МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ»

(КНИТУ-КАИ)

Институт компьютерных технологий и защиты информации

(наименование института (факультета), филиала)

Кафедра Прикладной Математики и Информатики

(наименование кафедры)

09.03.04 «Программная инженерия»

(шифр и наименование направления подготовки (специальности))

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

по дисциплине: Технологии Веб-программирования

на тему: «Клиент-серверное приложение «Автостоянка»»

Обучающийся 4311 Ильин А. А.

(номер группы) (подпись, дата) (Ф.И.О.)

Руководитель доцент каф. ПМИ Тутубалин П. И.

(должность) (Ф.И.О.)

Курсовая работа зачтена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

Казань, 2025 г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc202010604)

[1. СТАДИЯ «ТЕХНИЧЕСКОЕ ЗАДАНИЕ» 7](#_Toc202010605)

[1.1. Общая постановка задачи 7](#_Toc202010606)

[1.2. Разработка терминологического словаря предметной области 8](#_Toc202010607)

[1.3. Общие требования к проектируемой системе 9](#_Toc202010608)

[1.4. Категории пользователей проектируемой системы 9](#_Toc202010609)

[2. СТАДИЯ «ЭСКИЗНЫЙ ПРОЕКТ» 11](#_Toc202010610)

[2.1. Разработка программной архитектуры системы 11](#_Toc202010611)

[2.2. Разработка диаграмм вариантов использования компонентов 14](#_Toc202010612)

[3. СТАДИЯ «ТЕХНИЧЕСКИЙ ПРОЕКТ» 18](#_Toc202010613)

[3.1. Разработка диаграмм концептуальных классов 18](#_Toc202010614)

[3.2. Разработка диаграмм программных классов 20](#_Toc202010615)

[4. МАКЕТЫ HTML-СТРАНИЦ РАЗРАБАТЫВАЕМОЙ СИСТЕМЫ 23](#_Toc202010616)

[4.1. Макет страницы авторизации 23](#_Toc202010617)

[4.2. Макет страницы регистрации 23](#_Toc202010618)

[4.3. Макет главной страницы 24](#_Toc202010619)

[4.4. Макет системы администратора 24](#_Toc202010620)

[4.5. Макет страницы статистики 25](#_Toc202010621)

[5. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЕЙ СИСТЕМЫ 25](#_Toc202010622)

[ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ 31](#_Toc202010623)

[ПРИЛОЖЕНИЕ 1. Файл config/database.js 32](#_Toc202010624)

[ПРИЛОЖЕНИЕ 2. Файл controllers/artworkController.js 32](#_Toc202010625)

[ПРИЛОЖЕНИЕ 3. Файл controllers/authController.js 35](#_Toc202010626)

[ПРИЛОЖЕНИЕ 4. Файл controllers/authorController.js 38](#_Toc202010627)

[ПРИЛОЖЕНИЕ 5. Файл controllers/exhibitionController.js 40](#_Toc202010628)

[ПРИЛОЖЕНИЕ 6. Файл controllers/genreController.js 44](#_Toc202010629)

[ПРИЛОЖЕНИЕ 7. Файл src\controllers\managerController.js 46](#_Toc202010630)

[ПРИЛОЖЕНИЕ 8. Файл src\controllers\styleController.js 46](#_Toc202010631)

[ПРИЛОЖЕНИЕ 9. Файл src\controllers\ticketController.js 48](#_Toc202010632)

[ПРИЛОЖЕНИЕ 10. Файл src\controllers\userController.js 52](#_Toc202010633)

[ПРИЛОЖЕНИЕ 11. Файл src\middleware\auth.js 55](#_Toc202010634)

[ПРИЛОЖЕНИЕ 12. Файл src\middleware\setUser.js 55](#_Toc202010635)

[ПРИЛОЖЕНИЕ 13. Файл src\middleware\upload.js 55](#_Toc202010636)

[ПРИЛОЖЕНИЕ 14. Файл src\middleware\validation.js 56](#_Toc202010637)

[ПРИЛОЖЕНИЕ 15. Файл src\models\Artwork.js 59](#_Toc202010638)

[ПРИЛОЖЕНИЕ 16. Файл src\models\Author.js 60](#_Toc202010639)

[ПРИЛОЖЕНИЕ 17. Файл src\models\Exhibition.js 60](#_Toc202010640)

[ПРИЛОЖЕНИЕ 18. Файл src\models\ExhibitionArtwork.js 61](#_Toc202010641)

[ПРИЛОЖЕНИЕ 19. Файл src\models\Genre.js 61](#_Toc202010642)

[ПРИЛОЖЕНИЕ 20. Файл src\models\index.js 61](#_Toc202010643)

[ПРИЛОЖЕНИЕ 21. Файл src\models\Style.js 62](#_Toc202010644)

[ПРИЛОЖЕНИЕ 22. Файл src\models\Ticket.js 62](#_Toc202010645)

[ПРИЛОЖЕНИЕ 23. Файл src\models\User.js 63](#_Toc202010646)

[ПРИЛОЖЕНИЕ 24. Файл src\public\js\admin.js 64](#_Toc202010647)

[ПРИЛОЖЕНИЕ 25. Файл src\public\js\artworks.js 64](#_Toc202010648)

[ПРИЛОЖЕНИЕ 26. Файл src\public\js\authors.js 70](#_Toc202010649)

[ПРИЛОЖЕНИЕ 27. Файл src\public\js\exhibition.js 74](#_Toc202010650)

[ПРИЛОЖЕНИЕ 28. Файл src\public\js\genres.js 82](#_Toc202010651)

[ПРИЛОЖЕНИЕ 29. Файл src\public\js\main.js 84](#_Toc202010652)

[ПРИЛОЖЕНИЕ 30. Файл src\public\js\profile.js 86](#_Toc202010653)

[ПРИЛОЖЕНИЕ 31. Файл src\public\js\statistics.js 88](#_Toc202010654)

[ПРИЛОЖЕНИЕ 32. Файл src\public\js\styles.js 88](#_Toc202010655)

[ПРИЛОЖЕНИЕ 33. Файл src\public\js\ticket.js 91](#_Toc202010656)

[ПРИЛОЖЕНИЕ 34. Файл src\public\js\tickets.js 94](#_Toc202010657)

[ПРИЛОЖЕНИЕ 35. Файл src\public\js\users.js 96](#_Toc202010658)

[ПРИЛОЖЕНИЕ 36. Файл src\routes\artworkRoutes.js 99](#_Toc202010659)

[ПРИЛОЖЕНИЕ 37. Файл src\routes\authorRoutes.js 100](#_Toc202010660)

[ПРИЛОЖЕНИЕ 38. Файл src\routes\authRoutes.js 100](#_Toc202010661)

[ПРИЛОЖЕНИЕ 39. Файл src\routes\exhibitionRoutes.js 100](#_Toc202010662)

[ПРИЛОЖЕНИЕ 40. Файл src\routes\genreRoutes.js 101](#_Toc202010663)

[ПРИЛОЖЕНИЕ 41. Файл src\routes\index.js 101](#_Toc202010664)

[ПРИЛОЖЕНИЕ 42. Файл src\routes\reportRoutes.js 101](#_Toc202010665)

[ПРИЛОЖЕНИЕ 43. Файл src\routes\styleRoutes.js 101](#_Toc202010666)

[ПРИЛОЖЕНИЕ 44. Файл src\routes\ticketRoutes.js 102](#_Toc202010667)

[ПРИЛОЖЕНИЕ 45. Файл src\routes\userRoutes.js 102](#_Toc202010668)

[ПРИЛОЖЕНИЕ 46. Файл src\scripts\createAdmin.js 102](#_Toc202010669)

[ПРИЛОЖЕНИЕ 47. Файл src\views\admin\artworks.ejs 103](#_Toc202010670)

[ПРИЛОЖЕНИЕ 48. Файл src\views\admin\authors.ejs 105](#_Toc202010671)

[ПРИЛОЖЕНИЕ 49. Файл src\views\admin\genres.ejs 107](#_Toc202010672)

[ПРИЛОЖЕНИЕ 50. Файл src\views\admin\index.ejs 108](#_Toc202010673)

[ПРИЛОЖЕНИЕ 51. Файл src\views\admin\styles.ejs 109](#_Toc202010674)

[ПРИЛОЖЕНИЕ 52. Файл src\views\admin\tickets.ejs 110](#_Toc202010675)

[ПРИЛОЖЕНИЕ 53. Файл src\views\admin\users.ejs 112](#_Toc202010676)

[ПРИЛОЖЕНИЕ 54. Файл src\views\auth\login.ejs 114](#_Toc202010677)

[ПРИЛОЖЕНИЕ 55. Файл src\views\auth\register.ejs 114](#_Toc202010678)

[ПРИЛОЖЕНИЕ 56. Файл src\views\layouts\main.ejs 115](#_Toc202010679)

[ПРИЛОЖЕНИЕ 57. Файл src\views\manager\exhibitions.ejs 115](#_Toc202010680)

[ПРИЛОЖЕНИЕ 58. Файл src\views\manager\statistics.ejs 119](#_Toc202010681)

[ПРИЛОЖЕНИЕ 59. Файл src\views\partials\footer.ejs 119](#_Toc202010682)

[ПРИЛОЖЕНИЕ 61. Файл src\app.js 120](#_Toc202010683)

[ПРИЛОЖЕНИЕ 62. Файл src\views\index.ejs 123](#_Toc202010684)

# ВВЕДЕНИЕ

В условиях роста интереса к культурным мероприятиям и необходимости удобного доступа к выставкам возникает потребность в инструменте для управления посещением художественных галерей. Разработанное приложение «Художественная галерея» позволяет приобретать билеты на выставки, формировать выставки и получать аналитику продаж, обеспечивая удобство для посетителей и организаторов, а также повышая эффективность управления культурными мероприятиями.

Были поставлены следующие задачи: анализ требований к системе управления выставками и продажами билетов, проектирование архитектуры клиент-серверного приложения, разработка функциональных модулей (управление выставками, билетами, расписанием и аналитикой продаж), а также тестирование и отладка системы для обеспечения её стабильной работы. Для хранения и структурирования данных использовалась реляционная СУБД MySQL, что позволило обеспечить надёжное и эффективное взаимодействие с информацией о пользователях, выставках и транзакциях.

Разработка велась в среде WebStorm — современной и удобной IDE, оптимизированной под JavaScript и веб-технологии. В качестве системы контроля версий использовался Git, а исходный код размещался на GitHub, что обеспечило удобство совместной работы над проектом. Архитектура приложения построена на основе паттерна Model-View-Controller, что гарантировало чёткое разделение логики данных, управления и пользовательского интерфейса, упрощая сопровождение и масштабирование.

Клиентская часть приложения реализована с использованием HTML, CSS, JavaScript, библиотеки jQuery и покрыта модульными тестами на Mocha. Шаблонизатор Ejs позволяет формировать динамичные страницы, а клиентский JavaScript через AJAX-запросы взаимодействует с сервером, обеспечивая отзывчивый и интерактивный интерфейс.

Серверная часть построена на Node.js с фреймворком Express и менеджером пакетов NPM, обменивается данными в формате JSON, обрабатывает запросы пользователей и выполняет бизнес-логику (CRUD-операции, проверка доступности слотов, управление правами доступа). Такая связка технологий обеспечивает высокую производительность и масштабируемость сервиса «Художественная галерея».

# 1. СТАДИЯ «ТЕХНИЧЕСКОЕ ЗАДАНИЕ»

## 1.1. Общая постановка задачи

Наименование разработки: Клиент-серверное приложение «Художественная галерея».

Назначение системы – автоматизация процессов подготовки выставок и продажа билетов на них.

Область применения – публикация выставок со всеми необходимыми сведениями и продажа билетов для пользователей. Так же это удобная форма ведения контроля за картинами.

Масштаб развертывания системы – глобальная сеть с возможностью клиент-серверного взаимодействия.

Таблица 1.

Пользователи и бизнес-процессы предметной обрасти

|  |  |
| --- | --- |
| **Пользователи** | **Основные бизнес-процессы** |
| Администратор  Пользователь  Менеджер | **Администрирование системы:**   * управление правами доступа пользователей к компонентам системы; * формирование выставок   **Мониторинг:**   * просмотр выставок * бронирование билетов * просмотр сведений о картинах * просмотр статистики продаж |

## 1.2. Разработка терминологического словаря предметной области

Таблица 2.

Терминологический словарь

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Терминология** | **Применение** | **Комментарии** |
| 1 | Экспонат | Произведение искусства, представленное в галерее для просмотра | Может включать картины, скульптуры, фотографии, инсталляции и другие виды искусства. |
| 2 | Коллекция | Совокупность экспонатов, объединенных по тематике, автору, эпохе или стилю | Коллекции могут быть постоянными или временными, например, для выставок. |
| 3 | Менеджер | Специалист, отвечающий за подбор, организацию и представление экспонатов | Менеджер определяет концепцию выставки и взаимодействует с художниками. |
| 4 | Выставка | Временная или постоянная экспозиция произведений искусства | Выставки могут быть тематическими, персональными или групповыми. |
| 5 | Арт-объект | Произведение искусства, имеющее художественную ценность | Может быть частью экспозиции или храниться в фондах галереи. |
| 6 | Каталог | Печатное или электронное издание с информацией о выставке и экспонатах | Содержит изображения, описания и биографии художников. |
| 7 | Экспозиция | Размещение экспонатов в пространстве галереи | Экспозиция может быть линейной, тематической или хронологической. |

## 1.3. Общие требования к проектируемой системе

***Архитектурные и эксплуатационные требования***

* Масштаб развёртывания системы – глобальная сеть.
* Приложение: двухуровневая клиент серверная архитектура, клиентские приложения пользователей развернуты на рабочих станциях.
* Хранилище данных: централизованное, состоит из одной базы данных.
* Требования безопасности – регистрация и аутентификация пользователей, разграничение прав их доступа к ресурсам системы.

***Требования к функциональным характеристикам***

* Управление доступом пользователей к компонентам и информационным ресурсам системы.
* Поддержка сведений о картинах в актуальном состоянии: формирование и редактирование списков экспонатов, планируемых выставок.
* Бронирование билетов: пользователи будут бронировать места на выставки.
* Формирование выставок указание списка картин, указания даты, времени, периода проведения и места проведения.
* Анализ проведения и доходности проводимых выставок: все продажи сохраняются в базе, по ним будет доступна аналитика.

## 1.4. Категории пользователей проектируемой системы

Схема взаимодействия пользователей системы с ее функциональными модулями, реализующими основные бизнес-процессы (табл. 1), приведена на обобщенной диаграмме вари антов использования (рис. 1).

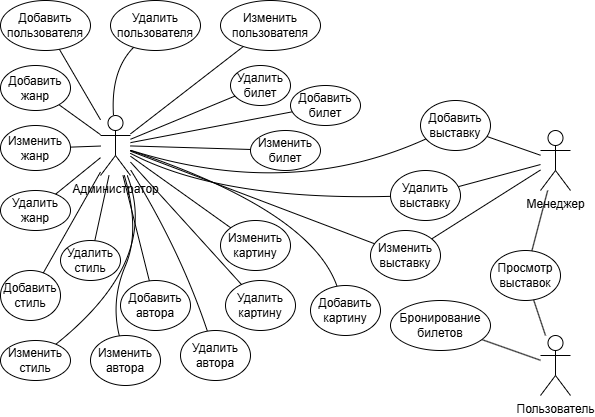


Рисунок 1. Обобщенная UML диаграмма вариантов использования.

**Администратор** отвечает за редактирование всех имеющихся данных в таблице.

**Менеджер** выполняет функции планирования и организации выставок, Может просматривать статистику продаж.

**Пользователь** выполняет функции просмотра выставок и сведений о картинах. Получает доступ к соответствующим локальным модулям системы с правами просмотра текущих выставок, информации о картинах и их авторах. Также может осуществлять бронирование билетов на выставки.

# 2. СТАДИЯ «ЭСКИЗНЫЙ ПРОЕКТ»

В соответствии с обобщенной диаграммой вариантов использования (рис. 1) производится функциональная декомпозиция проектируемой системы. Результаты декомпозиции представляются диаграммой пакетов системы (рис. 2), отражающей ее программную архитектуру, и множеством диаграммам вариантов использования (рис. 3 и 4), детализирующих процессы функционирования отдельных ее подсистем

## 2.1. Разработка программной архитектуры системы

Хранилище данных включает одну базу данных, управляемую единым сервером.

**База данных «Галерея»** используется для хранения информации о картинах, выставках, билетах, пользователях и менеджерах.

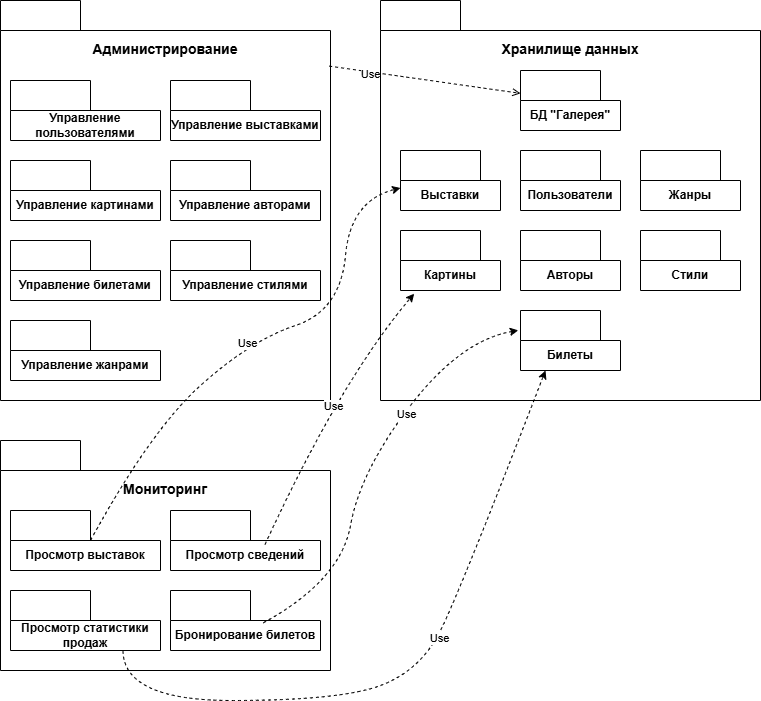


Рисунок 2­. UML диаграмма пакетов

Хранилище данных включает одну базу данных MySQL, управляемую единым сервером. Все таблицы взаимосвязаны через внешние ключи и управляются централизованно через сервер MySQL.

База данных содержит следующие таблицы:

**Пользователи**: используется для хранения информации о пользователях системы (администратор, менеджер, посетитель), включая фамилию, имя, отчество, адрес электронной почты, пароль, роль и временные метки создания и обновления записи.

**Авторы**: содержит информацию об авторах произведений искусства, включая фамилию, имя, отчество, даты рождения и смерти, а также временные метки создания и обновления записи.

**Стили**: хранит данные о художественных стилях, включая их названия и временные метки создания и обновления записи.

**Жанры**: используется для хранения информации о жанрах произведений искусства, включая их названия и временные метки создания и обновления записи.

**Выставки**: содержит данные о выставках, включая название, место проведения, путь к постеру, даты начала и окончания, время работы, стоимость билета, общее и оставшееся количество билетов, статус (предстоящая, активная, завершённая), описание и временные метки создания и обновления записи.

**Произведения искусства**: хранит информацию о произведениях искусства, включая название, размеры, год создания, путь к изображению, описание, а также ссылки на автора, стиль и жанр.

**Билеты**: используется для хранения данных о покупке билетов, включая количество билетов, дату бронирования, общую стоимость, а также ссылки на выставку и пользователя.

Для детализации процессов функционирования подсистем ниже приведены диаграммы вариантов использования для компонентов, включённых в пакеты **«Администрирование»** и **«Мониторинг»**, и сценарий базового варианта использования *«Добавить картину»*.

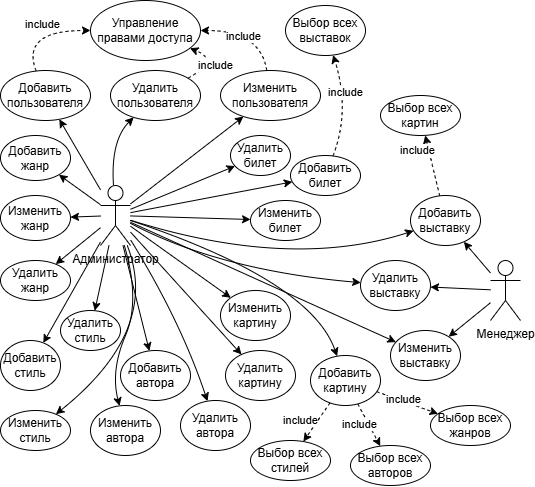


Рисунок 3. UseCase диаграмма подсистемы «Администрирование»



Рисунок 4. UseCase диаграмма подсистемы «Мониторинг»

## 2.2. Разработка диаграмм вариантов использования компонентов

Таблица 3.

**Сценарий варианта использования «Добавить картину»**

|  |  |  |
| --- | --- | --- |
| **Главный раздел** | | |
| Наименование | Добавить картину | |
| Тип | Базовый | |
| Актор | Менеджер | |
| Цель | Добавить картину в базу данных | |
| Краткое описание | Менеджер открывает окно для добавления картин. В нём ему нужно будет указать все необходимые сведения о картине: название, автор, год написания, стиль, жанр, краткое описание и размеры. Так же к картине нужно будет приложить изображение картины. При выполнении всех этих действий, а также успехе добавления со стороны сервера, картина будет сохранена в базе данных. | |
| Связанные варианты использования | Включаемые: «Выбрать все стили», «Выбрать все жанры», «Выбрать всех авторов». | |
| **Раздел «Типичный ход событий»** | | |
| Действия акторов | | Отклик системы |
| 1. Активирует рабочее окно добавления картины.  3. Вводит название.  4. Выбирает автора.  5. Выбирает стиль.  6. Выбирает жанр.  7.Указывает дату создания.  8. Указывает размеров картины.  9.Вводит описания.  10. Прикрепляет изображения.  11.Сохраняет картину.  *Исключение 1*: ошибка ввода полей.  *Исключение 2*: ошибка сохранения. | | 2. Запрашивает все необходимые поля.  12. Проверяет все поля.  13. Добавляет в базу данных. |
| **Раздел «Исключения»** | | |
| Действия акторов | | Действия системы |
| *Исключение 1*: ошибка ввода полей | | |
|  | | 14. вывод уведомления об ошибке под ошибочным полем |
| *Исключение 2*: ошибка сохранения. | | |
|  | | 15. Вывод уведомления об ошибке |

Таблица 4.

**Сценарий варианта использования «Добавить выставку»**

|  |  |  |
| --- | --- | --- |
| **Главный раздел** | | |
| Наименование | Добавить выставку | |
| Тип | Базовый | |
| Актор | Менеджер | |
| Цель | Добавить выставку в базу данных | |
| Краткое описание | Менеджер открывает окно для добавления выставки. В нём ему нужно будет указать все необходимые сведения о выставке: название, место проведения, дата и время, стоимость и количество билетов. Так же нужно будет указать список картин, которые будут представлены на выставке и прикрепить афишу выставки. При выполнении всех этих действий, а также успехе добавления со стороны сервера, выставка будет сохранена в базе данных. | |
| Связанные варианты использования | Включаемые: «Выбор всех картин». | |
| **Раздел «Типичный ход событий»** | | |
| Действия акторов | | Отклик системы |
| 1. Активирует рабочее окно добавления выставки.  3. Вводит название.  4. Вводит место проведения.  5. Выбирает дату начала.  6. Выбирает дату окончания.  7. Выбирает время открытия.  8. Выбирает время закрытия.  9. Вводит количество билетов.  9. Вводит описание  10. Выбирает картины.  11. Прикрепляет изображения.  12.Сохраняет картину.  *Исключение 1*: ошибка ввода полей.  *Исключение 2*: ошибка сохранения. | | 2. Запрашивает все необходимые поля.  13. Проверяет все поля.  14. Добавляет в базу данных. |
| **Раздел «Исключения»** | | |
| Действия акторов | | Действия системы |
| *Исключение 1*: ошибка ввода полей | | |
|  | | 14. вывод уведомления об ошибке под ошибочным полем |
| *Исключение 2*: ошибка сохранения. | | |
|  | | 15. Вывод уведомления об ошибке |

Таблица 5.

**Сценарий варианта использования «Бронирование билетов на выставки»**

|  |  |  |
| --- | --- | --- |
| **Главный раздел** | | |
| Наименование | Бронирование билетов на выставки | |
| Тип | Базовый | |
| Актор | Пользователь | |
| Цель | Забронировать билет на выставку | |
| Краткое описание | Пользователь открывает окно выставки. В нём ему нужно будет указать количество билетов. При наличии билетов, они будут сохранены в личном кабинете пользователя. | |
| Связанные варианты использования | Включаемые: «Добавить данные в базу данных».  Расширяет: «Поиск выставок». | |
| **Раздел «Типичный ход событий»** | | |
| Действия акторов | | Отклик системы |
| 1. Переходит на страницу выставки  2. Вводит количество билетов.  3. Покупает билеты  *Исключение 1*: нет указанного количество билетов. | | 4. Проверяет наличие билетов.  5. Сохраняет билеты в личном кабинете пользователя. |
| **Раздел «Исключения»** | | |
| Действия акторов | | Действия системы |
| *Исключение 1*: нет указанного количество билетов. | | |
|  | | 6. Высвечивает уведомление об ошибке |

# 3. СТАДИЯ «ТЕХНИЧЕСКИЙ ПРОЕКТ»

На стадии технического проекта проводится объектная декомпозиция системы, результаты которой представляются UML диаграммами классов ее компонентов.

## 3.1. Разработка диаграмм концептуальных классов

На этапе разработки диаграмм концептуальных классов проведем объектную декомпозиция подсистемы хранения данных системы Gallery. Диаграмма концептуальных классов (рис. 3.1) подсистемы хранения данных представляет структурную модель, описывающую управление пользователями, картинами, выставками, авторами, билетами, жанрами, стилями.

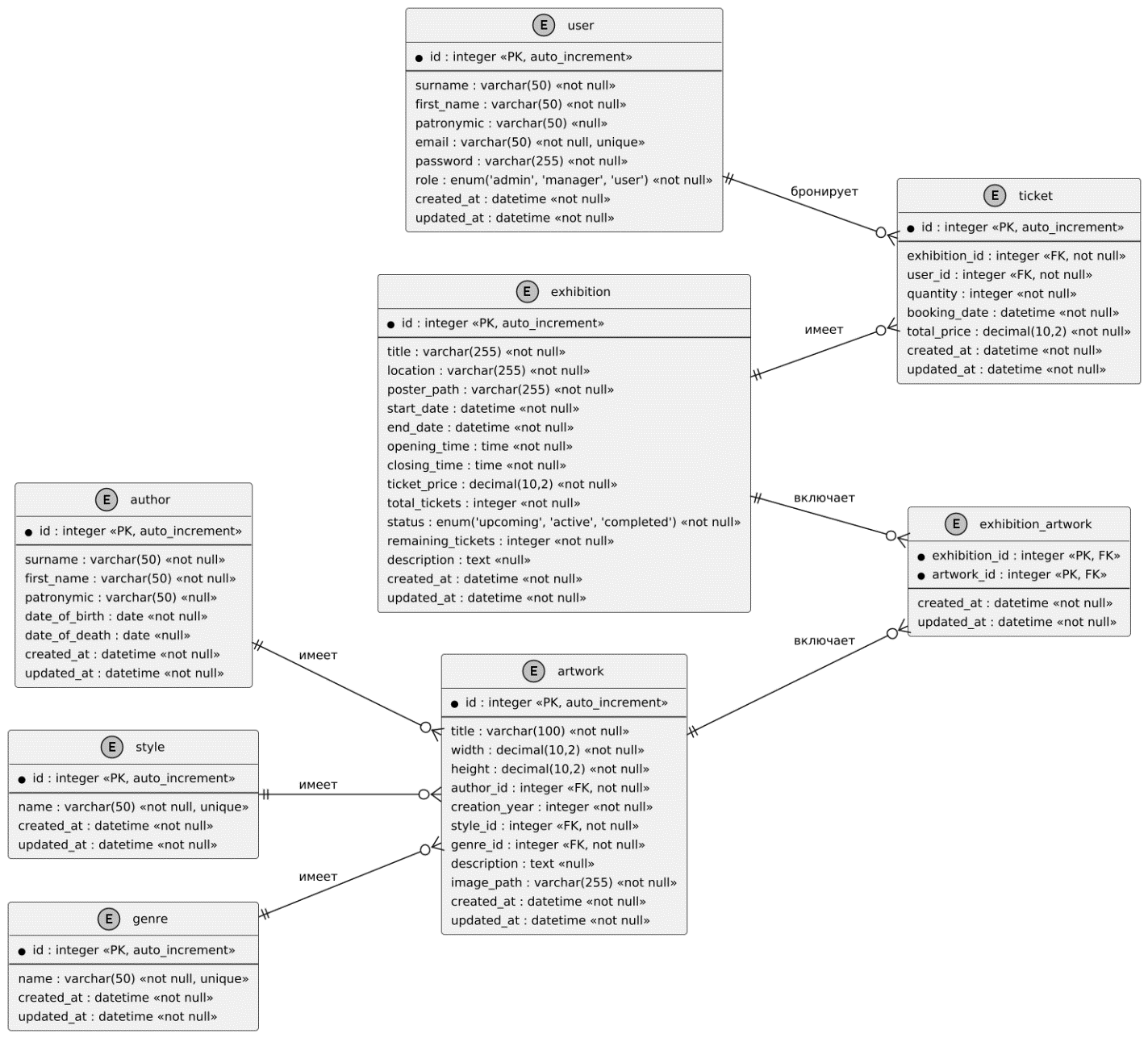
`

Рисунок 5. Диаграмма концептуальных классов подсистемы хранения данных

Объекты класса **User** связаны с объектами класса **Ticket** через атрибут user\_id в модели **Ticket**. Это позволяет определить, какие билеты приобрёл конкретный пользователь. Связь «один-ко-многим» означает, что один пользователь может купить несколько билетов на разные выставки.

Объекты класса **Exhibition** связаны с объектами класса **Ticket** через атрибут exhibition\_id в модели **Ticket**. Это указывает, на какую выставку были приобретены билеты. Связь «один-ко-многим» означает, что одна выставка может иметь множество проданных билетов.

Объекты класса **Author** связаны с объектами класса **Artwork** через атрибут author\_id в модели **Artwork**. Это позволяет определить, какой автор создал конкретное произведение искусства. Связь «один-ко-многим» означает, что один автор может создать несколько произведений искусства.

Объекты класса **Style** связаны с объектами класса **Artwork** через атрибут style\_id в модели **Artwork**. Это задаёт стиль, к которому относится произведение искусства. Связь «один-ко-многим» означает, что один стиль может быть применён к нескольким произведениям искусства.

Объекты класса **Genre** связаны с объектами класса **Artwork** через атрибут genre\_id в модели **Artwork**. Это определяет жанр произведения искусства. Связь «один-ко-многим» означает, что один жанр может классифицировать несколько произведений искусства.

Объекты класса **Exhibition** связаны с объектами класса **Exhibition\_Artwork** через атрибут exhibition\_id, а объекты класса **Artwork** связаны с **Exhibition\_Artwork** через атрибут artwork\_id. Это позволяет определить, какие произведения искусства включены в конкретную выставку. Связь «многие-ко-многим» означает, что одна выставка может включать несколько произведений искусства, а одно произведение искусства может быть представлено на нескольких выставках.

Модель поддерживает разграничение ролей пользователей (администратор, менеджер и обычный пользователь) через атрибут role в **User**. Администраторы и менеджеры управляют выставками, произведениями искусства, авторами, стилями и жанрами, а обычные пользователи покупают билеты на выставки. Система отслеживает статус каждой выставки — upcoming, active или completed.

Данные о билетах позволяют анализировать продажи, контролировать доступность билетов и предоставлять организаторам актуальную информацию о посещаемости выставок.

## 3.2. Разработка диаграмм программных классов

На рисунке 6 представлена диаграмма классов системы, разработанная в соответствии с диаграммой пакетов (рис. 2) и множеством диаграмм вариантов использования (рис. 3-4). В отличие от рассмотренных ранее диаграмм концептуальных классов, эта диаграмма содержит так называемые программные классы, в которых определены не только атрибуты, но и методы — операции классов.

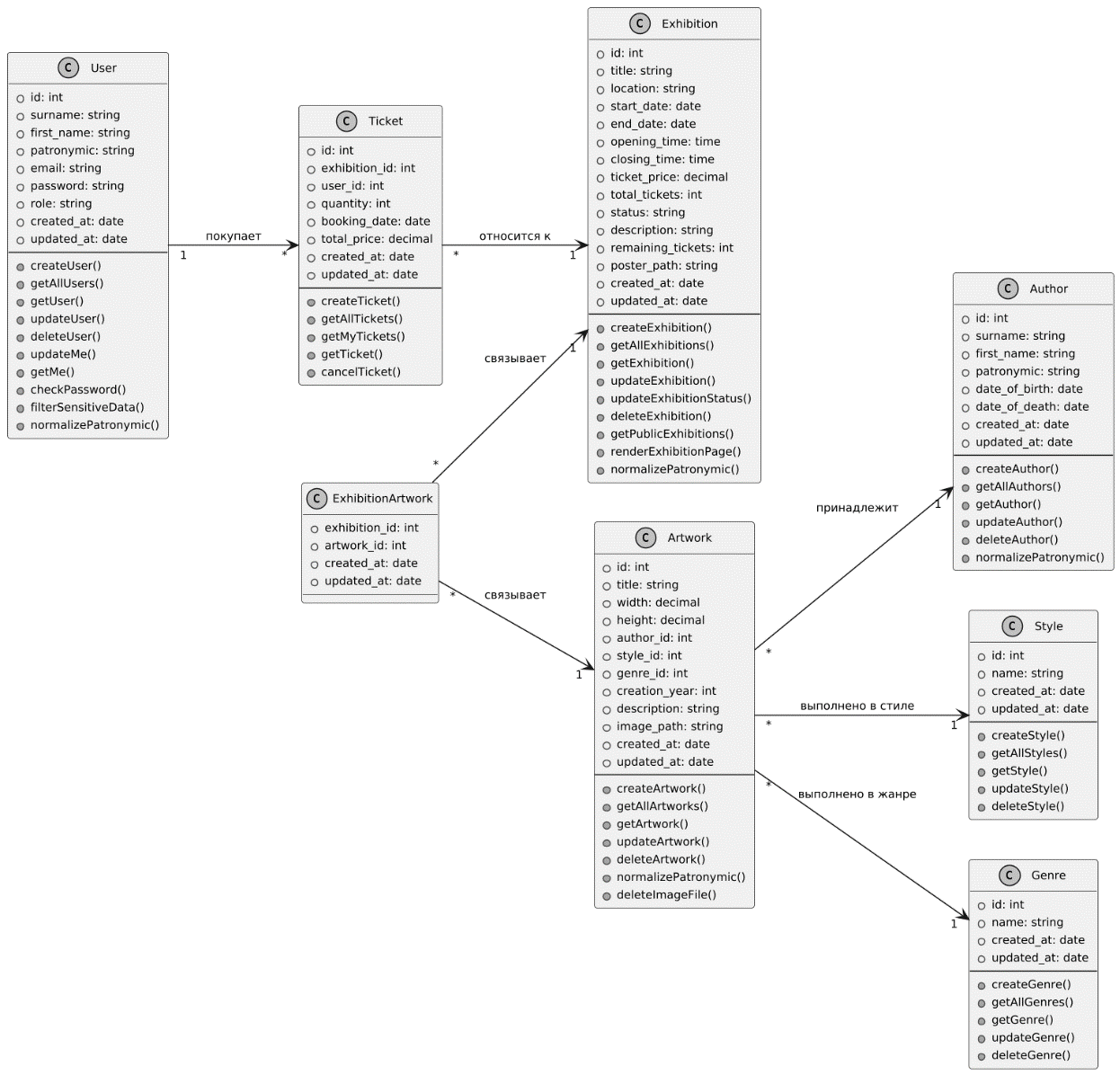


Рисунок 6. Диаграмма классов системы

Класс User отвечает за управление учётными записями пользователей системы (администраторы, менеджеры, обычные пользователи). В классе реализованы методы для создания, получения, обновления и удаления пользователей, а также для работы с собственным профилем (updateMe, getMe). Дополнительно реализованы методы проверки пароля и фильтрации чувствительных данных.

Класс Author хранит информацию об авторах произведений искусства, включая ФИО и даты жизни. Класс предоставляет методы для создания, получения, обновления и удаления авторов, а также вспомогательные методы для корректной обработки данных (например, normalizePatronymic).

Класс Artwork описывает произведения искусства, включая название, размеры, год создания, описание, путь к изображению и связи с автором, стилем и жанром. В классе реализованы методы для создания, получения, обновления и удаления произведений, а также вспомогательные методы для работы с изображениями и обработки данных.

Класс Style предназначен для хранения информации о художественных стилях. Класс предоставляет методы для создания, получения, обновления и удаления стилей.

Класс Genre отвечает за жанры произведений искусства. В классе реализованы методы для создания, получения, обновления и удаления жанров.

Класс Exhibition управляет выставками, храня информацию о названии, месте проведения, датах, времени работы, стоимости билета, количестве билетов, статусе, описании и афише. Класс предоставляет методы для создания, получения, обновления и удаления выставок, а также для получения публичных выставок и рендеринга страниц. Вспомогательные методы обеспечивают корректную обработку связанных данных.

Класс Ticket реализует функционал управления билетами, включая бронирование, хранение информации о количестве, дате бронирования, стоимости и связях с пользователем и выставкой. В классе реализованы методы для создания, получения, отмены и просмотра билетов.

Класс ExhibitionArtwork реализует связь многие-ко-многим между выставками и произведениями искусства, храня идентификаторы обеих сущностей и временные метки.

Связи между классами отражают бизнес-логику предметной области: пользователь может покупать билеты, произведение принадлежит автору, стилю и жанру, выставка содержит множество произведений и продаёт билеты, а билеты относятся к конкретной выставке и пользователю. Все методы классов обеспечивают полный цикл управления данными и поддерживают бизнес-процессы галереи.

# 4. МАКЕТЫ HTML-СТРАНИЦ РАЗРАБАТЫВАЕМОЙ СИСТЕМЫ

## 4.1. Макет страницы авторизации

Макет страницы авторизации пользователей в разрабатываемой системе изображен на рисунке 7.

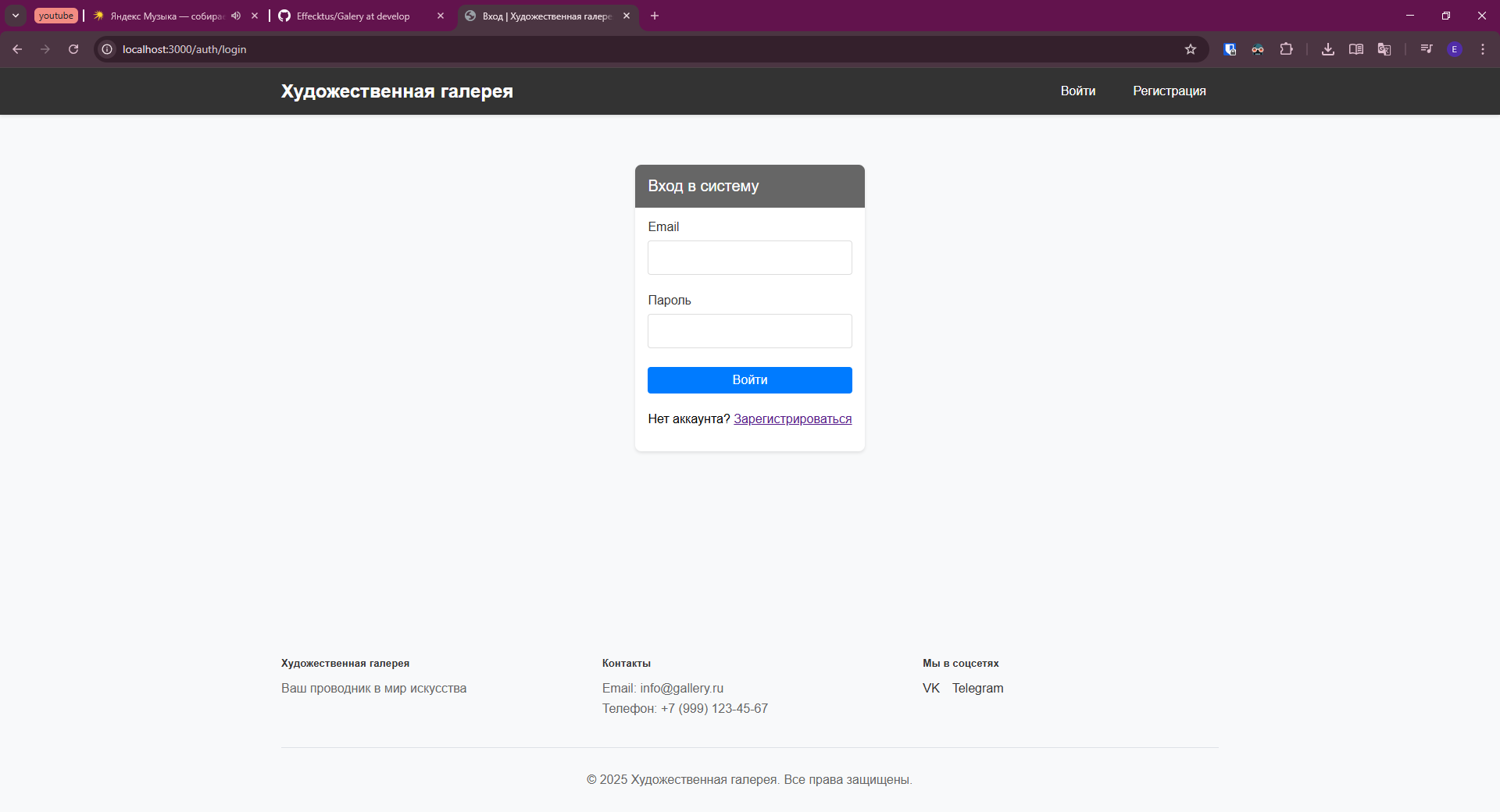


Рисунок 7. Макет страницы авторизации.

## 4.2. Макет страницы регистрации

Макет страницы регистрации пользователей в разрабатываемой системе изображен на рисунке 8.

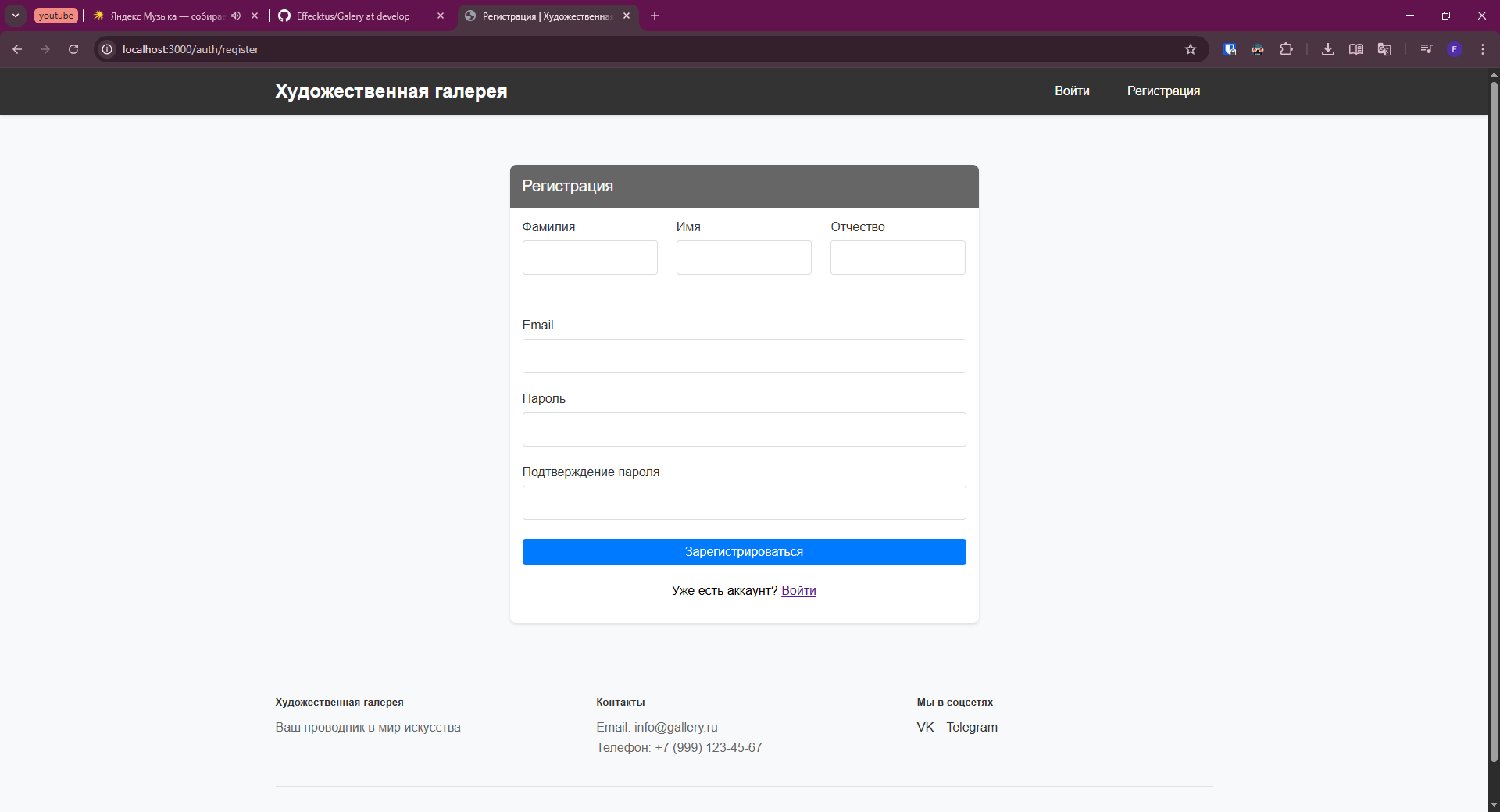


Рисунок 8. Макет страницы регистрации

## 4.3. Макет главной страницы

Макет главной страницы разрабатываемого веб-приложения изображено на рисунке 9.

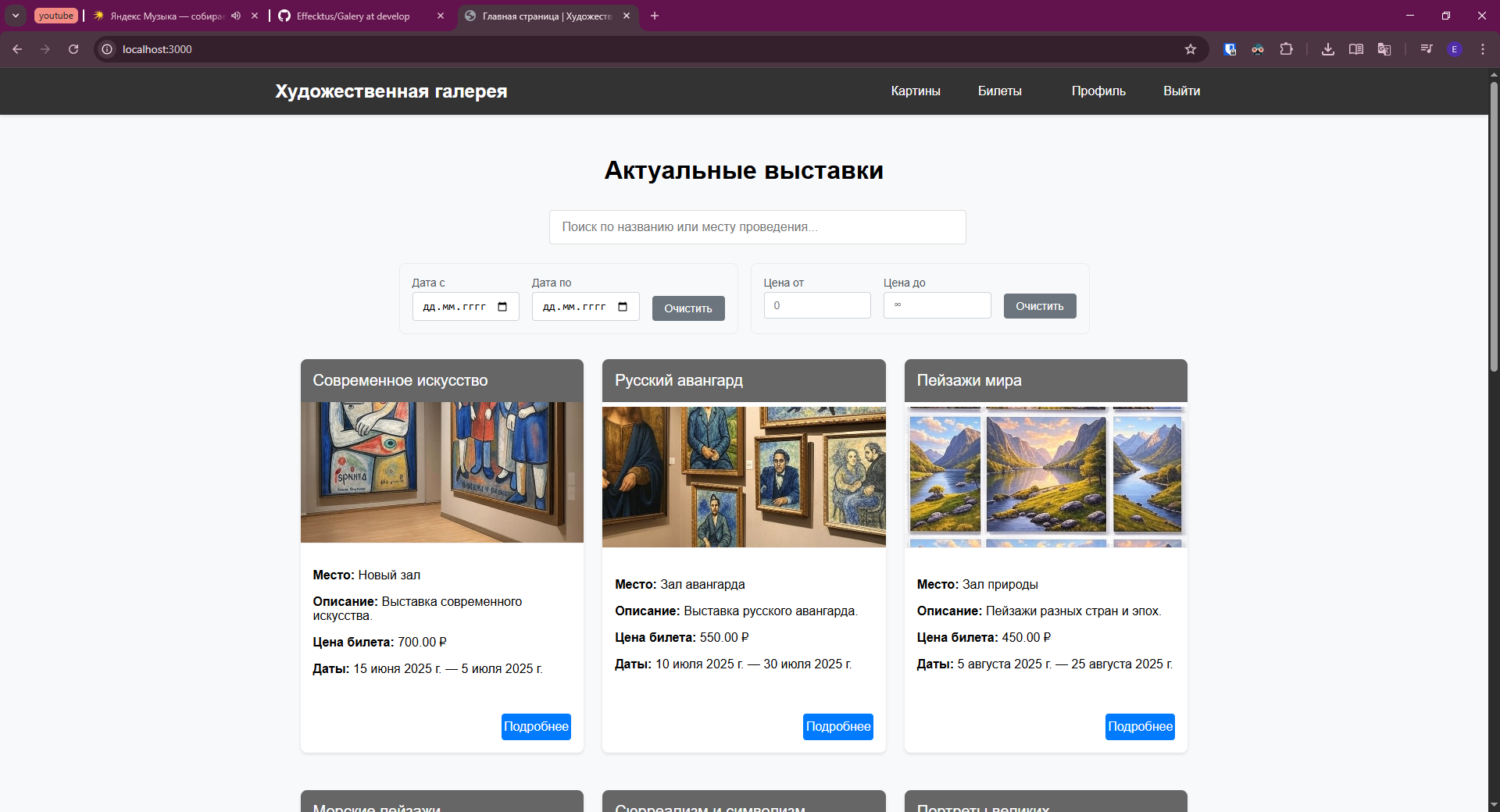


Рисунок 9. Макет главной страницы.

4.4. Макет системы администратора

Макет главной страницы админа в системе изображен на рисунке 10.

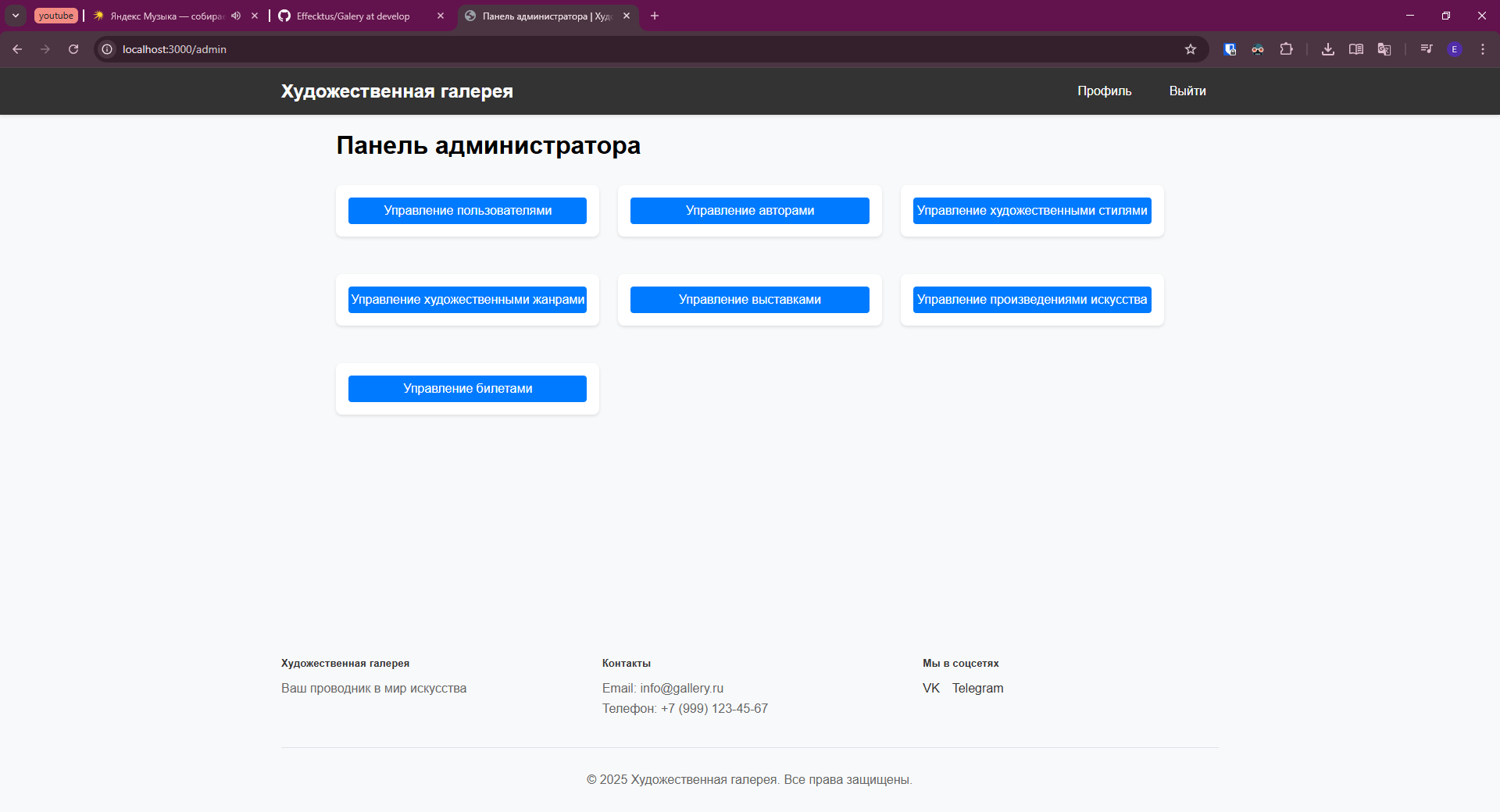


Рисунок 10. Макет системы администрирования.

## 4.5. Макет страницы статистики

Макет страницы приемов изображен на рисунке 11, так же страница отчетности будет включать в себя форму для формирования приема.

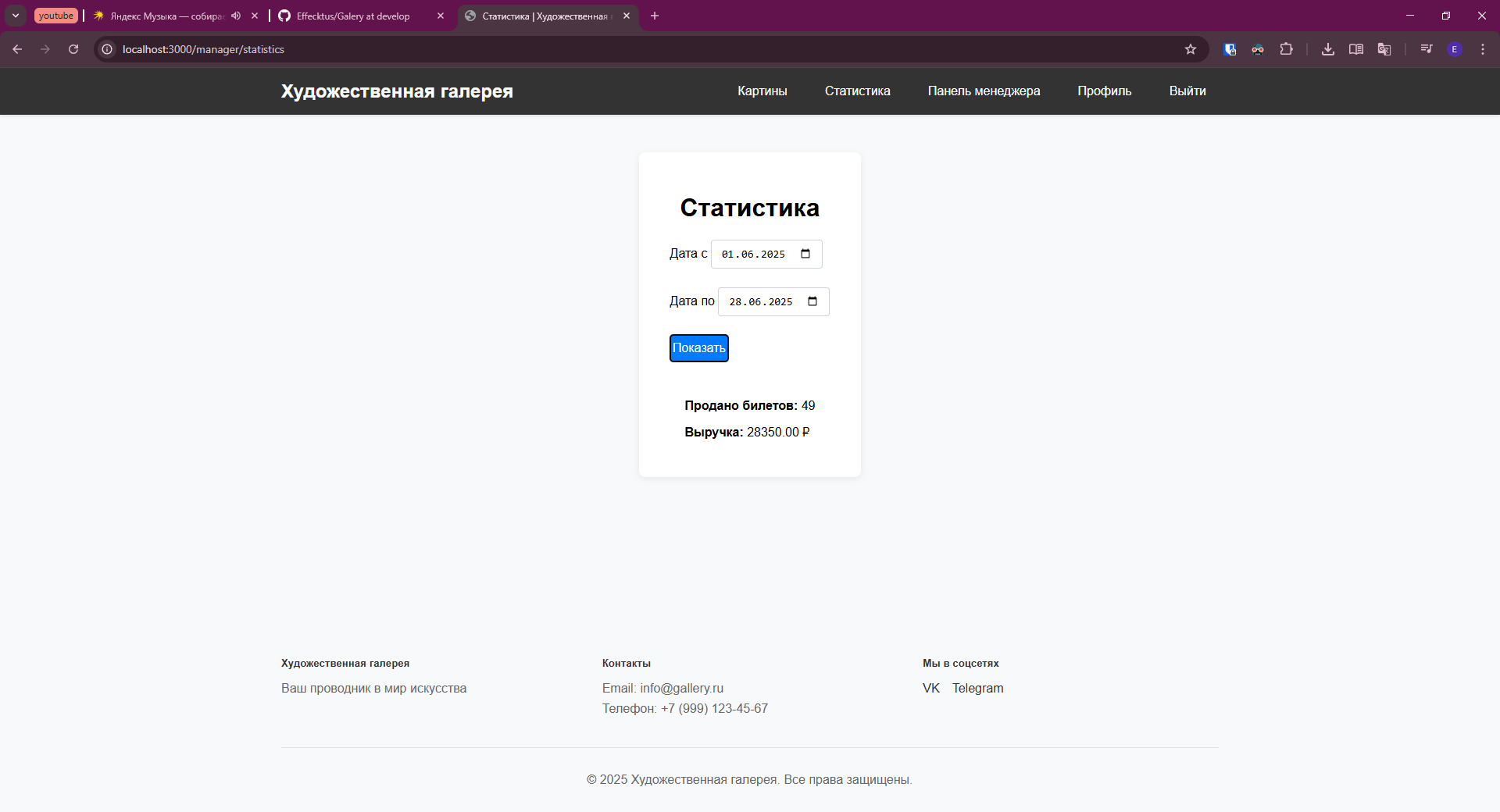


Рисунок 11. Макет страницы статистики.

# 5. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЕЙ СИСТЕМЫ

Для того, чтобы начать пользоваться системой необходимо открыть браузер и перейти на главную страницу веб-приложения. В зависимости от того вошел ли пользователь в систему внешний вид главной страницу будет отличаться: если пользователь еще не авторизирован, то будет открыта главная страница, изображенная на рисунке 12.

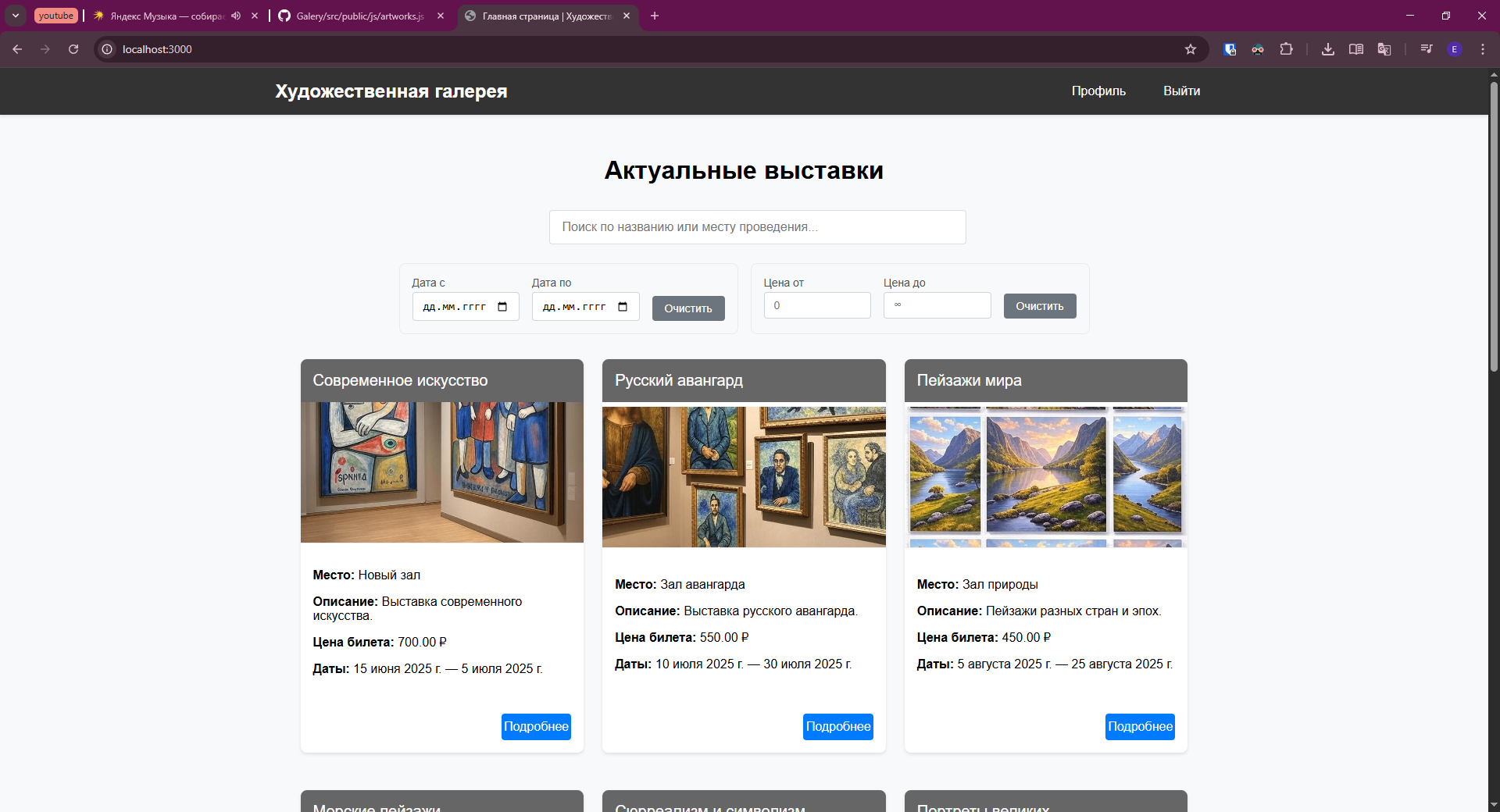


Рисунок 12. Главная страницы.

Если пользователь уже авторизирован в системе и имеет роль пользователя, то главная страница будет выглядеть следующим образом (рис. 13).

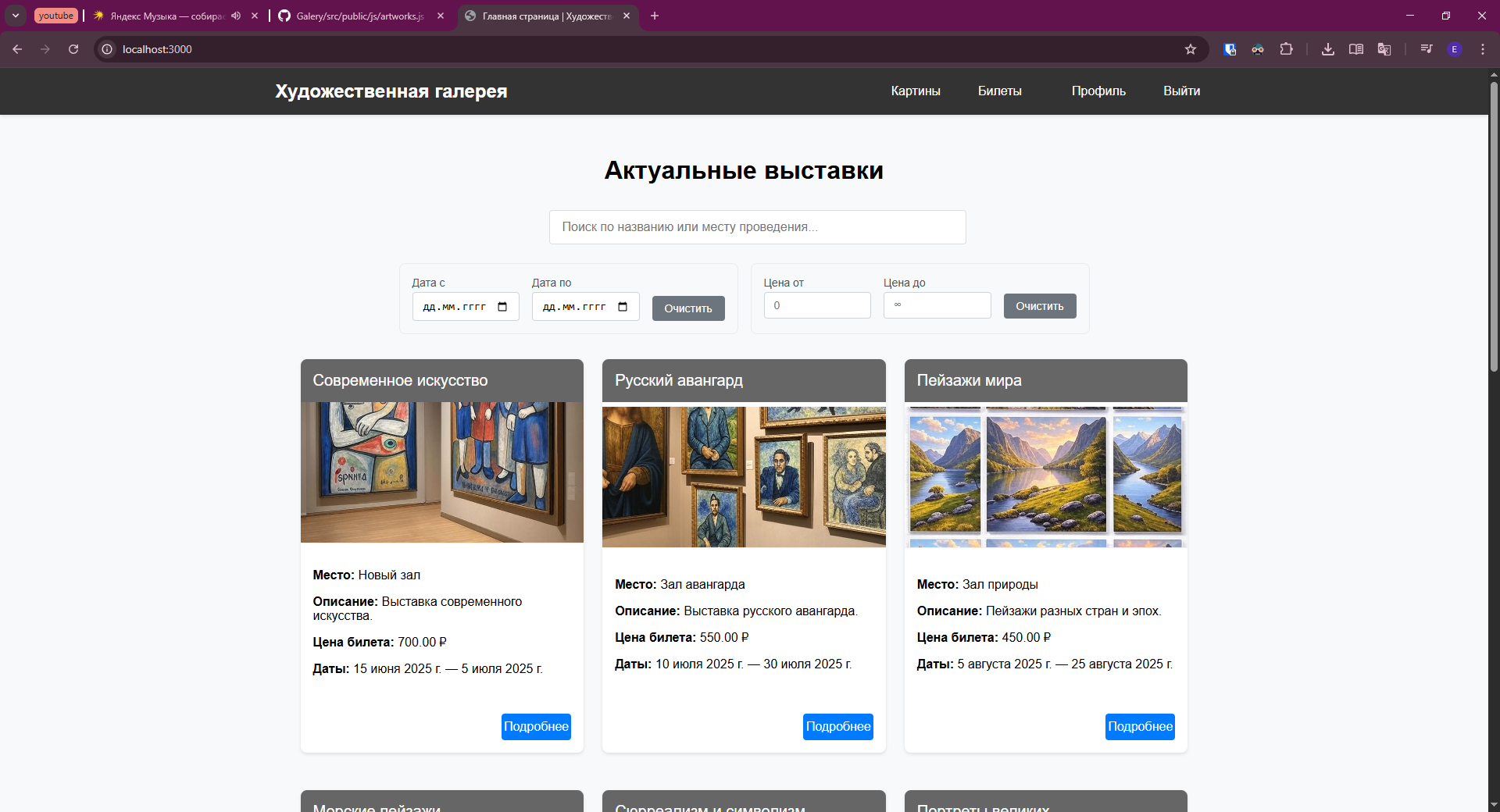


Рисунок 13. Главная страница пользователя.

Если пользователь уже авторизирован в системе и является администратором (рис. 14)

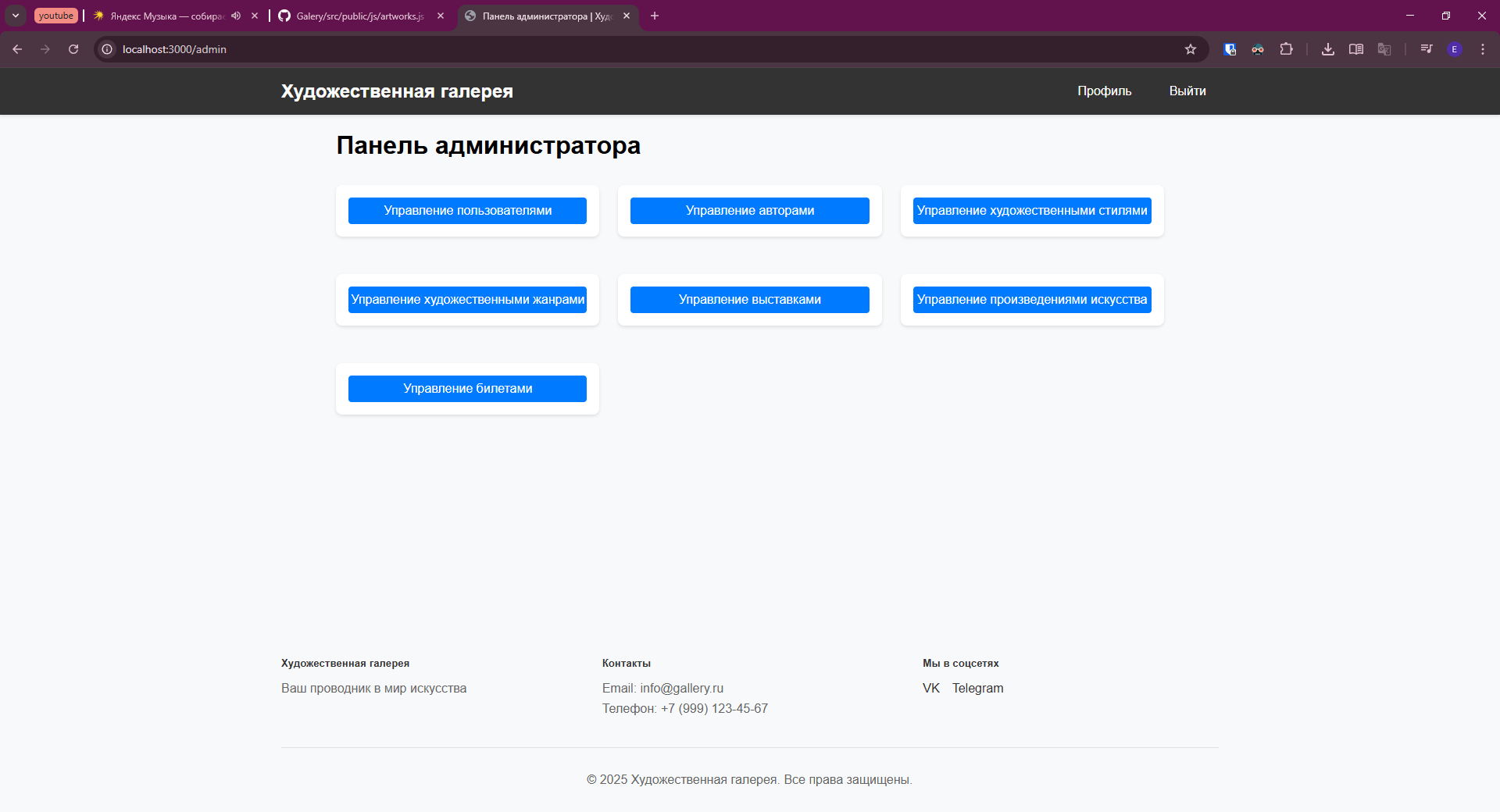


Рисунок 14. Главная страница администатора.

Если пользователь менеджер, то главная страница для него будет выглядеть следующим образом (рис. 15).

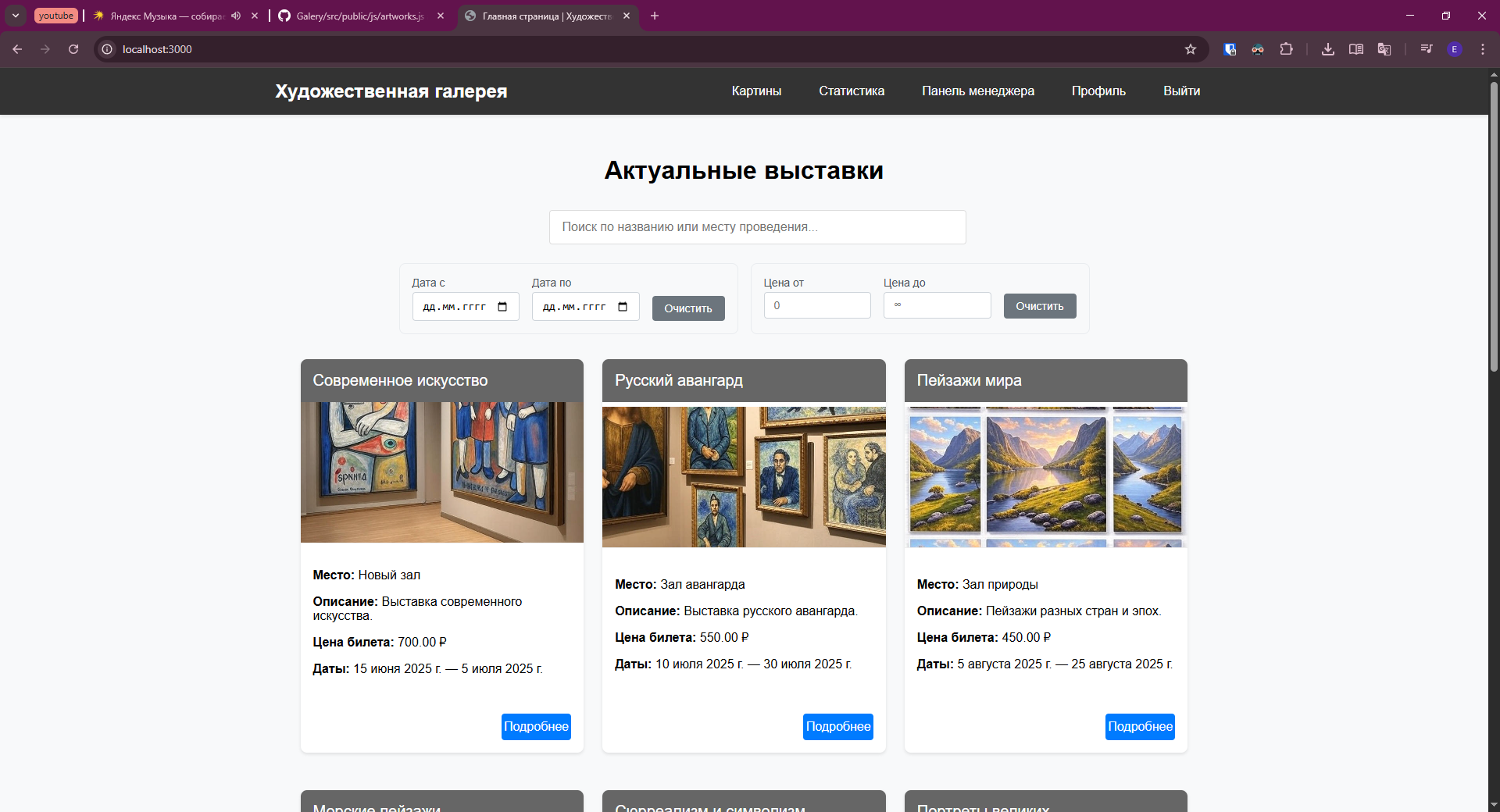


Рисунок 15. Главная страница менеджера.

Перейдем к рассмотрению страниц, доступных клиентам. Помимо главной страницы, на которую попадает пользователь после авторизации, ему также доступна страница просматривания своих билетов (рис. 16).

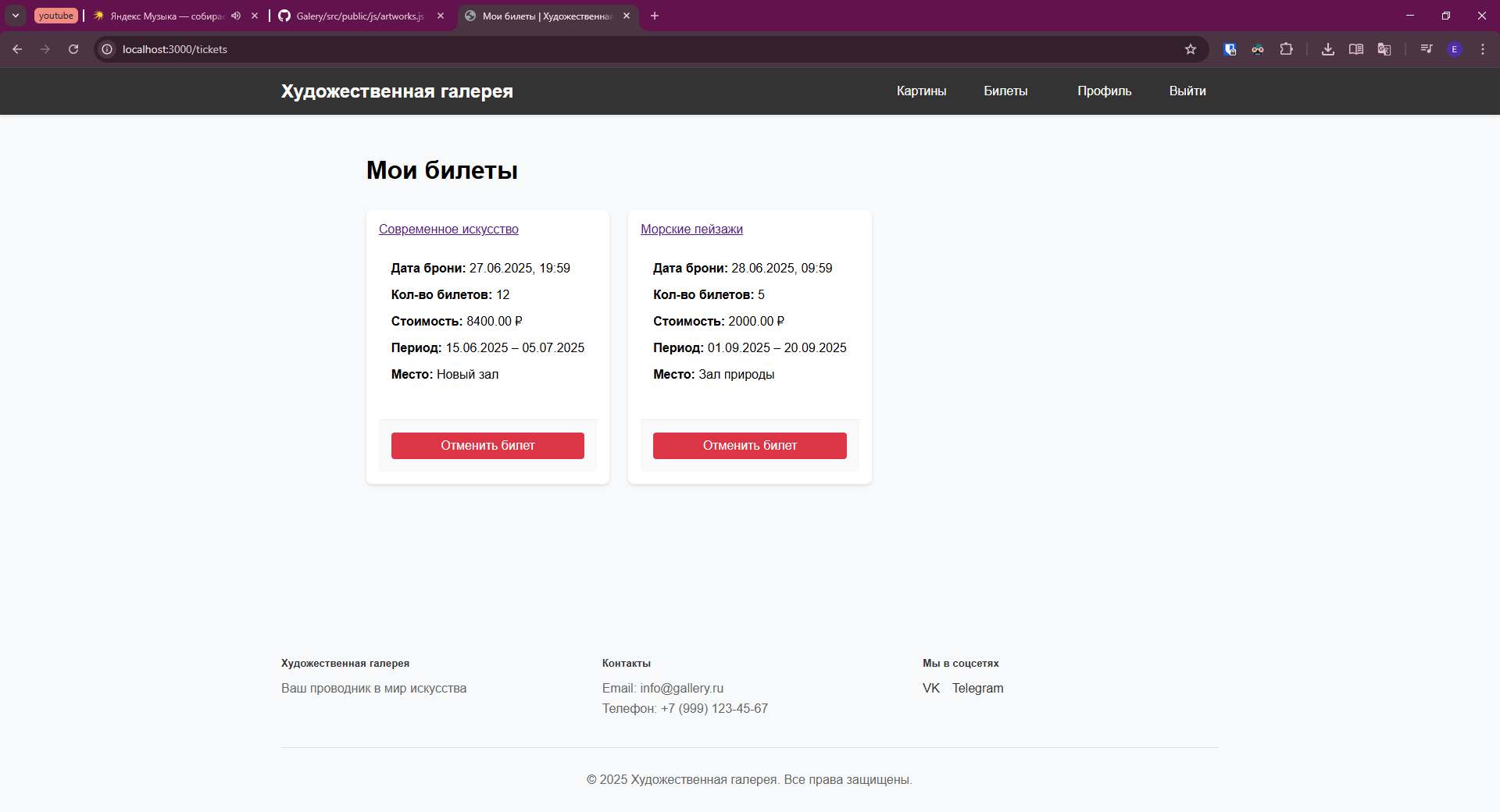


Рисунок 16. Страница «Билеты».

Рассмотрим создание бронирования (рис.17). Для этого на главной странице нужно перейти к выставке, там указать количество билетов и нажать «Купить билет».

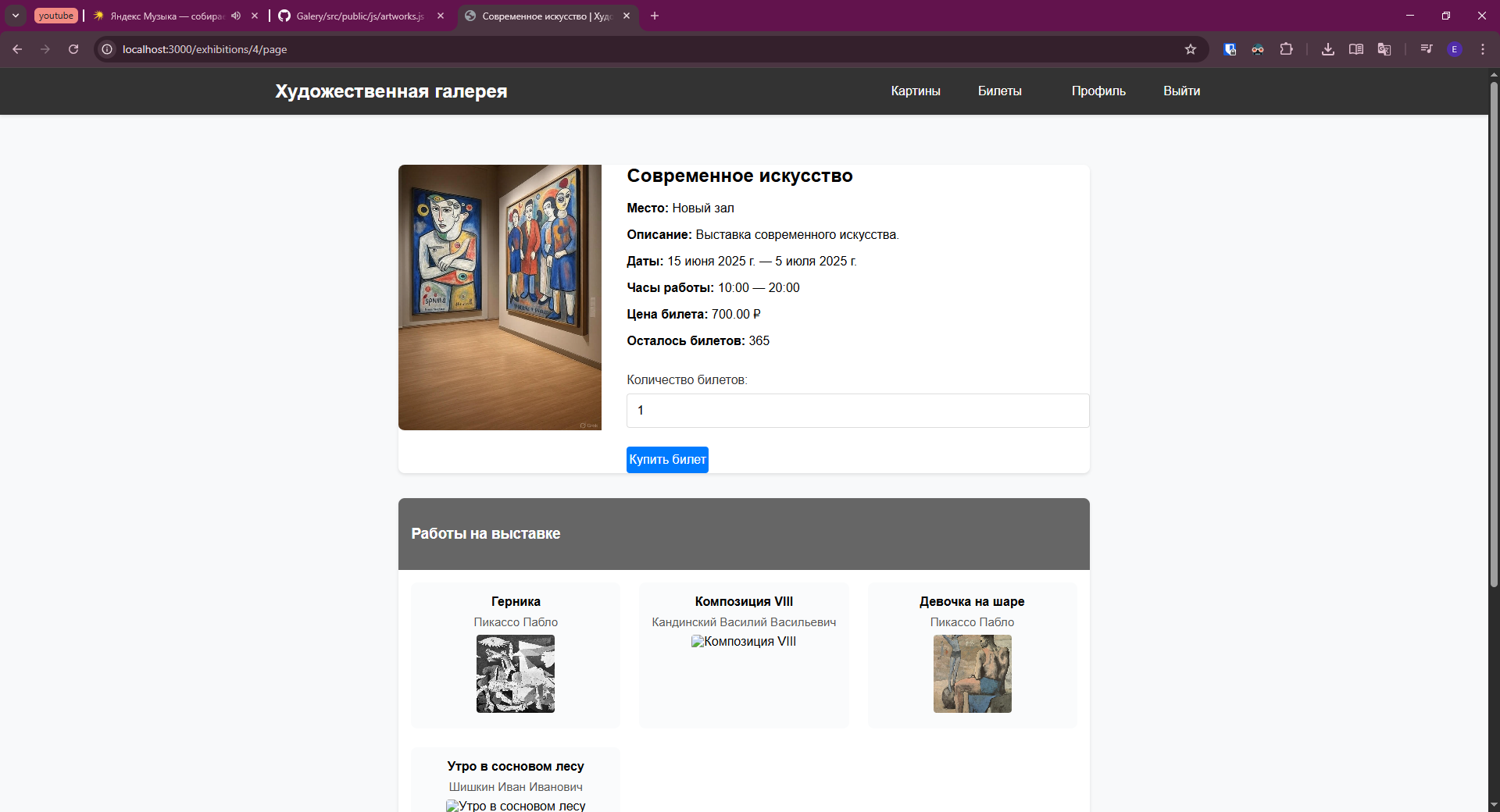


Рисунок 17. Страница «Выставка».

Рассмотрим страницу менеджера для управления выставками (рис. 18). На ней он полностью может осуществлять управления выставками.

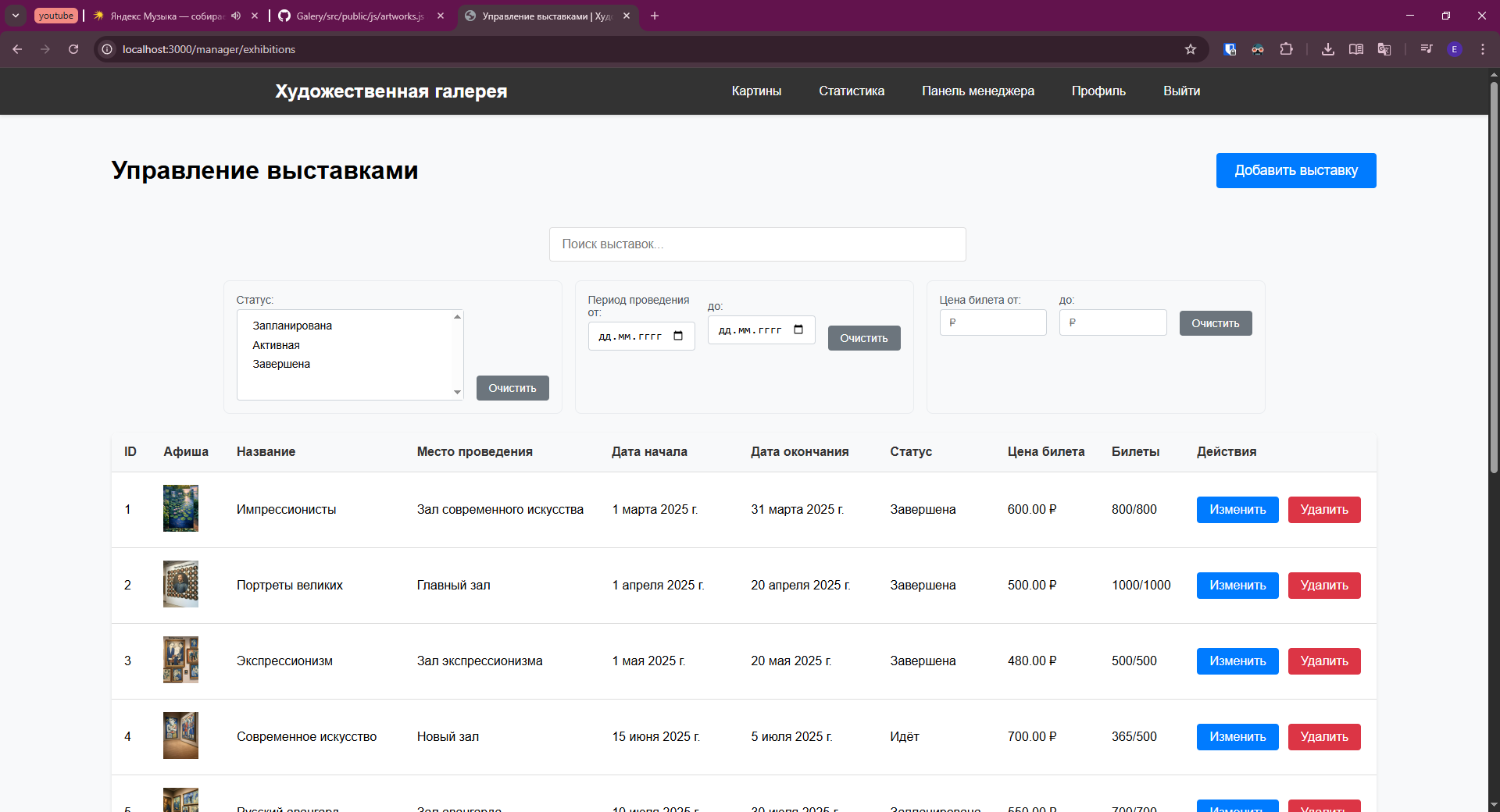


Рисунок 18. Страница «Управление выставками» менеджера и администратора.

При необходимости создать выставку, менеджер нажимает на кнопку «Добавить выставку», после чего заполняет все поля и сохраняет выставку (рис.19).

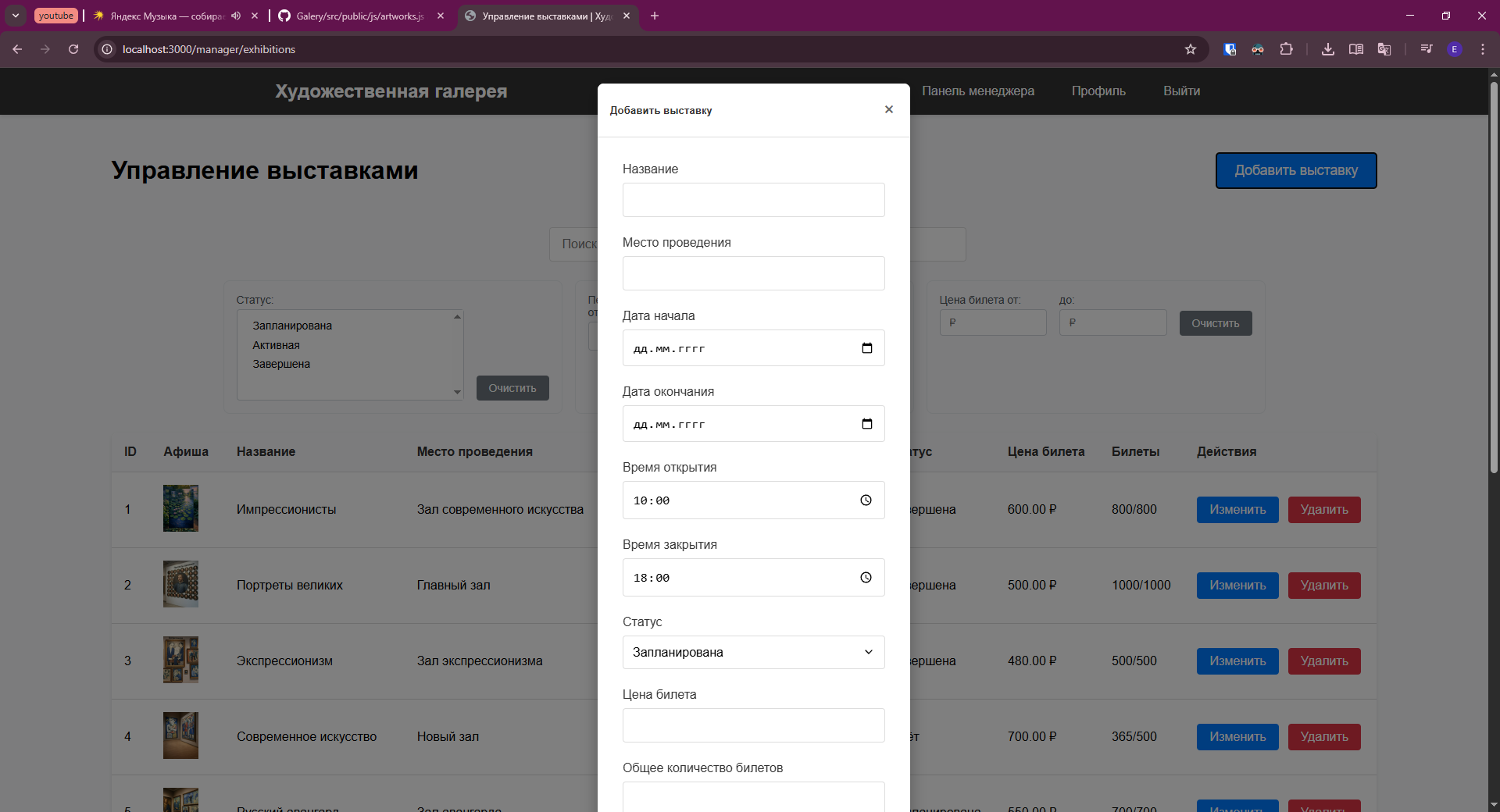


Рисунок 19. Страница «Управление выставками» добавление выставки.

Также менеджеру доступна возможность просмотра статистики продаж по датам (рис.20).



Рисунок 20. Страница «Статистика».

Перейдем к рассмотрению страницы «Панель администатора», которая доступна только администраторам (рис.21). Им доступны базовые CRUD-операции и поиск по всем таблицам. Все они сделаны по одному шаблону, который указан на рисунке 18

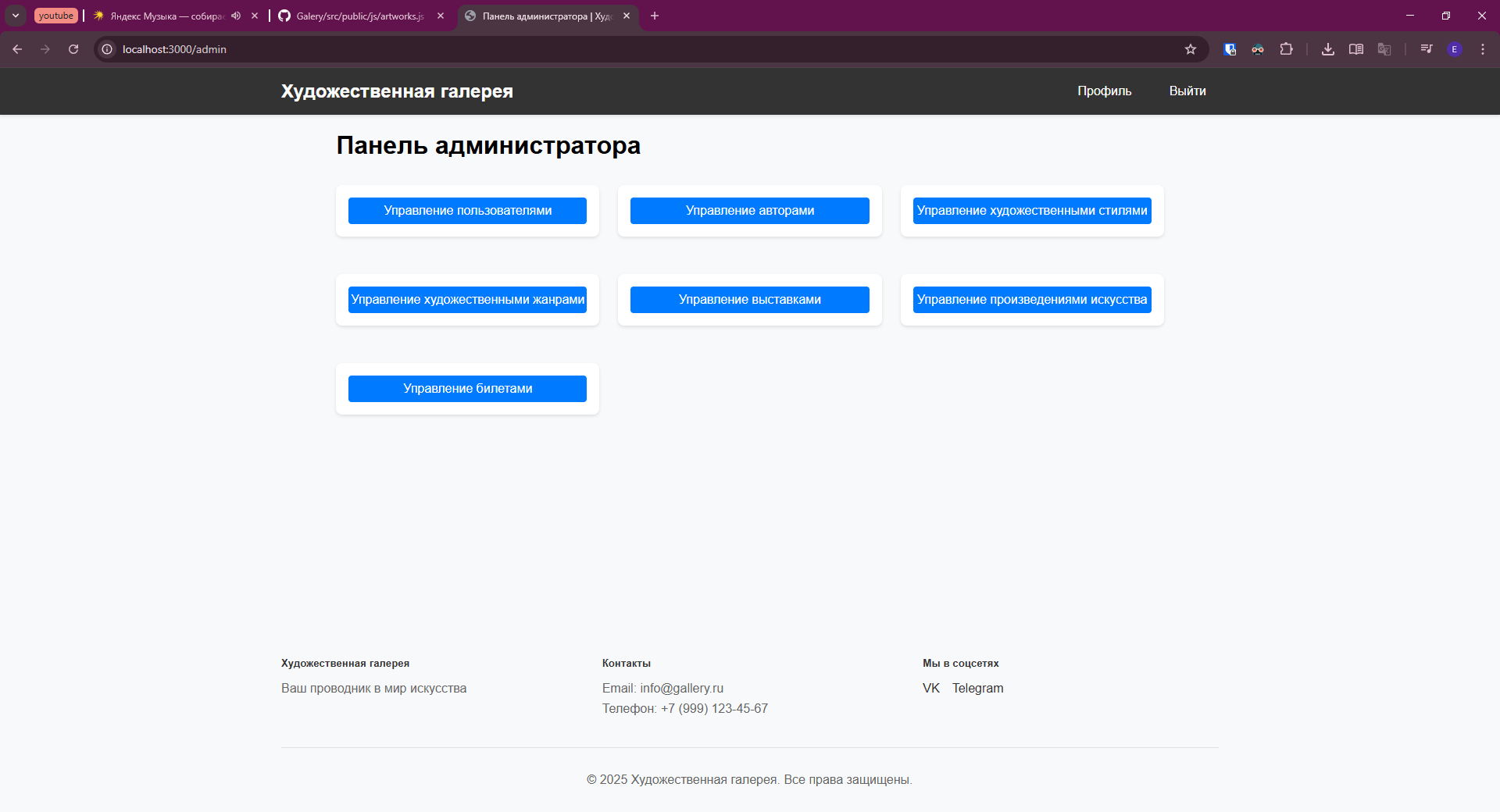


Рисунок 21. Страница «Панель администатора».

Так же имеется страница «Картины», на которой отображаются все произведения искусства, имеющиеся в галерее (рис.22).

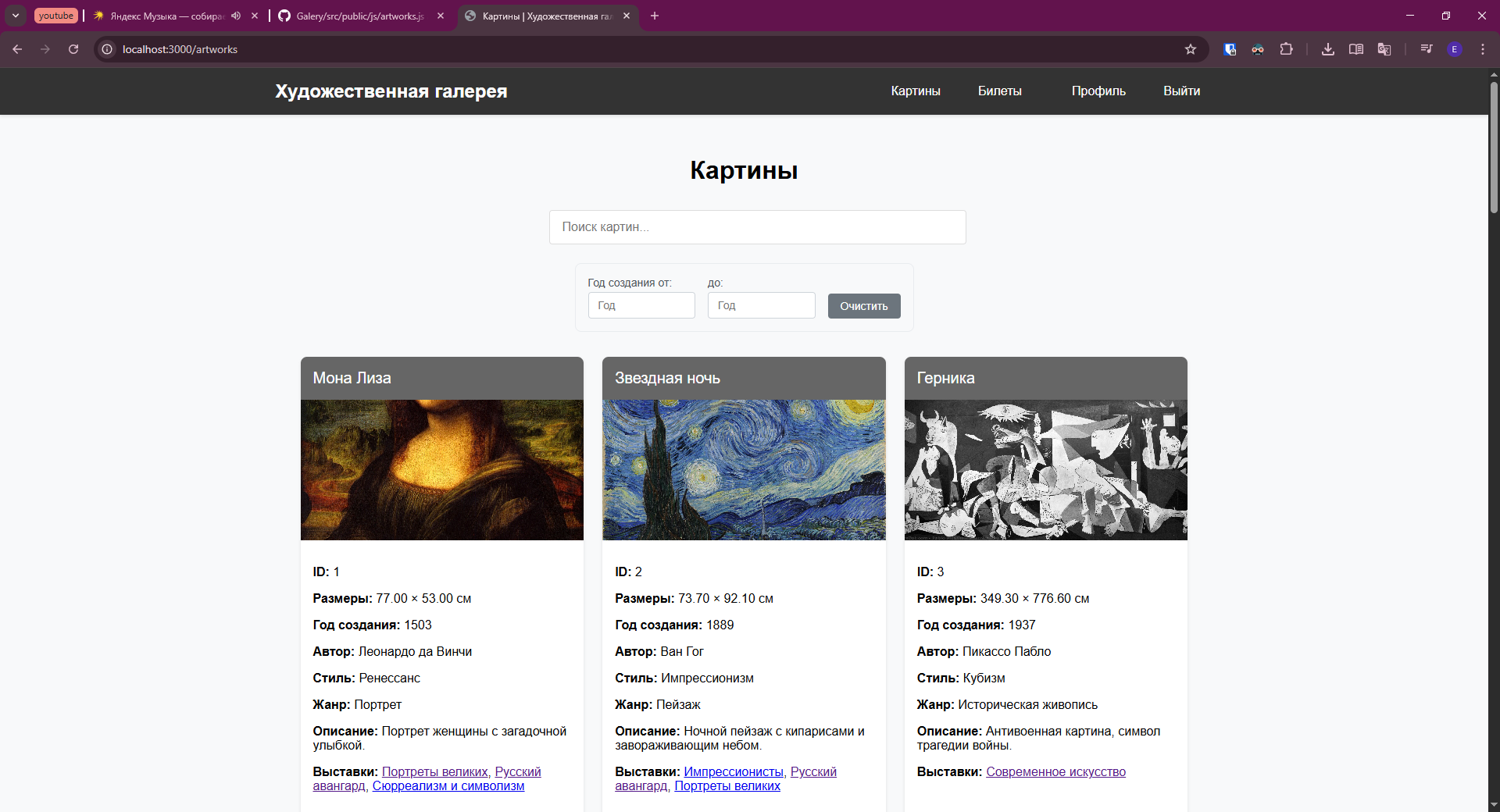


Рисунок 22. Страница «Картины».

# ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1. Волк В. К. Практическое введение в программную инженерию: учебное пособие //Текст: электронный//Электронно-библиотечная система «Лань»:[сайт].-Режим доступа: https://e. lanbook. com/book/119634.-для авториз. пользователей.
2. Браун И. Веб-разработка с применением Node и Express: Полноценное использование стека JavaScript. 2-е издание. – Питер, 2023.
3. Никсон Р. Создаем динамические веб-сайты с помощью MySQL, JavaScript, CSS и HTML5. 6-е изд. – Питер, 2023.
4. Кузнецов М. В. Самоучитель MySQL 5. – БХВ-Петербург, 2007.

ПРИЛОЖЕНИЕ 1. Файл config/database.js

// Конфигурация базы данных

module.exports = {

host: process.env.DB\_HOST || "localhost",

username: process.env.DB\_USER || "root",

password: process.env.DB\_PASSWORD || "root\_password",

database: process.env.DB\_NAME || "galerydb",

port: process.env.DB\_PORT || 3306,

dialect: "mysql",

logging: process.env.NODE\_ENV === "test" ? console.log : false,

pool: {

max: 5,

min: 0,

acquire: 30000,

idle: 10000,

},

define: {

timestamps: true,

underscored: true,

freezeTableName: true,

},

};

ПРИЛОЖЕНИЕ 2. Файл controllers/artworkController.js

const { Artwork, Author, Style, Genre, Exhibition } = require('../models');

const { Op, ValidationError } = require('sequelize');

const fs = require('fs');

const path = require('path');

// Функция для удаления файла изображения

const deleteImageFile = (imagePath) => {

if (!imagePath) return;

// Получаем полный путь к файлу

const fullPath = path.join(\_\_dirname, '../public', imagePath);

// Проверяем существование файла и удаляем его

if (fs.existsSync(fullPath)) {

fs.unlinkSync(fullPath);

}

};

// Создание нового произведения искусства

exports.createArtwork = async (req, res) => {

try {

// Проверяем наличие файла

if (!req.file) {

return res.status(400).json({

status: 'error',

message: 'Изображение обязательно'

});

}

// Проверяем существование связанных сущностей

const [author, style, genre] = await Promise.all([

Author.findByPk(req.body.author\_id),

Style.findByPk(req.body.style\_id),

Genre.findByPk(req.body.genre\_id)

]);

if (!author) {

return res.status(400).json({

status: 'error',

message: 'Указанный автор не существует'

});

}

if (!style) {

return res.status(400).json({

status: 'error',

message: 'Указанный стиль не существует'

});

}

if (!genre) {

return res.status(400).json({

status: 'error',

message: 'Указанный жанр не существует'

});

}

// Добавляем путь к изображению в данные произведения

req.body.image\_path = `/media/${req.file.filename}`;

const artwork = await Artwork.create(req.body);

res.status(201).json({

status: 'success',

data: {

artwork

}

});

} catch(err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Не удалось создать произведение: ' + err.message

});

}

};

// Получение всех произведений искусства

exports.getAllArtworks = async (req, res) => {

try {

const where = {};

const include = [

{

model: Author,

attributes: ['id', 'surname', 'first\_name', 'patronymic'],

required: false

},

{

model: Style,

attributes: ['id', 'name'],

required: false

},

{

model: Genre,

attributes: ['id', 'name'],

required: false

},

{

model: Exhibition,

attributes: ['id'],

through: { attributes: [] },

required: false

}

];

if (req.query.search) {

const searchTerms = req.query.search.toLowerCase().split(' ').filter(term => term.length > 0);

if (searchTerms.length > 0) {

where[Op.and] = searchTerms.map(term => ({

[Op.or]: [

{ title: { [Op.like]: `%${term}%` } },

{ description: { [Op.like]: `%${term}%` } },

{ creation\_year: { [Op.like]: `%${term}%` } },

{ width: { [Op.like]: `%${term}%` } },

{ height: { [Op.like]: `%${term}%` } },

{ '$Author.surname$': { [Op.like]: `%${term}%` } },

{ '$Author.first\_name$': { [Op.like]: `%${term}%` } },

{ '$Author.patronymic$': { [Op.like]: `%${term}%` } },

{ '$Style.name$': { [Op.like]: `%${term}%` } },

{ '$Genre.name$': { [Op.like]: `%${term}%` } }

]

}));

}

}

// Фильтрация по году создания

if (req.query.year\_from || req.query.year\_to) {

const yearConditions = [];

if (req.query.year\_from && req.query.year\_to) {

// Если указаны оба года, проверяем диапазон

yearConditions.push({

creation\_year: {

[Op.between]: [parseInt(req.query.year\_from), parseInt(req.query.year\_to)]

}

});

} else if (req.query.year\_from) {

// Если указан только начальный год

yearConditions.push({

creation\_year: { [Op.gte]: parseInt(req.query.year\_from) }

});

} else if (req.query.year\_to) {

// Если указан только конечный год

yearConditions.push({

creation\_year: { [Op.lte]: parseInt(req.query.year\_to) }

});

}

if (yearConditions.length > 0) {

if (where[Op.and]) {

where[Op.and].push({ [Op.and]: yearConditions });

} else {

where[Op.and] = yearConditions;

}

}

}

const artworks = await Artwork.findAll({

where,

include,

distinct: true

});

// Добавляем статистику по выставкам

const artworksWithStats = artworks.map(artwork => ({

...normalizePatronymic(artwork.toJSON()),

statistics: {

total\_exhibitions: artwork.Exhibitions ? artwork.Exhibitions.length : 0

}

}));

res.status(200).json({

status: 'success',

data: {

artworks: artworksWithStats

}

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении списка произведений: ' + err.message

});

}

};

// Получение произведения искусства по ID

exports.getArtwork = async (req, res) => {

try {

const artwork = await Artwork.findByPk(req.params.id, {

include: [

{

model: Author,

attributes: ['id', 'surname', 'first\_name', 'patronymic', 'date\_of\_birth', 'date\_of\_death']

},

{

model: Style,

attributes: ['id', 'name']

},

{

model: Genre,

attributes: ['id', 'name']

},

{

model: Exhibition,

attributes: ['id', 'title', 'start\_date', 'end\_date', 'location']

}

]

});

if (!artwork) {

return res.status(404).json({

status: 'error',

message: 'Произведение искусства с указанным ID не найдено'

});

}

res.status(200).json({

status: 'success',

data: {

artwork: normalizePatronymic(artwork)

}

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении информации о произведении: ' + err.message

});

}

};

// Обновление произведения искусства по ID

exports.updateArtwork = async (req, res) => {

try {

// Проверяем существование произведения

const artwork = await Artwork.findByPk(req.params.id);

if (!artwork) {

return res.status(404).json({

status: 'error',

message: 'Произведение искусства с указанным ID не найдено'

});

}

// Если обновляются связанные сущности, проверяем их существование

if (req.body.author\_id) {

const author = await Author.findByPk(req.body.author\_id);

if (!author) {

return res.status(400).json({

status: 'error',

message: 'Указанный автор не существует'

});

}

}

if (req.body.style\_id) {

const style = await Style.findByPk(req.body.style\_id);

if (!style) {

return res.status(400).json({

status: 'error',

message: 'Указанный стиль не существует'

});

}

}

if (req.body.genre\_id) {

const genre = await Genre.findByPk(req.body.genre\_id);

if (!genre) {

return res.status(400).json({

status: 'error',

message: 'Указанный жанр не существует'

});

}

}

// Если загружено новое изображение, удаляем старое

if (req.file) {

deleteImageFile(artwork.image\_path);

req.body.image\_path = `/media/${req.file.filename}`;

}

// Обновляем произведение

await artwork.update(req.body);

// Получаем обновленное произведение

const updatedArtwork = await Artwork.findByPk(req.params.id, {

include: [

{

model: Author,

attributes: ['id', 'surname', 'first\_name', 'patronymic']

},

{

model: Style,

attributes: ['id', 'name']

},

{

model: Genre,

attributes: ['id', 'name']

},

{

model: Exhibition,

attributes: ['id', 'title', 'start\_date', 'end\_date']

}

]

});

res.status(200).json({

status: 'success',

data: {

artwork: normalizePatronymic(updatedArtwork)

}

});

} catch(err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Ошибка при обновлении произведения: ' + err.message

});

}

};

exports.getArtworkExhibitions = async (req, res) => {

try {

const artworkId = req.params.id;

const exhibitions = await Exhibition.findAll({

include: [{ model: Artwork, attributes: [], through: { attributes: [] }, where: { id: artworkId } }],

attributes: ['id', 'title', 'start\_date', 'end\_date', 'location']

});

res.status(200).json({ status: 'success', data: { exhibitions } });

} catch (err) {

console.error(err);

res.status(500).json({ status: 'error', message: 'Ошибка при получении выставок для произведения: ' + err.message });

}

};

exports.renderArtworksPage = async (req, res, next) => {

try {

// Берём все картины вместе с автором, стилем и жанром

const artworks = await Artwork.findAll({

include: [

{ model: Author, attributes: ['surname','first\_name','patronymic'] },

{ model: Style, attributes: ['name'] },

{ model: Genre, attributes: ['name'] }

],

order: [['id', 'ASC']]

});

res.render('artworks', {

title: 'Картины',

user: res.locals.user || null,

artworks

});

} catch (err) {

next(err);

}

};

// Удаление произведения искусства по ID

exports.deleteArtwork = async (req, res) => {

try {

const artwork = await Artwork.findByPk(req.params.id, {

include: [{

model: Exhibition,

attributes: ['id', 'title'],

through: { attributes: [] }

}]

});

if (!artwork) {

return res.status(404).json({

status: 'error',

message: 'Произведение искусства с указанным ID не найдено'

});

}

// Проверяем, участвует ли произведение в каких-либо выставках

if (artwork.Exhibitions && artwork.Exhibitions.length > 0) {

return res.status(400).json({

status: 'error',

message: 'Невозможно удалить произведение, так как оно участвует в выставках',

data: {

exhibitions: artwork.Exhibitions.map(exhibition => ({

id: exhibition.id,

title: exhibition.title

}))

}

});

}

// Сохраняем путь к изображению перед удалением произведения

const imagePath = artwork.image\_path;

// Удаляем произведение

await artwork.destroy();

// Удаляем файл изображения

deleteImageFile(imagePath);

res.status(200).json({

status: 'success',

message: 'Произведение искусства успешно удалено',

data: null

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при удалении произведения: ' + err.message

});

}

};

// Утилита для замены null в patronymic на пустую строку

const normalizePatronymic = (obj, seen = new Set()) => {

if (!obj || typeof obj !== 'object') return obj;

if (seen.has(obj)) return obj; // предотвращаем цикл

seen.add(obj);

if (Array.isArray(obj)) {

return obj.map(item => normalizePatronymic(item, seen));

}

if ('patronymic' in obj && obj.patronymic === null) {

obj.patronymic = '';

}

for (const key in obj) {

if (typeof obj[key] === 'object' && obj[key] !== null) {

obj[key] = normalizePatronymic(obj[key], seen);

}

}

return obj;

};  
module.exports = sequelize;

ПРИЛОЖЕНИЕ 3. Файл controllers/authController.js

*const { User, Ticket, Exhibition } = require('../models');*

*const jwt = require('jsonwebtoken');*

*// Функция для фильтрации чувствительных данных*

*const filterSensitiveData = (user) => {*

*const { password, ...userWithoutSensitiveData } = user.toJSON();*

*return userWithoutSensitiveData;*

*};*

*// Функция для создания access токена*

*const signToken = (userId) => {*

*return jwt.sign(*

*{ id: userId },*

*process.env.JWT\_SECRET,*

*{ expiresIn: process.env.JWT\_EXPIRES\_IN || '1h' }*

*);*

*};*

*// Функция для создания refresh токена*

*const signRefreshToken = (userId) => {*

*return jwt.sign(*

*{ id: userId },*

*process.env.JWT\_REFRESH\_SECRET || process.env.JWT\_SECRET,*

*{ expiresIn: process.env.JWT\_REFRESH\_EXPIRES\_IN || '7d' }*

*);*

*};*

*// Функция для отправки ответа с токенами*

*const createSendToken = (user, statusCode, res, req) => {*

*const token = signToken(user.id);*

*const refreshToken = signRefreshToken(user.id);*

*console.log('Creating token for user:', user.email, 'Role:', user.role);*

*// Устанавливаем куки*

*res.cookie('token', token, {*

*expires: new Date(Date.now() + 24 \* 60 \* 60 \* 1000), // 24 часа*

*httpOnly: true,*

*secure: process.env.NODE\_ENV === 'production'*

*});*

*// Для API запросов отправляем JSON*

*if (req && (req.xhr || req.headers.accept?.includes('application/json'))) {*

*return res.status(statusCode).json({*

*status: 'success',*

*data: {*

*user: filterSensitiveData(user),*

*token,*

*refreshToken*

*}*

*});*

*}*

*// Для веб-запросов делаем редирект в зависимости от роли*

*console.log('Redirecting user with role:', user.role);*

*if (user.role === 'admin') {*

*res.redirect('/admin');*

*} else {*

*res.redirect('/');*

*}*

*};*

*// Регистрация нового пользователя*

*exports.register = async (req, res) => {*

*try {*

*const { email, password, password\_confirm, surname, first\_name, patronymic } = req.body;*

*// Проверяем совпадение паролей*

*if (password !== password\_confirm) {*

*if (req.xhr || req.headers.accept.includes('application/json')) {*

*return res.status(400).json({*

*status: 'error',*

*message: 'Пароли не совпадают'*

*});*

*}*

*return res.render('auth/register', {*

*title: 'Регистрация',*

*error: 'Пароли не совпадают',*

*user: null*

*});*

*}*

*const existingUser = await User.findOne({ where: { email } });*

*if (existingUser) {*

*if (req.xhr || req.headers.accept.includes('application/json')) {*

*return res.status(400).json({*

*status: 'error',*

*message: 'Пользователь с таким email уже существует'*

*});*

*}*

*return res.render('auth/register', {*

*title: 'Регистрация',*

*error: 'Пользователь с таким email уже существует',*

*user: null*

*});*

*}*

*const newUser = await User.create({*

*email,*

*password,*

*surname,*

*first\_name,*

*patronymic,*

*role: 'user'*

*});*

*createSendToken(newUser, 201, res, req);*

*} catch (err) {*

*console.error('Registration error:', err);*

*if (req.xhr || req.headers.accept.includes('application/json')) {*

*return res.status(500).json({*

*status: 'error',*

*message: 'Внутренняя ошибка сервера'*

*});*

*}*

*res.render('auth/register', {*

*title: 'Регистрация',*

*error: 'Произошла ошибка при регистрации',*

*user: null*

*});*

*}*

*};*

*// Вход пользователя*

*exports.login = async (req, res) => {*

*try {*

*const { email, password } = req.body;*

*if (!email || !password) {*

*if (req.xhr || req.headers.accept?.includes('application/json')) {*

*return res.status(400).json({*

*status: 'error',*

*message: 'Пожалуйста, укажите email и пароль'*

*});*

*}*

*return res.render('auth/login', {*

*title: 'Вход',*

*error: 'Пожалуйста, укажите email и пароль',*

*user: null*

*});*

*}*

*const user = await User.findOne({ where: { email } });*

*if (!user || !(await user.checkPassword(password))) {*

*if (req.xhr || req.headers.accept?.includes('application/json')) {*

*return res.status(401).json({*

*status: 'error',*

*message: 'Неверный email или пароль'*

*});*

*}*

*return res.render('auth/login', {*

*title: 'Вход',*

*error: 'Неверный email или пароль',*

*user: null*

*});*

*}*

*createSendToken(user, 200, res, req);*

*} catch (err) {*

*console.error('Login error:', err);*

*if (req.xhr || req.headers.accept?.includes('application/json')) {*

*return res.status(500).json({*

*status: 'error',*

*message: 'Внутренняя ошибка сервера'*

*});*

*}*

*res.render('auth/login', {*

*title: 'Вход',*

*error: 'Произошла ошибка при входе',*

*user: null*

*});*

*}*

*};*

*// Обновление токена*

*exports.refreshToken = async (req, res) => {*

*try {*

*const { refreshToken } = req.body;*

*if (!refreshToken) {*

*return res.status(400).json({*

*status: 'error',*

*message: 'Требуется refresh token'*

*});*

*}*

*const decoded = jwt.verify(refreshToken, process.env.JWT\_REFRESH\_SECRET);*

*const user = await User.findByPk(decoded.id);*

*if (!user) {*

*return res.status(401).json({*

*status: 'error',*

*message: 'Пользователь не найден'*

*});*

*}*

*createSendToken(user, 200, res, req);*

*} catch (err) {*

*if (err.name === 'JsonWebTokenError') {*

*return res.status(401).json({*

*status: 'error',*

*message: 'Недействительный токен'*

*});*

*}*

*if (err.name === 'TokenExpiredError') {*

*return res.status(401).json({*

*status: 'error',*

*message: 'Срок действия токена истек'*

*});*

*}*

*console.error('Refresh token error:', err);*

*res.status(500).json({*

*status: 'error',*

*message: 'Внутренняя ошибка сервера'*

*});*

*}*

*};*

*// Выход пользователя*

*exports.logout = (req, res) => {*

*res.cookie('token', 'loggedout', {*

*expires: new Date(Date.now() + 10 \* 1000),*

*httpOnly: true*

*});*

*if (req.xhr || req.headers.accept.includes('application/json')) {*

*return res.status(200).json({*

*status: 'success',*

*message: 'Выход выполнен успешно'*

*});*

*}*

*res.redirect('/auth/login');*

*};*

*// Изменение пароля*

*exports.changePassword = async (req, res) => {*

*try {*

*const { currentPassword, newPassword } = req.body;*

*if (!currentPassword || !newPassword) {*

*return res.status(400).json({*

*status: 'error',*

*message: 'Текущий и новый пароли обязательны'*

*});*

*}*

*const user = await User.findByPk(req.user.id);*

*if (!user || !(await user.checkPassword(currentPassword))) {*

*return res.status(401).json({*

*status: 'error',*

*message: 'Неверный текущий пароль'*

*});*

*}*

*user.password = newPassword;*

*await user.save();*

*createSendToken(user, 200, res, req);*

*} catch (err) {*

*if (err.message.includes('Пароль')) {*

*return res.status(400).json({*

*status: 'error',*

*message: err.message*

*});*

*}*

*console.error('Change password error:', err);*

*res.status(500).json({*

*status: 'error',*

*message: 'Внутренняя ошибка сервера'*

*});*

*}*

*};*

*// Получение данных текущего пользователя*

*exports.getMe = async (req, res) => {*

*try {*

*const user = await User.findByPk(req.user.id, {*

*include: [{*

*model: Ticket,*

*attributes: ['id', 'exhibition\_id', 'quantity', 'booking\_date', 'total\_price'],*

*include: [{*

*model: Exhibition,*

*attributes: ['id', 'title', 'start\_date', 'end\_date']*

*}]*

*}]*

*});*

*if (!user) {*

*return res.status(404).json({*

*status: 'error',*

*message: 'Пользователь не найден'*

*});*

*}*

*const userWithStats = {*

*...filterSensitiveData(user),*

*statistics: {*

*total\_tickets: user.Tickets.reduce((sum, ticket) => sum + ticket.quantity, 0)*

*}*

*};*

*res.status(200).json({*

*status: 'success',*

*data: {*

*user: userWithStats*

*}*

*});*

*} catch (err) {*

*console.error('Get me error:', err);*

*res.status(500).json({*

*status: 'error',*

*message: 'Внутренняя ошибка сервера'*

*});*

*}*

*}*;

ПРИЛОЖЕНИЕ 4. Файл controllers/authorController.js

const { Author, Artwork, Style, Genre } = require('../models');

const { Op, ValidationError } = require('sequelize');

// Создание нового автора

exports.createAuthor = async (req, res) => {

try {

const newAuthor = await Author.create(req.body);

res.status(201).json({

status: 'success',

message: 'Автор успешно создан',

data: { author: normalizePatronymic(newAuthor) }

});

} catch(err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Не удалось создать автора: ' + err.message

});

}

};

// Получение всех авторов

exports.getAllAuthors = async (req, res) => {

try {

const where = {};

if (req.query.name) {

const searchTerms = req.query.name.toLowerCase().split(' ').filter(term => term.length > 0);

if (searchTerms.length > 0) {

where[Op.and] = searchTerms.map(term => ({

[Op.or]: [

{ surname: { [Op.like]: `%${term}%` } },

{ first\_name: { [Op.like]: `%${term}%` } },

{ patronymic: { [Op.like]: `%${term}%` } }

]

}));

}

}

// Фильтрация по периоду жизни

if (req.query.date\_from || req.query.date\_to) {

const dateConditions = [];

if (req.query.date\_from && req.query.date\_to) {

// Если указаны обе даты, проверяем пересечение периодов

// Автор попадает в фильтр, если его период жизни пересекается с выбранным диапазоном

// Это означает: дата рождения <= date\_to И дата смерти >= date\_from

dateConditions.push({

[Op.and]: [

{ date\_of\_birth: { [Op.lte]: req.query.date\_to } },

{ date\_of\_death: { [Op.gte]: req.query.date\_from } }

]

});

} else if (req.query.date\_from) {

// Если указана только начальная дата, автор должен был жить после неё

dateConditions.push({

date\_of\_death: { [Op.gte]: req.query.date\_from }

});

} else if (req.query.date\_to) {

// Если указана только конечная дата, автор должен был родиться до неё

dateConditions.push({

date\_of\_birth: { [Op.lte]: req.query.date\_to }

});

}

if (dateConditions.length > 0) {

if (where[Op.and]) {

where[Op.and].push({ [Op.and]: dateConditions });

} else {

where[Op.and] = dateConditions;

}

}

}

const authors = await Author.findAll({

where,

include: [{

model: Artwork,

attributes: ['id', 'title'],

required: false

}],

distinct: true

});

// Добавляем статистику по произведениям

const authorsWithStats = authors.map(author => ({

...normalizePatronymic(author.toJSON()),

statistics: {

total\_artworks: author.Artworks ? author.Artworks.length : 0

}

}));

res.status(200).json({

status: 'success',

data: {

authors: authorsWithStats

}

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении списка авторов: ' + err.message

});

}

};

// Получение автора по ID

exports.getAuthor = async (req, res) => {

try {

const author = await Author.findByPk(req.params.id, {

include: [{

model: Artwork,

attributes: ['id', 'title', 'creation\_year', 'image\_path', 'style\_id', 'genre\_id'],

include: [

{

model: Style,

attributes: ['name']

},

{

model: Genre,

attributes: ['name']

}

]

}]

});

if (!author) {

return res.status(404).json({

status: 'error',

message: 'Автор с указанным ID не найден'

});

}

res.status(200).json({

status: 'success',

data: {

author: normalizePatronymic(author)

}

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении информации об авторе: ' + err.message

});

}

};

// Обновление автора по ID

exports.updateAuthor = async (req, res) => {

try {

const author = await Author.findByPk(req.params.id);

if (!author) {

return res.status(404).json({

status: 'error',

message: 'Автор с указанным ID не найден'

});

}

// Проверяем, не пытаемся ли мы изменить дату смерти на дату раньше даты рождения

if (req.body.date\_of\_death && author.date\_of\_birth) {

const newDeathDate = new Date(req.body.date\_of\_death);

const birthDate = new Date(author.date\_of\_birth);

if (newDeathDate <= birthDate) {

return res.status(400).json({

status: 'error',

message: 'Дата смерти должна быть позже даты рождения'

});

}

}

// Обновляем автора

await author.update(req.body);

// Получаем обновленного автора

const updatedAuthor = await Author.findByPk(req.params.id, {

include: [{

model: Artwork,

attributes: ['id', 'title']

}]

});

res.status(200).json({

status: 'success',

data: {

author: normalizePatronymic(updatedAuthor)

}

});

} catch(err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Ошибка при обновлении автора: ' + err.message

});

}

};

// Удаление автора по ID

exports.deleteAuthor = async (req, res) => {

try {

const author = await Author.findByPk(req.params.id, {

include: [{

model: Artwork,

attributes: ['id']

}]

});

if (!author) {

return res.status(404).json({

status: 'error',

message: 'Автор с указанным ID не найден'

});

}

// Проверяем, есть ли у автора произведения

if (author.Artworks && author.Artworks.length > 0) {

return res.status(400).json({

status: 'error',

message: 'Невозможно удалить автора, у которого есть произведения',

data: {

artworksCount: author.Artworks.length

}

});

}

await author.destroy();

res.status(200).json({

status: 'success',

message: 'Автор успешно удален',

data: null

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при удалении автора: ' + err.message

});

}

};

// Утилита для замены null в patronymic на пустую строку

const normalizePatronymic = (obj, seen = new WeakSet()) => {

if (obj && typeof obj === 'object') {

if (seen.has(obj)) return obj; // остановка на уже встреченном

seen.add(obj);

for (const key in obj) {

if (typeof obj[key] === 'object' && obj[key] !== null) {

obj[key] = normalizePatronymic(obj[key], seen);

}

}

}

return obj;

};

ПРИЛОЖЕНИЕ 5. Файл controllers/exhibitionController.js

const { Exhibition, Artwork, Ticket, Author, Style, Genre } = require('../models');

const { Op, ValidationError, Sequelize } = require('sequelize');

// Создание новой выставки

exports.createExhibition = async (req, res) => {

try {

const data = { ...req.body };

data.remaining\_tickets = data.total\_tickets;

if (req.file) data.poster\_path = '/media/' + req.file.filename;

const exhibition = await Exhibition.create(data);

let artworkIds = req.body.artwork\_ids || req.body['artwork\_ids[]'];

if (artworkIds) {

if (typeof artworkIds === 'string') {

try {

artworkIds = JSON.parse(artworkIds);

} catch {

artworkIds = artworkIds.split(',').map(id => id.trim());

}

}

if (Array.isArray(artworkIds)) {

// приводим к числам

const ids = artworkIds.map(i => parseInt(i, 10)).filter(i => !isNaN(i));

await exhibition.setArtworks(ids);

}

}

// возвращаем только что созданную с привязанными картинами

const exhibitionData = await Exhibition.findByPk(exhibition.id, {

include: [{

model: Artwork,

attributes: ['id', 'title', 'creation\_year', 'image\_path']

}]

});

return res.status(201).json({

status: 'success',

data: { exhibition: exhibitionData }

});

} catch(err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(400).json({

status: 'error',

message: 'Не удалось создать выставку: ' + err.message

});

}

};

// Получение всех выставок с фильтрацией и пагинацией

exports.getAllExhibitions = async (req, res) => {

try {

const where = {};

// Фильтрация по статусу (поддержка множественного выбора)

if (req.query.status) {

const statuses = Array.isArray(req.query.status) ? req.query.status : [req.query.status];

const validStatuses = statuses.filter(status => ['upcoming', 'active', 'completed'].includes(status));

if (validStatuses.length > 0) {

where.status = { [Op.in]: validStatuses };

}

}

// Фильтрация по месту проведения

if (req.query.location) {

where.location = { [Op.like]: `%${req.query.location}%` };

}

// Фильтрация по названию

if (req.query.title) {

where.title = { [Op.like]: `%${req.query.title}%` };

}

// Фильтрация по периоду проведения

if (req.query.start\_date || req.query.end\_date) {

const dateConditions = [];

if (req.query.start\_date && req.query.end\_date) {

// Если указаны обе даты, проверяем пересечение периодов

// Выставка попадает в фильтр, если её период пересекается с выбранным диапазоном

dateConditions.push({

[Op.and]: [

{ start\_date: { [Op.lte]: req.query.end\_date } },

{ end\_date: { [Op.gte]: req.query.start\_date } }

]

});

} else if (req.query.start\_date) {

// Если указана только начальная дата, выставка должна начаться после неё

dateConditions.push({

start\_date: { [Op.gte]: req.query.start\_date }

});

} else if (req.query.end\_date) {

// Если указана только конечная дата, выставка должна закончиться до неё

dateConditions.push({

end\_date: { [Op.lte]: req.query.end\_date }

});

}

if (dateConditions.length > 0) {

if (where[Op.and]) {

where[Op.and].push({ [Op.and]: dateConditions });

} else {

where[Op.and] = dateConditions;

}

}

}

// Фильтрация по цене билета

if (req.query.min\_price || req.query.max\_price) {

const priceConditions = [];

if (req.query.min\_price) {

const minPrice = parseFloat(req.query.min\_price);

if (!isNaN(minPrice) && minPrice >= 0) {

priceConditions.push({ ticket\_price: { [Op.gte]: minPrice } });

}

}

if (req.query.max\_price) {

const maxPrice = parseFloat(req.query.max\_price);

if (!isNaN(maxPrice) && maxPrice >= 0) {

priceConditions.push({ ticket\_price: { [Op.lte]: maxPrice } });

}

}

if (priceConditions.length > 0) {

if (where[Op.and]) {

where[Op.and].push({ [Op.and]: priceConditions });

} else {

where[Op.and] = priceConditions;

}

}

}

// Многословный поиск по всем основным полям

if (req.query.search) {

const searchWords = req.query.search.trim().split(/\s+/);

const searchConditions = searchWords.map(word => ({

[Op.or]: [

{ title: { [Op.like]: `%${word}%` } },

{ location: { [Op.like]: `%${word}%` } },

{ status: { [Op.like]: `%${word}%` } },

{ description: { [Op.like]: `%${word}%` } }

]

}));

if (where[Op.and]) {

where[Op.and].push(...searchConditions);

} else {

where[Op.and] = searchConditions;

}

}

const exhibitions = await Exhibition.findAll({

where,

order: [['start\_date', 'ASC']],

include: [

{

model: Artwork,

attributes: ['id', 'title', 'creation\_year', 'image\_path'],

include: [

{

model: Author,

attributes: ['id', 'surname', 'first\_name', 'patronymic']

}

]

},

{

model: Ticket,

attributes: ['id', 'quantity']

}

]

});

// Нормализуем patronymic у всех авторов

exhibitions.forEach(exh => {

if (exh.Artworks) {

exh.Artworks.forEach(art => {

if (art.Author) normalizePatronymic(art.Author);

});

}

});

res.status(200).json({

status: 'success',

data: {

exhibitions

}

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении списка выставок: ' + err.message

});

}

};

// Получение выставки по ID

exports.getExhibition = async (req, res) => {

try {

const exhibition = await Exhibition.findByPk(req.params.id, {

include: [

{

model: Artwork,

attributes: ['id', 'title', 'creation\_year', 'image\_path', 'author\_id', 'style\_id', 'genre\_id'],

include: [

{

model: Author,

attributes: ['id', 'surname', 'first\_name', 'patronymic']

},

{

model: Style,

attributes: ['id', 'name']

},

{

model: Genre,

attributes: ['id', 'name']

}

]

},

{

model: Ticket,

attributes: ['id', 'quantity', 'booking\_date', 'total\_price']

}

]

});

if (!exhibition) {

return res.status(404).json({

status: 'error',

message: 'Выставка с указанным ID не найдена'

});

}

// Нормализуем patronymic у всех авторов

if (exhibition && exhibition.Artworks) {

exhibition.Artworks.forEach(art => {

if (art.Author) normalizePatronymic(art.Author);

});

}

res.status(200).json({

status: 'success',

data: {

exhibition,

statistics: {

total\_artworks: exhibition.Artworks.length,

total\_tickets: exhibition.Tickets.reduce((sum, ticket) => sum + ticket.quantity, 0)

}

}

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении информации о выставке: ' + err.message

});

}

};

// Обновление выставки по ID

exports.updateExhibition = async (req, res) => {

try {

const exhibition = await Exhibition.findByPk(req.params.id);

if (!exhibition) {

return res.status(404).json({ status: 'error', message: 'Выставка не найдена' });

}

// … ваши валидации …

const updateData = { ...req.body };

if (req.body.total\_tickets !== undefined) {

updateData.remaining\_tickets = req.body.total\_tickets;

}

if (req.file) {

updateData.poster\_path = '/media/' + req.file.filename;

}

await exhibition.update(updateData);

// --- НОВОЕ: обрабатываем artwork\_ids так же, как в create ---

let artworkIds = req.body.artwork\_ids || req.body['artwork\_ids[]'];

if (artworkIds) {

if (typeof artworkIds === 'string') {

try {

artworkIds = JSON.parse(artworkIds);

} catch {

artworkIds = artworkIds.split(',').map(id => id.trim());

}

}

if (Array.isArray(artworkIds)) {

const ids = artworkIds.map(i => parseInt(i, 10)).filter(i => !isNaN(i));

await exhibition.setArtworks(ids);

}

}

const updated = await Exhibition.findByPk(req.params.id, {

include: [{ model: Artwork, attributes: ['id', 'title', 'creation\_year', 'image\_path'] }]

});

return res.status(200).json({

status: 'success',

data: { exhibition: updated }

});

} catch(err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Ошибка при обновлении выставки: ' + err.message

});

}

};

// Обновление статуса выставки

exports.updateExhibitionStatus = async (req, res) => {

return res.status(400).json({

status: 'error',

message: 'Статус выставки определяется автоматически по датам начала и окончания. Ручное изменение невозможно.'

});

};

// Удаление выставки по ID

exports.deleteExhibition = async (req, res) => {

try {

const exhibition = await Exhibition.findByPk(req.params.id, {

include: [{ model: Ticket }]

});

if (!exhibition) {

return res.status(404).json({

status: 'error',

message: 'Выставка с указанным ID не найдена'

});

}

if (exhibition.Tickets.length > 0) {

return res.status(400).json({

status: 'error',

message: 'Невозможно удалить выставку, на которую были забронированы билеты'

});

}

// Удаляем все связи с картинами (junction-таблица exhibition\_artwork)

await exhibition.setArtworks([]);

// Теперь можно удалить саму выставку

await exhibition.destroy();

return res.status(200).json({

status: 'success',

message: 'Выставка и связанные записи успешно удалены',

data: null

});

} catch (err) {

return res.status(500).json({

status: 'error',

message: 'Ошибка при удалении выставки: ' + err.message

});

}

};

exports.renderExhibitionsPage = (req, res) => {

res.render('exhibitions', {

title: 'Выставки',

user: res.locals.user || null

});

};

// Получение только активных и предстоящих выставок для главной страницы

exports.getPublicExhibitions = async (req, res) => {

try {

const where = {

status: { [Op.in]: ['active', 'upcoming'] }

};

// Фильтрация по дате проведения

if (req.query.start\_date || req.query.end\_date) {

const dateConditions = [];

if (req.query.start\_date && req.query.end\_date) {

dateConditions.push({

[Op.and]: [

{ start\_date: { [Op.lte]: req.query.end\_date } },

{ end\_date: { [Op.gte]: req.query.start\_date } }

]

});

} else if (req.query.start\_date) {

dateConditions.push({ start\_date: { [Op.gte]: req.query.start\_date } });

} else if (req.query.end\_date) {

dateConditions.push({ end\_date: { [Op.lte]: req.query.end\_date } });

}

if (dateConditions.length > 0) {

if (where[Op.and]) {

where[Op.and].push({ [Op.and]: dateConditions });

} else {

where[Op.and] = dateConditions;

}

}

}

// Фильтрация по цене билета

if (req.query.min\_price || req.query.max\_price) {

const priceConditions = [];

if (req.query.min\_price) {

const minPrice = parseFloat(req.query.min\_price);

if (!isNaN(minPrice) && minPrice >= 0) {

priceConditions.push({ ticket\_price: { [Op.gte]: minPrice } });

}

}

if (req.query.max\_price) {

const maxPrice = parseFloat(req.query.max\_price);

if (!isNaN(maxPrice) && maxPrice >= 0) {

priceConditions.push({ ticket\_price: { [Op.lte]: maxPrice } });

}

}

if (priceConditions.length > 0) {

if (where[Op.and]) {

where[Op.and].push({ [Op.and]: priceConditions });

} else {

where[Op.and] = priceConditions;

}

}

}

// Поиск по названию и месту проведения

if (req.query.search) {

const searchWords = req.query.search.trim().split(/\s+/);

const searchConditions = searchWords.map(word => ({

[Op.or]: [

{ title: { [Op.like]: `%${word}%` } },

{ location: { [Op.like]: `%${word}%` } }

]

}));

if (where[Op.and]) {

where[Op.and].push(...searchConditions);

} else {

where[Op.and] = searchConditions;

}

}

// Получаем выставки

let exhibitions = await Exhibition.findAll({

where,

order: [

// Сначала активные, потом предстоящие, внутри — по дате начала

[Sequelize.literal(`CASE WHEN status = 'active' THEN 0 WHEN status = 'upcoming' THEN 1 ELSE 2 END`), 'ASC'],

['start\_date', 'ASC']

],

attributes: ['id', 'title', 'location', 'description', 'ticket\_price', 'start\_date', 'end\_date', 'status', 'poster\_path']

});

// Нормализуем patronymic у всех авторов

exhibitions.forEach(exh => {

if (exh.Artworks) {

exh.Artworks.forEach(art => {

if (art.Author) normalizePatronymic(art.Author);

});

}

});

res.status(200).json({

status: 'success',

data: { exhibitions }

});

} catch (err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении выставок: ' + err.message

});

}

};

// Рендер подробной страницы выставки

exports.renderExhibitionPage = async (req, res) => {

try {

const exhibition = await Exhibition.findByPk(req.params.id, {

include: [

{

model: Artwork,

attributes: ['id', 'title', 'image\_path'],

include: [{

model: Author,

attributes: ['surname', 'first\_name', 'patronymic']

}]

}

]

});

if (!exhibition) {

return res.status(404).render('error', { message: 'Выставка не найдена' });

}

res.render('exhibition', {

title: exhibition.title,

exhibition: exhibition.toJSON(),

user: res.locals.user || null

});

} catch (err) {

res.status(500).render('error', { message: 'Ошибка при загрузке выставки: ' + err.message });

}

};

// Вспомогательная функция для нормализации patronymic

function normalizePatronymic(obj) {

if (!obj) return obj;

if (Array.isArray(obj)) {

return obj.map(normalizePatronymic);

}

if (typeof obj === 'object') {

if ('patronymic' in obj && obj.patronymic === null) {

obj.patronymic = '';

}

// Рекурсивно для вложенных объектов

for (const key in obj) {

if (typeof obj[key] === 'object' && obj[key] !== null) {

obj[key] = normalizePatronymic(obj[key]);

}

}

}

return obj;

}

ПРИЛОЖЕНИЕ 6. Файл controllers/genreController.js

const { Genre, Artwork, Author, Style } = require('../models');

const { Op, ValidationError } = require('sequelize');

// Создание нового жанра

exports.createGenre = async (req, res) => {

try {

const newGenre = await Genre.create(req.body);

res.status(201).json({

status: 'success',

message: 'Жанр успешно создан',

data: { genre: newGenre }

});

} catch (err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Не удалось создать жанр: ' + err.message

});

}

};

// Получение всех жанров

exports.getAllGenres = async (req, res) => {

try {

// Фильтрация по названию

const where = {};

if (req.query.name) {

where.name = { [Op.like]: `%${req.query.name}%` };

}

const genres = await Genre.findAll({

where,

include: [{

model: Artwork,

attributes: ['id', 'title'],

required: false

}],

distinct: true

});

// Добавляем статистику по произведениям

const genresWithStats = genres.map(genre => ({

...genre.toJSON(),

statistics: {

total\_artworks: genre.Artworks ? genre.Artworks.length : 0

}

}));

res.status(200).json({

status: 'success',

data: {

genres: genresWithStats

}

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении списка жанров: ' + err.message

});

}

};

// Получение жанра по ID

exports.getGenre = async (req, res) => {

try {

const genre = await Genre.findByPk(req.params.id, {

include: [{

model: Artwork,

attributes: ['id', 'title', 'creation\_year', 'image\_path', 'author\_id', 'style\_id'],

include: [

{

model: Author,

attributes: ['id', 'surname', 'first\_name', 'patronymic']

},

{

model: Style,

attributes: ['id', 'name']

}

]

}]

});

if (!genre) {

return res.status(404).json({

status: 'error',

message: 'Жанр с указанным ID не найден'

});

}

res.status(200).json({

status: 'success',

data: {

genre

}

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении информации о жанре: ' + err.message

});

}

};

// Обновление жанра по ID

exports.updateGenre = async (req, res) => {

try {

const genre = await Genre.findByPk(req.params.id);

if (!genre) {

return res.status(404).json({

status: 'error',

message: 'Жанр с указанным ID не найден'

});

}

// Проверяем, существует ли жанр с таким именем

if (req.body.name) {

const existingGenre = await Genre.findOne({

where: {

name: req.body.name,

id: { [Op.ne]: req.params.id }

}

});

if (existingGenre) {

return res.status(400).json({

status: 'error',

message: 'Жанр с таким именем уже существует'

});

}

}

// Обновляем жанр

await genre.update(req.body);

// Получаем обновленный жанр

const updatedGenre = await Genre.findByPk(req.params.id, {

include: [{

model: Artwork,

attributes: ['id', 'title']

}]

});

res.status(200).json({

status: 'success',

message: 'Жанр успешно обновлен',

data: {

genre: updatedGenre

}

});

} catch(err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Ошибка при обновлении жанра: ' + err.message

});

}

};

// Удаление жанра по ID

exports.deleteGenre = async (req, res) => {

try {

const genre = await Genre.findByPk(req.params.id, {

include: [{

model: Artwork,

attributes: ['id']

}]

});

if (!genre) {

return res.status(404).json({

status: 'error',

message: 'Жанр с указанным ID не найден'

});

}

// Проверяем, есть ли произведения с этим жанром

if (genre.Artworks && genre.Artworks.length > 0) {

return res.status(400).json({

status: 'error',

message: 'Невозможно удалить жанр, который используется в произведениях',

data: {

artworksCount: genre.Artworks.length

}

});

}

await genre.destroy();

res.status(200).json({

status: 'success',

message: 'Жанр успешно удален',

data: null

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при удалении жанра: ' + err.message

});

}

};

ПРИЛОЖЕНИЕ 7. Файл src\controllers\managerController.js

const { Op, fn, col } = require('sequelize');

const { Ticket } = require('../models');

exports.renderStatistics = (req, res) => {

// Здесь вы можете подготовить любые данные для отчётов, например:

// const reportData = await ReportService.getStats();

// Для начала просто рендерим пустой шаблон.

res.render('manager/statistics', {

title: 'Статистика',

user: res.locals.user

// , reportData

});

};

exports.getTicketStats = async (req, res) => {

try {

const { start\_date, end\_date } = req.query;

const where = {};

if (start\_date) {

where.booking\_date = { [Op.gte]: `${start\_date} 00:00:00` };

}

if (end\_date) {

where.booking\_date = {

...where.booking\_date,

[Op.lte]: `${end\_date} 23:59:59`

};

}

const stats = await Ticket.findAll({

where,

attributes: [

[ fn('SUM', col('quantity')), 'tickets\_sold' ],

[ fn('SUM', col('total\_price')), 'revenue' ]

]

});

// stats[0].dataValues содержит { tickets\_sold: ..., revenue: ... }

res.status(200).json({

status: 'success',

data: stats[0].dataValues

});

} catch (err) {

res.status(500).json({

status: 'error',

message: 'Не удалось получить статистику: ' + err.message

});

}

};

ПРИЛОЖЕНИЕ 8. Файл src\controllers\styleController.js

const { Style, Artwork, Author, Genre } = require('../models');

const { Op, ValidationError } = require('sequelize');

// Создание нового стиля

exports.createStyle = async (req, res) => {

try {

const newStyle = await Style.create(req.body);

res.status(201).json({

status: 'success',

message: 'Стиль успешно создан',

data: { style: newStyle }

});

} catch(err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Не удалось создать стиль: ' + err.message

});

}

};

// Получение всех стилей

exports.getAllStyles = async (req, res) => {

try {

// Фильтрация по названию

const where = {};

if (req.query.name) {

where.name = { [Op.like]: `%${req.query.name}%` };

}

const styles = await Style.findAll({

where,

include: [{

model: Artwork,

attributes: ['id', 'title'],

required: false

}],

distinct: true

});

// Добавляем статистику по произведениям

const stylesWithStats = styles.map(style => ({

...style.toJSON(),

statistics: {

total\_artworks: style.Artworks ? style.Artworks.length : 0

}

}));

res.status(200).json({

status: 'success',

data: {

styles: stylesWithStats

}

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении списка стилей: ' + err.message

});

}

};

// Получение стиля по ID

exports.getStyle = async (req, res) => {

try {

const style = await Style.findByPk(req.params.id, {

include: [{

model: Artwork,

attributes: ['id', 'title', 'creation\_year', 'image\_path', 'author\_id', 'genre\_id'],

include: [

{

model: Author,

attributes: ['id', 'surname', 'first\_name', 'patronymic']

},

{

model: Genre,

attributes: ['id', 'name']

}

]

}]

});

if (!style) {

return res.status(404).json({

status: 'error',

message: 'Стиль с указанным ID не найден'

});

}

res.status(200).json({

status: 'success',

data: {

style

}

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении информации о стиле: ' + err.message

});

}

};

// Обновление стиля по ID

exports.updateStyle = async (req, res) => {

try {

const style = await Style.findByPk(req.params.id);

if (!style) {

return res.status(404).json({

status: 'error',

message: 'Стиль с указанным ID не найден'

});

}

// Проверяем, существует ли стиль с таким именем

if (req.body.name) {

const existingStyle = await Style.findOne({

where: {

name: req.body.name,

id: { [Op.ne]: req.params.id }

}

});

if (existingStyle) {

return res.status(400).json({

status: 'error',

message: 'Стиль с таким именем уже существует'

});

}

}

// Обновляем стиль

await style.update(req.body);

// Получаем обновленный стиль

const updatedStyle = await Style.findByPk(req.params.id, {

include: [{

model: Artwork,

attributes: ['id', 'title', 'creation\_year', 'image\_path']

}]

});

res.status(200).json({

status: 'success',

message: 'Стиль успешно обновлен',

data: {

style: updatedStyle

}

});

} catch(err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Ошибка при обновлении стиля: ' + err.message

});

}

};

// Удаление стиля по ID

exports.deleteStyle = async (req, res) => {

try {

const style = await Style.findByPk(req.params.id, {

include: [{

model: Artwork,

attributes: ['id']

}]

});

if (!style) {

return res.status(404).json({

status: 'error',

message: 'Стиль с указанным ID не найден'

});

}

// Проверяем, есть ли произведения с этим стилем

if (style.Artworks && style.Artworks.length > 0) {

return res.status(400).json({

status: 'error',

message: 'Невозможно удалить стиль, который используется в произведениях',

data : {

artworksCount: style.Artwork.length

}

});

}

await style.destroy();

res.status(200).json({

status: 'success',

message: 'Стиль успешно удален',

data: null

});

} catch(err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при удалении стиля: ' + err.message

});

}

};

ПРИЛОЖЕНИЕ 9. Файл src\controllers\ticketController.js

// Предполагаемый код, адаптированный под изменения

const { Ticket, Exhibition, User } = require('../models');

const { Op, ValidationError } = require('sequelize');

// Создание билета

exports.createTicket = async (req, res) => {

try {

const { exhibition\_id, quantity } = req.body;

if (!exhibition\_id || !quantity) {

return res.status(400).json({

status: 'error',

message: 'Необходимо указать выставку и количество билетов'

});

}

const exhibition = await Exhibition.findByPk(exhibition\_id);

if (!exhibition) {

return res.status(404).json({

status: 'error',

message: 'Выставка не найдена'

});

}

if (exhibition.status === 'completed') {

return res.status(400).json({

status: 'error',

message: 'Выставка недоступна для покупки билетов'

});

}

if (exhibition.remaining\_tickets < quantity) {

return res.status(400).json({

status: 'error',

message: 'Недостаточно билетов'

});

}

const total\_price = exhibition.ticket\_price \* quantity;

const ticket = await Ticket.create({

user\_id: req.user.id,

exhibition\_id,

quantity,

booking\_date: new Date(),

total\_price

});

await exhibition.update({

remaining\_tickets: exhibition.remaining\_tickets - quantity

});

res.status(201).json({

status: 'success',

data: { ticket }

});

} catch (err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Ошибка при создании билета: ' + err.message

});

}

};

// Получение всех билетов

exports.getAllTickets = async (req, res) => {

try {

const where = {};

// Поиск по тексту

if (req.query.search) {

const searchWords = req.query.search.trim().split(/\s+/);

where[Op.and] = searchWords.map(word => ({

[Op.or]: [

{ id: { [Op.like]: `%${word}%` } },

{ user\_id: { [Op.like]: `%${word}%` } },

{ exhibition\_id: { [Op.like]: `%${word}%` } },

{ quantity: { [Op.like]: `%${word}%` } },

{ total\_price: { [Op.like]: `%${word}%` } },

{ booking\_date: { [Op.like]: `%${word}%` } },

{ '$User.surname$': { [Op.like]: `%${word}%` } },

{ '$User.first\_name$': { [Op.like]: `%${word}%` } },

{ '$User.patronymic$': { [Op.like]: `%${word}%` } },

{ '$Exhibition.title$': { [Op.like]: `%${word}%` } }

]

}));

}

// Фильтрация по дате бронирования

if (req.query.booking\_date\_from || req.query.booking\_date\_to) {

const dateConditions = [];

if (req.query.booking\_date\_from && req.query.booking\_date\_to) {

// Если указаны обе даты, проверяем диапазон

dateConditions.push({

[Op.and]: [

{ booking\_date: { [Op.gte]: req.query.booking\_date\_from } },

{ booking\_date: { [Op.lte]: req.query.booking\_date\_to + ' 23:59:59' } }

]

});

} else if (req.query.booking\_date\_from) {

// Если указана только начальная дата

dateConditions.push({

booking\_date: { [Op.gte]: req.query.booking\_date\_from }

});

} else if (req.query.booking\_date\_to) {

// Если указана только конечная дата

dateConditions.push({

booking\_date: { [Op.lte]: req.query.booking\_date\_to + ' 23:59:59' }

});

}

if (dateConditions.length > 0) {

if (where[Op.and]) {

where[Op.and].push({ [Op.and]: dateConditions });

} else {

where[Op.and] = dateConditions;

}

}

}

// Фильтрация по цене

if (req.query.min\_price || req.query.max\_price) {

const priceConditions = [];

if (req.query.min\_price) {

const minPrice = parseFloat(req.query.min\_price);

if (!isNaN(minPrice) && minPrice >= 0) {

priceConditions.push({ total\_price: { [Op.gte]: minPrice } });

}

}

if (req.query.max\_price) {

const maxPrice = parseFloat(req.query.max\_price);

if (!isNaN(maxPrice) && maxPrice >= 0) {

priceConditions.push({ total\_price: { [Op.lte]: maxPrice } });

}

}

if (priceConditions.length > 0) {

if (where[Op.and]) {

where[Op.and].push({ [Op.and]: priceConditions });

} else {

where[Op.and] = priceConditions;

}

}

}

const tickets = await Ticket.findAll({

where,

include: [

{ model: User, attributes: ['id', 'surname', 'first\_name', 'patronymic'], required: false },

{ model: Exhibition, attributes: ['id', 'title'], required: false }

],

order: [['booking\_date', 'DESC']]

});

res.status(200).json({

status: 'success',

data: { tickets }

});

} catch (err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении билетов: ' + err.message

});

}

};

exports.updateTicketAdmin = async (req, res) => {

try {

// 1) убедиться, что это админ

if (req.user.role !== 'admin') {

return res.status(403).json({ status: 'error', message: 'Доступ запрещён' });

}

const ticketId = req.params.id;

const { user\_id, exhibition\_id, quantity } = req.body;

// 2) базовая валидация входных данных

if (!user\_id || !exhibition\_id || !quantity) {

return res.status(400).json({

status: 'error',

message: 'Необходимо указать пользователя, выставку и количество билетов'

});

}

// 3) найти существующий билет

const ticket = await Ticket.findByPk(ticketId);

if (!ticket) {

return res.status(404).json({ status: 'error', message: 'Билет не найден' });

}

// 4) проверить, что новый пользователь существует

const user = await User.findByPk(user\_id);

if (!user) {

return res.status(404).json({ status: 'error', message: 'Пользователь не найден' });

}

// 5) найти старую и новую выставки

const oldExhibition = await Exhibition.findByPk(ticket.exhibition\_id);

const newExhibition = await Exhibition.findByPk(exhibition\_id);

if (!newExhibition) {

return res.status(404).json({ status: 'error', message: 'Новая выставка не найдена' });

}

if (newExhibition.status !== 'active') {

return res.status(400).json({ status: 'error', message: 'Выставка недоступна для бронирования' });

}

// 6) скорректировать остатки билетов

const oldQty = ticket.quantity;

const newQty = parseInt(quantity, 10);

if (ticket.exhibition\_id === exhibition\_id) {

// если выставка та же, смотрим разницу

const diff = newQty - oldQty;

if (diff > 0 && newExhibition.remaining\_tickets < diff) {

return res.status(400).json({ status: 'error', message: 'Недостаточно билетов на выставке' });

}

await newExhibition.update({ remaining\_tickets: newExhibition.remaining\_tickets - diff });

} else {

// возвращаем старые и списываем новые

if (oldExhibition) {

await oldExhibition.update({ remaining\_tickets: oldExhibition.remaining\_tickets + oldQty });

}

if (newExhibition.remaining\_tickets < newQty) {

return res.status(400).json({ status: 'error', message: 'Недостаточно билетов на новой выставке' });

}

await newExhibition.update({ remaining\_tickets: newExhibition.remaining\_tickets - newQty });

}

// 7) пересчитать итоговую цену и сохранить изменения

const total\_price = newExhibition.ticket\_price \* newQty;

await ticket.update({ user\_id, exhibition\_id, quantity: newQty, total\_price });

res.status(200).json({ status: 'success', data: { ticket } });

} catch (err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({ field: e.path, message: e.message }))

});

}

res.status(500).json({ status: 'error', message: 'Ошибка при обновлении билета: ' + err.message });

}

};

// Рендер страницы покупок пользователя

exports.renderUserTickets = async (req, res, next) => {

try {

const userId = res.locals.user.id;

const tickets = await Ticket.findAll({

where: { user\_id: userId },

include: [

{

model: Exhibition,

attributes: ['id', 'title', 'start\_date', 'end\_date', 'location']

}

],

order: [['booking\_date', 'DESC']]

});

res.render('tickets', {

title: 'Мои билеты',

user: res.locals.user,

tickets

});

} catch (err) {

next(err);

}

};

// Получение билетов текущего пользователя

exports.getMyTickets = async (req, res) => {

try {

const tickets = await Ticket.findAll({

where: { user\_id: req.user.id },

include: [{ model: Exhibition, attributes: ['id', 'title', 'start\_date', 'end\_date', 'location'] }]

});

res.status(200).json({

status: 'success',

data: { tickets }

});

} catch (err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении билетов: ' + err.message

});

}

};

exports.createTicketAdmin = async (req, res) => {

try {

// Проверяем роль

if (req.user.role !== 'admin') {

return res.status(403).json({

status: 'error',

message: 'Доступ запрещён'

});

}

const { user\_id, exhibition\_id, quantity } = req.body;

// Базовая валидация

if (!user\_id || !exhibition\_id || !quantity) {

return res.status(400).json({

status: 'error',

message: 'Необходимо указать пользователя, выставку и количество билетов'

});

}

// Проверяем, что пользователь существует

const user = await User.findByPk(user\_id);

if (!user) {

return res.status(404).json({

status: 'error',

message: 'Пользователь не найден'

});

}

// Проверяем выставку

const exhibition = await Exhibition.findByPk(exhibition\_id);

if (!exhibition) {

return res.status(404).json({

status: 'error',

message: 'Выставка не найдена'

});

}

if (exhibition.status === 'completed') {

return res.status(400).json({

status: 'error',

message: 'Выставка недоступна для покупки билетов'

});

}

if (exhibition.remaining\_tickets < quantity) {

return res.status(400).json({

status: 'error',

message: 'Недостаточно билетов'

});

}

// Подсчёт итоговой цены

const total\_price = exhibition.ticket\_price \* quantity;

// Создаём запись о билете

const ticket = await Ticket.create({

user\_id,

exhibition\_id,

quantity,

booking\_date: new Date(),

total\_price

});

// Уменьшаем количество доступных билетов

await exhibition.update({

remaining\_tickets: exhibition.remaining\_tickets - quantity

});

return res.status(201).json({

status: 'success',

data: { ticket }

});

} catch (err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({ field: e.path, message: e.message }))

});

}

return res.status(500).json({

status: 'error',

message: 'Ошибка при создании билета: ' + err.message

});

}

};

// Получение билета по ID

exports.getTicket = async (req, res) => {

try {

const ticket = await Ticket.findByPk(req.params.id, {

include: [{ model: Exhibition, attributes: ['id', 'title', 'start\_date', 'end\_date'] }]

});

if (!ticket) {

return res.status(404).json({

status: 'error',

message: 'Билет не найден'

});

}

if (req.user.role !== 'admin' && ticket.user\_id !== req.user.id) {

return res.status(403).json({

status: 'error',

message: 'Доступ запрещён'

});

}

res.status(200).json({

status: 'success',

data: { ticket }

});

} catch (err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении билета: ' + err.message

});

}

};

// Отмена (удаление) билета

exports.cancelTicket = async (req, res) => {

try {

const ticket = await Ticket.findByPk(req.params.id);

if (!ticket) {

return res.status(404).json({

status: 'error',

message: 'Билет не найден'

});

}

if (ticket.user\_id !== req.user.id && req.user.role !== 'admin') {

return res.status(403).json({

status: 'error',

message: 'Доступ запрещён'

});

}

const exhibition = await Exhibition.findByPk(ticket.exhibition\_id);

if (exhibition) {

await exhibition.update({

remaining\_tickets: exhibition.remaining\_tickets + ticket.quantity

});

}

await ticket.destroy();

res.status(200).json({

status: 'success',

message: 'Билет успешно отменён',

data: null

});

} catch (err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при отмене билета: ' + err.message

});

}

};

ПРИЛОЖЕНИЕ 10. Файл src\controllers\userController.js

const { User, Ticket, Exhibition } = require('../models');

const { Op, ValidationError } = require('sequelize');

// Функция для фильтрации чувствительных данных

const filterSensitiveData = (user) => {

const { password, ...userWithoutSensitiveData } = user.toJSON();

return userWithoutSensitiveData;

};

// Утилита для замены null в patronymic на пустую строку

const normalizePatronymic = (obj) => {

if (!obj) return obj;

if (Array.isArray(obj)) {

return obj.map(normalizePatronymic);

}

if (typeof obj === 'object') {

if ('patronymic' in obj && obj.patronymic === null) {

obj.patronymic = '';

}

// Рекурсивно для вложенных объектов (например, Tickets)

for (const key in obj) {

if (typeof obj[key] === 'object' && obj[key] !== null) {

obj[key] = normalizePatronymic(obj[key]);

}

}

}

return obj;

};

// Создание нового пользователя (админская операция)

exports.createUser = async (req, res) => {

try {

// Преобразуем пустую строку patronymic в null

if (req.body.patronymic === '') req.body.patronymic = null;

const newUser = await User.create(req.body);

res.status(201).json({

status: 'success',

message: 'Пользователь успешно создан',

data: {

user: normalizePatronymic(filterSensitiveData(newUser))

}

});

} catch (err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Не удалось создать пользователя'

});

}

};

// Получение всех пользователей

exports.getAllUsers = async (req, res) => {

try {

const where = {};

if (req.query.name) {

const searchWords = req.query.name.trim().split(/\s+/);

where[Op.and] = searchWords.map(word => ({

[Op.or]: [

{ surname: { [Op.like]: `%${word}%` } },

{ first\_name: { [Op.like]: `%${word}%` } },

{ patronymic: { [Op.like]: `%${word}%` } },

{ email: { [Op.like]: `%${word}%` } },

{ role: { [Op.like]: `%${word}%` } }

]

}));

}

const users = await User.findAll({

where,

include: [{

model: Ticket,

attributes: ['id'],

required: false

}],

distint: true

});

const usersWithStats = users.map(user => {

const userData = filterSensitiveData(user);

return {

...normalizePatronymic(userData),

statistics: {

total\_tickets: user.Tickets ? user.Tickets.length : 0

}

};

});

res.status(200).json({

status: 'success',

data: {

users: usersWithStats

}

});

} catch (err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении списка пользователей'

});

}

};

// Получение пользователя по ID

exports.getUser = async (req, res) => {

try {

const user = await User.findByPk(req.params.id, {

include: [{

model: Ticket,

attributes: ['id', 'exhibition\_id', 'quantity', 'booking\_date', 'total\_price'],

include: [{

model: Exhibition,

attributes: ['id', 'title', 'start\_date', 'end\_date']

}]

}]

});

if (!user) {

return res.status(404).json({

status: 'error',

message: 'Пользователь с указанным ID не найден'

});

}

const userWithStats = {

...normalizePatronymic(filterSensitiveData(user)),

statistics: {

total\_tickets: user.Tickets.reduce((sum, ticket) => sum + ticket.quantity, 0)

}

};

res.status(200).json({

status: 'success',

data: {

user: userWithStats

}

});

} catch (err) {

console.error('Get user error:', err);

res.status(500).json({

status: 'error',

message: 'Ошибка при получении информации о пользователе'

});

}

};

// Обновление пользователя по ID

exports.updateUser = async (req, res) => {

try {

const user = await User.findByPk(req.params.id);

if (!user) {

return res.status(404).json({

status: 'error',

message: 'Пользователь с указанным ID не найден'

});

}

// Проверка email

if (req.body.email && req.body.email !== user.email) {

const existingUser = await User.findOne({

where: {

email: req.body.email,

id: { [Op.ne]: req.params.id }

}

});

if (existingUser) {

return res.status(400).json({

status: 'error',

message: 'Пользователь с таким email уже существует'

});

}

}

// Преобразуем пустую строку patronymic в null

if (req.body.patronymic === '') req.body.patronymic = null;

// Обновляем пользователя

await user.update(req.body);

// Получение обновлённого пользователя

const updatedUser = await User.findByPk(req.params.id, {

include: [{

model: Ticket,

attributes: ['id']

}]

});

res.status(200).json({

status: 'success',

data: {

user: normalizePatronymic(filterSensitiveData(updatedUser))

}

});

} catch (err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Ошибка при обновлении пользователя'

});

}

};

// Удаление пользователя по ID

exports.deleteUser = async (req, res) => {

try {

const user = await User.findByPk(req.params.id, {

include: [{

model: Ticket,

attributes: ['id']

}]

});

if (!user) {

return res.status(404).json({

status: 'error',

message: 'Пользователь с указанным ID не найден'

});

}

if (user.Tickets && user.Tickets.length > 0) {

return res.status(400).json({

status: 'error',

message: 'Невозможно удалить пользователя с забронированными билетами',

data: {

ticketCount: user.Tickets.length

}

});

}

await user.destroy();

res.status(200).json({

status: 'success',

message: 'Пользователь успешно удален',

data: null

});

} catch (err) {

console.error('Delete user error:', err);

res.status(500).json({

status: 'error',

message: 'Ошибка при удалении пользователя'

});

}

};

// Обновление профиля текущего пользователя

exports.updateMe = async (req, res) => {

try {

const user = await User.findByPk(req.user.id);

if (!user) {

return res.status(404).json({

status: 'error',

message: 'Пользователь не найден'

});

}

if (req.body.role) {

delete req.body.role;

}

// Проверка email

if (req.body.email && req.body.email !== user.email) {

const existingUser = await User.findOne({

where: {

email: req.body.email,

id: { [Op.ne]: req.user.id }

}

});

if (existingUser) {

return res.status(400).json({

status: 'error',

message: 'Пользователь с таким email уже существует'

});

}

}

await user.update(req.body);

const updatedUser = await User.findByPk(req.user.id, {

include: [{

model: Ticket,

attributes: ['id']

}]

});

res.status(200).json({

status: 'success',

message: 'Профиль успешно обновлен',

data: {

user: normalizePatronymic(filterSensitiveData(updatedUser))

}

});

} catch (err) {

if (err instanceof ValidationError) {

return res.status(400).json({

status: 'error',

message: 'Ошибка валидации данных',

errors: err.errors.map(e => ({

field: e.path,

message: e.message

}))

});

}

res.status(500).json({

status: 'error',

message: 'Ошибка при обновлении профиля'

});

}

};

// Получение профиля текущего пользователя

exports.getMe = async (req, res) => {

try {

const user = await User.findByPk(req.user.id);

if (!user) {

return res.status(404).json({

status: 'error',

message: 'Пользователь не найден'

});

}

res.status(200).json({

status: 'success',

data: {

user: normalizePatronymic(filterSensitiveData(user))

}

});

} catch (err) {

res.status(500).json({

status: 'error',

message: 'Ошибка при получении профиля'

});

}

};

ПРИЛОЖЕНИЕ 11. Файл src\middleware\auth.js

const jwt = require('jsonwebtoken');

const { User } = require('../models');

// Middleware для проверки JWT токена

const protect = async (req, res, next) => {

try {

// Получаем токен из заголовка или куки

let token;

if (req.headers.authorization && req.headers.authorization.startsWith('Bearer')) {

token = req.headers.authorization.split(' ')[1];

} else if (req.cookies && req.cookies.token) {

token = req.cookies.token;

}

if (!token) {

return res.status(401).json({

status: 'error',

message: 'Вы не авторизованы'

});

}

// Проверяем токен

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

const user = await User.findByPk(decoded.id);

if (!user) {

return res.status(401).json({

status: 'error',

message: 'Пользователь не найден'

});

}

// Добавляем пользователя в запрос

req.user = user;

next();

} catch (err) {

if (err.name === 'JsonWebTokenError') {

return res.status(401).json({

status: 'error',

message: 'Недействительный токен'

});

}

if (err.name === 'TokenExpiredError') {

return res.status(401).json({

status: 'error',

message: 'Срок действия токена истек'

});

}

next(err);

}

};

// Middleware для проверки роли

const restrictTo = (...roles) => {

return (req, res, next) => {

if (!roles.includes(req.user.role)) {

return res.status(403).json({

status: 'error',

message: 'У вас нет прав для выполнения этого действия'

});

}

next();

};

};

module.exports = {

protect,

restrictTo

};

ПРИЛОЖЕНИЕ 12. Файл src\middleware\setUser.js

const jwt = require('jsonwebtoken');

const { User } = require('../models');

module.exports = async (req, res, next) => {

try {

// Получаем токен из куки

const token = req.cookies.token;

if (!token) {

res.locals.user = null;

return next();

}

// Проверяем токен

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

const user = await User.findByPk(decoded.id);

if (!user) {

res.locals.user = null;

return next();

}

// Добавляем пользователя в res.locals

res.locals.user = user;

next();

} catch (err) {

res.locals.user = null;

next();

}

};

ПРИЛОЖЕНИЕ 13. Файл src\middleware\upload.js

const multer = require('multer');

const path = require('path');

// Настройка хранилища для multer

const storage = multer.diskStorage({

destination: function (req, file, cb) {

cb(null, path.join(\_\_dirname, '../public/media'));

},

filename: function (req, file, cb) {

const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() \* 1E9);

cb(null, uniqueSuffix + path.extname(file.originalname));

}

});

// Фильтр файлов

const fileFilter = (req, file, cb) => {

if (file.mimetype.startsWith('image/')) {

cb(null, true);

} else {

cb(new Error('Только изображения разрешены!'), false);

}

};

// Создаем экземпляр multer

const upload = multer({

storage: storage,

fileFilter: fileFilter,

limits: {

fileSize: 5 \* 1024 \* 1024 // 5MB

}

});

module.exports = upload;

ПРИЛОЖЕНИЕ 14. Файл src\middleware\validation.js

const { validationResult } = require('express-validator');

const { body, param } = require('express-validator');

const { Genre, Artwork, Exhibition, Author, Style } = require('../models');

const validate = (req, res, next) => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

return res.status(400).json({ errors: errors.array() });

}

next();

};

const validateRegister = [

body('email').trim().notEmpty().withMessage('Email обязателен').isEmail().withMessage('Неверный формат email'),

body('password').trim().notEmpty().withMessage('Пароль обязателен').isLength({ min: 8 }).withMessage('Пароль должен быть не менее 8 символов'),

body('surname').trim().notEmpty().withMessage('Фамилия обязательна').isLength({ min: 2, max: 50 }).withMessage('Фамилия должна быть от 2 до 50 символов'),

body('first\_name').trim().notEmpty().withMessage('Имя обязательно').isLength({ min: 2, max: 50 }).withMessage('Имя должно быть от 2 до 50 символов'),

body('patronymic').optional().trim().isLength({ min: 2, max: 50 }).withMessage('Отчество должно быть от 2 до 50 символов')

];

const validateLogin = [

body('email').trim().notEmpty().withMessage('Email обязателен').isEmail().withMessage('Неверный формат email'),

body('password').trim().notEmpty().withMessage('Пароль обязателен')

];

const validatePasswordChange = [

body('currentPassword').trim().notEmpty().withMessage('Текущий пароль обязателен'),

body('newPassword').trim().notEmpty().withMessage('Новый пароль обязателен').isLength({ min: 8 }).withMessage('Пароль должен быть не менее 8 символов')

];

const validateCreateUser = [

body('email').trim().notEmpty().withMessage('Email обязателен').isEmail().withMessage('Неверный формат email'),

body('password').trim().notEmpty().withMessage('Пароль обязателен').isLength({ min: 8 }).withMessage('Пароль должен быть не менее 8 символов'),

body('surname').trim().notEmpty().withMessage('Фамилия обязательна').isLength({ min: 2, max: 50 }).withMessage('Фамилия должна быть от 2 до 50 символов'),

body('first\_name').trim().notEmpty().withMessage('Имя обязательно').isLength({ min: 2, max: 50 }).withMessage('Имя должно быть от 2 до 50 символов'),

body('patronymic').optional({ nullable: true }).trim().custom((value) => {

if (value === null || value === '') {

return true;

}

if (value.length < 2 || value.length > 50) {

throw new Error('Отчество должно быть от 2 до 50 символов');

}

return true;

}),

body('role').optional().isIn(['admin', 'manager', 'user']).withMessage('Неверная роль пользователя')

];

const validateStyle = [

body('name').trim().notEmpty().withMessage('Название стиля обязательно').isLength({ min: 2, max: 50 }).withMessage('Название должно быть от 2 до 50 символов')

];

const validateStyleUpdate = [

body('name').optional().trim().isLength({ min: 2, max: 50 }).withMessage('Название должно быть от 2 до 50 символов')

];

const validateStyleId = [

param('id').isInt().withMessage('ID стиля должен быть числом')

];

const validateGenre = [

body('name').trim().notEmpty().withMessage('Название жанра обязательно').isLength({ min: 2, max: 50 }).withMessage('Название должно быть от 2 до 50 символов')

];

const validateGenreUpdate = [

body('name').optional().trim().isLength({ min: 2, max: 50 }).withMessage('Название должно быть от 2 до 50 символов'),

body().custom((value, { req }) => {

if (!req.body.name) {

throw new Error('Поле "name" должно быть передано');

}

return true;

})

];

const validateGenreId = [

param('id').isInt().withMessage('ID жанра должен быть числом').custom(async (id) => {

const genre = await Genre.findByPk(id);

if (!genre) {

throw new Error('Жанр с указанным ID не найден');

}

return true;

})

];

const validateAuthor = [

body('surname').trim().notEmpty().withMessage('Фамилия автора обязательна').isLength({ min: 2, max: 50 }).withMessage('Фамилия должна быть от 2 до 50 символов'),

body('first\_name').trim().notEmpty().withMessage('Имя автора обязательно').isLength({ min: 2, max: 50 }).withMessage('Имя должно быть от 2 до 50 символов'),

body('patronymic').optional().trim().isLength({ min: 2, max: 50 }).withMessage('Отчество должно быть от 2 до 50 символов'),

body('date\_of\_birth').isDate().withMessage('Неверный формат даты рождения'),

body('date\_of\_death').optional().isDate().withMessage('Неверный формат даты смерти').custom((value, { req }) => {

if (value && new Date(value) > new Date()) {

throw new Error('Дата смерти не может быть в будущем');

}

return true;

})

];

const validateAuthorUpdate = [

body('surname').optional().trim().isLength({ min: 2, max: 50 }).withMessage('Фамилия должна быть от 2 до 50 символов'),

body('first\_name').optional().trim().isLength({ min: 2, max: 50 }).withMessage('Имя должно быть от 2 до 50 символов'),

body('patronymic').optional({ nullable: true }).trim().custom((value) => {

if (value === null || value === '') {

return true;

}

if (value.length < 2 || value.length > 50) {

throw new Error('Отчество должно быть от 2 до 50 символов');

}

return true;

}),

body('date\_of\_birth').isDate().withMessage('Неверный формат даты рождения'),

body('date\_of\_death').optional().isDate().withMessage('Неверный формат даты смерти').custom((value, { req }) => {

if (value && new Date(value) > new Date()) {

throw new Error('Дата смерти не может быть в будущем');

}

return true;

})

];

const validateAuthorId = [

param('id').isInt().withMessage('ID автора должен быть числом')

];

const validateArtwork = [

body('title').trim().notEmpty().withMessage('Название произведения обязательно').isLength({ min: 2, max: 100 }).withMessage('Название должно быть от 2 до 100 символов'),

body('width').notEmpty().withMessage('Ширина произведения обязательна').isFloat({ min: 0 }).withMessage('Ширина должна быть положительным числом'),

body('height').notEmpty().withMessage('Высота произведения обязательна').isFloat({ min: 0 }).withMessage('Высота должна быть положительным числом'),

body('creation\_year').notEmpty().withMessage('Год создания произведения обязателен').isInt({ max: new Date().getFullYear() }).withMessage('Неверный год создания'),

body('author\_id').isInt().withMessage('ID автора должен быть числом'),

body('style\_id').isInt().withMessage('ID стиля должен быть числом'),

body('genre\_id').isInt().withMessage('ID жанра должен быть числом'),

body('description').optional().trim().isLength({ max: 2000 }).withMessage('Описание не должно превышать 2000 символов'),

body('exhibition\_id').optional().isInt().withMessage('ID выставки должен быть числом')

];

const validateArtworkId = [

param('id').isInt().withMessage('ID произведения должен быть числом')

];

const validateArtworkUpdate = [

body('title').optional().trim().notEmpty().withMessage('Название произведения обязательно').isLength({ min: 2, max: 100 }).withMessage('Название должно быть от 2 до 100 символов'),

body('width').optional().isFloat({ min: 0 }).withMessage('Ширина должна быть положительным числом'),

body('height').optional().isFloat({ min: 0 }).withMessage('Высота должна быть положительным числом'),

body('creation\_year').optional().isInt({ max: new Date().getFullYear() }).withMessage('Неверный год создания'),

body('author\_id').optional().isInt().withMessage('ID автора должен быть числом'),

body('style\_id').optional().isInt().withMessage('ID стиля должен быть числом'),

body('genre\_id').optional().isInt().withMessage('ID жанра должен быть числом'),

body('description').optional().trim().isLength({ max: 2000 }).withMessage('Описание не должно превышать 2000 символов'),

body('image\_path').optional().trim().notEmpty().withMessage('Путь к изображению обязателен'),

body('exhibition\_id').optional().isInt().withMessage('ID выставки должен быть числом')

];

const validateExhibition = [

body('title').trim().notEmpty().withMessage('Название выставки обязательно').isLength({ min: 2, max: 100 }).withMessage('Название должно быть от 2 до 100 символов'),

body('location').trim().notEmpty().withMessage('Место проведения обязательно').isLength({ min: 2, max: 100 }).withMessage('Место проведения должно быть от 2 до 100 символов'),

body('poster\_path').trim().notEmpty().withMessage('Путь к афише обязателен').isLength({ max: 255 }).withMessage('Путь к афише не должен превышать 255 символов'),

body('start\_date').isDate().withMessage('Неверный формат даты начала'),

body('end\_date').isDate().withMessage('Неверный формат даты окончания').custom((value, { req }) => {

if (new Date(value) <= new Date(req.body.start\_date)) {

throw new Error('Дата окончания должна быть позже даты начала');

}

return true;

}),

body('ticket\_price').isFloat({ min: 0 }).withMessage('Цена должна быть положительным числом'),

body('total\_tickets').isInt({ min: 0 }).withMessage('Количество билетов должно быть положительным числом'),

body('status').optional().isIn(['upcoming', 'active', 'completed']).withMessage('Неверный статус выставки'),

body('description').optional().trim().isLength({ max: 2000 }).withMessage('Описание не должно превышать 2000 символов'),

body('opening\_time').notEmpty().withMessage('Время открытия обязательно').matches(/^\d{2}:\d{2}$/).withMessage('Время открытия должно быть в формате ЧЧ:ММ'),

body('closing\_time').notEmpty().withMessage('Время закрытия обязательно').matches(/^\d{2}:\d{2}$/).withMessage('Время закрытия должно быть в формате ЧЧ:ММ').custom((value, { req }) => {

if (req.body.opening\_time && value <= req.body.opening\_time) {

throw new Error('Время закрытия должно быть позже времени открытия');

}

return true;

})

];

const validateExhibitionUpdate = [

body('title').optional().trim().isLength({ min: 2, max: 100 }).withMessage('Название должно быть от 2 до 100 символов'),

body('location').optional().trim().isLength({ min: 2, max: 100 }).withMessage('Место проведения должно быть от 2 до 100 символов'),

body('poster\_path').optional().trim().notEmpty().withMessage('Путь к афише обязателен').isLength({ max: 255 }).withMessage('Путь к афише не должен превышать 255 символов'),

body('start\_date').optional().isDate().withMessage('Неверный формат даты начала'),

body('end\_date').optional().isDate().withMessage('Неверный формат даты окончания').custom((value, { req }) => {

if (req.body.start\_date && new Date(value) <= new Date(req.body.start\_date)) {

throw new Error('Дата окончания должна быть позже даты начала');

}

return true;

}),

body('ticket\_price').optional().isFloat({ min: 0 }).withMessage('Цена должна быть положительным числом'),

body('total\_tickets').optional().isInt({ min: 0 }).withMessage('Количество билетов должно быть положительным числом'),

body('status').optional().isIn(['upcoming', 'active', 'completed']).withMessage('Неверный статус выставки'),

body('description').optional().trim().isLength({ max: 2000 }).withMessage('Описание не должно превышать 2000 символов'),

body('opening\_time').optional().matches(/^\d{2}:\d{2}$/).withMessage('Время открытия должно быть в формате ЧЧ:ММ'),

body('closing\_time').optional().matches(/^\d{2}:\d{2}$/).withMessage('Время закрытия должно быть в формате ЧЧ:ММ').custom((value, { req }) => {

if (req.body.opening\_time && value <= req.body.opening\_time) {

throw new Error('Время закрытия должно быть позже времени открытия');

}

return true;

})

];

const validateExhibitionId = [

param('id').isInt().withMessage('ID выставки должен быть числом')

];

const validateUserUpdate = [

body('surname').optional().trim().isLength({ min: 2, max: 50 }).withMessage('Фамилия должна быть от 2 до 50 символов'),

body('first\_name').optional().trim().isLength({ min: 2, max: 50 }).withMessage('Имя должно быть от 2 до 50 символов'),

body('patronymic').optional({ nullable: true }).trim().custom((value) => {

if (value === null || value === '') {

return true;

}

if (value.length < 2 || value.length > 50) {

throw new Error('Отчество должно быть от 2 до 50 символов');

}

return true;

}),

body('email').optional().trim().isEmail().withMessage('Неверный формат email'),

body('password').optional().trim().isLength({ min: 8 }).withMessage('Пароль должен быть не менее 8 символов'),

body('role').optional().isIn(['admin', 'manager', 'user']).withMessage('Неверная роль пользователя')

];

const validateUserId = [

param('id').isInt().withMessage('ID пользователя должен быть числом')

];

const validateTicket = [

body('exhibition\_id').toInt().isInt().withMessage('ID выставки должен быть числом'),

body('quantity').toInt().isInt({ min: 1 }).withMessage('Количество билетов должно быть положительным числом')

];

const validateTicketId = [

param('id').isInt().withMessage('ID билета должен быть числом')

];

module.exports = {

validate,

validateRegister,

validateLogin,

validatePasswordChange,

validateCreateUser,

validateStyle,

validateStyleUpdate,

validateStyleId,

validateGenre,

validateGenreUpdate,

validateGenreId,

validateAuthor,

validateAuthorUpdate,

validateAuthorId,

validateArtwork,

validateArtworkId,

validateArtworkUpdate,

validateExhibition,

validateExhibitionUpdate,

validateExhibitionId,

validateUserUpdate,

validateUserId,

validateTicket,

validateTicketId

};

ПРИЛОЖЕНИЕ 15. Файл src\models\Artwork.js

const { DataTypes } = require('sequelize');

module.exports = (sequelize) => {

return sequelize.define('Artwork', {

id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true

},

title: {

type: DataTypes.STRING(100),

allowNull: false,

validate: {

notEmpty: true

}

},

width: {

type: DataTypes.DECIMAL(10, 2),

allowNull: true,

validate: {

min: 0,

notEmpty: true

}

},

height: {

type: DataTypes.DECIMAL(10, 2),

allowNull: true,

validate: {

min: 0,

notEmpty: true

}

},

author\_id: {

type: DataTypes.INTEGER,

allowNull: false,

references: {

model: 'Author',

key: 'id'

}

},

creation\_year: {

type: DataTypes.INTEGER,

allowNull: true,

validate: {

isNotFuture(value) {

if (value && value > new Date().getFullYear()) {

throw new Error('Год создания не может быть в будущем');

}

},

notEmpty: true

}

},

style\_id: {

type: DataTypes.INTEGER,

allowNull: false,

references: {

model: 'Style',

key: 'id'

}

},

genre\_id: {

type: DataTypes.INTEGER,

allowNull: false,

references: {

model: 'Genre',

key: 'id'

}

},

description: {

type: DataTypes.TEXT,

allowNull: true

},

image\_path: {

type: DataTypes.STRING(255),

allowNull: false

}

}, {

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at'

});

};

ПРИЛОЖЕНИЕ 16. Файл src\models\Author.js

const { DataTypes } = require('sequelize');

module.exports = (sequelize) => {

return sequelize.define('Author', {

id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true

},

surname: {

type: DataTypes.STRING(50),

allowNull: false,

validate: {

notEmpty: true

}

},

first\_name: {

type: DataTypes.STRING(50),

allowNull: false,

validate: {

notEmpty: true

}

},

patronymic: {

type: DataTypes.STRING(50),

allowNull: true,

validate: {

notEmpty: true

}

},

date\_of\_birth: {

type: DataTypes.DATEONLY,

allowNull: false,

validate: {

isDate: true

}

},

date\_of\_death: {

type: DataTypes.DATEONLY,

allowNull: true,

validate: {

isDate: true,

isAfterBirth(value) {

if (value && this.date\_of\_birth && value <= this.date\_of\_birth) {

throw new Error('Дата смерти должна быть позже даты рождения');

}

}

}

}

}, {

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at'

});

};

ПРИЛОЖЕНИЕ 17. Файл src\models\Exhibition.js

const { DataTypes } = require('sequelize');

module.exports = (sequelize) => {

const Exhibition = sequelize.define('Exhibition', {

id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true

},

title: {

type: DataTypes.STRING(255),

allowNull: false,

validate: { notEmpty: true }

},

location: {

type: DataTypes.STRING(255),

allowNull: false,

validate: { notEmpty: true }

},

start\_date: {

type: DataTypes.DATE,

allowNull: false

},

end\_date: {

type: DataTypes.DATE,

allowNull: false

},

opening\_time: {

type: DataTypes.TIME,

allowNull: false,

defaultValue: '10:00:00'

},

closing\_time: {

type: DataTypes.TIME,

allowNull: false,

defaultValue: '18:00:00'

},

ticket\_price: {

type: DataTypes.DECIMAL(10, 2),

allowNull: false,

validate: { min: 0 }

},

total\_tickets: {

type: DataTypes.INTEGER,

allowNull: false,

validate: { min: 1 }

},

status: {

type: DataTypes.ENUM('upcoming', 'active', 'completed'),

allowNull: false,

defaultValue: 'upcoming'

},

description: {

type: DataTypes.TEXT,

allowNull: true

},

remaining\_tickets: {

type: DataTypes.INTEGER,

allowNull: false,

validate: { min: 0 }

},

poster\_path: {

type: DataTypes.STRING(255),

allowNull: false,

validate: { notEmpty: true }

}

}, {

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at',

hooks: {

beforeSave: async (exhibition) => {

const now = new Date();

const startDateTime = new Date(

exhibition.start\_date.toISOString().slice(0, 10) + 'T' + (exhibition.opening\_time || '10:00:00')

);

const endDateTime = new Date(

exhibition.end\_date.toISOString().slice(0, 10) + 'T' + (exhibition.closing\_time || '18:00:00')

);

if (now < startDateTime) {

exhibition.status = 'upcoming';

} else if (now >= startDateTime && now <= endDateTime) {

exhibition.status = 'active';

} else if (now > endDateTime) {

exhibition.status = 'completed';

}

}

},

validate: {

endDateAfterStartDate() {

if (this.end\_date <= this.start\_date) {

throw new Error('Дата окончания должна быть позже даты начала');

}

},

remainingTicketsNotExceedTotal() {

if (this.remaining\_tickets > this.total\_tickets) {

throw new Error('Количество оставшихся билетов не может превышать общее количество билетов');

}

}

}

});

return Exhibition;

};

ПРИЛОЖЕНИЕ 18. Файл src\models\ExhibitionArtwork.js

const { DataTypes } = require('sequelize');

module.exports = (sequelize) => {

return sequelize.define('ExhibitionArtwork', {

exhibition\_id: {

type: DataTypes.INTEGER,

allowNull: false,

references: {

model: 'Exhibition',

key: 'id'

}

},

artwork\_id: {

type: DataTypes.INTEGER,

allowNull: false,

references: {

model: 'Artwork',

key: 'id'

}

}

}, {

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at',

tableName: 'exhibition\_artwork',

indexes: [

{

unique: true,

fields: ['exhibition\_id', 'artwork\_id']

}

]

});

};

ПРИЛОЖЕНИЕ 19. Файл src\models\Genre.js

const { DataTypes } = require('sequelize');

module.exports = (sequelize) => {

return sequelize.define('Genre', {

id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true

},

name: {

type: DataTypes.STRING(50),

allowNull: false,

unique: 'name',

validate: {

notEmpty: true

}

}

}, {

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at'

});

};

ПРИЛОЖЕНИЕ 20. Файл src\models\index.js

const { Sequelize } = require('sequelize');

const config = require('../config/database');

const sequelize = new Sequelize(

config.database,

config.username,

config.password,

{

host: config.host,

port: config.port,

dialect: config.dialect,

pool: config.pool,

define: config.define,

logging: config.logging

}

);

// Импортируем модели

const UserModel = require('./User');

const AuthorModel = require('./Author');

const StyleModel = require('./Style');

const GenreModel = require('./Genre');

const ExhibitionModel = require('./Exhibition');

const ArtworkModel = require('./Artwork');

const TicketModel = require('./Ticket');

const ExhibitionArtworkModel = require('./ExhibitionArtwork');

// Инициализируем модели

const User = UserModel(sequelize);

const Author = AuthorModel(sequelize);

const Style = StyleModel(sequelize);

const Genre = GenreModel(sequelize);

const Exhibition = ExhibitionModel(sequelize);

const Artwork = ArtworkModel(sequelize);

const Ticket = TicketModel(sequelize);

const ExhibitionArtwork = ExhibitionArtworkModel(sequelize);

// Определение связей

Artwork.belongsTo(Author, { foreignKey: 'author\_id' });

Author.hasMany(Artwork, { foreignKey: 'author\_id' });

Artwork.belongsTo(Style, { foreignKey: 'style\_id' });

Style.hasMany(Artwork, { foreignKey: 'style\_id' });

Artwork.belongsTo(Genre, { foreignKey: 'genre\_id' });

Genre.hasMany(Artwork, { foreignKey: 'genre\_id' });

// Многие-ко-многим связь между Exhibition и Artwork

Artwork.belongsToMany(Exhibition, {

through: ExhibitionArtwork,

foreignKey: 'artwork\_id'

});

Exhibition.belongsToMany(Artwork, {

through: ExhibitionArtwork,

foreignKey: 'exhibition\_id'

});

Ticket.belongsTo(Exhibition, { foreignKey: 'exhibition\_id' });

Exhibition.hasMany(Ticket, { foreignKey: 'exhibition\_id' });

Ticket.belongsTo(User, { foreignKey: 'user\_id' });

User.hasMany(Ticket, { foreignKey: 'user\_id' });

module.exports = {

sequelize,

Sequelize,

User,

Author,

Style,

Genre,

Exhibition,

Artwork,

Ticket,

ExhibitionArtwork

};

ПРИЛОЖЕНИЕ 21. Файл src\models\Style.js

const { DataTypes } = require('sequelize');

module.exports = (sequelize) => {

return sequelize.define('Style', {

id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true

},

name: {

type: DataTypes.STRING(50),

allowNull: false,

unique: 'name',

validate: {

notEmpty: true

}

}

}, {

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at'

});

};

ПРИЛОЖЕНИЕ 22. Файл src\models\Ticket.js

const { DataTypes } = require('sequelize');

module.exports = (sequelize) => {

const Ticket = sequelize.define('Ticket', {

id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true

},

exhibition\_id: {

type: DataTypes.INTEGER,

allowNull: false,

references: {

model: 'Exhibition',

key: 'id'

}

},

user\_id: {

type: DataTypes.INTEGER,

allowNull: false,

references: {

model: 'User',

key: 'id'

}

},

quantity: {

type: DataTypes.INTEGER,

allowNull: false,

validate: {

min: 1

}

},

booking\_date: {

type: DataTypes.DATE,

allowNull: false,

defaultValue: DataTypes.NOW

},

total\_price: {

type: DataTypes.DECIMAL(10, 2),

allowNull: false

}

}, {

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at',

hooks: {

beforeCreate: async (ticket) => {

// Получаем информацию о выставке

const exhibition = await sequelize.models.Exhibition.findByPk(ticket.exhibition\_id);

if (!exhibition) {

throw new Error('Выставка не найдена');

}

if (exhibition.status === 'completed') {

throw new Error('Нельзя забронировать билеты на завершенную выставку');

}

if (exhibition.remaining\_tickets < ticket.quantity) {

throw new Error('Недостаточно доступных билетов');

}

// Рассчитываем общую стоимость

ticket.total\_price = exhibition.ticket\_price \* ticket.quantity;

},

afterCreate: async (ticket) => {

// Обновляем количество оставшихся билетов

await sequelize.models.Exhibition.decrement('remaining\_tickets', {

by: ticket.quantity,

where: { id: ticket.exhibition\_id }

});

}

}

});

return Ticket;

};

ПРИЛОЖЕНИЕ 23. Файл src\models\User.js

const {DataTypes} = require('sequelize');

const bcrypt = require('bcryptjs');

module.exports = (sequelize) => {

const User = sequelize.define('User', {

id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true

},

surname: {

type: DataTypes.STRING(50),

allowNull: false,

validate: {

notEmpty: true

}

},

first\_name: {

type: DataTypes.STRING(50),

allowNull: false,

validate: {

notEmpty: true

}

},

patronymic: {

type: DataTypes.STRING(50),

allowNull: true,

validate: {

notEmpty: true

}

},

email: {

type: DataTypes.STRING(50),

allowNull: false,

unique: 'email',

validate: {

isEmail: true,

notEmpty: true

}

},

password: {

type: DataTypes.STRING(255),

allowNull: false,

validate: {

notEmpty: true

}

},

role: {

type: DataTypes.ENUM('admin', 'manager', 'user'),

allowNull: false,

defaultValue: 'user'

}

}, {

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at',

hooks: {

beforeCreate: async (user) => {

if (user.password) {

user.password = await bcrypt.hash(user.password, 10);

}

},

beforeUpdate: async (user) => {

if (user.changed('password')) {

user.password = await bcrypt.hash(user.password, 10);

}

}

}

});

// Метод для проверки пароля

User.prototype.checkPassword = async function (candidatePassword) {

return await bcrypt.compare(candidatePassword, this.password);

};

return User;

};

ПРИЛОЖЕНИЕ 24. Файл src\public\js\admin.js

$(document).ready(function() {

// Обработка модальных окон

$('[data-modal]').on('click', function() {

const modalId = $(this).data('modal');

$(`#${modalId}`).addClass('active');

});

$('.modal-close').on('click', function() {

$(this).closest('.modal').removeClass('active');

});

// Закрытие модального окна при нажатии Escape

$(document).on('keydown', function(e) {

if (e.key === 'Escape') {

$('.modal').removeClass('active');

}

});

});

ПРИЛОЖЕНИЕ 25. Файл src\public\js\artworks.js

// Глобальные переменные

let artworkCurrentSort = {

column: 'id',

direction: 'asc'

};

document.addEventListener('DOMContentLoaded', function () {

const list = document.getElementById('artworks-list');

const noMsg = document.getElementById('no-artworks-message');

if (!list) return;

list.innerHTML = '<div style="grid-column:1/-1;text-align:center;padding:2rem;">Загрузка...</div>';

list.style.display = 'grid';

if (noMsg) noMsg.style.display = 'none';

// 1) Сначала получаем все картины

fetch('/api/v1/artworks')

.then(res => res.json())

.then(artJson => {

const artworks = (artJson.status === 'success' && Array.isArray(artJson.data.artworks))

? artJson.data.artworks

: [];

if (!artworks.length) {

list.style.display = 'none';

if (noMsg) noMsg.style.display = 'block';

return;

}

// 2) Параллельно для каждой картины запрашиваем её выставки

const exhibitionsPromises = artworks.map(a =>

fetch(`/api/v1/artworks/${a.id}/exhibitions`)

.then(r => r.json())

.then(json => (json.status === 'success' && Array.isArray(json.data.exhibitions))

? json.data.exhibitions

: []

)

);

// 3) Когда все запросы завершатся — рендерим карточки

Promise.all(exhibitionsPromises)

.then(allExhibitions => {

list.innerHTML = '';

list.style.display = 'grid';

if (noMsg) noMsg.style.display = 'none';

artworks.forEach((a, idx) => {

const card = document.createElement('div');

card.className = 'card';

card.style.display = 'flex';

card.style.flexDirection = 'column';

card.style.justifyContent = 'space-between';

// HTML для списка выставок (если их несколько — разделяем запятыми)

const exhibitions = allExhibitions[idx];

const exhibitionsHTML = exhibitions.length

? exhibitions

.map(e => `<a href="/exhibitions/${e.id}/page" class="text-blue-600 hover:underline">${e.title}</a>`)

.join(', ')

: '—';

card.innerHTML = `

<div class="card-header" style="padding:1rem;">

<h4 class="text-lg font-semibold">${a.title}</h4>

</div>

${a.image\_path

? `<img src="${a.image\_path}"

class="artwork-image"

style="width:100%; height:180px; object-fit:cover;"

alt="${a.title}">`

: ''}

<div class="card-body" style="padding:1rem; flex-grow:1;">

<p><strong>ID:</strong> ${a.id}</p>

<p><strong>Размеры:</strong> ${a.width} × ${a.height} см</p>

<p><strong>Год создания:</strong> ${a.creation\_year}</p>

<p><strong>Автор:</strong> ${a.Author.surname} ${a.Author.first\_name} ${a.Author.patronymic !== "null" ? a.Author.patronymic : ''}</p>

<p><strong>Стиль:</strong> ${a.Style.name}</p>

<p><strong>Жанр:</strong> ${a.Genre.name}</p>

<p class="mt-2"><strong>Описание:</strong> ${a.description || '—'}</p>

<p class="mt-2">

<strong>Выставки:</strong> ${exhibitionsHTML}

</p>

</div>

`;

list.appendChild(card);

});

})

.catch(err => {

console.error('Ошибка при загрузке выставок для картин:', err);

list.style.display = 'none';

if (noMsg) noMsg.style.display = 'block';

});

})

.catch(err => {

console.error('Ошибка при загрузке картин:', err);

list.style.display = 'none';

if (noMsg) noMsg.style.display = 'block';

});

});

$(document).ready(function () {

// Загрузка произведений при загрузке страницы

loadArtworks();

// Загрузка списков для выпадающих меню

loadAuthors();

loadStyles();

loadGenres();

// Переменная для хранения таймера

let searchTimer;

// Обработчик поиска при вводе

$('#searchInput').on('input', function () {

const searchTerm = $(this).val();

// Очищаем предыдущий таймер

clearTimeout(searchTimer);

// Устанавливаем новый таймер

searchTimer = setTimeout(() => {

loadArtworks(searchTerm);

}, 300); // Задержка 300мс

});

// Обработчики фильтрации по году создания

$('#yearFrom, #yearTo').on('input', function() {

loadArtworks();

});

// Обработчик очистки фильтра по году

$('#clearYearFilter').on('click', function() {

$('#yearFrom').val('');

$('#yearTo').val('');

loadArtworks();

});

// Обработчик клика по заголовкам таблицы

$('.sortable').on('click', function() {

const column = $(this).data('column');

if (artworkCurrentSort.column === column) {

artworkCurrentSort.direction = artworkCurrentSort.direction === 'asc' ? 'desc' : 'asc';

} else {

artworkCurrentSort.column = column;

artworkCurrentSort.direction = 'asc';

}

loadArtworks($('#searchInput').val());

});

// Обработчик добавления произведения

$('#saveArtworkBtn').on('click', function () {

const errors = validateAddArtworkForm();

if (errors.length > 0) {

showErrors(errors);

return;

}

const formData = new FormData();

formData.append('title', $('#artworkTitle').val().trim());

formData.append('author\_id', parseInt($('#artworkAuthor').val()));

formData.append('style\_id', parseInt($('#artworkStyle').val()));

formData.append('genre\_id', parseInt($('#artworkGenre').val()));

formData.append('creation\_year', parseInt($('#artworkCreationYear').val()));

formData.append('width', parseFloat($('#artworkWidth').val()));

formData.append('height', parseFloat($('#artworkHeight').val()));

formData.append('description', $('#artworkDescription').val().trim());

const imageFile = $('#artworkImage')[0].files[0];

if (imageFile) {

formData.append('image\_path', imageFile);

}

addArtwork(formData);

});

// Обработчик обновления произведения

$('#updateArtworkBtn').on('click', function () {

const errors = validateEditArtworkForm();

if (errors.length > 0) {

showErrors(errors);

return;

}

const formData = new FormData();

const id = $('#editArtworkId').val();

formData.append('title', $('#editArtworkTitle').val().trim());

formData.append('author\_id', parseInt($('#editArtworkAuthor').val()));

formData.append('style\_id', parseInt($('#editArtworkStyle').val()));

formData.append('genre\_id', parseInt($('#editArtworkGenre').val()));

formData.append('creation\_year', parseInt($('#editArtworkCreationYear').val()));

formData.append('width', parseFloat($('#editArtworkWidth').val()));

formData.append('height', parseFloat($('#editArtworkHeight').val()));

formData.append('description', $('#editArtworkDescription').val().trim());

const imageFile = $('#editArtworkImage')[0].files[0];

if (imageFile) {

formData.append('image\_path', imageFile);

}

updateArtwork(id, formData);

});

// Обработчик удаления произведения

$('#confirmDeleteBtn').on('click', function () {

const id = $(this).data('artworkId');

if (id) {

deleteArtwork(id);

}

});

// Обработчик предпросмотра изображения при загрузке

$('#artworkImage, #editArtworkImage').on('change', function() {

const formGroup = $(this).closest('.form-group');

let previewElement = formGroup.find('.image-preview');

if (previewElement.length === 0) {

previewElement = $('<div class="image-preview"></div>');

formGroup.append(previewElement);

}

const file = this.files[0];

if (file) {

const reader = new FileReader();

reader.onload = function(e) {

previewElement.html(`<img src="${e.target.result}" alt="Preview" style="max-width: 100%;">`);

};

reader.readAsDataURL(file);

}

});

// Обработчик клика по миниатюре для просмотра

$(document).on('click', '.artwork-thumbnail', function() {

const imageUrl = $(this).data('full-image');

$('#previewImage').attr('src', imageUrl);

$('#imagePreviewModal').addClass('active');

});

// Закрытие предпросмотра изображения (только по клику вне изображения и по Esc)

$('#imagePreviewModal').on('click', function(e) {

if ($(e.target).is('#imagePreviewModal')) {

$('#imagePreviewModal').removeClass('active');

$('#previewImage').attr('src', '');

$('body').removeClass('modal-open');

}

});

$(document).on('keydown', function(e) {

if (e.key === 'Escape') {

$('#imagePreviewModal').removeClass('active');

$('#previewImage').attr('src', '');

$('body').removeClass('modal-open');

}

});

});

// Функция загрузки произведений

function loadArtworks(searchTerm = "") {

// Получаем значения фильтров по году

const yearFrom = $('#yearFrom').val();

const yearTo = $('#yearTo').val();

// Формируем параметры запроса

const params = new URLSearchParams();

if (searchTerm) {

params.append('search', searchTerm.trim());

}

if (yearFrom) {

params.append('year\_from', yearFrom);

}

if (yearTo) {

params.append('year\_to', yearTo);

}

const queryString = params.toString();

const url = `/api/v1/artworks${queryString ? `?${queryString}` : ""}`;

$.ajax({

url: url,

method: 'GET',

success: function(response) {

if (response.status === "success") {

const tbody = $('#artworksTableBody');

tbody.empty();

let artworks = response.data.artworks;

artworks.sort((a, b) => {

let valueA, valueB;

switch(artworkCurrentSort.column) {

case 'id':

valueA = a.id;

valueB = b.id;

break;

case 'title':

valueA = a.title.toLowerCase();

valueB = b.title.toLowerCase();

break;

case 'author':

valueA = `${a.Author.surname} ${a.Author.first\_name} ${a.Author.patronymic || ''}`.toLowerCase();

valueB = `${b.Author.surname} ${b.Author.first\_name} ${b.Author.patronymic || ''}`.toLowerCase();

break;

case 'style':

valueA = a.Style.name.toLowerCase();

valueB = b.Style.name.toLowerCase();

break;

case 'genre':

valueA = a.Genre.name.toLowerCase();

valueB = b.Genre.name.toLowerCase();

break;

case 'creation\_year':

valueA = a.creation\_year;

valueB = b.creation\_year;

break;

case 'dimensions':

valueA = (a.width || 0) \* (a.height || 0);

valueB = (b.width || 0) \* (b.height || 0);

break;

case 'exhibitions':

valueA = a.statistics.total\_exhibitions;

valueB = b.statistics.total\_exhibitions;

break;

default:

valueA = a.id;

valueB = b.id;

break;

}

if (artworkCurrentSort.direction === 'asc') {

return valueA > valueB ? 1 : -1;

} else {

return valueA < valueB ? 1 : -1;

}

});

artworks.forEach(function(artwork) {

// Преобразуем путь к изображению

const imagePath = artwork.image\_path.split('/').pop(); // Получаем только имя файла

const imageUrl = `/media/${imagePath}`; // Путь относительно public

const row = `

<tr>

<td>${artwork.id}</td>

<td class="artwork-image-cell">

<img src="${imageUrl}"

alt="${artwork.title}"

class="artwork-thumbnail"

data-full-image="${imageUrl}">

</td>

<td>${artwork.title}</td>

<td>${artwork.Author.surname} ${artwork.Author.first\_name} ${artwork.Author.patronymic}</td>

<td>${artwork.Style.name}</td>

<td>${artwork.Genre.name}</td>

<td>${artwork.creation\_year || '-'}</td>

<td>${artwork.width ? artwork.width + '×' + artwork.height + ' см' : '-'}</td>

<td>${artwork.statistics.total\_exhibitions}</td>

<td>

<button class="btn btn-sm btn-primary me-2" onclick="editArtwork(${artwork.id})">

<i class="fas fa-edit"></i>Изменить

</button>

<button class="btn btn-sm btn-danger" onclick="confirmDelete(${artwork.id})">

<i class="fas fa-trash"></i>Удалить

</button>

</td>

</tr>

`;

tbody.append(row);

});

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else {

alert("Ошибка при загрузке произведений");

}

}

});

}

// Функция загрузки авторов для выпадающего списка

function loadAuthors() {

$.ajax({

url: '/api/v1/authors',

method: 'GET',

success: function(response) {

if (response.status === "success") {

const authors = response.data.authors;

const authorSelects = $('#artworkAuthor, #editArtworkAuthor');

authorSelects.empty();

authors.forEach(function(author) {

const option = `<option value="${author.id}">${author.surname} ${author.first\_name} ${author.patronymic !== null ? author.patronymic : ''}</option>`;

authorSelects.append(option);

});

}

}

});

}

// Функция загрузки стилей для выпадающего списка

function loadStyles() {

$.ajax({

url: '/api/v1/styles',

method: 'GET',

success: function(response) {

if (response.status === "success") {

const styles = response.data.styles;

const styleSelects = $('#artworkStyle, #editArtworkStyle');

styleSelects.empty();

styles.forEach(function(style) {

const option = `<option value="${style.id}">${style.name}</option>`;

styleSelects.append(option);

});

}

}

});

}

// Функция загрузки жанров для выпадающего списка

function loadGenres() {

$.ajax({

url: '/api/v1/genres',

method: 'GET',

success: function(response) {

if (response.status === "success") {

const genres = response.data.genres;

const genreSelects = $('#artworkGenre, #editArtworkGenre');

genreSelects.empty();

genres.forEach(function(genre) {

const option = `<option value="${genre.id}">${genre.name}</option>`;

genreSelects.append(option);

});

}

}

});

}

// Функция для очистки ошибок

function clearErrors() {

$('.error-message').text('');

$('.form-group').removeClass('error');

$('.form-group input, .form-group select, .form-group textarea').removeClass('error');

}

// Функция для отображения ошибок

function showErrors(errors) {

clearErrors();

if (errors && errors.length > 0) {

errors.forEach(error => {

const field = error.field;

const message = error.message;

const formGroup = $(`#${field}`).closest('.form-group');

formGroup.addClass('error');

$(`#${field}`).addClass('error');

$(`#${field}Error`).text(message);

});

}

}

// Функция валидации формы добавления произведения

function validateAddArtworkForm() {

const title = $('#artworkTitle').val().trim();

const author\_id = $('#artworkAuthor').val();

const style\_id = $('#artworkStyle').val();

const genre\_id = $('#artworkGenre').val();

const creation\_year = $('#artworkCreationYear').val();

const width = $('#artworkWidth').val();

const height = $('#artworkHeight').val();

const description = $('#artworkDescription').val().trim();

const image = $('#artworkImage')[0].files[0];

const errors = [];

if (!title) {

errors.push({ field: 'artworkTitle', message: 'Название произведения обязательно' });

} else if (title.length < 2 || title.length > 100) {

errors.push({ field: 'artworkTitle', message: 'Название должно быть от 2 до 100 символов' });

}

if (!author\_id) {

errors.push({ field: 'artworkAuthor', message: 'Выберите автора' });

}

if (!style\_id) {

errors.push({ field: 'artworkStyle', message: 'Выберите стиль' });

}

if (!genre\_id) {

errors.push({ field: 'artworkGenre', message: 'Выберите жанр' });

}

if (!creation\_year) {

errors.push({ field: 'artworkCreationYear', message: 'Год создания произведения обязателен' });

} else {

const year = parseInt(creation\_year);

if (isNaN(year) || year > new Date().getFullYear()) {

errors.push({ field: 'artworkCreationYear', message: 'Неверный год создания' });

}

}

if (!width) {

errors.push({ field: 'artworkWidth', message: 'Ширина произведения обязательна' });

} else {

const widthValue = parseFloat(width);

if (isNaN(widthValue) || widthValue <= 0) {

errors.push({ field: 'artworkWidth', message: 'Ширина должна быть положительным числом' });

}

}

if (!height) {

errors.push({ field: 'artworkHeight', message: 'Высота произведения обязательна' });

} else {

const heightValue = parseFloat(height);

if (isNaN(heightValue) || heightValue <= 0) {

errors.push({ field: 'artworkHeight', message: 'Высота должна быть положительным числом' });

}

}

if (description && description.length > 2000) {

errors.push({ field: 'artworkDescription', message: 'Описание не должно превышать 2000 символов' });

}

if (!image) {

errors.push({ field: 'artworkImage', message: 'Выберите изображение' });

}

return errors;

}

// Функция валидации формы редактирования произведения

function validateEditArtworkForm() {

const title = $('#editArtworkTitle').val().trim();

const author\_id = $('#editArtworkAuthor').val();

const style\_id = $('#editArtworkStyle').val();

const genre\_id = $('#editArtworkGenre').val();

const creation\_year = $('#editArtworkCreationYear').val();

const width = $('#editArtworkWidth').val();

const height = $('#editArtworkHeight').val();

const description = $('#editArtworkDescription').val().trim();

const image = $('#editArtworkImage')[0].files[0];

const errors = [];

if (!title) {

errors.push({ field: 'editArtworkTitle', message: 'Название обязательно' });

} else if (title.length < 2 || title.length > 100) {

errors.push({ field: 'editArtworkTitle', message: 'Название должно быть от 2 до 100 символов' });

}

if (!author\_id) {

errors.push({ field: 'editArtworkAuthor', message: 'Выберите автора' });

}

if (!style\_id) {

errors.push({ field: 'editArtworkStyle', message: 'Выберите стиль' });

}

if (!genre\_id) {

errors.push({ field: 'editArtworkGenre', message: 'Выберите жанр' });

}

if (creation\_year) {

const year = parseInt(creation\_year);

if (isNaN(year) || year > new Date().getFullYear()) {

errors.push({ field: 'editArtworkCreationYear', message: 'Неверный год создания' });

}

}

if (width) {

const widthValue = parseFloat(width);

if (isNaN(widthValue) || widthValue <= 0) {

errors.push({ field: 'editArtworkWidth', message: 'Ширина должна быть положительным числом' });

}

}

if (height) {

const heightValue = parseFloat(height);

if (isNaN(heightValue) || heightValue <= 0) {

errors.push({ field: 'editArtworkHeight', message: 'Высота должна быть положительным числом' });

}

}

if (description && description.length > 2000) {

errors.push({ field: 'editArtworkDescription', message: 'Описание не должно превышать 2000 символов' });

}

return errors;

}

// Функция добавления произведения

function addArtwork(formData) {

$.ajax({

url: '/api/v1/artworks',

method: 'POST',

data: formData,

processData: false,

contentType: false,

success: function(response) {

if (response.status === "success") {

$('#addArtworkModal').removeClass('active');

clearErrors();

$('#addArtworkForm')[0].reset();

$('#artworkImage').val('');

loadArtworks();

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const response = xhr.responseJSON;

if (response.errors && response.errors.length > 0) {

showErrors(response.errors);

} else {

alert(response.message || "Ошибка при добавлении произведения");

}

} else {

alert("Ошибка при добавлении произведения");

}

}

});

}

// Функция редактирования произведения

function editArtwork(id) {

clearErrors();

$.ajax({

url: `/api/v1/artworks/${id}`,

method: 'GET',

success: function(response) {

if (response.status === "success") {

const artwork = response.data.artwork;

$('#editArtworkId').val(artwork.id);

$('#editArtworkTitle').val(artwork.title);

$('#editArtworkAuthor').val(artwork.author\_id);

$('#editArtworkStyle').val(artwork.style\_id);

$('#editArtworkGenre').val(artwork.genre\_id);

$('#editArtworkCreationYear').val(artwork.creation\_year);

$('#editArtworkWidth').val(artwork.width);

$('#editArtworkHeight').val(artwork.height);

$('#editArtworkDescription').val(artwork.description);

// Преобразуем путь к изображению

const imagePath = artwork.image\_path.split('/').pop();

const imageUrl = `/media/${imagePath}`;

// Отображаем текущее изображение

const previewHtml = `

<img src="${imageUrl}" alt="${artwork.title}" style="max-width: 100%;">

`;

$('#currentImagePreview').html(previewHtml);

$('#editArtworkModal').addClass('active');

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else {

alert("Ошибка при загрузке данных произведения");

}

}

});

}

// Функция обновления произведения

function updateArtwork(id, formData) {

$.ajax({

url: `/api/v1/artworks/${id}`,

method: 'PATCH',

data: formData,

processData: false,

contentType: false,

success: function(response) {

if (response.status === "success") {

$('#editArtworkModal').removeClass('active');

clearErrors();

$('#editArtworkForm')[0].reset();

$('#editArtworkImage').val('');

$('#currentImagePreview').empty();

loadArtworks();

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const response = xhr.responseJSON;

if (response.errors && response.errors.length > 0) {

showErrors(response.errors);

} else {

alert(response.message || "Ошибка при обновлении произведения");

}

} else {

alert("Ошибка при обновлении произведения");

}

}

});

}

// Функция удаления произведения

function deleteArtwork(id) {

$.ajax({

url: `/api/v1/artworks/${id}`,

method: 'DELETE',

success: function(response) {

if (response.status === "success") {

$('#deleteArtworkModal').removeClass('active');

loadArtworks();

} else {

alert(response.message || "Ошибка при удалении произведения");

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const response = xhr.responseJSON;

if (response.data && response.data.exhibitions) {

const exhibitionsList = response.data.exhibitions

.map(ex => `"${ex.title}"`)

.join(', ');

alert(`Невозможно удалить произведение: оно участвует в выставках: ${exhibitionsList}`);

} else {

alert(response.message || "Невозможно удалить произведение");

}

} else if (xhr.status === 404) {

alert("Произв едение не найдено");

} else {

alert("Ошибка при удалении произведения");

}

}

});

}

// Функция подтверждения удаления

function confirmDelete(id) {

$('#confirmDeleteBtn').data('artworkId', id);

$('#deleteArtworkModal').addClass('active');

}

// Добавляем очистку ошибок при открытии модальных окон

$('[data-modal="addArtworkModal"]').on('click', function() {

clearErrors();

})  
*module*.exports.BookingService = BookingService;

ПРИЛОЖЕНИЕ 26. Файл src\public\js\authors.js

// Глобальные переменные

let authorCurrentSort = {

column: 'id',

direction: 'asc'

};

$(document).ready(function () {

// Загрузка авторов при загрузке страницы

loadAuthors();

// Переменная для хранения таймера

let searchTimer;

// Обработчик поиска при вводе

$('#searchInput').on('input', function () {

const searchTerm = $(this).val();

// Очищаем предыдущий таймер

clearTimeout(searchTimer);

// Устанавливаем новый таймер

searchTimer = setTimeout(() => {

loadAuthors(searchTerm);

}, 300); // Задержка 300мс

});

// Обработчики фильтрации по дате

$('#dateFrom, #dateTo').on('change', function() {

loadAuthors();

});

// Обработчик очистки фильтра по дате

$('#clearDateFilter').on('click', function() {

$('#dateFrom').val('');

$('#dateTo').val('');

loadAuthors();

});

// Обработчик клика по заголовкам таблицы

$('.sortable').on('click', function() {

const column = $(this).data('column');

// Если кликнули по текущему столбцу, меняем направление

if (authorCurrentSort.column === column) {

authorCurrentSort.direction = authorCurrentSort.direction === 'asc' ? 'desc' : 'asc';

} else {

// Если кликнули по новому столбцу, устанавливаем сортировку по возрастанию

authorCurrentSort.column = column;

authorCurrentSort.direction = 'asc';

}

// Перезагружаем данные

loadAuthors($('#searchInput').val());

});

// Обработчик добавления автора

$('#saveAuthorBtn').on('click', function () {

const errors = validateAddAuthorForm();

if (errors.length > 0) {

showErrors(errors);

return;

}

const surname = $('#authorSurname').val();

const first\_name = $('#authorFirstName').val();

const patronymic = $('#authorPatronymic').val();

const date\_of\_birth = $('#authorDateOfBirth').val();

const date\_of\_death = $('#authorDateOfDeath').val();

addAuthor(surname, first\_name, patronymic, date\_of\_birth, date\_of\_death);

});

// Обработчик обновления автора

$('#updateAuthorBtn').on('click', function () {

const errors = validateEditAuthorForm();

if (errors.length > 0) {

showErrors(errors);

return;

}

const id = $('#editAuthorId').val();

const surname = $('#editAuthorSurname').val();

const first\_name = $('#editAuthorFirstName').val();

const patronymic = $('#editAuthorPatronymic').val();

const date\_of\_birth = $('#editAuthorDateOfBirth').val();

const date\_of\_death = $('#editAuthorDateOfDeath').val();

updateAuthor(id, surname, first\_name, patronymic, date\_of\_birth, date\_of\_death);

});

// Обработчик удаления автора

$('#confirmDeleteBtn').on('click', function () {

const id = $(this).data('authorId');

if (id) {

deleteAuthor(id);

}

});

});

// Функция загрузки авторов

function loadAuthors(searchTerm = "") {

// Получаем значения фильтров по дате

const dateFrom = $('#dateFrom').val();

const dateTo = $('#dateTo').val();

// Формируем параметры запроса

const params = new URLSearchParams();

if (searchTerm) {

params.append('name', searchTerm.trim());

}

if (dateFrom) {

params.append('date\_from', dateFrom);

}

if (dateTo) {

params.append('date\_to', dateTo);

}

const queryString = params.toString();

const url = `/api/v1/authors${queryString ? `?${queryString}` : ""}`;

$.ajax({

url: url,

method: 'GET',

success: function(response) {

if (response.status === "success") {

const tbody = $('#authorsTableBody');

tbody.empty();

// Сортируем данные на клиенте

let authors = response.data.authors;

authors.sort((a, b) => {

let valueA, valueB;

switch(authorCurrentSort.column) {

case 'id':

valueA = a.id;

valueB = b.id;

break;

case 'surname':

valueA = a.surname.toLowerCase();

valueB = b.surname.toLowerCase();

break;

case 'first\_name':

valueA = a.first\_name.toLowerCase();

valueB = b.first\_name.toLowerCase();

break;

case 'patronymic':

valueA = (a.patronymic || '').toLowerCase();

valueB = (b.patronymic || '').toLowerCase();

break;

case 'date\_of\_birth':

valueA = a.date\_of\_birth;

valueB = b.date\_of\_birth;

break;

case 'date\_of\_death':

valueA = a.date\_of\_death;

valueB = b.date\_of\_death;

break;

case 'artworks':

valueA = a.statistics.total\_artworks;

valueB = b.statistics.total\_artworks;

break;

default:

valueA = a.surname.toLowerCase();

valueB = b.surname.toLowerCase();

break;

}

if (authorCurrentSort.direction === 'asc') {

return valueA > valueB ? 1 : -1;

} else {

return valueA < valueB ? 1 : -1;

}

});

authors.forEach(function(author) {

const row = `

<tr>

<td>${author.id}</td>

<td>${author.surname}</td>

<td>${author.first\_name}</td>

<td>${author.patronymic || ''}</td>

<td>${formatDate(author.date\_of\_birth)}</td>

<td>${formatDate(author.date\_of\_death)}</td>

<td>${author.statistics.total\_artworks}</td>

<td>

<button class="btn btn-sm btn-primary me-2" onclick="editAuthor(${author.id},

'${author.surname}', '${author.first\_name}', '${author.patronymic || ''}',

'${author.date\_of\_birth}', '${author.date\_of\_death}')">

<i class="fas fa-edit"></i>Изменить

</button>

<button class="btn btn-sm btn-danger" onclick="confirmDelete(${author.id})">

<i class="fas fa-trash"></i>Удалить

</button>

</td>

</tr>

`;

tbody.append(row);

});

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else {

alert("Ошибка при загрузке авторов");

}

}

});

}

// Функция добавления автора

function addAuthor(surname, first\_name, patronymic, date\_of\_birth, date\_of\_death) {

$.ajax({

url: '/api/v1/authors',

method: 'POST',

contentType: 'application/json',

data: JSON.stringify({ surname, first\_name, patronymic, date\_of\_birth, date\_of\_death }),

success: function(response) {

if (response.status === "success") {

$('#addAuthorModal').removeClass('active');

clearErrors();

$('#authorSurname').val('');

$('#authorFirstName').val('');

$('#authorPatronymic').val('');

$('#authorDateOfBirth').val('');

$('#authorDateOfDeath').val('');

loadAuthors();

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const response = xhr.responseJSON;

if (response.errors && response.errors.length > 0) {

showErrors(response.errors);

} else {

alert(response.message || "Ошибка при добавлении автора");

}

} else {

alert("Ошибка при добавлении автора");

}

}

});

}

// Функция редактирования автора

function editAuthor(id, surname, first\_name, patronymic, date\_of\_birth, date\_of\_death) {

clearErrors();

$('#editAuthorId').val(id);

$('#editAuthorSurname').val(surname);

$('#editAuthorFirstName').val(first\_name);

$('#editAuthorPatronymic').val(patronymic);

$('#editAuthorDateOfBirth').val(date\_of\_birth);

$('#editAuthorDateOfDeath').val(date\_of\_death);

$('#editAuthorModal').addClass('active');

}

// Функция обновления автора

function updateAuthor(id, surname, first\_name, patronymic, date\_of\_birth, date\_of\_death) {

// Преобразуем пустую строку в null для отчества

const patronymicValue = patronymic.trim() === '' ? null : patronymic;

$.ajax({

url: `/api/v1/authors/${id}`,

method: 'PATCH',

contentType: 'application/json',

data: JSON.stringify({

surname,

first\_name,

patronymic: patronymicValue,

date\_of\_birth,

date\_of\_death

}),

success: function(response) {

if (response.status === "success") {

$('#editAuthorModal').removeClass('active');

clearErrors();

loadAuthors();

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const response = xhr.responseJSON;

if (response.errors && response.errors.length > 0) {

showErrors(response.errors);

} else {

alert(response.message || "Ошибка при обновлении автора");

}

} else {

alert("Ошибка при обновлении автора");

}

}

});

}

// Функция удаления автора

function deleteAuthor(id) {

$.ajax({

url: `/api/v1/authors/${id}`,

method: 'DELETE',

success: function(response) {

if (response.status === "success") {

// Закрываем модальное окно

$('#deleteAuthorModal').removeClass('active');

// Перезагружаем список авторов

loadAuthors();

} else {

alert(response.message || "Ошибка при удалении автора");

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const response = xhr.responseJSON;

if (response.errors && response.errors.length > 0) {

// Обработка ошибок валидации

const errorMessage = response.errors.map(err => err.msg).join('\n');

alert(errorMessage);

} else if (response.data && response.data.artworksCount) {

alert(`Невозможно удалить автора: он используется в ${response.data.artworksCount} произведениях`);

} else {

alert(response?.message || "Невозможно удалить автора");

}

} else if (xhr.status === 404) {

alert("Автор не найден");

} else {

alert("Ошибка при удалении автора");

}

}

});

}

// Функция подтверждения удаления

function confirmDelete(id) {

$('#confirmDeleteBtn').data('authorId', id);

$('#deleteAuthorModal').addClass('active');

}

// Функция для очистки ошибок

function clearErrors() {

$('.error-message').text('');

$('.form-group').removeClass('error');

$('.form-group input').removeClass('error');

}

// Функция для отображения ошибок

function showErrors(errors) {

clearErrors();

if (errors && errors.length > 0) {

errors.forEach(error => {

const field = error.field;

const message = error.message;

const formGroup = $(`#${field}`).closest('.form-group');

formGroup.addClass('error');

$(`#${field}`).addClass('error');

$(`#${field}Error`).text(message);

});

}

}

// Функция валидации формы добавления автора

function validateAddAuthorForm() {

const surname = $('#authorSurname').val().trim();

const first\_name = $('#authorFirstName').val().trim();

const patronymic = $('#authorPatronymic').val().trim();

const date\_of\_birth = $('#authorDateOfBirth').val();

const date\_of\_death = $('#authorDateOfDeath').val();

const errors = [];

if (!surname) {

errors.push({ field: 'authorSurname', message: 'Фамилия обязательна' });

} else if (surname.length < 2 || surname.length > 50) {

errors.push({ field: 'authorSurname', message: 'Фамилия должна быть от 2 до 50 символов' });

}

if (!first\_name) {

errors.push({ field: 'authorFirstName', message: 'Имя обязательно' });

} else if (first\_name.length < 2 || first\_name.length > 50) {

errors.push({ field: 'authorFirstName', message: 'Имя должно быть от 2 до 50 символов' });

}

if (patronymic && (patronymic.length < 2 || patronymic.length > 50)) {

errors.push({ field: 'authorPatronymic', message: 'Отчество должно быть от 2 до 50 символов' });

}

if (!date\_of\_birth) {

errors.push({ field: 'authorDateOfBirth', message: 'Дата рождения обязательна' });

}

if (!date\_of\_death) {

errors.push({ field: 'authorDateOfDeath', message: 'Дата смерти обязательна' });

} else if (new Date(date\_of\_death) <= new Date(date\_of\_birth)) {

errors.push({ field: 'authorDateOfDeath', message: 'Дата смерти должна быть позже даты рождения' });

}

return errors;

}

// Функция валидации формы редактирования автора

function validateEditAuthorForm() {

const surname = $('#editAuthorSurname').val().trim();

const first\_name = $('#editAuthorFirstName').val().trim();

const patronymic = $('#editAuthorPatronymic').val().trim();

const date\_of\_birth = $('#editAuthorDateOfBirth').val();

const date\_of\_death = $('#editAuthorDateOfDeath').val();

const errors = [];

if (!surname) {

errors.push({ field: 'editAuthorSurname', message: 'Фамилия обязательна' });

} else if (surname.length < 2 || surname.length > 50) {

errors.push({ field: 'editAuthorSurname', message: 'Фамилия должна быть от 2 до 50 символов' });

}

if (!first\_name) {

errors.push({ field: 'editAuthorFirstName', message: 'Имя обязательно' });

} else if (first\_name.length < 2 || first\_name.length > 50) {

errors.push({ field: 'editAuthorFirstName', message: 'Имя должно быть от 2 до 50 символов' });

}

if (patronymic && (patronymic.length < 2 || patronymic.length > 50)) {

errors.push({ field: 'editAuthorPatronymic', message: 'Отчество должно быть от 2 до 50 символов' });

}

if (!date\_of\_birth) {

errors.push({ field: 'editAuthorDateOfBirth', message: 'Дата рождения обязательна' });

}

if (!date\_of\_death) {

errors.push({ field: 'editAuthorDateOfDeath', message: 'Дата смерти обязательна' });

} else if (new Date(date\_of\_death) <= new Date(date\_of\_birth)) {

errors.push({ field: 'editAuthorDateOfDeath', message: 'Дата смерти должна быть позже даты рождения' });

}

return errors;

}

// Добавляем очистку ошибок при открытии модальных окон

$('[data-modal="addAuthorModal"]').on('click', function() {

clearErrors();

});  
*module*.exports.CarService = CarService;

ПРИЛОЖЕНИЕ 27. Файл src\public\js\exhibition.js

// Глобальные переменные

let exhibitionCurrentSort = {

column: 'id',

direction: 'asc'

};

$(document).ready(function () {

// Загрузка выставок при загрузке страницы

loadExhibitions();

// Загрузка списка произведений

loadArtworks();

// Переменная для хранения таймера

let searchTimer;

// Обработчик поиска при вводе

$('#searchInput').on('input', function () {

const searchTerm = $(this).val();

// Очищаем предыдущий таймер

clearTimeout(searchTimer);

// Устанавливаем новый таймер

searchTimer = setTimeout(() => {

loadExhibitions(searchTerm);

}, 300); // Задержка 300мс

});

// Обработчики фильтрации

$('#dateFrom, #dateTo').on('change', function() {

loadExhibitions();

});

$('#statusFilter').on('change', function() {

loadExhibitions();

});

$('#priceFrom, #priceTo').on('input', function() {

loadExhibitions();

});

// Обработчики очистки отдельных фильтров

$('#clearDateFilter').on('click', function() {

$('#dateFrom').val('');

$('#dateTo').val('');

loadExhibitions();

});

$('#clearStatusFilter').on('click', function() {

$('#statusFilter').val([]);

loadExhibitions();

});

$('#clearPriceFilter').on('click', function() {

$('#priceFrom').val('');

$('#priceTo').val('');

loadExhibitions();

});

// Обработчик клика по заголовкам таблицы

$('.sortable').on('click', function() {

const column = $(this).data('column');

if (exhibitionCurrentSort.column === column) {

exhibitionCurrentSort.direction = exhibitionCurrentSort.direction === 'asc' ? 'desc' : 'asc';

} else {

exhibitionCurrentSort.column = column;

exhibitionCurrentSort.direction = 'asc';

}

loadExhibitions($('#searchInput').val());

});

// Обработчик добавления выставки

$('#saveExhibitionBtn').off('click').on('click', function(e) {

e.preventDefault(); const errors = validateAddExhibitionForm();

if (errors.length) { showErrors(errors); return; }

const formData = new FormData();

formData.append('title', $('#exhibitionTitle').val().trim());

formData.append('location', $('#exhibitionLocation').val().trim());

formData.append('start\_date', $('#exhibitionStartDate').val());

formData.append('end\_date', $('#exhibitionEndDate').val());

formData.append('opening\_time', $('#exhibitionOpeningTime').val());

formData.append('closing\_time', $('#exhibitionClosingTime').val());

formData.append('status', $('#exhibitionStatus').val());

formData.append('ticket\_price', parseFloat($('#exhibitionTicketPrice').val()));

formData.append('total\_tickets', parseInt($('#exhibitionTotalTickets').val(), 10));

formData.append('description', $('#exhibitionDescription').val().trim());

// Правильная передача массива artwork\_ids[]

const artworkIds = getCheckedArtworkIds('#exhibitionArtworksCheckboxes');

artworkIds.forEach(id => formData.append('artwork\_ids[]', id));

const posterFile = $('#exhibitionPoster')[0].files[0];

if (posterFile) {

formData.append('poster', posterFile);

// и под хранить poster\_path, чтобы express-validator не жаловался:

formData.append('poster\_path', '/media/' + posterFile.name);

}

// Логи для отладки

console.group('Exhibition FormData');

for (const [key, value] of formData.entries()) console.log(key, value);

console.groupEnd();

addExhibition(formData);

});

// Обработчик обновления выставки

$('#updateExhibitionBtn').on('click', function () {

const errors = validateEditExhibitionForm();

if (errors.length > 0) {

showErrors(errors);

return;

}

const id = $('#editExhibitionId').val();

const formData = new FormData();

formData.append('title', $('#editExhibitionTitle').val().trim());

formData.append('location', $('#editExhibitionLocation').val().trim());

formData.append('start\_date', $('#editExhibitionStartDate').val());

formData.append('end\_date', $('#editExhibitionEndDate').val());

formData.append('opening\_time', $('#editExhibitionOpeningTime').val());

formData.append('closing\_time', $('#editExhibitionClosingTime').val());

formData.append('status', $('#editExhibitionStatus').val());

formData.append('ticket\_price', parseFloat($('#editExhibitionTicketPrice').val()));

formData.append('total\_tickets', parseInt($('#editExhibitionTotalTickets').val()));

formData.append('description', $('#editExhibitionDescription').val().trim());

getCheckedArtworkIds('#editExhibitionArtworksCheckboxes')

.forEach(id => formData.append('artwork\_ids[]', id)); const posterFile = $('#editExhibitionPoster')[0].files[0];

if (posterFile) {

formData.append('poster', posterFile);

}

updateExhibition(id, formData);

});

// Предпросмотр афиши при выборе файла (добавление)

$('#exhibitionPoster').on('change', function() {

const file = this.files[0];

const preview = $('#addPosterPreview');

preview.empty();

if (file) {

const reader = new FileReader();

reader.onload = function(e) {

preview.html(`<img src="${e.target.result}" alt="Preview" style="max-width: 100%;">`);

};

reader.readAsDataURL(file);

}

});

// Предпросмотр афиши при выборе файла (редактирование)

$('#editExhibitionPoster').on('change', function() {

const file = this.files[0];

const preview = $('#editPosterPreview');

preview.empty();

if (file) {

const reader = new FileReader();

reader.onload = function(e) {

preview.html(`<img src="${e.target.result}" alt="Preview" style="max-width: 100%;">`);

};

reader.readAsDataURL(file);

}

});

// Обработчик удаления выставки

$('#confirmDeleteBtn').on('click', function () {

const id = $(this).data('exhibitionId');

if (id) {

deleteExhibition(id);

}

});

// Делегирование: клик по строке таблицы

$('#exhibitionsTableBody').on('click', '.exhibition-row', function(e) {

if ($(e.target).closest('button').length || $(e.target).hasClass('poster-thumbnail')) return;

const exhibitionId = $(this).data('exhibition-id');

showExhibitionDetails(exhibitionId);

});

// Делегирование: клик по миниатюре афиши

$('#exhibitionsTableBody').on('click', '.poster-thumbnail', function(e) {

e.stopPropagation();

const posterUrl = $(this).data('full-poster');

$('#previewImage').attr('src', posterUrl);

$('#imagePreviewModal').addClass('active');

$('body').addClass('modal-open');

});

// Добавляем обработчик предпросмотра афиши

$('.poster-thumbnail').on('click', function(e) {

e.stopPropagation();

const posterUrl = $(this).data('full-poster');

$('#previewImage').attr('src', posterUrl);

$('#imagePreviewModal').addClass('active');

$('body').addClass('modal-open');

});

});

// Функция загрузки выставок

function loadExhibitions(searchTerm = "") {

// Получаем значения всех фильтров

const dateFrom = $('#dateFrom').val();

const dateTo = $('#dateTo').val();

const statusFilter = $('#statusFilter').val();

const priceFrom = $('#priceFrom').val();

const priceTo = $('#priceTo').val();

// Формируем параметры запроса

const params = new URLSearchParams();

if (searchTerm) {

params.append('search', searchTerm.trim());

}

if (dateFrom) {

params.append('start\_date', dateFrom);

}

if (dateTo) {

params.append('end\_date', dateTo);

}

if (statusFilter && statusFilter.length > 0) {

statusFilter.forEach(status => {

params.append('status', status);

});

}

if (priceFrom) {

params.append('min\_price', priceFrom);

}

if (priceTo) {

params.append('max\_price', priceTo);

}

const queryString = params.toString();

const url = `/api/v1/exhibitions${queryString ? `?${queryString}` : ""}`;

$.ajax({

url: url,

method: 'GET',

success: function(response) {

if (response.status === "success") {

const tbody = $('#exhibitionsTableBody');

tbody.empty();

let exhibitions = response.data.exhibitions;

exhibitions.sort((a, b) => {

let valueA, valueB;

switch(exhibitionCurrentSort.column) {

case 'id':

valueA = a.id;

valueB = b.id;

break;

case 'title':

valueA = a.title.toLowerCase();

valueB = b.title.toLowerCase();

break;

case 'location':

valueA = a.location.toLowerCase();

valueB = b.location.toLowerCase();

break;

case 'start\_date':

valueA = new Date(a.start\_date);

valueB = new Date(b.start\_date);

break;

case 'end\_date':

valueA = new Date(a.end\_date);

valueB = new Date(b.end\_date);

break;

case 'status':

valueA = a.status.toLowerCase();

valueB = b.status.toLowerCase();

break;

case 'ticket\_price':

valueA = a.ticket\_price;

valueB = b.ticket\_price;

break;

default:

valueA = a.id;

valueB = b.id;

break;

}

if (exhibitionCurrentSort.direction === 'asc') {

return valueA > valueB ? 1 : -1;

} else {

return valueA < valueB ? 1 : -1;

}

});

exhibitions.forEach(function(exhibition) {

// Преобразуем путь к афише

const posterPath = exhibition.poster\_path ? exhibition.poster\_path.split('/').pop() : '';

const posterUrl = posterPath ? `/media/${posterPath}` : '';

const posterCell = posterUrl ? `<img src="${posterUrl}" alt="Афиша" class="poster-thumbnail" data-full-poster="${posterUrl}" style="max-width: 60px; max-height: 60px; cursor: pointer;">` : '';

const row = `

<tr class="exhibition-row" data-exhibition-id="${exhibition.id}">

<td>${exhibition.id}</td>

<td>${posterCell}</td>

<td>${exhibition.title}</td>

<td>${exhibition.location}</td>

<td>${formatDate(exhibition.start\_date)}</td>

<td>${formatDate(exhibition.end\_date)}</td>

<td>${formatStatus(exhibition.status)}</td>

<td>${exhibition.ticket\_price} ₽</td>

<td>${exhibition.remaining\_tickets}/${exhibition.total\_tickets}</td>

<td>

<button class="btn btn-sm btn-primary me-2" onclick="editExhibition(${exhibition.id})">

<i class="fas fa-edit"></i>Изменить

</button>

<button class="btn btn-sm btn-danger" onclick="confirmDelete(${exhibition.id})">

<i class="fas fa-trash"></i>Удалить

</button>

</td>

</tr>

`;

tbody.append(row);

});

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else {

alert("Ошибка при загрузке выставок");

}

}

});

}

// --- Новый способ выбора картин через чекбоксы ---

function renderArtworksCheckboxes(artworks, containerId, selectedIds = []) {

const container = $(containerId);

container.empty();

artworks.forEach(artwork => {

const checked = selectedIds.includes(artwork.id) ? 'checked' : '';

// ФИО автора полностью

const author = artwork.Author;

const authorFio = [author.surname, author.first\_name, author.patronymic].filter(Boolean).join(' ');

const label = `${artwork.title} (${authorFio})`;

// Выровнять чекбоксы по правому краю

const checkbox = `<label class="artwork-checkbox-item" style="display:flex;align-items:center;justify-content:space-between;margin-bottom:4px;cursor:pointer;gap:8px;">

<span style="flex:1;">${label}</span>

<input type="checkbox" value="${artwork.id}" style="margin-left:8px;width:18px;height:18px;" ${checked} />

</label>`;

container.append(checkbox);

});

}

function getCheckedArtworkIds(containerId) {

return $(containerId + ' input[type="checkbox"]:checked')

.map(function() { return Number($(this).val()); })

.get();

}

// --- Переопределяю loadArtworks ---

function loadArtworks(selectedAdd = [], selectedEdit = []) {

$.ajax({

url: '/api/v1/artworks',

method: 'GET',

success: function(response) {

if (response.status === "success") {

const artworks = response.data.artworks;

renderArtworksCheckboxes(artworks, '#exhibitionArtworksCheckboxes', selectedAdd);

renderArtworksCheckboxes(artworks, '#editExhibitionArtworksCheckboxes', selectedEdit);

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else {

alert("Ошибка при загрузке списка произведений");

}

}

});

}

// Функция добавления выставки

function addExhibition(formData) {

$.ajax({

url: '/api/v1/exhibitions',

method: 'POST',

data: formData,

processData: false,

contentType: false,

success: function(response) {

if (response.status === "success") {

$('#addExhibitionModal').removeClass('active');

clearErrors();

$('#addExhibitionForm')[0].reset();

$('#exhibitionPoster').val('');

$('#addPosterPreview').empty();

loadExhibitions();

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const response = xhr.responseJSON;

if (response.errors && response.errors.length > 0) {

showErrors(response.errors);

} else {

alert(response.message || "Ошибка при добавлении выставки");

}

} else {

alert("Ошибка при добавлении выставки");

}

}

});

}

// Функция редактирования выставки

function editExhibition(id) {

clearErrors();

$.ajax({

url: `/api/v1/exhibitions/${id}`,

method: 'GET',

success: function(response) {

if (response.status === "success") {

const exhibition = response.data.exhibition;

$('#editExhibitionId').val(exhibition.id);

$('#editExhibitionTitle').val(exhibition.title);

$('#editExhibitionLocation').val(exhibition.location);

$('#editExhibitionStartDate').val(formatDateForInput(exhibition.start\_date));

$('#editExhibitionEndDate').val(formatDateForInput(exhibition.end\_date));

$('#editExhibitionOpeningTime').val(exhibition.opening\_time ? exhibition.opening\_time.slice(0,5) : '10:00');

$('#editExhibitionClosingTime').val(exhibition.closing\_time ? exhibition.closing\_time.slice(0,5) : '18:00');

$('#editExhibitionStatus').val(exhibition.status);

$('#editExhibitionTicketPrice').val(exhibition.ticket\_price);

$('#editExhibitionDescription').val(exhibition.description);

$('#editExhibitionTotalTickets').val(exhibition.total\_tickets);

// Устанавливаем выбранные произведения через чекбоксы

const selectedArtworkIds = exhibition.Artworks.map(artwork => artwork.id);

loadArtworks([], selectedArtworkIds);

$('#editExhibitionModal').addClass('active');

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else {

alert("Ошибка при загрузке данных выставки");

}

}

});

}

// Функция обновления выставки

function updateExhibition(id, formData) {

$.ajax({

url: `/api/v1/exhibitions/${id}`,

method: 'PATCH',

data: formData,

processData: false,

contentType: false,

success: function(response) {

if (response.status === "success") {

$('#editExhibitionModal').removeClass('active');

clearErrors();

$('#editExhibitionForm')[0].reset();

$('#editExhibitionPoster').val('');

$('#editPosterPreview').empty();

loadExhibitions();

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const response = xhr.responseJSON;

if (response.errors && response.errors.length > 0) {

showErrors(response.errors);

} else {

alert(response.message || "Ошибка при обновлении выставки");

}

} else {

alert("Ошибка при обновлении выставки");

}

}

});

}

// Функция удаления выставки

function deleteExhibition(id) {

$.ajax({

url: `/api/v1/exhibitions/${id}`,

method: 'DELETE',

success: function(response) {

if (response.status === "success") {

$('#deleteExhibitionModal').removeClass('active');

loadExhibitions();

} else {

alert(response.message || "Ошибка при удалении выставки");

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const response = xhr.responseJSON;

if (response.data && response.data.tickets) {

alert("Невозможно удалить выставку: на неё были забронированы билеты");

} else {

alert(response.message || "Невозможно удалить выставку");

}

} else if (xhr.status === 404) {

alert("Выставка не найдена");

} else {

alert("Ошибка при удалении выставки");

}

}

});

}

// Функция подтверждения удаления

function confirmDelete(id) {

$('#confirmDeleteBtn').data('exhibitionId', id);

$('#deleteExhibitionModal').addClass('active');

}

// Вспомогательные функции форматирования

function formatDate(dateString) {

const date = new Date(dateString);

return date.toLocaleDateString('ru-RU');

}

function formatDateForInput(dateString) {

const date = new Date(dateString);

return date.toISOString().split('T')[0];

}

function formatStatus(status) {

const statusMap = {

'upcoming': 'Запланирована',

'active': 'Идёт',

'completed': 'Завершена'

};

return statusMap[status] || status;

}

// Функция для отображения детальной информации о выставке

function showExhibitionDetails(id) {

$.ajax({

url: `/api/v1/exhibitions/${id}`,

method: 'GET',

success: function(response) {

if (response.status === "success") {

const exhibition = response.data.exhibition;

// Заполняем основную информацию

$('#detailId').text(exhibition.id);

$('#detailTitle').text(exhibition.title);

$('#detailLocation').text(exhibition.location);

$('#detailStartDate').text(formatDate(exhibition.start\_date));

$('#detailEndDate').text(formatDate(exhibition.end\_date));

$('#detailWorkingHours').text(`${formatTime(exhibition.opening\_time)} - ${formatTime(exhibition.closing\_time)}`);

$('#detailStatus').text(formatStatus(exhibition.status));

$('#detailTicketPrice').text(`${exhibition.ticket\_price} ₽`);

$('#detailDescription').text(exhibition.description || 'Нет описания');

// Заполняем список произведений

const artworksList = exhibition.Artworks.map(artwork => {

const authorName = artwork.Author ?

`${artwork.Author.surname} ${artwork.Author.first\_name} ${artwork.Author.patronymic}` :

'Автор не указан';

return `<div class="artwork-item">

<strong>${artwork.title}</strong> (${authorName})

<br>

<small>Год создания: ${artwork.creation\_year}</small>

</div>`;

}).join('');

$('#detailArtworks').html(artworksList || 'Нет произведений');

// Заполняем статистику по билетам

const ticketsInfo = `Всего билетов: ${exhibition.total\_tickets}, Доступно: ${exhibition.remaining\_tickets}`;

$('#detailTickets').html(ticketsInfo);

// Показываем модальное окно

$('#exhibitionDetailsModal').addClass('active');

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else {

alert("Ошибка при загрузке информации о выставке");

}

}

});

}

// Функция для очистки ошибок

function clearErrors() {

$('.error-message').text('');

$('.form-group').removeClass('error');

$('.form-group input, .form-group select, .form-group textarea').removeClass('error');

}

// Функция для отображения ошибок

function showErrors(errors) {

clearErrors();

if (errors && errors.length > 0) {

errors.forEach(error => {

const field = error.field;

const message = error.message;

const formGroup = $(`#${field}`).closest('.form-group');

formGroup.addClass('error');

$(`#${field}`).addClass('error');

$(`#${field}Error`).text(message);

});

}

}

// Функция валидации формы добавления выставки

function validateAddExhibitionForm() {

const title = $('#exhibitionTitle').val().trim();

const location = $('#exhibitionLocation').val().trim();

const start\_date = $('#exhibitionStartDate').val();

const end\_date = $('#exhibitionEndDate').val();

const opening\_time = $('#exhibitionOpeningTime').val();

const closing\_time = $('#exhibitionClosingTime').val();

const status = $('#exhibitionStatus').val();

const ticket\_price = $('#exhibitionTicketPrice').val();

const total\_tickets = $('#exhibitionTotalTickets').val();

const description = $('#exhibitionDescription').val().trim();

const artwork\_ids = getCheckedArtworkIds('#exhibitionArtworksCheckboxes');

const errors = [];

if (!title) {

errors.push({ field: 'exhibitionTitle', message: 'Название обязательно' });

} else if (title.length < 2 || title.length > 100) {

errors.push({ field: 'exhibitionTitle', message: 'Название должно быть от 2 до 100 символов' });

}

if (!location) {

errors.push({ field: 'exhibitionLocation', message: 'Место проведения обязательно' });

} else if (location.length < 2 || location.length > 100) {

errors.push({ field: 'exhibitionLocation', message: 'Место проведения должно быть от 2 до 100 символов' });

}

if (!start\_date) {

errors.push({ field: 'exhibitionStartDate', message: 'Дата начала обязательна' });

}

if (!end\_date) {

errors.push({ field: 'exhibitionEndDate', message: 'Дата окончания обязательна' });

} else if (start\_date && new Date(end\_date) <= new Date(start\_date)) {

errors.push({ field: 'exhibitionEndDate', message: 'Дата окончания должна быть позже даты начала' });

}

if (!opening\_time) {

errors.push({ field: 'exhibitionOpeningTime', message: 'Время открытия обязательно' });

}

if (!closing\_time) {

errors.push({ field: 'exhibitionClosingTime', message: 'Время закрытия обязательно' });

}

if (opening\_time && closing\_time && opening\_time >= closing\_time) {

errors.push({ field: 'exhibitionClosingTime', message: 'Время закрытия должно быть позже времени открытия' });

}

if (!status) {

errors.push({ field: 'exhibitionStatus', message: 'Статус обязателен' });

} else if (!['upcoming', 'active', 'completed'].includes(status)) {

errors.push({ field: 'exhibitionStatus', message: 'Неверный статус выставки' });

}

if (!ticket\_price) {

errors.push({ field: 'exhibitionTicketPrice', message: 'Цена билета обязательна' });

} else {

const price = parseFloat(ticket\_price);

if (isNaN(price) || price < 0) {

errors.push({ field: 'exhibitionTicketPrice', message: 'Цена билета должна быть положительным числом' });

}

}

if (!total\_tickets) {

errors.push({ field: 'exhibitionTotalTickets', message: 'Общее количество билетов обязательно' });

} else if (isNaN(parseInt(total\_tickets)) || parseInt(total\_tickets) < 1) {

errors.push({ field: 'exhibitionTotalTickets', message: 'Общее количество билетов должно быть положительным числом' });

}

if (description && description.length > 2000) {

errors.push({ field: 'exhibitionDescription', message: 'Описание не должно превышать 2000 символов' });

}

if (!artwork\_ids || artwork\_ids.length === 0) {

errors.push({ field: 'exhibitionArtworksCheckboxes', message: 'Выберите хотя бы одно произведение искусства' });

}

const posterFile = $('#exhibitionPoster')[0].files[0];

if (!posterFile) {

errors.push({

field: 'exhibitionPoster',

message: 'Афиша обязательна'

});

}

return errors;

}

// Функция валидации формы редактирования выставки

function validateEditExhibitionForm() {

const title = $('#editExhibitionTitle').val().trim();

const location = $('#editExhibitionLocation').val().trim();

const start\_date = $('#editExhibitionStartDate').val();

const end\_date = $('#editExhibitionEndDate').val();

const opening\_time = $('#editExhibitionOpeningTime').val();

const closing\_time = $('#editExhibitionClosingTime').val();

const status = $('#editExhibitionStatus').val();

const ticket\_price = $('#editExhibitionTicketPrice').val();

const total\_tickets = $('#editExhibitionTotalTickets').val();

const description = $('#editExhibitionDescription').val().trim();

const artwork\_ids = getCheckedArtworkIds('#editExhibitionArtworksCheckboxes');

const errors = [];

if (!title) {

errors.push({ field: 'editExhibitionTitle', message: 'Название обязательно' });

} else if (title.length < 2 || title.length > 100) {

errors.push({ field: 'editExhibitionTitle', message: 'Название должно быть от 2 до 100 символов' });

}

if (!location) {

errors.push({ field: 'editExhibitionLocation', message: 'Место проведения обязательно' });

} else if (location.length < 2 || location.length > 100) {

errors.push({ field: 'editExhibitionLocation', message: 'Место проведения должно быть от 2 до 100 символов' });

}

if (!start\_date) {

errors.push({ field: 'editExhibitionStartDate', message: 'Дата начала обязательна' });

}

if (!end\_date) {

errors.push({ field: 'editExhibitionEndDate', message: 'Дата окончания обязательна' });

} else if (start\_date && new Date(end\_date) <= new Date(start\_date)) {

errors.push({ field: 'editExhibitionEndDate', message: 'Дата окончания должна быть позже даты начала' });

}

if (!opening\_time) {

errors.push({ field: 'editExhibitionOpeningTime', message: 'Время открытия обязательно' });

}

if (!closing\_time) {

errors.push({ field: 'editExhibitionClosingTime', message: 'Время закрытия обязательно' });

}

if (opening\_time && closing\_time && opening\_time >= closing\_time) {

errors.push({ field: 'editExhibitionClosingTime', message: 'Время закрытия должно быть позже времени открытия' });

}

if (!status) {

errors.push({ field: 'editExhibitionStatus', message: 'Статус обязателен' });

} else if (!['upcoming', 'active', 'completed'].includes(status)) {

errors.push({ field: 'editExhibitionStatus', message: 'Неверный статус выставки' });

}

if (!ticket\_price) {

errors.push({ field: 'editExhibitionTicketPrice', message: 'Цена билета обязательна' });

} else {

const price = parseFloat(ticket\_price);

if (isNaN(price) || price < 0) {

errors.push({ field: 'editExhibitionTicketPrice', message: 'Цена билета должна быть положительным числом' });

}

}

if (!total\_tickets) {

errors.push({ field: 'editExhibitionTotalTickets', message: 'Общее количество билетов обязательно' });

} else if (isNaN(parseInt(total\_tickets)) || parseInt(total\_tickets) < 1) {

errors.push({ field: 'editExhibitionTotalTickets', message: 'Общее количество билетов должно быть положительным числом' });

}

if (description && description.length > 2000) {

errors.push({ field: 'editExhibitionDescription', message: 'Описание не должно превышать 2000 символов' });

}

if (!artwork\_ids || artwork\_ids.length === 0) {

errors.push({ field: 'editExhibitionArtworksCheckboxes', message: 'Выберите хотя бы одно произведение искусства' });

}

return errors;

}

// Добавляем очистку ошибок при открытии модальных окон

$('[data-modal="addExhibitionModal"]').on('click', function() {

clearErrors();

loadArtworks();

});

// Закрытие предпросмотра афиши (только по клику вне изображения и по Esc)

$('#imagePreviewModal').on('click', function(e) {

if ($(e.target).is('#imagePreviewModal')) {

$('#imagePreviewModal').removeClass('active');

$('#previewImage').attr('src', '');

$('body').removeClass('modal-open');

}

});

$(document).on('keydown', function(e) {

if (e.key === 'Escape') {

$('#imagePreviewModal').removeClass('active');

$('#previewImage').attr('src', '');

$('body').removeClass('modal-open');

}

});

document.addEventListener('DOMContentLoaded', function () {

const lightbox = document.getElementById('image-lightbox');

const lightboxImg = document.getElementById('lightbox-img');

// Универсальный обработчик для изображений афиши и картин

document.querySelector('.content-container').addEventListener('click', function(e) {

const img = e.target.closest('.exhibition-poster, .artwork-thumbnail');

if (!img) return;

const src = img.getAttribute('data-full-image') || img.src;

lightboxImg.src = src;

lightbox.style.display = 'flex';

});

lightbox.addEventListener('click', function() {

lightbox.style.display = 'none';

lightboxImg.src = '';

});

document.addEventListener('keydown', function(e) {

if (e.key === 'Escape') {

lightbox.style.display = 'none';

lightboxImg.src = '';

}

});

});

ПРИЛОЖЕНИЕ 28. Файл src\public\js\genres.js

// Глобальные переменные

let genreCurrentSort = {

column: 'id',

direction: 'asc'

};

$(document).ready(function () {

// Загрузка жанров при загрузке страницы

loadGenres();

// Обработчик открытия модального окна добавления

$('[data-modal="addGenreModal"]').on('click', function() {

$('#addGenreModal').addClass('active');

setTimeout(() => {

clearErrors('addGenreForm');

}, 0);

});

// Обработчик закрытия модальных окон

$('.modal-close').on('click', function() {

$(this).closest('.modal').removeClass('active');

});

// Переменная для хранения таймера

let searchTimer;

// Обработчик поиска при вводе

$('#searchInput').on('input', function () {

const searchTerm = $(this).val();

// Очищаем предыдущий таймер

clearTimeout(searchTimer);

// Устанавливаем новый таймер

searchTimer = setTimeout(() => {

loadGenres(searchTerm);

}, 300); // Задержка 300мс

});

// Обработчик клика по заголовкам таблицы

$('.sortable').on('click', function() {

const column = $(this).data('column');

// Если кликнули по текущему столбцу, меняем направление

if (genreCurrentSort.column === column) {

genreCurrentSort.direction = genreCurrentSort.direction === 'asc' ? 'desc' : 'asc';

} else {

// Если кликнули по новому столбцу, устанавливаем сортировку по возрастанию

genreCurrentSort.column = column;

genreCurrentSort.direction = 'asc';

}

// Перезагружаем данные

loadGenres($('#searchInput').val());

});

// Обработчик добавления жанра

$('#saveGenreBtn').on('click', function () {

if (validateGenreForm('addGenreForm', 'genreName')) {

const name = $('#genreName').val();

addGenre(name);

}

});

// Обработчик обновления жанра

$('#updateGenreBtn').on('click', function () {

if (validateGenreForm('editGenreForm', 'editGenreName')) {

const id = $('#editGenreId').val();

const name = $('#editGenreName').val();

updateGenre(id, name);

}

});

// Обработчик удаления жанра

$('#confirmDeleteBtn').on('click', function () {

const id = $(this).data('genreId');

if (id) {

deleteGenre(id);

}

});

});

// Функция для очистки ошибок

function clearErrors(formId) {

const form = document.getElementById(formId);

if (!form) return;

const errorContainer = document.getElementById(formId + 'Error');

const errorMessages = form.querySelectorAll('.error-message');

const formGroups = form.querySelectorAll('.form-group');

if (errorContainer) errorContainer.classList.remove('active');

if (errorContainer) errorContainer.textContent = '';

errorMessages.forEach(msg => msg.textContent = '');

formGroups.forEach(group => group.classList.remove('error'));

}

// Функция для отображения ошибок

function showError(formId, fieldName, message) {

const form = document.getElementById(formId);

const field = form.querySelector(`#${fieldName}`);

const formGroup = field.closest('.form-group');

const errorMessage = formGroup.querySelector('.error-message');

const errorContainer = document.getElementById(formId + 'Error');

formGroup.classList.add('error');

errorMessage.textContent = message;

errorContainer.textContent = 'Пожалуйста, исправьте ошибки в форме';

errorContainer.classList.add('active');

}

// Функция валидации формы

function validateGenreForm(formId, nameFieldId) {

const name = document.getElementById(nameFieldId).value.trim();

let isValid = true;

clearErrors(formId);

if (!name) {

showError(formId, nameFieldId, 'Название жанра обязательно');

isValid = false;

} else if (name.length < 2 || name.length > 50) {

showError(formId, nameFieldId, 'Название должно быть от 2 до 50 символов');

isValid = false;

}

return isValid;

}

// Функция загрузки жанров

function loadGenres(searchTerm = "") {

$.ajax({

url: `/api/v1/genres${searchTerm ? `?name=${searchTerm}` : ""}`,

method: 'GET',

success: function(response) {

if (response.status === "success") {

const tbody = $('#genresTableBody');

tbody.empty();

// Сортируем данные на клиенте

let genres = response.data.genres;

genres.sort((a, b) => {

let valueA, valueB;

switch(genreCurrentSort.column) {

case 'id':

valueA = a.id;

valueB = b.id;

break;

case 'name':

valueA = a.name.toLowerCase();

valueB = b.name.toLowerCase();

break;

case 'artworks':

valueA = a.statistics.total\_artworks;

valueB = b.statistics.total\_artworks;

break;

default:

valueA = a.name.toLowerCase();

valueB = b.name.toLowerCase();

break;

}

if (genreCurrentSort.direction === 'asc') {

return valueA > valueB ? 1 : -1;

} else {

return valueA < valueB ? 1 : -1;

}

});

genres.forEach(function(genre) {

const row = `

<tr>

<td>${genre.id}</td>

<td>${genre.name}</td>

<td>${genre.statistics.total\_artworks}</td>

<td>

<button class="btn btn-sm btn-primary me-2 edit-genre-btn" data-id="${genre.id}" data-name="${genre.name}">

<i class="fas fa-edit"></i> Изменить

</button>

<button class="btn btn-sm btn-danger delete-genre-btn" data-id="${genre.id}">

<i class="fas fa-trash"></i> Удалить

</button>

</td>

</tr>

`;

tbody.append(row);

});

// Добавляем обработчики для кнопок после добавления строк в таблицу

$('.edit-genre-btn').on('click', function() {

const id = $(this).data('id');

const name = $(this).data('name');

editGenre(id, name);

});

$('.delete-genre-btn').on('click', function() {

const id = $(this).data('id');

confirmDelete(id);

});

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else {

alert("Ошибка при загрузке жанров");

}

}

});

}

// Функция добавления жанра

function addGenre(name) {

$.ajax({

url: '/api/v1/genres',

method: 'POST',

contentType: 'application/json',

data: JSON.stringify({ name }),

success: function(response) {

if (response.status === "success") {

$('#addGenreModal').removeClass('active');

$('#genreName').val('');

clearErrors('addGenreForm');

loadGenres();

} else {

showError('addGenreForm', 'genreName', response.message || "Ошибка при добавлении жанра");

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const response = xhr.responseJSON;

if (response.errors && response.errors.length > 0) {

response.errors.forEach(error => {

showError('addGenreForm', 'genreName', error.message);

});

} else {

showError('addGenreForm', 'genreName', response.message || "Ошибка при добавлении жанра");

}

} else {

showError('addGenreForm', 'genreName', "Ошибка при добавлении жанра");

}

}

});

}

// Функция редактирования жанра

function editGenre(id, name) {

clearErrors('editGenreForm');

$('#editGenreId').val(id);

$('#editGenreName').val(name);

$('#editGenreModal').addClass('active');

}

// Функция обновления жанра

function updateGenre(id, name) {

$.ajax({

url: `/api/v1/genres/${id}`,

method: 'PATCH',

contentType: 'application/json',

data: JSON.stringify({ name }),

success: function(response) {

if (response.status === "success") {

// Закрываем модальное окно

$('#editGenreModal').removeClass('active');

clearErrors('editGenreForm');

loadGenres();

} else {

showError('editGenreForm', 'editGenreName', response.message || "Ошибка при обновлении жанра");

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const response = xhr.responseJSON;

if (response.errors && response.errors.length > 0) {

response.errors.forEach(error => {

showError('editGenreForm', 'editGenreName', error.message);

});

} else {

showError('editGenreForm', 'editGenreName', response.message || "Ошибка при обновлении жанра");

}

} else {

showError('editGenreForm', 'editGenreName', "Ошибка при обновлении жанра");

}

}

});

}

// Функция удаления жанра

function deleteGenre(id) {

$.ajax({

url: `/api/v1/genres/${id}`,

method: 'DELETE',

success: function(response) {

if (response.status === "success") {

// Закрываем модальное окно

$('#deleteGenreModal').removeClass('active');

// Перезагружаем список жанров

loadGenres();

} else {

alert(response.message || "Ошибка при удалении жанра");

}

},

error: function(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const response = xhr.responseJSON;

if (response.errors && response.errors.length > 0) {

alert(response.errors.map(err => err.msg).join('\n'));

} else if (response.data && response.data.artworksCount) {

alert(`Невозможно удалить жанр: он используется в ${response.data.artworksCount} произведениях`);

} else {

alert(response?.message || "Невозможно удалить жанр");

}

} else if (xhr.status === 404) {

alert("Жанр не найден");

} else {

alert("Ошибка при удалении жанра");

}

}

});

}

// Функция подтверждения удаления

function confirmDelete(id) {

$('#confirmDeleteBtn').data('genreId', id);

$('#deleteGenreModal').addClass('active');

};

ПРИЛОЖЕНИЕ 29. Файл src\public\js\main.js

// Функция для отображения уведомлений

function showNotification(message, type = 'success') {

const alertDiv = $('<div>', {

class: `alert alert-${type} alert-dismissible fade show`,

html: `

${message}

<button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Закрыть">&times;</button>

`

});

$('main').prepend(alertDiv);

// Автоматически скрыть через 5 секунд

setTimeout(() => {

alertDiv.remove();

}, 5000);

}

// Функция для форматирования даты

function formatDate(dateString) {

const options = {

year: 'numeric',

month: 'long',

day: 'numeric'

};

return new Date(dateString).toLocaleDateString('ru-RU', options);

}

// Функция для форматирования цены

function formatPrice(price) {

return new Intl.NumberFormat('ru-RU', {

style: 'currency',

currency: 'RUB'

}).format(price);

}

// Функция для форматирования времени без секунд

function formatTime(timeString) {

if (!timeString) return '';

return timeString.slice(0,5);

}

// Инициализация при загрузке страницы

$(document).ready(function() {

// Обработка AJAX форм

$('form[data-ajax="true"]').on('submit', function(e) {

e.preventDefault();

const form = $(this);

// Если есть input[type=file], используем FormData, иначе serialize

let formData;

if (form.find('input[type="file"]').length > 0) {

formData = new FormData(this);

} else {

formData = form.serialize();

}

const submitButton = form.find('[type="submit"]');

const originalButtonText = submitButton.html();

submitButton.prop('disabled', true).html('Загрузка...');

$.ajax({

url: form.attr('action'),

method: form.attr('method'),

data: formData,

// Только если есть файл — processData/contentType false

...(form.find('input[type="file"]').length > 0 ? { processData: false, contentType: false } : {}),

headers: {

'X-Requested-With': 'XMLHttpRequest'

},

success: function(data) {

alert( 'Операция выполнена успешно');

if (form.attr('id') === 'buy-ticket-form') {

const quantity = parseInt(form.find('input[name="quantity"]').val(), 10);

const leftSpan = $(form).closest('.exhibition-info-col').find('p:contains("Осталось билетов")');

if (leftSpan.length) {

// Извлекаем текущее значение

const match = leftSpan.text().match(/(\d+)/);

if (match) {

let left = parseInt(match[1], 10);

left = Math.max(0, left - quantity);

leftSpan.html(`<strong>Осталось билетов:</strong> ${left}`);

}

}

}

if (data.redirect) {

window.location.href = data.redirect;

}

},

error: function(xhr) {

const response = xhr.responseJSON;

showNotification(response?.message || 'Произошла ошибка', 'danger');

},

complete: function() {

submitButton.prop('disabled', false).html(originalButtonText);

}

});

});

// Обработка кнопки выхода

$('a[data-action="logout"]').on('click', function(e) {

e.preventDefault();

window.location.href = '/auth/logout';

});

// Обработка мобильного меню

$('.menu-toggle').on('click', function() {

$('.main-nav').toggleClass('active');

});

});

// Функция для загрузки изображений с предпросмотром

function handleImageUpload(input, previewElement) {

if (input.files && input.files[0]) {

const reader = new FileReader();

reader.onload = (e) => {

previewElement.src = e.target.result;

previewElement.style.display = 'block';

};

reader.readAsDataURL(input.files[0]);

}

}

document.addEventListener('DOMContentLoaded', function () {

const form = document.getElementById('exhibition-filter-form');

const list = document.getElementById('exhibitions-list');

const noMsg = document.getElementById('no-exhibitions-message');

function renderExhibitions(exhibitions) {

list.innerHTML = '';

if (!exhibitions.length) {

list.style.display = 'none';

noMsg.style.display = 'block';

return;

}

list.style.display = 'grid';

noMsg.style.display = 'none';

exhibitions.forEach(exh => {

const card = document.createElement('div');

card.className = 'card';

card.style.display = 'flex';

card.style.flexDirection = 'column';

card.style.justifyContent = 'space-between';

card.innerHTML = `

<div class=\"card-header\"><h4>${exh.title}</h4></div>

${exh.poster\_path ? `<img src=\"${exh.poster\_path}\" class=\"exhibition-poster\" style=\"width:100%;height:180px;object-fit:cover;\" alt=\"Афиша выставки\">` : ''}

<div class=\"card-body\">

<p><strong>Место:</strong> ${exh.location || '-'}<\/p>

<p><strong>Описание:</strong> ${exh.description || '-'}<\/p>

<p><strong>Цена билета:</strong> ${exh.ticket\_price ? exh.ticket\_price + ' ₽' : 'Бесплатно'}<\/p>

<p><strong>Даты:</strong> ${formatDate(exh.start\_date)} — ${formatDate(exh.end\_date)}<\/p>

<\/div>

<div style=\"padding:1rem; text-align:right;\">

<a href=\"/exhibitions/${exh.id}/page" class=\"btn btn-primary\">Подробнее<\/a>

<\/div>

`;

list.appendChild(card);

});

}

function fetchExhibitions(params = {}) {

const url = '/api/v1/exhibitions/public';

const queryParams = $.param(params);

const fullUrl = queryParams ? `${url}?${queryParams}` : url;

$('#exhibitions-list').html('<div style="grid-column:1/-1;text-align:center;padding:2rem;">Загрузка...</div>');

$('#no-exhibitions-message').hide();

$.ajax({

url: fullUrl,

method: 'GET',

dataType: 'json',

success: function(data) {

if (data.status === 'success') {

renderExhibitions(data.data.exhibitions);

} else {

$('#exhibitions-list').html('');

$('#no-exhibitions-message').show();

}

},

error: function(xhr, status, error) {

console.error('Ошибка при загрузке выставок:', error);

$('#exhibitions-list').html('');

$('#no-exhibitions-message').show();

}

});

}

// Сбор параметров фильтра

function getFilterParams() {

return {

search: $('#search').val().trim(),

start\_date: $('#start\_date').val(),

end\_date: $('#end\_date').val(),

min\_price: $('#min\_price').val(),

max\_price: $('#max\_price').val()

};

}

// Обработчик отправки формы

$('#exhibition-filter-form').on('submit', function (e) {

e.preventDefault();

fetchExhibitions(getFilterParams());

});

// Автозагрузка при изменении фильтров

$('#search, #start\_date, #end\_date, #min\_price, #max\_price').on('input change', function () {

fetchExhibitions(getFilterParams());

});

// Обработчики кнопок очистки фильтров

$('#clearDateFilter').on('click', function() {

$('#start\_date').val('');

$('#end\_date').val('');

fetchExhibitions(getFilterParams());

});

$('#clearPriceFilter').on('click', function() {

$('#min\_price').val('');

$('#max\_price').val('');

fetchExhibitions(getFilterParams());

});

// Первая загрузка

fetchExhibitions();

});

document.addEventListener('DOMContentLoaded', function () {

// Получаем элементы lightbox

const lightbox = document.getElementById('image-lightbox');

const lightboxImg = document.getElementById('lightbox-img');

// Делаем делегирование кликов по списку выставок

document.getElementById('exhibitions-list').addEventListener('click', function(e) {

const img = e.target.closest('.exhibition-poster');

if (!img) return;

// Устанавливаем в модалку src и показываем её

lightboxImg.src = img.src;

lightbox.style.display = 'flex';

});

// Закрываем по клику на фон

lightbox.addEventListener('click', function() {

lightbox.style.display = 'none';

lightboxImg.src = '';

});

});

ПРИЛОЖЕНИЕ 30. Файл src\public\js\profile.js

$(document).ready(function () {

// Открытие модального окна и заполнение формы

$('#editProfileBtn').on('click', function () {

$('#editProfileSurname').val($('#profileSurname').text());

$('#editProfileFirstName').val($('#profileFirstName').text());

$('#editProfilePatronymic').val($('#profilePatronymic').text());

$('#editProfileEmail').val($('#profileEmail').text());

$('#editProfilePassword').val('');

$('#editProfilePasswordConfirm').val('');

clearErrors();

$('#editProfileModal').addClass('active');

});

// Закрытие модального окна

$('.modal-close').on('click', function () {

$(this).closest('.modal').removeClass('active');

});

// Блокируем дефолтный сабмит формы, чтобы не было перезагрузки

$('#editProfileForm').on('submit', function(e) {

e.preventDefault();

});

// Сохранить изменения

$('#saveProfileBtn').on('click', function () {

const errors = validateProfileForm();

if (errors.length > 0) {

showErrors(errors);

return;

}

updateProfile();

});

});

function clearErrors() {

$('.error-message').text('');

$('.form-group').removeClass('error');

$('.form-group input').removeClass('error');

}

function showErrors(errors) {

clearErrors();

errors.forEach(({ field, message }) => {

const input = $('#' + field);

const formGroup = input.closest('.form-group');

formGroup.addClass('error');

input.addClass('error');

$('#' + field + 'Error').text(message);

});

}

function validateProfileForm() {

const surname = $('#editProfileSurname').val().trim();

const first = $('#editProfileFirstName').val().trim();

const patronym = $('#editProfilePatronymic').val().trim();

const email = $('#editProfileEmail').val().trim();

const pass = $('#editProfilePassword').val();

const pass2 = $('#editProfilePasswordConfirm').val();

const errors = [];

// Фамилия: 2–50 символов, только буквы и дефис

if (!surname) {

errors.push({ field: 'editProfileSurname', message: 'Фамилия обязательна' });

} else if (surname.length < 2 || surname.length > 50) {

errors.push({ field: 'editProfileSurname', message: 'Должна быть от 2 до 50 символов' });

} else if (!/^[A-Za-zА-Яа-яЁё-]+$/.test(surname)) {

errors.push({ field: 'editProfileSurname', message: 'Только буквы и дефис' });

}

// Имя: 2–50 символов, только буквы и дефис

if (!first) {

errors.push({ field: 'editProfileFirstName', message: 'Имя обязательно' });

} else if (first.length < 2 || first.length > 50) {

errors.push({ field: 'editProfileFirstName', message: 'Должно быть от 2 до 50 символов' });

} else if (!/^[A-Za-zА-Яа-яЁё-]+$/.test(first)) {

errors.push({ field: 'editProfileFirstName', message: 'Только буквы и дефис' });

}

// Отчество: если указано, 2–50 символов, только буквы и дефис

if (patronym) {

if (patronym.length < 2 || patronym.length > 50) {

errors.push({ field: 'editProfilePatronymic', message: 'Должно быть от 2 до 50 символов' });

} else if (!/^[A-Za-zА-Яа-яЁё-]+$/.test(patronym)) {

errors.push({ field: 'editProfilePatronymic', message: 'Только буквы и дефис' });

}

}

// Email: 5–100 символов, валидный формат

if (!email) {

errors.push({ field: 'editProfileEmail', message: 'Email обязателен' });

} else if (email.length < 5 || email.length > 100) {

errors.push({ field: 'editProfileEmail', message: 'От 5 до 100 символов' });

} else if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)) {

errors.push({ field: 'editProfileEmail', message: 'Неверный формат email' });

}

// Пароль: если меняем, >=8 и совпадает с подтверждением

if (pass) {

if (pass.length < 8) {

errors.push({ field: 'editProfilePassword', message: 'Не менее 8 символов' });

}

if (!pass2) {

errors.push({ field: 'editProfilePasswordConfirm', message: 'Подтвердите пароль' });

} else if (pass !== pass2) {

errors.push({ field: 'editProfilePasswordConfirm', message: 'Пароли не совпадают' });

}

}

return errors;

}

function updateProfile() {

const data = {

surname: $('#editProfileSurname').val().trim(),

first\_name: $('#editProfileFirstName').val().trim(),

patronymic: ($('#editProfilePatronymic').val().trim() || null),

email: $('#editProfileEmail').val().trim()

};

const pass = $('#editProfilePassword').val();

if (pass) data.password = pass;

$.ajax({

url: '/api/v1/users/me',

method: 'PATCH',

contentType: 'application/json',

data: JSON.stringify(data),

success(response) {

if (response.status === 'success') {

const u = response.data.user;

$('#profileSurname').text(u.surname);

$('#profileFirstName').text(u.first\_name);

$('#profilePatronymic').text(u.patronymic || '');

$('#profileEmail').text(u.email);

$('#editProfileModal').removeClass('active');

clearErrors();

alert('Профиль успешно обновлен');

}

},

error(xhr) {

if (xhr.status === 401) {

window.location.href = '/auth/login';

} else if (xhr.status === 400) {

const resp = xhr.responseJSON;

if (resp.errors && resp.errors.length) {

showErrors(resp.errors);

} else {

alert(resp.message || 'Ошибка при обновлении профиля');

}

} else {

alert('Ошибка при обновлении профиля');

}

}

});

}

ПРИЛОЖЕНИЕ 31. Файл src\public\js\statistics.js

document.addEventListener('DOMContentLoaded', () => {

const startInput = document.getElementById('statStart');

const endInput = document.getElementById('statEnd');

const btn = document.getElementById('fetchStatsBtn');

const loading = document.getElementById('statsLoading');

const content = document.getElementById('statsContent');

const emptyMsg = document.getElementById('statsEmpty');

const soldEl = document.getElementById('ticketsSold');

const revenueEl = document.getElementById('totalRevenue');

btn.addEventListener('click', () => {

const start = startInput.value;

const end = endInput.value;

content.style.display = 'none';

emptyMsg.style.display = 'none';

loading.style.display = 'block';

const params = new URLSearchParams();

if (start) params.append('start\_date', start);

if (end) params.append('end\_date', end);

fetch(`/api/v1/reports/tickets?${params}`)

.then(r => r.json())

.then(json => {

loading.style.display = 'none';

if (json.status === 'success') {

const { tickets\_sold, revenue } = json.data;

// Если оба = null или 0 — считаем, что нет данных

if ((!tickets\_sold && tickets\_sold !== 0) ||

(!revenue && revenue !== 0)) {

emptyMsg.style.display = 'block';

} else {

soldEl.textContent = tickets\_sold || 0;

revenueEl.textContent = Number(revenue).toFixed(2);

content.style.display = 'block';

}

} else {

emptyMsg.textContent = json.message || 'Нет данных';

emptyMsg.style.display = 'block';

}

})

.catch(err => {

console.error('Ошибка при загрузке статистики:', err);

loading.style.display = 'none';

emptyMsg.textContent = 'Ошибка при загрузке';

emptyMsg.style.display = 'block';

});

});

});

ПРИЛОЖЕНИЕ 32. Файл src\public\js\styles.js

*// Глобальные переменные*

*let stylesCurrentSort = {*

*column: 'id',*

*direction: 'asc'*

*};*

*$(document).ready(function () {*

*// Загрузка стилей при загрузке страницы*

*loadStyles();*

*// Обработчик открытия модального окна добавления*

*$('[data-modal="addStyleModal"]').on('click', function() {*

*$('#addStyleModal').addClass('active');*

*setTimeout(() => {*

*clearErrors('addStyleForm');*

*}, 0);*

*});*

*// Обработчик закрытия модальных окон*

*$('.modal-close').on('click', function() {*

*$(this).closest('.modal').removeClass('active');*

*});*

*// Переменная для хранения таймера*

*let searchTimer;*

*// Обработчик поиска при вводе*

*$('#searchInput').on('input', function () {*

*const searchTerm = $(this).val();*

*clearTimeout(searchTimer);*

*searchTimer = setTimeout(() => {*

*loadStyles(searchTerm);*

*}, 300);*

*});*

*// Обработчик клика по заголовкам таблицы*

*$('.sortable').on('click', function() {*

*const column = $(this).data('column');*

*// Если кликнули по текущему столбцу, меняем направление*

*if (stylesCurrentSort.column === column) {*

*stylesCurrentSort.direction = stylesCurrentSort.direction === 'asc' ? 'desc' : 'asc';*

*} else {*

*// Если кликнули по новому столбцу, устанавливаем сортировку по возрастанию*

*stylesCurrentSort.column = column;*

*stylesCurrentSort.direction = 'asc';*

*}*

*// Перезагружаем данные*

*loadStyles($('#searchInput').val());*

*});*

*// Обработчик добавления стиля*

*$('#saveStyleBtn').on('click', function () {*

*if (validateStyleForm('addStyleForm', 'styleName')) {*

*const name = $('#styleName').val();*

*addStyle(name);*

*}*

*});*

*// Обработчик обновления стиля*

*$('#updateStyleBtn').on('click', function () {*

*if (validateStyleForm('editStyleForm', 'editStyleName')) {*

*const id = $('#editStyleId').val();*

*const name = $('#editStyleName').val();*

*updateStyle(id, name);*

*}*

*});*

*// Обработчик удаления стиля*

*$('#confirmDeleteBtn').on('click', function () {*

*const id = $(this).data('styleId');*

*if (id) {*

*deleteStyle(id);*

*}*

*});*

*});*

*// Функция загрузки стилей*

*function loadStyles(searchTerm="") {*

*$.ajax({*

*url: `/api/v1/styles${searchTerm ? `?name=${searchTerm}` : ""}`,*

*method: 'GET',*

*success: function(response) {*

*if(response.status === "success") {*

*const tbody = $('#stylesTableBody');*

*tbody.empty();*

*// Сортировка данных на клиенте*

*let styles = response.data.styles;*

*styles.sort((a, b) => {*

*let valueA, valueB;*

*switch (stylesCurrentSort.column) {*

*case 'id':*

*valueA = a.id;*

*valueB = b.id;*

*break;*

*case 'name':*

*valueA = a.name.toLowerCase();*

*valueB = b.name.toLowerCase();*

*break;*

*case 'artworks':*

*valueA = a.statistics?.total\_artworks;*

*valueB = b.statistics?.total\_artworks;*

*break;*

*default:*

*valueA = a.name.toLowerCase();*

*valueB = b.name.toLowerCase();*

*break;*

*}*

*if (stylesCurrentSort.direction === 'asc') {*

*return valueA > valueB ? 1 : -1;*

*} else {*

*return valueA < valueB ? 1 : -1;*

*}*

*});*

*styles.forEach(function (style) {*

*const row = `*

*<tr>*

*<td>${style.id}</td>*

*<td>${style.name}</td>*

*<td>${style.statistics?.total\_artworks}</td>*

*<td>*

*<button class="btn btn-sm btn-primary me-2 edit-style-btn" data-id="${style.id}" data-name="${style.name}">*

*<i class="fas fa-edit"></i> Изменить*

*</button>*

*<button class="btn btn-sm btn-danger delete-style-btn" data-id="${style.id}">*

*<i class="fas fa-trash"></i> Удалить*

*</button>*

*</td>*

*</tr>*

*`;*

*tbody.append(row);*

*});*

*// Добавляем обработчики для кнопок после добавления строк в таблицу*

*$('.edit-style-btn').on('click', function() {*

*const id = $(this).data('id');*

*const name = $(this).data('name');*

*editStyle(id, name);*

*});*

*$('.delete-style-btn').on('click', function() {*

*const id = $(this).data('id');*

*confirmDelete(id);*

*});*

*}*

*},*

*error: function(xhr) {*

*if (xhr.status === 401) {*

*window.location.href = '/auth/login';*

*} else {*

*alert("Ошибка при загрузке стилей");*

*}*

*}*

*});*

*}*

*// Функция для очистки ошибок*

*function clearErrors(formId) {*

*const form = document.getElementById(formId);*

*if (!form) return;*

*const errorContainer = document.getElementById(formId + 'Error');*

*const errorMessages = form.querySelectorAll('.error-message');*

*const formGroups = form.querySelectorAll('.form-group');*

*if (errorContainer) errorContainer.classList.remove('active');*

*if (errorContainer) errorContainer.textContent = '';*

*errorMessages.forEach(msg => msg.textContent = '');*

*formGroups.forEach(group => group.classList.remove('error'));*

*}*

*// Функция для отображения ошибок*

*function showError(formId, fieldName, message) {*

*const form = document.getElementById(formId);*

*const field = form.querySelector(`#${fieldName}`);*

*const formGroup = field.closest('.form-group');*

*const errorMessage = formGroup.querySelector('.error-message');*

*const errorContainer = document.getElementById(formId + 'Error');*

*formGroup.classList.add('error');*

*errorMessage.textContent = message;*

*errorContainer.textContent = 'Пожалуйста, исправьте ошибки в форме';*

*errorContainer.classList.add('active');*

*}*

*// Функция валидации формы*

*function validateStyleForm(formId, nameFieldId) {*

*const name = document.getElementById(nameFieldId).value.trim();*

*let isValid = true;*

*clearErrors(formId);*

*if (!name) {*

*showError(formId, nameFieldId, 'Название стиля обязательно');*

*isValid = false;*

*} else if (name.length < 2 || name.length > 50) {*

*showError(formId, nameFieldId, 'Название должно быть от 2 до 50 символов');*

*isValid = false;*

*}*

*return isValid;*

*}*

*// Функция добавления стиля*

*function addStyle(name) {*

*$.ajax({*

*url: '/api/v1/styles',*

*method: 'POST',*

*contentType: 'application/json',*

*data: JSON.stringify({ name }),*

*success: function(response) {*

*if (response.status === "success") {*

*$('#addStyleModal').removeClass('active');*

*$('#styleName').val('');*

*clearErrors('addStyleForm');*

*loadStyles();*

*} else {*

*showError('addStyleForm', 'styleName', response.message || "Ошибка при добавлении стиля");*

*}*

*},*

*error: function(xhr) {*

*if (xhr.status === 401) {*

*window.location.href = '/auth/login';*

*} else if (xhr.status === 400) {*

*const response = xhr.responseJSON;*

*if (response.errors && response.errors.length > 0) {*

*response.errors.forEach(error => {*

*showError('addStyleForm', 'styleName', error.message);*

*});*

*} else {*

*showError('addStyleForm', 'styleName', response.message || "Ошибка при добавлении стиля");*

*}*

*} else {*

*showError('addStyleForm', 'styleName', "Ошибка при добавлении стиля");*

*}*

*}*

*});*

*}*

*// Функция редактирования стиля*

*window.editStyle = function(id, name) {*

*clearErrors('editStyleForm');*

*$('#editStyleId').val(id);*

*$('#editStyleName').val(name);*

*$('#editStyleModal').addClass('active');*

*};*

*// Функция обновления стиля*

*function updateStyle(id, name) {*

*$.ajax({*

*url: `/api/v1/styles/${id}`,*

*method: 'PATCH',*

*contentType: 'application/json',*

*data: JSON.stringify({ name }),*

*success: function(response) {*

*if (response.status === "success") {*

*$('#editStyleModal').removeClass('active');*

*clearErrors('editStyleForm');*

*loadStyles();*

*} else {*

*showError('editStyleForm', 'editStyleName', response.message || "Ошибка при обновлении стиля");*

*}*

*},*

*error: function(xhr) {*

*if (xhr.status === 401) {*

*window.location.href = '/auth/login';*

*} else if (xhr.status === 400) {*

*const response = xhr.responseJSON;*

*if (response.errors && response.errors.length > 0) {*

*response.errors.forEach(error => {*

*showError('editStyleForm', 'editStyleName', error.message);*

*});*

*} else {*

*showError('editStyleForm', 'editStyleName', response.message || "Ошибка при обновлении стиля");*

*}*

*} else {*

*showError('editStyleForm', 'editStyleName', "Ошибка при обновлении стиля");*

*}*

*}*

*});*

*}*

*// Функция удаления стиля*

*function deleteStyle(id) {*

*$.ajax({*

*url: `/api/v1/styles/${id}`,*

*method: 'DELETE',*

*success: function(response) {*

*if (response.status === "success") {*

*// Закрываем модальное окно*

*$('#deleteStyleModal').removeClass('active');*

*// Перезагружаем список стилей*

*loadStyles();*

*} else {*

*alert(response.message || "Ошибка при удалении стиля");*

*}*

*},*

*error: function(xhr) {*

*if (xhr.status === 401) {*

*window.location.href = '/auth/login';*

*} else if (xhr.status === 400) {*

*const response = xhr.responseJSON;*

*if (response.errors && response.errors.length > 0) {*

*alert(response.errors.map(err => err.msg).join('\n'));*

*} else if (response.data && response.data.artworksCount) {*

*alert(`Невозможно удалить стиль: он используется в ${response.data.artworksCount} произведениях`);*

*} else {*

*alert(response?.message || "Невозможно удалить стиль");*

*}*

*} else if (xhr.status === 404) {*

*alert("Стиль не найден");*

*} else {*

*alert("Ошибка при удалении стиля");*

*}*

*}*

*});*

*}*

*// Функция подтверждения удаления*

*window.confirmDelete = function(id) {*

*$('#confirmDeleteBtn').data('styleId', id);*

*$('#deleteStyleModal').addClass('active');*

*};*

ПРИЛОЖЕНИЕ 33. Файл src\public\js\ticket.js

*// Глобальные переменные*

*let ticketCurrentSort = {*

*column: 'id',*

*direction: 'asc'*

*};*

*$(document).ready(function () {*

*loadTickets();*

*loadUsers();*

*loadExhibitions();*

*let searchTimer;*

*// Поиск*

*$('#searchTicketInput').on('input', function () {*

*const searchTerm = $(this).val();*

*clearTimeout(searchTimer);*

*searchTimer = setTimeout(() => {*

*loadTickets(searchTerm);*

*}, 300);*

*});*

*// Обработчики фильтрации*

*$('#bookingDateFrom, #bookingDateTo').on('change', function() {*

*loadTickets();*

*});*

*$('#priceFrom, #priceTo').on('input', function() {*

*loadTickets();*

*});*

*// Обработчики очистки отдельных фильтров*

*$('#clearBookingDateFilter').on('click', function() {*

*$('#bookingDateFrom').val('');*

*$('#bookingDateTo').val('');*

*loadTickets();*

*});*

*$('#clearPriceFilter').on('click', function() {*

*$('#priceFrom').val('');*

*$('#priceTo').val('');*

*loadTickets();*

*});*

*// Сортировка*

*$('.sortable').on('click', function() {*

*const column = $(this).data('column');*

*if (ticketCurrentSort.column === column) {*

*ticketCurrentSort.direction = ticketCurrentSort.direction === 'asc' ? 'desc' : 'asc';*

*} else {*

*ticketCurrentSort.column = column;*

*ticketCurrentSort.direction = 'asc';*

*}*

*loadTickets($('#searchTicketInput').val());*

*});*

*// Открытие модального окна добавления*

*$('[data-modal="addTicketModal"]').on('click', function() {*

*clearTicketErrors();*

*$('#addTicketForm')[0].reset();*

*$('#addTicketModal').addClass('active');*

*});*

*// Закрытие модальных окон*

*$('.modal-close').on('click', function() {*

*$(this).closest('.modal').removeClass('active');*

*});*

*// Добавление билета*

*$('#saveTicketBtn').on('click', function () {*

*const errors = validateTicketForm('add');*

*if (errors.length > 0) {*

*showTicketErrors(errors, 'add');*

*return;*

*}*

*console.log("hi")*

*const data = {*

*user\_id: $('#ticketUser').val(),*

*exhibition\_id: $('#ticketExhibition').val(),*

*quantity: $('#ticketQuantity').val()*

*};*

*$.ajax({*

*url: '/api/v1/tickets/admin/tickets',*

*method: 'POST',*

*contentType: 'application/json',*

*data: JSON.stringify(data),*

*success: function(response) {*

*if (response.status === 'success') {*

*$('#addTicketModal').removeClass('active');*

*loadTickets();*

*}*

*},*

*error: function(xhr) {*

*if (xhr.status === 400) {*

*const response = xhr.responseJSON;*

*if (response.errors && response.errors.length > 0) {*

*showTicketErrors(response.errors, 'add');*

*} else {*

*alert(response.message || 'Ошибка при добавлении билета');*

*}*

*} else {*

*alert('Ошибка при добавлении билета');*

*}*

*}*

*});*

*});*

*// Открытие модального окна редактирования*

*$(document).on('click', '.edit-ticket-btn', function() {*

*clearTicketErrors();*

*const id = $(this).data('id');*

*$.ajax({*

*url: `/api/v1/tickets/${id}`,*

*method: 'GET',*

*success: function(response) {*

*if (response.status === 'success') {*

*const ticket = response.data.ticket;*

*$('#editTicketId').val(ticket.id);*

*$('#editTicketUser').val(ticket.user\_id);*

*$('#editTicketExhibition').val(ticket.exhibition\_id);*

*$('#editTicketQuantity').val(ticket.quantity);*

*$('#editTicketModal').addClass('active');*

*}*

*},*

*error: function() {*

*alert('Ошибка при загрузке данных билета');*

*}*

*});*

*});*

*// Сохранение изменений билета*

*$('#updateTicketBtn').on('click', function () {*

*const errors = validateTicketForm('edit');*

*if (errors.length > 0) {*

*showTicketErrors(errors, 'edit');*

*return;*

*}*

*const id = $('#editTicketId').val();*

*const data = {*

*user\_id: $('#editTicketUser').val(),*

*exhibition\_id: $('#editTicketExhibition').val(),*

*quantity: $('#editTicketQuantity').val()*

*};*

*$.ajax({*

*url: `/api/v1/tickets/${id}`,*

*method: 'PATCH',*

*contentType: 'application/json',*

*data: JSON.stringify(data),*

*success: function(response) {*

*if (response.status === 'success') {*

*$('#editTicketModal').removeClass('active');*

*loadTickets();*

*}*

*},*

*error: function(xhr) {*

*if (xhr.status === 400) {*

*const response = xhr.responseJSON;*

*if (response.errors && response.errors.length > 0) {*

*showTicketErrors(response.errors, 'edit');*

*} else {*

*alert(response.message || 'Ошибка при обновлении билета');*

*}*

*} else {*

*alert('Ошибка при обновлении билета');*

*}*

*}*

*});*

*});*

*// Открытие модального окна удаления*

*$(document).on('click', '.delete-ticket-btn', function() {*

*const id = $(this).data('id');*

*$('#confirmDeleteTicketBtn').data('ticketId', id);*

*$('#deleteTicketModal').addClass('active');*

*});*

*// Подтверждение удаления*

*$('#confirmDeleteTicketBtn').on('click', function () {*

*const id = $(this).data('ticketId');*

*$.ajax({*

*url: `/api/v1/tickets/${id}`,*

*method: 'DELETE',*

*success: function(response) {*

*if (response.status === 'success') {*

*$('#deleteTicketModal').removeClass('active');*

*loadTickets();*

*} else {*

*alert(response.message || 'Ошибка при удалении билета');*

*}*

*},*

*error: function(xhr) {*

*alert('Ошибка при удалении билета');*

*}*

*});*

*});*

*});*

*// Загрузка билетов*

*function loadTickets(searchTerm = "") {*

*// Получаем значения всех фильтров*

*const bookingDateFrom = $('#bookingDateFrom').val();*

*const bookingDateTo = $('#bookingDateTo').val();*

*const priceFrom = $('#priceFrom').val();*

*const priceTo = $('#priceTo').val();*

*// Формируем параметры запроса*

*const params = new URLSearchParams();*

*if (searchTerm) {*

*params.append('search', searchTerm.trim());*

*}*

*if (bookingDateFrom) {*

*params.append('booking\_date\_from', bookingDateFrom);*

*}*

*if (bookingDateTo) {*

*params.append('booking\_date\_to', bookingDateTo);*

*}*

*if (priceFrom) {*

*params.append('min\_price', priceFrom);*

*}*

*if (priceTo) {*

*params.append('max\_price', priceTo);*

*}*

*const queryString = params.toString();*

*const url = `/api/v1/tickets${queryString ? `?${queryString}` : ""}`;*

*$.ajax({*

*url: url,*

*method: 'GET',*

*success: function(response) {*

*if (response.status === 'success') {*

*const tbody = $('#ticketsTableBody');*

*tbody.empty();*

*let tickets = response.data.tickets;*

*tickets.sort((a, b) => {*

*let valueA, valueB;*

*switch(ticketCurrentSort.column) {*

*case 'id':*

*valueA = a.id;*

*valueB = b.id;*

*break;*

*case 'user':*

*valueA = `${a.User.surname} ${a.User.first\_name} ${a.User.patronymic}`.toLowerCase();*

*valueB = `${b.User.surname} ${b.User.first\_name} ${b.User.patronymic}`.toLowerCase();*

*break;*

*case 'exhibition':*

*valueA = a.Exhibition.title.toLowerCase();*

*valueB = b.Exhibition.title.toLowerCase();*

*break;*

*case 'booking\_date':*

*valueA = new Date(a.booking\_date);*

*valueB = new Date(b.booking\_date);*

*break;*

*case 'total\_price':*

*valueA = a.total\_price;*

*valueB = b.total\_price;*

*break;*

*case 'quantity':*

*valueA = a.quantity;*

*valueB = b.quantity;*

*break;*

*default:*

*valueA = a.id;*

*valueB = b.id;*

*break;*

*}*

*if (ticketCurrentSort.direction === 'asc') {*

*return valueA > valueB ? 1 : -1;*

*} else {*

*return valueA < valueB ? 1 : -1;*

*}*

*});*

*tickets.forEach(function(ticket) {*

*const row = `*

*<tr>*

*<td>${ticket.id}</td>*

*<td>${ticket.User.surname} ${ticket.User.first\_name} ${ticket.User.patronymic}</td>*

*<td>${ticket.Exhibition.title}</td>*

*<td>${formatDate(ticket.booking\_date)}</td>*

*<td>${ticket.total\_price}</td>*

*<td>${ticket.quantity}</td>*

*<td>*

*<button class="btn btn-sm btn-primary me-2 edit-ticket-btn" data-id="${ticket.id}">*

*<i class="fas fa-edit"></i>Изменить*

*</button>*

*<button class="btn btn-sm btn-danger delete-ticket-btn" data-id="${ticket.id}">*

*<i class="fas fa-trash"></i>Удалить*

*</button>*

*</td>*

*</tr>*

*`;*

*tbody.append(row);*

*});*

*}*

*},*

*error: function(xhr) {*

*alert('Ошибка при загрузке билетов');*

*}*

*});*

*}*

*// Загрузка пользователей для выпадающего списка*

*function loadUsers() {*

*$.ajax({*

*url: '/api/v1/users',*

*method: 'GET',*

*success: function(response) {*

*if (response.status === 'success') {*

*const users = response.data.users;*

*const userSelects = $('#ticketUser, #editTicketUser');*

*userSelects.empty();*

*users.forEach(function(user) {*

*const option = `<option value="${user.id}">${user.surname} ${user.first\_name} ${user.patronymic}</option>`;*

*userSelects.append(option);*

*});*

*}*

*}*

*});*

*}*

*// Загрузка выставок для выпадающего списка*

*function loadExhibitions() {*

*$.ajax({*

*url: '/api/v1/exhibitions',*

*method: 'GET',*

*success: function(response) {*

*if (response.status === 'success') {*

*const exhibitions = response.data.exhibitions;*

*const exhibitionSelects = $('#ticketExhibition, #editTicketExhibition');*

*exhibitionSelects.empty();*

*exhibitions.forEach(function(exhibition) {*

*const option = `<option value="${exhibition.id}">${exhibition.title}</option>`;*

*exhibitionSelects.append(option);*

*});*

*}*

*}*

*});*

*}*

*// Форматирование даты*

*function formatDate(dateStr) {*

*const date = new Date(dateStr);*

*return date.toLocaleDateString() + ' ' + date.toLocaleTimeString();*

*}*

*// Очистка ошибок*

*function clearTicketErrors() {*

*$('.error-message').text('');*

*$('.form-group').removeClass('error');*

*$('.form-group input, .form-group select').removeClass('error');*

*}*

*// Отображение ошибок*

*function showTicketErrors(errors, prefix) {*

*clearTicketErrors();*

*if (errors && errors.length > 0) {*

*errors.forEach(error => {*

*const field = error.field;*

*const message = error.message;*

*const formGroup = $(`#${prefix}Ticket${capitalizeFirst(field)}`).closest('.form-group');*

*formGroup.addClass('error');*

*$(`#${prefix}Ticket${capitalizeFirst(field)}`).addClass('error');*

*$(`#${prefix}Ticket${capitalizeFirst(field)}Error`).text(message);*

*});*

*}*

*}*

*// Валидация формы билета*

*function validateTicketForm(prefix) {*

*const userSel = prefix === 'add' ? '#ticketUser' : `#${prefix}TicketUser`;*

*const exhibitionSel = prefix === 'add'*

*? '#ticketExhibition'*

*: `#${prefix}TicketExhibition`;*

*const quantitySel = prefix === 'add'*

*? '#ticketQuantity'*

*: `#${prefix}TicketQuantity`;*

*const user\_id = $(userSel).val();*

*const exhibition\_id = $(exhibitionSel).val();*

*const quantity = $(quantitySel).val();*

*const errors = [];*

*if (!user\_id) {*

*errors.push({ field: 'User', message: 'Выберите пользователя' });*

*}*

*if (!exhibition\_id) {*

*errors.push({ field: 'Exhibition', message: 'Выберите выставку' });*

*}*

*if (!quantity || isNaN(quantity) || quantity <= 0) {*

*errors.push({ field: 'Quantity', message: 'Введите корректное количество' });*

*}*

*return errors;*

*}*

*function capitalizeFirst(str) {*

*return str.charAt(0).toUpperCase() + str.slice(1);*

*}*

ПРИЛОЖЕНИЕ 34. Файл src\public\js\tickets.js

*// public/js/tickets.js*

*document.addEventListener('DOMContentLoaded', function () {*

*const list = document.getElementById('tickets-list');*

*const noMsg = document.getElementById('no-tickets-message');*

*const modal = document.getElementById('cancel-ticket-modal');*

*const cancelBtn = document.getElementById('cancel-confirmation-btn');*

*const confirmBtn = document.getElementById('confirm-cancel-btn');*

*if (!list) return;*

*// Переменная для хранения ID билета для отмены*

*let ticketToCancel = null;*

*// Показать индикатор*

*list.innerHTML = '<div style="grid-column:1/-1;text-align:center;padding:2rem;">Загрузка...</div>';*

*list.style.display = 'grid';*

*if (noMsg) noMsg.style.display = 'none';*

*// Функция для закрытия модального окна*

*function closeCancelModal() {*

*modal.style.display = 'none';*

*ticketToCancel = null;*

*}*

*// Обработчики для модального окна*

*cancelBtn.addEventListener('click', closeCancelModal);*

*// Закрытие модального окна при клике вне его*

*window.addEventListener('click', function(event) {*

*if (event.target === modal) {*

*closeCancelModal();*

*}*

*});*

*// Функция для отмены билета*

*function cancelTicket(ticketId) {*

*fetch(`/api/v1/tickets/${ticketId}`, {*

*method: 'DELETE',*

*headers: {*

*'Content-Type': 'application/json'*

*}*

*})*

*.then(res => res.json())*

*.then(json => {*

*if (json.status === 'success') {*

*// Удаляем карточку билета из DOM*

*const ticketCard = document.querySelector(`[data-ticket-id="${ticketId}"]`);*

*if (ticketCard) {*

*ticketCard.remove();*

*}*

*// Проверяем, остались ли еще билеты*

*const remainingTickets = list.querySelectorAll('.card');*

*if (remainingTickets.length === 0) {*

*list.style.display = 'none';*

*if (noMsg) noMsg.style.display = 'block';*

*}*

*closeCancelModal();*

*// Показываем уведомление об успешной отмене*

*alert('Билет успешно отменен');*

*} else {*

*alert(json.message || 'Ошибка при отмене билета');*

*}*

*})*

*.catch(err => {*

*console.error('Ошибка при отмене билета:', err);*

*alert('Ошибка при отмене билета');*

*});*

*}*

*// Обработчик подтверждения отмены*

*confirmBtn.addEventListener('click', function() {*

*if (ticketToCancel) {*

*cancelTicket(ticketToCancel);*

*}*

*});*

*// Функция для показа модального окна подтверждения*

*function showCancelConfirmation(ticket) {*

*ticketToCancel = ticket.id;*

*// Показываем модальное окно*

*modal.style.display = 'flex';*

*}*

*// Запросить свои билеты*

*fetch('/api/v1/tickets/my-tickets')*

*.then(res => res.json())*

*.then(json => {*

*const tickets = (json.status === 'success' && Array.isArray(json.data.tickets))*

*? json.data.tickets*

*: [];*

*list.innerHTML = '';*

*if (!tickets.length) {*

*list.style.display = 'none';*

*if (noMsg) noMsg.style.display = 'block';*

*return;*

*}*

*// Рендерим карточки*

*list.style.display = 'grid';*

*if (noMsg) noMsg.style.display = 'none';*

*tickets.forEach(t => {*

*const card = document.createElement('div');*

*card.className = 'card';*

*card.setAttribute('data-ticket-id', t.id);*

*card.style.display = 'flex';*

*card.style.flexDirection = 'column';*

*card.style.justifyContent = 'space-between';*

*card.style.padding = '1rem';*

*card.style.marginBottom = '1rem';*

*// дата*

*const booked = new Date(t.booking\_date);*

*const bookedStr = booked.toLocaleString('ru-RU', {*

*day: '2-digit', month: '2-digit', year: 'numeric',*

*hour: '2-digit', minute: '2-digit'*

*});*

*card.innerHTML = `*

*<a href="/exhibitions/${t.Exhibition.id}/page" class="hover:underline text-blue-600">*

*${t.Exhibition.title}*

*</a>*

*<div class="card-body" style="flex-grow:1;">*

*<p><strong>Дата брони:</strong> ${bookedStr}</p>*

*<p><strong>Кол-во билетов:</strong> ${t.quantity}</p>*

*<p><strong>Стоимость:</strong> ${t.total\_price} ₽</p>*

*<p><strong>Период:</strong>*

*${new Date(t.Exhibition.start\_date).toLocaleDateString('ru-RU')} –*

*${new Date(t.Exhibition.end\_date).toLocaleDateString('ru-RU')}*

*</p>*

*<p><strong>Место:</strong> ${t.Exhibition.location}</p>*

*</div>*

*<div class="card-footer" style="margin-top: 1rem; text-align: center;">*

*<button type="button" class="btn btn-danger cancel-ticket-btn" style="width: 100%;">*

*Отменить билет*

*</button>*

*</div>*

*`;*

*// Добавляем обработчик для кнопки отмены*

*const cancelButton = card.querySelector('.cancel-ticket-btn');*

*cancelButton.addEventListener('click', function() {*

*showCancelConfirmation(t);*

*});*

*list.appendChild(card);*

*});*

*})*

*.catch(err => {*

*console.error('Ошибка при загрузке билетов:', err);*

*list.style.display = 'none';*

*if (noMsg) noMsg.style.display = 'block';*

*});*

*})*

ПРИЛОЖЕНИЕ 35. Файл src\public\js\users.js

*// Глобальные переменные*

*let userCurrentSort = {*

*column: 'id',*

*direction: 'asc'*

*};*

*$(document).ready(function () {*

*// Загрузка пользователей при загрузке страницы*

*loadUsers();*

*// Переменная для хранения таймера*

*let searchTimer;*

*// Обработчик поиска при вводе*

*$('#searchInput').on('input', function () {*

*const searchTerm = $(this).val();*

*// Очищаем предыдущий таймер*

*clearTimeout(searchTimer);*

*// Устанавливаем новый таймер*

*searchTimer = setTimeout(() => {*

*loadUsers(searchTerm);*

*}, 300); // Задержка 300мс*

*});*

*// Обработчик клика по заголовкам таблицы*

*$('.sortable').on('click', function() {*

*const column = $(this).data('column');*

*// Если кликнули по текущему столбцу, меняем направление*

*if (userCurrentSort.column === column) {*

*userCurrentSort.direction = userCurrentSort.direction === 'asc' ? 'desc' : 'asc';*

*} else {*

*// Если кликнули по новому столбцу, устанавливаем сортировку по возрастанию*

*userCurrentSort.column = column;*

*userCurrentSort.direction = 'asc';*

*}*

*// Перезагружаем данные*

*loadUsers($('#searchInput').val());*

*});*

*// Обработчик добавления пользователя*

*$('#saveUserBtn').on('click', function () {*

*const errors = validateAddUserForm();*

*if (errors.length > 0) {*

*showErrors(errors);*

*return;*

*}*

*const surname = $('#userSurname').val();*

*const first\_name = $('#userFirstName').val();*

*const patronymic = $('#userPatronymic').val();*

*const email = $('#userEmail').val();*

*const password = $('#userPassword').val();*

*const role = $('#userRole').val();*

*addUser(surname, first\_name, patronymic, email, password, role);*

*});*

*// Обработчик обновления пользователя*

*$('#updateUserBtn').on('click', function () {*

*const errors = validateEditUserForm();*

*if (errors.length > 0) {*

*showErrors(errors);*

*return;*

*}*

*const id = $('#editUserId').val();*

*const surname = $('#editUserSurname').val();*

*const first\_name = $('#editUserFirstName').val();*

*const patronymic = $('#editUserPatronymic').val();*

*const email = $('#editUserEmail').val();*

*const password = $('#editUserPassword').val();*

*const role = $('#editUserRole').val();*

*updateUser(id, surname, first\_name, patronymic, email, password, role);*

*});*

*// Обработчик удаления пользователя*

*$('#confirmDeleteBtn').on('click', function () {*

*const id = $(this).data('userId');*

*if (id) {*

*deleteUser(id);*

*}*

*});*

*});*

*// Функция загрузки пользователей*

*function loadUsers(searchTerm = "") {*

*$.ajax({*

*url: `/api/v1/users${searchTerm ? `?name=${searchTerm}` : ""}`,*

*method: 'GET',*

*success: function(response) {*

*if (response.status === "success") {*

*const tbody = $('#usersTableBody');*

*tbody.empty();*

*// Сортируем данные на клиенте*

*let users = response.data.users;*

*users.sort((a, b) => {*

*let valueA, valueB;*

*switch(userCurrentSort.column) {*

*case 'id':*

*valueA = a.id;*

*valueB = b.id;*

*break;*

*case 'surname':*

*valueA = a.surname.toLowerCase();*

*valueB = b.surname.toLowerCase();*

*break;*

*case 'first\_name':*

*valueA = a.first\_name.toLowerCase();*

*valueB = b.first\_name.toLowerCase();*

*break;*

*case 'patronymic':*

*valueA = (a.patronymic || '').toLowerCase();*

*valueB = (b.patronymic || '').toLowerCase();*

*break;*

*case 'email':*

*valueA = a.email.toLowerCase();*

*valueB = b.email.toLowerCase();*

*break;*

*case 'role':*

*valueA = a.role.toLowerCase();*

*valueB = b.role.toLowerCase();*

*break;*

*case 'tickets':*

*valueA = a.statistics.total\_tickets;*

*valueB = b.statistics.total\_tickets;*

*break;*

*default:*

*valueA = a.surname.toLowerCase();*

*valueB = b.surname.toLowerCase();*

*break;*

*}*

*if (userCurrentSort.direction === 'asc') {*

*return valueA > valueB ? 1 : -1;*

*} else {*

*return valueA < valueB ? 1 : -1;*

*}*

*});*

*users.forEach(function(user) {*

*const row = `*

*<tr>*

*<td>${user.id}</td>*

*<td>${user.surname}</td>*

*<td>${user.first\_name}</td>*

*<td>${user.patronymic || ''}</td>*

*<td>${user.email}</td>*

*<td>${user.role}</td>*

*<td>${user.statistics.total\_tickets}</td>*

*<td>*

*<button class="btn btn-sm btn-primary me-2" onclick="editUser(${user.id},*

*'${user.surname}', '${user.first\_name}', '${user.patronymic || ''}',*

*'${user.email}', '${user.role}')">*

*<i class="fas fa-edit"></i>Изменить*

*</button>*

*<button class="btn btn-sm btn-danger" onclick="confirmDelete(${user.id})">*

*<i class="fas fa-trash"></i>Удалить*

*</button>*

*</td>*

*</tr>*

*`;*

*tbody.append(row);*

*});*

*}*

*},*

*error: function(xhr) {*

*if (xhr.status === 401) {*

*window.location.href = '/auth/login';*

*} else {*

*alert("Ошибка при загрузке пользователей");*

*}*

*}*

*});*

*}*

*// Функция для очистки ошибок*

*function clearErrors() {*

*$('.error-message').text('');*

*$('.form-group').removeClass('error');*

*$('.form-group input, .form-group select').removeClass('error');*

*}*

*// Функция для отображения ошибок*

*function showErrors(errors) {*

*clearErrors();*

*if (errors && errors.length > 0) {*

*errors.forEach(error => {*

*const field = error.field;*

*const message = error.message;*

*const formGroup = $(`#${field}`).closest('.form-group');*

*formGroup.addClass('error');*

*$(`#${field}`).addClass('error');*

*$(`#${field}Error`).text(message);*

*});*

*}*

*}*

*// Функция валидации формы добавления пользователя*

*function validateAddUserForm() {*

*const surname = $('#userSurname').val().trim();*

*const first\_name = $('#userFirstName').val().trim();*

*const patronymic = $('#userPatronymic').val().trim();*

*const email = $('#userEmail').val().trim();*

*const password = $('#userPassword').val();*

*const passwordConfirm = $('#userPasswordConfirm').val();*

*const role = $('#userRole').val();*

*const errors = [];*

*if (!surname) {*

*errors.push({ field: 'userSurname', message: 'Фамилия обязательна' });*

*} else if (surname.length < 2 || surname.length > 50) {*

*errors.push({ field: 'userSurname', message: 'Фамилия должна быть от 2 до 50 символов' });*

*}*

*if (!first\_name) {*

*errors.push({ field: 'userFirstName', message: 'Имя обязательно' });*

*} else if (first\_name.length < 2 || first\_name.length > 50) {*

*errors.push({ field: 'userFirstName', message: 'Имя должно быть от 2 до 50 символов' });*

*}*

*if (patronymic && (patronymic.length < 2 || patronymic.length > 50)) {*

*errors.push({ field: 'userPatronymic', message: 'Отчество должно быть от 2 до 50 символов' });*

*}*

*if (!email) {*

*errors.push({ field: 'userEmail', message: 'Email обязателен' });*

*} else if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)) {*

*errors.push({ field: 'userEmail', message: 'Неверный формат email' });*

*}*

*if (!password) {*

*errors.push({ field: 'userPassword', message: 'Пароль обязателен' });*

*} else if (password.length < 8) {*

*errors.push({ field: 'userPassword', message: 'Пароль должен быть не менее 8 символов' });*

*}*

*if (!passwordConfirm) {*

*errors.push({ field: 'userPasswordConfirm', message: 'Подтверждение пароля обязательно' });*

*} else if (password !== passwordConfirm) {*

*errors.push({ field: 'userPasswordConfirm', message: 'Пароли не совпадают' });*

*}*

*if (!role) {*

*errors.push({ field: 'userRole', message: 'Роль обязательна' });*

*}*

*return errors;*

*}*

*// Функция валидации формы редактирования пользователя*

*function validateEditUserForm() {*

*const surname = $('#editUserSurname').val().trim();*

*const first\_name = $('#editUserFirstName').val().trim();*

*const patronymic = $('#editUserPatronymic').val().trim();*

*const email = $('#editUserEmail').val().trim();*

*const password = $('#editUserPassword').val();*

*const passwordConfirm = $('#editUserPasswordConfirm').val();*

*const role = $('#editUserRole').val();*

*const errors = [];*

*if (!surname) {*

*errors.push({ field: 'editUserSurname', message: 'Фамилия обязательна' });*

*} else if (surname.length < 2 || surname.length > 50) {*

*errors.push({ field: 'editUserSurname', message: 'Фамилия должна быть от 2 до 50 символов' });*

*}*

*if (!first\_name) {*

*errors.push({ field: 'editUserFirstName', message: 'Имя обязательно' });*

*} else if (first\_name.length < 2 || first\_name.length > 50) {*

*errors.push({ field: 'editUserFirstName', message: 'Имя должно быть от 2 до 50 символов' });*

*}*

*if (patronymic && (patronymic.length < 2 || patronymic.length > 50)) {*

*errors.push({ field: 'editUserPatronymic', message: 'Отчество должно быть от 2 до 50 символов' });*

*}*

*if (!email) {*

*errors.push({ field: 'editUserEmail', message: 'Email обязателен' });*

*} else if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)) {*

*errors.push({ field: 'editUserEmail', message: 'Неверный формат email' });*

*}*

*if (password) {*

*if (password.length < 8) {*

*errors.push({ field: 'editUserPassword', message: 'Пароль должен быть не менее 8 символов' });*

*}*

*if (!passwordConfirm) {*

*errors.push({ field: 'editUserPasswordConfirm', message: 'Подтверждение пароля обязательно' });*

*} else if (password !== passwordConfirm) {*

*errors.push({ field: 'editUserPasswordConfirm', message: 'Пароли не совпадают' });*

*}*

*}*

*if (!role) {*

*errors.push({ field: 'editUserRole', message: 'Роль обязательна' });*

*}*

*return errors;*

*}*

*// Функция добавления пользователя*

*function addUser(surname, first\_name, patronymic, email, password, role) {*

*$.ajax({*

*url: '/api/v1/users',*

*method: 'POST',*

*contentType: 'application/json',*

*data: JSON.stringify({ surname, first\_name, patronymic, email, password, role }),*

*success: function(response) {*

*if (response.status === "success") {*

*$('#addUserModal').removeClass('active');*

*clearErrors();*

*$('#userSurname').val('');*

*$('#userFirstName').val('');*

*$('#userPatronymic').val('');*

*$('#userEmail').val('');*

*$('#userPassword').val('');*

*$('#userRole').val('user');*

*loadUsers();*

*}*

*},*

*error: function(xhr) {*

*if (xhr.status === 401) {*

*window.location.href = '/auth/login';*

*} else if (xhr.status === 400) {*

*const response = xhr.responseJSON;*

*if (response.errors && response.errors.length > 0) {*

*showErrors(response.errors);*

*} else {*

*alert(response.message || "Ошибка при добавлении пользователя");*

*}*

*} else {*

*alert("Ошибка при добавлении пользователя");*

*}*

*}*

*});*

*}*

*// Функция редактирования пользователя*

*function editUser(id, surname, first\_name, patronymic, email, role) {*

*clearErrors();*

*$('#editUserId').val(id);*

*$('#editUserSurname').val(surname);*

*$('#editUserFirstName').val(first\_name);*

*$('#editUserPatronymic').val(patronymic);*

*$('#editUserEmail').val(email);*

*$('#editUserPassword').val(''); // Очищаем поле пароля*

*$('#editUserRole').val(role);*

*$('#editUserModal').addClass('active');*

*}*

*// Функция обновления пользователя*

*function updateUser(id, surname, first\_name, patronymic, email, password, role) {*

*// Преобразуем пустую строку в null для отчества*

*const patronymicValue = patronymic.trim() === '' ? null : patronymic;*

*const data = {*

*surname,*

*first\_name,*

*patronymic: patronymicValue,*

*email,*

*role*

*};*

*if (password) {*

*data.password = password;*

*}*

*$.ajax({*

*url: `/api/v1/users/${id}`,*

*method: 'PATCH',*

*contentType: 'application/json',*

*data: JSON.stringify(data),*

*success: function(response) {*

*if (response.status === "success") {*

*$('#editUserModal').removeClass('active');*

*clearErrors();*

*loadUsers();*

*}*

*},*

*error: function(xhr) {*

*if (xhr.status === 401) {*

*window.location.href = '/auth/login';*

*} else if (xhr.status === 400) {*

*const response = xhr.responseJSON;*

*if (response.errors && response.errors.length > 0) {*

*showErrors(response.errors);*

*} else {*

*alert(response.message || "Ошибка при обновлении пользователя");*

*}*

*} else {*

*alert("Ошибка при обновлении пользователя");*

*}*

*}*

*});*

*}*

*// Функция удаления пользователя*

*function deleteUser(id) {*

*$.ajax({*

*url: `/api/v1/users/${id}`,*

*method: 'DELETE',*

*success: function(response) {*

*if (response.status === "success") {*

*$('#deleteUserModal').removeClass('active');*

*loadUsers();*

*} else {*

*alert(response.message || "Ошибка при удалении пользователя");*

*}*

*},*

*error: function(xhr) {*

*if (xhr.status === 401) {*

*window.location.href = '/auth/login';*

*} else if (xhr.status === 400) {*

*const response = xhr.responseJSON;*

*if (response.data && response.data.ticketCount) {*

*alert(`Невозможно удалить пользователя: у него есть ${response.data.ticketCount} забронированных билетов`);*

*} else {*

*alert(response.message || "Невозможно удалить пользователя");*

*}*

*} else if (xhr.status === 404) {*

*alert("Пользователь не найден");*

*} else {*

*alert("Ошибка при удалении пользователя");*

*}*

*}*

*});*

*}*

*// Функция подтверждения удаления*

*function confirmDelete(id) {*

*$('#confirmDeleteBtn').data('userId', id);*

*$('#deleteUserModal').addClass('active');*

*}*

*// Добавляем очистку ошибок при открытии модальных окон*

*$('[data-modal="addUserModal"]').on('click', function() {*

*clearErrors();*

*});*

ПРИЛОЖЕНИЕ 36. Файл src\routes\artworkRoutes.js

*const express = require('express');*

*const router = express.Router();*

*const artworkController = require('../controllers/artworkController');*

*const auth = require('../middleware/auth');*

*const {validateArtwork, validateArtworkId, validateArtworkUpdate, validate} = require('../middleware/validation');*

*const upload = require('../middleware/upload');*

*const logRequest = (req, res, next) => {*

*const timestamp = new Date().toISOString();*

*console.log(`[${timestamp}] ${req.method} ${req.originalUrl} - User: ${req.user?.id || 'anonymous'}`);*

*next();*

*};*

*router.get('/', artworkController.getAllArtworks);*

*router.get('/:id', validateArtworkId, validate, artworkController.getArtwork);*

*router.get('/:id/exhibitions', artworkController.getArtworkExhibitions);*

*router.use(auth.protect);*

*router.use(auth.restrictTo('admin'));*

*router.use(logRequest);*

*router.post('/', upload.single('image\_path'), validateArtwork, validate, artworkController.createArtwork);*

*router.patch('/:id', upload.single('image\_path'), validateArtworkId, validateArtworkUpdate, validate, artworkController.updateArtwork);*

*router.delete('/:id', validateArtworkId, validate, artworkController.deleteArtwork);*

*module.exports = router;*

ПРИЛОЖЕНИЕ 37. Файл src\routes\authorRoutes.js

*const express = require('express');*

*const router = express.Router();*

*const authorController = require('../controllers/authorController');*

*const auth = require('../middleware/auth');*

*const { validateAuthor, validateAuthorUpdate, validateAuthorId, validate } = require('../middleware/validation');*

*const logRequest = (req, res, next) => {*

*const timestamp = new Date().toISOString();*

*console.log(`[${timestamp}] ${req.method} ${req.originalUrl} - User: ${req.user?.id || 'anonymous'}`);*

*next();*

*};*

*router.get('/', authorController.getAllAuthors);*

*router.get('/:id', validateAuthorId, validate, authorController.getAuthor);*

*router.use(auth.protect);*

*router.use(auth.restrictTo('admin'));*

*router.use(logRequest);*

*router.post('/', validateAuthor, validate, authorController.createAuthor);*

*router.patch('/:id', validateAuthorId, validateAuthorUpdate, validate, authorController.updateAuthor);*

*router.delete('/:id', validateAuthorId, validate, authorController.deleteAuthor);*

*module.exports = router;*

ПРИЛОЖЕНИЕ 38. Файл src\routes\authRoutes.js

*const express = require('express');*

*const router = express.Router();*

*const authController = require('../controllers/authController');*

*const auth = require('../middleware/auth');*

*const { validateLogin, validateRegister, validatePasswordChange, validate } = require('../middleware/validation');*

*const logRequest = (req, res, next) => {*

*const timestamp = new Date().toISOString();*

*console.log(`[${timestamp}] ${req.method} ${req.originalUrl} - User: ${req.user?.id || 'anonymous'}`);*

*next();*

*};*

*router.post('/register', validateRegister, validate, authController.register);*

*router.post('/login', validateLogin, validate, authController.login);*

*router.post('/refresh-token', authController.refreshToken);*

*router.use(auth.protect);*

*router.use(logRequest);*

*router.delete('/logout', authController.logout);*

*router.patch('/change-password', validatePasswordChange, validate, authController.changePassword);*

*router.get('/me', authController.getMe);*

*module.exports = router;*

ПРИЛОЖЕНИЕ 39. Файл src\routes\exhibitionRoutes.js

const express = require('express');

const router = express.Router();

const exhibitionController = require('../controllers/exhibitionController');

const auth = require('../middleware/auth');

const { validateExhibition, validateExhibitionId, validateExhibitionUpdate, validate } = require('../middleware/validation');

const upload = require('../middleware/upload');

const logRequest = (req, res, next) => {

const timestamp = new Date().toISOString();

console.log(`[${timestamp}] ${req.method} ${req.originalUrl} - User: ${req.user?.id || 'anonymous'}`);

next();

};

// Публичные маршруты (до middleware аутентификации)

router.get('/public', exhibitionController.getPublicExhibitions);

// Публичная страница выставки (рендер EJS)

router.get('/:id/page', validateExhibitionId, validate, exhibitionController.renderExhibitionPage);

// Защищенные маршруты

router.get('/', exhibitionController.getAllExhibitions);

router.get('/:id', validateExhibitionId, validate, exhibitionController.getExhibition);

router.use(auth.protect);

router.use(auth.restrictTo('admin', 'manager'));

router.use(logRequest);

router.post('/', upload.single('poster'), validateExhibition, validate, exhibitionController.createExhibition);

router.patch('/:id', upload.single('poster'), validateExhibitionId, validateExhibitionUpdate, validate, exhibitionController.updateExhibition);

router.delete('/:id', validateExhibitionId, validate, exhibitionController.deleteExhibition);

module.exports = router;

ПРИЛОЖЕНИЕ 40. Файл src\routes\genreRoutes.js

const express = require('express');

const router = express.Router();

const genreController = require('../controllers/genreController');

const auth = require('../middleware/auth');

const { validateGenre, validateGenreUpdate, validateGenreId, validate } = require('../middleware/validation');

const logRequest = (req, res, next) => {

const timestamp = new Date().toISOString();

console.log(`[${timestamp}] ${req.method} ${req.originalUrl} - User: ${req.user?.id || 'anonymous'}`);

next();

};

router.get('/', genreController.getAllGenres);

router.get('/:id', validateGenreId, validate, genreController.getGenre);

router.use(auth.protect);

router.use(auth.restrictTo('admin'));

router.use(logRequest);

router.post('/', validateGenre, validate, genreController.createGenre);

router.patch('/:id', validateGenreId, validateGenreUpdate, validate, genreController.updateGenre);

router.delete('/:id', validateGenreId, validate, genreController.deleteGenre);

module.exports = router;

ПРИЛОЖЕНИЕ 41. Файл src\routes\index.js

const express = require('express');

const router = express.Router();

// Импорт API-маршрутов

const authRoutes = require('./authRoutes');

const exhibitionApi = require('./exhibitionRoutes');

const ticketRoutes = require('./ticketRoutes');

const styleRoutes = require('./styleRoutes');

const genreRoutes = require('./genreRoutes');

const authorRoutes = require('./authorRoutes');

const artworkApiRoutes = require('./artworkRoutes');

const userRoutes = require('./userRoutes');

// Группировка маршрутов по версиям API

const apiV1 = express.Router();

// Публичные API-маршруты

apiV1.use('/auth', authRoutes);

apiV1.use('/exhibitions', exhibitionApi);

apiV1.use('/authors', authorRoutes);

apiV1.use('/styles', styleRoutes);

apiV1.use('/genres', genreRoutes);

apiV1.use('/artworks', artworkApiRoutes);

// Защищённые API-маршруты

apiV1.use('/tickets', ticketRoutes);

apiV1.use('/users', userRoutes);

// Монтируем API v1

router.use('/api/v1', apiV1);

router.use('/api/v1/reports', require('./reportRoutes'));

// Страница списка выставок (HTML)

const exhibitionController = require('../controllers/exhibitionController');

router.get('/exhibitions', exhibitionController.renderExhibitionsPage);

// Страница списка картин (HTML)

const artworkController = require('../controllers/artworkController');

router.get('/artworks', artworkController.renderArtworksPage);

const ticketController = require('../controllers/ticketController');

router.get('/tickets', ticketController.renderUserTickets);

const auth = require('../middleware/auth');

const managerController = require('../controllers/managerController');

router.get('/manager/statistics',auth.protect, auth.restrictTo('admin','manager'), managerController.renderStatistics);

module.exports = router;

ПРИЛОЖЕНИЕ 42. Файл src\routes\reportRoutes.js

const express = require('express');

const router = express.Router();

const auth = require('../middleware/auth');

const { getTicketStats } = require('../controllers/managerController');

// GET /api/v1/reports/tickets?start\_date=YYYY-MM-DD&end\_date=YYYY-MM-DD

router.get(

'/tickets',

auth.protect,

auth.restrictTo('admin','manager'),

getTicketStats

);

module.exports = router;

ПРИЛОЖЕНИЕ 43. Файл src\routes\styleRoutes.js

const express = require('express');

const router = express.Router();

const styleController = require('../controllers/styleController');

const auth = require('../middleware/auth');

const { validateStyle, validateStyleUpdate, validateStyleId, validate } = require('../middleware/validation');

const logRequest = (req, res, next) => {

const timestamp = new Date().toISOString();

console.log(`[${timestamp}] ${req.method} ${req.originalUrl} - User: ${req.user?.id || 'anonymous'}`);

next();

};

router.get('/', styleController.getAllStyles);

router.get('/:id', validateStyleId, validate, styleController.getStyle);

router.use(auth.protect);

router.use(auth.restrictTo('admin', 'manager'));

router.use(logRequest);

router.post('/', validateStyle, validate, styleController.createStyle);

router.patch('/:id', validateStyleId, validateStyleUpdate, validate, styleController.updateStyle);

router.delete('/:id', validateStyleId, validate, styleController.deleteStyle);

module.exports = router;

ПРИЛОЖЕНИЕ 44. Файл src\routes\ticketRoutes.js

const express = require('express');

const router = express.Router();

const { createTicket, getAllTickets, getMyTickets, getTicket, cancelTicket, createTicketAdmin, updateTicketAdmin} = require('../controllers/ticketController');

const auth = require('../middleware/auth');

const { validateTicket, validateTicketId, validate } = require('../middleware/validation');

const logRequest = (req, res, next) => {

const timestamp = new Date().toISOString();

console.log(`[${timestamp}] ${req.method} ${req.originalUrl} - User: ${req.user?.id || 'anonymous'}`);

next();

};

router.use(auth.protect);

router.use(logRequest);

router.post('/', auth.restrictTo('user'), validateTicket, validate, createTicket);

router.get('/my-tickets', getMyTickets);

router.get('/:id', validateTicketId, validate, getTicket);

router.delete('/:id', validateTicketId, validate, cancelTicket);

router.post('/admin/tickets', auth.restrictTo('admin'), createTicketAdmin);

router.patch(

'/:id',

auth.restrictTo('admin'),

validateTicketId,

validate,

updateTicketAdmin

);

router.use(auth.restrictTo('admin'));

router.get('/', getAllTickets);

module.exports = router;

ПРИЛОЖЕНИЕ 45. Файл src\routes\userRoutes.js

const express = require('express');

const router = express.Router();

const userController = require('../controllers/userController');

const auth = require('../middleware/auth');

const { validateCreateUser, validateUserId, validateUserUpdate, validate } = require('../middleware/validation');

const logRequest = (req, res, next) => {

const timestamp = new Date().toISOString();

console.log(`[${timestamp}] ${req.method} ${req.originalUrl} - User: ${req.user?.id || 'anonymous'}`);

next();

};

router.use(auth.protect);

router.patch('/me', validateUserUpdate, validate, userController.updateMe);

router.use(auth.restrictTo('admin'));

router.use(logRequest);

router.get('/me', userController.getMe);

router.post('/', validateCreateUser, validate, userController.createUser);

router.get('/', userController.getAllUsers);

router.get('/:id', validateUserId, validate, userController.getUser);

router.patch('/:id', validateUserId, validateUserUpdate, validate, userController.updateUser);

router.delete('/:id', validateUserId, validate, userController.deleteUser);

module.exports = router;

ПРИЛОЖЕНИЕ 46. Файл src\scripts\createAdmin.js

const express = require('express');

const router = express.Router();

const userController = require('../controllers/userController');

const auth = require('../middleware/auth');

const { validateCreateUser, validateUserId, validateUserUpdate, validate } = require('../middleware/validation');

const logRequest = (req, res, next) => {

const timestamp = new Date().toISOString();

console.log(`[${timestamp}] ${req.method} ${req.originalUrl} - User: ${req.user?.id || 'anonymous'}`);

next();

};

router.use(auth.protect);

router.patch('/me', validateUserUpdate, validate, userController.updateMe);

router.use(auth.restrictTo('admin'));

router.use(logRequest);

router.get('/me', userController.getMe);

router.post('/', validateCreateUser, validate, userController.createUser);

router.get('/', userController.getAllUsers);

router.get('/:id', validateUserId, validate, userController.getUser);

router.patch('/:id', validateUserId, validateUserUpdate, validate, userController.updateUser);

router.delete('/:id', validateUserId, validate, userController.deleteUser);

module.exports = router;

ПРИЛОЖЕНИЕ 47. Файл src\views\admin\artworks.ejs

<div class="admin-container table-extra-large">

<div class="admin-header">

<h1>Управление произведениями искусства</h1>

<button type="button" class="btn btn-primary" data-modal="addArtworkModal">

<i class="fas fa-plus"></i>Добавить произведение

</button>

</div>

<!-- Поиск и фильтрация -->

<div class="search-filter-section centered-search">

<!-- Поисковая строка -->

<div class="search-section">

<div class="search-container">

<input

type="text"

id="searchInput"

class="search-input"

placeholder="Поиск произведений..."

aria-label="Поиск произведений"

/>

</div>

</div>

<!-- Фильтры на одной строке -->

<div class="filters-row">

<!-- Фильтрация по году создания -->

<div class="filter-section">

<div class="filter-container">

<div class="filter-group">

<label for="yearFrom">Год создания от:</label>

<input

type="number"

id="yearFrom"

class="filter-input"

min="0"

max="<%= new Date().getFullYear() %>"

placeholder="Год"

aria-label="Год начала периода"

/>

</div>

<div class="filter-group">

<label for="yearTo">до:</label>

<input

type="number"

id="yearTo"

class="filter-input"

min="0"

max="<%= new Date().getFullYear() %>"

placeholder="Год"

aria-label="Год окончания периода"

/>

</div>

<button type="button" class="btn btn-secondary" id="clearYearFilter">

<i class="fas fa-times"></i>Очистить

</button>

</div>

</div>

</div>

</div>

<!-- Таблица произведений -->

<div class="table-container">

<table class="data-table">

<thead>

<tr>

<th class="sortable" data-column="id">ID</th>

<th>Изображение</th>

<th class="sortable" data-column="title">Название</th>

<th class="sortable" data-column="author">Автор</th>

<th class="sortable" data-column="style">Стиль</th>

<th class="sortable" data-column="genre">Жанр</th>

<th class="sortable" data-column="creation\_year">Год создания</th>

<th class="sortable" data-column="dimensions">Размеры</th>

<th class="sortable" data-column="exhibitions">Выставки</th>

<th>Действия</th>

</tr>

</thead>

<tbody id="artworksTableBody">

<!-- Данные будут загружены через JavaScript -->

</tbody>

</table>

</div>

</div>

<!-- Модальное окно для просмотра изображения -->

<div class="modal" id="imagePreviewModal">

<div class="modal-content">

<div class="modal-body">

<img

id="previewImage"

src=""

alt="Предпросмотр изображения"

style="max-width: 100%; height: auto"

/>

</div>

</div>

</div>

<!-- Модальное окно добавления произведения -->

<div class="modal" id="addArtworkModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Добавить произведение</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<form id="addArtworkForm">

<div class="form-group">

<label for="artworkTitle">Название</label>

<input type="text" id="artworkTitle" required />

<div class="error-message" id="artworkTitleError"></div>

</div>

<div class="form-group">

<label for="artworkAuthor">Автор</label>

<select id="artworkAuthor" required>

<!-- Опции будут загружены через JavaScript -->

</select>

<div class="error-message" id="artworkAuthorError"></div>

</div>

<div class="form-group">

<label for="artworkStyle">Стиль</label>

<select id="artworkStyle" required>

<!-- Опции будут загружены через JavaScript -->

</select>

<div class="error-message" id="artworkStyleError"></div>

</div>

<div class="form-group">

<label for="artworkGenre">Жанр</label>

<select id="artworkGenre" required>

<!-- Опции будут загружены через JavaScript -->

</select>

<div class="error-message" id="artworkGenreError"></div>

</div>

<div class="form-group">

<label for="artworkCreationYear">Год создания</label>

<input

type="number"

id="artworkCreationYear"

min="0"

max="<%= new Date().getFullYear() %>"

/>

<div class="error-message" id="artworkCreationYearError"></div>

</div>

<div class="form-group">

<label for="artworkWidth">Ширина (см)</label>

<input type="number" id="artworkWidth" min="0" step="0.01" />

<div class="error-message" id="artworkWidthError"></div>

</div>

<div class="form-group">

<label for="artworkHeight">Высота (см)</label>

<input type="number" id="artworkHeight" min="0" step="0.01" />

<div class="error-message" id="artworkHeightError"></div>

</div>

<div class="form-group">

<label for="artworkDescription">Описание</label>

<textarea

id="artworkDescription"

rows="4"

style="width: 100%; box-sizing: border-box"

></textarea>

<div class="error-message" id="artworkDescriptionError"></div>

</div>

<div class="form-group">

<label for="artworkImage">Изображение</label>

<input type="file" id="artworkImage" accept="image/\*" required />

<div class="error-message" id="artworkImageError"></div>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="saveArtworkBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно редактирования произведения -->

<div class="modal" id="editArtworkModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Редактировать произведение</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<form id="editArtworkForm">

<input type="hidden" id="editArtworkId" />

<div class="form-group">

<label for="editArtworkTitle">Название</label>

<input type="text" id="editArtworkTitle" required />

<div class="error-message" id="editArtworkTitleError"></div>

</div>

<div class="form-group">

<label for="editArtworkAuthor">Автор</label>

<select id="editArtworkAuthor" required>

<!-- Опции будут загружены через JavaScript -->

</select>

<div class="error-message" id="editArtworkAuthorError"></div>

</div>

<div class="form-group">

<label for="editArtworkStyle">Стиль</label>

<select id="editArtworkStyle" required>

<!-- Опции будут загружены через JavaScript -->

</select>

<div class="error-message" id="editArtworkStyleError"></div>

</div>

<div class="form-group">

<label for="editArtworkGenre">Жанр</label>

<select id="editArtworkGenre" required>

<!-- Опции будут загружены через JavaScript -->

</select>

<div class="error-message" id="editArtworkGenreError"></div>

</div>

<div class="form-group">

<label for="editArtworkCreationYear">Год создания</label>

<input

type="number"

id="editArtworkCreationYear"

min="0"

max="<%= new Date().getFullYear() %>"

/>

<div class="error-message" id="editArtworkCreationYearError"></div>

</div>

<div class="form-group">

<label for="editArtworkWidth">Ширина (см)</label>

<input type="number" id="editArtworkWidth" min="0" step="0.01" />

<div class="error-message" id="editArtworkWidthError"></div>

</div>

<div class="form-group">

<label for="editArtworkHeight">Высота (см)</label>

<input type="number" id="editArtworkHeight" min="0" step="0.01" />

<div class="error-message" id="editArtworkHeightError"></div>

</div>

<div class="form-group">

<label for="editArtworkDescription">Описание</label>

<textarea

id="editArtworkDescription"

rows="4"

style="width: 100%; box-sizing: border-box"

></textarea>

<div class="error-message" id="editArtworkDescriptionError"></div>

</div>

<div class="form-group">

<label for="editArtworkImage">Изображение</label>

<input type="file" id="editArtworkImage" accept="image/\*" />

<div class="error-message" id="editArtworkImageError"></div>

<div id="currentImagePreview" class="image-preview"></div>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="updateArtworkBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно подтверждения удаления -->

<div class="modal" id="deleteArtworkModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Подтверждение удаления</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<p>Вы уверены, что хотите удалить это произведение?</p>

<p class="warning-text">

Внимание: Если произведение участвует в выставках, удаление будет

невозможно.

</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-danger" id="confirmDeleteBtn">

Удалить

</button>

</div>

</div>

</div>

<script src="/js/artworks.js"></script>

ПРИЛОЖЕНИЕ 48. Файл src\views\admin\authors.ejs

<div class="admin-container table-small">

<div class="admin-header">

<h1>Управление авторами</h1>

<button type="button" class="btn btn-primary" data-modal="addAuthorModal">

<i class="fas fa-plus"></i>Добавить автора

</button>

</div>

<!-- Поиск и фильтрация -->

<div class="search-filter-section centered-search">

<!-- Поисковая строка -->

<div class="search-section">

<div class="search-container">

<input

type="text"

id="searchInput"

class="search-input"

placeholder="Поиск авторов..."

aria-label="Поиск авторов"

/>

</div>

</div>

<!-- Фильтры на одной строке -->

<div class="filters-row">

<!-- Фильтрация по дате -->

<div class="filter-section">

<div class="filter-container">

<div class="filter-group">

<label for="dateFrom">Период жизни от:</label>

<input

type="date"

id="dateFrom"

class="filter-input"

aria-label="Дата начала периода"

/>

</div>

<div class="filter-group">

<label for="dateTo">до:</label>

<input

type="date"

id="dateTo"

class="filter-input"

aria-label="Дата окончания периода"

/>

</div>

<button type="button" class="btn btn-secondary" id="clearDateFilter">

<i class="fas fa-times"></i>Очистить

</button>

</div>

</div>

</div>

</div>

<!-- Таблица авторов -->

<div class="table-container">

<table class="data-table">

<thead>

<tr>

<th class="sortable" data-column="id">ID</th>

<th class="sortable" data-column="surname">Фамилия</th>

<th class="sortable" data-column="first\_name">Имя</th>

<th class="sortable" data-column="patronymic">Отчество</th>

<th class="sortable" data-column="date\_of\_birth">Дата рождения</th>

<th class="sortable" data-column="date\_of\_death">Дата смерти</th>

<th class="sortable" data-column="artworks">

Количество произведений

</th>

<th>Действия</th>

</tr>

</thead>

<tbody id="authorsTableBody">

<!-- Данные будут загружены через JavaScript -->

</tbody>

</table>

</div>

</div>

<!-- Модальное окно добавления автора -->

<div class="modal" id="addAuthorModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Добавить автор</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<form id="addAuthorForm">

<div class="form-group">

<label for="authorSurname">Фамилия</label>

<input type="text" id="authorSurname" required />

<div class="error-message" id="authorSurnameError"></div>

</div>

<div class="form-group">

<label for="authorFirstName">Имя</label>

<input type="text" id="authorFirstName" required />

<div class="error-message" id="authorFirstNameError"></div>

</div>

<div class="form-group">

<label for="authorPatronymic">Отчество</label>

<input type="text" id="authorPatronymic" />

<div class="error-message" id="authorPatronymicError"></div>

</div>

<div class="form-group">

<label for="authorDateOfBirth">Дата рождения</label>

<input type="date" id="authorDateOfBirth" required />

<div class="error-message" id="authorDateOfBirthError"></div>

</div>

<div class="form-group">

<label for="authorDateOfDeath">Дата смерти</label>

<input type="date" id="authorDateOfDeath" required />

<div class="error-message" id="authorDateOfDeathError"></div>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="saveAuthorBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно редактирования автора -->

<div class="modal" id="editAuthorModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Редактировать автора</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<form id="editAuthorForm">

<input type="hidden" id="editAuthorId" />

<div class="form-group">

<label for="editAuthorSurname">Фамилия</label>

<input type="text" id="editAuthorSurname" required />

<div class="error-message" id="editAuthorSurnameError"></div>

</div>

<div class="form-group">

<label for="editAuthorFirstName">Имя</label>

<input type="text" id="editAuthorFirstName" required />

<div class="error-message" id="editAuthorFirstNameError"></div>

</div>

<div class="form-group">

<label for="editAuthorPatronymic">Отчество</label>

<input type="text" id="editAuthorPatronymic" />

<div class="error-message" id="editAuthorPatronymicError"></div>

</div>

<div class="form-group">

<label for="editAuthorDateOfBirth">Дата рождения</label>

<input type="date" id="editAuthorDateOfBirth" required />

<div class="error-message" id="editAuthorDateOfBirthError"></div>

</div>

<div class="form-group">

<label for="editAuthorDateOfDeath">Дата смерти</label>

<input type="date" id="editAuthorDateOfDeath" required />

<div class="error-message" id="editAuthorDateOfDeathError"></div>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="updateAuthorBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно подтверждения удаления -->

<div class="modal" id="deleteAuthorModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Подтверждение удаления</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<p>Вы уверены, что хотите удалить этого автора?</p>

<p class="warning-text">

Внимание: Если с автором связаны произведения, удаление будет

невозможно.

</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-danger" id="confirmDeleteBtn">

Удалить

</button>

</div>

</div>

</div>

<script src="/js/authors.js"></script>

ПРИЛОЖЕНИЕ 49. Файл src\views\admin\genres.ejs

<div class="admin-container table-small">

<div class="admin-header">

<h1>Управление жанрами</h1>

<button type="button" class="btn btn-primary" data-modal="addGenreModal">

<i class="fas fa-plus"></i>Добавить жанр

</button>

</div>

<!-- Поисковая строка -->

<div class="search-section centered">

<div class="search-container">

<input

type="text"

id="searchInput"

class="search-input"

placeholder="Поиск жанров..."

aria-label="Поиск жанров"

/>

</div>

</div>

<!-- Таблица жанров -->

<div class="table-container">

<table class="data-table">

<thead>

<tr>

<th class="sortable" data-column="id">ID</th>

<th class="sortable" data-column="name">Название</th>

<th class="sortable" data-column="artworks">

Количество произведений

</th>

<th>Действия</th>

</tr>

</thead>

<tbody id="genresTableBody">

<!-- Данные будут загружены через JavaScript -->

</tbody>

</table>

</div>

</div>

<!-- Модальное окно добавления жанра -->

<div class="modal" id="addGenreModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Добавить жанр</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<div class="modal-error" id="addGenreError"></div>

<form id="addGenreForm">

<div class="form-group">

<label for="genreName">Название жанра</label>

<input type="text" id="genreName" required />

<span class="error-message"></span>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="saveGenreBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно редактирования жанра -->

<div class="modal" id="editGenreModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Редактировать жанр</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<div class="modal-error" id="editGenreError"></div>

<form id="editGenreForm">

<input type="hidden" id="editGenreId" />

<div class="form-group">

<label for="editGenreName">Название жанра</label>

<input type="text" id="editGenreName" required />

<span class="error-message"></span>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="updateGenreBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно подтверждения удаления -->

<div class="modal" id="deleteGenreModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Подтверждение удаления</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<p>Вы уверены, что хотите удалить этот жанр?</p>

<p class="warning-text">

Внимание: Если с жанром связаны произведения, удаление будет невозможно.

</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-danger" id="confirmDeleteBtn">

Удалить

</button>

</div>

</div>

</div>

<script src="/js/genres.js"></script>

ПРИЛОЖЕНИЕ 50. Файл src\views\admin\index.ejs

<div class="content-container">

<h1 style="margin-bottom: 2rem">Панель администратора</h1>

<div class="grid grid-3">

<div class="card">

<div class="card-body">

<a href="/admin/users" class="btn btn-primary"

>Управление пользователями</a

>

</div>

</div>

<div class="card">

<div class="card-body">

<a href="/admin/authors" class="btn btn-primary">Управление авторами</a>

</div>

</div>

<div class="card">

<div class="card-body">

<a href="/admin/styles" class="btn btn-primary"

>Управление художественными стилями</a

>

</div>

</div>

<div class="card">

<div class="card-body">

<a href="/admin/genres" class="btn btn-primary"

>Управление художественными жанрами</a

>

</div>

</div>

<div class="card">

<div class="card-body">

<a href="/manager/exhibitions" class="btn btn-primary"

>Управление выставками</a

>

</div>

</div>

<div class="card">

<div class="card-body">

<a href="/admin/artworks" class="btn btn-primary"

>Управление произведениями искусства</a

>

</div>

</div>

<div class="card">

<div class="card-body">

<a href="/admin/tickets" class="btn btn-primary">Управление билетами</a>

</div>

</div>

</div>

</div>

ПРИЛОЖЕНИЕ 51. Файл src\views\admin\styles.ejs

<div class="admin-container table-small">

<div class="admin-header">

<h1>Управление стилями</h1>

<button type="button" class="btn btn-primary" data-modal="addStyleModal">

<i class="fas fa-plus"></i>Добавить стиль

</button>

</div>

<!-- Поисковая строка-->

<div class="search-section centered">

<div class="search-container">

<input

type="text"

id="searchInput"

class="search-input"

placeholder="Поиск стилей..."

aria-label="Поиск стилей"

/>

</div>

</div>

<!-- Таблица стилей -->

<div class="table-container">

<table class="data-table">

<thead>

<tr>

<th class="sortable" data-column="id">ID</th>

<th class="sortable" data-column="name">Название</th>

<th class="sortable" data-column="artworks">

Количество произведений

</th>

<th>Действия</th>

</tr>

</thead>

<tbody id="stylesTableBody">

<!-- Данные будут загружены через JavaScript -->

</tbody>

</table>

</div>

</div>

<!-- Модальное окно добавления стиля -->

<div class="modal" id="addStyleModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Добавить стиль</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<div class="modal-error" id="addStyleError"></div>

<form id="addStyleForm">

<div class="form-group">

<label for="styleName">Название стиля</label>

<input type="text" id="styleName" required />

<span class="error-message"></span>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="saveStyleBtn">

<i class="fas fa-save"></i> Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно редактирования стиля -->

<div class="modal" id="editStyleModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Редактировать стиль</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<div class="modal-error" id="editStyleError"></div>

<form id="editStyleForm">

<input type="hidden" id="editStyleId" />

<div class="form-group">

<label for="editStyleName">Название стиля</label>

<input type="text" id="editStyleName" required />

<span class="error-message"></span>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="updateStyleBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно подтверждения удаления -->

<div class="modal" id="deleteStyleModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Подтверждение удаления</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<p>Вы уверены, что хотите удалить этот стиль?</p>

<p class="warning-text">

Внимание: Если со стилем связаны произведения, удаление будет

невозможно.

</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-danger" id="confirmDeleteBtn">

Удалить

</button>

</div>

</div>

</div>

<script src="/js/styles.js"></script>

ПРИЛОЖЕНИЕ 52. Файл src\views\admin\tickets.ejs

<div class="admin-container table-medium">

<div class="admin-header">

<h1>Управление билетами</h1>

<button type="button" class="btn btn-primary" data-modal="addTicketModal">

<i class="fas fa-plus"></i>Добавить билет

</button>

</div>

<!-- Поиск и фильтры -->

<div class="search-filter-section centered-search">

<!-- Поисковая строка -->

<div class="search-section">

<div class="search-container">

<input

type="text"

id="searchTicketInput"

class="search-input"

placeholder="Поиск билетов..."

aria-label="Поиск билетов"

/>

</div>

</div>

<!-- Фильтры на одной строке -->

<div class="filters-row">

<!-- Фильтрация по дате бронирования -->

<div class="filter-section">

<div class="filter-container">

<div class="filter-group">

<label for="bookingDateFrom">Дата бронирования от:</label>

<input

type="date"

id="bookingDateFrom"

class="filter-input"

aria-label="Дата начала периода бронирования"

/>

</div>

<div class="filter-group">

<label for="bookingDateTo">до:</label>

<input

type="date"

id="bookingDateTo"

class="filter-input"

aria-label="Дата окончания периода бронирования"

/>

</div>

<button

type="button"

class="btn btn-secondary"

id="clearBookingDateFilter"

>

<i class="fas fa-times"></i>Очистить

</button>

</div>

</div>

<!-- Фильтрация по цене -->

<div class="filter-section">

<div class="filter-container">

<div class="filter-group">

<label for="priceFrom">Цена от:</label>

<input

type="number"

id="priceFrom"

class="filter-input"

min="0"

step="0.01"

placeholder="₽"

aria-label="Минимальная цена"

/>

</div>

<div class="filter-group">

<label for="priceTo">до:</label>

<input

type="number"

id="priceTo"

class="filter-input"

min="0"

step="0.01"

placeholder="₽"

aria-label="Максимальная цена"

/>

</div>

<button type="button" class="btn btn-secondary" id="clearPriceFilter">

<i class="fas fa-times"></i>Очистить

</button>

</div>

</div>

</div>

</div>

<!-- Таблица билетов -->

<div class="table-container">

<table class="data-table">

<thead>

<tr>

<th class="sortable" data-column="id">ID</th>

<th class="sortable" data-column="user">Пользователь</th>

<th class="sortable" data-column="exhibition">Выставка</th>

<th class="sortable" data-column="booking\_date">Дата бронирования</th>

<th class="sortable" data-column="total\_price">Цена</th>

<th class="sortable" data-column="quantity">Количество</th>

<th>Действия</th>

</tr>

</thead>

<tbody id="ticketsTableBody">

<!-- Данные будут загружены через JavaScript -->

</tbody>

</table>

</div>

</div>

<!-- Модальное окно добавления билета -->

<div class="modal" id="addTicketModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Добавить билет</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<form id="addTicketForm">

<div class="form-group">

<label for="ticketUser">Пользователь</label>

<select id="ticketUser" required></select>

<div class="error-message" id="ticketUserError"></div>

</div>

<div class="form-group">

<label for="ticketExhibition">Выставка</label>

<select id="ticketExhibition" required></select>

<div class="error-message" id="ticketExhibitionError"></div>

</div>

<div class="form-group">

<label for="ticketQuantity">Количество</label>

<input type="number" id="ticketQuantity" min="1" required />

<div class="error-message" id="ticketQuantityError"></div>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="saveTicketBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно редактирования билета -->

<div class="modal" id="editTicketModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Редактировать билет</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<form id="editTicketForm">

<input type="hidden" id="editTicketId" />

<div class="form-group">

<label for="editTicketUser">Пользователь</label>

<select id="editTicketUser" required></select>

<div class="error-message" id="editTicketUserError"></div>

</div>

<div class="form-group">

<label for="editTicketExhibition">Выставка</label>

<select id="editTicketExhibition" required></select>

<div class="error-message" id="editTicketExhibitionError"></div>

</div>

<div class="form-group">

<label for="editTicketQuantity">Количество</label>

<input type="number" id="editTicketQuantity" min="1" required />

<div class="error-message" id="editTicketQuantityError"></div>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="updateTicketBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно подтверждения удаления -->

<div class="modal" id="deleteTicketModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Подтверждение удаления</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<p>Вы уверены, что хотите удалить этот билет?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-danger" id="confirmDeleteTicketBtn">

Удалить

</button>

</div>

</div>

</div>

<script src="/js/ticket.js"></script>

ПРИЛОЖЕНИЕ 53. Файл src\views\admin\users.ejs

<div class="admin-container table-medium">

<div class="admin-header">

<h1>Управление пользователями</h1>

<button type="button" class="btn btn-primary" data-modal="addUserModal">

<i class="fas fa-plus"></i>Добавить пользователя

</button>

</div>

<!-- Поисковая строка -->

<div class="search-section centered">

<div class="search-container">

<input

type="text"

id="searchInput"

class="search-input"

placeholder="Поиск пользователей..."

aria-label="Поиск пользователей"

/>

</div>

</div>

<!-- Таблица пользователей -->

<div class="table-container">

<table class="data-table">

<thead>

<tr>

<th class="sortable" data-column="id">ID</th>

<th class="sortable" data-column="surname">Фамилия</th>

<th class="sortable" data-column="first\_name">Имя</th>

<th class="sortable" data-column="patronymic">Отчество</th>

<th class="sortable" data-column="email">Email</th>

<th class="sortable" data-column="role">Роль</th>

<th class="sortable" data-column="tickets">Количество билетов</th>

<th>Действия</th>

</tr>

</thead>

<tbody id="usersTableBody">

<!-- Данные будут загружены через JavaScript -->

</tbody>

</table>

</div>

</div>

<!-- Модальное окно добавления пользователя -->

<div class="modal" id="addUserModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Добавить пользователя</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<form id="addUserForm">

<div class="form-group">

<label for="userSurname">Фамилия</label>

<input type="text" id="userSurname" required />

<div class="error-message" id="userSurnameError"></div>

</div>

<div class="form-group">

<label for="userFirstName">Имя</label>

<input type="text" id="userFirstName" required />

<div class="error-message" id="userFirstNameError"></div>

</div>

<div class="form-group">

<label for="userPatronymic">Отчество</label>

<input type="text" id="userPatronymic" />

<div class="error-message" id="userPatronymicError"></div>

</div>

<div class="form-group">

<label for="userEmail">Email</label>

<input type="email" id="userEmail" required />

<div class="error-message" id="userEmailError"></div>

</div>

<div class="form-group">

<label for="userPassword">Пароль</label>

<div class="password-wrapper">

<input type="password" id="userPassword" required />

<span class="toggle-password" data-target="userPassword"

><i class="fas fa-eye"></i

></span>

</div>

<div class="error-message" id="userPasswordError"></div>

</div>

<div class="form-group">

<label for="userPasswordConfirm">Подтверждение пароля</label>

<div class="password-wrapper">

<input type="password" id="userPasswordConfirm" required />

<span class="toggle-password" data-target="userPasswordConfirm"

><i class="fas fa-eye"></i

></span>

</div>

<div class="error-message" id="userPasswordConfirmError"></div>

</div>

<div class="form-group">

<label for="userRole">Роль</label>

<select id="userRole" required>

<option value="user">Пользователь</option>

<option value="manager">Менеджер</option>

<option value="admin">Администратор</option>

</select>

<div class="error-message" id="userRoleError"></div>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="saveUserBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно редактирования пользователя -->

<div class="modal" id="editUserModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Редактировать пользователя</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<form id="editUserForm">

<input type="hidden" id="editUserId" />

<div class="form-group">

<label for="editUserSurname">Фамилия</label>

<input type="text" id="editUserSurname" required />

<div class="error-message" id="editUserSurnameError"></div>

</div>

<div class="form-group">

<label for="editUserFirstName">Имя</label>

<input type="text" id="editUserFirstName" required />

<div class="error-message" id="editUserFirstNameError"></div>

</div>

<div class="form-group">

<label for="editUserPatronymic">Отчество</label>

<input type="text" id="editUserPatronymic" />

<div class="error-message" id="editUserPatronymicError"></div>

</div>

<div class="form-group">

<label for="editUserEmail">Email</label>

<input type="email" id="editUserEmail" required />

<div class="error-message" id="editUserEmailError"></div>

</div>

<div class="form-group">

<label for="editUserPassword">Новый пароль</label>

<div class="password-wrapper">

<input type="password" id="editUserPassword" />

<span class="toggle-password" data-target="editUserPassword"

><i class="fas fa-eye"></i

></span>

</div>

<small class="form-text text-muted"

>Оставьте пустым, чтобы не менять пароль</small

>

<div class="error-message" id="editUserPasswordError"></div>

</div>

<div class="form-group">

<label for="editUserPasswordConfirm">Подтверждение пароля</label>

<div class="password-wrapper">

<input type="password" id="editUserPasswordConfirm" />

<span class="toggle-password" data-target="editUserPasswordConfirm"

><i class="fas fa-eye"></i

></span>

</div>

<div class="error-message" id="editUserPasswordConfirmError"></div>

</div>

<div class="form-group">

<label for="editUserRole">Роль</label>

<select id="editUserRole" required>

<option value="user">Пользователь</option>

<option value="manager">Менеджер</option>

<option value="admin">Администратор</option>

</select>

<div class="error-message" id="editUserRoleError"></div>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="updateUserBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно подтверждения удаления -->

<div class="modal" id="deleteUserModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Подтверждение удаления</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<p>Вы уверены, что хотите удалить этого пользователя?</p>

<p class="warning-text">

Внимание: Если у пользователя есть забронированные билеты, удаление

будет невозможно.

</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-danger" id="confirmDeleteBtn">

Удалить

</button>

</div>

</div>

</div>

<script src="/js/users.js"></script>

<script>

// Показывать/скрывать пароль

document.querySelectorAll(".toggle-password").forEach(function (el) {

el.addEventListener("click", function () {

const target = document.getElementById(el.getAttribute("data-target"));

if (target.type === "password") {

target.type = "text";

el.innerHTML = '<i class="fas fa-eye-slash"></i>';

} else {

target.type = "password";

el.innerHTML = '<i class="fas fa-eye"></i>';

}

});

});

</script>

ПРИЛОЖЕНИЕ 54. Файл src\views\auth\login.ejs

<div class="content-container auth-container">

<div class="form-container">

<div class="card">

<div class="card-header">

<h4>Вход в систему</h4>

</div>

<div class="card-body">

<% if (locals.error) { %>

<div class="alert alert-danger"><%= error %></div>

<% } %>

<form action="/auth/login" method="POST">

<div class="form-group">

<label for="email">Email</label>

<input type="email" id="email" name="email" required />

</div>

<div class="form-group">

<label for="password">Пароль</label>

<input type="password" id="password" name="password" required />

</div>

<div class="form-group">

<button type="submit" class="btn btn-primary" style="width: 100%">

Войти

</button>

</div>

</form>

<div style="text-align: center; margin-top: 1rem">

<p>Нет аккаунта? <a href="/auth/register">Зарегистрироваться</a></p>

</div>

</div>

</div>

</div>

</div>

ПРИЛОЖЕНИЕ 55. Файл src\views\auth\register.ejs

<div class="content-container auth-container">

<div class="form-container">

<div class="card">

<div class="card-header">

<h4>Регистрация</h4>

</div>

<div class="card-body">

<% if (locals.error) { %>

<div class="alert alert-danger"><%= error %></div>

<% } %>

<form action="/auth/register" method="POST">

<div class="grid grid-3">

<div class="form-group">

<label for="surname">Фамилия</label>

<input type="text" id="surname" name="surname" required />

</div>

<div class="form-group">

<label for="first\_name">Имя</label>

<input type="text" id="first\_name" name="first\_name" required />

</div>

<div class="form-group">

<label for="patronymic">Отчество</label>

<input type="text" id="patronymic" name="patronymic" />

</div>

</div>

<div class="form-group">

<label for="email">Email</label>

<input type="email" id="email" name="email" required />

</div>

<div class="form-group">

<label for="password">Пароль</label>

<input type="password" id="password" name="password" required />

</div>

<div class="form-group">

<label for="password\_confirm">Подтверждение пароля</label>

<input

type="password"

id="password\_confirm"

name="password\_confirm"

required

/>

</div>

<div class="form-group">

<button type="submit" class="btn btn-primary" style="width: 100%">

Зарегистрироваться

</button>

</div>

</form>

<div style="text-align: center; margin-top: 1rem">

<p>Уже есть аккаунт? <a href="/auth/login">Войти</a></p>

</div>

</div>

</div>

</div>

</div>

ПРИЛОЖЕНИЕ 56. Файл src\views\layouts\main.ejs

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title><%= title %> | Художественная галерея</title>

<link rel="stylesheet" href="/css/style.css" />

<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>

</head>

<body>

<%- include('../partials/header') %> <%- body %> <%-

include('../partials/footer') %>

<script src="/js/main.js"></script>

<script src="/js/admin.js"></script>

</body>

</html>

ПРИЛОЖЕНИЕ 57. Файл src\views\manager\exhibitions.ejs

<div class="admin-container">

<div class="admin-header">

<h1>Управление выставками</h1>

<button

type="button"

class="btn btn-primary"

data-modal="addExhibitionModal"

>

<i class="fas fa-plus"></i>Добавить выставку

</button>

</div>

<!-- Поиск и фильтрация -->

<div class="search-filter-section centered-search">

<!-- Поисковая строка -->

<div class="search-section">

<div class="search-container">

<input

type="text"

id="searchInput"

class="search-input"

placeholder="Поиск выставок..."

aria-label="Поиск выставок"

/>

</div>

</div>

<!-- Фильтры на одной строке -->

<div class="filters-row">

<!-- Фильтрация по статусу -->

<div class="filter-section">

<div class="filter-container">

<div class="filter-group">

<label for="statusFilter">Статус:</label>

<select id="statusFilter" class="filter-input" multiple>

<option value="upcoming">Запланирована</option>

<option value="active">Активная</option>

<option value="completed">Завершена</option>

</select>

</div>

<button

type="button"

class="btn btn-secondary"

id="clearStatusFilter"

>

<i class="fas fa-times"></i>Очистить

</button>

</div>

</div>

<!-- Фильтрация по периоду проведения -->

<div class="filter-section">

<div class="filter-container">

<div class="filter-group">

<label for="dateFrom">Период проведения от:</label>

<input

type="date"

id="dateFrom"

class="filter-input"

aria-label="Дата начала периода"

/>

</div>

<div class="filter-group">

<label for="dateTo">до:</label>

<input

type="date"

id="dateTo"

class="filter-input"

aria-label="Дата окончания периода"

/>

</div>

<button type="button" class="btn btn-secondary" id="clearDateFilter">

<i class="fas fa-times"></i>Очистить

</button>

</div>

</div>

<!-- Фильтрация по цене билета -->

<div class="filter-section">

<div class="filter-container">

<div class="filter-group">

<label for="priceFrom">Цена билета от:</label>

<input

type="number"

id="priceFrom"

class="filter-input"

min="0"

step="0.01"

placeholder="₽"

aria-label="Минимальная цена"

/>

</div>

<div class="filter-group">

<label for="priceTo">до:</label>

<input

type="number"

id="priceTo"

class="filter-input"

min="0"

step="0.01"

placeholder="₽"

aria-label="Максимальная цена"

/>

</div>

<button type="button" class="btn btn-secondary" id="clearPriceFilter">

<i class="fas fa-times"></i>Очистить

</button>

</div>

</div>

</div>

</div>

<!-- Таблица выставок -->

<div class="table-container">

<table class="data-table">

<thead>

<tr>

<th class="sortable" data-column="id">ID</th>

<th>Афиша</th>

<th class="sortable" data-column="title">Название</th>

<th class="sortable" data-column="location">Место проведения</th>

<th class="sortable" data-column="start\_date">Дата начала</th>

<th class="sortable" data-column="end\_date">Дата окончания</th>

<th class="sortable" data-column="status">Статус</th>

<th class="sortable" data-column="ticket\_price">Цена билета</th>

<th>Билеты</th>

<th>Действия</th>

</tr>

</thead>

<tbody id="exhibitionsTableBody">

<!-- Данные будут загружены через JavaScript -->

</tbody>

</table>

</div>

</div>

<!-- Модальное окно с детальной информацией о выставке -->

<div class="modal" id="exhibitionDetailsModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Информация о выставке</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<div class="exhibition-details">

<div class="detail-group">

<h3>Основная информация</h3>

<p><strong>ID:</strong> <span id="detailId"></span></p>

<p><strong>Название:</strong> <span id="detailTitle"></span></p>

<p>

<strong>Место проведения:</strong> <span id="detailLocation"></span>

</p>

<p>

<strong>Дата начала:</strong> <span id="detailStartDate"></span>

</p>

<p>

<strong>Дата окончания:</strong> <span id="detailEndDate"></span>

</p>

<p>

<strong>Часы работы:</strong> <span id="detailWorkingHours"></span>

</p>

<p><strong>Статус:</strong> <span id="detailStatus"></span></p>

<p>

<strong>Цена билета:</strong> <span id="detailTicketPrice"></span>

</p>

<p><strong>Описание:</strong> <span id="detailDescription"></span></p>

</div>

<div class="detail-group">

<h3>Произведения искусства</h3>

<div id="detailArtworks" class="artworks-list"></div>

</div>

<div class="detail-group">

<h3>Статистика билетов</h3>

<div id="detailTickets" class="tickets-info"></div>

</div>

</div>

</div>

</div>

</div>

<!-- Модальное окно добавления выставки -->

<div class="modal" id="addExhibitionModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Добавить выставку</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<form id="addExhibitionForm">

<div class="form-group">

<label for="exhibitionTitle">Название</label>

<input type="text" id="exhibitionTitle" required />

<div class="error-message" id="exhibitionTitleError"></div>

</div>

<div class="form-group">

<label for="exhibitionLocation">Место проведения</label>

<input type="text" id="exhibitionLocation" required />

<div class="error-message" id="exhibitionLocationError"></div>

</div>

<div class="form-group">

<label for="exhibitionStartDate">Дата начала</label>

<input type="date" id="exhibitionStartDate" required />

<div class="error-message" id="exhibitionStartDateError"></div>

</div>

<div class="form-group">

<label for="exhibitionEndDate">Дата окончания</label>

<input type="date" id="exhibitionEndDate" required />

<div class="error-message" id="exhibitionEndDateError"></div>

</div>

<div class="form-group">

<label for="exhibitionOpeningTime">Время открытия</label>

<input

type="time"

id="exhibitionOpeningTime"

required

value="10:00"

/>

<div class="error-message" id="exhibitionOpeningTimeError"></div>

</div>

<div class="form-group">

<label for="exhibitionClosingTime">Время закрытия</label>

<input

type="time"

id="exhibitionClosingTime"

required

value="18:00"

/>

<div class="error-message" id="exhibitionClosingTimeError"></div>

</div>

<div class="form-group">

<label for="exhibitionStatus">Статус</label>

<select id="exhibitionStatus" required>

<option value="upcoming">Запланирована</option>

<option value="active">Идёт</option>

<option value="completed">Завершена</option>

</select>

<div class="error-message" id="exhibitionStatusError"></div>

</div>

<div class="form-group">

<label for="exhibitionTicketPrice">Цена билета</label>

<input

type="number"

id="exhibitionTicketPrice"

min="0"

step="0.01"

required

/>

<div class="error-message" id="exhibitionTicketPriceError"></div>

</div>

<div class="form-group">

<label for="exhibitionTotalTickets">Общее количество билетов</label>

<input type="number" id="exhibitionTotalTickets" min="1" required />

<div class="error-message" id="exhibitionTotalTicketsError"></div>

</div>

<div class="form-group">

<label for="exhibitionDescription">Описание</label>

<textarea id="exhibitionDescription" rows="4"></textarea>

<div class="error-message" id="exhibitionDescriptionError"></div>

</div>

<div class="form-group">

<label for="exhibitionArtworksCheckboxes"

>Произведения искусства</label

>

<div

id="exhibitionArtworksCheckboxes"

class="artworks-checkbox-list"

></div>

<div

class="error-message"

id="exhibitionArtworksCheckboxesError"

></div>

</div>

<div class="form-group">

<label for="exhibitionPoster">Афиша (изображение)</label>

<input

type="file"

id="exhibitionPoster"

name="poster"

accept="image/\*"

/>

<div class="image-preview" id="addPosterPreview"></div>

<div class="error-message" id="exhibitionPosterError"></div>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="saveExhibitionBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно редактирования выставки -->

<div class="modal" id="editExhibitionModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Редактировать выставку</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<form id="editExhibitionForm">

<input type="hidden" id="editExhibitionId" />

<div class="form-group">

<label for="editExhibitionTitle">Название</label>

<input type="text" id="editExhibitionTitle" required />

<div class="error-message" id="editExhibitionTitleError"></div>

</div>

<div class="form-group">

<label for="editExhibitionLocation">Место проведения</label>

<input type="text" id="editExhibitionLocation" required />

<div class="error-message" id="editExhibitionLocationError"></div>

</div>

<div class="form-group">

<label for="editExhibitionStartDate">Дата начала</label>

<input type="date" id="editExhibitionStartDate" required />

<div class="error-message" id="editExhibitionStartDateError"></div>

</div>

<div class="form-group">

<label for="editExhibitionEndDate">Дата окончания</label>

<input type="date" id="editExhibitionEndDate" required />

<div class="error-message" id="editExhibitionEndDateError"></div>

</div>

<div class="form-group">

<label for="editExhibitionOpeningTime">Время открытия</label>

<input

type="time"

id="editExhibitionOpeningTime"

required

value="10:00"

/>

<div class="error-message" id="editExhibitionOpeningTimeError"></div>

</div>

<div class="form-group">

<label for="editExhibitionClosingTime">Время закрытия</label>

<input

type="time"

id="editExhibitionClosingTime"

required

value="18:00"

/>

<div class="error-message" id="editExhibitionClosingTimeError"></div>

</div>

<div class="form-group">

<label for="editExhibitionStatus">Статус</label>

<select id="editExhibitionStatus" required>

<option value="upcoming">Запланирована</option>

<option value="active">Идёт</option>

<option value="completed">Завершена</option>

</select>

<div class="error-message" id="editExhibitionStatusError"></div>

</div>

<div class="form-group">

<label for="editExhibitionTicketPrice">Цена билета</label>

<input

type="number"

id="editExhibitionTicketPrice"

min="0"

step="0.01"

required

/>

<div class="error-message" id="editExhibitionTicketPriceError"></div>

</div>

<div class="form-group">

<label for="editExhibitionTotalTickets"

>Общее количество билетов</label

>

<input

type="number"

id="editExhibitionTotalTickets"

min="1"

required

/>

<div class="error-message" id="editExhibitionTotalTicketsError"></div>

</div>

<div class="form-group">

<label for="editExhibitionDescription">Описание</label>

<textarea id="editExhibitionDescription" rows="4"></textarea>

<div class="error-message" id="editExhibitionDescriptionError"></div>

</div>

<div class="form-group">

<label for="editExhibitionArtworksCheckboxes"

>Произведения искусства</label

>

<div

id="editExhibitionArtworksCheckboxes"

class="artworks-checkbox-list"

></div>

<div

class="error-message"

id="editexhibitionArtworksCheckboxesError"

></div>

</div>

<div class="form-group">

<label for="editExhibitionPoster">Афиша (изображение)</label>

<input type="file" id="editExhibitionPoster" accept="image/\*" />

<div class="image-preview" id="editPosterPreview"></div>

<div class="error-message" id="editExhibitionPosterError"></div>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-primary" id="updateExhibitionBtn">

Сохранить

</button>

</div>

</div>

</div>

<!-- Модальное окно подтверждения удаления -->

<div class="modal" id="deleteExhibitionModal">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Подтверждение удаления</h5>

<button type="button" class="modal-close" aria-label="Close">

&times;

</button>

</div>

<div class="modal-body">

<p>Вы уверены, что хотите удалить эту выставку?</p>

<p class="warning-text">

Внимание: Если на выставку были забронированы билеты, удаление будет

невозможно.

</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary modal-close">

Отмена

</button>

<button type="button" class="btn btn-danger" id="confirmDeleteBtn">

Удалить

</button>

</div>

</div>

</div>

<!-- Модальное окно для просмотра изображения афиши (идентично artworks) -->

<div class="modal" id="imagePreviewModal">

<div class="modal-content">

<div class="modal-body">

<img

id="previewImage"

src=""

alt="Предпросмотр изображения"

style="max-width: 100%; height: auto"

/>

</div>

</div>

</div>

<script src="/js/exhibition.js"></script>

ПРИЛОЖЕНИЕ 58. Файл src\views\manager\statistics.ejs

<div class="stat-container">

<h1 class="text-3xl font-bold mb-6 text-center"><%= title %></h1>

<div class="grid grid-cols-1 md:grid-cols-3 gap-4 mb-6">

<div>

<label for="statStart" class="block font-medium">Дата с</label>

<input type="date" id="statStart" class="filter-input w-full" />

</div>

<div>

<label for="statEnd" class="block font-medium">Дата по</label>

<input type="date" id="statEnd" class="filter-input w-full" />

</div>

<div class="flex items-end">

<button id="fetchStatsBtn" class="btn btn-primary w-full">Показать</button>

</div>

</div>

<div id="statsResult" class="space-y-4 text-center">

<p id="statsLoading" class="text-gray-600" style="display:none;">

Загрузка...

</p>

<div id="statsContent" class="space-y-2 text-center" style="display:none;">

<p><strong>Продано билетов:</strong> <span id="ticketsSold">0</span></p>

<p><strong>Выручка:</strong> <span id="totalRevenue">0</span> ₽</p>

</div>

<p id="statsEmpty" class="text-gray-600 text-center" style="display:none;">

Нет данных за выбранный период.

</p>

</div>

</div>

<script src="/js/statistics.js"></script>

ПРИЛОЖЕНИЕ 59. Файл src\views\partials\footer.ejs

<footer class="site-footer">

<div class="footer-container">

<div class="footer-content">

<div class="footer-section">

<h5>Художественная галерея</h5>

<p>Ваш проводник в мир искусства</p>

</div>

<div class="footer-section">

<h5>Контакты</h5>

<p>Email: info@gallery.ru</p>

<p>Телефон: +7 (999) 123-45-67</p>

</div>

<div class="footer-section">

<h5>Мы в соцсетях</h5>

<div class="social-links">

<a href="#">VK</a>

<a href="#">Telegram</a>

</div>

</div>

</div>

<div class="footer-divider"></div>

<div class="footer-copyright">

<p>

&copy; <%= new Date().getFullYear() %> Художественная галерея. Все права

защищены.

</p>

</div>

</div>

</footer>

ПРИЛОЖЕНИЕ 60. Файл src\views\partials\header.ejs

<header class="site-header">

<div class="header-container">

<a

class="site-logo"

href="<% if (locals.user && user.role === 'admin') { %>/admin<% } else { %>/<% } %>"

>Художественная галерея</a

>

<button class="menu-toggle" type="button" aria-label="Toggle navigation">

<span class="menu-icon"></span>

</button>

<nav class="main-nav">

<ul class="nav-list">

<% if (locals.user && user.role === 'user') { %>

<li class="nav-item">

<a class="nav-link" href="/artworks">Картины</a>

</li>

<li class="nav-item">

<a class="nav-link" href="/tickets">Билеты</a>

</li>

<% } %>

</ul>

<ul class="nav-list">

<% if (locals.user) { %> <% if (user.role === 'manager') { %>

<li class="nav-item">

<a class="nav-link" href="/artworks">Картины</a>

</li>

<li class="nav-item">

<a class="nav-link" href="/manager/statistics">Статистика</a>

</li>

<li class="nav-item">

<a class="nav-link" href="/manager/exhibitions">Панель менеджера</a>

</li>

<% } %>

<li class="nav-item">

<a class="nav-link" href="/profile">Профиль</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#" data-action="logout">Выйти</a>

</li>

<% } else { %>

<li class="nav-item">

<a class="nav-link" href="/auth/login">Войти</a>

</li>

<li class="nav-item">

<a class="nav-link" href="/auth/register">Регистрация</a>

</li>

<% } %>

</ul>

</nav>

</div>

</header>

ПРИЛОЖЕНИЕ 61. Файл src\app.js

// app.js

const path = require('path');

const cookieParser = require('cookie-parser');

// Загружаем переменные окружения

if (process.env.NODE\_ENV === 'test') {

require('dotenv').config({ path: path.join(\_\_dirname, '../.env.test') });

} else {

require('dotenv').config();

}

const express = require('express');

const cors = require('cors');

const expressLayouts = require('express-ejs-layouts');

const db = require('./models');

const setUser = require('./middleware/setUser');

const authController = require('./controllers/authController');

const app = express();

// Middleware

app.use(cors({

origin: process.env.CORS\_ORIGIN || 'http://localhost:3000',

credentials: true

}));

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

app.use(express.static(path.join(\_\_dirname, 'public')));

app.use(cookieParser());

// Подключение к базе данных

db.sequelize.authenticate()

.then(() => console.log('Database connected...'))

.catch(err => {

console.error('Unable to connect to the database:', err);

process.exit(1);

});

// View engine setup

app.set('view engine', 'ejs');

app.set('views', path.join(\_\_dirname, 'views'));

app.use(expressLayouts);

app.set('layout', 'layouts/main');

// Устанавливаем пользователя в res.locals

app.use(setUser);

// ===== API routes =====

const authRoutes = require('./routes/authRoutes');

const userRoutes = require('./routes/userRoutes');

const artworkRoutes = require('./routes/artworkRoutes');

const exhibitionApi = require('./routes/exhibitionRoutes');

const ticketRoutes = require('./routes/ticketRoutes');

const styleRoutes = require('./routes/styleRoutes');

const genreRoutes = require('./routes/genreRoutes');

const authorRoutes = require('./routes/authorRoutes');

app.use('/api/v1/auth', authRoutes);

app.use('/api/v1/users', userRoutes);

app.use('/api/v1/artworks', artworkRoutes);

app.use('/api/v1/exhibitions', exhibitionApi);

app.use('/api/v1/tickets', ticketRoutes);

app.use('/api/v1/styles', styleRoutes);

app.use('/api/v1/genres', genreRoutes);

app.use('/api/v1/authors', authorRoutes);

// ===== Frontend routes for exhibitions =====

const exhibitionsPageRoutes = require('./routes/exhibitionRoutes');

app.use('/exhibitions', exhibitionsPageRoutes);

// Подключение основного роутера для прочих публичных страниц

const mainRoutes = require('./routes/index');

app.use(mainRoutes);

// ===== Остальные ваши фронтенд-маршруты =====

app.get('/', (req, res) => {

res.render('index', {

title: 'Главная страница',

user: res.locals.user || null

});

});

app.get('/auth/login', (req, res) => {

if (res.locals.user) return res.redirect('/admin');

res.render('auth/login', { title: 'Вход', user: null });

});

app.post('/auth/login', authController.login);

app.get('/auth/register', (req, res) => {

if (res.locals.user) return res.redirect('/');

res.render('auth/register', { title: 'Регистрация', user: null });

});

app.post('/auth/register', authController.register);

app.get('/auth/logout', (req, res) => {

res.clearCookie('token');

res.clearCookie('connect.sid');

res.redirect('/auth/login');

});

app.get('/admin', (req, res) => {

if (!res.locals.user || res.locals.user.role !== 'admin') {

return res.status(403).render('error', {

title: 'Доступ запрещён',

message: 'Требуются права администратора',

error: { status: 403 },

user: res.locals.user

});

}

res.render('admin/index', { title: 'Панель администратора', user: res.locals.user });

});

app.get('/admin/genres', (req, res) => {

if (!res.locals.user || res.locals.user.role !== 'admin') {

return res.status(403).render('error', {

title: 'Доступ запрещён',

message: 'Требуются права администратора',

error: { status: 403 },

user: res.locals.user

});

}

res.render('admin/genres', { title: 'Управление жанрами', user: res.locals.user });

});

app.get('/admin/styles', (req, res) => {

if (!res.locals.user || res.locals.user.role !== 'admin') {

return res.status(403).render('error', {

title: 'Доступ запрещён',

message: 'Требуются права администратора',

error: { status: 403 },

user: res.locals.user

});

}

res.render('admin/styles', { title: 'Управление стилями', user: res.locals.user });

});

app.get('/admin/authors', (req, res) => {

if (!res.locals.user || res.locals.user.role !== 'admin') {

return res.status(403).render('error', {

title: 'Доступ запрещён',

message: 'Требуются права администратора',

error: { status: 403 },

user: res.locals.user

});

}

res.render('admin/authors', { title: 'Управление авторами', user: res.locals.user });

});

app.get('/admin/users', (req, res) => {

if (!res.locals.user || res.locals.user.role !== 'admin') {

return res.status(403).render('error', {

title: 'Доступ запрещён',

message: 'Требуются права администратора',

error: { status: 403 },

user: res.locals.user

});

}

res.render('admin/users', { title: 'Управление пользователями', user: res.locals.user });

});

app.get('/admin/artworks', (req, res) => {

if (!res.locals.user || res.locals.user.role !== 'admin') {

return res.status(403).render('error', {

title: 'Доступ запрещён',

message: 'Требуются права администратора',

error: { status: 403 },

user: res.locals.user

});

}

res.render('admin/artworks', { title: 'Управление произведениями искусства', user: res.locals.user });

});

app.get('/manager/exhibitions', (req, res) => {

if (!res.locals.user || (res.locals.user.role !== 'admin' && res.locals.user.role !== 'manager')) {

return res.status(403).render('error', {

title: 'Доступ запрещён',

message: 'Требуются права администратора/менеджера',

error: { status: 403 },

user: res.locals.user

});

}

res.render('manager/exhibitions', { title: 'Управление выставками', user: res.locals.user });

});

app.get('/admin/tickets', (req, res) => {

if (!res.locals.user || res.locals.user.role !== 'admin') {

return res.status(403).render('error', {

title: 'Доступ запрещён',

message: 'Требуются права администратора',

error: { status: 403 },

user: res.locals.user

});

}

res.render('admin/tickets', { title: 'Управление билетами', user: res.locals.user });

});

app.get('/profile', (req, res) => {

if (!res.locals.user) return res.redirect('/auth/login');

res.render('profile', { title: 'Профиль', user: res.locals.user });

});

// Обработка 404 ошибок

app.use((req, res) => {

res.status(404).render('error', {

title: 'Страница не найдена',

message: 'Запрашиваемая страница не существует',

error: { status: 404 },

user: res.locals.user || null

});

});

// Error handling middleware

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(err.status || 500).render('error', {

title: 'Ошибка',

message: err.message || 'Что-то пошло не так',

error: process.env.NODE\_ENV === 'development' ? err : {},

user: res.locals.user || null

});

});

module.exports = app;

// Start server

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

console.log(`Environment: ${process.env.NODE\_ENV}`);

});

ПРИЛОЖЕНИЕ 62. Файл src\views\index.ejs

<!-- Main content -->

<main class="main-content">

<div class="content-container">

<h1 class="page-title">Актуальные выставки</h1>

<form

id="exhibition-filter-form"

class="search-filter-section centered-search"

autocomplete="off"

>

<!-- Поисковая строка -->

<div class="search-section">

<div class="search-container">

<input

type="text"

name="search"

id="search"

class="search-input"

placeholder="Поиск по названию или месту проведения..."

/>

</div>

</div>

<!-- Фильтры на одной строке -->

<div class="filters-row">

<!-- Фильтр по датам -->

<div class="filter-section">

<div class="filter-container">

<div class="filter-group">

<label for="start\_date">Дата с</label>

<input

type="date"

name="start\_date"

id="start\_date"

class="filter-input"

/>

</div>

<div class="filter-group">

<label for="end\_date">Дата по</label>

<input

type="date"

name="end\_date"

id="end\_date"

class="filter-input"

/>

</div>

<button

type="button"

class="btn btn-secondary"

id="clearDateFilter"

>

<i class="fas fa-times"></i>Очистить

</button>

</div>

</div>

<!-- Фильтр по цене -->

<div class="filter-section">

<div class="filter-container">

<div class="filter-group">

<label for="min\_price">Цена от</label>

<input

type="number"

name="min\_price"

id="min\_price"

class="filter-input"

min="0"

step="1"

placeholder="0"

/>

</div>

<div class="filter-group">

<label for="max\_price">Цена до</label>

<input

type="number"

name="max\_price"

id="max\_price"

class="filter-input"

min="0"

step="1"

placeholder="∞"

/>

</div>

<button

type="button"

class="btn btn-secondary"

id="clearPriceFilter"

>

<i class="fas fa-times"></i>Очистить

</button>

</div>

</div>

</div>

</form>

<div id="exhibitions-list" class="grid grid-3 exhibitions-list"></div>

<div id="no-exhibitions-message" class="no-exhibitions-message">

Нет выставок по выбранным критериям.

</div>

</div>

</main>

<div id="image-lightbox" class="image-lightbox">

<img

id="lightbox-img"

src=""

alt="Увеличенное изображение"

class="lightbox-img"

/>

</div>

<script src="/js/main.js"></script>