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:

«Распределенные программные системы»

на тему:

«Программная система «Виртуальная арт-галерея»»

Модуль «Администрирование»

Выполнил: студент гр. ИСТ-122

Жминьковская М.А.

Принял: доц. каф. ИСПИ

Проскурина Г.В.

Владимир, 2025

**АННОТАЦИЯ**

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

Пояснительная записка содержит 45 листов, в том числе 28 рисунков, 4 использованных источника, 3 приложения.

**ANNOTATION**

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

The explanatory note contains 45 sheets, including 28 figures, 4 sources used, 3 appendices.

**СОДЕЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc198501094)

[1 ОСПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc198501095)

[1.1 Цель и задачи 5](#_Toc198501096)

[1.2 Описание предметной области 5](#_Toc198501097)

[1.3 Словарь предметной области 6](#_Toc198501098)

[2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ ДЛЯ ИС 7](#_Toc198501099)

[2.1 Диаграмма прецедентов 7](#_Toc198501100)

[2.2 Диаграмма классов 11](#_Toc198501101)

[2.3 Диаграмма состояний 11](#_Toc198501102)

[2.4 Диаграмма последовательностей 12](#_Toc198501103)

[2.5 Схема бизнес-процесса (BPMN) 13](#_Toc198501104)

[3 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ ДЛЯ ИС 15](#_Toc198501105)

[3.1 Разработка базы данных 15](#_Toc198501106)

[3.2 Разработка приложения для ИС 15](#_Toc198501107)

[3.3 Реализация приложения для ИС 17](#_Toc198501108)

[3.4 Шаблоны страниц 18](#_Toc198501109)

[3.5 Страницы готового приложения 21](#_Toc198501110)

[4 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ 26](#_Toc198501111)

[ЗАКЛЮЧЕНИЕ 28](#_Toc198501112)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29](#_Toc198501113)

[ПРИЛОЖЕНИЕ А 30](#_Toc198501114)

[ПРИЛОЖЕНИЕ Б 40](#_Toc198501115)

[ПРИЛОЖЕНИЕ В 44](#_Toc198501116)

# 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, категориями. Приложение будет ориентировано на создание уникального пользовательского опыта, что сделает виртуальную арт-галерею важным инструментом для популяризации искусства в цифровую эпоху.

# 1 ОСПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

# 1.1 Цель и задачи

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

**Задачи для достижения цели:**

1)Анализ предметной области;

2)Проектирование системы.

3)Проектирование базы данных.

4)Реализация системы

5)Тестирование системы

6)Составление пояснительной записки.

**Функциональные требования:**

— Авторизация и регистрация (художник, посетитель);

–– Добавление публикаций (художник);

–– Выполнений действий, связанных с публикациями (художник, посетитель)

–– Система статусов публикаций (администратор, художник);

— Система оценки публикаций (художник);

— Управление категориями публикаций (администратор);

— Удаление публикаций (художник);

— Выбор категорий для публикации (художник).

# 1.2 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.

# 1.3 Словарь предметной области

* Посетитель – незарегистрированный пользователь, который обладает минимальными правами; характеристики: нет; действия: просматривать категории, просматривать публикацию, просматривать профили художников.
* Художник – пользователь, который зарегистрировался и прошёл авторизацию; характеристики: имя, пароль, электронная почта; действия: регистрация, вход, выход, создание/редактирование/удаление публикации, оценка публикации, комментирование публикации.
* Администратор – зарегистрированный пользователь с расширенными правами доступа; характеристика: имя, пароль, электронная почта; действия: управление категорией, подтверждение/отклонение публикации.
* Публикация – работа художника в профиле; характеристика: название, описание, фото, рейтинг, автор; действия: просмотр/удаление публикации, оценивание публикации.
* Категория публикации – тип изображения в зависимости от содержания, тематики и стиля; характеристика: название, описание; действия: просмотр/редактирование/удаление категории.

# 2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ ДЛЯ ИС

# 2.1 Диаграмма прецедентов

Диаграмма прецедентов [1] представлена на рисунке 1.

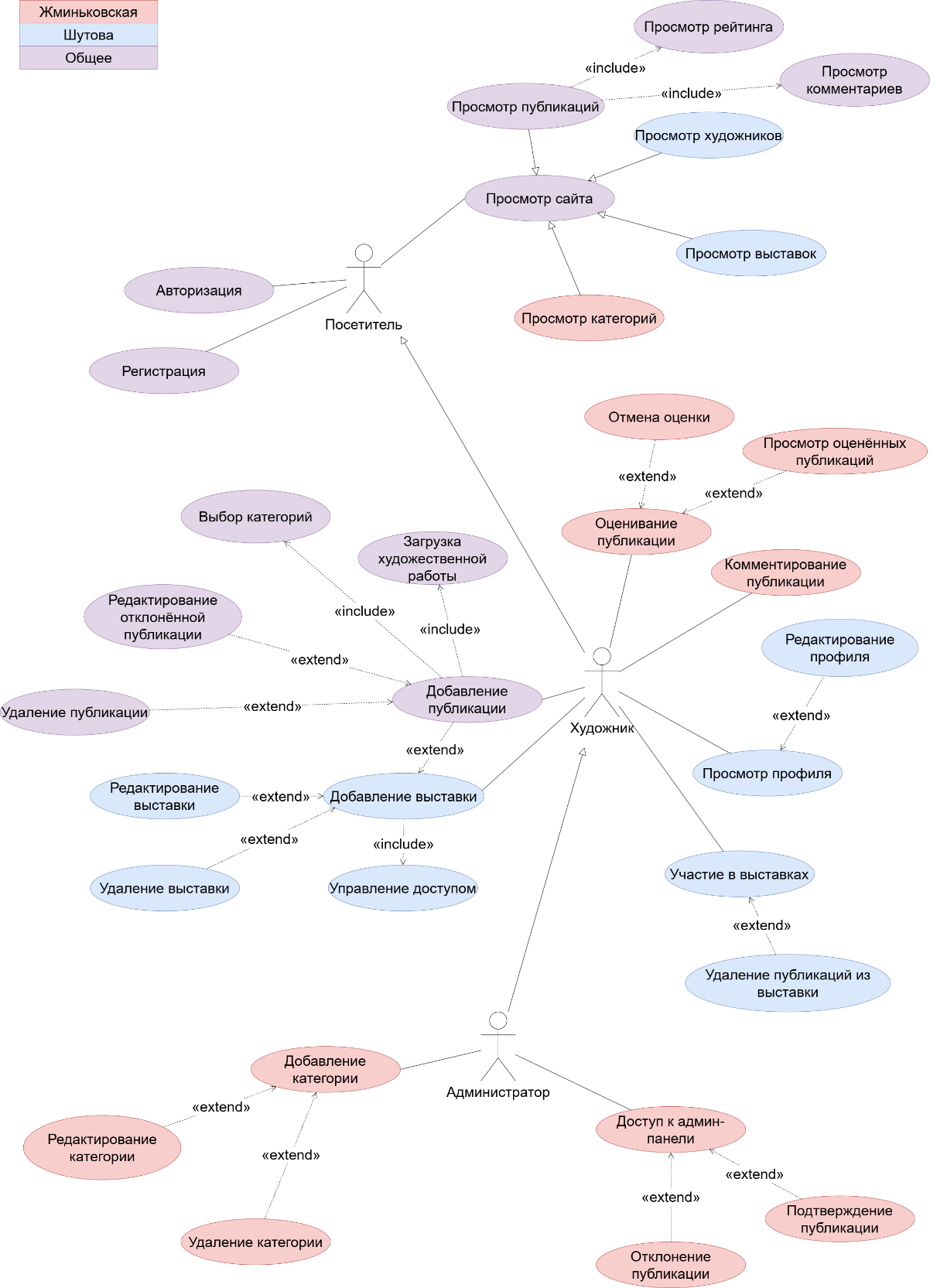


Рисунок 1. Диаграмма прецедентов

**Описание прецедентов:**

**1) Название:** Регистрация

**Предусловие**: Нет

**Действующее лицо:** Посетитель

**Основной поток:** Регистрация

Посетитель нажимает на иконку аватара в правом верхнем углу или на кнопки «Добавить публикацию», «Создать выставку», «Оценить», после чего его перекидывает на страницу со входом и регистрацией. Посетитель выбирает регистрацию и вводит основные данные о себе. После нажимает на кнопку «Создать профиль».

**Альтернативный поток:** Нет

**Постусловие:** Посетитель авторизован с ролью художник, его профиль автоматически подтверждён.

**2) Название:** Авторизация

**Предусловие**: Профиль зарегистрирован в базе данных

**Действующее лицо:** Посетитель

**Основной поток:** Авторизация

Посетитель нажимает на иконку аватара в правом верхнем углу или на кнопки «Добавить публикацию», «Создать выставку», «Оценить», после чего его перекидывает на страницу со входом и регистрацией. Посетитель выбирает авторизацию и вводит свои почту и пароль. После правильного ввода данных пользователь авторизуется в системе.

**Альтернативный поток:** Неправильный ввод

Если система не нашла в базе данных пользователя с указанной почтой, посетитель увидит уведомление о том, что такого профиля нет. Если был неправильно введён пароль, посетитель увидит уведомление о том, что пароль не соответствует паролю, сохранённому в базе данных.

**Альтернативный поток:** Сброс пароля

В случае если посетитель забыл пароль от своего аккаунта, он может выбрать ссылку «Сбросить пароль». Его перебрасывает на страницу подтверждения пользователя через отправку на нужный почтовый адрес кода подтверждения (для прототипа системы настоящей функции отправки кода подтверждения не будет). После проверки адреса почты пользователь устанавливает новый пароль, сохраняет его, после чего может вернуться к авторизации.

**Постусловие:** Посетитель авторизован с ролью художник или администратор.

**3)** **Название:** Просмотр публикации

**Предусловие**: Нет

**Действующее лицо:** Посетитель, художник, администратор

**Основной поток:** Просмотр публикации

Пользователь нажимает на выбранную публикацию в разделе «Публикации». Его перебрасывает в окно просмотра публикации, где он видит саму публикацию, название, категории, рейтинг, автора и описание.

**Альтернативный поток:** Нет

**Постусловие:** Пользователь просматривает публикацию.

**4)** **Название:** Просмотр художников

**Предусловие**: Нет

**Действующее лицо:** Посетитель, художник, администратор

**Основной поток:** Просмотр выставки

Пользователь нажимает на выбранную аватарку художника в разделе «Художники». Его перебрасывает в профиль художника, где он видит никнейм художника, описание профиля, его аватарку, выбор между просмотром публикаций художника (по умолчанию) и его выставками или выставками, где он участвовал. При переходе на просмотр выставок отображаются выставки художника, в которых он участвовал или которые создал.

**Альтернативный поток:** Нет

**Постусловие:** Пользователь просматривает профиль художника

**5) Название:** Оценка публикации

**Предусловие**: Нет

**Действующее лицо:** Художник, администратор

**Основной поток:** Оценка публикации

Пользователь нажимает на понравившуюся публикацию, которую хочет оценить, в разделе «Публикации». Его перебрасывает в окно просмотра публикации. Пользователь ставит оценку, нажимая на «нравится» под публикацией. Система сохраняет оценку и обновляет рейтинг публикации.

**Альтернативный поток:** нет

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

**6) Название:** Добавить публикацию

**Предусловие**: Нет

**Действующее лицо:** Художник, администратор

**Основной поток:** Добавление публикации

Пользователь нажимает на кнопку «Публикация» и его перебрасывает в окно создания публикации. Пользователь загружает изображение, добавляет описание и категорию работы. После пользователь нажимает на кнопку «Сохранить» и публикации присваивается статус «Ожидает проверки». Система сохраняет публикацию в базе данных. Пользователь может удалить публикацию пока она находится в состоянии «Ожидает проверки».

**Альтернативный поток:** Публикация не добавлена

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

**Постусловие:** Публикация находится в состоянии «Ожидает проверки» и ждет подтверждения от администратора.

**7) Название:** Создание категории

**Предусловие**: Нет

**Действующее лицо:** Администратор

**Основной поток:** Создание категории

Администратор нажимает на кнопку «Управление категориями». После этого его перебрасывает на страницу, где отображается информация о категориях. Он нажимает кнопку «Создать новую категорию». Его перебрасывает в окно создания, где нужно вписать название и описание категории, после необходимо нажать кнопку «Сохранить», и категория будет создана.

**Альтернативный поток:** Нет

**Постусловие:** Категория создана.

**8) Название:** Редактирование категории

**Предусловие**: Нет

**Действующее лицо:** Администратор

**Основной поток:** Редактирование категории

Администратор нажимает на кнопку «Управление категориями». После этого его перебрасывает на страницу, где отображается информация о категориях. Он нажимает кнопку «Редактировать». Его перебрасывает в окно редактирования категории, где можно изменить название и описание категории, после необходимо нажать кнопку «Сохранить изменения», и категория будет изменена.

**Альтернативный поток:** Не заполнены обязательные поля

Если поле с названием пусто или категория с таким названием уже существуют, изменения в категорию не вносятся, администратор получает уведомление об ошибке.

**Постусловие:** Категория отредактирована.

**9) Название:** Удаление категории

**Предусловие**: Нет

**Действующее лицо:** Администратор

**Основной поток:** Удаление категории

Администратор нажимает на кнопку «Управление категориями». После этого его перебрасывает на страницу, где отображается информация о категориях. Он нажимает кнопку «Удалить категорию». На экране появляется запрос «Вы уверены, что хотите удалить категорию?» Администратор подтверждает свое действие, и категория удаляется.

**Альтернативный поток:** Нет

**Постусловие:** Категория удалена.

**10) Название:** Подтверждение публикации

**Предусловие**: Нет

**Действующее лицо:** Администратор

**Основной поток:** Подтверждение публикации

Администратор нажимает на кнопку «Админ-панель». После этого его перебрасывает на страницу с публикациями, где отображается информация о статусе. Администратор нажимает на публикацию, которую хочет подтвердить, проверяет публикацию и нажимает на кнопку «Одобрить».

**Альтернативный поток:** Публикация не подтверждена

Если публикация не удовлетворяет правилам, администратор выбирает вариант «Отклонить»

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

# 2.2 Диаграмма классов

Диаграмма классов [1] представлена на рисунке 2.

На диаграмме классов представлена взаимосвязь моделей.



Рисунок 2. Диаграмма классов

# 2.3 Диаграмма состояний

Описание диаграммы состояний [1] для объекта «Подтверждение публикации»:

1. **Создание публикации:**

Художник создает новую публикацию, добавляя название, описание, выбирая категорию и загружая свою работу. Объект «Публикация» создается в системе. Публикация получает статус «Ожидает проверки». Публикация может быть удалена.

1. **Модерация отклонена:**

Администратор проверил публикацию и отклонил её по причине нарушения правил или недостаточного качества. Публикация получает состояние «Отклонена», чтобы художник мог посмотреть публикацию и отредактировать или удалить.

1. **Модерация одобрена:**

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

**4.** **Удаление публикации**:

Художник удаляет публикацию.

Диаграмма состояний для объекта «Подтверждение публикации» представлена на рисунке 3.

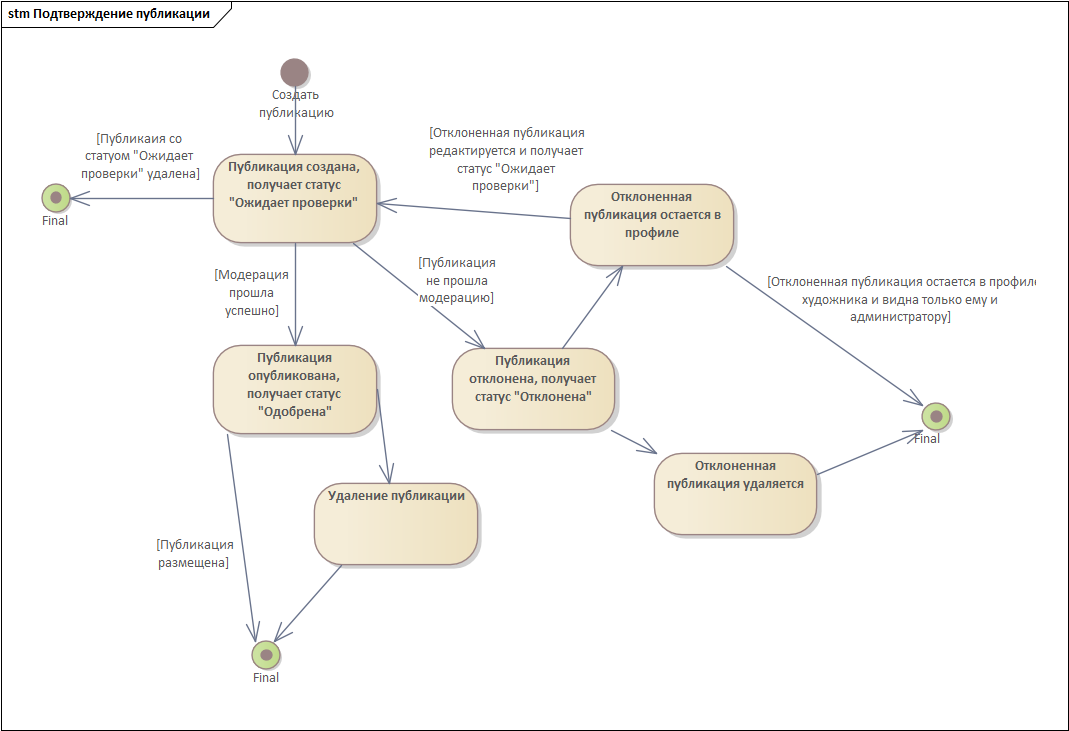


Рисунок 3. Диаграмма состояний для объекта «Подтверждение публикации»

# 2.4 Диаграмма последовательностей

Описание диаграммы последовательностей [1] для объекта **«**Добавление категории**»:**

Admin отправляет запрос на создание новой категории.

AdminController принимает запрос и вызывает метод CreateCategory(category).

CategoryService выполняет:

ValidateNameUniqueness — проверяет уникальность имени категории.

save(category) — сохраняет категорию в системе (например, в базе данных).

После успешного сохранения возвращается объект savedCategory.

Процесс завершается статусом Created category (категория создана).

Диаграмма последовательностей для объекта «Добавление категории» представлена на рисунке 4.



Рисунок 4. Диаграмма последовательностей для объекта «Добавление категории»

# 2.5 Схема бизнес-процесса (BPMN)

Схема бизнес-процесса [2] представлена на рисунке 5.

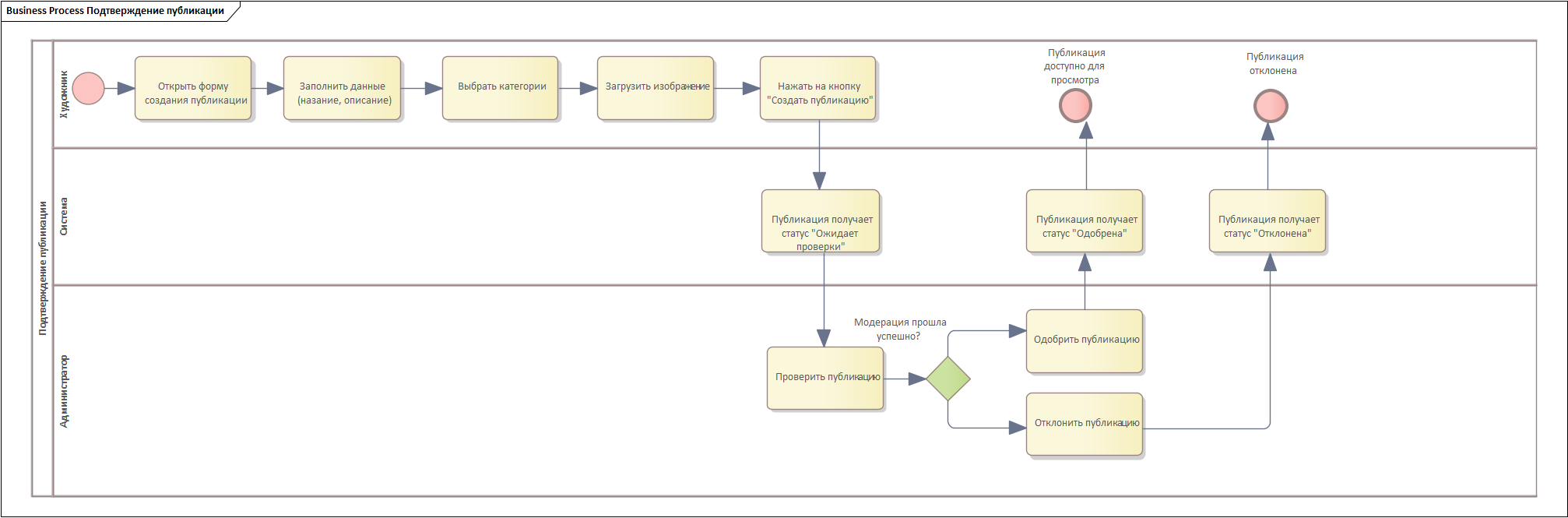


Рисунок 5. Схема бизнес-процесса

# 3 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ ДЛЯ ИС

# 3.1 Разработка базы данных

На рисунке 6 предствлена ER-диаграмма, состоящая из 9 таблиц.

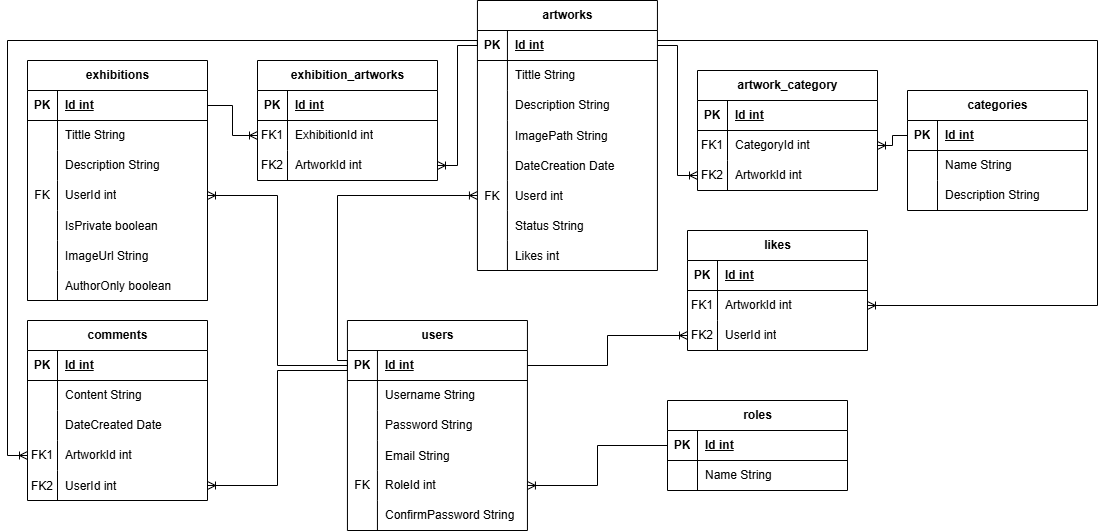


Рисунок 6. ER-диаграмма

# 3.2 Разработка приложения для ИС

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

**Были разработаны 6 моделей:**

Artwork – публикации с атрибутами;

Category – категории публикаций;

Comment –комментраии к публикациям;

Like – лайки пользователей к публикациям;

Role – роли пользователей;

User – пользователи системы.

**Были разработаны 7 контроллеров:**

AdminController – администрирование пользователей, публикаций и категорий;

ArtworkController – управление публикациями;

AuthController – аутентификация и регистрация пользователей;

CategoryController – управление категориями;

HomeController – главная страница;

UserController – профили пользователей.

**Были разработаны 6 репозиториев:**

ArtworkRepository – работа с публикациями;

CategoryRepository – работа с категориями;

CommentRepository – работа с комментариями;

LikeRepository – работа с лайками;

RoleRepository – работа с ролями;

UserRepository – работа с пользователями.

**Были разработаны 3 сервиса:**

ArtworkService – бизнес-логика для публикаций;

CategoryService – бизнес-логика для категорий;

UserService – бизнес-логика для управления пользователями.

Диаграмма компенентов [1] представлена на рисунке 7, на которой показана связь всех комнентов ситемы.

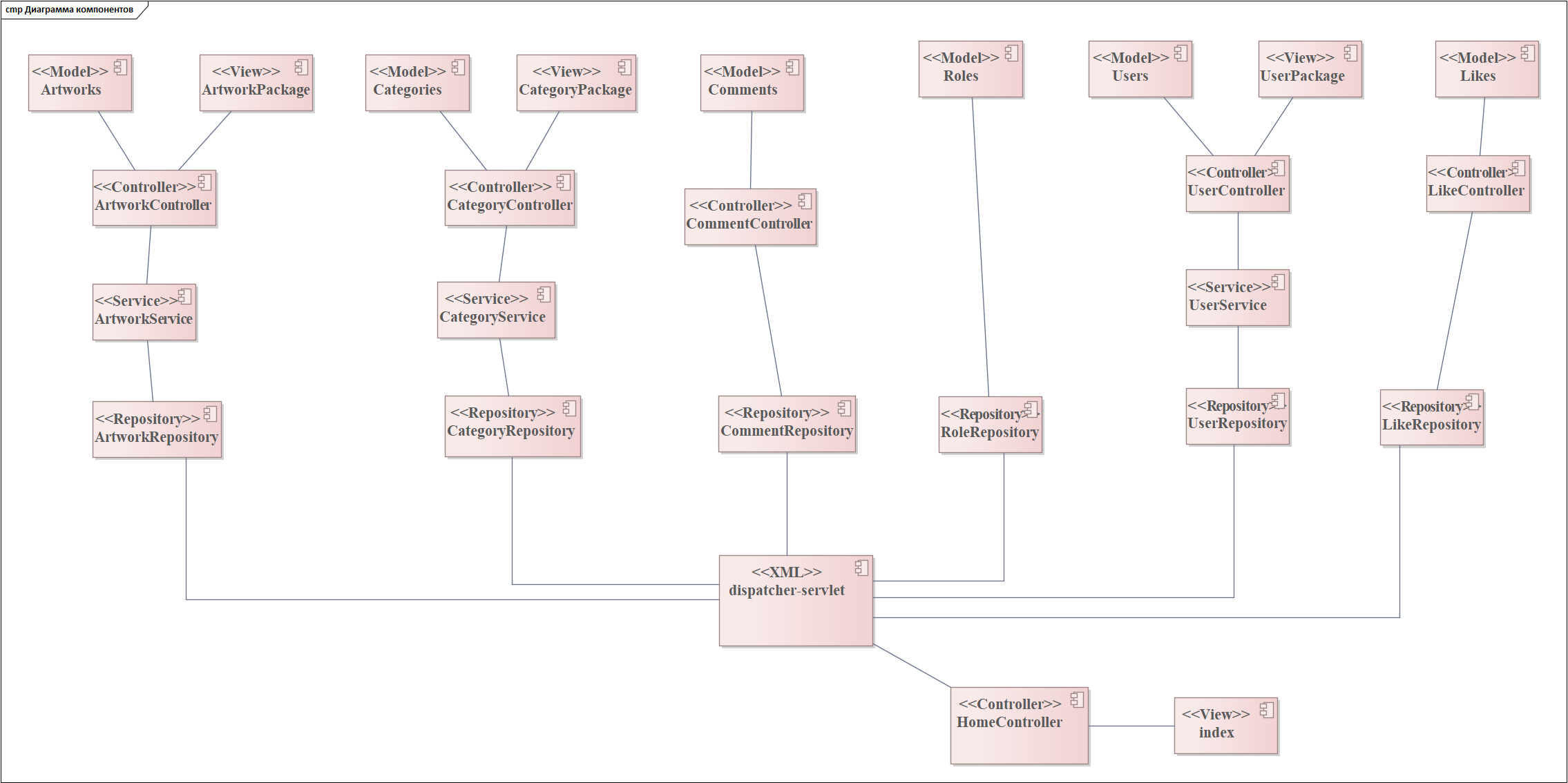


Рисунок 7. Диаграмма компонентов

На рисунке 8 представлена диаграмма развертывания [1] для визуализации архитектуры системы.



Рисунок 8. Диаграмма развертывания

На рисунке 9 представлена схема взаимодействия программ (ГОСТ 19.701-90) [3].

Рисунок 9. Схема взаимодействия программ

# 3.3 Реализация приложения для ИС

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

3.3.1 Реализация публикаций

Код контроллера представлен в приложении А

3.3.2 Реализация управления публикациями

Код представлен в приложении Б

3.3.3 Реализация категорий

Код представлен в приложении В

# 3.4 Шаблоны страниц

Шаблоны страницприложенияпредставлены на страницах 10-18.

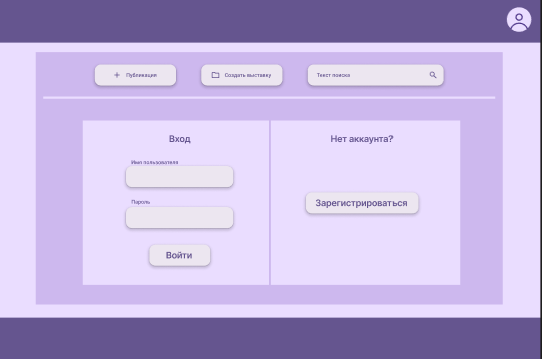


Рисунок 10. Страница авторизации

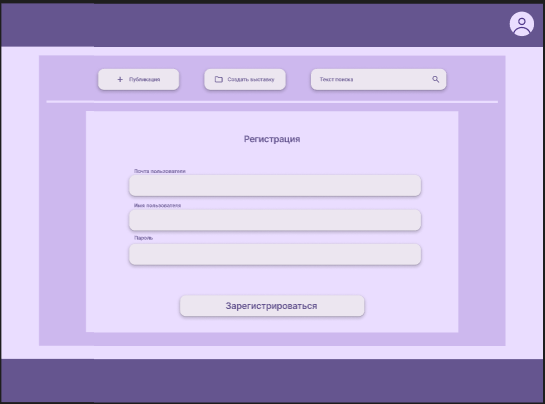


Рисунок 11. Регистрации

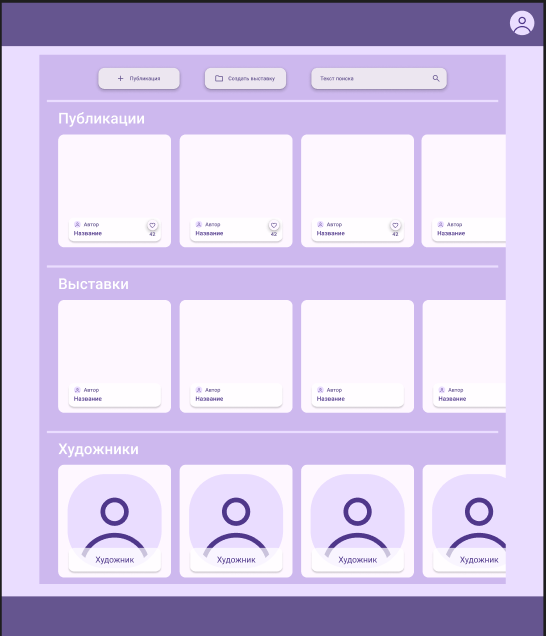


Рисунок 12. Главная страница

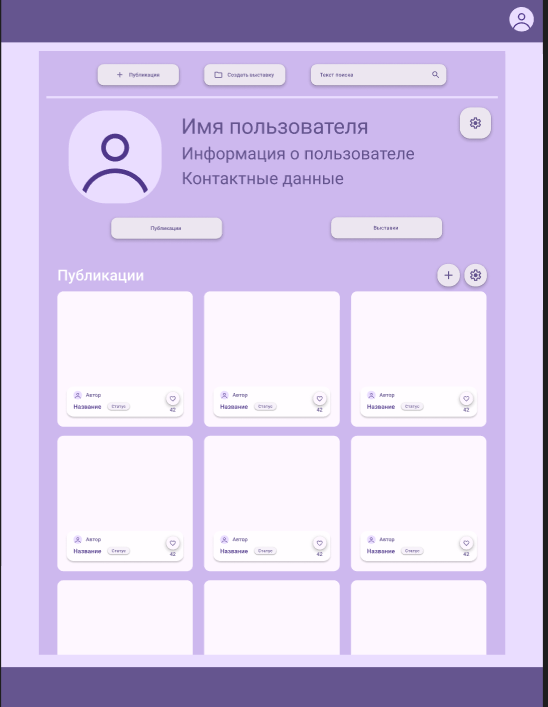


Рисунок 13. Страница с профилем художника



Рисунок 14. Страница просмотра публикации

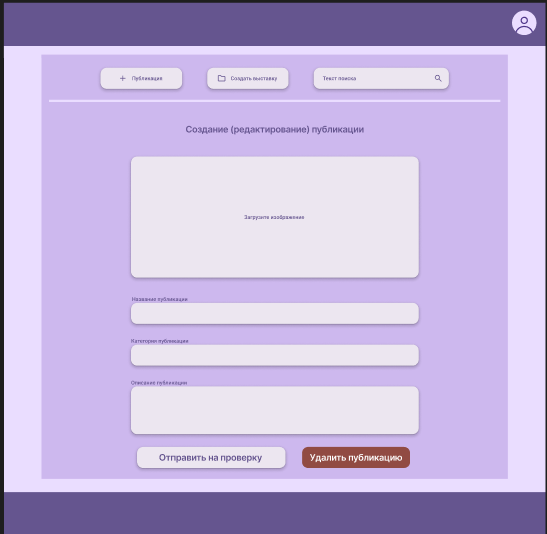


Рисунок 15. Страница создания публикации

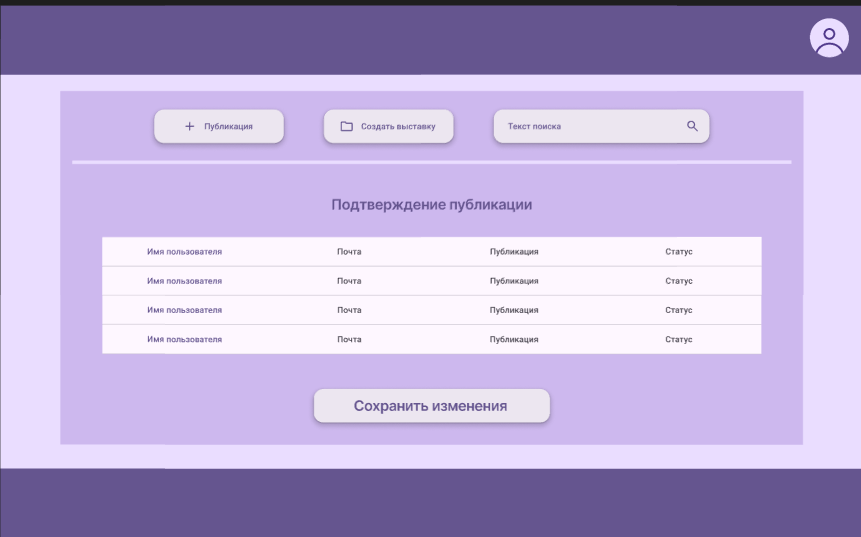


Рисунок 16. Страница с управлением публикаций

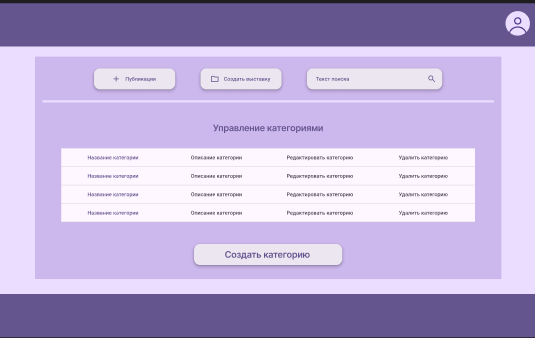


Рисунок 17. Страница с управлением категорий

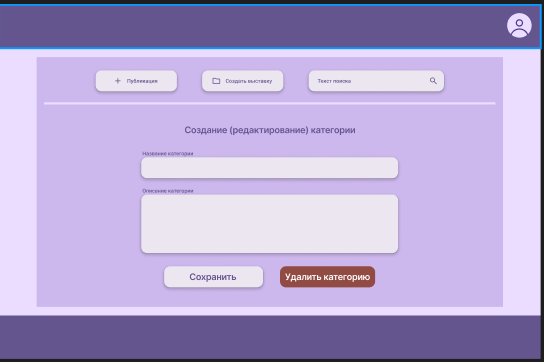


Рисунок 18. Страница создания категорий

# 3.5 Страницы готового приложения

Страницыготового приложенияпредставлены на страницах 19-28.

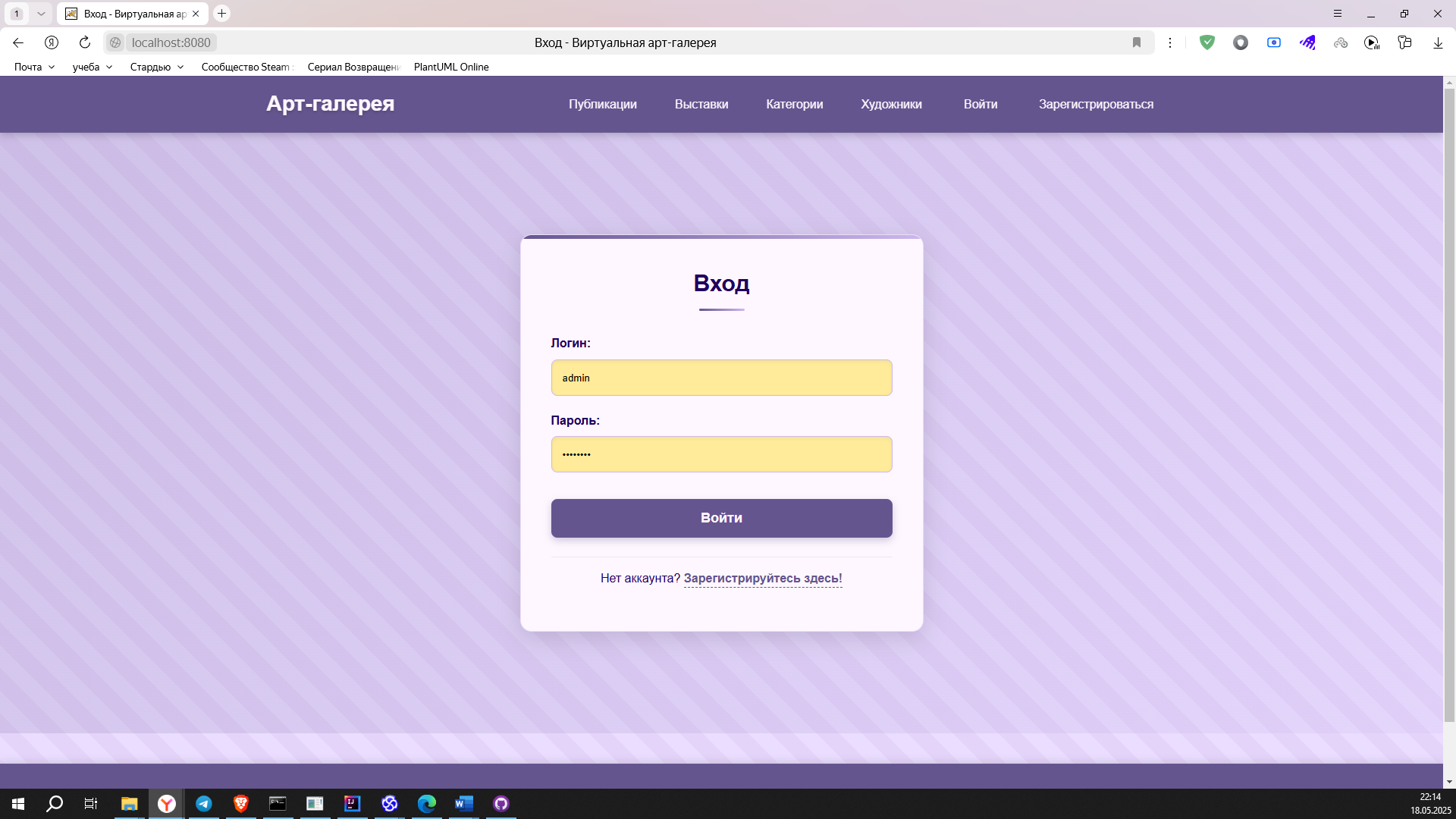


Рисунок 19. Страница авторизации

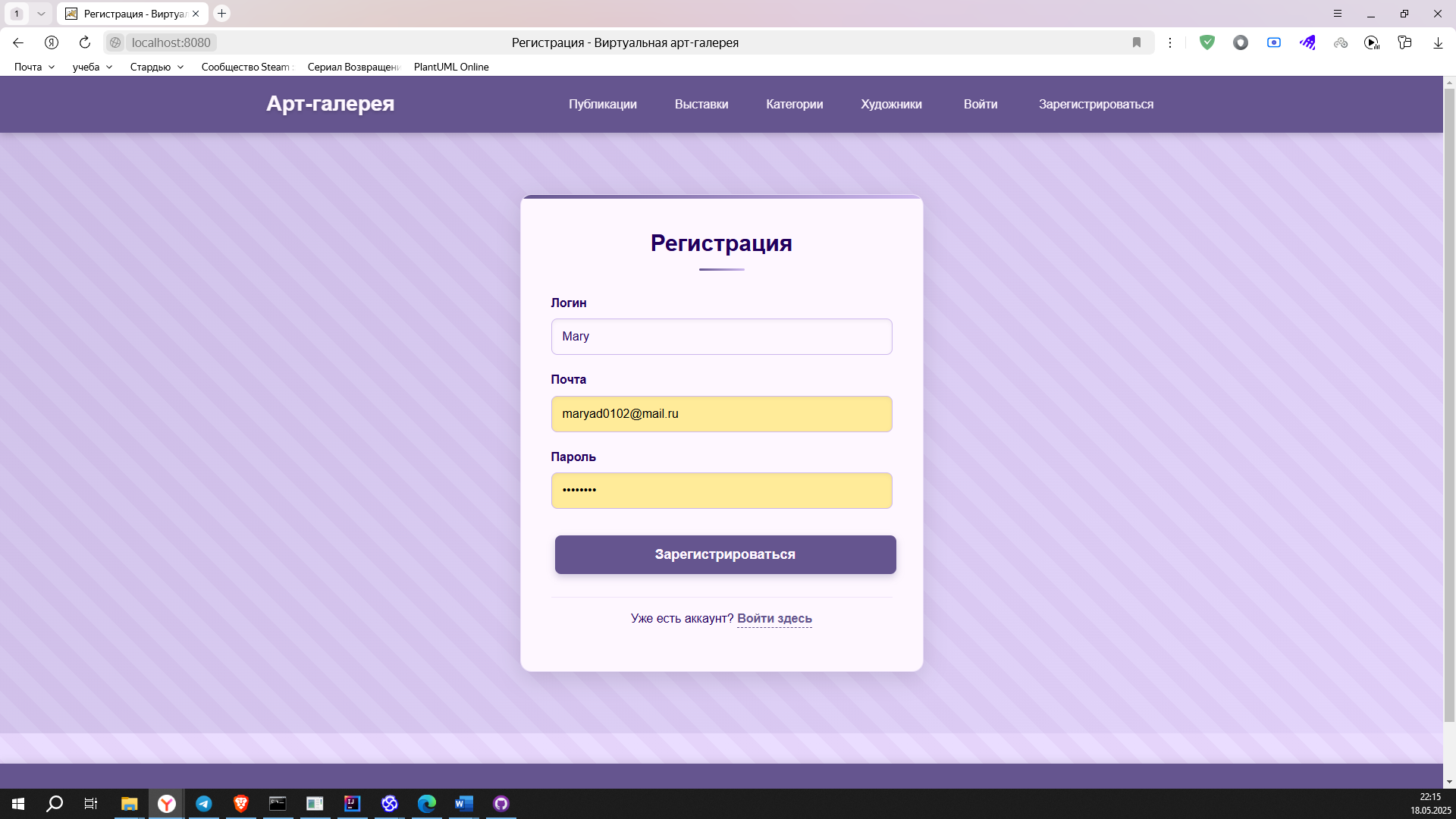


Рисунок 20. Регистрации

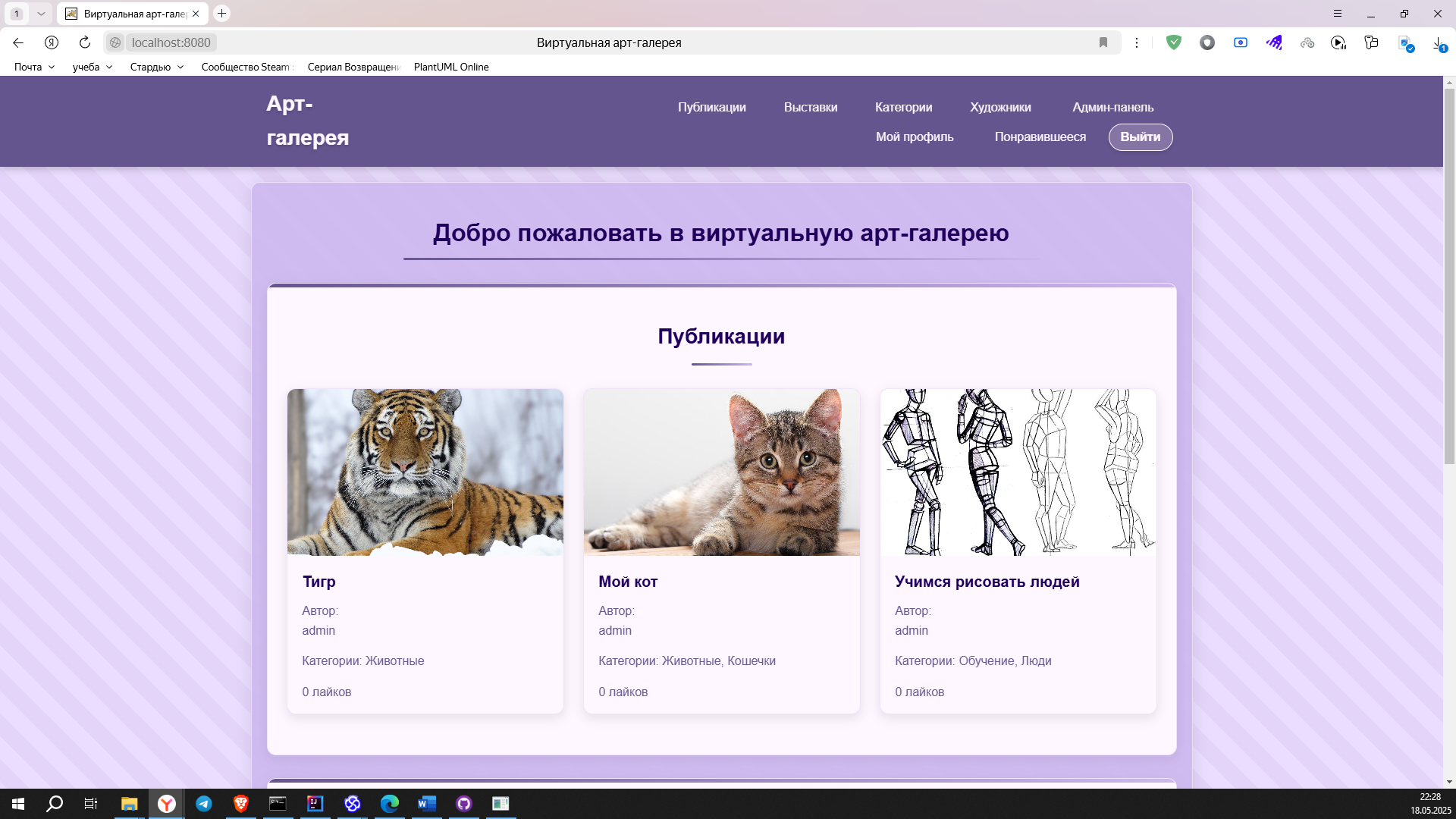


Рисунок 21. Главная страница

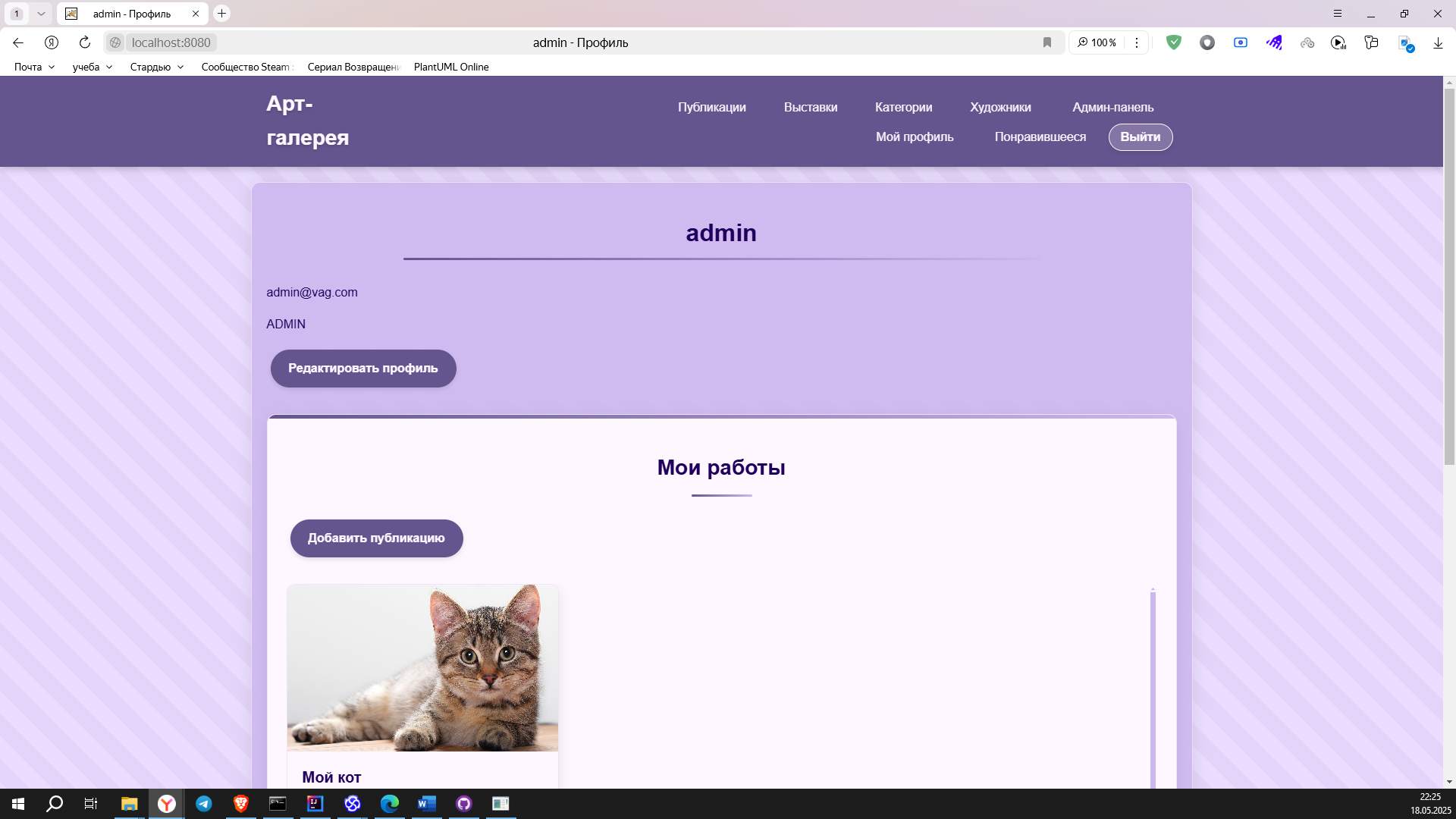


Рисунок 22. Страница с профилем художника

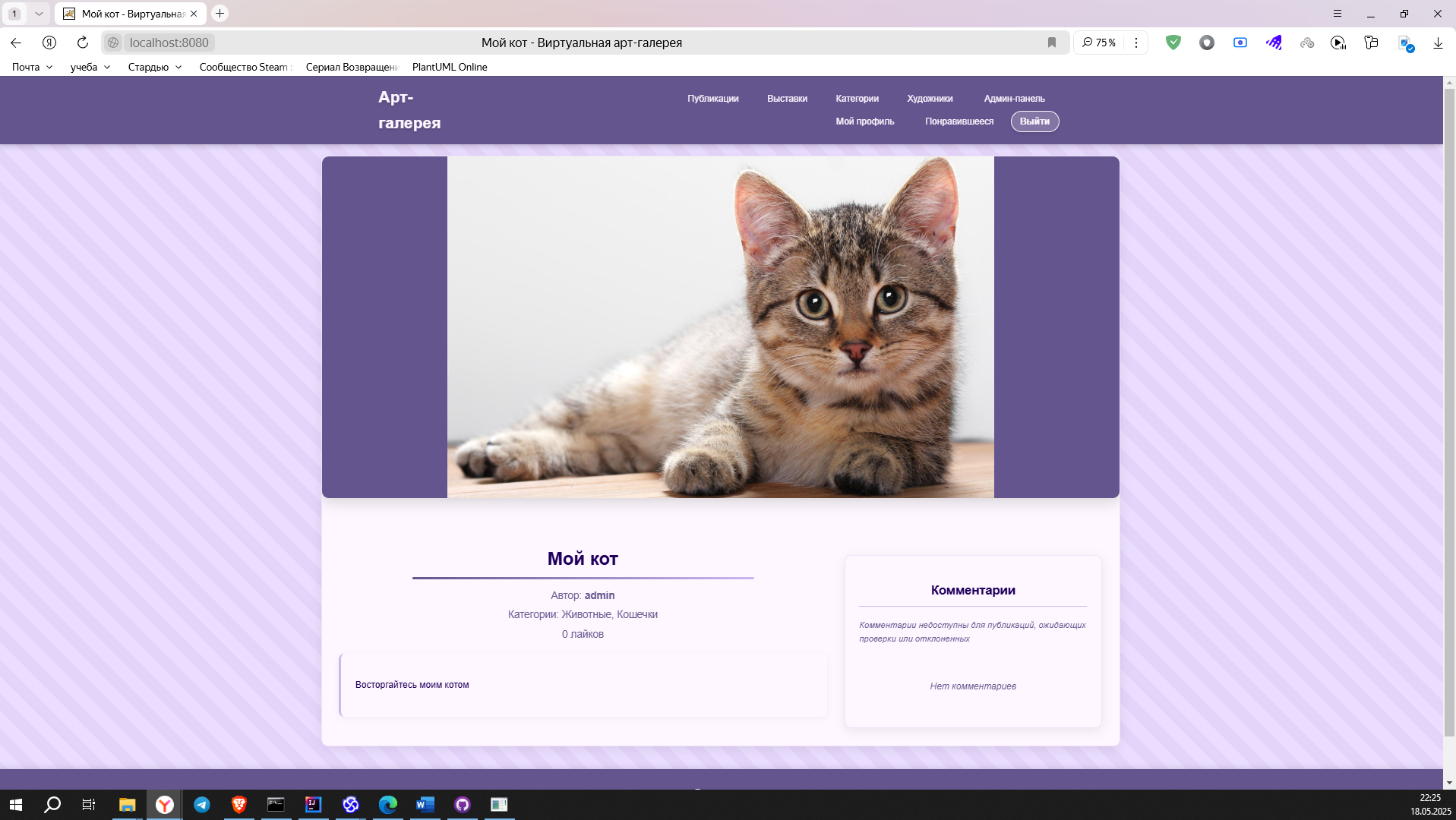


Рисунок 23. Страница просмотра публикации

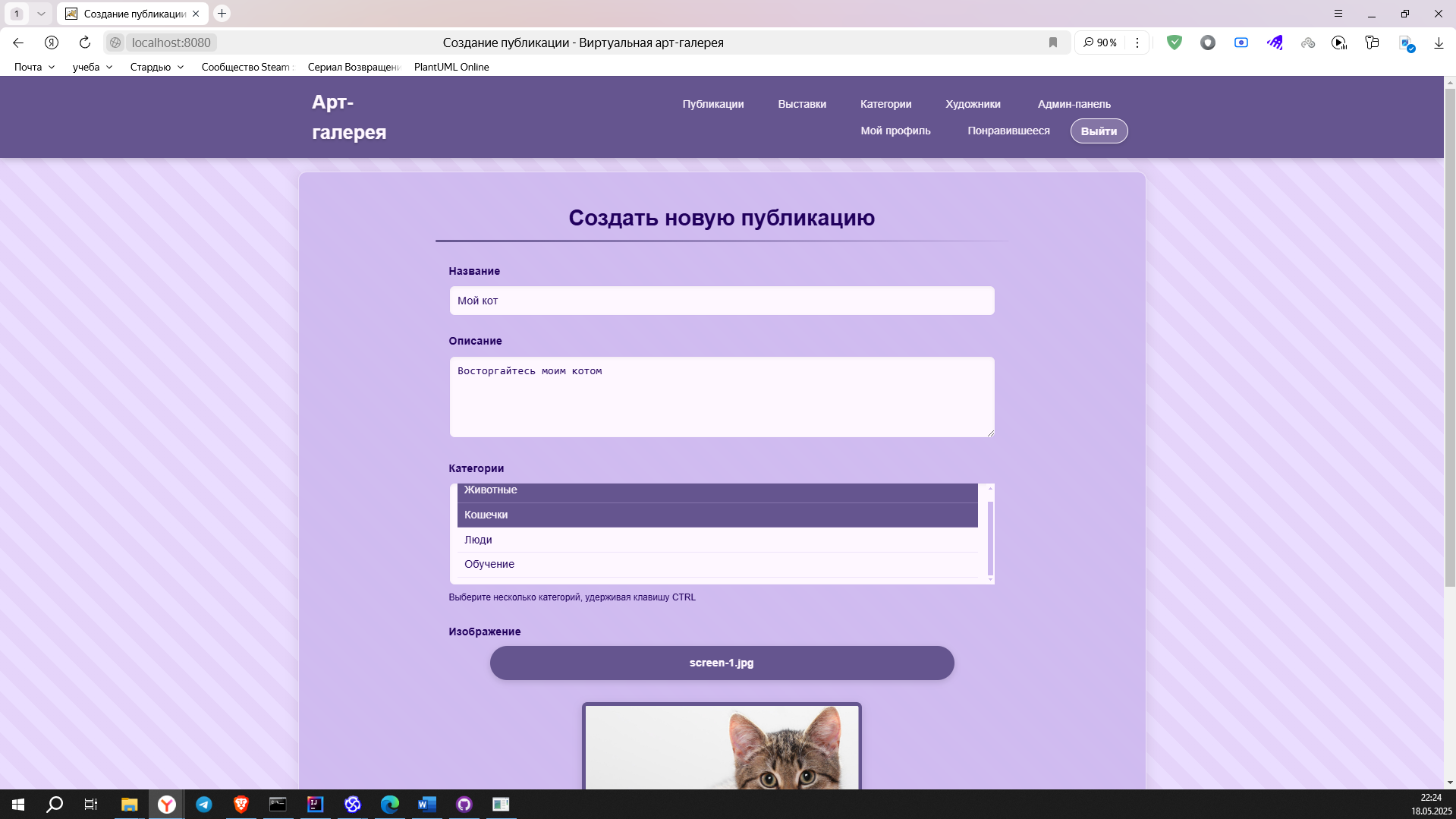


Рисунок 24. Страница создания публикации

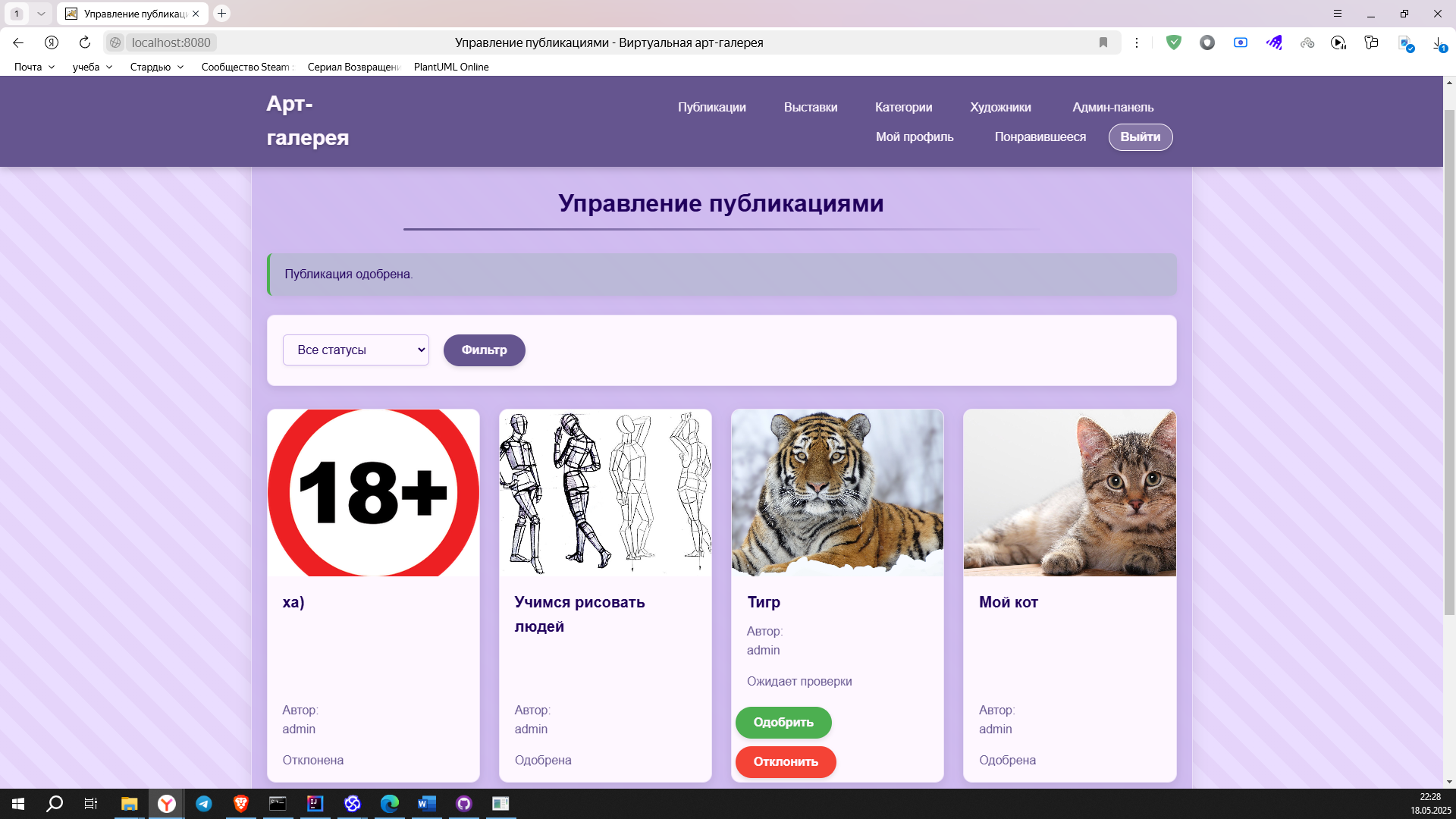


Рисунок 25. Страница с управлением публикациями

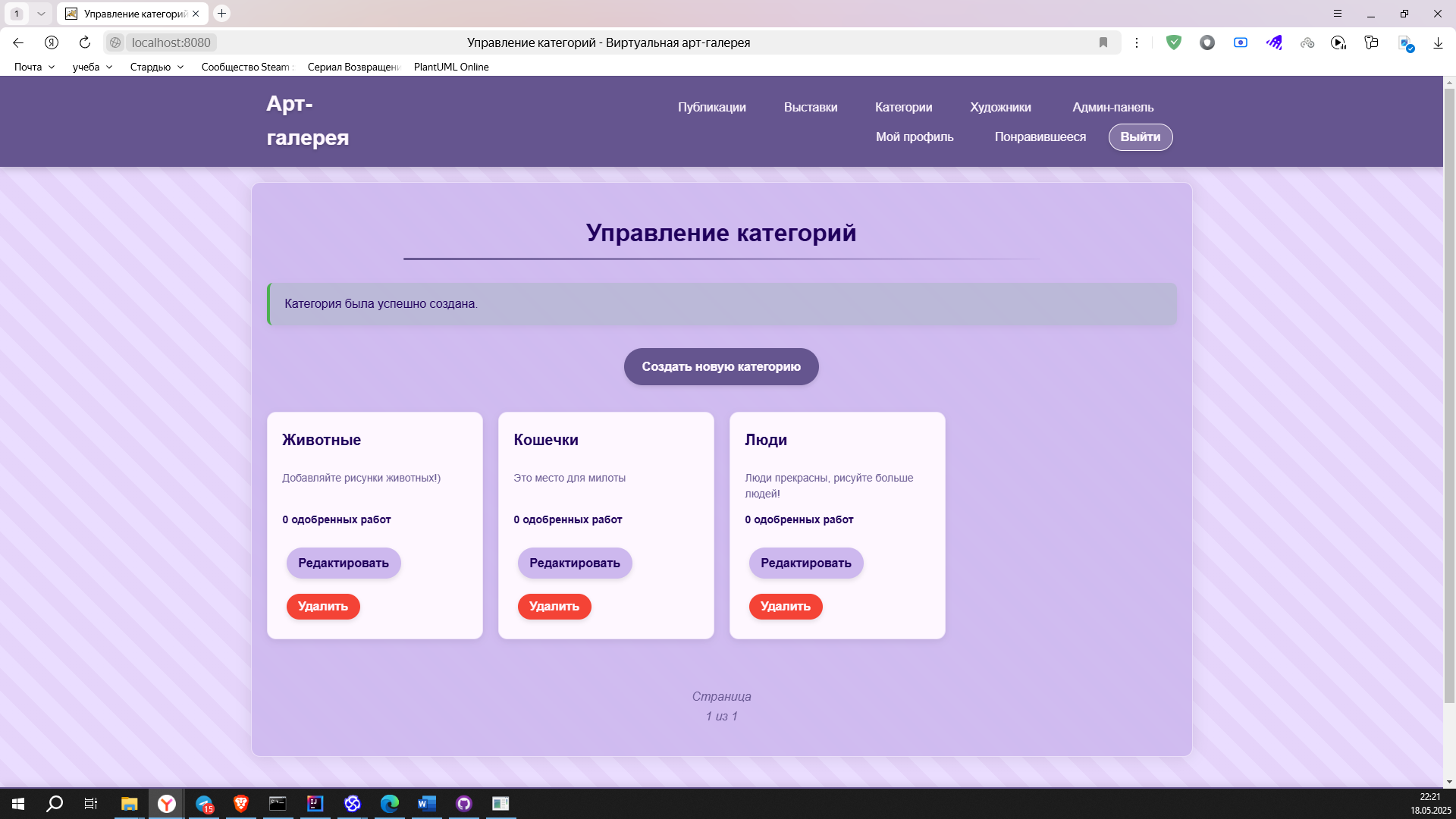


Рисунок 26. Страница с управлением категорий

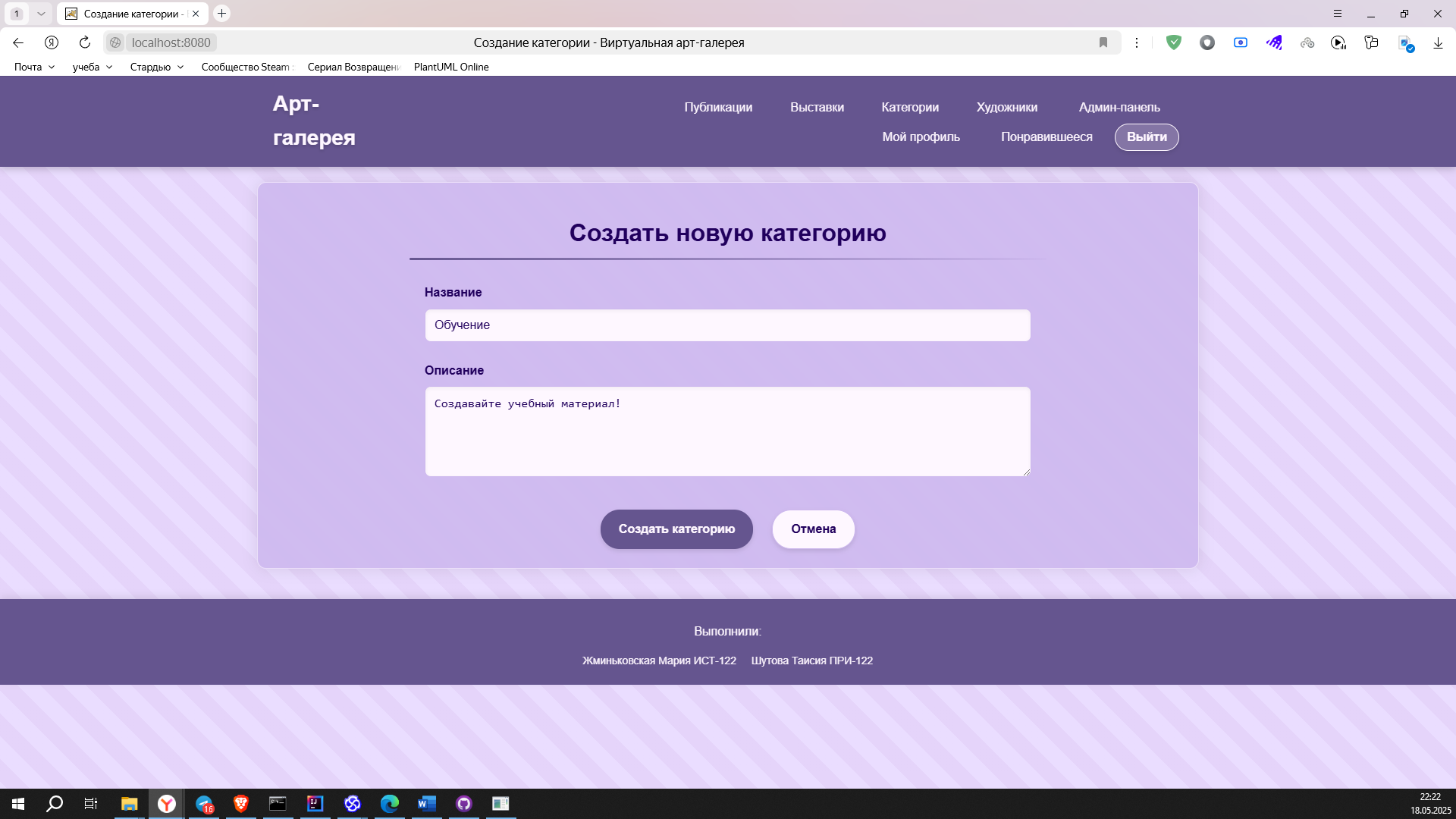


Рисунок 27. Страница создания категорий

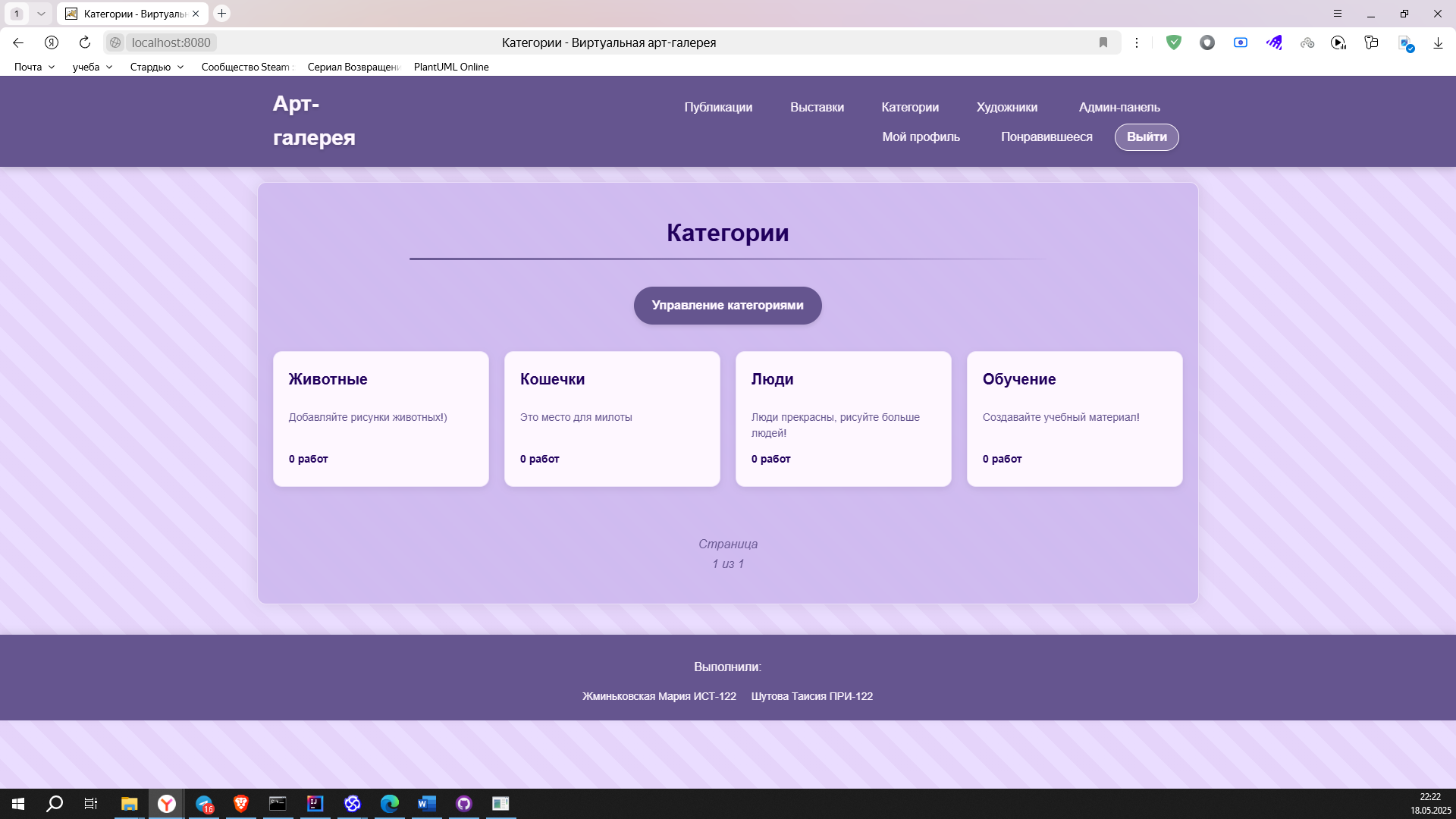


Рисунок 28. Просмотр категорий

# 4 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ

Для того чтобы определить производительность разрабатываемой системы необходимо провести нагрузочное тестирование с постепенным увеличением нагрузки. В качестве объекта тестирования был выбран объект «Публикация». Нагрузочное тестирование позволит выявить уязвимые места в быстром добавлении публикаций различными пользователями, выполняющими действие одновременно.

Нагрузочное тестирование проводится с помощью инструмента Apache JMeter [4]. Были определены следующие параметры потока:

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

Объект «Публикация» будет подвержен нагрузочному тестированию согласно последовательности действий, описанной ниже.

1. GET-запрос на страницу авторизации. Необходимо для того, чтобы авторизовать пользователя.
2. POST-запрос на авторизацию. Авторизуется единственный пользователь, заранее созданный в базе данных.
3. GET-запрос на страницу отображения всех существующий публикаций. Один из вариантов перехода к действию «Добавить публикацию».
4. GET-запрос на форму добавления публикации.
5. POST-запрос на добавление публикации. В качестве параметров для создания новой записи в базе данных передаются ID категории, заранее созданной в системе, название публикации и описание публикации. Также передаётся файл изображения, расположенный по одному единственному маршруту.

Нагрузочное тестирование с указанными выше условиями и параметрами было успешно проведено без перебоев, ошибок и провальных попыток исполнения запроса (рис. 29).

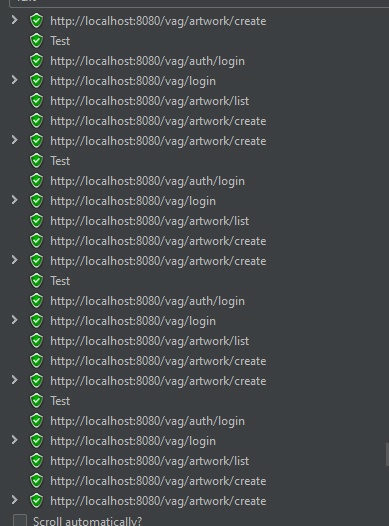


Рисунок 29. Нагрузочное тестирование публикаций

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 пагинация данных. Позволяет избежать загрузки всех записей при отображении списков.
* Оптимизация загрузки связанных сущностей с помощью EntityGraph и Join Fetch.
* Кэширование соединений с базой данных с помощью HikariCP.
* Транзакционная обработка.
* Ленивая загрузка данных для некоторых типов объектов.
* Эффективное управление базой данных.
* Отсутствие блокирующих операций.
* Изоляция и безопасность операций.

# ЗАКЛЮЧЕНИЕ

В рамках выполнения курсового проекта была реализована информационная система «Виртуальная арт-галерея», модуль «Администрирование». Разработка данного модуля позволила создать функциональный инструмент для управления публикациями, взаимодействия с художниками и посетителями, а также анализа статистики посещаемости.

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. Моисеев А.Н., Литовченко М.И. Основы языка UML: учеб. пособие. – Томск: Издательство Томского государственного университета, 2023. – 96 с. ISBN 978-5-907722-06-4

2. Брюс Сильвер BPMN – Метод и стиль [Текст] / Брюс Сильвер — 1-е изд.: Zerde Publishing, 2011 — 27 c.

3. ГОСТ 19.701-90 Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения / [Электронный ресурс] // Интернет и Право : [сайт]. — URL: https://internet-law.ru/gosts/gost/28346/ (дата обращения: 18.05.2025).

4. JMeter как инструмент для автоматизации функционального тестирования / [Электронный ресурс] // Хабр : [сайт]. — URL: https://habr.com/ru/companies/tochka/articles/741698/ (дата обращения: 18.05.2025).

# ПРИЛОЖЕНИЕ А

**ArtworkController:**

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:**

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

**AdminController:**

package com.example.vag.controller;

import com.example.vag.model.Artwork;

import com.example.vag.model.Category;

import com.example.vag.model.Exhibition;

import com.example.vag.model.User;

import com.example.vag.service.ArtworkService;

import com.example.vag.service.CategoryService;

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.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

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

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

import javax.validation.Valid;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.stream.Collectors;

@Controller

@RequestMapping("/admin")

public class AdminController {

private final UserService userService;

private final ArtworkService artworkService;

private final CategoryService categoryService;

private final ExhibitionService exhibitionService;

public AdminController(UserService userService, ArtworkService artworkService,

CategoryService categoryService, ExhibitionService exhibitionService) {

this.userService = userService;

this.artworkService = artworkService;

this.categoryService = categoryService;

this.exhibitionService = exhibitionService;

}

@GetMapping("/users")

public String listUsers(Model model) {

List<User> users = userService.findAll();

model.addAttribute("users", users);

return "admin/users";

}

@GetMapping("/artworks")

public String listArtworks(

@RequestParam(required = false) String status,

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

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

Model model) {

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

Page<Artwork> artworkPage;

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

artworkPage = artworkService.findByStatus(status.toUpperCase(), pageable);

System.out.println("Выбран статус фильтрации: " + status);

} else {

artworkPage = artworkService.findAllPaginated(pageable);

System.out.println("Найдено " + artworkPage.getTotalElements() + " публикаций");

Map<String, Long> statusCounts = new HashMap<>();

for (Artwork artwork : artworkPage.getContent()) {

String stat = artwork.getStatus();

statusCounts.put(stat, statusCounts.getOrDefault(stat, 0L) + 1);

}

statusCounts.forEach((stat, count) ->

System.out.println("Статус " + stat + ": " + count + " публикаций"));

}

model.addAttribute("artworks", artworkPage);

return "admin/artworks";

}

@PostMapping("/artworks/approve/{id}")

public String approveArtwork(@PathVariable Long id) {

artworkService.approveArtwork(id);

return "redirect:/admin/artworks?approved";

}

@PostMapping("/artworks/reject/{id}")

public String rejectArtwork(@PathVariable Long id) {

artworkService.rejectArtwork(id);

return "redirect:/admin/artworks?rejected";

}

@GetMapping("/categories")

public String listCategories(

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

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

Model model) {

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

Page<Category> categoryPage = categoryService.findAll(pageable);

categoryPage.forEach(category ->

category.setApprovedArtworksCount(

artworkService.countApprovedArtworksByCategoryId(category.getId())

));

model.addAttribute("categories", categoryPage);

return "admin/categories";

}

@GetMapping("/categories/create")

public String showCreateForm(Model model) {

model.addAttribute("category", new Category());

return "admin/create-category";

}

@PostMapping("/categories/create")

public String createCategories(@Valid @ModelAttribute("category") Category category,

BindingResult bindingResult,

RedirectAttributes redirectAttributes,

Model model) {

if (categoryService.findByName(category.getName()).isPresent()) {

bindingResult.rejectValue("name", "error.category", "Категория с таким именем уже существует");

}

if (bindingResult.hasErrors()) {

model.addAttribute("category", category);

return "admin/create-category";

}

categoryService.save(category);

redirectAttributes.addFlashAttribute("message", "Категория успешно создана!");

redirectAttributes.addAttribute("created", true);

return "redirect:/admin/categories";

}

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

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

Category category = categoryService.findById(id).orElseThrow();

model.addAttribute("category", category);

return "admin/edit-category";

}

@PostMapping("/categories/edit/{id}")

public String updateCategory(

@PathVariable("id") Long id,

@Valid @ModelAttribute("category") Category category,

BindingResult result,

RedirectAttributes redirectAttributes

) {

if (result.hasErrors()) {

return "admin/edit-category";

}

categoryService.update(id, category);

redirectAttributes.addFlashAttribute("success", "Category updated");

return "redirect:/admin/categories";

}

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

public String deleteCategory(@PathVariable Long id, RedirectAttributes redirectAttributes) {

Category category = categoryService.findById(id).orElseThrow();

categoryService.delete(category);

redirectAttributes.addAttribute("deleted", true);

return "redirect:/admin/categories";

}

@GetMapping("/exhibitions")

public String listExhibitions(Model model) {

List<Exhibition> exhibitions = exhibitionService.findAll();

model.addAttribute("exhibitions", exhibitions);

return "admin/exhibitions";

}

@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;

}

}

# ПРИЛОЖЕНИЕ В

**CategoryController:**

package com.example.vag.controller;

import com.example.vag.model.Artwork;

import com.example.vag.model.Category;

import com.example.vag.service.ArtworkService;

import com.example.vag.service.CategoryService;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

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

import java.util.List;

@Controller

@RequestMapping("/category")

public class CategoryController {

private final CategoryService categoryService;

private final ArtworkService artworkService;

public CategoryController(CategoryService categoryService, ArtworkService artworkService) {

this.categoryService = categoryService;

this.artworkService = artworkService;

}

@GetMapping("/list")

public String listCategories(

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

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

Model model) {

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

Page<Category> categoryPage = categoryService.findAll(pageable);

categoryPage.forEach(category ->

category.setApprovedArtworksCount(

artworkService.countApprovedArtworksByCategoryId(category.getId())

));

model.addAttribute("categories", categoryPage);

return "category/list";

}

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

public String viewCategory(@PathVariable Long id,

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

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

Model model) {

Category category = categoryService.findById(id).orElseThrow();

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

model.addAttribute("category", category);

model.addAttribute("artworks", artworkPage);

return "category/details";

}

}

**CategoryRepository:**

package com.example.vag.repository;

import com.example.vag.model.Artwork;

import com.example.vag.model.Category;

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 CategoryRepository extends JpaRepository<Category, Long> {

@Query("SELECT c FROM Category c WHERE c.id IN :ids")

List<Category> findAllByIds(@Param("ids") List<Long> ids);

@Query("SELECT DISTINCT c FROM Category c LEFT JOIN FETCH c.artworks")

List<Category> findAllWithArtworks();

Page<Category> findAll(Pageable pageable);

Optional<Category> findByName(String name);

}