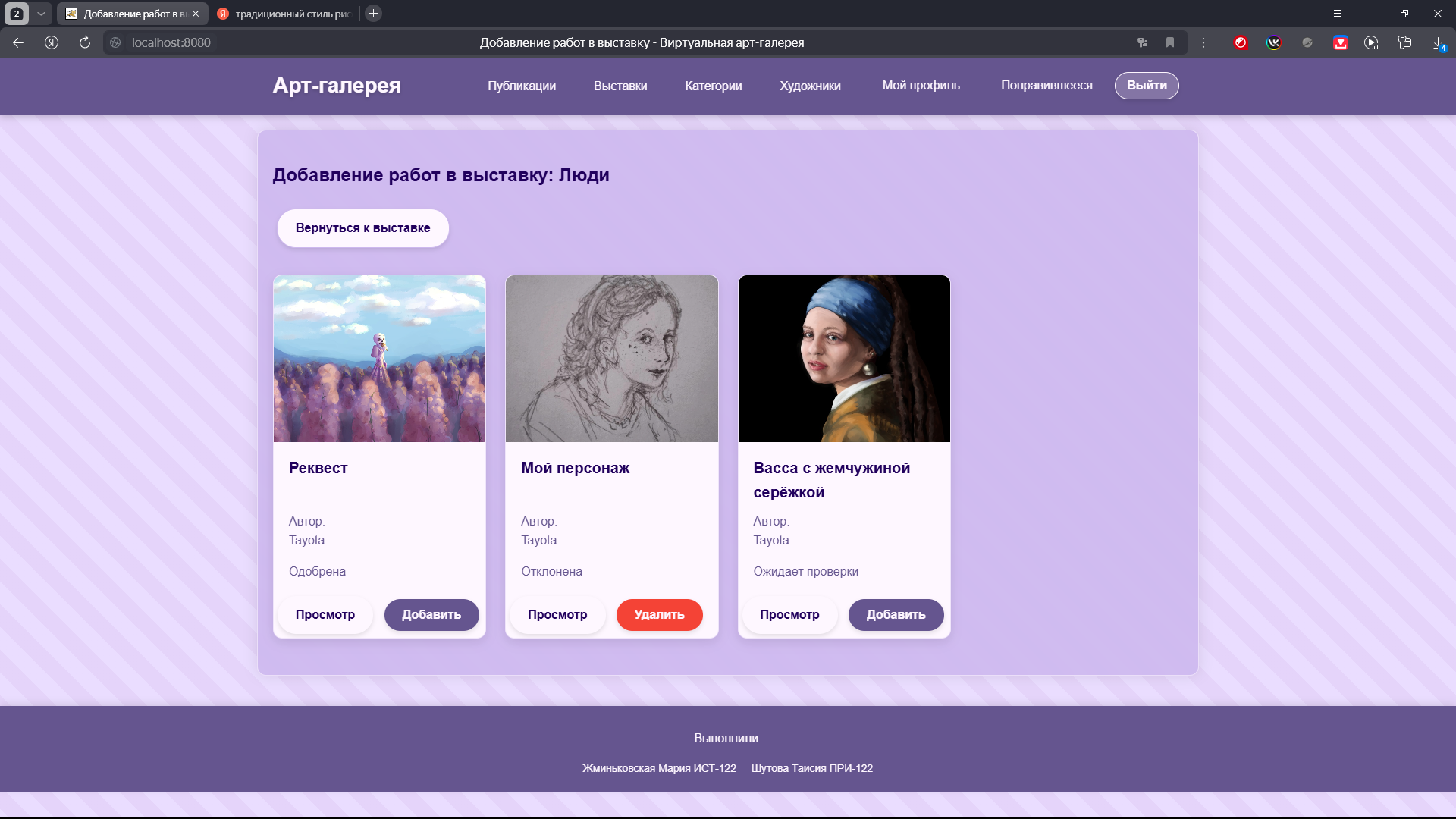


Рисунок 22. Добавление существующих работ в выставку

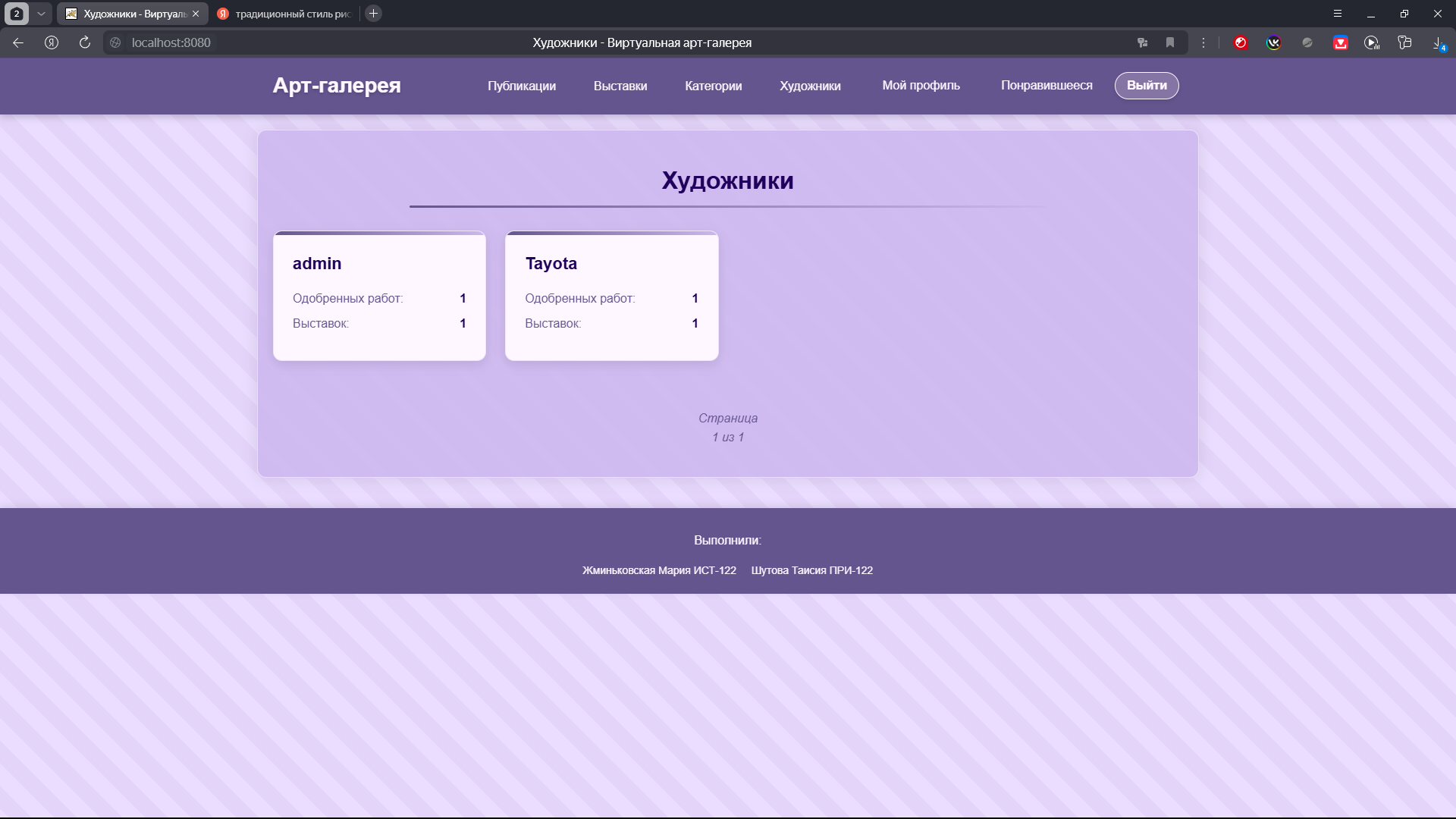


Рисунок 23. Отображение пользователей

5 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ

Для того чтобы определить производительность разрабатываемой системы необходимо провести нагрузочное тестирование с постепенным увеличением нагрузки. В качестве объектов тестирования были выбраны объекты «Выставка» и «Комментарий». Нагрузочное тестирование позволит выявить уязвимые места в быстром добавлении выставок и комментариев различными пользователями, выполняющими действия одновременно.

Нагрузочное тестирование проводится с помощью инструмента Apache JMeter [14]. Были определены следующие параметры потока:

* количество пользователей в потоке – 10;
* период запуска потоков – 10 секунд;
* количество итераций – 100;
* единственный пользователь для всех итераций.

Объекты «Выставка» и «Комментарий» будут подвержены нагрузочному тестированию согласно последовательности действий, описанной ниже.

1. GET-запрос на страницу авторизации. Необходимо для того, чтобы авторизовать пользователя.
2. POST-запрос на авторизацию. Авторизуются 10 различных пользователей, описанных в CSV-файле, созданном заранее.
3. GET-запрос на переход к деталям публикации с ID 8.
4. POST-запрос на добавление комментария. В параметрах комментария вводится текст комментария.
5. GET-запрос на страницу отображения всех существующих выставок. Один из вариантов перехода к функции «Создать выставку».
6. GET-запрос на переход на форму создания выставки.
7. POST-запрос на создание выставки. В параметрах выставки указывается приватность выставки (приватная), описание выставки и название выставки.
8. GET-запрос на страницу отображения всех существующих выставок. Необходимо для проверки быстрой загрузки данных.
9. GET-запрос на детали конкретной выставки. Необходимо для проверки быстрой загрузки всех данных выставки.

Нагрузочное тестирование с указанными выше условиями и параметрами было успешно проведено без перебоев, ошибок и провальных попыток исполнения запроса (рис. 24, 25).

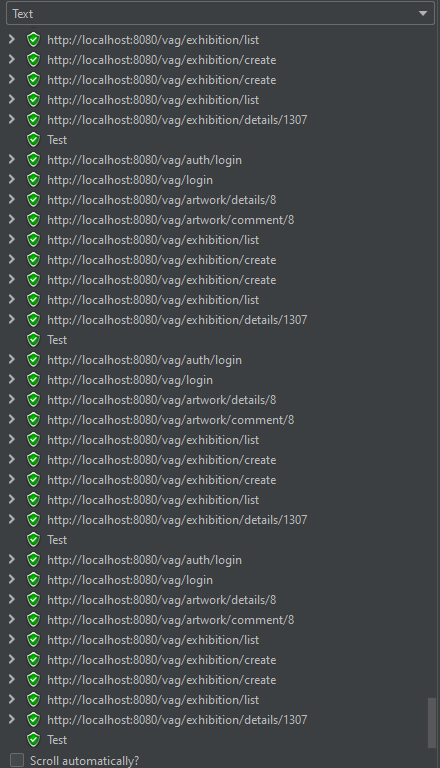


Рисунок 24. Нагрузочное тестирование выставок и комментариев

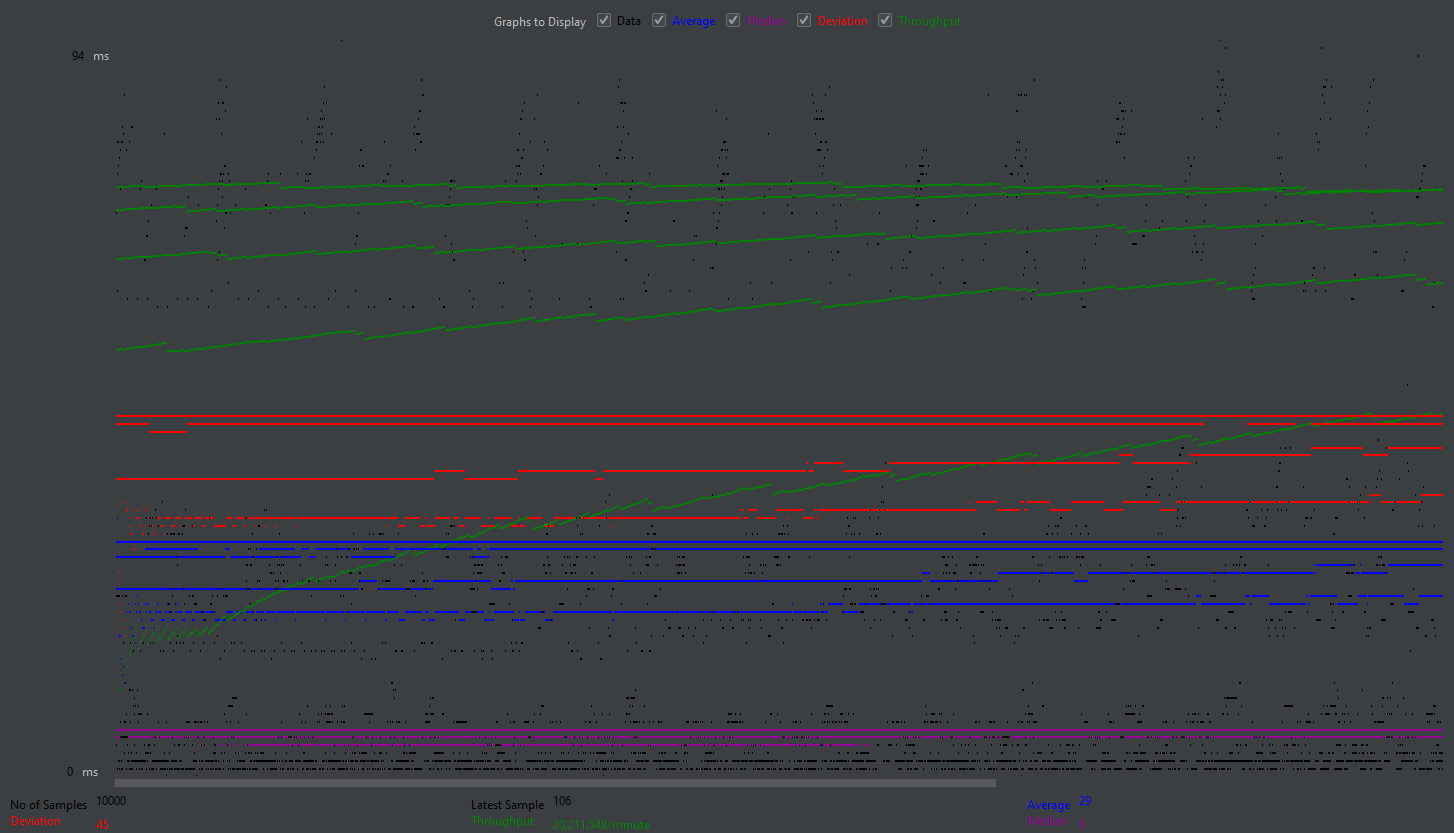


Рисунок 25. Графический результат тестирования

Нагрузочное тестирование показало, что система достаточно оптимизирована для текущих требований. Проект не требует доработки для оптимизации. Для успешного прохождения тестирования в проекте имеются следующие аспекты:

* Эффективная работа с комментариями.
* Оптимизация управления выставками.
* Параллельная обработка запросов (независимая обработка запросов).
* Оптимизированная загрузка списков с помощью EntityGraph и пагинации.
* Эффективное управление транзакциями.
* Оптимизированное отображение деталей с помощью Join Fetch.

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 и выставками, взаимодействия с художниками и посетителями, а также просмотра 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 комментирования.

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. UML Use Case Diagrams: A Complete Guide [Электронный ресурс] // Visual Paradigm. – 2023. – URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/> (дата обращения: 16.09.2024)
2. Class Diagrams in UML Explained [Электронный ресурс] // Lucidchart. – 2024. – URL: <https://www.lucidchart.com/pages/uml-class-diagram> (дата обращения: 17.09.2024)
3. State Machine Diagrams in UML [Электронный ресурс] // Sparx Systems. – 2023. – URL: <https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html> (дата обращения: 22.09.2024)
4. UML Sequence Diagrams: Tutorial with Examples [Электронный ресурс] // Creately. – 2024. – URL: <https://creately.com/blog/diagrams/sequence-diagram-tutorial/> (дата обращения: 19.10.2024)
5. BPMN 2.0 Specification [Электронный ресурс] // Object Management Group. – 2024. – URL: <https://www.omg.org/spec/BPMN/2.0/> (дата обращения: 27.12.2024)
6. MySQL 8.0 Reference Manual [Электронный ресурс] // Oracle. – 2024. – URL: <https://dev.mysql.com/doc/refman/8.0/en/> (дата обращения: 02.02.2025)
7. Entity-Relationship Diagram (ERD) Basics [Электронный ресурс] // Vertabelo. – 2023. – URL: <https://www.vertabelo.com/blog/erd-basics/> (дата обращения: 03.02.2025)
8. Spring Framework Documentation [Электронный ресурс] // Spring.io. – 2024. – URL: <https://docs.spring.io/spring-framework/docs/current/reference/html/> (дата обращения: 25.03.2025)
9. Spring Security Reference [Электронный ресурс] // Spring.io. – 2024. – URL: <https://docs.spring.io/spring-security/reference/index.html> (дата обращения: 25.03.2025)
10. Thymeleaf Tutorial [Электронный ресурс] // Thymeleaf.org. – 2024. – URL: <https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html> (дата обращения: 25.03.2025)
11. Java Persistence API (JPA) Guide [Электронный ресурс] // Oracle. – 2024. – URL: <https://docs.oracle.com/javaee/7/tutorial/persistence-intro.html> (дата обращения: 25.03.2025)
12. UML Component Diagram Tutorial [Электронный ресурс] // SmartDraw. – 2023. – URL: <https://www.smartdraw.com/component-diagram/> (дата обращения: 15.04.2025)
13. UML Deployment Diagram Tutorial [Электронный ресурс] // EdrawMax. – 2024. – URL: <https://www.edrawmax.com/deployment-diagram/> (дата обращения: 15.04.2025)
14. Apache JMeter User Manual [Электронный ресурс] // JMeter.apache.org. – 2024. – URL: <https://jmeter.apache.org/usermanual/index.html> (дата обращения: 20.04.2025)

ПРИЛОЖЕНИЕ А. ПУБЛИКАЦИИ

**ArtworkController.java:**

package com.example.vag.controller;

import com.example.vag.model.Artwork;

import com.example.vag.model.\*;

import com.example.vag.service.ArtworkService;

import com.example.vag.service.CategoryService;

import com.example.vag.service.UserService;

import com.example.vag.service.ExhibitionService;

import com.example.vag.util.FileUploadUtil;

import org.slf4j.LoggerFactory;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Controller;

import org.springframework.transaction.annotation.Transactional;

import org.springframework.ui.Model;

import org.springframework.util.StringUtils;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.multipart.MultipartFile;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import javax.validation.Valid;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.logging.Logger;

import java.util.stream.Collectors;

@Controller

@RequestMapping("/artwork")

public class ArtworkController {

private final ArtworkService artworkService;

private final CategoryService categoryService;

private final UserService userService;

private final ExhibitionService exhibitionService;

private final FileUploadUtil fileUploadUtil;

public ArtworkController(ArtworkService artworkService, CategoryService categoryService, UserService userService, ExhibitionService exhibitionService, FileUploadUtil fileUploadUtil) {

this.artworkService = artworkService;

this.categoryService = categoryService;

this.userService = userService;

this.exhibitionService = exhibitionService;

this.fileUploadUtil = fileUploadUtil;

}

@GetMapping("/list")

public String listArtworks(

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "12") int size,

Model model) {

page = Math.max(0, page);

Page<Artwork> artworkPage = artworkService.findPaginatedApprovedArtworks(PageRequest.of(page, size));

model.addAttribute("artworks", artworkPage);

return "artwork/list";

}

@GetMapping("/details/{id}")

public String viewArtwork(@PathVariable("id") Long id, Model model) {

Artwork artwork = artworkService.findByIdWithComments(id);

User currentUser = null;

try {

currentUser = userService.getCurrentUser();

} catch (Exception e) {

}

boolean isApproved = Artwork.ArtworkStatus.APPROVED.name().equals(artwork.getStatus());

boolean isAuthor = currentUser != null && currentUser.getId().equals(artwork.getUser().getId());

boolean isAdmin = currentUser != null && currentUser.hasRole("ADMIN");

if (!isApproved && !isAuthor && !isAdmin) {

return "redirect:/auth/access-denied";

}

model.addAttribute("artwork", artwork);

model.addAttribute("isLiked", currentUser != null && artworkService.isLikedByUser(artwork, currentUser));

model.addAttribute("isAuthenticated", currentUser != null);

model.addAttribute("isAdmin", isAdmin);

return "artwork/details";

}

@GetMapping("/create")

public String showCreateForm(@RequestParam(required = false) Long exhibitionId, Model model) {

model.addAttribute("artwork", new Artwork());

model.addAttribute("categories", categoryService.findAll());

model.addAttribute("selectedCategoryIds", new ArrayList<Long>());

model.addAttribute("exhibitionId", exhibitionId);

return "artwork/create";

}

@PostMapping("/create")

@Transactional

public String createArtwork(

@Valid @ModelAttribute("artwork") Artwork artwork,

BindingResult bindingResult,

@RequestParam("categoryIds") List<Long> categoryIds,

@RequestParam("imageFile") MultipartFile imageFile,

@RequestParam(required = false) Long exhibitionId,

Model model) throws IOException {

User currentUser = userService.getCurrentUser();

if (currentUser == null) {

return "redirect:/auth/login";

}

if (bindingResult.hasErrors()) {

model.addAttribute("categories", categoryService.findAll());

model.addAttribute("selectedCategoryIds", categoryIds);

model.addAttribute("exhibitionId", exhibitionId);

return "artwork/create";

}

Artwork savedArtwork = artworkService.create(artwork, imageFile, currentUser);

if (exhibitionId != null) {

Exhibition exhibition = exhibitionService.findById(exhibitionId).orElseThrow();

Long currentUserId = currentUser.getId();

Long exhibitionUserId = exhibition.getUser().getId();

if (!exhibition.isAuthorOnly() || currentUserId.equals(exhibitionUserId)) {

exhibition.getArtworks().add(savedArtwork);

savedArtwork.getExhibitions().add(exhibition);

exhibitionService.save(exhibition);

artworkService.save(savedArtwork);

return "redirect:/exhibition/details/" + exhibitionId;

}

}

return "redirect:/user/profile?created";

}

@GetMapping("/edit/{id}")

public String showEditForm(@PathVariable Long id, Model model) {

Artwork existingArtwork = artworkService.findByIdWithCategories(id).orElseThrow();

User currentUser = userService.getCurrentUser();

if (!existingArtwork.getUser().getId().equals(currentUser.getId()) &&

!currentUser.hasRole("ADMIN")) {

return "redirect:/auth/access-denied";

}

List<Long> selectedCategoryIds = existingArtwork.getCategories().stream()

.map(Category::getId)

.collect(Collectors.toList());

model.addAttribute("selectedCategoryIds", selectedCategoryIds);

model.addAttribute("artwork", existingArtwork);

model.addAttribute("categories", categoryService.findAll());

return "artwork/edit";

}

@PostMapping("/comment/{id}")

public String addComment(@PathVariable Long id,

@RequestParam String content,

RedirectAttributes redirectAttributes) {

User user = userService.getCurrentUser();

artworkService.addComment(id, user, content);

return "redirect:/artwork/details/" + id;

}

@PostMapping("/edit")

@Transactional

public String updateArtwork(

@Valid @ModelAttribute("artwork") Artwork artwork,

BindingResult bindingResult,

@RequestParam("categoryIds") List<Long> categoryIds,

@RequestParam(value = "imageFile", required = false) MultipartFile imageFile,

Model model) throws IOException {

if (bindingResult.hasErrors()) {

model.addAttribute("categories", categoryService.findAll());

model.addAttribute("selectedCategoryIds", categoryIds);

return "artwork/edit";

}

Artwork existingArtwork = artworkService.findByIdWithCategories(artwork.getId()).orElseThrow();

User currentUser = userService.getCurrentUser();

if (!existingArtwork.getUser().getId().equals(currentUser.getId()) &&

!currentUser.hasRole("ADMIN")) {

return "redirect:/auth/access-denied";

}

List<Category> categories = categoryService.findAllByIds(categoryIds);

existingArtwork.setCategories(new HashSet<>(categories));

existingArtwork.setTitle(artwork.getTitle());

existingArtwork.setDescription(artwork.getDescription());

existingArtwork.setStatus(Artwork.ArtworkStatus.PENDING.name());

if (imageFile != null && !imageFile.isEmpty()) {

String fileName = StringUtils.cleanPath(imageFile.getOriginalFilename());

String safeFileName = fileName

.replace(" ", "\_")

.replaceAll("[^a-zA-Z0-9.\_-]", "");

String relativePath = "artwork-images/" + currentUser.getId() + "/" + safeFileName;

existingArtwork.setImagePath(relativePath);

fileUploadUtil.saveFile(currentUser.getId(), safeFileName, imageFile);

}

artworkService.save(existingArtwork);

return "redirect:/user/profile?updated";

}

@PostMapping("/delete/{id}")

public String deleteArtwork(@PathVariable Long id) {

Artwork artwork = artworkService.findById(id).orElseThrow();

User currentUser = userService.getCurrentUser();

if (!artwork.getUser().getId().equals(currentUser.getId()) &&

!currentUser.hasRole("ADMIN")) {

return "redirect:/auth/access-denied";

}

artworkService.delete(artwork);

return "redirect:/user/profile?deleted";

}

@PostMapping("/like/{id}")

public String likeArtwork(@PathVariable Long id) {

User user = userService.getCurrentUser();

artworkService.likeArtwork(id, user);

return "redirect:/artwork/details/" + id;

}

@PostMapping("/unlike/{id}")

public String unlikeArtwork(@PathVariable Long id) {

User user = userService.getCurrentUser();

artworkService.unlikeArtwork(id, user);

return "redirect:/artwork/details/" + id;

}

@GetMapping("/artworks")

public String showArtworks(Model model, @RequestParam(defaultValue = "0") int page) {

int pageSize = 12;

page = Math.max(0, page);

Pageable pageable = PageRequest.of(page, pageSize, Sort.by("id").descending());

Page<Artwork> artworkPage = artworkService.getApprovedArtworks(pageable);

if (page > 0 && artworkPage.getContent().isEmpty()) {

return "redirect:/artwork/artworks?page=" + (page - 1);

}

model.addAttribute("artworks", artworkPage);

return "artwork/list";

}

}

**ArtworkRepository.java:**

package com.example.vag.repository;

import com.example.vag.model.Artwork;

import com.example.vag.model.User;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.jpa.repository.EntityGraph;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

import java.util.List;

import java.util.Optional;

@Repository

public interface ArtworkRepository extends JpaRepository<Artwork, Long> {

@EntityGraph(attributePaths = {"user", "categories"})

Page<Artwork> findAll(Pageable pageable);

@EntityGraph(attributePaths = {"user", "categories"})

@Query("SELECT a FROM Artwork a WHERE a.status = :status")

List<Artwork> findByStatus(@Param("status") String status);

@EntityGraph(attributePaths = {"user", "categories"})

List<Artwork> findByUser\_Id(Long userId);

@EntityGraph(attributePaths = {"user", "categories", "exhibitions"})

@Query("SELECT a FROM Artwork a WHERE a.user.id = :userId")

List<Artwork> findByUserWithDetails(@Param("userId") Long userId);

@Query(

value = "SELECT DISTINCT a FROM Artwork a " +

"JOIN FETCH a.user " +

"JOIN a.categories c " +

"WHERE c.id = :categoryId AND a.status = 'APPROVED'",

countQuery = "SELECT COUNT(DISTINCT a) FROM Artwork a " +

"JOIN a.categories c " +

"WHERE c.id = :categoryId AND a.status = 'APPROVED'"

)

Page<Artwork> findByCategoryId(@Param("categoryId") Long categoryId, Pageable pageable);

@EntityGraph(attributePaths = {"user", "categories"})

@Query("SELECT DISTINCT a FROM Artwork a JOIN a.exhibitions e WHERE e.id = :exhibitionId")

List<Artwork> findByExhibitionId(@Param("exhibitionId") Long exhibitionId);

@Query("SELECT DISTINCT a FROM Artwork a " +

"LEFT JOIN FETCH a.user " +

"LEFT JOIN FETCH a.comments c " +

"LEFT JOIN FETCH c.user " +

"WHERE a.id = :id")

Optional<Artwork> findByIdWithComments(@Param("id") Long id);

@Query("SELECT DISTINCT a FROM Artwork a " +

"LEFT JOIN FETCH a.categories " +

"LEFT JOIN FETCH a.user " +

"LEFT JOIN FETCH a.exhibitions " +

"WHERE a.id = :id")

Optional<Artwork> findByIdWithCategories(@Param("id") Long id);

@Query(

value = "SELECT DISTINCT a FROM Artwork a " +

"LEFT JOIN FETCH a.user " +

"LEFT JOIN FETCH a.categories " +

"WHERE a.status = 'APPROVED' " +

"ORDER BY a.id DESC",

countQuery = "SELECT COUNT(DISTINCT a) FROM Artwork a " +

"WHERE a.status = 'APPROVED'"

)

Page<Artwork> findApprovedArtworks(Pageable pageable);

@Query("SELECT CASE WHEN COUNT(l) > 0 THEN true ELSE false END FROM Like l WHERE l.artwork = :artwork AND l.user = :user")

boolean existsByArtworkAndUser(

@Param("artwork") Artwork artwork,

@Param("user") User user

);

@Query("SELECT COUNT(DISTINCT a) FROM Artwork a " +

"JOIN a.categories c " +

"WHERE c.id = :categoryId AND a.status = 'APPROVED'")

long countApprovedArtworksByCategoryId(@Param("categoryId") Long categoryId);

@EntityGraph(attributePaths = {"user", "categories"})

@Query("SELECT a FROM Artwork a WHERE a.status = :status")

Page<Artwork> findByStatus(@Param("status") String status, Pageable pageable);

Page<Artwork> findByUser(User user, Pageable pageable);

@Query("SELECT a FROM Artwork a WHERE a.user = :user AND a.status = :status")

Page<Artwork> findByUserAndStatus(

@Param("user") User user,

@Param("status") String status,

Pageable pageable);

@Query("SELECT a FROM Artwork a JOIN a.artworkLikes l WHERE l.user = :user")

Page<Artwork> findLikedByUser(@Param("user") User user, Pageable pageable);

@Query("SELECT a FROM Artwork a JOIN a.exhibitions e WHERE e.id = :exhibitionId")

Page<Artwork> findByExhibitionId(@Param("exhibitionId") Long exhibitionId, Pageable pageable);

@Query("SELECT DISTINCT a FROM Artwork a LEFT JOIN FETCH a.user WHERE a.id IN :ids")

List<Artwork> findAllByIdWithUser(@Param("ids") List<Long> ids);

@EntityGraph(attributePaths = {"user", "categories"})

@Query(value = "SELECT DISTINCT a FROM Artwork a JOIN FETCH a.user ORDER BY a.id DESC",

countQuery = "SELECT COUNT(a) FROM Artwork a")

Page<Artwork> findAllPaginated(Pageable pageable);

@Query(value = "SELECT a FROM Artwork a WHERE a.status = :status",

countQuery = "SELECT COUNT(a) FROM Artwork a WHERE a.status = :status")

Page<Artwork> findByStatusPaginated(@Param("status") String status, Pageable pageable);

}

ПРИЛОЖЕНИЕ Б. ВЫСТАВКИ

**ExhibitionController.java:**

package com.example.vag.controller;

import com.example.vag.model.Artwork;

import com.example.vag.model.Exhibition;

import com.example.vag.model.User;

import com.example.vag.service.ArtworkService;

import com.example.vag.service.ExhibitionService;

import com.example.vag.service.UserService;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.\*;

import org.springframework.transaction.annotation.Transactional;

import javax.validation.Valid;

import java.util.List;

import java.util.Set;

@Controller

@RequestMapping("/exhibition")

public class ExhibitionController {

private final ExhibitionService exhibitionService;

private final ArtworkService artworkService;

private final UserService userService;

public ExhibitionController(ExhibitionService exhibitionService,

ArtworkService artworkService,

UserService userService) {

this.exhibitionService = exhibitionService;

this.artworkService = artworkService;

this.userService = userService;

}

@GetMapping("/list")

public String listExhibitions(@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "12") int size,

Model model) {

page = Math.max(0, page);

Page<Exhibition> exhibitionPage = exhibitionService.findPaginatedExhibitions(page, size);

exhibitionPage.getContent().forEach(exhibition -> {

model.addAttribute("approvedArtworksCount\_" + exhibition.getId(),

exhibitionService.countApprovedArtworksInExhibition(exhibition.getId()));

model.addAttribute("firstApprovedArtwork\_" + exhibition.getId(),

exhibitionService.getFirstApprovedArtworkInExhibition(exhibition.getId()));

});

model.addAttribute("exhibitions", exhibitionPage);

model.addAttribute("currentPage", page);

model.addAttribute("totalPages", exhibitionPage.getTotalPages());

return "exhibition/list";

}

@GetMapping("/details/{id}")

public String viewExhibition(

@PathVariable Long id,

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "12") int size,

Model model) {

Exhibition exhibition = exhibitionService.findById(id).orElseThrow();

Pageable pageable = PageRequest.of(page, size);

Page<Artwork> artworkPage = artworkService.findByExhibitionId(id, pageable);

model.addAttribute("exhibition", exhibition);

model.addAttribute("artworks", artworkPage);

model.addAttribute("approvedArtworksCount", exhibitionService.countApprovedArtworksInExhibition(id));

model.addAttribute("firstApprovedArtwork", exhibitionService.getFirstApprovedArtworkInExhibition(id));

return "exhibition/details";

}

@GetMapping("/create")

public String showCreateExhibition(Model model) {

model.addAttribute("exhibition", new Exhibition());

return "exhibition/create";

}

@PostMapping("/create")

public String createExhibition(@Valid @ModelAttribute("exhibition") Exhibition exhibition,

BindingResult bindingResult) {

if (bindingResult.hasErrors()) {

return "exhibition/create";

}

User user = userService.getCurrentUser();

exhibitionService.create(exhibition, user);

return "redirect:/user/profile?created";

}

@GetMapping("/edit/{id}")

public String showEditForm(@PathVariable Long id, Model model) {

Exhibition existingExhibition = exhibitionService.findById(id).orElseThrow();

User currentUser = userService.getCurrentUser();

if (!existingExhibition.getUser().getId().equals(currentUser.getId()) &&

!currentUser.hasRole("ADMIN")) {

return "redirect:/auth/access-denied";

}

model.addAttribute("exhibition", existingExhibition);

return "exhibition/edit";

}

@PostMapping("/edit")

public String updateExhibition(@Valid @ModelAttribute("exhibition") Exhibition exhibition,

BindingResult bindingResult) {

if (bindingResult.hasErrors()) {

return "exhibition/edit";

}

exhibitionService.update(exhibition);

return "redirect:/user/profile?updated";

}

@PostMapping("/delete/{id}")

public String deleteExhibition(@PathVariable Long id) {

Exhibition exhibition = exhibitionService.findById(id).orElseThrow();

User currentUser = userService.getCurrentUser();

if (!exhibition.getUser().getId().equals(currentUser.getId()) &&

!currentUser.hasRole("ADMIN")) {

return "redirect:/auth/access-denied";

}

exhibitionService.delete(exhibition);

return "redirect:/user/profile?deleted";

}

@PostMapping("/add-artwork/{exhibitionId}/{artworkId}")

public String addArtworkToExhibition(@PathVariable Long exhibitionId,

@PathVariable Long artworkId) {

Exhibition exhibition = exhibitionService.findById(exhibitionId).orElseThrow();

Artwork artwork = artworkService.findById(artworkId).orElseThrow();

User currentUser = userService.getCurrentUser();

if (exhibition.isAuthorOnly() && !exhibition.getUser().getId().equals(currentUser.getId())) {

return "redirect:/auth/access-denied";

}

exhibition.getArtworks().add(artwork);

artwork.getExhibitions().add(exhibition);

exhibitionService.save(exhibition);

artworkService.save(artwork);

return "redirect:/exhibition/details/" + exhibitionId;

}

@PostMapping("/remove-artwork/{exhibitionId}/{artworkId}")

public String removeArtworkFromExhibition(@PathVariable Long exhibitionId,

@PathVariable Long artworkId) {

Exhibition exhibition = exhibitionService.findById(exhibitionId).orElseThrow();

Artwork artwork = artworkService.findById(artworkId).orElseThrow();

User currentUser = userService.getCurrentUser();

if (!exhibition.getUser().getId().equals(currentUser.getId()) &&

!artwork.getUser().getId().equals(currentUser.getId()) &&

!currentUser.hasRole("ADMIN")) {

return "redirect:/auth/access-denied";

}

exhibitionService.removeArtworkFromExhibition(exhibitionId, artworkId);

return "redirect:/exhibition/details/" + exhibitionId;

}

@GetMapping("/exhibitions")

public String showExhibitions(Model model, @RequestParam(defaultValue = "0") int page) {

page = Math.max(0, page);

int pageSize = 6;

Pageable pageable = PageRequest.of(page, pageSize, Sort.by("id").descending());

Page<Exhibition> exhibitionPage = exhibitionService.getPublicExhibitions(pageable);

if (page > 0 && exhibitionPage.getContent().isEmpty()) {

return "redirect:/exhibition/exhibitions?page=" + (page - 1);

}

exhibitionPage.getContent().forEach(exhibition -> {

model.addAttribute("approvedArtworksCount\_" + exhibition.getId(),

exhibitionService.countApprovedArtworksInExhibition(exhibition.getId()));

model.addAttribute("firstApprovedArtwork\_" + exhibition.getId(),

exhibitionService.getFirstApprovedArtworkInExhibition(exhibition.getId()));

});

model.addAttribute("exhibitions", exhibitionPage);

return "exhibition/list";

}

@GetMapping("/add-exist-artwork/{exhibitionId}")

@Transactional(readOnly = true)

public String addExistArtwork(@PathVariable Long exhibitionId, Model model) {

Exhibition exhibition = exhibitionService.findById(exhibitionId).orElseThrow();

User currentUser = userService.getCurrentUser();

List<Artwork> userArtworks = artworkService.findByUserWithDetails(currentUser);

Set<Artwork> exhibitionArtworks = exhibition.getArtworks();

model.addAttribute("exhibition", exhibition);

model.addAttribute("artworks", userArtworks);

model.addAttribute("exhibitionArtworks", exhibitionArtworks);

return "exhibition/add-exist-artwork";

}

@PostMapping("/add-exist-artwork/{exhibitionId}/{artworkId}")

@Transactional

public String addExistArtwork(@PathVariable Long exhibitionId, @PathVariable Long artworkId) {

Exhibition exhibition = exhibitionService.findById(exhibitionId).orElseThrow();

Artwork artwork = artworkService.findByIdWithCategories(artworkId).orElseThrow();

User currentUser = userService.getCurrentUser();

if (exhibition.isAuthorOnly() && !exhibition.getUser().getId().equals(currentUser.getId())) {

return "redirect:/auth/access-denied";

}

if (!artwork.getUser().getId().equals(currentUser.getId())) {

return "redirect:/auth/access-denied";

}

exhibition.getArtworks().add(artwork);

artwork.getExhibitions().add(exhibition);

exhibitionService.save(exhibition);

artworkService.save(artwork);

return "redirect:/exhibition/details/" + exhibitionId;

}

@PostMapping("/remove-exist-artwork/{exhibitionId}/{artworkId}")

public String removeExistArtwork(@PathVariable Long exhibitionId, @PathVariable Long artworkId) {

Exhibition exhibition = exhibitionService.findById(exhibitionId).orElseThrow();

Artwork artwork = artworkService.findById(artworkId).orElseThrow();

User currentUser = userService.getCurrentUser();

if (!exhibition.getUser().getId().equals(currentUser.getId()) &&

!artwork.getUser().getId().equals(currentUser.getId()) &&

!currentUser.hasRole("ADMIN")) {

return "redirect:/auth/access-denied";

}

exhibitionService.removeArtworkFromExhibition(exhibitionId, artworkId);

return "redirect:/exhibition/add-exist-artwork/" + exhibitionId;

}

}

**ExhibitionRepository.java:**

package com.example.vag.repository;

import com.example.vag.model.Exhibition;

import com.example.vag.model.Artwork;

import com.example.vag.model.User;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.jpa.repository.EntityGraph;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

import java.util.List;

import java.util.Optional;

@Repository

public interface ExhibitionRepository extends JpaRepository<Exhibition, Long> {

@EntityGraph(attributePaths = {"artworks", "user"})

List<Exhibition> findByUserId(Long userId);

@EntityGraph(attributePaths = {"artworks", "user"})

List<Exhibition> findByAuthorOnlyFalse();

@Query(

value = "SELECT DISTINCT e FROM Exhibition e " +

"LEFT JOIN FETCH e.user " +

"LEFT JOIN FETCH e.artworks " +

"WHERE e.authorOnly = false " +

"ORDER BY e.id DESC",

countQuery = "SELECT COUNT(DISTINCT e) FROM Exhibition e " +

"WHERE e.authorOnly = false"

)

Page<Exhibition> findPublicExhibitions(Pageable pageable);

@Query(

value = "SELECT DISTINCT e FROM Exhibition e " +

"LEFT JOIN FETCH e.user " +

"LEFT JOIN FETCH e.artworks " +

"ORDER BY e.id DESC",

countQuery = "SELECT COUNT(DISTINCT e) FROM Exhibition e"

)

Page<Exhibition> findAllExhibitions(Pageable pageable);

@Query("SELECT COUNT(e) FROM Exhibition e")

long countPublicExhibitions();

@Query("SELECT COUNT(a) FROM Exhibition e JOIN e.artworks a WHERE e.id = :exhibitionId AND a.status = 'APPROVED'")

long countApprovedArtworksInExhibition(@Param("exhibitionId") Long exhibitionId);

@Query("SELECT a FROM Exhibition e JOIN e.artworks a WHERE e.id = :exhibitionId AND a.status = 'APPROVED' ORDER BY a.id ASC")

List<Artwork> findFirstApprovedArtworkInExhibition(@Param("exhibitionId") Long exhibitionId);

@EntityGraph(attributePaths = {"artworks", "user", "artworks.user"})

@Override

Optional<Exhibition> findById(Long id);

Page<Exhibition> findByUser(User user, Pageable pageable);

}

ПРИЛОЖЕНИЕ В. ПОЛЬЗОВАТЕЛИ

**UserController.java:**

package com.example.vag.controller;

import com.example.vag.model.Artwork;

import com.example.vag.model.Exhibition;

import com.example.vag.model.User;

import com.example.vag.service.ArtworkService;

import com.example.vag.service.UserService;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.stereotype.Controller;

import org.springframework.transaction.annotation.Transactional;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.\*;

import com.example.vag.service.ExhibitionService;

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

@Controller

@RequestMapping("/user")

public class UserController {

private final UserService userService;

private final ArtworkService artworkService;

private final ExhibitionService exhibitionService;

public UserController(UserService userService, ArtworkService artworkService, ExhibitionService exhibitionService) {

this.userService = userService;

this.artworkService = artworkService;

this.exhibitionService = exhibitionService;

}

@GetMapping("/profile")

@Transactional(readOnly = true)

public String showProfile(Model model) {

User user = userService.getCurrentUser();

Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

boolean isOwnProfile = authentication != null && authentication.getName().equals(user.getUsername());

List<Artwork> artworks;

if (isOwnProfile) {

artworks = artworkService.findByUserWithDetails(user);

} else {

artworks = artworkService.findByUserWithDetails(user).stream()

.filter(artwork -> "APPROVED".equals(artwork.getStatus()))

.collect(Collectors.toList());

}

List<Exhibition> exhibitions = exhibitionService.findByUser(user);

exhibitions.forEach(exhibition -> {

exhibition.getUser().getUsername();

if (!exhibition.getArtworks().isEmpty()) {

exhibition.getArtworks().forEach(artwork -> {

artwork.getStatus();

});

}

});

model.addAttribute("user", user);

model.addAttribute("artworks", artworks);

model.addAttribute("exhibitions", exhibitions);

model.addAttribute("isOwnProfile", isOwnProfile);

return "user/profile";

}

@GetMapping("/profile/{id}")

@Transactional(readOnly = true)

public String showUserProfile(@PathVariable Long id, Model model) {

User user = userService.findById(id).orElseThrow();

Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

boolean isOwnProfile = authentication != null && authentication.getName().equals(user.getUsername());

List<Artwork> artworks;

if (isOwnProfile) {

artworks = artworkService.findByUserWithDetails(user);

} else {

artworks = artworkService.findByUserWithDetails(user).stream()

.filter(artwork -> "APPROVED".equals(artwork.getStatus()))

.collect(Collectors.toList());

}

List<Exhibition> exhibitions = exhibitionService.findByUser(user);

exhibitions.forEach(exhibition -> {

exhibition.getUser().getUsername();

if (!exhibition.getArtworks().isEmpty()) {

exhibition.getArtworks().forEach(artwork -> {

artwork.getStatus();

});

}

});

model.addAttribute("user", user);

model.addAttribute("artworks", artworks);

model.addAttribute("exhibitions", exhibitions);

model.addAttribute("isOwnProfile", isOwnProfile);

model.addAttribute("isAuthenticated", authentication != null);

return "user/profile";

}

@GetMapping("/settings")

public String showSettings(Model model) {

User user = userService.getCurrentUser();

model.addAttribute("user", user);

return "user/settings";

}

@PostMapping("/settings")

public String updateSettings(@Valid @ModelAttribute("user") User user,

BindingResult bindingResult, Model model) {

if (bindingResult.hasErrors()) {

return "user/settings";

}

User currentUser = userService.getCurrentUser();

if (!user.getUsername().equals(currentUser.getUsername())) {

if (userService.findByUsername(user.getUsername()).isPresent()) {

bindingResult.rejectValue("username", "error.user", "Пользователь с таким именем уже существует");

return "user/settings";

}

}

if (!user.getEmail().equals(currentUser.getEmail())) {

if (userService.findByEmail(user.getEmail()).isPresent()) {

bindingResult.rejectValue("email", "error.user", "Пользователь с таким email уже существует");

return "user/settings";

}

}

if (user.getPassword() != null && !user.getPassword().isEmpty()) {

if (!user.getPassword().equals(user.getConfirmPassword())) {

bindingResult.rejectValue("confirmPassword", "error.user", "Пароли не совпадают");

return "user/settings";

}

}

userService.update(user);

if (!user.getUsername().equals(currentUser.getUsername())) {

Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

SecurityContextHolder.getContext().setAuthentication(null);

}

return "redirect:/user/profile?updated";

}

@GetMapping("/liked")

public String likedArtworks(

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "12") int size,

Model model) {

User user = userService.getCurrentUser();

Pageable pageable = PageRequest.of(page, size);

Page<Artwork> artworkPage = artworkService.findLikedArtworks(user, pageable);

model.addAttribute("artworks", artworkPage);

return "user/liked";

}

@GetMapping("/list")

@Transactional(readOnly = true)

public String listUsers(

Model model,

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "12") int size) {

Pageable pageable = PageRequest.of(page, size, Sort.by("username"));

Page<User> userPage = userService.findAll(pageable);

userPage.getContent().forEach(user -> {

user.getArtworks().size();

user.getExhibitions().size();

});

model.addAttribute("users", userPage);

return "user/list";

}

}

**UserRepository.java:**

package com.example.vag.repository;

import com.example.vag.model.User;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByUsername(String username);

Optional<User> findByEmail(String email);

Page<User> findAll(Pageable pageable);

}