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](#_Toc200723444)

[1. СТАДИЯ «ТЕХНИЧЕСКОЕ ЗАДАНИЕ» 6](#_Toc200723445)

[1.1. Общая постановка задачи 6](#_Toc200723446)

[1.2. Разработка терминологического словаря предметной области 7](#_Toc200723447)

[1.3. Общие требования к проектируемой системе 7](#_Toc200723448)

[1.4. Категории пользователей проектируемой системы 8](#_Toc200723449)

[2. СТАДИЯ «ЭСКИЗНЫЙ ПРОЕКТ» 11](#_Toc200723450)

[2.1. Разработка программной архитектуры системы 11](#_Toc200723451)

[2.2. Разработка диаграмм вариантов использования компонентов 14](#_Toc200723452)

[3. СТАДИЯ «ТЕХНИЧЕСКИЙ ПРОЕКТ» 18](#_Toc200723453)

[3.1. Разработка диаграмм концептуальных классов 18](#_Toc200723454)

[3.2. Разработка диаграмм программных классов 20](#_Toc200723455)

[3.3. Разработка диаграмм состояний 21](#_Toc200723456)

[4. МАКЕТЫ HTML-СТРАНИЦ РАЗРАБАТЫВАЕМОЙ СИСТЕМЫ 29](#_Toc200723457)

[4.1. Макет страницы авторизации 29](#_Toc200723458)

[4.2. Макет страницы регистрации 29](#_Toc200723459)

[4.3. Макет главной страницы 30](#_Toc200723460)

[4.4. Макет системы администратора 30](#_Toc200723461)

[4.5. Макет страницы статистики 30](#_Toc200723462)

[5. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЕЙ СИСТЕМЫ 31](#_Toc200723463)

[ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ 37](#_Toc200723464)

[ПРИЛОЖЕНИЕ 1. Файл server/app.js 38](#_Toc200723465)

[ПРИЛОЖЕНИЕ 2. Файл server/db.js 38](#_Toc200723466)

[ПРИЛОЖЕНИЕ 3. Файл server/config/app.js 38](#_Toc200723467)

[ПРИЛОЖЕНИЕ 3. Файл server/config/database.js 38](#_Toc200723468)

[ПРИЛОЖЕНИЕ 4. Файл server/controllers/AuthController.js 38](#_Toc200723469)

[ПРИЛОЖЕНИЕ 5. Файл server/controllers/BookingController.js 39](#_Toc200723470)

[ПРИЛОЖЕНИЕ 6. Файл server/controllers/CarController.js 40](#_Toc200723471)

[ПРИЛОЖЕНИЕ 7. Файл server/controllers/LogController.js 40](#_Toc200723472)

[ПРИЛОЖЕНИЕ 8. Файл server/controllers/ParkingSlotController.js 40](#_Toc200723473)

[ПРИЛОЖЕНИЕ 9. Файл server/controllers/TariffController.js 41](#_Toc200723474)

[ПРИЛОЖЕНИЕ 10. Файл server/controllers/UserController.js 42](#_Toc200723475)

[ПРИЛОЖЕНИЕ 11. Файл server/controllers/ZoneController.js 42](#_Toc200723476)

[ПРИЛОЖЕНИЕ 12. Файл server/middlewares /AuthMiddleware.js 43](#_Toc200723477)

[ПРИЛОЖЕНИЕ 13. Файл server/middlewares /PageAuth.js 43](#_Toc200723478)

[ПРИЛОЖЕНИЕ 14. Файл server/models /Init.js 43](#_Toc200723479)

[ПРИЛОЖЕНИЕ 15. Файл server/models /Models.js 43](#_Toc200723480)

[ПРИЛОЖЕНИЕ 16. Файл server/routes/Auth.js 45](#_Toc200723481)

[ПРИЛОЖЕНИЕ 17. Файл server/routes/Booking.js 45](#_Toc200723482)

[ПРИЛОЖЕНИЕ 18. Файл server/routes/Cars.js 45](#_Toc200723483)

[ПРИЛОЖЕНИЕ 19. Файл server/routes/Logs.js 45](#_Toc200723484)

[ПРИЛОЖЕНИЕ 20. Файл server/routes/Slots.js 45](#_Toc200723485)

[ПРИЛОЖЕНИЕ 21. Файл server/routes/Tariffs.js 46](#_Toc200723486)

[ПРИЛОЖЕНИЕ 22. Файл server/routes/Users.js 46](#_Toc200723487)

[ПРИЛОЖЕНИЕ 23. Файл server/routes/Zones.js 46](#_Toc200723488)

[ПРИЛОЖЕНИЕ 24. Файл server/services/AuthService.js 46](#_Toc200723489)

[ПРИЛОЖЕНИЕ 25. Файл server/services/BookingService.js 47](#_Toc200723490)

[ПРИЛОЖЕНИЕ 26. Файл server/services/CarService.js 48](#_Toc200723491)

[ПРИЛОЖЕНИЕ 27. Файл server/services/LogService.js 49](#_Toc200723492)

[ПРИЛОЖЕНИЕ 28. Файл server/services/ParkingSlotService.js 49](#_Toc200723493)

[ПРИЛОЖЕНИЕ 29. Файл server/services/TariffService.js 49](#_Toc200723494)

[ПРИЛОЖЕНИЕ 30. Файл server/services/UserService.js 50](#_Toc200723495)

[ПРИЛОЖЕНИЕ 31. Файл server/services/ZoneService.js 50](#_Toc200723496)

[ПРИЛОЖЕНИЕ 32. Файл test/auth.test.js 50](#_Toc200723497)

[ПРИЛОЖЕНИЕ 33. Файл test/booking.test.js 51](#_Toc200723498)

[ПРИЛОЖЕНИЕ 34. Файл test/cars.test.js 52](#_Toc200723499)

[ПРИЛОЖЕНИЕ 35. Файл test/slots.test.js 53](#_Toc200723500)

[ПРИЛОЖЕНИЕ 36. Файл test/tariff.test.js 54](#_Toc200723501)

[ПРИЛОЖЕНИЕ 37. Файл test/user.test.js 54](#_Toc200723502)

[ПРИЛОЖЕНИЕ 38. Файл test/user.test.js 55](#_Toc200723503)

[ПРИЛОЖЕНИЕ 39. Файл client/public/css/main.css 55](#_Toc200723504)

[ПРИЛОЖЕНИЕ 40. Файл client/public/js/api.js 55](#_Toc200723505)

[ПРИЛОЖЕНИЕ 41. Файл client/public/js/auth.js 55](#_Toc200723506)

[ПРИЛОЖЕНИЕ 42. Файл client/public/js/bookings.js 56](#_Toc200723507)

[ПРИЛОЖЕНИЕ 43. Файл client/public/js/cars.js 57](#_Toc200723508)

[ПРИЛОЖЕНИЕ 44. Файл client/public/js/dashboard.js 58](#_Toc200723509)

[ПРИЛОЖЕНИЕ 45. Файл client/public/js/slots.js 59](#_Toc200723510)

[ПРИЛОЖЕНИЕ 46. Файл client/public/js/tariffs.js 60](#_Toc200723511)

[ПРИЛОЖЕНИЕ 47. Файл client/public/js/users.js 61](#_Toc200723512)

[ПРИЛОЖЕНИЕ 48. Файл client/public/js/zones.js 62](#_Toc200723513)

[ПРИЛОЖЕНИЕ 49. Файл client/public/views /bookings.pug 62](#_Toc200723514)

[ПРИЛОЖЕНИЕ 50. Файл client/public/views /cars.pug 63](#_Toc200723515)

[ПРИЛОЖЕНИЕ 51. Файл client/public/views /dashboard.pug 63](#_Toc200723516)

[ПРИЛОЖЕНИЕ 52. Файл client/public/views /layout.pug 64](#_Toc200723517)

[ПРИЛОЖЕНИЕ 53. Файл client/public/views /login.pug 64](#_Toc200723518)

[ПРИЛОЖЕНИЕ 54. Файл client/public/views /register.pug 64](#_Toc200723519)

[ПРИЛОЖЕНИЕ 55. Файл client/public/views /slots.pug 65](#_Toc200723520)

[ПРИЛОЖЕНИЕ 56. Файл client/public/views /tariffs.pug 65](#_Toc200723521)

[ПРИЛОЖЕНИЕ 57. Файл client/public/views /users.pug 66](#_Toc200723522)

[ПРИЛОЖЕНИЕ 58. Файл client/public/views /zones.pug 66](#_Toc200723523)

[ПРИЛОЖЕНИЕ 59. Файл server/services/StatisticsService.js 67](#_Toc200723524)

[ПРИЛОЖЕНИЕ 60. Файл server/routes/Statistics.js 68](#_Toc200723525)

[ПРИЛОЖЕНИЕ 61. Файл client/public/views/statistics.pug 68](#_Toc200723526)

[ПРИЛОЖЕНИЕ 62. Файл client/public/js /statistics.js 69](#_Toc200723527)

# 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).

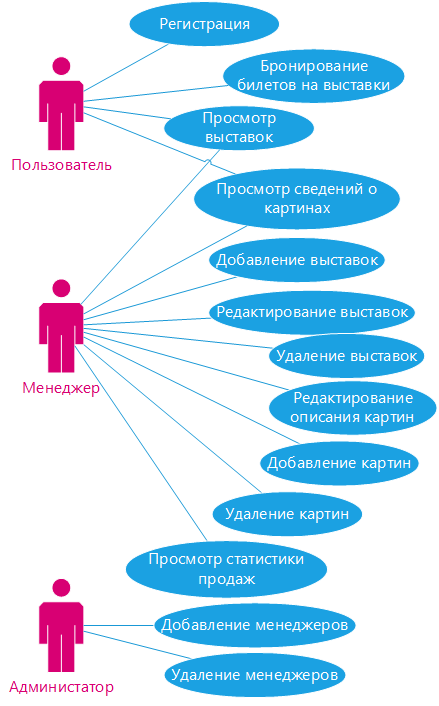


Рисунок 1. Обобщенная UML диаграмма вариантов использования.

**Администратор** отвечает за добавление и удаление менеджеров, управление выставками и картинами, а также редактирование описаний картин.

**Менеджер** выполняет функции планирования и организации выставок, управления экспонатами (добавление, удаление и редактирование описаний картин). Получает доступ к локальным модулям системы с правами формирования и редактирования выставок, а также работы с базой данных экспонатов. 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.

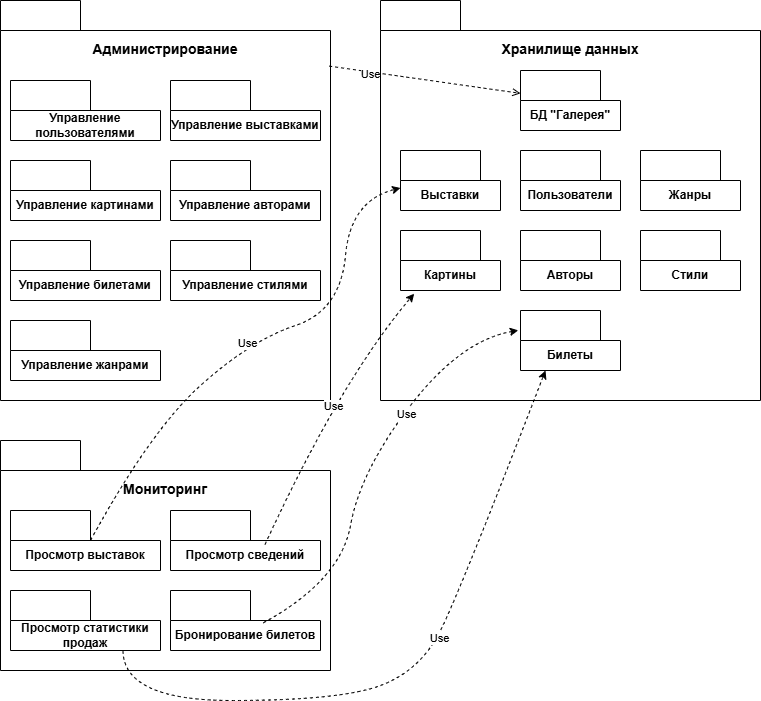


Рисунок 2­. UML диаграмма пакетов

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»*.

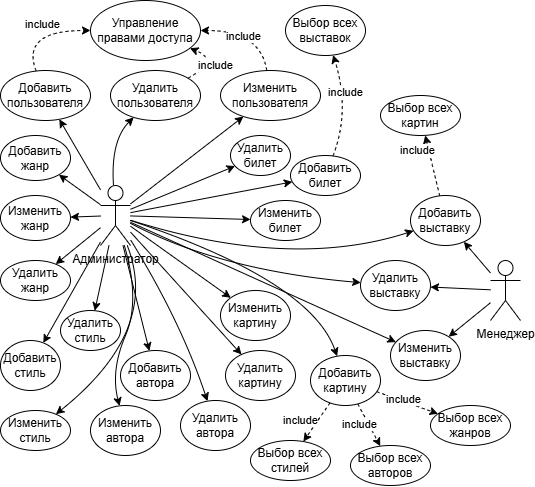


Рисунок 3. UseCase диаграмма подсистемы «Администрирование»



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.

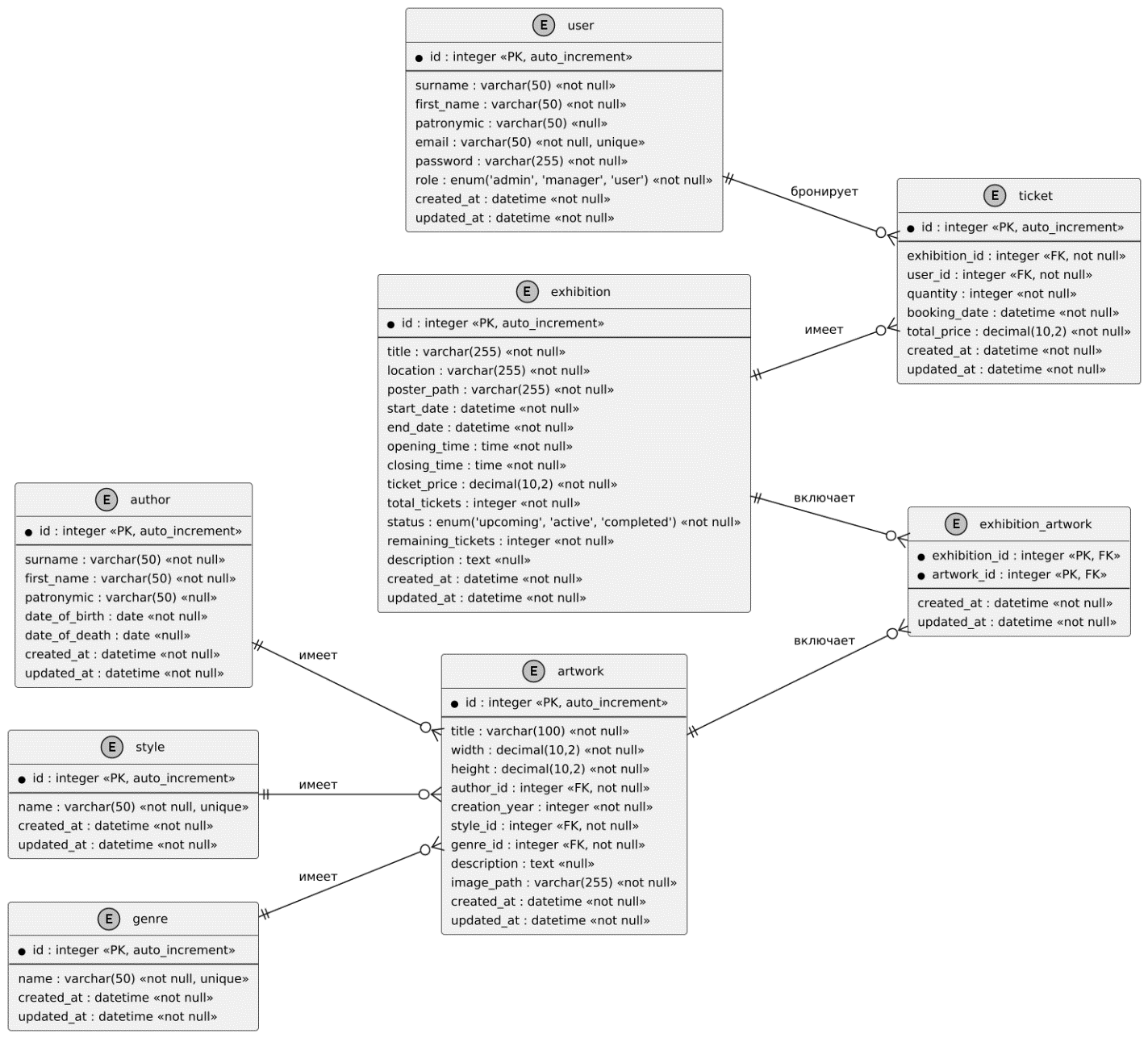


Рисунок 5. Диаграмма концептуальных классов подсистемы хранения данных

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.

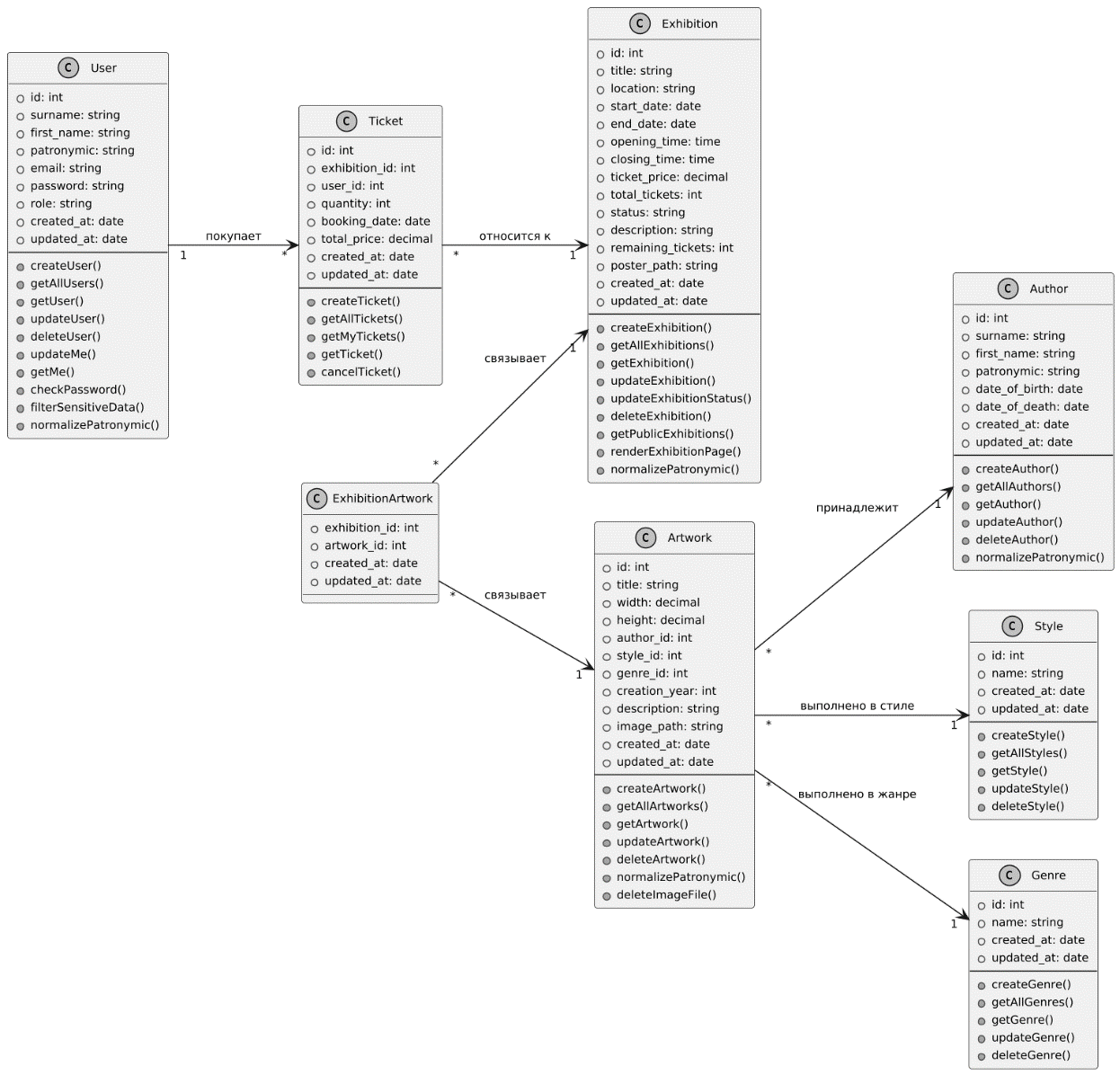


Рисунок 6. Диаграмма классов системы

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.

## 3.3. Разработка диаграмм состояний

Рассмотренные выше диаграммы классов представляют статические модели компонентов проектируемой системы и отражают результаты её объектной декомпозиции. Перейдем к реализации динамических моделей логического уровня. Так как система ориентирована на три различные группы пользователей (каждая из групп имеет свои возможности), рассмотрим каждую из них отдельно.

Первое из целевых состояний (рис. 7) «Authentication» завершается переходом альтернативным переходом в случае, если аутентификация пользователя завершена успешно, он получает доступ к системе переходит в следующее целевое состояние «Main\_Menu», в противном случае — сразу переходит в конечное состояние.

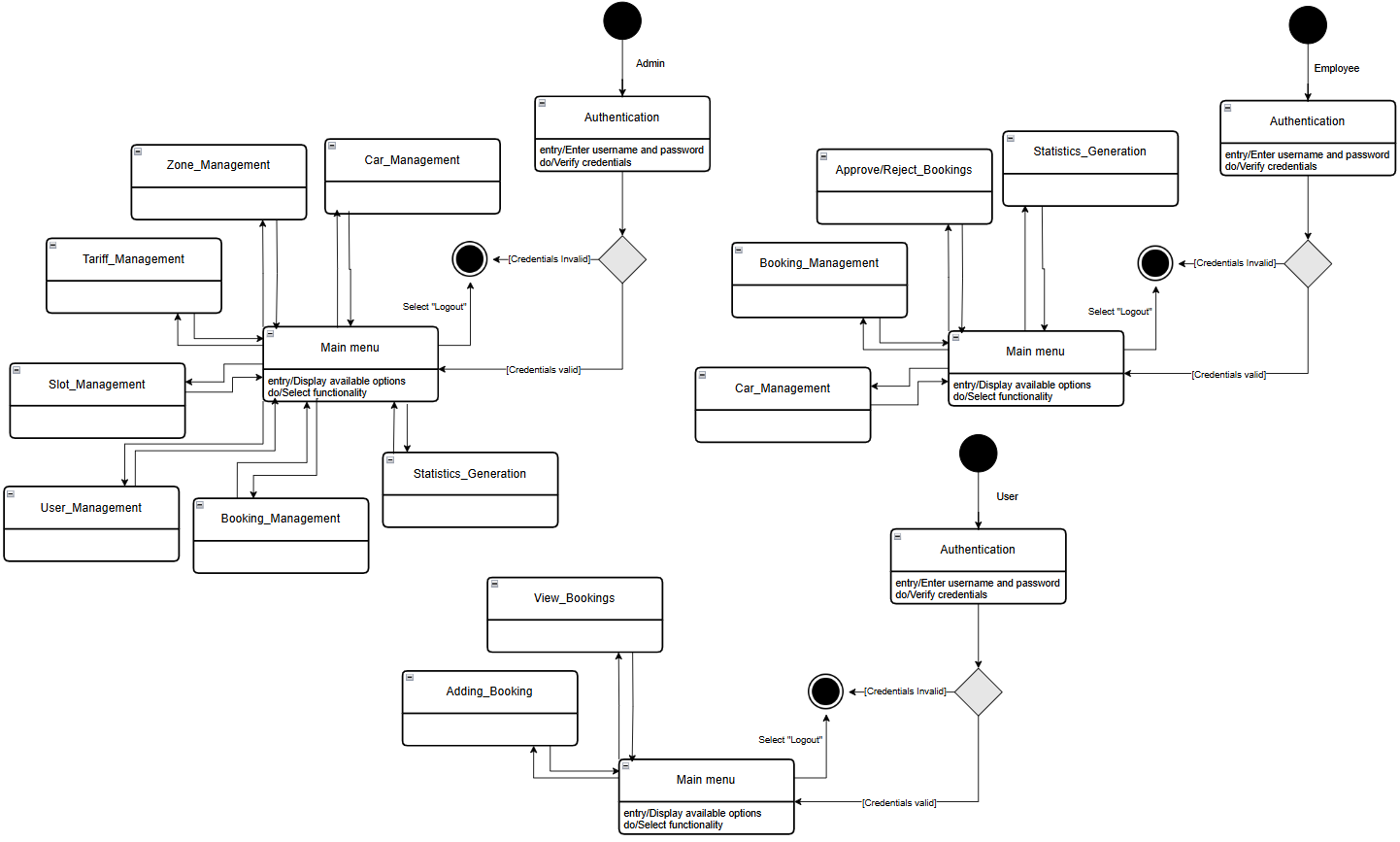


Рисунок 7. Обобщенная диаграмма состояний

В состоянии **«Main\_Menu»** выполняется входное действие entry / Display available options, при котором система отображает перечень доступных функций в зависимости от роли пользователя. Последующее действие do / Select functionality предполагает выбор пользователем нужного режима работы, после чего происходит переход к соответствующему состоянию.

Доступный функционал в **«Main\_Menu»** различается по ролям:

* **Admin** может перейти к: Zone\_Management, Tariff\_Management, Slot\_Management, User\_Management, Booking\_Management, Statistics\_Generation.
* **Employee** — к: Booking\_Management, Approve/Reject\_Bookings, Car\_Management.
* **User** — к: View\_Bookings, Adding\_Booking.

Состояние **«Statistics\_Generation»** доступно сотрудникам и администраторам. Рассмотрим каждое из состояний входящих в это состояние (рис. 8). Первое целевое состояние «Parameter\_Selection» представляет этап, на котором пользователь задаёт параметры для генерации статистического отчёта. Входное действие (entry): Система выполняет загрузку доступных фильтров, таких как временной диапазон, категории транзакций или типы операций, которые пользователь может выбрать для формирования отчёта. Текущее действие (do): Пользователь задаёт фильтры для отчёта, выбирая необходимые параметры из предложенных системой.

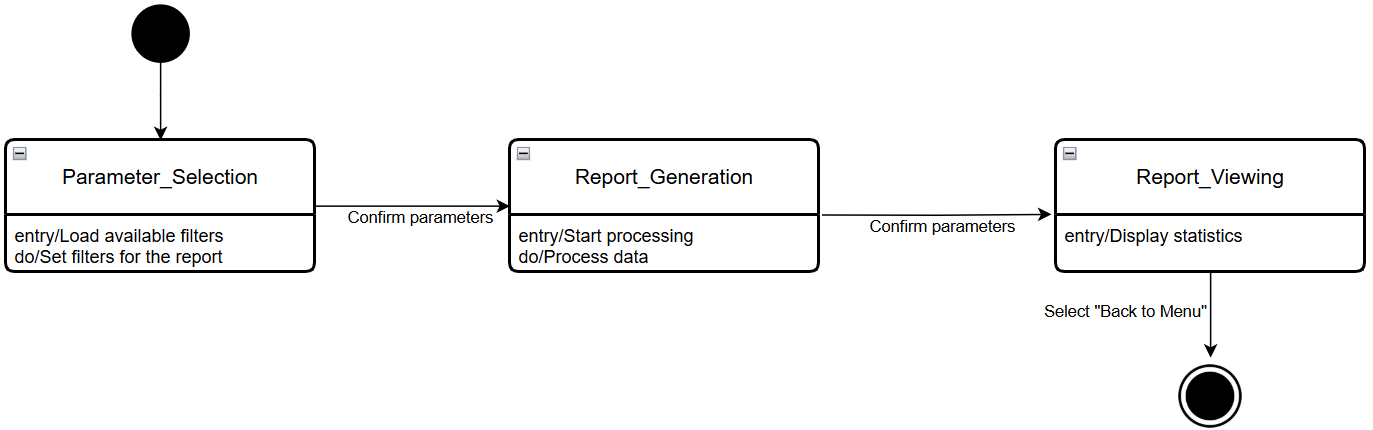


Рисунок 8. Диаграмма состояний блока «Statistics\_Generation»

После подтверждения параметров выполняется переход в состояние **«Report\_Generation»**.

* **Входное действие (entry):** инициируется запуск процесса генерации отчёта.
* **Текущее действие (do):** система обрабатывает введённые параметры, выполняет необходимые запросы к базе данных, агрегирует полученные данные и формирует отчёт.

После завершения обработки происходит переход в состояние **«Report\_Viewing»**.

* **Входное действие (entry):** пользователю отображается готовый статистический отчёт — в виде таблиц, карточек и других визуальных элементов.

На этом этапе пользователь может ознакомиться с результатами и при необходимости вернуться в главное меню.

Состояние **«Booking\_Management»** (рис. 9) доступно администраторам и сотрудникам, однако отдельные под-состояния, такие как **«Viewing\_Bookings»** и **«Adding\_Booking»**, доступны всем ролям, включая обычных пользователей. Оно используется для управления бронированиями парковочных мест: их просмотром, созданием, подтверждением, отклонением и удалением.

Первое под-состояние — **«Viewing\_Bookings»** — представляет этап, на котором пользователь просматривает список всех доступных ему бронирований. При входе в это состояние система загружает данные о бронированиях из базы данных. Текущее действие заключается в отображении информации в виде таблицы с возможностью фильтрации и выбора дальнейших операций: добавления, обработки статуса или удаления бронирования. Доступные действия зависят от роли пользователя.

При выборе действия **«Add booking»** происходит переход в состояние **«Adding\_Booking»**. Входное действие включает отображение формы создания бронирования. Текущее действие заключается в заполнении данных: парковочная зона, слот, транспортное средство, дата и время начала и окончания брони. После этого система переходит в состояние **«Validating\_Booking\_Data»**, где выполняется проверка корректности введённых сведений. Если данные проходят валидацию, выполняется переход в состояние **«Saving\_Booking»**, где бронирование сохраняется в базу данных, после чего система возвращается к просмотру. В случае ошибки выполняется возврат к форме добавления.

Если пользователь с ролью администратора или сотрудника выбирает действие **«Process booking status»**, система переходит в состояние **«Processing\_Booking\_Status»**. В этом состоянии загружаются данные выбранного бронирования, после чего пользователь подтверждает или отклоняет заявку. Затем происходит переход в состояние **«Validating\_Booking\_Data»**, где данные проходят проверку. При положительном результате система выполняет обновление записи в состоянии **«Updating\_Booking»**, затем возвращается к просмотру.

При выборе действия **«Delete booking»** осуществляется переход в состояние **«Deleting\_Booking»**, где выполняется подтверждение удаления. После подтверждения система удаляет бронирование из базы данных и возвращается в состояние **«Viewing\_Bookings»**.

Завершение работы с модулем осуществляется через действие **«Return to menu»**, которое переводит систему в конечное состояние.

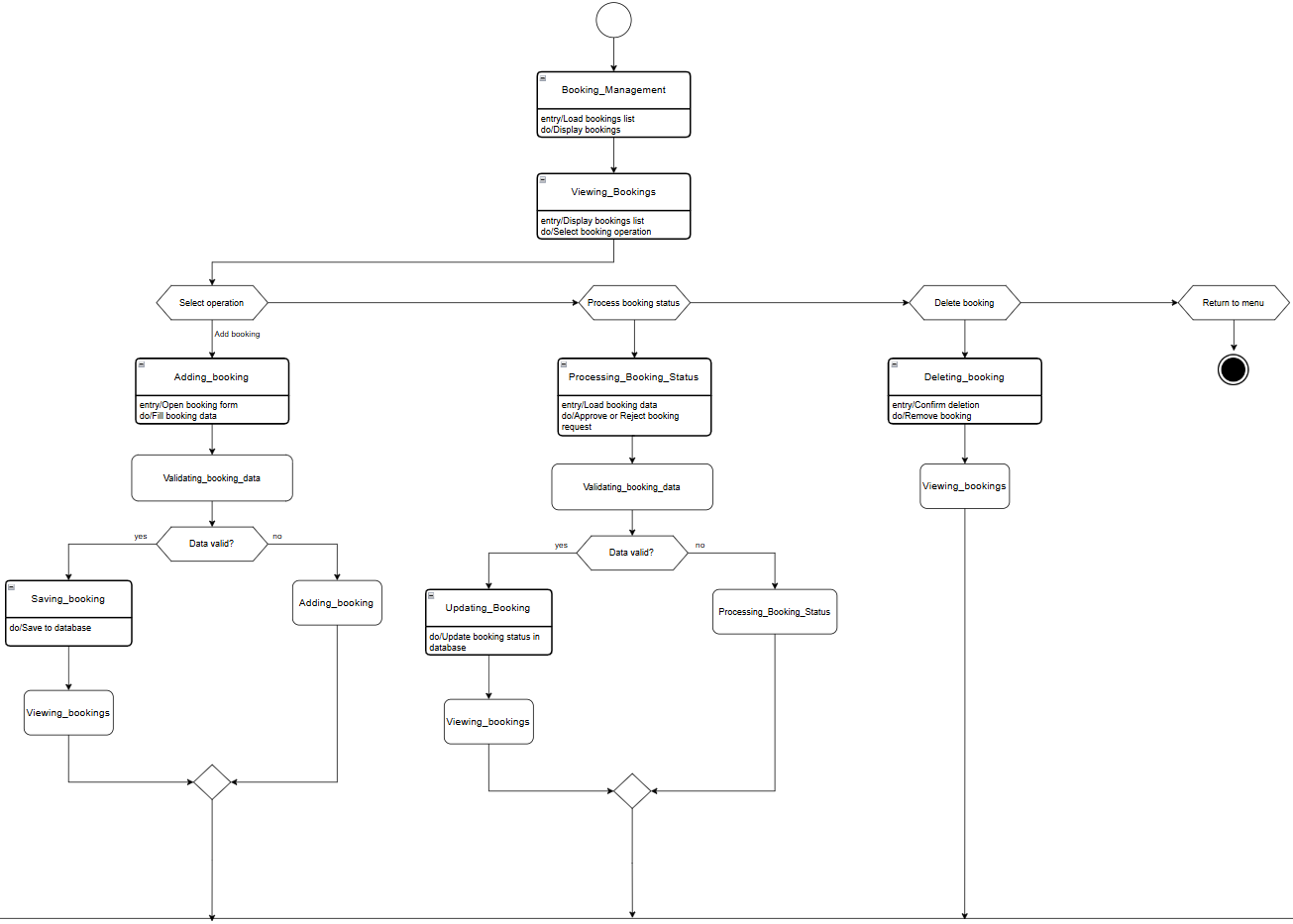


Рисунок 9. Диаграмма состояний блока «Booking\_Management»

Состояние **«User\_Management»** (рис. 10) доступно только администраторам и используется для управления учётными записями пользователей в системе. Оно включает в себя просмотр, добавление, редактирование и удаление пользователей.

Первое под-состояние — **«Viewing\_Users»** — представляет этап, на котором администратор просматривает список всех зарегистрированных пользователей. При входе в это состояние система загружает данные пользователей из базы и отображает их в виде таблицы. Текущее действие — выбор операции над пользователем: добавление, редактирование или удаление.

При выборе действия **«Add user»** осуществляется переход в состояние **«Adding\_User»**. Входное действие включает открытие формы регистрации нового пользователя, а текущее действие — заполнение данных (имя, логин, роль, пароль и т. д.). Далее происходит переход в состояние **«Validating\_User\_Data»**, где система проверяет корректность введённой информации. Если данные валидны, система переходит в состояние **«Saving\_User»**, где выполняется сохранение новой учётной записи в базу данных, после чего происходит возврат в состояние просмотра. В случае ошибки система возвращается к форме добавления.

Аналогично, при выборе действия **«Edit user»** осуществляется переход в состояние **«Editing\_User»**, где происходит загрузка и редактирование данных выбранного пользователя. После внесения изменений выполняется валидация данных. Если проверка успешна, происходит переход в состояние **«Updating\_User»**, где обновлённые данные сохраняются в базе. При ошибке — возврат в режим редактирования.

При выборе действия **«Delete user»** осуществляется переход в состояние **«Deleting\_User»**, где отображается запрос на подтверждение удаления. После подтверждения система удаляет пользователя из базы и возвращается в состояние **Viewing\_Users**.

Завершение работы осуществляется через действие **«Return to menu»**, которое переводит систему в конечное состояние.

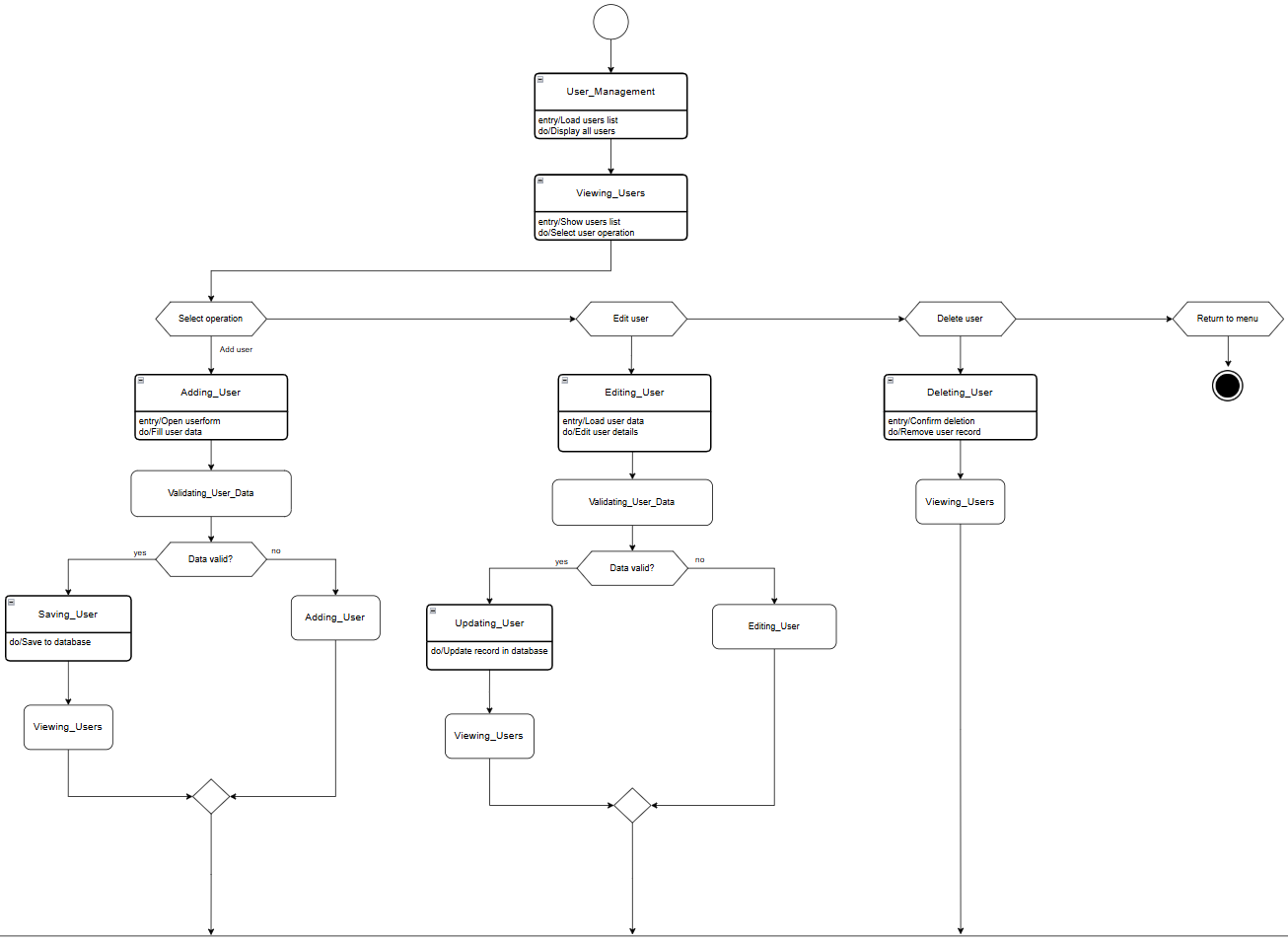


Рисунок 10. Диаграмма состояний блока «Users\_Management»

Состояние **«Car\_Management»** доступно администраторам и сотрудникам и используется для управления данными об автомобилях в системе. Оно включает в себя просмотр, добавление, редактирование, удаление, а также освобождение автомобилей путём установки времени их выезда.

Первое под-состояние — **«Viewing\_Cars»** — представляет этап, на котором пользователь просматривает список всех зарегистрированных автомобилей. При входе в это состояние система загружает данные об автомобилях из базы данных и отображает их в виде таблицы. Текущее действие предполагает выбор одной из доступных операций: добавление, редактирование, удаление или освобождение автомобиля.

При выборе действия **«Add car»** осуществляется переход в состояние **«Adding\_Car»**. Входное действие включает открытие формы добавления автомобиля, а текущее действие — ввод информации о транспортном средстве (госномер, модель, владелец и т.д.). После ввода данных система переходит в состояние **«Validating\_Car\_Data»**, где проверяет корректность введённой информации. Если данные валидны, выполняется переход в состояние **«Saving\_Car»**, где создаётся запись в базе данных, после чего система возвращается в состояние просмотра. В случае ошибки выполняется возврат в форму добавления.

Аналогично, при выборе действия **«Edit car»** выполняется переход в состояние **«Editing\_Car»**, где происходит загрузка и редактирование данных выбранного автомобиля. Далее выполняется валидация. При успешной проверке осуществляется переход в состояние **«Updating\_Car»**, где информация обновляется в базе. В случае ошибки происходит возврат к редактированию.

Выбор действия **«Delete car»** приводит к переходу в состояние **«Deleting\_Car»**, где отображается подтверждение удаления. После подтверждения система удаляет автомобиль из базы и возвращается в состояние **Viewing\_Cars**.

Отдельное действие **«Release car»** реализуется через переход в состояние **«Releasing\_Car»**. В нём система загружает данные об автомобиле и устанавливает время выезда. После выполнения действия система возвращается к просмотру автомобилей.

Завершение работы осуществляется через действие **«Return to menu»**, которое переводит систему в конечное состояние.

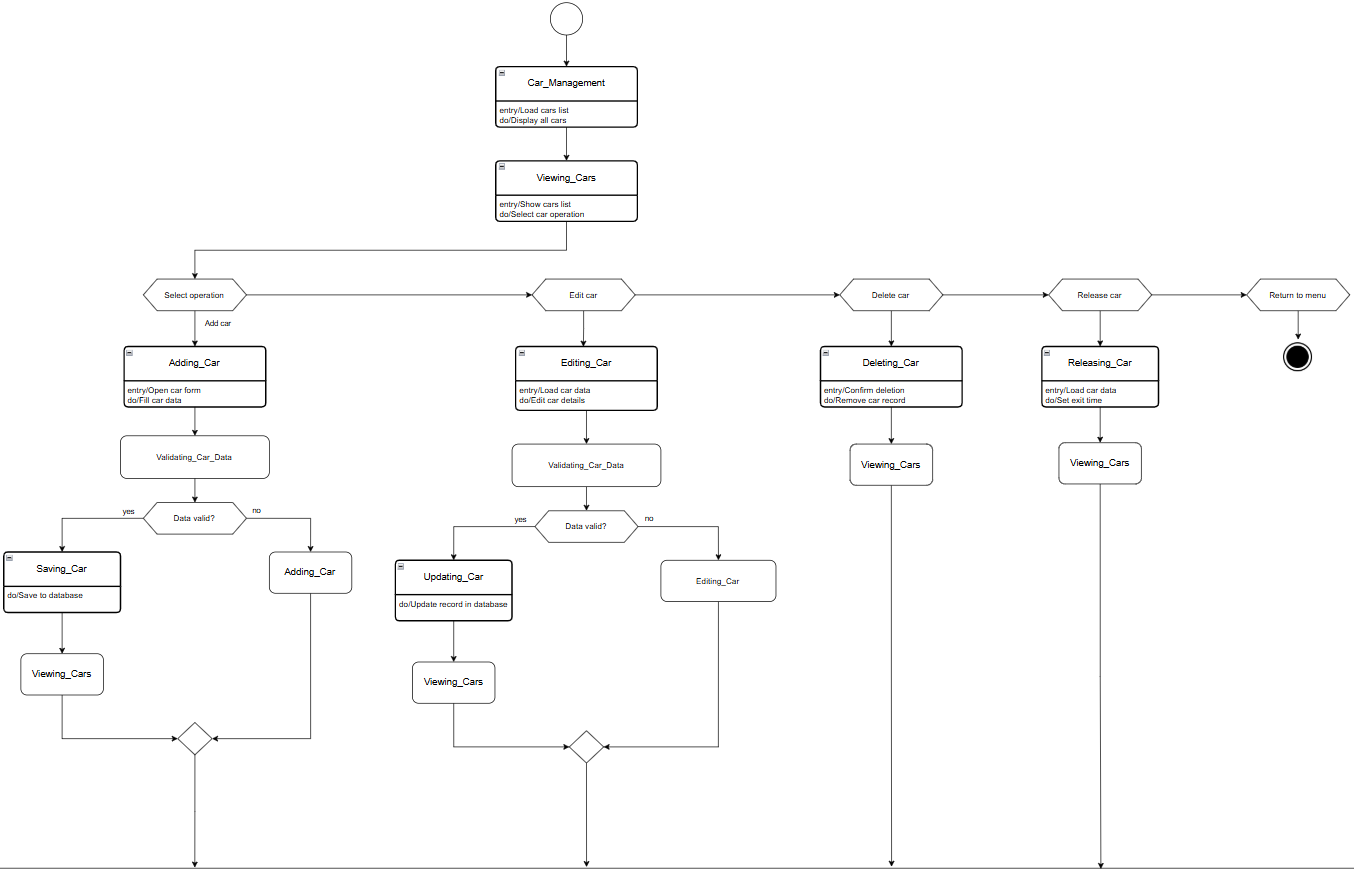


Рисунок 11. Диаграмма состояний блока «Users\_Management»

Остальные модули управления, такие как **Zone\_Management**, **Slot\_Management** и **Tariff\_Management**, имеют аналогичную структуру состояний. Каждый из них включает под-состояния для просмотра списка элементов, добавления новых записей, редактирования существующих данных, валидации введённой информации и удаления записей. Логика переходов между состояниями и последовательность действий повторяют общую модель, реализованную в модулях User\_Management и Car\_Management, что обеспечивает единообразие интерфейса и предсказуемость поведения системы.

# 4. МАКЕТЫ HTML-СТРАНИЦ РАЗРАБАТЫВАЕМОЙ СИСТЕМЫ

## 4.1. Макет страницы авторизации

Макет страницы авторизации пользователей в разрабатываемой системе изображен на рисунке 12.

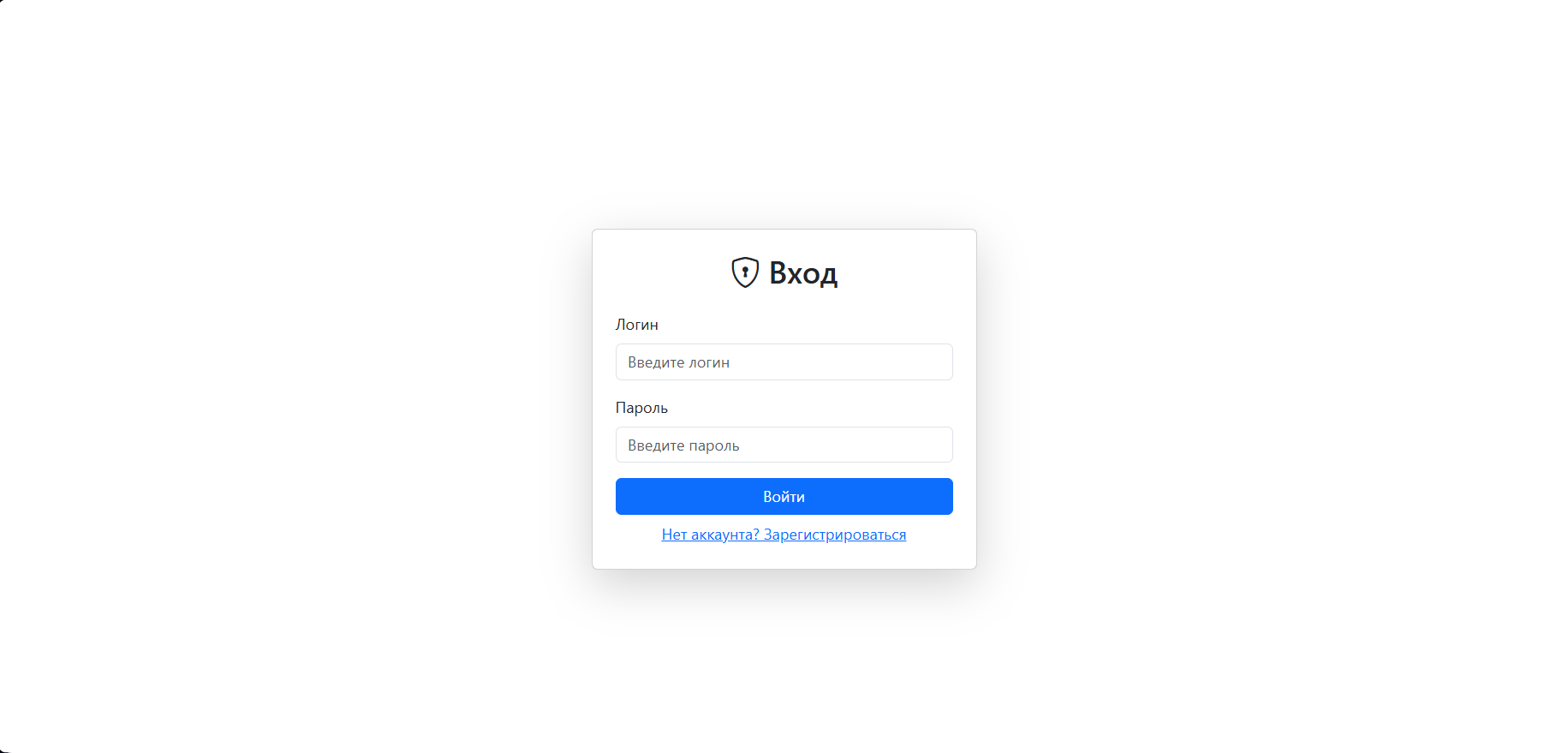


Рисунок 12. Макет страницы авторизации.

## 4.2. Макет страницы регистрации

Макет страницы регистрации пользователей в разрабатываемой системе изображен на рисунке 13.

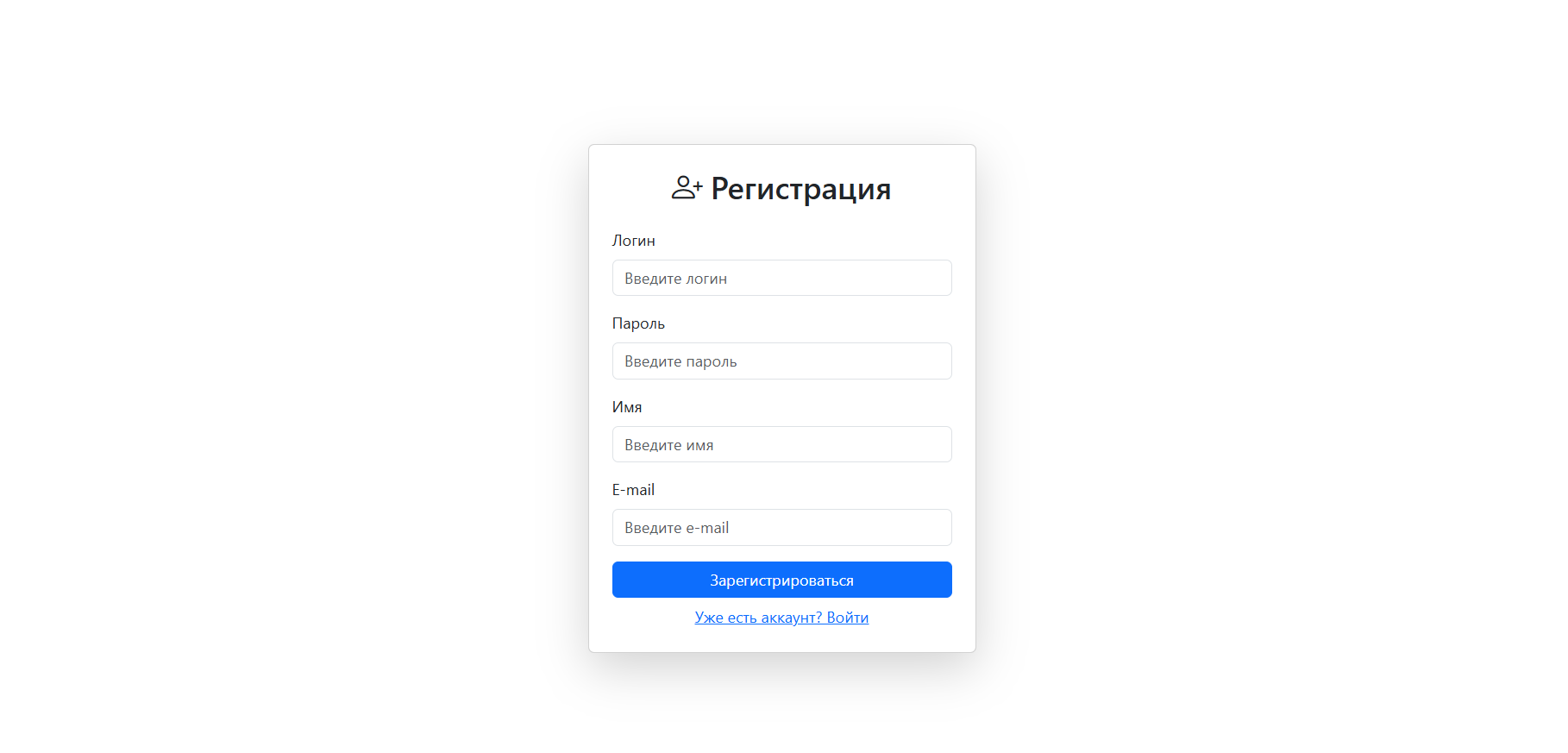


Рисунок 13. Макет страницы регистрации

## 4.3. Макет главной страницы

Макет главной страницы разрабатываемого веб-приложения изображено на рисунке 14.

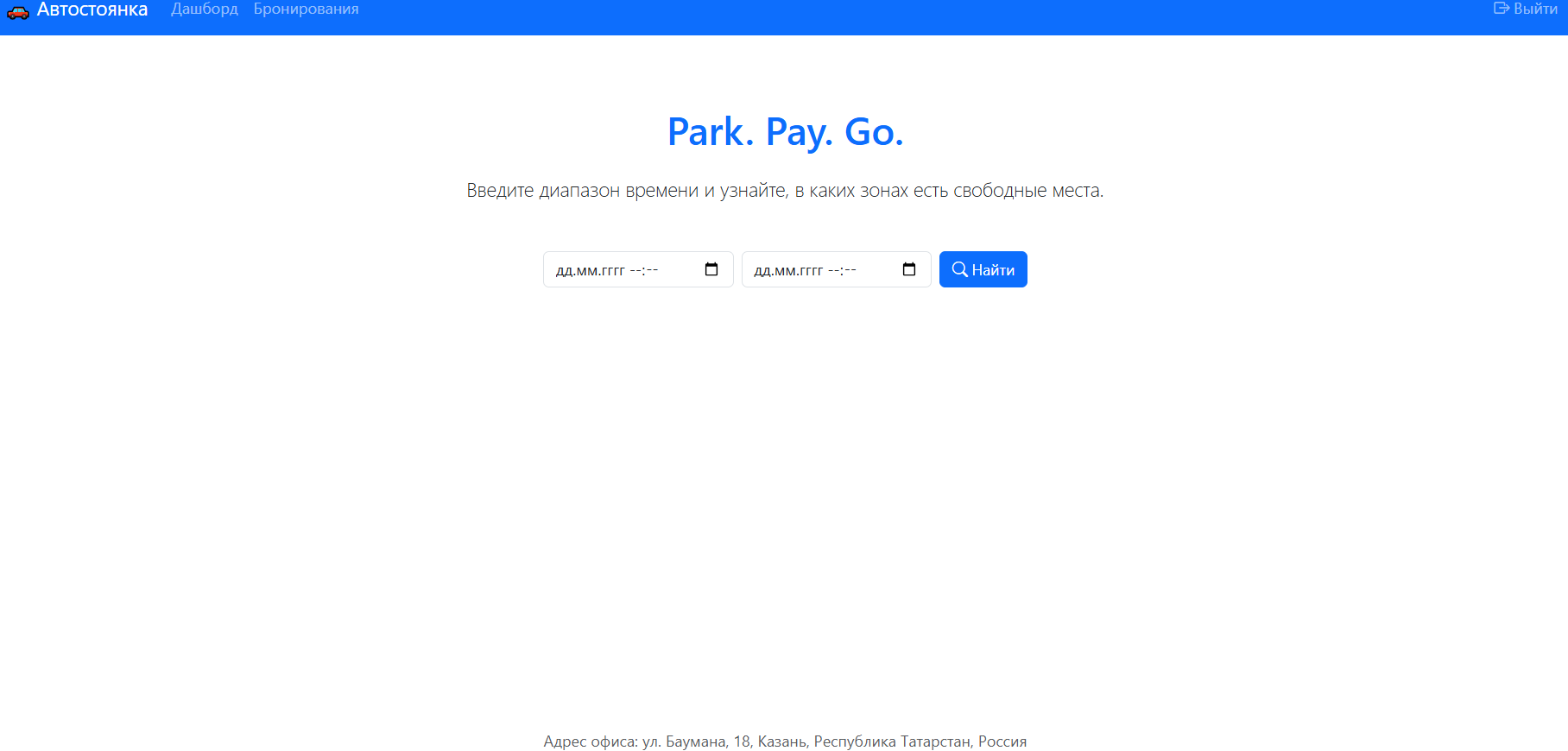


Рисунок 14. Макет страницы отчетности.

4.4. Макет системы администратора

Макет главной страницы админа в системе изображен на рисунке 15.

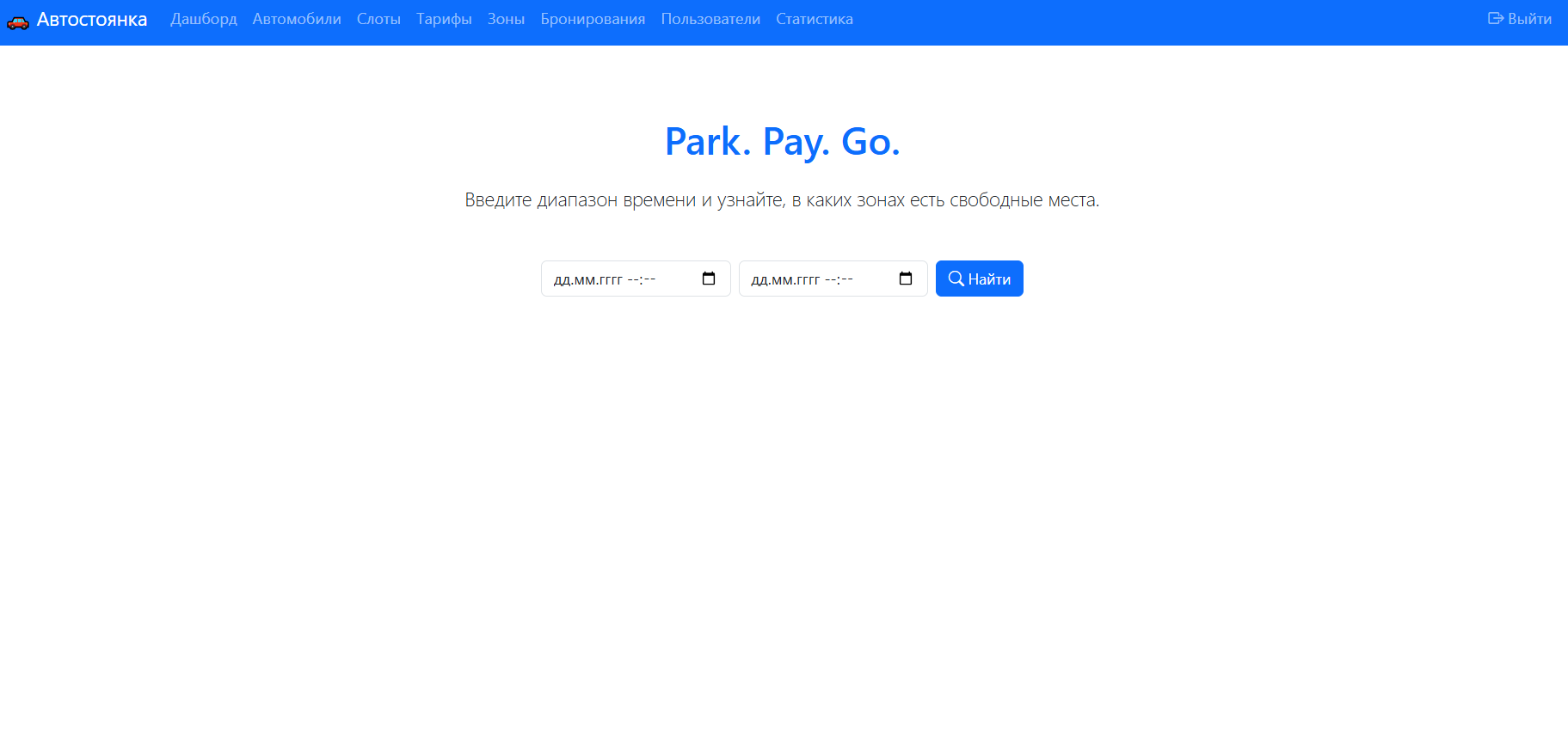


Рисунок 15. Макет системы администрирования.

## 4.5. Макет страницы статистики

Макет страницы приемов изображен на рисунке 16, так же страница отчетности будет включать в себя форму для формирования приема.

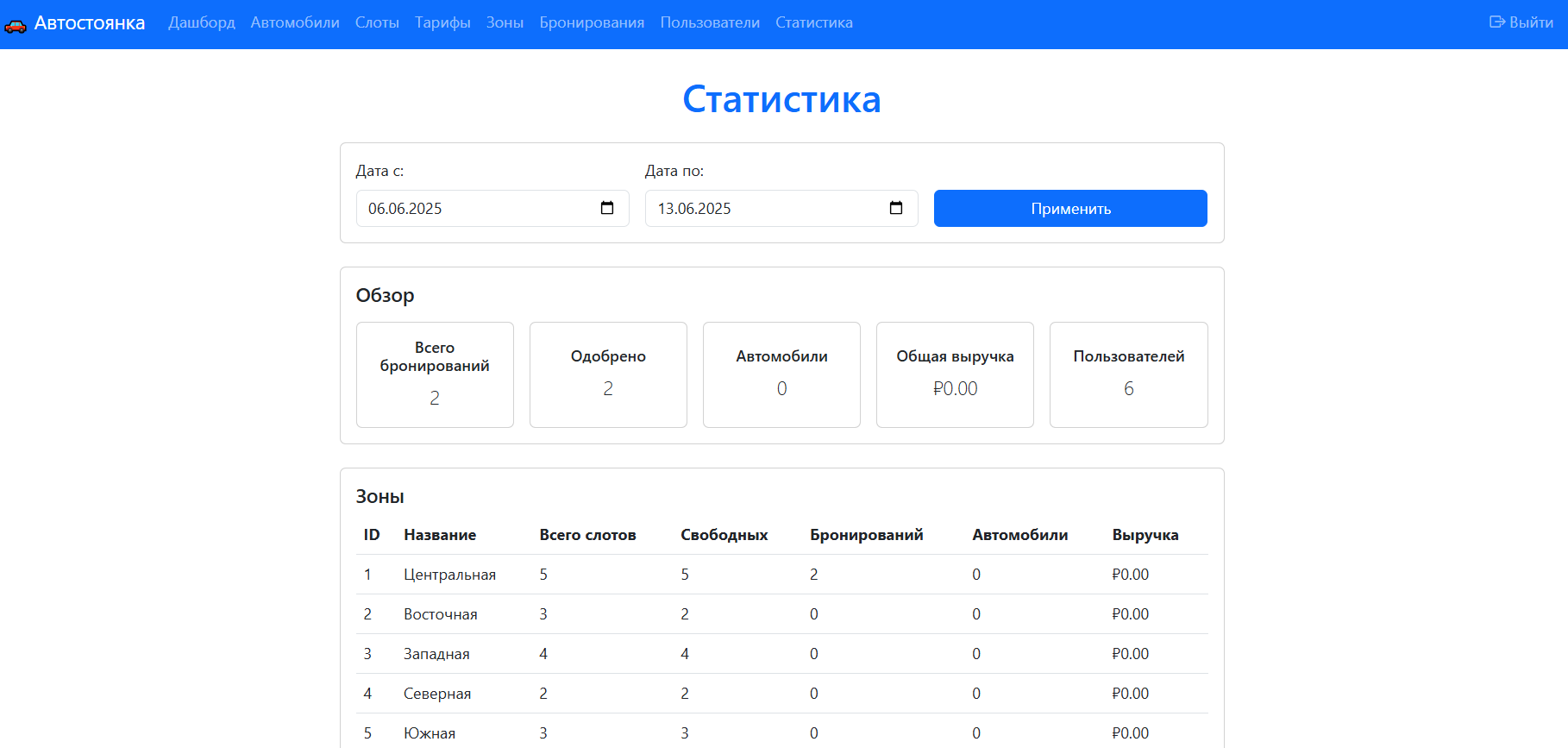


Рисунок 16. Макет страницы статистики.

# 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 страница авторизации, изображенная на рисунке 17.

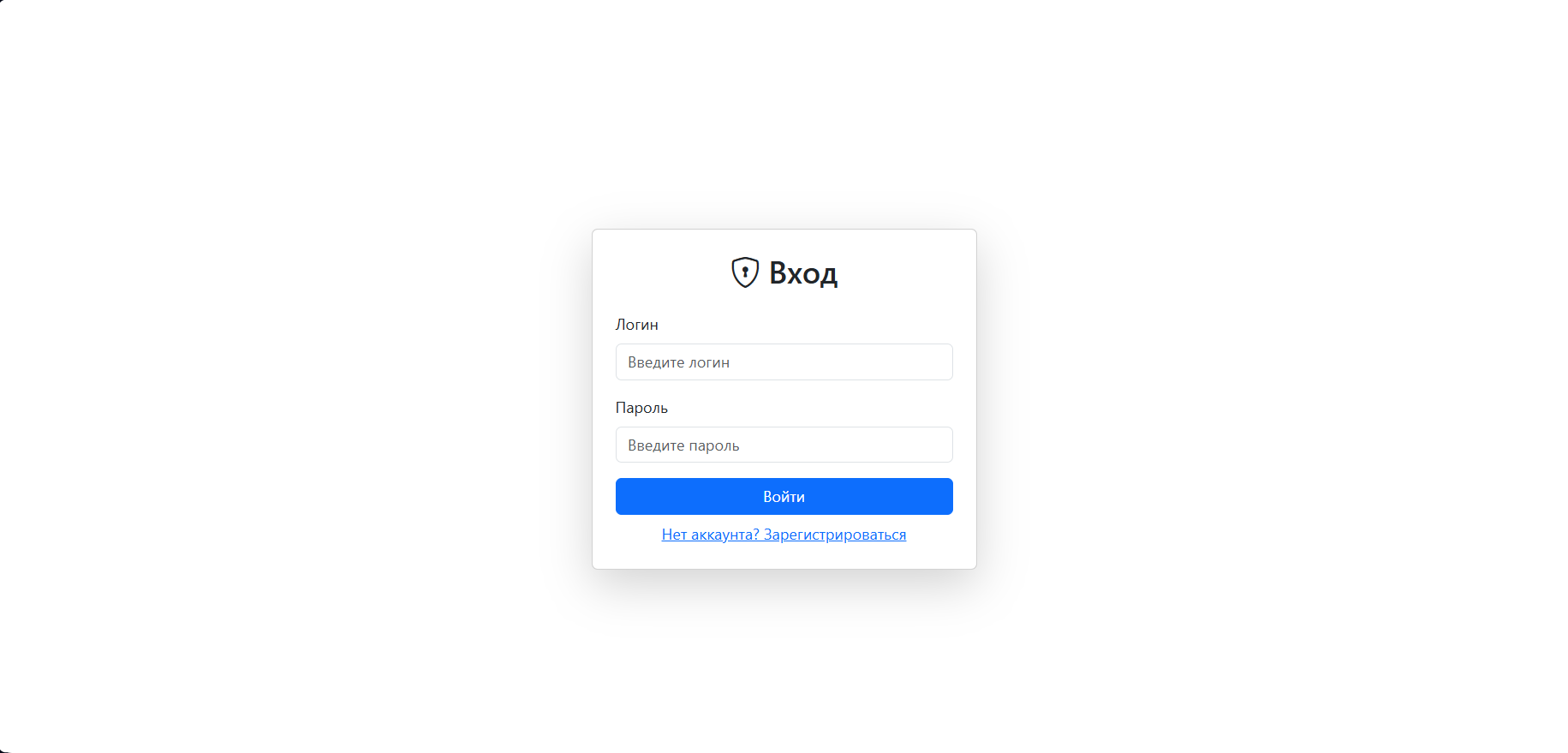


Рисунок 17. Страница авторизации в системе.

Если пользователь уже авторизирован в системе и имеет роль администратор, то главная страница будет выглядеть следующим образом (рис. 18).

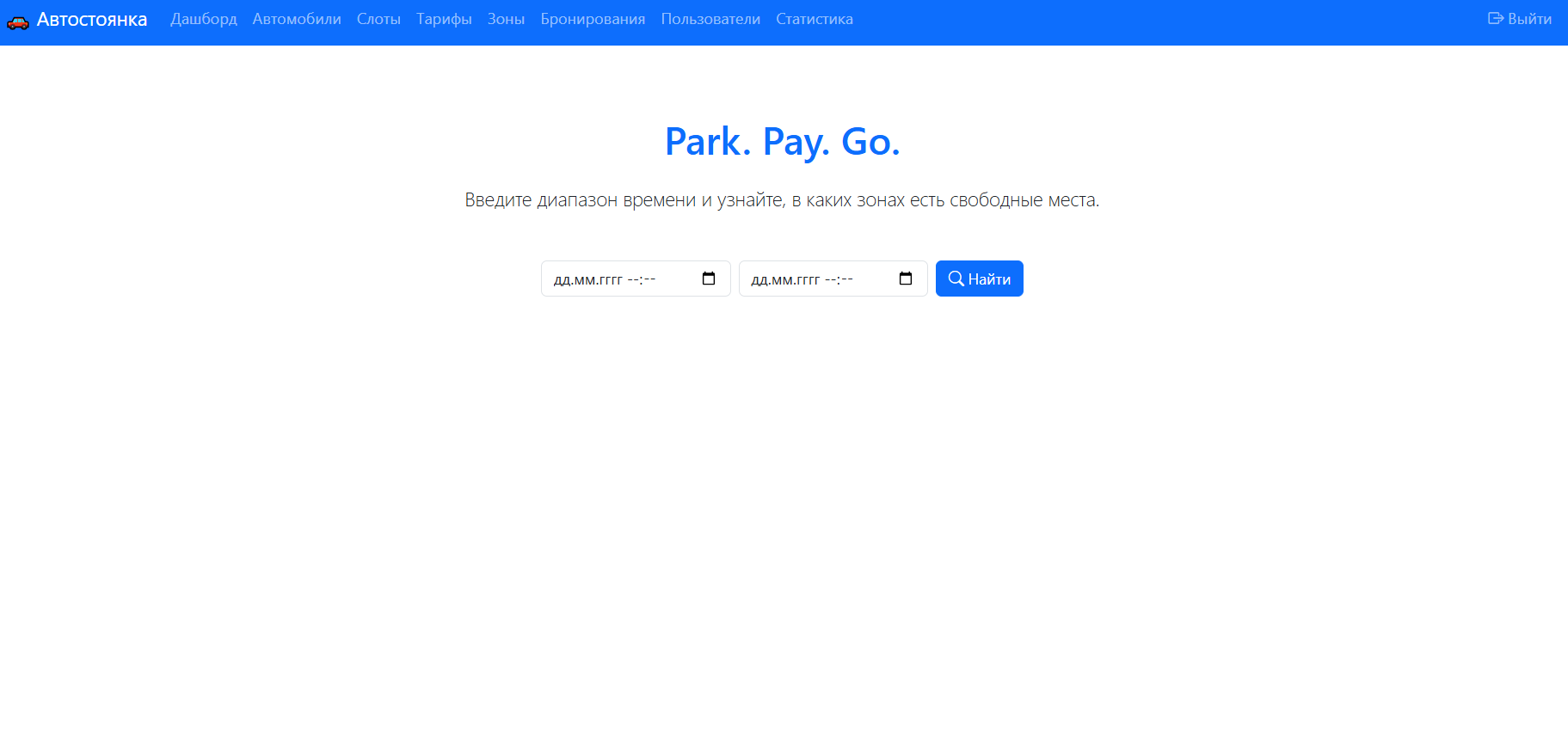


Рисунок 18. Главная страница администратора системы.

Если пользователь уже авторизирован в системе и является сотрудником (рис. 19)

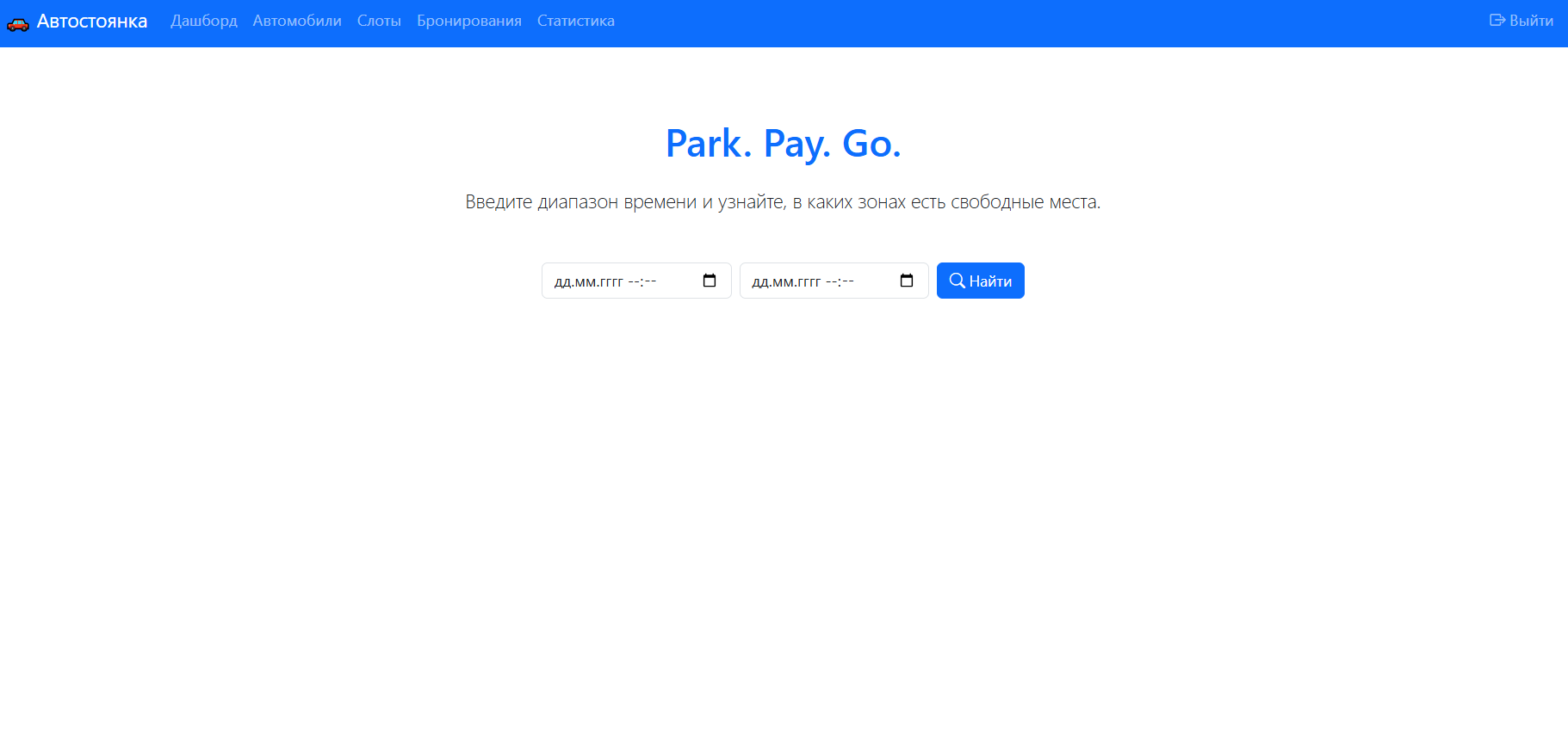


Рисунок 19. Главная страница сотрудника.

Если пользователь клиент, то главная страница для него будет выглядеть следующим образом (рис. 20).

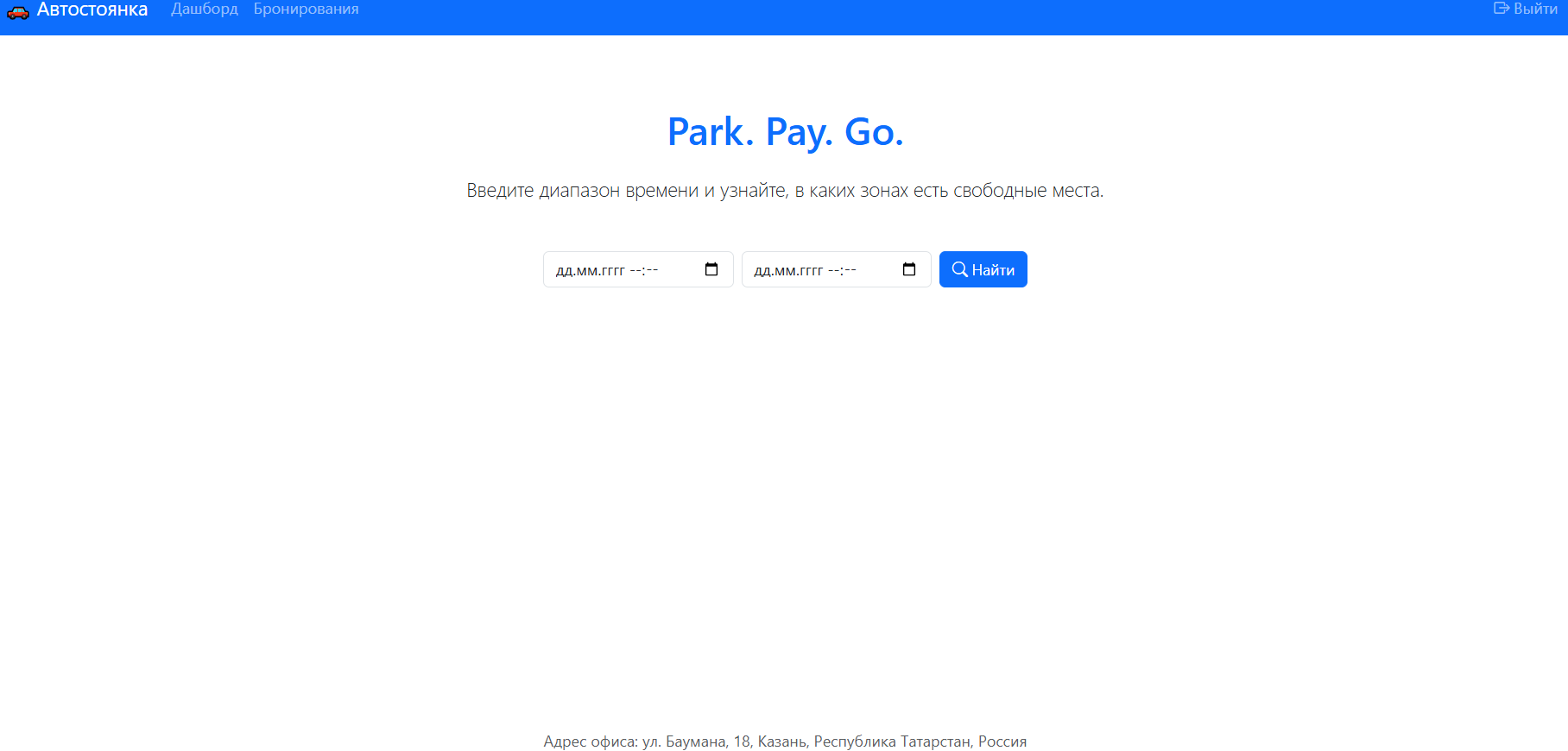


Рисунок 20. Главная страница клиента.

Перейдем к рассмотрению страниц, доступных клиентам. Помимо главной страницы, на которую попадает пользователь после авторизации, ему также доступна страница просматривания своих бронирований (рис. 21).

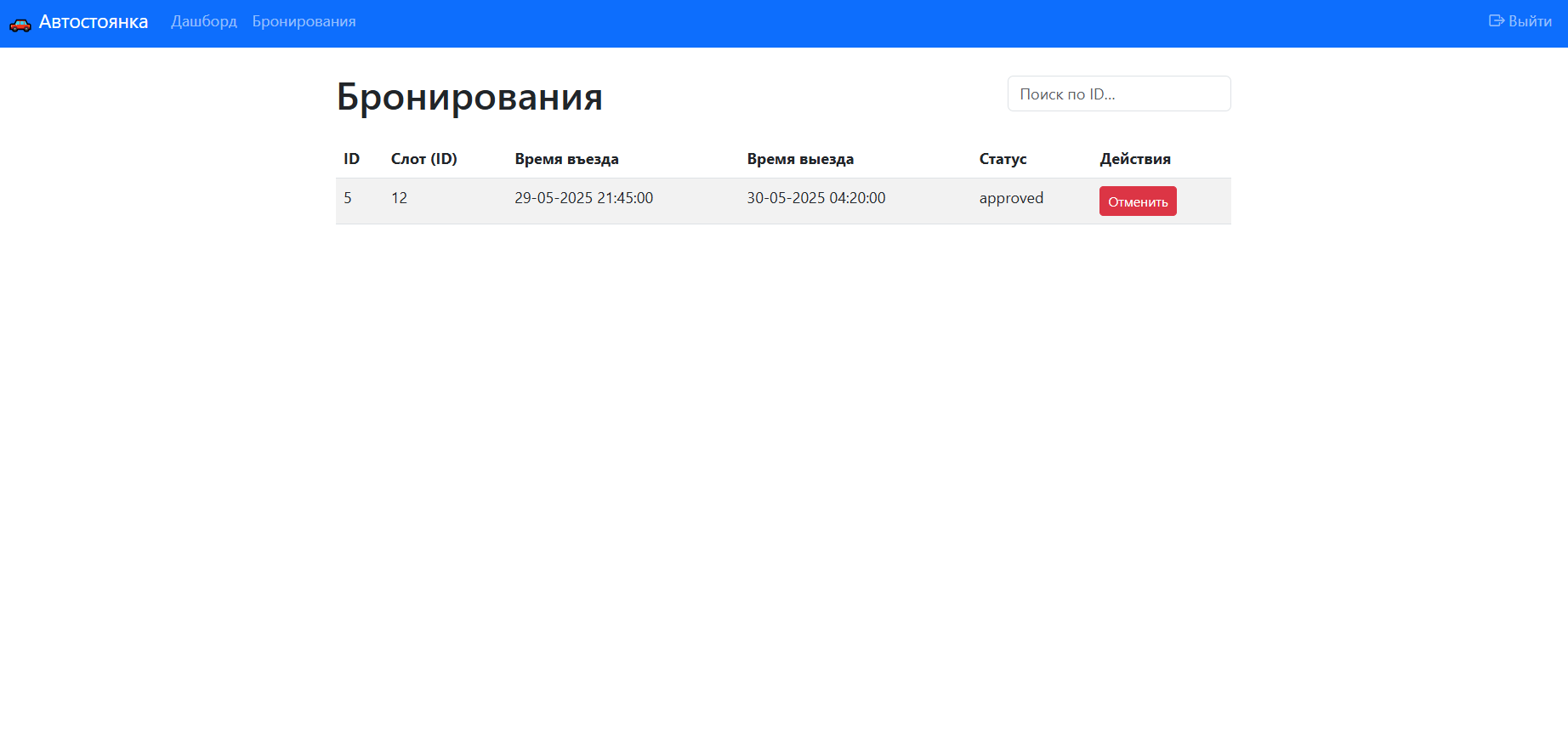


Рисунок 21. Страница «Бронирования».

Рассмотрим создание бронирования (рис.22). Для этого на главной странице нужно выбрать время въезда и выезда и пользователю будут выведены свободные слоты.

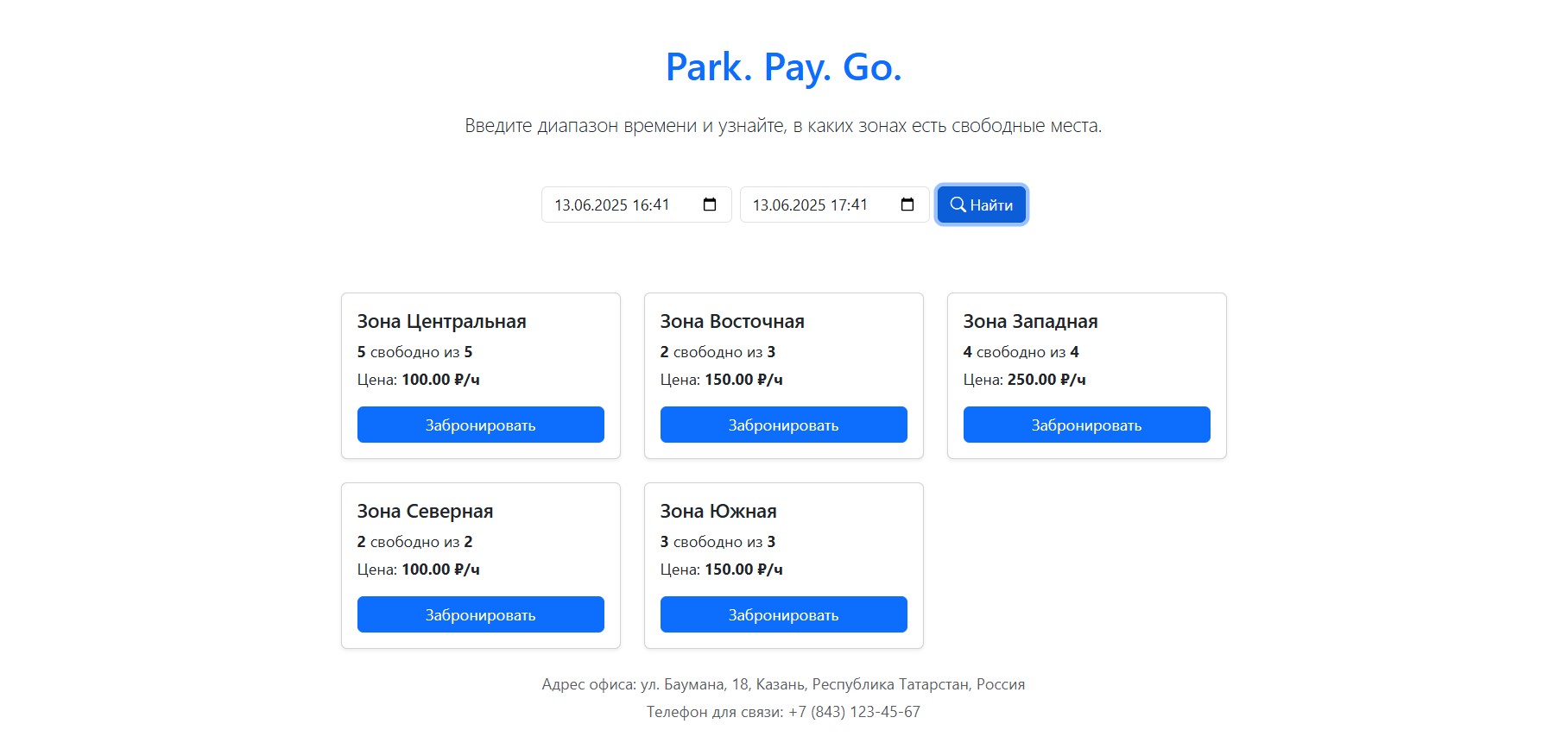


Рисунок 22. Страница «Бронирования».

Рассмотрим страницу бронирований для сотрудников и администраторов (рис.23). На ней работник может изменить статус бронирования.

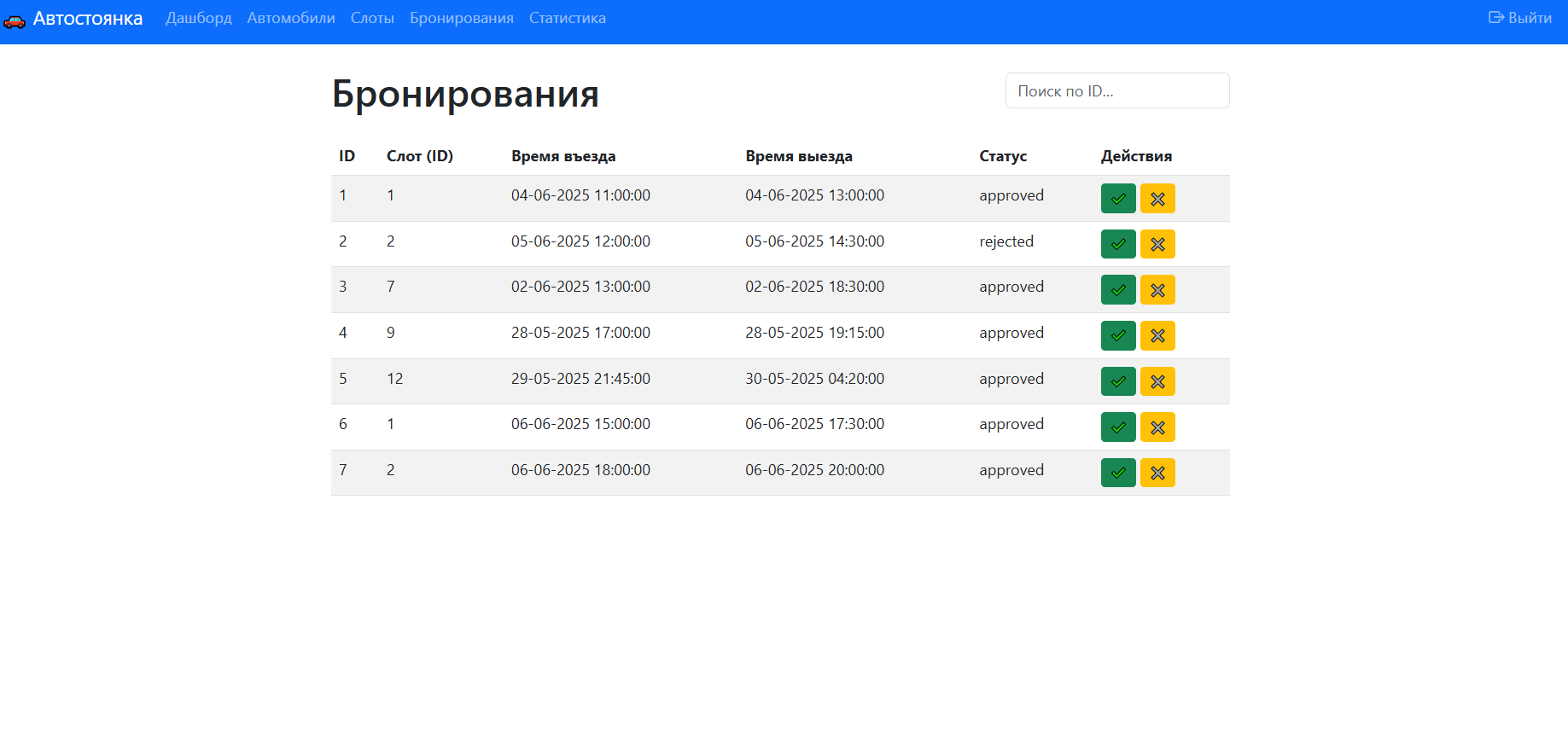


Рисунок 23. Страница «Бронирования» сотрудника и администратора.

При въезде автомобиля на парковку сотрудник должен создать автомобиль и зарегистрировать его выезд. Сотрудник делает это на странице «Автомобили» (рис.24).

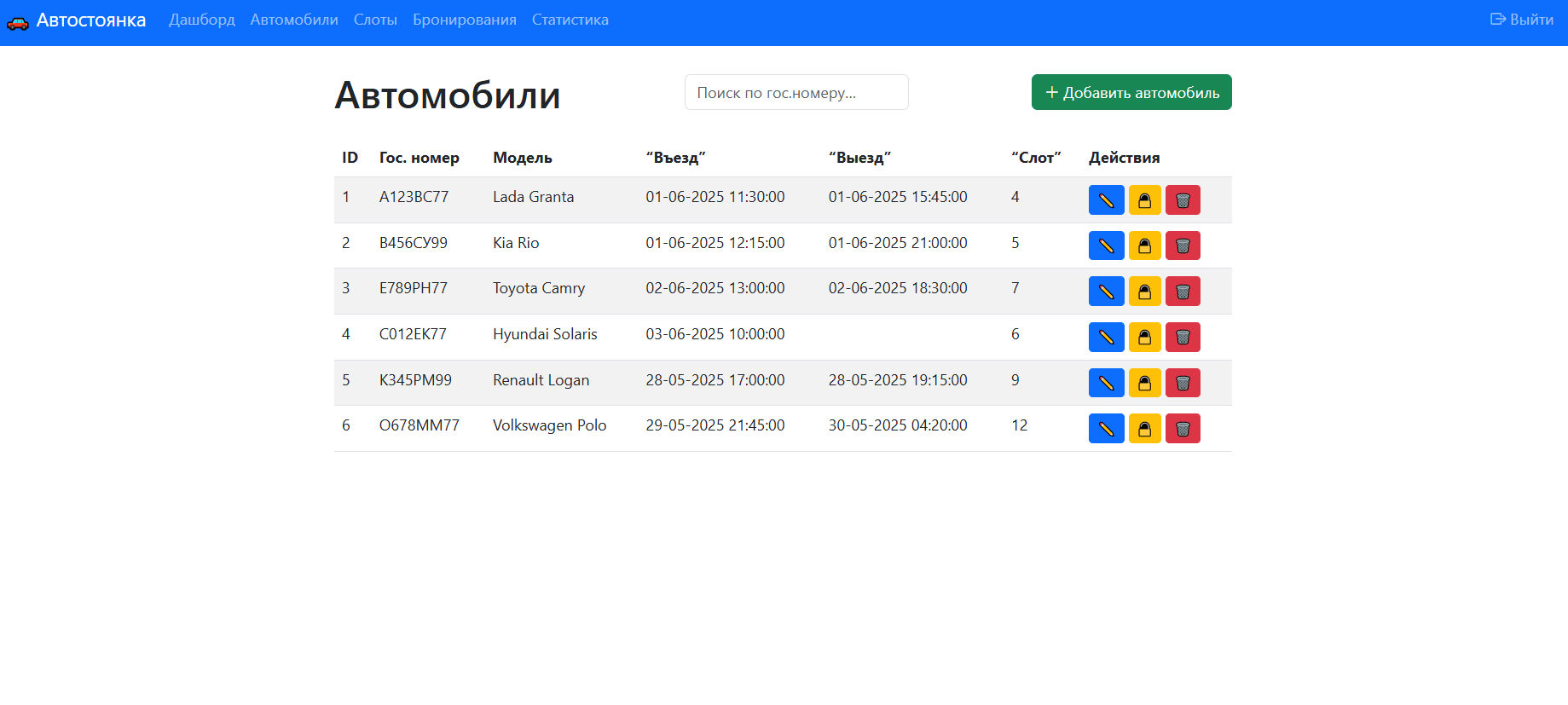


Рисунок 24. Страница «Автомобили».

Также сотрудникам и администраторам доступно управление слотами (рис.25). Они могут менять их статус, добавлять новые, менять зоны и номера

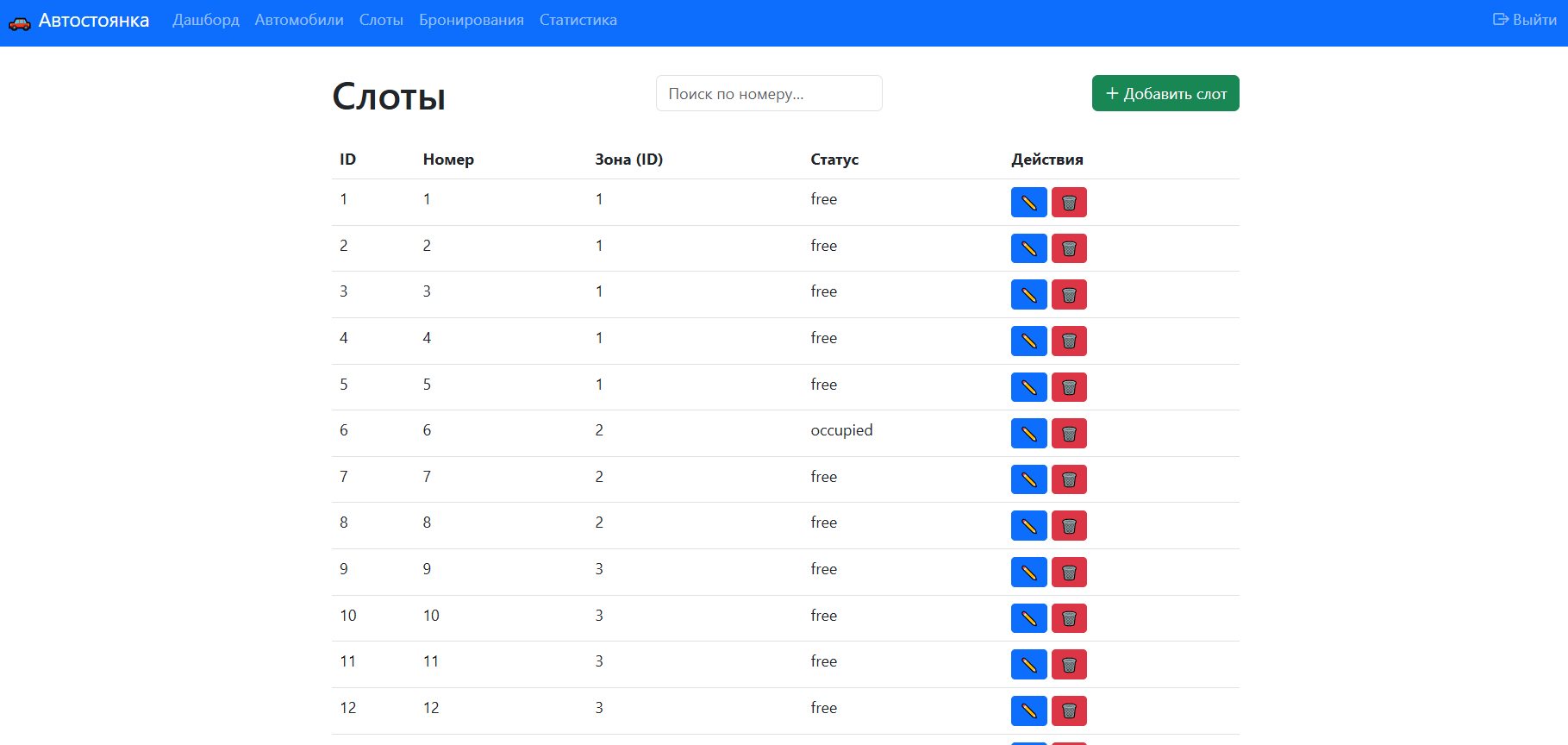


Рисунок 25. Страница «Слоты».

Перейдем к рассмотрению страницы «Тарифы», которая доступна только администраторам (рис.26). Им доступны базовые CRUD-операции и поиск

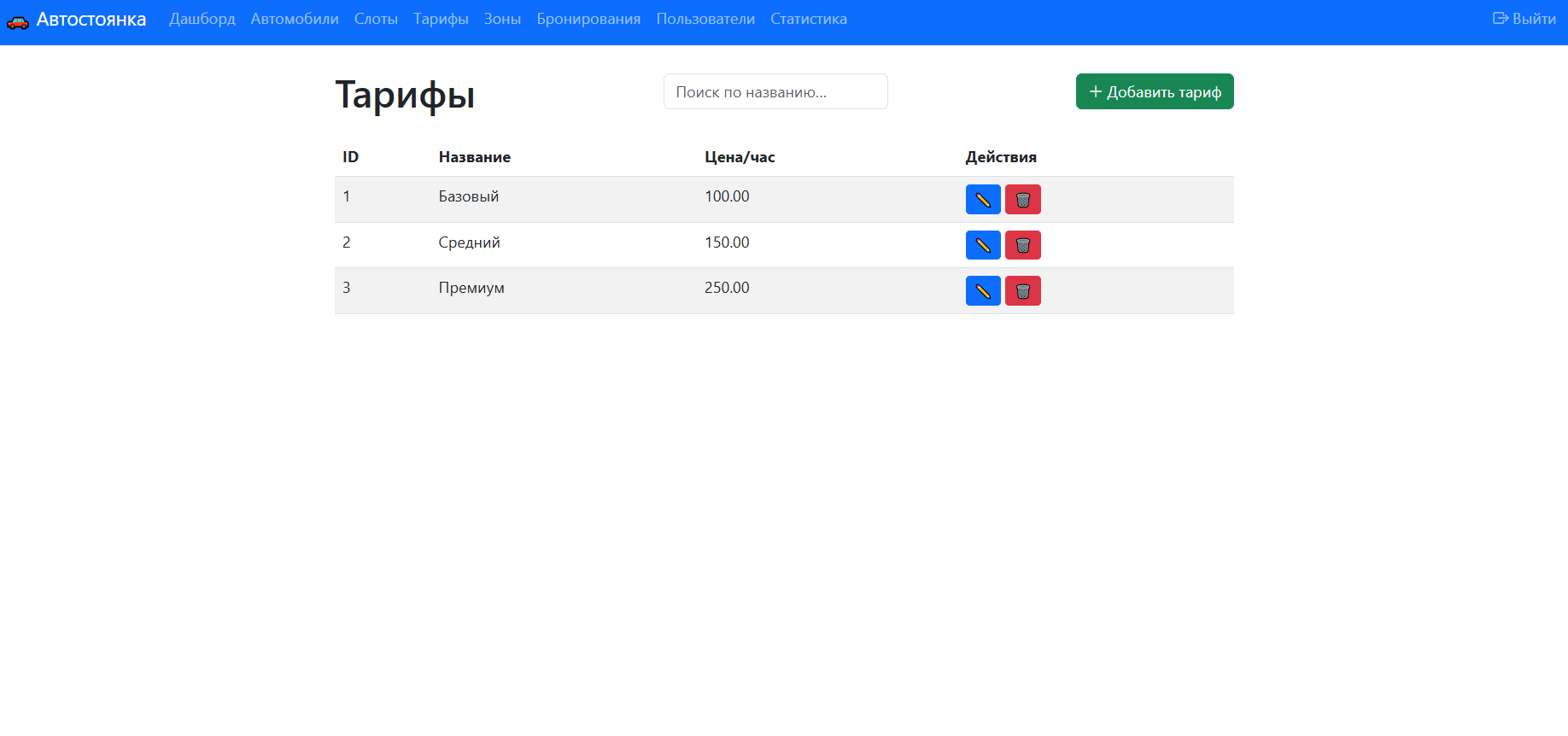


Рисунок 26. Страница «Тарифы».

Теперь рассмотрим страницу «Пользователи» доступную только администраторам (рис.27). Они могут создавать и удалять пользователей, менять им имена, пароли, логины и роли.

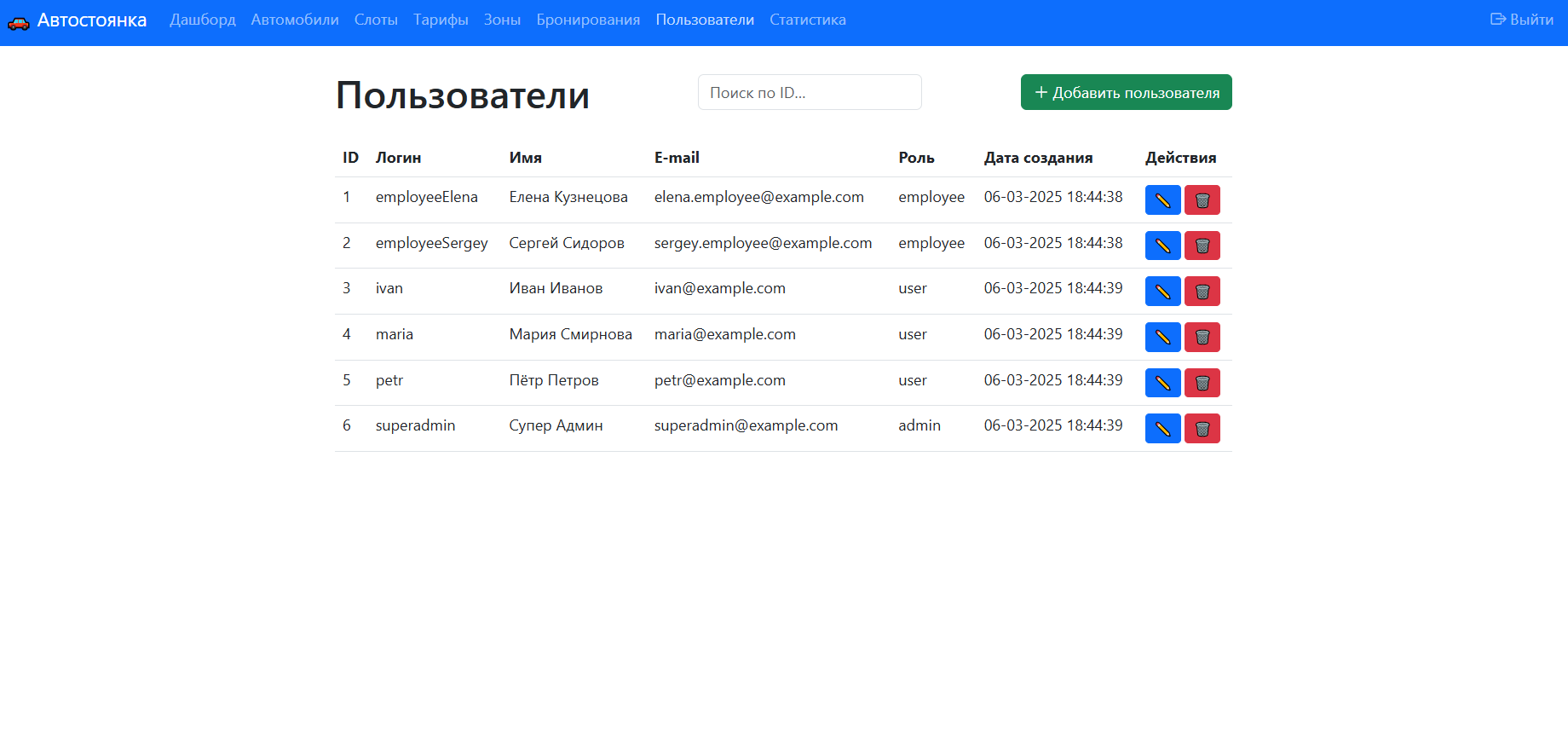


Рисунок 27. Страница «Пользователи».

Теперь рассмотрим страницу «Статистика» доступную только администраторам и сотрудникам(рис.28). Они вводят период, в котором их интересует статистика и она генерируется

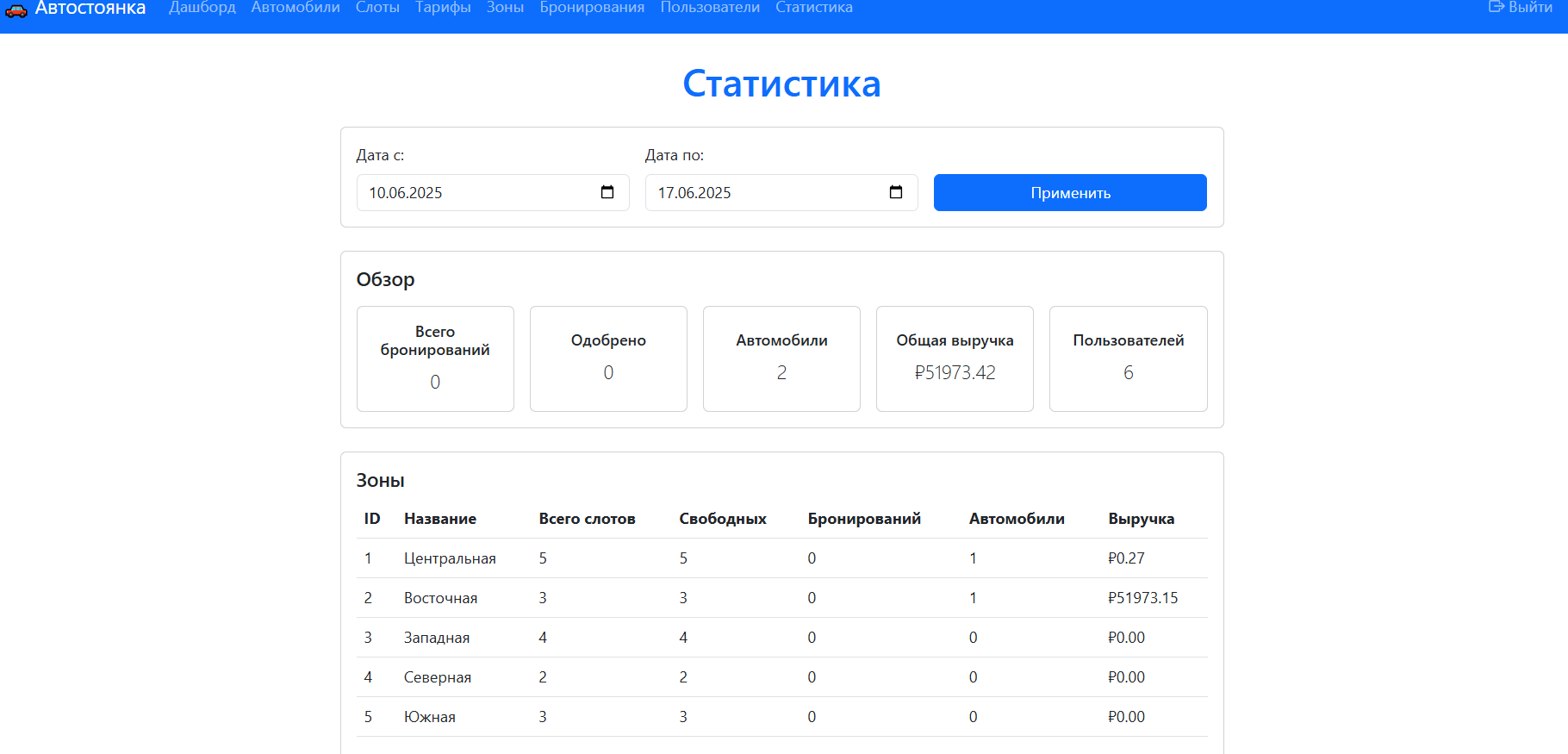


Рисунок 28. Страница «Статистика».

# 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 server/app.js

const express = *require*('express');  
const path = *require*('path');  
  
const initDB = *require*('./models/Init');  
const *config* = *require*('./config/app');  
  
const { authMiddleware } = *require*('./middlewares/AuthMiddleware');  
  
const *authRoutes* = *require*('./routes/Auth');  
const *userRoutes* = *require*('./routes/Users');  
const *tariffRoutes* = *require*('./routes/Tariffs');  
const *zoneRoutes* = *require*('./routes/Zones');  
const *slotRoutes* = *require*('./routes/Slots');  
const *carRoutes* = *require*('./routes/Cars');  
const *bookingRoutes* = *require*('./routes/Bookings');  
const *logRoutes* = *require*('./routes/Logs');  
  
const *app* = express();  
  
  
*app*.set('view engine', 'pug');  
*app*.set('views', path.join(*\_\_dirname*, '../client/views'));  
*app*.use('/static', express.static(path.join(*\_\_dirname*, '../client/public')));  
*app*.use(express.*json*());  
  
  
  
*app*.get('/', (req, res) => res.redirect('/login'));  
*app*.get('/login', (req, res) => res.render('login', { pageTitle: 'Вход', showNav: false }));  
*app*.get('/register', (req, res) => res.render('register', { pageTitle: 'Регистрация', showNav: false }));  
  
  
  
*app*.get('/dashboard', (req, res) =>  
 res.render('dashboard', { pageTitle: 'Дашборд', showNav: true })  
);  
*app*.get('/cars', (req, res) =>  
 res.render('cars', { pageTitle: 'Автомобили', showNav: true })  
);  
*app*.get('/slots', (req, res) =>  
 res.render('slots', { pageTitle: 'Слоты', showNav: true })  
);  
*app*.get('/tariffs', (req, res) =>  
 res.render('tariffs', { pageTitle: 'Тарифы', showNav: true })  
);  
*app*.get('/zones', (req, res) =>  
 res.render('zones', { pageTitle: 'Зоны', showNav: true })  
);  
*app*.get('/bookings', (req, res) =>  
 res.render('bookings', { pageTitle: 'Бронирования', showNav: true })  
);  
*app*.get('/users', (req, res) =>  
 res.render('users', { pageTitle: 'Пользователи', showNav: true })  
);  
  
*app*.use('/api/auth', *authRoutes*);  
  
*app*.use('/api', authMiddleware);  
  
*app*.use('/api/users', *userRoutes*);  
*app*.use('/api/tariffs', *tariffRoutes*);  
*app*.use('/api/zones', *zoneRoutes*);  
*app*.use('/api/slots', *slotRoutes*);  
*app*.use('/api/cars', *carRoutes*);  
*app*.use('/api/bookings', *bookingRoutes*);  
*app*.use('/api/logs', *logRoutes*);  
  
  
*app*.get('/health', (req, res) => res.json({ status: 'OK' }));  
  
  
if (*require*.main === *module*) {  
 initDB()  
 .then(() => *app*.listen(*config*.port, () => {  
 *console*.log(`🚀 Сервер на http://localhost:${*config*.port}`);  
 }))  
 .catch(err => {  
 *console*.error('Ошибка инициализации БД:', err);  
 *process*.exit(1);  
 });  
}  
  
*module*.exports = *app*;

ПРИЛОЖЕНИЕ 2. Файл server/db.js

const { Sequelize } = require('sequelize');  
const dbConfig = require('./config/database');  
  
const sequelize = new Sequelize(  
 dbConfig.database,  
 dbConfig.username,  
 dbConfig.password,  
 {  
 host: dbConfig.host,  
 port: dbConfig.port,  
 dialect: dbConfig.dialect,  
 logging: dbConfig.logging,  
 timezone: '+03:00',  
 dialectOptions: {  
 useUTC: true,  
 },  
 }  
);  
  
module.exports = sequelize;

ПРИЛОЖЕНИЕ 3. Файл server/config/app.js

*require('dotenv').config();  
  
module.exports = {  
 port: process.env.PORT || 3000  
};*

ПРИЛОЖЕНИЕ 3. Файл server/config/database.js

*require*('dotenv').config();  
  
*module*.exports = {  
 database: *process*.env.DB\_NAME,  
 username: *process*.env.DB\_USER,  
 password: *process*.env.DB\_PASSWORD,  
 host: *process*.env.DB\_HOST,  
 port: *process*.env.DB\_PORT,  
 dialect: 'postgres',  
 logging: false  
};

ПРИЛОЖЕНИЕ 4. Файл server/controllers/AuthController.js

const AuthService = *require*('../services/AuthService');  
  
class AuthController {  
 constructor(authService) {  
 this.authService = authService;  
 this.register = this.register.bind(this);  
 this.login = this.login.bind(this);  
 }  
  
   
 async register(req, res) {  
 try {  
 const { token } = await this.authService.register(req.body);  
 res.status(201).json({ token });  
 } catch (err) {  
 res.status(err.status || 500).json({ message: err.message });  
 }  
 }  
  
   
 async login(req, res) {  
 try {  
 const { token } = await this.authService.login(req.body);  
 res.json({ token });  
 } catch (err) {  
 res.status(err.status || 500).json({ message: err.message });  
 }  
 }  
}  
  
*module*.exports = AuthController;

ПРИЛОЖЕНИЕ 5. Файл server/controllers/BookingController.js

class BookingController {  
 constructor(bookingService) {  
 this.bookingService = bookingService;  
 this.createBooking = this.createBooking.bind(this);  
 this.getMyBookings = this.getMyBookings.bind(this);  
 this.getPending = this.getPending.bind(this);  
 this.approveBooking = this.approveBooking.bind(this);  
 this.rejectBooking = this.rejectBooking.bind(this);  
 this.cancelBooking = this.cancelBooking.bind(this);  
 this.getAll = this.getAll.bind(this);  
 }   
 async createBooking(req, res, next) {  
 try {  
 const userId = req.user.id;  
 const {parking\_slot\_id, start\_time, end\_time} = req.body;  
 const booking = await this.bookingService.createBooking({userId, parking\_slot\_id, start\_time, end\_time});  
 return res.status(201).json(booking);  
 } catch (err) {  
 const status = err.status || 500;  
 return res.status(status).json({message: err.message});  
 }  
 }  
 async getMyBookings(req, res, next) {  
 try {  
 const bookings = await this.bookingService.getByUser(req.user.id);  
 return res.json(bookings);  
 } catch (err) {  
 next(err);  
 }  
 }  
 async getPending(req, res, next) {  
 try {  
 const bookings = await this.bookingService.getPending();  
 return res.json(bookings);  
 } catch (err) {  
 next(err);  
 }  
 }  
 async approveBooking(req, res, next) {  
 try {  
 const updated = await this.bookingService.approveBooking(  
 req.params.id,  
 req.user.id  
 );  
 return res.json(updated);  
 } catch (err) {  
 next(err);  
 }  
 }   
 async rejectBooking(req, res, next) {  
 try {  
 const updated = await this.bookingService.rejectBooking(  
 req.params.id,  
 req.user.id  
 );  
 return res.json(updated);  
 } catch (err) {  
 next(err);  
 }  
 }  
 async cancelBooking(req, res, next) {  
 try {  
 const updated = await this.bookingService.cancelBooking(  
 req.params.id,  
 req.user.id  
 );  
 return res.json(updated);  
 } catch (err) {  
 next(err);  
 }  
 }  
 async getAll(req, res, next) {  
 try {  
 const { id } = req.query;  
 const list = await this.bookingService.getAll({ id });  
 res.json(list);  
 } catch (err) {  
 next(err);  
 }  
 }  
}  
*module*.exports = BookingController;

*module*.exports = BookingController;

ПРИЛОЖЕНИЕ 6. Файл server/controllers/CarController.js

const { CarService } = *require*('../services/CarService');  
class CarController {  
 constructor(carService) {  
 this.carService = carService;  
 this.getAll = this.getAll.bind(this);  
 this.getActive = this.getActive.bind(this);  
 this.filter = this.filter.bind(this);  
 this.create = this.create.bind(this);  
 this.release = this.release.bind(this);  
 this.update = this.update.bind(this);  
 this.delete = this.delete.bind(this);  
 }  
 async getAll(req, res, next) {  
 try {  
 const { license\_plate } = req.query;  
 const cars = await this.carService.*findAll*({ license\_plate });  
 res.json(cars);  
 } catch (err) {  
 next(err);  
 }  
 }  
 async getActive(req, res) {  
 const cars = await this.carService.getActiveCars();  
 res.json(cars);  
 }  
 async filter(req, res) {  
 try {  
 const cars = await this.carService.filter(req.query);  
 res.json(cars);  
 } catch (err) {  
 res.status(400).json({ message: err.message });  
 }  
 }  
 async create(req, res) {  
 try {  
 const car = await this.carService.create(req.body, req.user.id);  
 res.status(201).json(car);  
 } catch (err) {  
 res.status(400).json({ message: err.message });  
 }  
 }  
 async release(req, res) {  
 try {  
 const car = await this.carService.release(req.params.id, req.user.id);  
 res.json(car);  
 } catch (err) {  
 res.status(400).json({ message: err.message });  
 }  
 }  
 async update(req, res) {  
 try {  
 const car = await this.carService.update(req.params.id, req.body, req.user.id);  
 res.json(car);  
 } catch (err) {  
 res.status(400).json({ message: err.message });  
 }  
 }  
 async delete(req, res) {  
 try {  
 await this.carService.delete(req.params.id, req.user.id);  
 res.status(204).end();  
 } catch (err) {  
 res.status(400).json({ message: err.message });  
 }  
 }  
}  
*module*.exports = CarController;

ПРИЛОЖЕНИЕ 7. Файл server/controllers/LogController.js

const { LogService } = *require*('../services/LogService');  
  
class LogController {  
 constructor(logService) {  
 this.logService = logService;  
 this.getAll = this.getAll.bind(this);  
 this.getByUser = this.getByUser.bind(this);  
 }  
  
   
 async getAll(req, res) {  
 try {  
 const logs = await this.logService.getAll();  
 res.json(logs);  
 } catch (err) {  
 res.status(500).json({ message: err.message });  
 }  
 }  
  
   
 async getByUser(req, res) {  
 try {  
 const userId = req.params.userId;  
 const logs = await this.logService.getByUser(userId);  
 res.json(logs);  
 } catch (err) {  
 res.status(500).json({ message: err.message });  
 }  
 }  
}  
  
*module*.exports = LogController;

ПРИЛОЖЕНИЕ 8. Файл server/controllers/ParkingSlotController.js

class ParkingSlotController {  
 constructor(slotService) {  
 this.slotService = slotService;  
 this.getAll = this.getAll.bind(this);  
 this.getFree = this.getFree.bind(this);  
 this.getOccupied = this.getOccupied.bind(this);  
 this.getByZone = this.getByZone.bind(this);  
 this.create = this.create.bind(this);  
 this.update = this.update.bind(this);  
 this.delete = this.delete.bind(this);  
 this.checkAvailability = this.checkAvailability.bind(this);  
 }  
 async getAll(req, res, next) {  
 try {  
 const { slot\_number } = req.query;  
 const slots = await this.slotService.getAll({ slot\_number });  
 res.json(slots);  
 } catch (err) {  
 next(err);  
 }  
 }   
 async getFree(req, res, next) {  
 try {  
 const slots = await this.slotService.getFreeSlots();  
 res.json(slots);  
 } catch (err) {  
 next(err);  
 }  
 }  
 async getOccupied(req, res, next) {  
 try {  
 const slots = await this.slotService.getOccupiedSlots();  
 res.json(slots);  
 } catch (err) {  
 next(err);  
 }  
 }  
 async getByZone(req, res, next) {  
 try {  
 const slots = await this.slotService.getSlotsByZone(req.params.zoneId);  
 res.json(slots);  
 } catch (err) {  
 next(err);  
 }  
 }  
 async create(req, res, next) {  
 try {  
 const slot = await this.slotService.create(req.body, req.user.id);  
 res.status(201).json(slot);  
 } catch (err) {  
 res.status(err.status || 400).json({ message: err.message });  
 }  
 }  
 async update(req, res, next) {  
 try {  
 const slot = await this.slotService.update(req.params.id, req.body, req.user.id);  
 res.json(slot);  
 } catch (err) {  
 res.status(err.status || 400).json({ message: err.message });  
 }  
 }  
 async delete(req, res, next) {  
 try {  
 await this.slotService.delete(req.params.id, req.user.id);  
 res.status(204).end();  
 } catch (err) {  
 res.status(err.status || 400).json({ message: err.message });  
 }  
 }  
 async checkAvailability(req, res, next) {  
 try {  
 const { start, end } = req.query;  
 const zones = await this.slotService.getZonesAvailability(start, end);  
 res.json(zones);  
 } catch (err) {  
 next(err);  
 }  
 }  
}  
*module*.exports = ParkingSlotController;

ПРИЛОЖЕНИЕ 9. Файл server/controllers/TariffController.js

const { TariffService } = *require*('../services/TariffService');  
class TariffController {  
 constructor(tariffService) {  
 this.tariffService = tariffService;  
 this.getAll = this.getAll.bind(this);  
 this.getById = this.getById.bind(this);  
 this.create = this.create.bind(this);  
 this.update = this.update.bind(this);  
 this.delete = this.delete.bind(this);  
 }  
 async getAll(req, res, next) {  
 try {  
 const { name } = req.query;  
 const list = await this.tariffService.getAll({ name });  
 res.json(list);  
 } catch (err) {  
 next(err);  
 }  
 }  
 async getById(req, res) {  
 const item = await this.tariffService.getById(req.params.id);  
 if (!item) return res.status(404).json({ message: 'Tariff not found' });  
 res.json(item);  
 }  
 async create(req, res) {  
 try {  
 const tariff = await this.tariffService.create(req.body, req.user.id);  
 res.status(201).json(tariff);  
 } catch (err) {  
 res.status(400).json({ message: err.message });  
 }  
 }  
 async update(req, res) {  
 try {  
 const tariff = await this.tariffService.update(req.params.id, req.body, req.user.id);  
 res.json(tariff);  
 } catch (err) {  
 res.status(400).json({ message: err.message });  
 }  
 }  
 async delete(req, res) {  
 try {  
 await this.tariffService.delete(req.params.id, req.user.id);  
 res.status(204).end();  
 } catch (err) {  
 res.status(400).json({ message: err.message });  
 }  
 }  
}  
*module*.exports = TariffController;

ПРИЛОЖЕНИЕ 10. Файл server/controllers/UserController.js

const {UserService} = *require*('../services/UserService');  
class UserController {  
 constructor(userService) {  
 this.userService = userService;  
 this.getAll = this.getAll.bind(this);  
 this.getById = this.getById.bind(this);  
 this.create = this.create.bind(this);  
 this.update = this.update.bind(this);  
 this.delete = this.delete.bind(this);  
 }  
 async getAll(req, res, next) {  
 try {  
 const { id } = req.query;  
 const users = await this.userService.getAll({ id });  
 res.json(users);  
 } catch(err) { next(err); }  
 }  
 async getById(req, res) {  
 const user = await this.userService.getById(req.params.id);  
 if (!user) return res.status(404).json({message: 'User not found'});  
 res.json(user);  
 }  
 async update(req, res) {  
 try {  
 const user = await this.userService.update(req.params.id, req.body, req.user.id);  
 res.json(user);  
 } catch (err) {  
 res.status(400).json({message: err.message});  
 }  
 }  
 async delete(req, res) {  
 try {  
 await this.userService.delete(req.params.id, req.user.id);  
 res.status(204).end();  
 } catch (err) {  
 res.status(400).json({message: err.message});  
 }  
 }  
 async create(req, res) {  
 try {  
 const newUser = await this.userService.create(req.body, req.user.id);  
 res.status(201).json(newUser);  
 } catch (err) {  
 res.status(err.status || 500).json({message: err.message});  
 }  
 }  
}  
*module*.exports = UserController;

ПРИЛОЖЕНИЕ 11. Файл server/controllers/ZoneController.js

const { ZoneService } = *require*('../services/ZoneService');  
class ZoneController {  
 constructor(zoneService) {  
 this.zoneService = zoneService;  
 this.getAll = this.getAll.bind(this);  
 this.getById = this.getById.bind(this);  
 this.create = this.create.bind(this);  
 this.update = this.update.bind(this);  
 this.delete = this.delete.bind(this);  
 }  
 async getAll(req, res, next) {  
 try {  
 const { name } = req.query;  
 const list = await this.zoneService.getAll({ name });  
 res.json(list);  
 } catch (err) {  
 next(err);  
 }  
 }  
 async getById(req, res) {  
 const item = await this.zoneService.getById(req.params.id);  
 if (!item) return res.status(404).json({ message: 'Zone not found' });  
 res.json(item);  
 }  
 async create(req, res) {  
 try {  
 const zone = await this.zoneService.create(req.body, req.user.id);  
 res.status(201).json(zone);  
 } catch (err) {  
 res.status(400).json({ message: err.message });  
 }  
 }  
 async update(req, res) {  
 try {  
 const zone = await this.zoneService.update(req.params.id, req.body, req.user.id);  
 res.json(zone);  
 } catch (err) {  
 res.status(400).json({ message: err.message });  
 }  
 }  
 async delete(req, res) {  
 try {  
 await this.zoneService.delete(req.params.id, req.user.id);  
 res.status(204).end();  
 } catch (err) {  
 res.status(400).json({ message: err.message });  
 }  
 }  
}  
*module*.exports = ZoneController;

ПРИЛОЖЕНИЕ 12. Файл server/middlewares /AuthMiddleware.js

const *jwt* = *require*('jsonwebtoken');  
const { User } = *require*('../models/Models');  
*require*('dotenv').config();  
  
async function authMiddleware(req, res, next) {  
 const auth = req.headers.authorization;  
 if (!auth || !auth.startsWith('Bearer ')) {  
 return res.status(401).json({ message: 'Токен не передан' });  
 }  
 const token = auth.slice(7);  
 try {  
 const payload = *jwt*.verify(token, *process*.env.JWT\_SECRET);  
 const user = await User.*findByPk*(payload.id);  
 if (!user) {  
 return res.status(401).json({ message: 'Пользователь не найден' });  
 }  
 req.user = user;   
 next();  
 } catch (err) {  
 return res.status(401).json({ message: 'Неверный токен' });  
 }  
}  
  
function roleCheck(allowedRoles) {  
 return (req, res, next) => {  
 if (!req.user) {  
 return res.status(401).json({ message: 'Неавторизован' });  
 }  
 if (!allowedRoles.includes(req.user.role)) {  
 return res.status(403).json({ message: 'Доступ запрещён' });  
 }  
 next();  
 };  
}  
  
*module*.exports = { authMiddleware, roleCheck };

ПРИЛОЖЕНИЕ 13. Файл server/middlewares /PageAuth.js

const { authMiddleware } = *require*('./AuthMiddleware');  
  
function pageRoleCheck(roles) {  
 return [  
 authMiddleware,  
 (req, res, next) => {  
 const role = req.user?.role;  
 if (!role || !roles.includes(role)) {  
 return res.redirect('/dashboard');  
 }  
 next();  
 }  
 ];  
}  
  
*module*.exports = { pageRoleCheck };

ПРИЛОЖЕНИЕ 14. Файл server/models /Init.js

const *sequelize* = *require*('../db');  
const *models* = *require*('./Models');   
  
let initialized = false;  
  
async function initDB() {  
 try {  
 await *sequelize*.authenticate();  
 *console*.log('🔌 Подключение к базе данных установлено.');  
  
 if (*process*.env.NODE\_ENV === 'test') {  
   
 await *sequelize*.sync({ force: true });  
 *console*.log('✅ Все модели принудительно пересозданы (test mode).');  
 } else {  
   
 if (!initialized) {  
 await *sequelize*.sync({ alter: true });  
 initialized = true;  
 *console*.log('✅ Все модели синхронизированы.');  
 }  
 }  
 } catch (error) {  
 *console*.error('❌ Ошибка подключения к БД:', error);  
 }  
}  
  
*module*.exports = initDB;

ПРИЛОЖЕНИЕ 15. Файл server/models /Models.js

const *sequelize* = *require*('../db');  
const { DataTypes } = *require*('sequelize');  
function formatDateUTC3(date) {  
 if (!date) return null;   
 const dt = new *Date*(date.getTime() + 3 \* 60 \* 60 \* 1000);  
 const dd = String(dt.getUTCDate()).padStart(2, '0');  
 const mm = String(dt.getUTCMonth() + 1).padStart(2, '0');  
 const yyyy = dt.getUTCFullYear();  
 const hh = String(dt.getUTCHours()).padStart(2, '0');  
 const mi = String(dt.getUTCMinutes()).padStart(2, '0');  
 const ss = String(dt.getUTCSeconds()).padStart(2, '0');  
 return `${dd}-${mm}-${yyyy} ${hh}:${mi}:${ss}`;  
}  
const User = *sequelize*.define('user', {  
 id: { type: DataTypes.*INTEGER*, primaryKey: true, autoIncrement: true },  
 username: { type: DataTypes.*STRING*, unique: true, allowNull: false },  
 password: { type: DataTypes.*STRING*, allowNull: false },  
 role: { type: DataTypes.*ENUM*('admin','employee','user'), defaultValue: 'user' },  
 fullname: { type: DataTypes.*STRING* },  
 email: { type: DataTypes.*STRING*, unique: true },  
 createdAt: {  
 type: DataTypes.*DATE*,  
 defaultValue: DataTypes.NOW,  
 get() {  
 return formatDateUTC3(this.getDataValue('createdAt'));  
 }  
 }  
}, {  
 updatedAt: false   
});  
const Tariff = *sequelize*.define('tariff', {  
 id: { type: DataTypes.*INTEGER*, primaryKey: true, autoIncrement: true },  
 name: { type: DataTypes.*STRING*, unique: true, allowNull: false },  
 price\_per\_hour: { type: DataTypes.*DECIMAL*(10,2), allowNull: false }  
}, {  
 timestamps: false  
});  
const Zone = *sequelize*.define('zone', {  
 id: { type: DataTypes.*INTEGER*, primaryKey: true, autoIncrement: true },  
 name: { type: DataTypes.*STRING*, unique: true, allowNull: false },  
 tariff\_id: { type: DataTypes.*INTEGER*, allowNull: false }  
}, {  
 timestamps: false  
});  
const ParkingSlot = *sequelize*.define('parking\_slot', {  
 id: { type: DataTypes.*INTEGER*, primaryKey: true, autoIncrement: true },  
 slot\_number: { type: DataTypes.*INTEGER*, unique: true, allowNull: false },  
 status: { type: DataTypes.*ENUM*('free','occupied'), defaultValue: 'free' },  
 zone\_id: { type: DataTypes.*INTEGER*, allowNull: false }  
}, {  
 timestamps: false  
});  
const Car = *sequelize*.define('car', {  
 id: { type: DataTypes.*INTEGER*, primaryKey: true, autoIncrement: true },  
 license\_plate: { type: DataTypes.*STRING*, unique: true },  
 model: { type: DataTypes.*STRING* },  
 entry\_time: {  
 type: DataTypes.*DATE*,  
 defaultValue: DataTypes.NOW,  
 get() {  
 return formatDateUTC3(this.getDataValue('entry\_time'));  
 }  
 },  
 exit\_time: {  
 type: DataTypes.*DATE*,  
 allowNull: true,  
 get() {  
 return formatDateUTC3(this.getDataValue('exit\_time'));  
 }  
 },  
 user\_id: { type: DataTypes.*INTEGER*, allowNull: false },  
 parking\_slot\_id: { type: DataTypes.*INTEGER*, allowNull: false }  
}, {  
 updatedAt: false   
});  
const Booking = *sequelize*.define('booking', {  
 id: { type: DataTypes.*INTEGER*, primaryKey: true, autoIncrement: true },  
 start\_time: {  
 type: DataTypes.*DATE*,  
 allowNull: false,  
 get() {  
 return formatDateUTC3(this.getDataValue('start\_time'));  
 }  
 },  
 end\_time: {  
 type: DataTypes.*DATE*,  
 allowNull: false,  
 get() {  
 return formatDateUTC3(this.getDataValue('end\_time'));  
 }  
 },  
 status: { type: DataTypes.*ENUM*('pending','approved','rejected'), defaultValue: 'pending' },  
 user\_id: { type: DataTypes.*INTEGER*, allowNull: false },  
 parking\_slot\_id: { type: DataTypes.*INTEGER*, allowNull: false }  
}, {  
 timestamps: false  
});  
  
  
const Log = *sequelize*.define('log', {  
 id: { type: DataTypes.*INTEGER*, primaryKey: true, autoIncrement: true },  
 user\_id: { type: DataTypes.*INTEGER*, allowNull: false },  
 action: { type: DataTypes.*STRING*, allowNull: false },   
 createdAt: {  
 type: DataTypes.*DATE*,  
 defaultValue: DataTypes.NOW,  
 field: 'created\_at',  
 get() {  
 return formatDateUTC3(this.getDataValue('createdAt'));  
 }  
 },  
 updatedAt: {  
 type: DataTypes.*DATE*,  
 field: 'updated\_at',  
 get() {  
 return formatDateUTC3(this.getDataValue('updatedAt'));  
 }  
 }  
}, {  
 tableName: 'logs',  
 timestamps: true  
});  
Tariff.*hasMany*(Zone, { foreignKey: { name: 'tariff\_id', allowNull: false }, onDelete: 'RESTRICT', onUpdate: 'CASCADE' });  
Zone.*belongsTo*(Tariff, { foreignKey: { name: 'tariff\_id', allowNull: false }, onDelete: 'RESTRICT', onUpdate: 'CASCADE' });  
Zone.*hasMany*(ParkingSlot,{ foreignKey: { name: 'zone\_id', allowNull: false }, onDelete: 'RESTRICT', onUpdate: 'CASCADE' });  
ParkingSlot.*belongsTo*(Zone,{ foreignKey:{ name:'zone\_id', allowNull:false }, onDelete:'RESTRICT', onUpdate:'CASCADE' });  
User.*hasMany*(Booking, { foreignKey:{ name:'user\_id', allowNull:false }, onDelete:'RESTRICT', onUpdate:'CASCADE' });  
Booking.*belongsTo*(User, { foreignKey:{ name:'user\_id', allowNull:false }, onDelete:'RESTRICT', onUpdate:'CASCADE' });  
ParkingSlot.*hasMany*(Booking, { foreignKey:{ name:'parking\_slot\_id', allowNull:false }, onDelete:'RESTRICT', onUpdate:'CASCADE' });  
Booking.*belongsTo*(ParkingSlot,{ foreignKey:{ name:'parking\_slot\_id', allowNull:false }, onDelete:'RESTRICT', onUpdate:'CASCADE' });  
User.*hasMany*(Car, { foreignKey:{ name:'user\_id', allowNull:false }, onDelete:'RESTRICT', onUpdate:'CASCADE' });  
Car.*belongsTo*(User, { foreignKey:{ name:'user\_id', allowNull:false }, onDelete:'RESTRICT', onUpdate:'CASCADE' });  
ParkingSlot.*hasOne*(Car, { foreignKey:{ name:'parking\_slot\_id', allowNull:false }, onDelete:'RESTRICT', onUpdate:'CASCADE' });  
Car.*belongsTo*(ParkingSlot,{ foreignKey:{ name:'parking\_slot\_id', allowNull:false }, onDelete:'RESTRICT', onUpdate:'CASCADE' });  
User.*hasMany*(Log, { foreignKey:{ name:'user\_id', allowNull:false }, onDelete:'CASCADE', hooks:true, onUpdate:'CASCADE' });  
Log.*belongsTo*(User, { foreignKey:{ name:'user\_id', allowNull:false }, onDelete:'CASCADE', onUpdate:'CASCADE' });  
*module*.exports = { User, Tariff, Zone, ParkingSlot, Car, Booking, Log };

ПРИЛОЖЕНИЕ 16. Файл server/routes/Auth.js

const express = *require*('express');  
const AuthController = *require*('../controllers/AuthController');  
const { AuthService } = *require*('../services/AuthService');  
const *router* = express.Router();  
const authService = new AuthService();  
const authController = new AuthController(authService);  
*router*.post('/register', authController.register);  
*router*.post('/login', authController.login);  
*module*.exports = *router*;

ПРИЛОЖЕНИЕ 17. Файл server/routes/Booking.js

const express = *require*('express');  
const { authMiddleware, roleCheck } = *require*('../middlewares/AuthMiddleware');  
const BookingController = *require*('../controllers/BookingController');  
const { BookingService } = *require*('../services/BookingService');  
const *router* = express.Router();  
const service = new BookingService();  
const controller = new BookingController(service);  
*router*.use(authMiddleware);  
*router*.post('/', controller.createBooking);  
*router*.get('/my', controller.getMyBookings);  
*router*.get('/pending', roleCheck(['employee','admin']), controller.getPending);  
*router*.post('/approve/:id', roleCheck(['employee','admin']), controller.approveBooking);  
*router*.post('/reject/:id', roleCheck(['employee','admin']), controller.rejectBooking);  
*router*.post('/cancel/:id', controller.cancelBooking);  
*router*.get('/', roleCheck(['employee','admin']), controller.getAll);  
*module*.exports = *router*;

ПРИЛОЖЕНИЕ 18. Файл server/routes/Cars.js

const express = *require*('express');  
const { authMiddleware, roleCheck } = *require*('../middlewares/authMiddleware');  
const CarController = *require*('../controllers/CarController');  
const *router* = express.Router();  
const { CarService } = *require*('../services/CarService');  
const carService = new CarService();  
const carController = new CarController(carService);  
*router*.use(roleCheck(['admin','employee','user']));  
*router*.get('/', carController.getAll);  
*router*.get('/active', carController.getActive);  
*router*.get('/filter', carController.filter);  
*router*.post('/', roleCheck(['admin','employee']), carController.create);  
*router*.post('/release/:id', roleCheck(['admin','employee']), carController.release);  
*router*.put('/:id', roleCheck(['admin','employee']), carController.update);  
*router*.delete('/:id', roleCheck(['admin','employee']), carController.delete);  
*module*.exports = *router*;

ПРИЛОЖЕНИЕ 19. Файл server/routes/Logs.js

const express = *require*('express');  
const { authMiddleware, roleCheck } = *require*('../middlewares/authMiddleware');  
const { LogService } = *require*('../services/LogService');  
const LogController = *require*('../controllers/LogController');  
const *router* = express.Router();  
const logService = new LogService();  
const logController = new LogController(logService);  
*router*.use(roleCheck(['admin']));  
*router*.get('/', logController.getAll);  
*router*.get('/user/:userId', logController.getByUser);  
*module*.exports = *router*;

ПРИЛОЖЕНИЕ 20. Файл server/routes/Slots.js

const express = *require*('express');  
const ParkingSlotController = *require*('../controllers/ParkingSlotController');  
const { ParkingSlotService } = *require*('../services/ParkingSlotService');  
const { roleCheck, authMiddleware } = *require*('../middlewares/AuthMiddleware');  
const *router* = express.Router();  
*router*.use(authMiddleware);   
const service = new ParkingSlotService();  
const controller = new ParkingSlotController(service);  
*router*.get('/free', controller.getFree);  
*router*.get('/occupied', controller.getOccupied);  
*router*.get('/zone/:zoneId', controller.getByZone);  
*router*.get('/availability', controller.checkAvailability);  
*router*.use(roleCheck(['employee','admin']));  
*router*.get('/', controller.getAll);  
*router*.post('/', controller.create);  
*router*.put('/:id', controller.update);  
*router*.delete('/:id', controller.delete);  
*module*.exports = *router*;

ПРИЛОЖЕНИЕ 21. Файл server/routes/Tariffs.js

const express = *require*('express');  
const { authMiddleware, roleCheck } = *require*('../middlewares/authMiddleware');  
const TariffController = *require*('../controllers/TariffController');  
const *router* = express.Router();  
const { TariffService } = *require*('../services/TariffService');  
const tariffService = new TariffService();  
const tariffController = new TariffController(tariffService);  
*router*.use(roleCheck(['admin']));  
*router*.get('/', tariffController.getAll);  
*router*.get('/:id', tariffController.getById);  
*router*.post('/', tariffController.create);  
*router*.put('/:id', tariffController.update);  
*router*.delete('/:id', tariffController.delete);  
*module*.exports = *router*;

ПРИЛОЖЕНИЕ 22. Файл server/routes/Users.js

const express = *require*('express');  
const { roleCheck } = *require*('../middlewares/AuthMiddleware');  
const { UserService } = *require*('../services/UserService');  
const UserController = *require*('../controllers/UserController');  
const *router* = express.Router();  
const userService = new UserService();  
const userController = new UserController(userService);  
*router*.use(roleCheck(['admin']));  
*router*.post('/', userController.create);  
*router*.get('/', userController.getAll);  
*router*.get('/:id', userController.getById);  
*router*.put('/:id', userController.update);  
*router*.delete('/:id',userController.delete);  
*module*.exports = *router*;

ПРИЛОЖЕНИЕ 23. Файл server/routes/Zones.js

const express = *require*('express');  
const ZoneController = *require*('../controllers/ZoneController');  
const { ZoneService }= *require*('../services/ZoneService');  
const { roleCheck } = *require*('../middlewares/AuthMiddleware');  
const *router* = express.Router();  
const service = new ZoneService();  
const controller = new ZoneController(service);  
*router*.get('/', controller.getAll);  
*router*.use(roleCheck(['admin']));  
*router*.post('/', controller.create);  
*router*.put('/:id', roleCheck(['admin']), controller.update);  
*router*.delete('/:id', controller.delete);  
*module*.exports = *router*;

ПРИЛОЖЕНИЕ 24. Файл server/services/AuthService.js

const *bcrypt* = *require*('bcrypt');  
const *jwt* = *require*('jsonwebtoken');  
const { User } = *require*('../models/Models');  
*require*('dotenv').config();  
class AuthService {  
 constructor() {  
 this.userModel = User;  
 this.jwtSecret = *process*.env.JWT\_SECRET;  
 this.jwtExpiresIn= *process*.env.JWT\_EXPIRES\_IN;  
 }  
 async register({ username, password, fullname, email }) {  
 if (!username || !password || !email) {  
 const err = new Error('username, password и email обязательны');  
 err.status = 400;  
 throw err;  
 }  
 if (await this.userModel.*findOne*({ where: { username } })) {  
 const err = new Error('Пользователь с таким именем уже существует');  
 err.status = 409;  
 throw err;  
 }  
 const hash = await *bcrypt*.hash(password, 10);  
 const user = await this.userModel.*create*({  
 username,  
 password: hash,  
 fullname,  
 email,  
 role: 'user'  
 });  
 const token = *jwt*.sign(  
 { id: user.id, role: user.role },  
 this.jwtSecret,  
 { expiresIn: this.jwtExpiresIn }  
 );  
 return { token };  
 }  
 async login({ username, password }) {  
 if (!username || !password) {  
 const err = new Error('username и password обязательны');  
 err.status = 400;  
 throw err;  
 }  
 const user = await this.userModel.*findOne*({ where: { username } });  
 if (!user || !(await *bcrypt*.compare(password, user.password))) {  
 const err = new Error('Неверные учётные данные');  
 err.status = 401;  
 throw err;  
 }  
 const token = *jwt*.sign(  
 { id: user.id, role: user.role },  
 this.jwtSecret,  
 { expiresIn: this.jwtExpiresIn }  
 );  
 return { token };  
 }  
}  
*module*.exports.AuthService = AuthService;

ПРИЛОЖЕНИЕ 25. Файл server/services/BookingService.js

const *sequelize* = *require*('../db');  
const {*Op*} = *require*('sequelize');  
const {Booking, ParkingSlot, Log} = *require*('../models/Models');  
class BookingService {  
 async getAll({ id }) {  
 const where = {};  
 if (id) where.id = id;  
 return Booking.*findAll*({ where, include: [ParkingSlot] });  
 }  
 async getPending() {  
 return Booking.*findAll*({  
 where: {status: 'pending'},  
 include: [ParkingSlot]  
 });  
 }  
 async getByUser(userId) {  
 return Booking.*findAll*({where: {user\_id: userId}});  
 }  
 async checkAvailability(parking\_slot\_id, start\_time, end\_time) {  
 const overlap = await Booking.*findOne*({  
 where: {  
 parking\_slot\_id,  
 status: 'approved',  
 [*Op*.or]: [  
 {start\_time: {[*Op*.lt]: end\_time}, end\_time: {[*Op*.gt]: start\_time}}  
 ]  
 }  
 });  
 return !overlap;  
 }  
 async createBooking({userId, parking\_slot\_id, start\_time, end\_time}) {  
 const startDt = new *Date*(start\_time);  
 const endDt = new *Date*(end\_time);  
 if (isNaN(startDt) || isNaN(endDt)) {  
 const e = new Error('Неверный формат даты');  
 e.status = 400;  
 throw e;  
 }  
 if (endDt <= startDt) {  
 const e = new Error('Дата окончания должна быть позже даты начала');  
 e.status = 400;  
 throw e;  
 }  
 const slot = await ParkingSlot.*findByPk*(parking\_slot\_id);  
 if (!slot) {  
 const e = new Error('Слот не найден');  
 e.status = 404;  
 throw e;  
 }  
 if (slot.status !== 'free') {  
 const e = new Error('Слот сейчас недоступен');  
 e.status = 409;  
 throw e;  
 }  
 const overlapCount = await Booking.*count*({  
 where: {  
 parking\_slot\_id,  
 status: {[*Op*.in]: ['approved']},  
 [*Op*.or]: [  
 {start\_time: {[*Op*.lt]: endDt}, end\_time: {[*Op*.gt]: startDt}}  
 ]  
 }  
 });  
 if (overlapCount > 0) {  
 const e = new Error('Слот занят в указанное время');  
 e.status = 409;  
 throw e;  
 }  
 const booking = await Booking.*create*({  
 user\_id: userId,  
 parking\_slot\_id,  
 start\_time: startDt,  
 end\_time: endDt,  
 status: 'pending'  
 });  
 return booking;  
 }  
 async approveBooking(id, userId) {  
 return *sequelize*.transaction(async t => {  
 const booking = await Booking.*findByPk*(id, {transaction: t});  
 if (!booking) throw new Error('Booking not found');  
 await booking.update({status: 'approved'}, {transaction: t});  
 await Log.*create*(  
 {user\_id: userId, action: `Approved booking ${id}`},  
 {transaction: t}  
 );  
 return booking;  
 });  
 }  
 async rejectBooking(id, userId) {  
 return *sequelize*.transaction(async t => {  
 const booking = await Booking.*findByPk*(id, {transaction: t});  
 if (!booking) throw new Error('Booking not found');  
 await booking.update({status: 'rejected'}, {transaction: t});  
 await ParkingSlot.*update*(  
 {status: 'free'},  
 {where: {id: booking.parking\_slot\_id}, transaction: t}  
 );  
 await Log.*create*(  
 {user\_id: userId, action: `Rejected booking ${id}`},  
 {transaction: t}  
 );  
 return booking;  
 });  
 }  
 async cancelBooking(id, userId) {  
 return *sequelize*.transaction(async t => {  
 const booking = await Booking.*findByPk*(id, {transaction: t});  
 if (!booking) throw new Error('Booking not found');  
 if (booking.status !== 'pending') {  
 const e = new Error('Только заявки в статусе pending можно отменять');  
 e.status = 400;  
 throw e;  
 }  
 await booking.update({status: 'rejected'}, {transaction: t});  
 await ParkingSlot.*update*(  
 {status: 'free'},  
 {where: {id: booking.parking\_slot\_id}, transaction: t}  
 );  
 await Log.*create*(  
 {user\_id: userId, action: `Canceled booking ${id}`},  
 {transaction: t}  
 );  
 return booking;  
 });  
 }  
}  
*module*.exports.BookingService = BookingService;

ПРИЛОЖЕНИЕ 26. Файл server/services/CarService.js

const *sequelize* = *require*('../db');  
const { *Op* } = *require*('sequelize');  
const { Car, ParkingSlot, Log } = *require*('../models/Models');  
class CarService {  
 async findAll({ license\_plate }) {  
 const where = {};  
 if (license\_plate) {  
 where.license\_plate = { [*Op*.iLike]: `%${license\_plate}%` };  
 }  
 return Car.*findAll*({ where });  
 }  
 async getActiveCars() {  
 return Car.*findAll*({ where: { exit\_time: null }, include: [ParkingSlot] });  
 }  
 async filter(criteria) {  
 const where = {};  
 if (criteria.license\_plate) where.license\_plate = criteria.license\_plate;  
 if (criteria.model) where.model = criteria.model;  
 return Car.*findAll*({  
 include: [{  
 model: ParkingSlot,  
 where: criteria.slot\_status ? { status: criteria.slot\_status } : {},  
 }],  
 where,  
 });  
 }  
 async create(data, userId) {  
 return *sequelize*.transaction(async t => {  
 const slot = await ParkingSlot.*findByPk*(data.parking\_slot\_id, { transaction: t });  
 if (!slot || !['free'].includes(slot.status)) {  
 const e = new Error('Slot is not available');  
 e.status = 400;  
 throw e;  
 }  
 const carData = {  
 ...data,  
 user\_id: userId,  
 };  
 const car = await Car.*create*(carData, { transaction: t });  
 await slot.update({ status: 'occupied' }, { transaction: t });  
 await Log.*create*({  
 user\_id: userId,  
 action: `Created car ${car.id}`  
 }, { transaction: t });  
 return car;  
 });  
 }  
 async release(id, userId) {  
 return *sequelize*.transaction(async t => {  
 const car = await Car.*findByPk*(id, { transaction: t });  
 if (!car) throw new Error('Car not found');  
 if (car.exit\_time) throw new Error('Car already released');  
 car.exit\_time = new *Date*();  
 await car.save({ transaction: t });  
 const slot = await ParkingSlot.*findByPk*(car.parking\_slot\_id, { transaction: t });  
 if (slot) await slot.update({ status: 'free' }, { transaction: t });  
 await Log.*create*({ user\_id: userId, action: `Released car ${id}` }, { transaction: t });  
 return car;  
 });  
 }  
 async update(id, data, userId) {  
 const car = await Car.*findByPk*(id);  
 if (!car) throw new Error('Car not found');  
 return *sequelize*.transaction(async t => {  
 await car.update(data, { transaction: t });  
 await Log.*create*({ user\_id: userId, action: `Updated car ${id}` }, { transaction: t });  
 return car;  
 });  
 }  
 async delete(id, userId) {  
 return *sequelize*.transaction(async t => {  
 const car = await Car.*findByPk*(id, { transaction: t });  
 if (!car) throw new Error('Car not found');  
 const slot = await ParkingSlot.*findByPk*(car.parking\_slot\_id, { transaction: t });  
 if (slot) await slot.update({ status: 'free' }, { transaction: t });  
 await car.destroy({ transaction: t });  
 await Log.*create*({ user\_id: userId, action: `Deleted car ${id}` }, { transaction: t });  
 return car;  
 });  
 }  
}  
*module*.exports.CarService = CarService;

ПРИЛОЖЕНИЕ 27. Файл server/services/LogService.js

const { Log, User } = *require*('../models/Models');  
class LogService {  
 async getAll() { return Log.*findAll*({ include: [User] }); }  
 async getByUser(userId) { return Log.*findAll*({ where: { user\_id: userId } }); }  
 async create(userId, action) { return Log.*create*({ user\_id: userId, action }); }  
}  
*module*.exports.LogService = LogService;

ПРИЛОЖЕНИЕ 28. Файл server/services/ParkingSlotService.js

const {ParkingSlot, Log, Zone, Booking, Tariff} = *require*('../models/Models');  
const {*Op*} = *require*("sequelize");  
class ParkingSlotService {  
 async getAll({ slot\_number }) {  
 const where = {};  
 if (slot\_number) {  
 where.slot\_number = { [*Op*.eq]: slot\_number };  
 }  
 return ParkingSlot.*findAll*({ where });  
 }  
 async getFreeSlots() {  
 return ParkingSlot.*findAll*({where: {status: 'free'}});  
 }  
 async getOccupiedSlots() {  
 return ParkingSlot.*findAll*({where: {status: 'occupied'}});  
 }  
 async getSlotsByZone(zoneId) {  
 return ParkingSlot.*findAll*({where: {zone\_id: zoneId}});  
 }  
 async create(data, userId) {  
 const slot = await ParkingSlot.*create*(data);  
 await Log.*create*({user\_id: userId, action: `Created slot ${slot.id}`});  
 return slot;  
 }  
 async update(id, data, userId) {  
 const slot = await ParkingSlot.*findByPk*(id);  
 if (!slot) throw new Error('Slot not found');  
 await slot.update(data);  
 await Log.*create*({user\_id: userId, action: `Updated slot ${id}`});  
 return slot;  
 }  
 async getZonesAvailability(start, end) {  
 const startDt = new *Date*(start);  
 const endDt = new *Date*(end);  
 const zones = await Zone.*findAll*({  
 include: [  
 {model: Tariff, attributes: ['price\_per\_hour']},  
 {  
 model: ParkingSlot,  
 attributes: ['id', 'status'],  
 include: [{  
 model: Booking,  
 where: {  
 status: 'approved',   
 start\_time: {[*Op*.lt]: endDt},  
 end\_time: {[*Op*.gt]: startDt}  
 },  
 required: false,  
 attributes: ['id']  
 }]  
 }  
 ]  
 });  
 return zones.map(z => {  
 const slots = z.parking\_slots;  
 const total = slots.length;  
 const freeSlots = slots.filter(s =>  
 s.status === 'free' && s.bookings.length === 0  
 );  
 return {  
 id: z.id,  
 name: z.name,  
 pricePerHour: z.tariff.price\_per\_hour,   
 totalCount: total,  
 freeCount: freeSlots.length,  
 freeSlotIds: freeSlots.map(s => s.id)   
 };  
 });  
 }  
 async delete(id, userId) {  
 const slot = await ParkingSlot.*findByPk*(id);  
 if (!slot) throw new Error('Slot not found');  
 await slot.destroy();  
 await Log.*create*({user\_id: userId, action: `Deleted slot ${id}`});  
 return slot;  
 }  
}  
*module*.exports.ParkingSlotService = ParkingSlotService;

ПРИЛОЖЕНИЕ 29. Файл server/services/TariffService.js

const { Tariff, Log } = *require*('../models/Models');  
class TariffService {  
 async getAll({ name }) {  
 const where = {};  
 if (name) {  
 where.name = { [Op.iLike]: `%${name}%` };  
 }  
 return Tariff.*findAll*({ where });  
 }  
 async getById(id) { return Tariff.*findByPk*(id); }  
 async create(data, userId) {  
 const tariff = await Tariff.*create*({ name: data.name, price\_per\_hour: data.price\_per\_hour });  
 await Log.*create*({ user\_id: userId, action: `Created tariff ${tariff.id}` });  
 return tariff;  
 }  
 async update(id, data, userId) {  
 const tariff = await Tariff.*findByPk*(id);  
 if (!tariff) throw new Error('Tariff not found');  
 await tariff.update(data);  
 await Log.*create*({ user\_id: userId, action: `Updated tariff ${id}` });  
 return tariff;  
 }  
 async delete(id, userId) {  
 const tariff = await Tariff.*findByPk*(id);  
 if (!tariff) throw new Error('Tariff not found');  
 await tariff.destroy();  
 await Log.*create*({ user\_id: userId, action: `Deleted tariff ${id}` });  
 return tariff;  
 }  
}  
*module*.exports.TariffService = TariffService;

ПРИЛОЖЕНИЕ 30. Файл server/services/UserService.js

const { User, Log } = *require*('../models/Models');  
const *bcrypt* = *require*('bcrypt');  
class UserService {  
 async getAll({ id }) {  
 const where = {};  
 if (id) where.id = id;  
 return User.*findAll*({ where });  
 }  
 async getById(id) {  
 return User.*findByPk*(id, { attributes: { exclude: ['password'] } });  
 }  
 async update(id, data, userId) {  
 const user = await User.*findByPk*(id);  
 if (!user) throw new Error('User not found');  
 delete data.password;  
 delete data.role;  
 await user.update(data);  
 await Log.*create*({ user\_id: userId, action: `Updated user ${id}` });  
 return user;  
 }  
 async delete(id, userId) {  
 const user = await User.*findByPk*(id);  
 if (!user) throw new Error('User not found');  
 await user.destroy();  
 await Log.*create*({ user\_id: userId, action: `Deleted user ${id}` });  
 return user;  
 }  
 async create(data, creatorId) {  
 const { username, password, fullname, email, role } = data;  
  
 if (!username || !password || !email || !role) {  
 const err = new Error('username, password, email и role обязательны');  
 err.status = 400;  
 throw err;  
 }  
 if (await User.*findOne*({ where: { username } })) {  
 const err = new Error('Пользователь с таким именем уже существует');  
 err.status = 409;  
 throw err;  
 }  
 const hash = await *bcrypt*.hash(password, 10);  
 const user = await User.*create*({  
 username,  
 password: hash,  
 fullname,  
 email,  
 role   
 });  
 await Log.*create*({  
 user\_id: creatorId,  
 action: `Created user ${user.id} with role ${role}`  
 });  
 const { password: \_, ...safe } = user.get({ plain: true });  
 return safe;  
 }  
}  
*module*.exports.UserService = UserService;

ПРИЛОЖЕНИЕ 31. Файл server/services/ZoneService.js

const { Zone, Tariff, Log } = *require*('../models/Models');  
const {*Op*} = *require*("sequelize");  
class ZoneService {  
 async getAll({ name }) {  
 const where = {};  
 if (name) {  
 where.name = { [*Op*.iLike]: `%${name}%` };  
 }  
 return Zone.*findAll*({ where });  
 }  
 async getById(id) { return Zone.*findByPk*(id, { include: [Tariff] }); }  
 async create(data, userId) {  
 const zone = await Zone.*create*({ name: data.name, tariff\_id: data.tariff\_id });  
 await Log.*create*({ user\_id: userId, action: `Created zone ${zone.id}` });  
 return zone;  
 }  
 async update(id, data, userId) {  
 const zone = await Zone.*findByPk*(id);  
 if (!zone) throw new Error('Zone not found');  
 await zone.update(data);  
 await Log.*create*({ user\_id: userId, action: `Updated zone ${id}` });  
 return zone;  
 }  
 async delete(id, userId) {  
 const zone = await Zone.*findByPk*(id);  
 if (!zone) throw new Error('Zone not found');  
 await zone.destroy();  
 await Log.*create*({ user\_id: userId, action: `Deleted zone ${id}` });  
 return zone;  
 }  
}  
*module*.exports.ZoneService = ZoneService;

ПРИЛОЖЕНИЕ 32. Файл test/auth.test.js

*process*.env.NODE\_ENV = 'test';  
const request = *require*('supertest');  
const { expect } = *require*('chai');  
const *app* = *require*('../server/app');  
const initDB = *require*('../server/models/Init');  
*describe*('Auth API', function() {  
 this.timeout(5000);  
 let userToken;  
 let adminToken;  
 *before*(async () => {  
 await initDB();  
 const regUser = await request(*app*).post('/api/auth/register')  
 .send({ username: 'testuser', password: 'Pass1234', fullname: 'Test', email: 'test@example.com' });  
 userToken = regUser.body.token;  
 const regAdmin = await request(*app*).post('/api/auth/register')  
 .send({ username: 'admin', password: 'Admin123', fullname: 'Admin', email: 'admin@example.com' });  
 const { User } = *require*('../server/models/Models');  
 const admin = await User.*findOne*({ where: { username: 'admin' } });  
 await admin.update({ role: 'admin' });  
 const loginAdmin = await request(*app*).post('/api/auth/login')  
 .send({ username: 'admin', password: 'Admin123' });  
 adminToken = loginAdmin.body.token;  
 });  
 *it*('POST /api/auth/register returns token', () =>  
 request(*app*)  
 .post('/api/auth/register')  
 .send({ username: 'newuser', password: 'NewPass1', fullname: 'New', email: 'new@example.com' })  
 .expect(201)  
 .then(res => expect(res.body).to.have.property('token'))  
 );  
 *it*('POST /api/auth/login returns token', () =>  
 request(*app*)  
 .post('/api/auth/login')  
 .send({ username: 'testuser', password: 'Pass1234' })  
 .expect(200)  
 .then(res => expect(res.body).to.have.property('token'))  
 );  
 *it*('GET /api/users without token returns 401', () =>  
 request(*app*).get('/api/users').expect(401)  
 );  
 *it*('GET /api/users with user token returns 403', () =>  
 request(*app*)  
 .get('/api/users')  
 .set('Authorization', `Bearer ${userToken}`)  
 .expect(403)  
 );  
});

ПРИЛОЖЕНИЕ 33. Файл test/booking.test.js

*process*.env.NODE\_ENV = 'test';  
const request = *require*('supertest');  
const { expect } = *require*('chai');  
const *app* = *require*('../server/app');  
const initDB = *require*('../server/models/Init');  
*describe*('Booking API', function() {  
 this.timeout(5000);  
 let userToken;  
 let adminToken;  
 let slotId;  
 let booking;  
 *before*(async () => {  
 await initDB();  
 const resUser = await request(*app*)  
 .post('/api/auth/register')  
 .send({  
 username: 'uBook',  
 password: 'Book123',  
 fullname: 'BookUser',  
 email: 'book@example.com'  
 })  
 .expect(201);  
 userToken = resUser.body.token;  
 const resAdm = await request(*app*)  
 .post('/api/auth/register')  
 .send({  
 username: 'admBook',  
 password: 'Admin123',  
 fullname: 'AdmBook',  
 email: 'admbook@example.com'  
 })  
 .expect(201);  
 const { User } = *require*('../server/models/Models');  
 const adm = await User.*findOne*({ where: { username: 'admBook' } });  
 await adm.update({ role: 'admin' });  
 const resLoginAdm = await request(*app*)  
 .post('/api/auth/login')  
 .send({ username: 'admBook', password: 'Admin123' })  
 .expect(200);  
 adminToken = resLoginAdm.body.token;  
 const resTariff = await request(*app*)  
 .post('/api/tariffs')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ name: 'BookTar', price\_per\_hour: 6 })  
 .expect(201);  
 const resZone = await request(*app*)  
 .post('/api/zones')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ name: 'BookZone', tariff\_id: resTariff.body.id })  
 .expect(201);  
 const resSlot = await request(*app*)  
 .post('/api/slots')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ slot\_number: 301, zone\_id: resZone.body.id })  
 .expect(201);  
 slotId = resSlot.body.id;  
 const resBooking = await request(*app*)  
 .post('/api/bookings')  
 .set('Authorization', `Bearer ${userToken}`)  
 .send({  
 parking\_slot\_id: slotId,  
 start\_time: new *Date*().toISOString(),  
 end\_time: new *Date*(Date.now() + 3600000).toISOString()  
 })  
 .expect(201);  
 booking = resBooking.body;  
 });  
 *it*('GET /api/bookings/my returns array', () =>  
 request(*app*)  
 .get('/api/bookings/my')  
 .set('Authorization', `Bearer ${userToken}`)  
 .expect(200)  
 .then(res => expect(res.body).to.be.an('array'))  
 );  
 *it*('GET /api/bookings/pending returns array', () =>  
 request(*app*)  
 .get('/api/bookings/pending')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 .then(res => expect(res.body).to.be.an('array'))  
 );  
 *it*('POST /api/bookings/cancel/:id cancels booking', () =>  
 request(*app*)  
 .post(`/api/bookings/cancel/${booking.id}`)  
 .set('Authorization', `Bearer ${userToken}`)  
 .expect(200)  
 );  
 *it*('POST /api/bookings/approve/:id approves booking', () =>  
 request(*app*)  
 .post(`/api/bookings/approve/${booking.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 );  
 *it*('POST /api/bookings/reject/:id rejects booking', async () => {  
   
 const res2 = await request(*app*)  
 .post('/api/bookings')  
 .set('Authorization', `Bearer ${userToken}`)  
 .send({  
 parking\_slot\_id: slotId,  
 start\_time: new *Date*().toISOString(),  
 end\_time: new *Date*(Date.now() + 3600000).toISOString()  
 })  
 .expect(201);  
 const toReject = res2.body;  
 await request(*app*)  
 .post(`/api/bookings/reject/${toReject.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200);  
 });  
 *it*('GET /api/bookings returns array for admin', () =>  
 request(*app*)  
 .get('/api/bookings')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 .then(res => expect(res.body).to.be.an('array'))  
 );  
});

ПРИЛОЖЕНИЕ 34. Файл test/cars.test.js

*process*.env.NODE\_ENV = 'test';  
const requestCars = *require*('supertest');  
const { expect: expCars } = *require*('chai');  
const *appCars* = *require*('../server/app');  
const initDBCars = *require*('../server/models/Init');  
*describe*('Car API', function() {  
 this.timeout(5000);  
 let adminToken;  
 let slot;  
 let car;  
 *before*(async () => {  
 await initDBCars();  
 await requestCars(*appCars*).post('/api/auth/register')  
 .send({ username: 'adm6', password: 'Admin123', fullname: 'Adm6', email: 'adm6@example.com' });  
 const { User } = *require*('../server/models/Models');  
 const adm = await User.*findOne*({ where: { username: 'adm6' } });  
 await adm.update({ role: 'admin' });  
 const login = await requestCars(*appCars*).post('/api/auth/login')  
 .send({ username: 'adm6', password: 'Admin123' });  
 adminToken = login.body.token;  
 const tTariff = await requestCars(*appCars*)  
 .post('/api/tariffs')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ name: 'CarTar', price\_per\_hour: 5 });  
 const tZone = await requestCars(*appCars*)  
 .post('/api/zones')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ name: 'CarZone', tariff\_id: tTariff.body.id });  
 const sRes = await requestCars(*appCars*)  
 .post('/api/slots')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ slot\_number: 201, zone\_id: tZone.body.id });  
 slot = sRes.body;  
 });  
 *it*('POST /api/cars creates car', () =>  
 requestCars(*appCars*)  
 .post('/api/cars')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ license\_plate: 'ABC123', model: 'TestModel', parking\_slot\_id: slot.id })  
 .expect(201)  
 .then(res => { car = res.body; expCars(car).to.include({ license\_plate: 'ABC123' }); })  
 );  
 *it*('GET /api/cars returns array', () =>  
 requestCars(*appCars*)  
 .get('/api/cars')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 .then(res => expCars(res.body).to.be.an('array'))  
 );  
 *it*('GET /api/cars/active returns array', () =>  
 requestCars(*appCars*)  
 .get('/api/cars/active')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 .then(res => expCars(res.body).to.be.an('array'))  
 );  
 *it*('GET /api/cars/filter returns array', () =>  
 requestCars(*appCars*)  
 .get(`/api/cars/filter?license\_plate=ABC123`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 .then(res => expCars(res.body).to.be.an('array'))  
 );  
 *it*('PUT /api/cars/:id updates car', () =>  
 requestCars(*appCars*)  
 .put(`/api/cars/${car.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ model: 'UpdatedModel' })  
 .expect(200)  
 .then(res => expCars(res.body).to.include({ model: 'UpdatedModel' }))  
 );  
 *it*('POST /api/cars/release/:id releases car', () =>  
 requestCars(*appCars*)  
 .post(`/api/cars/release/${car.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 );  
 *it*('DELETE /api/cars/:id deletes car', () =>  
 requestCars(*appCars*)  
 .delete(`/api/cars/${car.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(204)  
 );  
});

ПРИЛОЖЕНИЕ 35. Файл test/slots.test.js

*process*.env.NODE\_ENV = 'test';  
const requestSlots = *require*('supertest');  
const { expect: expSlots } = *require*('chai');  
const *appSlots* = *require*('../server/app');  
const initDBSlots = *require*('../server/models/Init');  
*describe*('ParkingSlot API', function() {  
 this.timeout(5000);  
 let adminToken;  
 let tariff;  
 let zone;  
 let slot;  
 *before*(async () => {  
 await initDBSlots();  
 await requestSlots(*appSlots*).post('/api/auth/register')  
 .send({ username: 'adm5', password: 'Admin123', fullname: 'Adm5', email: 'adm5@example.com' });  
 const { User } = *require*('../server/models/Models');  
 const adm = await User.*findOne*({ where: { username: 'adm5' } });  
 await adm.update({ role: 'admin' });  
 const login = await requestSlots(*appSlots*).post('/api/auth/login')  
 .send({ username: 'adm5', password: 'Admin123' });  
 adminToken = login.body.token;  
 const tRes = await requestSlots(*appSlots*)  
 .post('/api/tariffs')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ name: 'TSlot', price\_per\_hour: 4 });  
 tariff = tRes.body;  
 const zRes = await requestSlots(*appSlots*)  
 .post('/api/zones')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ name: 'ZSlot', tariff\_id: tariff.id });  
 zone = zRes.body;  
 });  
 *it*('POST /api/slots creates slot', () =>  
 requestSlots(*appSlots*)  
 .post('/api/slots')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ slot\_number: 101, zone\_id: zone.id })  
 .expect(201)  
 .then(res => { slot = res.body; expSlots(slot).to.include({ slot\_number: 101 }); })  
 );  
 *it*('GET /api/slots returns array', () =>  
 requestSlots(*appSlots*)  
 .get('/api/slots')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 .then(res => expSlots(res.body).to.be.an('array'))  
 );  
 *it*('GET /api/slots/free returns array', () =>  
 requestSlots(*appSlots*)  
 .get('/api/slots/free')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 .then(res => expSlots(res.body).to.be.an('array'))  
 );  
 *it*('GET /api/slots/occupied returns array', () =>  
 requestSlots(*appSlots*)  
 .get('/api/slots/occupied')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 .then(res => expSlots(res.body).to.be.an('array'))  
 );  
 *it*('GET /api/slots/zone/:id returns array', () =>  
 requestSlots(*appSlots*)  
 .get(`/api/slots/zone/${zone.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 .then(res => expSlots(res.body).to.be.an('array'))  
 );  
 *it*('PUT /api/slots/:id updates slot', () =>  
 requestSlots(*appSlots*)  
 .put(`/api/slots/${slot.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ slot\_number: 102 })  
 .expect(200)  
 .then(res => expSlots(res.body).to.include({ slot\_number: 102 }))  
 );  
 *it*('DELETE /api/slots/:id deletes slot', () =>  
 requestSlots(*appSlots*)  
 .delete(`/api/slots/${slot.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(204)  
 );  
});

ПРИЛОЖЕНИЕ 36. Файл test/tariff.test.js

*process*.env.NODE\_ENV = 'test';  
const requestTariff = *require*('supertest');  
const { expect: expT } = *require*('chai');  
const *appTariff* = *require*('../server/app');  
const initDBTariff = *require*('../server/models/Init');  
*describe*('Tariff API', function() {  
 this.timeout(5000);  
 let adminToken;  
 let created;  
 *before*(async () => {  
 await initDBTariff();  
 await requestTariff(*appTariff*).post('/api/auth/register')  
 .send({ username: 'adm3', password: 'Admin123', fullname: 'Adm3', email: 'adm3@example.com' });  
 const { User } = *require*('../server/models/Models');  
 const adm = await User.*findOne*({ where: { username: 'adm3' } });  
 await adm.update({ role: 'admin' });  
 const login = await requestTariff(*appTariff*).post('/api/auth/login')  
 .send({ username: 'adm3', password: 'Admin123' });  
 adminToken = login.body.token;  
 });  
 *it*('POST /api/tariffs creates tariff', () =>  
 requestTariff(*appTariff*)  
 .post('/api/tariffs')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ name: 'T1', price\_per\_hour: 2 })  
 .expect(201)  
 .then(res => { created = res.body; expT(created).to.include({ name: 'T1' }); })  
 );  
 *it*('GET /api/tariffs returns array', () =>  
 requestTariff(*appTariff*)  
 .get('/api/tariffs')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 .then(res => expT(res.body).to.be.an('array'))  
 );  
 *it*('DELETE /api/tariffs/:id deletes tariff', () =>  
 requestTariff(*appTariff*)  
 .delete(`/api/tariffs/${created.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(204)  
 );  
});

ПРИЛОЖЕНИЕ 37. Файл test/user.test.js

*process*.env.NODE\_ENV = 'test';  
const requestUser = *require*('supertest');  
const { expect: expUser } = *require*('chai');  
const *appUser* = *require*('../server/app');  
const initDBUser = *require*('../server/models/Init');  
*describe*('User API', function() {  
 this.timeout(5000);  
 let adminToken;  
 let newUser;  
 *before*(async () => {  
 await initDBUser();  
 await requestUser(*appUser*).post('/api/auth/register')  
 .send({ username: 'admin2', password: 'Admin123', fullname: 'Admin2', email: 'admin2@example.com' });  
 const { User } = *require*('../server/models/Models');  
 const adm = await User.*findOne*({ where: { username: 'admin2' } });  
 await adm.update({ role: 'admin' });  
 const login = await requestUser(*appUser*).post('/api/auth/login')  
 .send({ username: 'admin2', password: 'Admin123' });  
 adminToken = login.body.token;  
 });  
 *it*('POST /api/users creates user', () =>  
 requestUser(*appUser*)  
 .post('/api/users')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ username: 'u1', password: 'U1pass', fullname: 'User1', email: 'u1@example.com', role: 'user' })  
 .expect(201)  
 .then(res => { expUser(res.body).to.include({ username: 'u1' }); newUser = res.body; })  
 );  
 *it*('GET /api/users returns array', () =>  
 requestUser(*appUser*)  
 .get('/api/users')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 .then(res => expUser(res.body).to.be.an('array'))  
 );  
 *it*('GET /api/users/:id returns user', () =>  
 requestUser(*appUser*)  
 .get(`/api/users/${newUser.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 );  
 *it*('PUT /api/users/:id updates user', () =>  
 requestUser(*appUser*)  
 .put(`/api/users/${newUser.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ fullname: 'Updated' })  
 .expect(200)  
 .then(res => expUser(res.body).to.include({ fullname: 'Updated' }))  
 );  
 *it*('DELETE /api/users/:id deletes user', () =>  
 requestUser(*appUser*)  
 .delete(`/api/users/${newUser.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(204)  
 );  
});

ПРИЛОЖЕНИЕ 38. Файл test/user.test.js

*process*.env.NODE\_ENV = 'test';  
const requestZone = *require*('supertest');  
const { expect: expZ } = *require*('chai');  
const *appZone* = *require*('../server/app');  
const initDBZone = *require*('../server/models/Init');  
*describe*('Zone API', function() {  
 this.timeout(5000);  
 let adminToken;  
 let tariff;  
 let zone;  
 *before*(async () => {  
 await initDBZone();  
 await requestZone(*appZone*).post('/api/auth/register')  
 .send({ username: 'adm4', password: 'Admin123', fullname: 'Adm4', email: 'adm4@example.com' });  
 const { User } = *require*('../server/models/Models');  
 const adm = await User.*findOne*({ where: { username: 'adm4' } });  
 await adm.update({ role: 'admin' });  
 const login = await requestZone(*appZone*).post('/api/auth/login')  
 .send({ username: 'adm4', password: 'Admin123' });  
 adminToken = login.body.token;  
 const t = await requestZone(*appZone*)  
 .post('/api/tariffs')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ name: 'TarZ', price\_per\_hour: 3 });  
 tariff = t.body;  
 });  
 *it*('POST /api/zones creates zone', () =>  
 requestZone(*appZone*)  
 .post('/api/zones')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({ name: 'Z1', tariff\_id: tariff.id })  
 .expect(201)  
 .then(res => { zone = res.body; expZ(zone).to.include({ name: 'Z1' }); })  
 );  
 *it*('GET /api/zones returns array', () =>  
 requestZone(*appZone*)  
 .get('/api/zones')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(200)  
 .then(res => expZ(res.body).to.be.an('array'))  
 );  
 *it*('DELETE /api/zones/:id deletes zone', () =>  
 requestZone(*appZone*)  
 .delete(`/api/zones/${zone.id}`)  
 .set('Authorization', `Bearer ${adminToken}`)  
 .expect(204)  
 );  
});

ПРИЛОЖЕНИЕ 39. Файл client/public/css/main.css

.hero {  
 color: #fff;  
}  
.hero .form-control {  
 opacity: 0.9;  
}  
  
.how-it-works i {  
 color: #0d6efd;  
}  
  
.container { max-width: 960px; }

ПРИЛОЖЕНИЕ 40. Файл client/public/js/api.js

function api(method, url, data) {  
 return $.ajax({  
 method, url,  
 contentType: 'application/json',  
 data: data?*JSON*.stringify(data):undefined,  
 headers: { Authorization: 'Bearer '+*localStorage*.getItem('token') }  
 });  
}

ПРИЛОЖЕНИЕ 41. Файл client/public/js/auth.js

$(function() {  
 const publicPaths = ['/login', '/register'];  
 const path = *window*.location.pathname;  
 const token = *localStorage*.getItem('token');  
 function parseJwt(t) {  
 const base64Url = t.split('.')[1];  
 let base64 = base64Url.replace(/-/g, '+').replace(/\_/g, '/');  
 const pad = base64.length % 4;  
 if (pad) base64 += '='.repeat(4 - pad);  
 return *JSON*.parse(*window*.atob(base64));  
 }  
 const roleAllowed = {  
 admin: ['/dashboard','/cars','/slots','/tariffs','/zones','/bookings','/users'],  
 employee: ['/dashboard','/cars','/slots','/bookings'],  
 user: ['/dashboard','/bookings'],  
 guest: ['/dashboard','/bookings']  
 };  
 if (!publicPaths.includes(path) && !token) {  
 *window*.location.href = '/login';  
 return;  
 }  
 if (path === '/login' || path === '/register') {  
 $('.navbar').hide();  
 } else {  
 let role = null;  
 try {  
 const payload = parseJwt(token);  
 role = payload.role;  
 } catch (e) {  
 *console*.error('JWT parse error:', e);  
 return *window*.location.href = '/login';  
 }  
 const allowed = roleAllowed[role] || ['/dashboard'];  
 if (!allowed.includes(path)) {  
 return *window*.location.href = '/dashboard';  
 }  
 $('nav.navbar .nav-link').each(function() {  
 const href = $(this).attr('href') || '';  
 if (href === '#' || href === '/logout') return;  
 if (!roleAllowed[role].includes(href)) {  
 $(this).closest('li.nav-item').hide();  
 }  
 });  
 }  
 $('#logout').on('click', e => {  
 e.preventDefault();  
 *localStorage*.removeItem('token');  
 *window*.location.href = '/login';  
 });  
 if (path === '/login' && $('#loginForm').length) {  
 $('#loginForm').on('submit', async function(e) {  
 e.preventDefault();  
 $('#loginError').hide();  
 try {  
 const res = await api('POST', '/api/auth/login', {  
 username: $('#username').val(),  
 password: $('#password').val()  
 });  
 *localStorage*.setItem('token', res.token);  
 *window*.location.href = '/dashboard';  
 } catch {  
 $('#loginError').text('Неправильный логин или пароль').show();  
 }  
 });  
 }  
 if (path === '/register' && $('#registerForm').length) {  
 $('#registerForm').on('submit', async function(e) {  
 e.preventDefault();  
 $('#registerError').hide();  
 try {  
 const res = await api('POST', '/api/auth/register', {  
 username: $('#username').val(),  
 password: $('#password').val(),  
 fullname: $('#fullname').val(),  
 email: $('#email').val()  
 });  
 *localStorage*.setItem('token', res.token);  
 *window*.location.href = '/dashboard';  
 } catch (err) {  
 const msg = err.responseJSON?.message || 'Ошибка регистрации';  
 $('#registerError').text(msg).show();  
 }  
 });  
 }  
});

ПРИЛОЖЕНИЕ 42. Файл client/public/js/bookings.js

$(function() {  
 const $tbody = $('#bookingsTableBody');  
 const $modal = $('#bookingModal');  
 const $form = $('#bookingForm');  
 const $slotSel = $('#parking\_slot\_id');  
 const $addBtn = $('#addBookingBtn');  
 const $search = $('#bookingSearch');  
 let allBookings = [];  
 function parseJwt(token) {  
 const base64Url = token.split('.')[1];  
 const base64 = base64Url.replace(/-/g, '+').replace(/\_/g, '/');  
 return *JSON*.parse(*window*.atob(base64));  
 }  
 let role = null;  
 const token = *localStorage*.getItem('token');  
 if (token) {  
 try {  
 role = parseJwt(token).role;  
 } catch (e) {  
 *console*.error('JWT parse error:', e);  
 }  
 }  
 (async function autoBook() {  
 const params = new *URLSearchParams*(*window*.location.search);  
 if (params.has('zone') && params.has('start') && params.has('end')) {  
 const zoneId = parseInt(params.get('zone'), 10);  
 const start = params.get('start');  
 const end = params.get('end');  
 try {  
 const slots = await api('GET', `/api/slots/zone/${zoneId}`);  
 const free = slots.filter(s => s.status === 'free');  
 if (!free.length) {  
 alert('Нет свободных слотов в выбранной зоне');  
 } else {  
 await api('POST', '/api/bookings', {  
 parking\_slot\_id: free[0].id,  
 start\_time: start,  
 end\_time: end  
 });  
 }  
 } catch (err) {  
 *console*.error('Ошибка авто-бронирования:', err);  
 alert(err.responseJSON?.message || 'Ошибка создания брони');  
 }  
 *window*.history.replaceState({}, '', '/bookings');  
 }  
 })();  
 async function loadFreeSlots() {  
 const slots = await api('GET', '/api/slots/free');  
 $slotSel.empty().append('<option value="" disabled selected>Выберите слот</option>');  
 slots.forEach(s => {  
 $slotSel.append(`<option value="${s.id}">${s.slot\_number} (Зона ${s.zone\_id})</option>`);  
 });  
 }  
 function renderBookings(bookings) {  
 $tbody.empty();  
 bookings.forEach(b => {  
 let btns = '';  
 if (role === 'user' && ['pending','approved'].includes(b.status)) {  
 btns = `<button class="btn btn-sm btn-danger cancel-booking">Отменить</button>`;  
 }  
 if ((role === 'employee' || role === 'admin') && (b.status === 'pending'||b.status === 'approved'||b.status === 'rejected')) {  
 btns = `<button class="btn btn-sm btn-success approve-booking">✔</button>` +  
 `<button class="btn btn-sm btn-warning reject-booking ms-1">✖</button>`;  
 }  
 $tbody.append(`  
 <tr data-id="${b.id}">  
 <td>${b.id}</td>  
 <td>${b.parking\_slot\_id}</td>  
 <td>${b.start\_time}</td>  
 <td>${b.end\_time}</td>  
 <td>${b.status}</td>  
 <td>${btns}</td>  
 </tr>  
 `);  
 });  
 }  
 async function loadMy() {  
 const arr = await api('GET', '/api/bookings/my');  
 allBookings = arr;  
 renderBookings(allBookings);  
 }  
 async function loadSorted() {  
 const arr = await api('GET', '/api/bookings');  
 allBookings = [...arr.filter(b => b.status === 'pending'), ...arr.filter(b => b.status !== 'pending')];  
 renderBookings(allBookings);  
 }  
 (async function init() {  
 if (role === 'user') {  
 $addBtn.hide();  
 await loadMy();  
 } else {  
 $addBtn.show();  
 await loadSorted();  
 }  
 })();  
 $search.on('input', function() {  
 const q = $(this).val().trim();  
 if (!q) {  
 renderBookings(allBookings);  
 return;  
 }  
 const id = parseInt(q, 10);  
 if (isNaN(id)) {  
 $tbody.empty();  
 return;  
 }  
 const filtered = allBookings.filter(b => b.id === id);  
 renderBookings(filtered);  
 });  
 $addBtn.on('click', async () => {  
 $form[0].reset();  
 await loadFreeSlots();  
 });  
 $form.on('submit', async e => {  
 e.preventDefault();  
 const payload = {  
 parking\_slot\_id: parseInt($slotSel.val(), 10),  
 start\_time: $('#start\_time').val(),  
 end\_time: $('#end\_time').val()  
 };  
 try {  
 await api('POST', '/api/bookings', payload);  
 $('#bookingModal').modal('hide');  
 await loadMy();  
 } catch (err) {  
 alert(err.responseJSON?.message || 'Ошибка создания брони');  
 }  
 });  
 $tbody  
 .on('click', '.cancel-booking', async function() {  
 const id = $(this).closest('tr').data('id');  
 if (!confirm('Отменить бронирование?')) return;  
 await api('POST', `/api/bookings/cancel/${id}`);  
 await loadMy();  
 })  
 .on('click', '.approve-booking', async function() {  
 const id = $(this).closest('tr').data('id');  
 await api('POST', `/api/bookings/approve/${id}`);  
 await loadSorted();  
 })  
 .on('click', '.reject-booking', async function() {  
 const id = $(this).closest('tr').data('id');  
 await api('POST', `/api/bookings/reject/${id}`);  
 await loadSorted();  
 });  
});

ПРИЛОЖЕНИЕ 43. Файл client/public/js/cars.js

$(function() {  
 const $tableBody = $('#carsTableBody');  
 const $modal = $('#carModal');  
 const $form = $('#carForm');  
 const $label = $('#carModalLabel');  
 const $search = $('#carSearch');  
 async function loadCars(filter = '') {  
 try {  
 const url = filter  
 ? `/api/cars/filter?license\_plate=${encodeURIComponent(filter)}`  
 : '/api/cars';  
 const cars = await api('GET', url);  
 $tableBody.empty();  
 cars.forEach(c => {  
 $tableBody.append(`  
 <tr data-id="${c.id}">  
 <td>${c.id}</td>  
 <td>${c.license\_plate}</td>  
 <td>${c.model}</td>  
 <td>${c.entry\_time||''}</td>  
 <td>${c.exit\_time||''}</td>  
 <td>${c.parking\_slot\_id}</td>  
 <td>  
 <button class="btn btn-sm btn-primary edit-btn">✎</button>  
 <button class="btn btn-sm btn-warning release-btn">🔓</button>  
 <button class="btn btn-sm btn-danger delete-btn">🗑</button>  
 </td>  
 </tr>  
 `);  
 });  
 } catch {  
 alert('Ошибка загрузки автомобилей');  
 }  
 }  
 loadCars();  
 let timeout;  
 $search.on('input', function(){  
 clearTimeout(timeout);  
 const q = this.value.trim();  
 timeout = setTimeout(() => loadCars(q), 300);  
 });  
 $('#addCarBtn').on('click', () => {  
 $label.text('Добавить автомобиль');  
 $form[0].reset();  
 $('#carId').val('');  
 });  
 $tableBody.on('click', '.edit-btn', function() {  
 const $tr = $(this).closest('tr');  
 const id = $tr.data('id');  
 $label.text(`Редактировать #${id}`);  
 $('#carId').val(id);  
 $('#license\_plate').val($tr.find('td').eq(1).text());  
 $('#model').val($tr.find('td').eq(2).text());  
 $('#parking\_slot\_id').val($tr.find('td').eq(5).text());  
 $modal.modal('show');  
 });  
 $form.on('submit', async function(e) {  
 e.preventDefault();  
 const id = $('#carId').val();  
 const data = {  
 license\_plate: $('#license\_plate').val(),  
 model: $('#model').val(),  
 parking\_slot\_id: parseInt($('#parking\_slot\_id').val(), 10)  
 };  
 try {  
 if (id) {  
 await api('PUT', `/api/cars/${id}`, data);  
 } else {  
 await api('POST', '/api/cars', data);  
 }  
 $modal.modal('hide');  
 loadCars();  
 } catch (err) {  
 alert(err.responseJSON?.message || 'Ошибка сохранения');  
 }  
 });  
 $tableBody.on('click', '.release-btn', async function() {  
 const id = $(this).closest('tr').data('id');  
 try {  
 await api('POST', `/api/cars/release/${id}`);  
 loadCars();  
 } catch {  
 alert('Ошибка освобождения слота');  
 }  
 });  
 $tableBody.on('click', '.delete-btn', async function() {  
 if (!confirm('Удалить эту машину?')) return;  
 const id = $(this).closest('tr').data('id');  
 try {  
 await api('DELETE', `/api/cars/${id}`);  
 loadCars();  
 } catch {  
 alert('Ошибка удаления');  
 }  
 });  
});

ПРИЛОЖЕНИЕ 44. Файл client/public/js/dashboard.js

$(function() {  
 $('#searchForm').on('submit', async function(e) {  
 e.preventDefault();  
 $('#zoneResults').empty();  
 const start = $('#start').val();  
 const end = $('#end').val();  
 if (!start || !end) {  
 return alert('Пожалуйста, выберите оба времени начала и конца.');  
 }  
 if (new *Date*(end) <= new *Date*(start)) {  
 return alert('Дата окончания должна быть позже даты начала.');  
 }  
 try {  
 const zones = await api(  
 'GET',  
 `/api/slots/availability?start=${encodeURIComponent(start)}&end=${encodeURIComponent(end)}`  
 );  
 if (!zones.length) {  
 return $('#zoneResults').append(`  
 <div class="col-12">  
 <div class="alert alert-warning text-center">  
 Нет зон с свободными местами в этом интервале.  
 </div>  
 </div>  
 `);  
 }  
 zones.forEach(z => {  
 const freeIds = z.freeSlotIds || [];  
 const bookBtn = freeIds.length  
 ? `<button  
 class="btn btn-primary mt-auto book-zone-btn"  
 data-free-slot-ids='${*JSON*.stringify(freeIds)}'  
 data-start="${start}"  
 data-end="${end}"  
 >  
 Забронировать  
 </button>`  
 : `<span class="mt-auto text-danger">Нет свободных мест</span>`;  
 $('#zoneResults').append(`  
 <div class="col-md-4">  
 <div class="card h-100 shadow-sm d-flex flex-column">  
 <div class="card-body d-flex flex-column">  
 <h5 class="card-title">Зона ${z.name}</h5>  
 <p class="card-text mb-1">  
 <strong>${z.freeCount}</strong> свободно из  
 <strong>${z.totalCount}</strong>  
 </p>  
 <p class="card-text mb-3">  
 Цена: <strong>${z.pricePerHour} ₽/ч</strong>  
 </p>  
 ${bookBtn}  
 </div>  
 </div>  
 </div>  
 `);  
 });  
 } catch (err) {  
 *console*.error('Ошибка поиска зон:', err);  
 alert('Не удалось выполнить поиск зон.');  
 }  
 });  
 $('#zoneResults').on('click', '.book-zone-btn', async function() {  
 const freeIds = *JSON*.parse($(this).attr('data-free-slot-ids'));  
 const slotId = freeIds[0];   
 const start = $(this).data('start');  
 const end = $(this).data('end');  
 if (!slotId) {  
 return alert('Ошибка: не найден свободный слот.');  
 }  
 try {  
 await api('POST', '/api/bookings', {  
 parking\_slot\_id: slotId,  
 start\_time: start,  
 end\_time: end  
 });  
 *window*.location.href = '/bookings';  
 } catch (err) {  
 *console*.error('Ошибка создания брони:', err);  
 const msg = err.responseJSON?.message || 'Не удалось забронировать слот.';  
 alert(msg);  
 }  
 });  
});

ПРИЛОЖЕНИЕ 45. Файл client/public/js/slots.js

$(function() {  
 const $tbody = $('#slotsTableBody');  
 const $search = $('#slotSearch');  
 const $modal = $('#slotModal');  
 const $form = $('#slotForm');  
 const $label = $('#slotModalLabel');  
 const $selectZone = $('#zone\_id');  
 let zones = [];  
 async function loadZones() {  
 try {  
 zones = await api('GET', '/api/zones');  
 $selectZone.empty().append(  
 '<option value="" disabled selected>Выберите зону</option>'  
 );  
 zones.forEach(z => {  
 $selectZone.append(  
 `<option value="${z.id}">${z.id} — ${z.name}</option>`  
 );  
 });  
 } catch {  
 alert('Не удалось загрузить зоны');  
 }  
 }  
 async function loadSlots(filter = '') {  
 try {  
 const url = filter  
 ? `/api/slots?slot\_number=${encodeURIComponent(filter)}`  
 : '/api/slots';  
 const slots = await api('GET', url);  
 $tbody.empty();  
 slots.forEach(s => {  
 $tbody.append(`  
 <tr data-id="${s.id}">  
 <td>${s.id}</td>  
 <td>${s.slot\_number}</td>  
 <td>${s.zone\_id}</td>  
 <td>${s.status}</td>  
 <td>  
 <button class="btn btn-sm btn-primary edit-slot">✎</button>  
 <button class="btn btn-sm btn-danger delete-slot">🗑</button>  
 </td>  
 </tr>  
 `);  
 });  
 } catch {  
 alert('Ошибка загрузки слотов');  
 }  
 }  
 loadSlots();  
 let timeout;  
 $search.on('input', function(){  
 clearTimeout(timeout);  
 const q = this.value.trim();  
 timeout = setTimeout(() => loadSlots(q), 200);  
 });  
 (async function init() {  
 await loadZones();  
 await loadSlots();  
 })();  
 $('#addSlotBtn').on('click', () => {  
 $label.text('Добавить слот');  
 $form[0].reset();  
 $('#slotId').val('');  
 });  
 $tbody.on('click', '.edit-slot', function() {  
 const $tr = $(this).closest('tr');  
 const id = $tr.data('id');  
 const number = $tr.find('td').eq(1).text();  
 const zone = $tr.find('td').eq(2).text();  
 const status= $tr.find('td').eq(3).text();  
  
 $label.text(`Редактировать слот #${id}`);  
 $('#slotId').val(id);  
 $('#slot\_number').val(number);  
 $('#zone\_id').val(zone);  
 $('#status').val(status);  
 $modal.modal('show');  
 });  
 $form.on('submit', async function(e) {  
 e.preventDefault();  
 const id = $('#slotId').val();  
 const payload = {  
 slot\_number: parseInt($('#slot\_number').val(), 10),  
 zone\_id: parseInt($('#zone\_id').val(), 10),  
 status: $('#status').val()  
 };  
 try {  
 if (id) {  
 await api('PUT', `/api/slots/${id}`, payload);  
 } else {  
 await api('POST', '/api/slots', payload);  
 }  
 $modal.modal('hide');  
 loadSlots();  
 } catch (err) {  
 alert(err.responseJSON?.message || 'Ошибка сохранения слота');  
 }  
 });  
 $tbody.on('click', '.delete-slot', async function() {  
 if (!confirm('Удалить слот?')) return;  
 const id = $(this).closest('tr').data('id');  
 try {  
 await api('DELETE', `/api/slots/${id}`);  
 loadSlots();  
 } catch {  
 alert('Ошибка удаления слота');  
 }  
 });  
});

ПРИЛОЖЕНИЕ 46. Файл client/public/js/tariffs.js

$(function () {  
 const $tbody = $('#tariffsTableBody');  
 const $modal = $('#tariffModal');  
 const $form = $('#tariffForm');  
 const $label = $('#tariffModalLabel');  
 const $search = $('#tariffSearch');  
 async function loadTariffs(filter = '') {  
 try {  
 const tariffs = await api('GET', '/api/tariffs');  
 const list = filter  
 ? tariffs.filter(t => t.name.toLowerCase().includes(filter.toLowerCase()))  
 : tariffs;  
 $tbody.empty();  
 list.forEach(t => {  
 $tbody.append(`  
 <tr data-id="${t.id}">  
 <td>${t.id}</td>  
 <td>${t.name}</td>  
 <td>${t.price\_per\_hour}</td>  
 <td>  
 <button class="btn btn-sm btn-primary edit-tariff">✎</button>  
 <button class="btn btn-sm btn-danger delete-tariff">🗑</button>  
 </td>  
 </tr>  
 `);  
 });  
 } catch {  
 alert('Ошибка загрузки тарифов');  
 }  
 }  
 loadTariffs();  
 let timeout;  
 $search.on('input', function () {  
 clearTimeout(timeout);  
 const q = this.value.trim();  
 timeout = setTimeout(() => loadTariffs(q), 200);  
 });  
 $('#addTariffBtn').on('click', () => {  
 $label.text('Добавить тариф');  
 $form[0].reset();  
 $('#tariffId').val('');  
 });  
 $tbody.on('click', '.edit-tariff', function () {  
 const $tr = $(this).closest('tr');  
 const id = $tr.data('id');  
 const cols = $tr.find('td');  
 $label.text(`Редактировать тариф #${id}`);  
 $('#tariffId').val(id);  
 $('#name').val(cols.eq(1).text());  
 $('#price\_per\_hour').val(cols.eq(2).text());  
 $modal.modal('show');  
 });  
 $form.on('submit', async function (e) {  
 e.preventDefault();  
 const id = $('#tariffId').val();  
 const data = {  
 name: $('#name').val(),  
 price\_per\_hour: parseFloat($('#price\_per\_hour').val())  
 };  
 try {  
 if (id) {  
 await api('PUT', `/api/tariffs/${id}`, data);  
 } else {  
 await api('POST', '/api/tariffs', data);  
 }  
 $modal.modal('hide');  
 loadTariffs();  
 } catch (err) {  
 alert(err.responseJSON?.message || 'Ошибка сохранения тарифа');  
 }  
 });  
 $tbody.on('click', '.delete-tariff', async function () {  
 if (!confirm('Удалить тариф?')) return;  
 const id = $(this).closest('tr').data('id');  
 try {  
 await api('DELETE', `/api/tariffs/${id}`);  
 loadTariffs();  
 } catch {  
 alert('Ошибка удаления тарифа');  
 }  
 });  
});

ПРИЛОЖЕНИЕ 47. Файл client/public/js/users.js

$(function () {  
 const $tbody = $('#usersTableBody');  
 const $modal = $('#userModal');  
 const $form = $('#userForm');  
 const $label = $('#userModalLabel');  
 const $search = $('#searchUserInput');  
 let allUsers = [];  
 function formatDateTime(input) {  
 if (/^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}$/.test(input)) {  
 return input;  
 }  
 const d = new *Date*(input);  
 if (isNaN(d)) return input;  
 const pad = n => String(n).padStart(2, '0');  
 return `${pad(d.getDate())}-${pad(d.getMonth() + 1)}-${d.getFullYear()} ` +  
 `${pad(d.getHours())}:${pad(d.getMinutes())}:${pad(d.getSeconds())}`;  
 }  
 async function loadUsers() {  
 try {  
 const users = await api('GET', '/api/users');  
 allUsers = users;  
 $tbody.empty();  
 users.forEach(u => {  
 $tbody.append(`  
 <tr data-id="${u.id}">  
 <td>${u.id}</td>  
 <td>${u.username}</td>  
 <td>${u.fullname || ''}</td>  
 <td>${u.email}</td>  
 <td>${u.role}</td>  
 <td>${formatDateTime(u.createdAt)}</td>  
 <td>  
 <button class="btn btn-sm btn-primary edit-user">✎</button>  
 <button class="btn btn-sm btn-danger delete-user">🗑</button>  
 </td>  
 </tr>  
 `);  
 });  
 } catch (err) {  
 *console*.error('Ошибка загрузки пользователей:', err);  
 alert('Ошибка загрузки пользователей');  
 }  
 }  
 loadUsers();  
 $search.on('input', function() {  
 const q = $(this).val().trim();  
 if (!q) {  
 loadUsers();  
 return;  
 }  
 const id = parseInt(q, 10);  
 if (isNaN(id)) {  
 $tbody.empty();  
 return;  
 }  
 const filtered = allUsers.filter(u => u.id === id);  
 $tbody.empty();  
 filtered.forEach(u => {  
 $tbody.append(`  
 <tr data-id="${u.id}">  
 <td>${u.id}</td>  
 <td>${u.username}</td>  
 <td>${u.fullname || ''}</td>  
 <td>${u.email}</td>  
 <td>${u.role}</td>  
 <td>${formatDateTime(u.createdAt)}</td>  
 <td>  
 <button class="btn btn-sm btn-primary edit-user">✎</button>  
 <button class="btn btn-sm btn-danger delete-user">🗑</button>  
 </td>  
 </tr>  
 `);  
 });  
 });  
 $('#addUserBtn').on('click', () => {  
 $label.text('Добавить пользователя');  
 $form[0].reset();  
 $('#userId').val('');  
 });  
 $tbody.on('click', '.edit-user', function () {  
 const $tr = $(this).closest('tr');  
 const id = $tr.data('id');  
 const cols = $tr.find('td');  
 $label.text(`Редактировать #${id}`);  
 $('#userId').val(id);  
 $('#username').val(cols.eq(1).text());  
 $('#password').val('');  
 $('#fullname').val(cols.eq(2).text());  
 $('#email').val(cols.eq(3).text());  
 $('#role').val(cols.eq(4).text());  
 $modal.modal('show');  
 });  
 $form.on('submit', async function (e) {  
 e.preventDefault();  
 const id = $('#userId').val();  
 const payload = {  
 username: $('#username').val(),  
 fullname: $('#fullname').val(),  
 email: $('#email').val(),  
 role: $('#role').val()  
 };  
 if ($('#password').val()) {  
 payload.password = $('#password').val();  
 }  
 try {  
 if (id) {  
 await api('PUT', `/api/users/${id}`, payload);  
 } else {  
 await api('POST', '/api/users', payload);  
 }  
 $modal.modal('hide');  
 loadUsers();  
 } catch (err) {  
 *console*.error('Ошибка сохранения пользователя:', err);  
 alert(err.responseJSON?.message || 'Ошибка сохранения пользователя');  
 }  
 });  
 $tbody.on('click', '.delete-user', async function () {  
 if (!confirm('Удалить пользователя?')) return;  
 const id = $(this).closest('tr').data('id');  
  
 try {  
 await api('DELETE', `/api/users/${id}`);  
 loadUsers();  
 } catch (err) {  
 *console*.error('Ошибка удаления пользователя:', err);  
 alert(err.responseJSON?.message || 'Ошибка удаления пользователя');  
 }  
 });  
});

ПРИЛОЖЕНИЕ 48. Файл client/public/js/zones.js

$(function() {  
 const $tbody = $('#zonesTableBody');  
 const $modal = $('#zoneModal');  
 const $form = $('#zoneForm');  
 const $label = $('#zoneModalLabel');  
 const $tariffSel= $('#tariff\_id');  
 const $search = $('#zoneSearch');  
 let tariffs = [];  
 async function loadTariffsList() {  
 try {  
 tariffs = await api('GET', '/api/tariffs');  
 $tariffSel.empty().append(  
 '<option value="" disabled selected>Выберите тариф</option>'  
 );  
 tariffs.forEach(t => {  
 $tariffSel.append(`<option value="${t.id}">${t.name} (${t.price\_per\_hour})</option>`);  
 });  
 } catch {  
 alert('Не удалось загрузить тарифы');  
 }  
 }  
 async function loadZones(filter='') {  
 try {  
 const base = '/api/zones';  
 const url = filter  
 ? `${base}?name=${encodeURIComponent(filter)}`  
 : base;  
 const list = await api('GET', url);  
 $tbody.empty();  
 list.forEach(z => {  
 $tbody.append(`  
 <tr data-id="${z.id}">  
 <td>${z.id}</td>  
 <td>${z.name}</td>  
 <td>${z.tariff\_id}</td>  
 <td>  
 <button class="btn btn-sm btn-primary edit-zone">✎</button>  
 <button class="btn btn-sm btn-danger delete-zone">🗑</button>  
 </td>  
 </tr>  
 `);  
 });  
 } catch {  
 alert('Ошибка загрузки зон');  
 }  
 }  
 loadZones();  
 let timeout;  
 $search.on('input', function(){  
 clearTimeout(timeout);  
 const q = this.value.trim();  
 timeout = setTimeout(() => loadZones(q), 200);  
 });  
 (async function init() {  
 await loadTariffsList();  
 await loadZones();  
 })();  
 $('#addZoneBtn').on('click', () => {  
 $label.text('Добавить зону');  
 $form[0].reset();  
 $('#zoneId').val('');  
 });  
 $tbody.on('click', '.edit-zone', function() {  
 const $tr = $(this).closest('tr');  
 const id = $tr.data('id');  
 const cols = $tr.find('td');  
 $label.text(`Редактировать зону #${id}`);  
 $('#zoneId').val(id);  
 $('#name').val(cols.eq(1).text());  
 $('#tariff\_id').val(cols.eq(2).text());  
 $modal.modal('show');  
 });  
 $form.on('submit', async function(e) {  
 e.preventDefault();  
 const id = $('#zoneId').val();  
 const data = {  
 name: $('#name').val(),  
 tariff\_id: parseInt($('#tariff\_id').val(), 10)  
 };  
 try {  
 if (id) {  
 await api('PUT', `/api/zones/${id}`, data);  
 } else {  
 await api('POST', '/api/zones', data);  
 }  
 $modal.modal('hide');  
 await loadTariffsList();  
 loadZones();  
 } catch (err) {  
 alert(err.responseJSON?.message || 'Ошибка сохранения зоны');  
 }  
 });  
 $tbody.on('click', '.delete-zone', async function() {  
 if (!confirm('Удалить зону?')) return;  
 const id = $(this).closest('tr').data('id');  
 try {  
 await api('DELETE', `/api/zones/${id}`);  
 loadZones();  
 } catch {  
 alert('Ошибка удаления зоны');  
 }  
 });  
});

ПРИЛОЖЕНИЕ 49. Файл client/public/views /bookings.pug

extends layout  
block content  
 .*d-flex*.*justify-content-between*.*align-items-center*.*mb-4* h1.*mb-0* Бронирования  
 input#bookingSearch.*form-control*.*w-25*(type="number" placeholder="Поиск по ID…")  
 if role === 'user'  
 button.*btn*.*btn-success*#addBookingBtn(type="button" data-bs-toggle="modal" data-bs-target="#bookingModal")  
 i.*bi*.*bi-plus-lg*.*me-1* | Новая бронь  
 table.*table*.*table-striped* thead  
 tr  
 th ID  
 th Слот (ID)  
 th Время въезда  
 th Время выезда  
 th Статус  
 th Действия  
 tbody#bookingsTableBody  
 .*modal*.*fade*#bookingModal(tabindex="-1" aria-hidden="true")  
 .*modal-dialog* .*modal-content* .*modal-header* h5.*modal-title*#bookingModalLabel Новое бронирование  
 button.*btn-close*(type="button" data-bs-dismiss="modal")  
 .*modal-body* form#bookingForm  
 .*mb-3* label.*form-label*(for="parking\_slot\_id") Слот  
 select.*form-select*#parking\_slot\_id(required)  
 option(value="" disabled selected) Загрузка слотов...  
 .*mb-3* label.*form-label*(for="start\_time") Время въезда  
 input.*form-control*#start\_time(type="datetime-local" required)  
 .*mb-3* label.*form-label*(for="end\_time") Время выезда  
 input.*form-control*#end\_time(type="datetime-local" required)  
 .*modal-footer* button.*btn*.*btn-secondary*(type="button" data-bs-dismiss="modal") Отмена  
 button.*btn*.*btn-primary*(type="submit" form="bookingForm") Забронировать  
block scripts  
 script(src="/static/js/bookings.js")

ПРИЛОЖЕНИЕ 50. Файл client/public/views /cars.pug

extends layout  
block content  
 .*d-flex*.*justify-content-between*.*align-items-center*.*mb-4* h1.*mb-0* Автомобили  
 input#carSearch.*form-control*.*w-25*(type="text" placeholder="Поиск по гос.номеру…")  
 button.*btn*.*btn-success*#addCarBtn(type="button" data-bs-toggle="modal" data-bs-target="#carModal")  
 i.*bi*.*bi-plus-lg*.*me-1* | Добавить автомобиль  
 table.*table*.*table-striped* thead  
 tr  
 th ID  
 th Гос. номер  
 th Модель  
 th “Въезд”  
 th “Выезд”  
 th “Слот”  
 th Действия  
 tbody#carsTableBody  
 .*modal*.*fade*#carModal(tabindex="-1" aria-hidden="true")  
 .*modal-dialog* .*modal-content* .*modal-header* h5.*modal-title*#carModalLabel Добавить автомобиль  
 button.*btn-close*(type="button" data-bs-dismiss="modal")  
 .*modal-body* form#carForm  
 input(type="hidden" id="carId")  
 .*mb-3* label.*form-label*(for="license\_plate") Гос. номер  
 input.*form-control*#license\_plate(type="text" required)  
 .*mb-3* label.*form-label*(for="model") Модель  
 input.*form-control*#model(type="text" required)  
 .*mb-3* label.*form-label*(for="parking\_slot\_id") Слот (ID)  
 input.*form-control*#parking\_slot\_id(type="number" required)  
 .*modal-footer* button.*btn*.*btn-secondary*(type="button" data-bs-dismiss="modal") Отмена  
 button.*btn*.*btn-primary*(type="submit" form="carForm") Сохранить  
block scripts  
 script(src="/static/js/cars.js")

ПРИЛОЖЕНИЕ 51. Файл client/public/views /dashboard.pug

extends layout  
block content  
 .*py-5*.*bg-white* .*container* h1.*text-center*.*mb-4*.*text-primary* Park. Pay. Go.  
 p.*lead*.*text-center*.*mb-5* Введите диапазон времени и узнайте, в каких зонах есть свободные места.  
 form#searchForm.*row*.*g-2*.*justify-content-center* .*col-auto* input.*form-control*(type="datetime-local" id="start" name="start" required)  
 .*col-auto* input.*form-control*(type="datetime-local" id="end" name="end" required)  
 .*col-auto* button.*btn*.*btn-primary*(type="submit")  
 i.*bi*.*bi-search*.*me-1* | Найти  
 .*container*.*mt-4* #zoneResults.*row*.*g-4*block scripts  
 script(src="/static/js/dashboard.js")

ПРИЛОЖЕНИЕ 52. Файл client/public/views /layout.pug

doctype html  
html(lang="ru")  
 head  
 meta(charset="utf-8")  
 meta(name="viewport" content="width=device-width, initial-scale=1")  
 title #{pageTitle || 'Автостоянка'}  
 link(rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css")  
 link(rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.5/font/bootstrap-icons.css")  
 link(rel="stylesheet" href="/static/css/main.css")  
 body  
 if showNav  
 nav.*navbar*.*navbar-expand-lg*.*navbar-dark*.*bg-primary* .*container-fluid* a.*navbar-brand*(href="/") 🚗 Автостоянка  
 button.*navbar-toggler*(type="button", data-bs-toggle="collapse", data-bs-target="#navbarNav")  
 span.*navbar-toggler-icon* #navbarNav.*collapse*.*navbar-collapse* ul.*navbar-nav*.*me-auto*.*mb-2*.*mb-lg-0* li.*nav-item*: a.*nav-link*(href="/dashboard") Дашборд  
 li.*nav-item*: a.*nav-link*(href="/cars") Автомобили  
 li.*nav-item*: a.*nav-link*(href="/slots") Слоты  
 li.*nav-item*: a.*nav-link*(href="/tariffs") Тарифы  
 li.*nav-item*: a.*nav-link*(href="/zones") Зоны  
 li.*nav-item*: a.*nav-link*(href="/bookings") Бронирования  
 li.*nav-item*: a.*nav-link*(href="/users") Пользователи  
 ul.*navbar-nav* li.*nav-item* a.*nav-link*#logout(href="#")  
 i.*bi*.*bi-box-arrow-right*.*me-1* | Выйти  
 main.*container*.*mt-4* block content  
 script(src="https://code.jquery.com/jquery-3.6.0.min.js")  
 script(src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js")  
 script(src="/static/js/api.js")  
 script(src="/static/js/auth.js")  
 block scripts

ПРИЛОЖЕНИЕ 53. Файл client/public/views /login.pug

extends layout  
block content  
 .*vh-100*.*d-flex*.*align-items-center*.*justify-content-center*.*bg-white* .*card*.*shadow-lg*(style="max-width: 400px; width: 100%;")  
 .*card-body*.*p-4* h2.*text-center*.*mb-4* i.*bi*.*bi-shield-lock*.*me-2* | Вход  
 form#loginForm  
 .*mb-3* label.*form-label*(for="username") Логин  
 input.*form-control*#username(type="text" name="username" placeholder="Введите логин" required)  
 .*mb-3* label.*form-label*(for="password") Пароль  
 input.*form-control*#password(type="password" name="password" placeholder="Введите пароль" required)  
 .*d-grid* button.*btn*.*btn-primary*(type="submit") Войти  
 p.*text-center*.*text-danger*.*mt-3*#loginError(style="display:none;")  
 .*text-center*.*mt-2* a(href="/register") Нет аккаунта? Зарегистрироваться  
block scripts  
 script(src="/static/js/auth.js")

ПРИЛОЖЕНИЕ 54. Файл client/public/views /register.pug

extends layout  
block content  
 .*vh-100*.*d-flex*.*align-items-center*.*justify-content-center*.*bg-white* .*card*.*shadow-lg*(style="max-width: 400px; width: 100%;")  
 .*card-body*.*p-4* h2.*text-center*.*mb-4* i.*bi*.*bi-person-plus*.*me-2* | Регистрация  
 form#registerForm  
 .*mb-3* label.*form-label*(for="username") Логин  
 input.*form-control*#username(type="text" name="username" placeholder="Введите логин" required)  
 .*mb-3* label.*form-label*(for="password") Пароль  
 input.*form-control*#password(type="password" name="password" placeholder="Введите пароль" required)  
 .*mb-3* label.*form-label*(for="fullname") Имя  
 input.*form-control*#fullname(type="text" name="fullname" placeholder="Введите имя" required)  
 .*mb-3* label.*form-label*(for="email") E-mail  
 input.*form-control*#email(type="email" name="email" placeholder="Введите e-mail" required)  
 .*d-grid* button.*btn*.*btn-primary*(type="submit") Зарегистрироваться  
 p.*text-center*.*text-danger*.*mt-3*#registerError(style="display:none;")  
 .*text-center*.*mt-2* a(href="/login") Уже есть аккаунт? Войти  
block scripts  
 script(src="/static/js/auth.js")

ПРИЛОЖЕНИЕ 55. Файл client/public/views /slots.pug

extends layout  
block content  
 .*d-flex*.*justify-content-between*.*align-items-center*.*mb-4* h1.*mb-0* Слоты  
 input#slotSearch.*form-control*.*w-25*(type="number" placeholder="Поиск по номеру…")  
 button.*btn*.*btn-success*#addSlotBtn(type="button" data-bs-toggle="modal" data-bs-target="#slotModal")  
 i.*bi*.*bi-plus-lg*.*me-1* | Добавить слот  
 table.*table*.*table-hover* thead  
 tr  
 th ID  
 th Номер  
 th Зона (ID)  
 th Статус  
 th Действия  
 tbody#slotsTableBody  
 .*modal*.*fade*#slotModal(tabindex="-1" aria-hidden="true")  
 .*modal-dialog* .*modal-content* .*modal-header* h5.*modal-title*#slotModalLabel Добавить слот  
 button.*btn-close*(type="button" data-bs-dismiss="modal")  
 .*modal-body* form#slotForm  
 input(type="hidden" id="slotId")  
 .*mb-3* label.*form-label*(for="slot\_number") Номер слота  
 input.*form-control*#slot\_number(type="number" required)  
 .*mb-3* label.*form-label*(for="zone\_id") Зона  
 select.*form-select*#zone\_id(required)  
 option(value="" disabled selected) Выберите зону  
 .*mb-3* label.*form-label*(for="status") Статус  
 select.*form-select*#status  
 option(value="free") free  
 option(value="occupied") occupied  
 .*modal-footer* button.*btn*.*btn-secondary*(type="button" data-bs-dismiss="modal") Отмена  
 button.*btn*.*btn-primary*(type="submit" form="slotForm") Сохранить  
block scripts  
 script(src="/static/js/slots.js")

ПРИЛОЖЕНИЕ 56. Файл client/public/views /tariffs.pug

extends layout  
block content  
 .*d-flex*.*justify-content-between*.*align-items-center*.*mb-4* h1.*mb-0* Тарифы  
 input#tariffSearch.*form-control*.*w-25*(type="text" placeholder="Поиск по названию…")  
 button.*btn*.*btn-success*#addTariffBtn(type="button" data-bs-toggle="modal" data-bs-target="#tariffModal")  
 i.*bi*.*bi-plus-lg*.*me-1* | Добавить тариф  
 table.*table*.*table-striped* thead  
 tr  
 th ID  
 th Название  
 th Цена/час  
 th Действия  
 tbody#tariffsTableBody  
 .*modal*.*fade*#tariffModal(tabindex="-1" aria-hidden="true")  
 .*modal-dialog* .*modal-content* .*modal-header* h5.*modal-title*#tariffModalLabel Добавить тариф  
 button.*btn-close*(type="button" data-bs-dismiss="modal")  
 .*modal-body* form#tariffForm  
 input(type="hidden" id="tariffId")  
 .*mb-3* label.*form-label*(for="name") Название  
 input.*form-control*#name(type="text" required)  
 .*mb-3* label.*form-label*(for="price\_per\_hour") Цена за час  
 input.*form-control*#price\_per\_hour(type="number" step="0.01" required)  
 .*modal-footer* button.*btn*.*btn-secondary*(type="button" data-bs-dismiss="modal") Отмена  
 button.*btn*.*btn-primary*(type="submit" form="tariffForm") Сохранить  
block scripts  
 script(src="/static/js/tariffs.js")

ПРИЛОЖЕНИЕ 57. Файл client/public/views /users.pug

extends layout  
block content  
 .*d-flex*.*justify-content-between*.*align-items-center*.*mb-4* h1.*mb-0* Пользователи  
 input#searchUserInput.*form-control*.*w-25*.*ms-2*(type="number" placeholder="Поиск по ID…")  
 button.*btn*.*btn-success*#addUserBtn(type="button" data-bs-toggle="modal" data-bs-target="#userModal")  
 i.*bi*.*bi-plus-lg*.*me-1* | Добавить пользователя  
 table.*table*.*table-hover* thead  
 tr  
 th ID  
 th Логин  
 th Имя  
 th E-mail  
 th Роль  
 th Дата создания  
 th Действия  
 tbody#usersTableBody  
 .*modal*.*fade*#userModal(tabindex="-1" aria-hidden="true")  
 .*modal-dialog* .*modal-content* .*modal-header* h5.*modal-title*#userModalLabel Добавить пользователя  
 button.*btn-close*(type="button" data-bs-dismiss="modal")  
 .*modal-body* form#userForm  
 input(type="hidden" id="userId")  
 .*mb-3* label.*form-label*(for="username") Логин  
 input.*form-control*#username(type="text" required)  
 .*mb-3* label.*form-label*(for="password") Пароль  
 input.*form-control*#password(type="password")  
 small.*text-muted* пароля не будет обновлено, если оставить пустым  
 .*mb-3* label.*form-label*(for="fullname") Имя  
 input.*form-control*#fullname(type="text" required)  
 .*mb-3* label.*form-label*(for="email") E-mail  
 input.*form-control*#email(type="email" required)  
 .*mb-3* label.*form-label*(for="role") Роль  
 select.*form-select*#role(required)  
 option(value="admin") admin  
 option(value="employee") employee  
 option(value="user", selected) user  
 .*modal-footer* button.*btn*.*btn-secondary*(type="button" data-bs-dismiss="modal") Отмена  
 button.*btn*.*btn-primary*(type="submit" form="userForm") Сохранить  
block scripts  
 script(src="/static/js/users.js")

ПРИЛОЖЕНИЕ 58. Файл client/public/views /zones.pug

extends layout  
block content  
 .*d-flex*.*justify-content-between*.*align-items-center*.*mb-4* h1.*mb-0* Зоны  
 input#zoneSearch.*form-control*.*w-25*(type="text" placeholder="Поиск по названию…")  
 button.*btn*.*btn-success*#addZoneBtn(type="button" data-bs-toggle="modal" data-bs-target="#zoneModal")  
 i.*bi*.*bi-plus-lg*.*me-1* | Добавить зону  
 table.*table*.*table-hover* thead  
 tr  
 th ID  
 th Название  
 th Тариф (ID)  
 th Действия  
 tbody#zonesTableBody  
 .*modal*.*fade*#zoneModal(tabindex="-1" aria-hidden="true")  
 .*modal-dialog* .*modal-content* .*modal-header* h5.*modal-title*#zoneModalLabel Добавить зону  
 button.*btn-close*(type="button" data-bs-dismiss="modal")  
 .*modal-body* form#zoneForm  
 input(type="hidden" id="zoneId")  
 .*mb-3* label.*form-label*(for="name") Название  
 input.*form-control*#name(type="text" required)  
 .*mb-3* label.*form-label*(for="tariff\_id") Тариф  
 select.*form-select*#tariff\_id(required)  
 option(value="" disabled selected) Загрузка тарифов...  
 .*modal-footer* button.*btn*.*btn-secondary*(type="button" data-bs-dismiss="modal") Отмена  
 button.*btn*.*btn-primary*(type="submit" form="zoneForm") Сохранить  
block scripts  
 script(src="/static/js/zones.js")

ПРИЛОЖЕНИЕ 59. Файл server/services/StatisticsService.js

const {Sequelize} = *require*('sequelize');  
const {ParkingSlot, Zone, Tariff, User, Car} = *require*('../models/Models');  
const *sequelize* = *require*('../db');  
  
class StatisticsService {  
 async getOverview(start, end) {  
 let dateCondBookings = '';  
 let dateCondCars = '';  
 const replacements = {};  
  
 if (start) {  
 dateCondBookings += ` AND b.start\_time::date >= :startDate`;  
 replacements.startDate = start;  
 }  
 if (end) {  
 dateCondBookings += ` AND b.end\_time::date <= :endDate`;  
 replacements.endDate = end;  
 }  
  
 if (start) {  
 dateCondCars += ` AND c.exit\_time::date >= :startDate`;  
 replacements.startDate = start;  
 }  
 if (end) {  
 dateCondCars += ` AND c.entry\_time::date <= :endDate`;  
 replacements.endDate = end;  
 }  
  
 const totalBookingsQuery = `  
 SELECT *COUNT*(\*) AS total\_bookings  
 FROM bookings b  
 WHERE 1 = 1  
 ${dateCondBookings}  
 `;  
 const totalBookingsResult = await *sequelize*.query(totalBookingsQuery, {  
 type: Sequelize.QueryTypes.*SELECT*,  
 replacements  
 });  
 const totalBookings = parseInt(totalBookingsResult[0].total\_bookings, 10) || 0;  
  
 const approvedCountQuery = `  
 SELECT *COUNT*(\*) AS approved\_count  
 FROM bookings b  
 WHERE b.status = 'approved'  
 ${dateCondBookings}  
 `;  
 const approvedCountResult = await *sequelize*.query(approvedCountQuery, {  
 type: Sequelize.QueryTypes.*SELECT*,  
 replacements  
 });  
 const approvedCount = parseInt(approvedCountResult[0].approved\_count, 10) || 0;  
  
 const totalCarsQuery = `  
 SELECT *COUNT*(\*) AS total\_cars  
 FROM cars c  
 WHERE c.exit\_time IS NOT NULL  
 ${dateCondCars}  
 `;  
 const totalCarsResult = await *sequelize*.query(totalCarsQuery, {  
 type: Sequelize.QueryTypes.*SELECT*,  
 replacements  
 });  
 const totalCars = parseInt(totalCarsResult[0].total\_cars, 10) || 0;  
  
 const revenueQuery = `  
 SELECT *COALESCE*(*SUM*(  
 *EXTRACT*(EPOCH FROM (c.exit\_time - c.entry\_time)) / 3600 \* t.price\_per\_hour  
 ), 0) AS total\_revenue  
 FROM cars c  
 JOIN parking\_slots ps ON ps.id = c.parking\_slot\_id  
 JOIN zones z ON z.id = ps.zone\_id  
 JOIN tariffs t ON t.id = z.tariff\_id  
 WHERE c.exit\_time IS NOT NULL  
 ${dateCondCars}  
 `;  
 const revenueResult = await *sequelize*.query(revenueQuery, {  
 type: Sequelize.QueryTypes.*SELECT*,  
 replacements  
 });  
 const totalRevenue = parseFloat(revenueResult[0].total\_revenue) || 0.0;  
  
 const totalUsers = await User.*count*();  
  
 return {  
 totalBookings,  
 approvedCount,  
 totalCars,  
 totalRevenue,  
 totalUsers  
 };  
 }  
  
 async getByZone(start, end) {  
 let dateCondBookings = '';  
 let dateCondCars = '';  
 const replacements = {};  
  
 if (start) {  
 dateCondBookings += ` AND b.start\_time::date >= :startDate`;  
 replacements.startDate = start;  
 }  
 if (end) {  
 dateCondBookings += ` AND b.end\_time::date <= :endDate`;  
 replacements.endDate = end;  
 }  
  
 if (start) {  
 dateCondCars += ` AND c.exit\_time::date >= :startDate`;  
 replacements.startDate = start;  
 }  
 if (end) {  
 dateCondCars += ` AND c.entry\_time::date <= :endDate`;  
 replacements.endDate = end;  
 }  
  
 const zonesStatsQuery = `  
 SELECT z.id AS zone\_id,  
 z.*name* AS zone\_name,  
  
 *COUNT*(DISTINCT ps.id) AS total\_slots,  
  
 *COUNT*(  
 DISTINCT CASE  
 WHEN ps.status = 'free' THEN ps.id  
 ELSE NULL  
 END  
 ) AS free\_slots,  
  
 *COUNT*(b.id) FILTER (  
 WHERE b.status = 'approved'  
 ${dateCondBookings}  
 ) AS approved\_bookings,  
  
 *COUNT*(c.id) FILTER (  
 WHERE c.exit\_time IS NOT NULL  
 ${dateCondCars} ) AS cars\_count,  
  
 COALESCE(  
 SUM(  
 EXTRACT(EPOCH FROM (c.exit\_time - c.entry\_time)) / 3600 \* t.price\_per\_hour  
 ) FILTER (  
 WHERE c.exit\_time IS NOT NULL ${dateCondCars}  
 ), 0  
 ) AS revenue  
  
 FROM zones z  
 JOIN tariffs t  
 ON t.id = z.tariff\_id  
 LEFT JOIN parking\_slots ps ON ps.zone\_id = z.id  
 LEFT JOIN bookings b  
 ON b.parking\_slot\_id = ps.id  
 AND b.status = 'approved'  
 LEFT JOIN cars c  
 ON c.parking\_slot\_id = ps.id  
  
 GROUP BY z.id, z.name, t.price\_per\_hour  
 ORDER BY z.id;  
 `;  
  
 const zonesStats = await *sequelize*.query(zonesStatsQuery, {  
 type: Sequelize.QueryTypes.*SELECT*,  
 replacements  
 });  
  
 return zonesStats.map(row => ({  
 zoneId: row.zone\_id,  
 zoneName: row.zone\_name,  
 totalSlots: parseInt(row.total\_slots, 10),  
 freeSlots: parseInt(row.free\_slots, 10),  
 approvedBookings: parseInt(row.approved\_bookings, 10),  
 carsCount: parseInt(row.cars\_count, 10),  
 revenue: parseFloat(row.revenue)  
 }));  
 }  
}  
  
*module*.exports.StatisticsService = StatisticsService;

ПРИЛОЖЕНИЕ 60. Файл server/routes/Statistics.js

const express = *require*('express');  
const { roleCheck, authMiddleware } = *require*('../middlewares/AuthMiddleware');  
const { StatisticsService } = *require*('../services/StatisticsService');  
const StatisticsController = *require*('../controllers/StatisticsController');  
  
const *router* = express.Router();  
*router*.use(authMiddleware);  
*router*.use(roleCheck(['employee','admin']));  
  
const statisticsService = new StatisticsService();  
const statisticsController = new StatisticsController(statisticsService);  
  
*router*.get('/overview', statisticsController.getOverview);  
  
*router*.get('/zones', statisticsController.getByZone);  
  
*module*.exports = *router*;

ПРИЛОЖЕНИЕ 60. Файл server/controllers/StatisticsController.js

const { StatisticsService } = *require*('../services/StatisticsService');  
  
class StatisticsController {  
 constructor(statisticsService) {  
 this.statisticsService = statisticsService;  
 this.getOverview = this.getOverview.bind(this);  
 this.getByZone = this.getByZone.bind(this);  
 }  
  
  
 async getOverview(req, res, next) {  
 try {  
 const { start, end } = req.query;  
 const data = await this.statisticsService.getOverview(start, end);  
 res.json(data);  
 } catch (err) {  
 next(err);  
 }  
 }  
  
  
 async getByZone(req, res, next) {  
 try {  
 const { start, end } = req.query;  
 const data = await this.statisticsService.getByZone(start, end);  
 res.json(data);  
 } catch (err) {  
 next(err);  
 }  
 }  
}  
  
*module*.exports = StatisticsController;

ПРИЛОЖЕНИЕ 61. Файл client/public/views/statistics.pug

extends layout  
  
block content  
 .container.mt-4  
 h1.text-center.mb-4.text-primary Статистика  
  
 .card.mb-4  
 .card-body  
 form#dateFilterForm  
 .row.g-3.align-items-end  
 .col-md-4  
 label.form-label(for="startDate") Дата с:  
 input#startDate.form-control(type="date")  
  
 .col-md-4  
 label.form-label(for="endDate") Дата по:  
 input#endDate.form-control(type="date")  
  
 .col-md-4  
 button#applyFilter.btn.btn-primary.w-100(type="submit") Применить  
  
 .card.mb-4  
 .card-body  
 h5.card-title.mb-3 Обзор  
 .row.g-3  
 .col  
 .card.h-100  
 .card-body.d-flex.flex-column.align-items-center.justify-content-center  
 h6.text-center.mb-2 Всего бронирований  
 p#totalBookings.lead.mb-0 0  
 .col  
 .card.h-100  
 .card-body.d-flex.flex-column.align-items-center.justify-content-center  
 h6.text-center.mb-2 Одобрено  
 p#approvedCount.lead.mb-0 0  
 .col  
 .card.h-100  
 .card-body.d-flex.flex-column.align-items-center.justify-content-center  
 h6.text-center.mb-2 Автомобили  
 p#totalCars.lead.mb-0 0  
 .col  
 .card.h-100  
 .card-body.d-flex.flex-column.align-items-center.justify-content-center  
 h6.text-center.mb-2 Общая выручка  
 p#totalRevenue.lead.mb-0 ₸0.00  
 .col  
 .card.h-100  
 .card-body.d-flex.flex-column.align-items-center.justify-content-center  
 h6.text-center.mb-2 Пользователей  
 p#totalUsers.lead.mb-0 0  
  
 .card.mb-4  
 .card-body  
 h5.card-title Зоны  
 table.table.table-hover  
 thead  
 tr  
 th ID  
 th Название  
 th Всего слотов  
 th Свободных  
 th Бронирований  
 th Автомобили  
 th Выручка  
 tbody#zonesStatsBody  
  
block scripts  
 script(src="/static/js/statistics.js")

ПРИЛОЖЕНИЕ 62. Файл client/public/js /statistics.js

$(function() {  
 if ($('#totalBookings').length === 0) {  
 return;  
 }  
  
 function todayISO() {  
 const d = new *Date*();  
 const yyyy = d.getFullYear();  
 const mm = String(d.getMonth() + 1).padStart(2, '0');  
 const dd = String(d.getDate()).padStart(2, '0');  
 return `${yyyy}-${mm}-${dd}`;  
 }  
  
  
 const defaultEnd = todayISO();  
 const defaultStart = (() => {  
 const d = new *Date*();  
 d.setDate(d.getDate() - 7);  
 const yyyy = d.getFullYear();  
 const mm = String(d.getMonth() + 1).padStart(2, '0');  
 const dd = String(d.getDate()).padStart(2, '0');  
 return `${yyyy}-${mm}-${dd}`;  
 })();  
  
  
 $('#startDate').val(defaultStart);  
 $('#endDate').val(defaultEnd);  
  
  
 function getDateQueryParams() {  
 const start = $('#startDate').val();  
 const end = $('#endDate').val();  
 const params = new *URLSearchParams*();  
 if (start) params.append('start', start);  
 if (end) params.append('end', end);  
 return params.toString();  
 }  
  
  
 async function loadOverview() {  
 try {  
 const qs = getDateQueryParams();  
 const url = `/api/statistics/overview${qs ? '?' + qs : ''}`;  
 const data = await api('GET', url);  
  
 $('#totalBookings').text(data.totalBookings);  
 $('#approvedCount').text(data.approvedCount);  
 $('#totalCars').text(data.totalCars);  
 $('#totalRevenue').text(`₽${data.totalRevenue.toFixed(2)}`);  
 $('#totalUsers').text(data.totalUsers);  
 } catch (err) {  
 *console*.error('Ошибка loadOverview():', err);  
 alert('Не удалось загрузить сводную статистику');  
 }  
 }  
  
  
 async function loadZonesStats() {  
 try {  
 const qs = getDateQueryParams();  
 const url = `/api/statistics/zones${qs ? '?' + qs : ''}`;  
 const zones = await api('GET', url);  
  
 const $tbody = $('#zonesStatsBody');  
 $tbody.empty();  
  
 zones.forEach(z => {  
 $tbody.append(`  
 <tr>  
 <td>${z.zoneId}</td>  
 <td>${z.zoneName}</td>  
 <td>${z.totalSlots}</td>  
 <td>${z.freeSlots}</td>  
 <td>${z.approvedBookings}</td>  
 <td>${z.carsCount}</td>  
 <td>₽${z.revenue.toFixed(2)}</td>  
 </tr>  
 `);  
 });  
 } catch (err) {  
 *console*.error('Ошибка loadZonesStats():', err);  
 alert('Не удалось загрузить статистику по зонам');  
 }  
 }  
  
  
 $('#dateFilterForm').on('submit', function(e) {  
 e.preventDefault();  
 const start = $('#startDate').val();  
 const end = $('#endDate').val();  
 if (start && end && start > end) {  
 alert('Дата «с» не может быть больше даты «по»');  
 return;  
 }  
 reloadAll();  
 });  
  
 async function reloadAll() {  
 await loadOverview();  
 await loadZonesStats();  
 }  
  
 reloadAll();  
});