Министерство образования Республики Беларусь
УО «Витебский государственный технологический университет»
Кафедра «Информационные системы и технологии»

# Расчетно-графическая работа
# по теме «Театральная касса»

Вариант № 27

Выполнила:
студентка гр. Ит-13
Стукалова Д.А.
Проверил:
ст. Преподаватель каф. ИСиТ
Бизюк А. Н.

Витебск, 2026

# СОДЕРЖАНИЕ

# 1. ВВЕДЕНИЕ

**Актуальность.** Театральные организации нуждаются в цифровых системах бронирования и продажи билетов с учетом схемы зала, ценовых уровней и расписания показов. Веб-приложение позволяет снизить операционные затраты, повысить прозрачность учета и улучшить пользовательский опыт за счет онлайн-покупки и проверки билетов.

**Цель работы.** Спроектировать и реализовать веб-приложение «Театральная касса» на Laravel (PHP 8.2), поддерживающее управление постановками, сеансами, билетами и заказами, загрузку файлов (постеров), серверную валидацию, аутентификацию и разграничение ролей (администратор/пользователь), а также базовую статистику посещаемости и продаж.

**Задачи работы:**

1. Проанализировать предметную область (театральные показы, места, цены, заказы, билеты).
2. Спроектировать структуру базы данных (ER-модель и реляционная схема).
3. Реализовать основные сущности и связи в Laravel (модели и миграции).
4. Создать основные интерфейсы (контроллеры, маршруты, представления).
5. Реализовать валидацию данных через FormRequest.
6. Настроить аутентификацию и авторизацию.
7. Реализовать бизнес-логику покупки и проверки билетов (QR-код, check-in).
8. Написать набор функциональных тестов и настроить CI/CD.

**Объект исследования:** процессы продажи и учета билетов в театре.

**Предмет исследования:** программные методы автоматизации учета показов, мест и билетов.

**Методы:** анализ предметной области, проектирование БД, моделирование, реализация MVC, тестирование.

# 2. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Театральная касса — это система, которая поддерживает полный цикл работы с показами и билетами: ведение репертуара, расписания, цен, бронирование мест, оформление заказов, выдачу электронных билетов и фиксацию факта прохода (check-in).

**Ключевые сущности:**

- **Площадки (venues)** — описание театров и залов (название, адрес, вместимость).
- **Постановки (shows)** — спектакли (название, описание, режиссёр, длительность, язык, возрастной рейтинг, постер).
- **Сеансы (performances)** — конкретные показы спектаклей (дата/время, статус).
- **Секции и места (seat_sections, seats)** — схема зала.
- **Ценовые уровни (price_tiers)** — стоимость (в центах) с привязкой к площадке/постановке/секции.
- **Заказы (orders)** — агрегируют покупку одного или нескольких билетов конкретным пользователем.
- **Билеты (tickets)** — конкретные единицы продажи (сеанс-место) со статусом, ценой и QR-кодом.
- **Статистика (performance_stats)** — агрегированные показатели (продажи/выручка/check-in) по сеансам.

**Особенности предметной области:**

- Билет уникален для пары «сеанс — место».
- Один заказ может включать несколько билетов.
- Цена может зависеть от секции и/или постановки.
- Билет может быть «зарезервирован», «продан» либо «отменен».

**Типовые сценарии:**

1. Гость открывает афишу, выполняет поиск/фильтрацию/сортировку.
2. Пользователь выбирает спектакль и сеанс.
3. Пользователь бронирует место — создаётся ticket со статусом `reserved`.
4. Пользователь оформляет заказ — билеты привязываются к заказу, становятся `sold`, генерируется QR-код.
5. На входе билет отмечается временем `checked_in_at`.

**Нефункциональные требования:** надежность (предотвращение двойной продажи), производительность (пагинация и индексы), безопасность (CSRF, валидация, RBAC), сопровождаемость (FormRequest, тесты).

**Функциональные требования**

1. Управление постановками: создание, просмотр, редактирование, удаление.
2. Управление сеансами: создание сеансов для постановок.
3. Управление билетами: резервирование, продажа, отмена.
4. Оформление заказов и привязка билетов к заказу.
5. Генерация QR-кода при продаже билета.
6. Проверка (check-in) билета на входе.
7. Регистрация и аутентификация пользователей.

**Ограничения и допущения**

- Интеграция с реальными платежными провайдерами не реализована (модель данных предусмотрена).
- Сквозная аналитика по продажам вынесена в будущие улучшения.
- В рамках лабораторной работы допускается упрощенная роль администратора.

# 3. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ (ER-ДИАГРАММА)

В проекте подготовлены схемы: `theater_er_chen_conceptual.drawio`, `theater_er_chen_relational.drawio`, `theater_uml_class.drawio`.

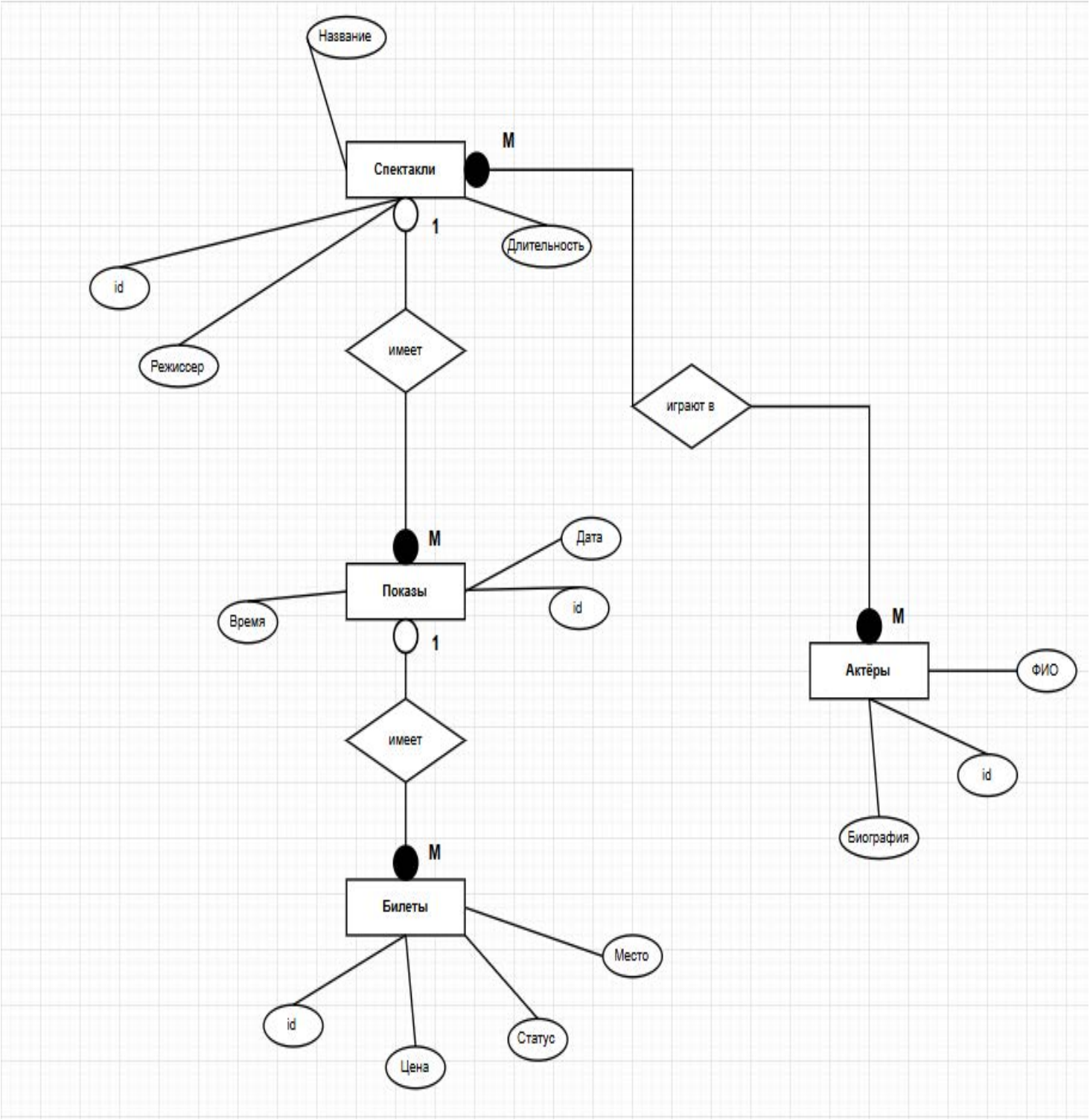



Рисунок 1 — Диаграмма нотации Чена (концептуальная).

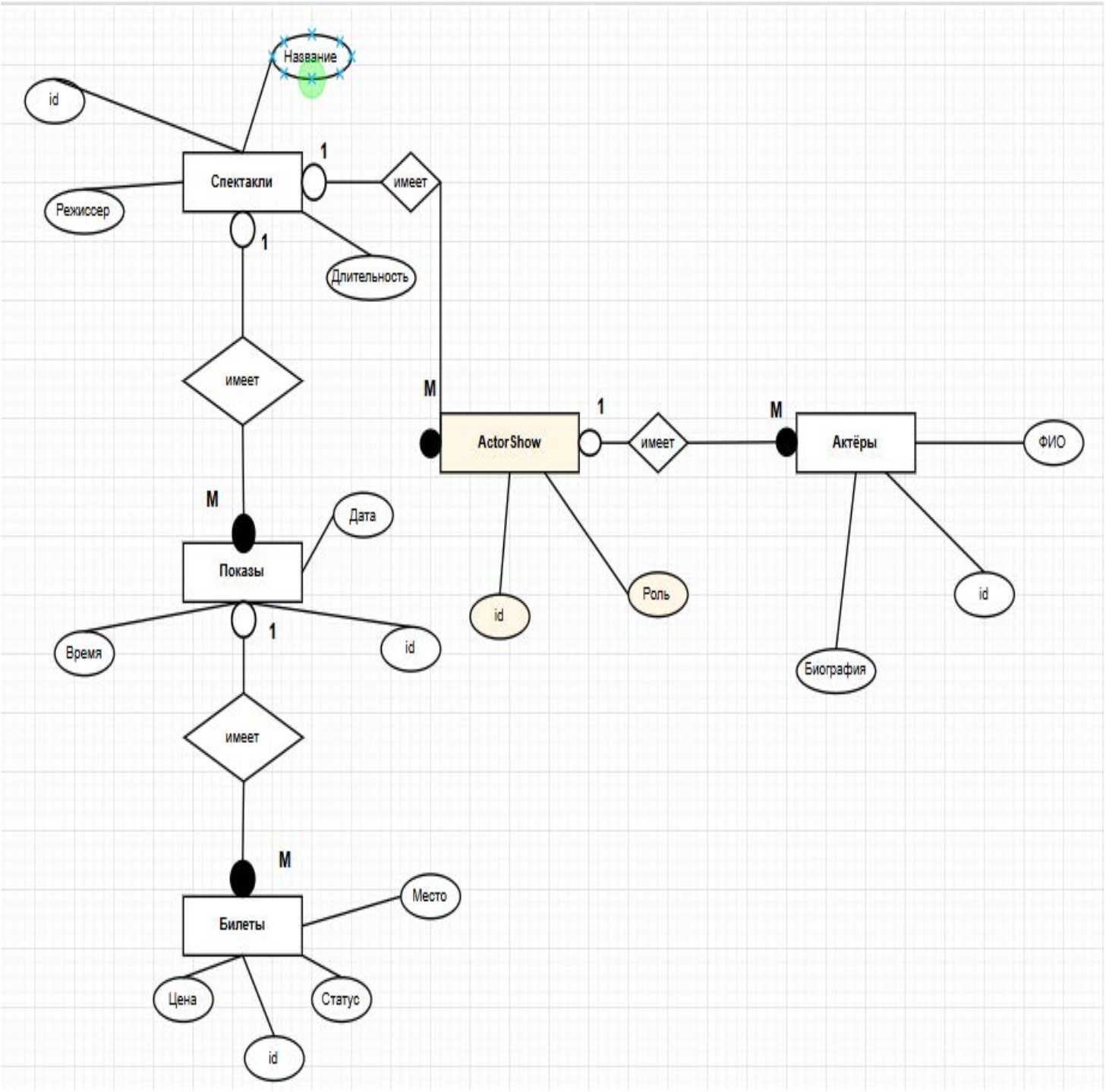



Рисунок 2 — Диаграмма нотации Чена (реляционная).

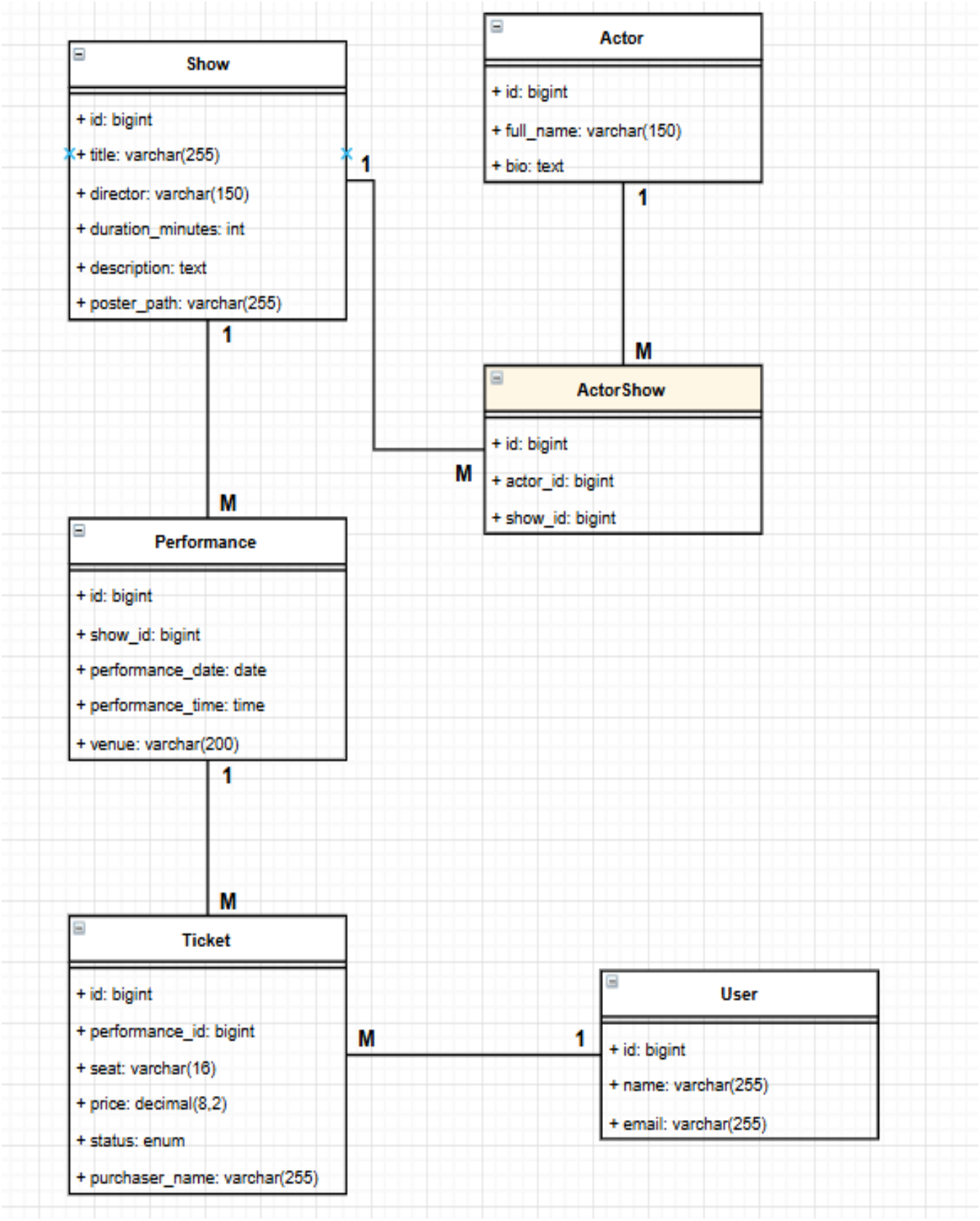



Рисунок 3 — ER-диаграмма (концептуальная)

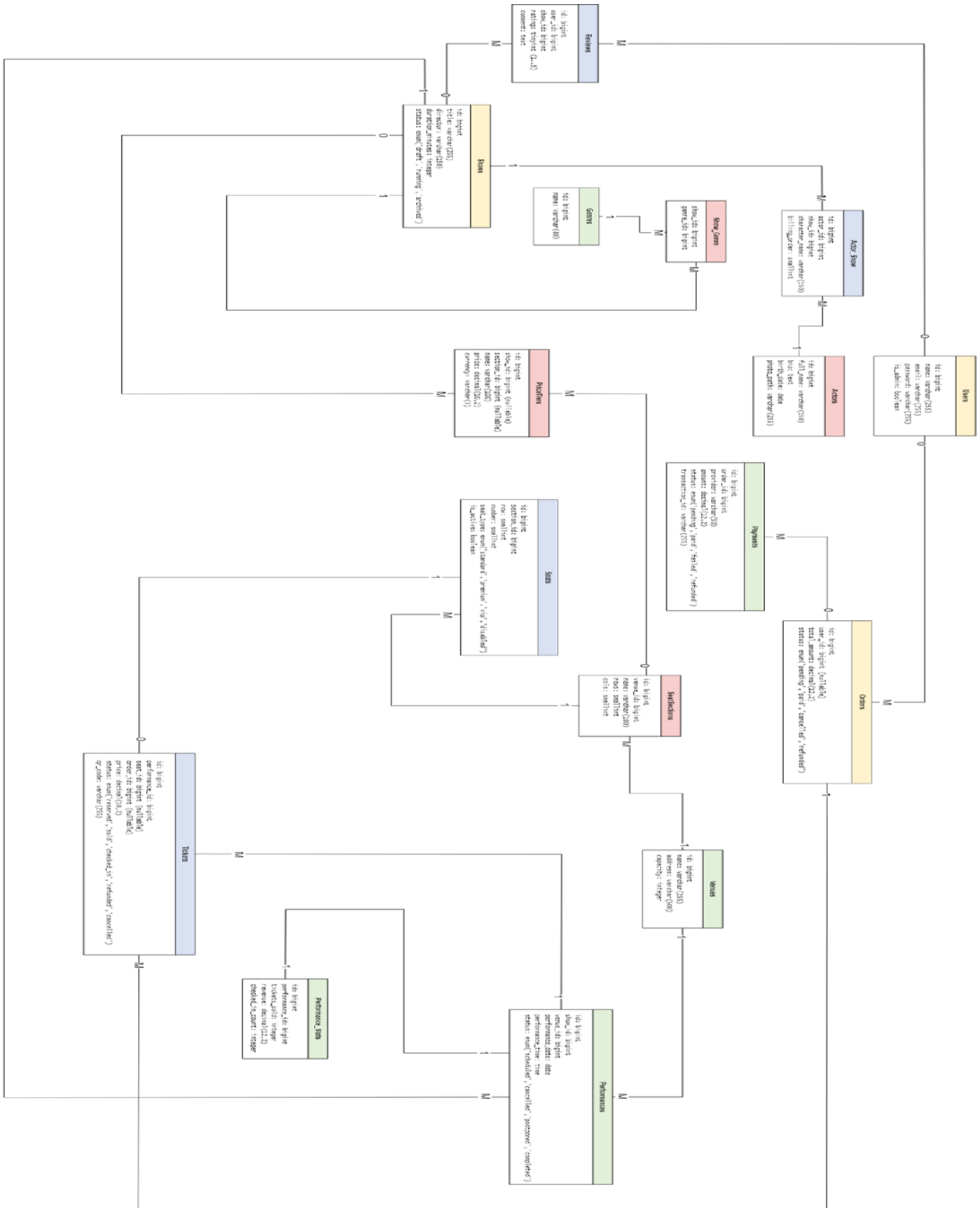



Рисунок 4 — ER-диаграмма (реляционная окончательная).

Таблица 1 — Словарь данных (основные таблицы)

| Таблица | Ключевые поля | Назначение |
|---|---|---|
| venues | id, name, address, capacity | Площадки/залы |
| shows | id, venue_id, title, director, description, duration_minutes, language, age_rating, poster_url | Постановки |
| performances | id, show_id, starts_at, status | Сеансы |
| seat_sections | id, venue_id, name | Секции зала |
| seats | id, section_id, row, number, seat_type, is_active | Места |
| price_tiers | id, venue_id?, show_id?, section_id?, amount_cents, currency | Цены |
| orders | id, user_id, total_amount, status | Заказы |
| tickets | id, performance_id, seat_id, order_id?, purchaser_id?, price, status, qr_code, issued_at?, checked_in_at? | Билеты |
| performance_stats | performance_id, tickets_sold, checked_in_count, revenue | Статистика по сеансам |

Таблица 2 — Детализированная структура таблиц

**Таблица `tickets`**

| Поле | Тип | Назначение | Примечание |
|---|---|---|---|
| id | bigint | Идентификатор билета | PK |
| performance_id | bigint | Сеанс | FK → performances.id |
| seat_id | bigint | Место | FK → seats.id |
| order_id | bigint | Заказ | FK → orders.id, nullable |
| purchaser_id | bigint | Покупатель | FK → users.id, nullable |
| price | decimal(10,2) | Цена билета | Рассчитана по price_tiers |
| status | string | Состояние | reserved/sold/cancelled |
| qr_code | string | QR-код | nullable |
| issued_at | timestamp | Время выдачи | nullable |
| checked_in_at | timestamp | Время прохода | nullable |

**Таблица `orders`**

| Поле | Тип | Назначение | Примечание |
|---|---|---|---|
| id | bigint | Идентификатор заказа | PK |
| user_id | bigint | Покупатель | FK → users.id |
| total_amount | decimal(10,2) | Сумма | >= 0 |
| status | string | Состояние | pending/paid/cancelled |

**Таблица `performances`**

| Поле | Тип | Назначение | Примечание |
|---|---|---|---|
| id | bigint | Идентификатор сеанса | PK |
| show_id | bigint | Постановка | FK → shows.id |
| starts_at | datetime | Дата/время | |
| status | string | Статус | optional |

**Таблица `seats`**

| Поле | Тип | Назначение | Примечание |
|---|---|---|---|
| id | bigint | Идентификатор места | PK |
| section_id | bigint | Секция | FK → seat_sections.id |
| row | string | Ряд | |
| number | string | Номер | |
| seat_type | string | Тип | standard/VIP |

# 4. ОПИСАНИЕ АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ

Приложение реализовано на Laravel 12.x и придерживается архитектуры MVC:

- **Model** — Eloquent-модели, отражающие сущности предметной области.
- **View** — Blade-шаблоны, отвечающие за отображение.
- **Controller** — обработка запросов, применение валидации и бизнес-логики, формирование ответов/редиректов.

**Слои приложения:** маршрутизация (`routes/web.php`), валидация (FormRequest), бизнес-логика (покупка/QR/check-in/обновление статистики), доступ (auth + admin middleware), представления (layout/partials).

**Технологии:** PHP 8.2, Laravel 12.x, Blade, Eloquent ORM, PHPUnit, SQLite (для тестов), GitHub Actions (CI).

**Структура проекта (кратко)**

- `app/Models` — модели Eloquent
- `app/Http/Controllers` — контроллеры
- `app/Http/Requests` — FormRequest-валидация
- `resources/views` — Blade-шаблоны
- `routes/web.php` — маршрутизация
- `database/migrations` — структура БД
- `database/factories` — фабрики данных для тестов
- `tests/Feature` — функциональные тесты

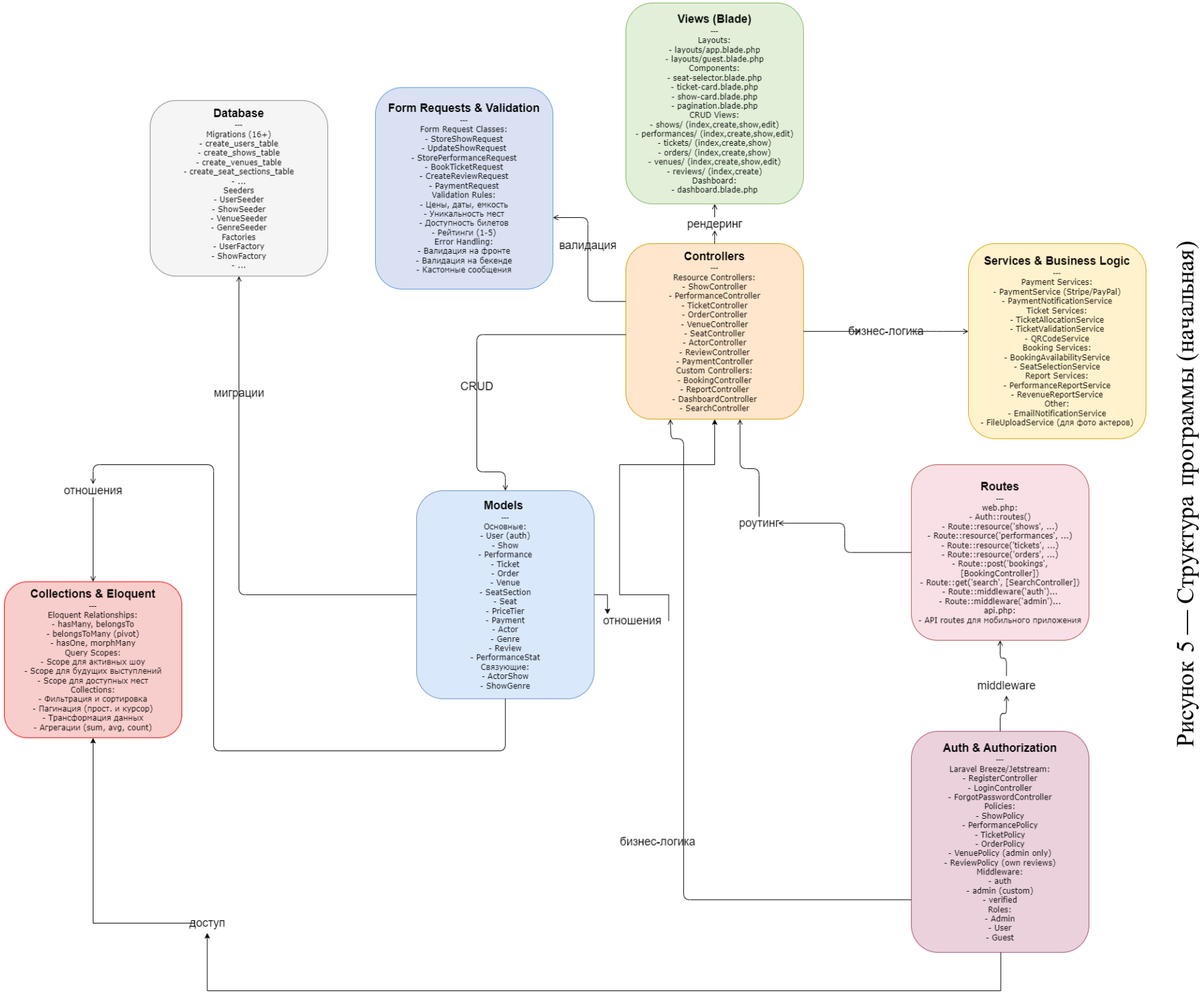



Рисунок 5 — Структура программы (начальная)

**Поток обработки запроса**

1. Запрос приходит по маршруту `web.php`.
2. Контроллер получает объект `Request`/`FormRequest`.
3. Валидация выполняется автоматически в FormRequest.
4. Контроллер вызывает модель, формирует ответ/редирект.
5. Ответ возвращается пользователю (Blade view или редирект).

**Требования к среде выполнения**

- ОС: Windows/Linux/macOS.
- PHP 8.2+.
- Composer.
- Node.js (для сборки фронтенда, при необходимости).
- База данных (SQLite/MySQL/PostgreSQL).

**Инструкция по запуску (кратко)**

1. Установить зависимости Composer.
2. Скопировать `.env.example` в `.env`, настроить БД.
3. Выполнить миграции и сиды.
4. Запустить приложение (`php artisan serve`).

Github: https://github.com/top-secret666/php

# 5. РЕАЛИЗАЦИЯ ОСНОВНЫХ КОМПОНЕНТОВ

## 5.1. Модели и отношения

Ключевые модели: Show, Performance, Ticket, Order, Seat, SeatSection, Venue, PriceTier, Payment, User.

Примеры отношений: Show принадлежит Venue; Performance принадлежит Show; Ticket принадлежит Performance/Seat/Order и содержит purchaser (User); Order принадлежит User и содержит Tickets.

Листинг 1 — Фрагмент (пример)

```
1.class Ticket extends Model
2.{
3.    protected $fillable =
  ['performance_id','seat_id','order_id','purchaser_id','pric
  e','status','qr_code','issued_at','checked_in_at'];
4.
5.    public function performance() { return $this-
  >belongsTo(Performance::class); }
6.    public function seat() { return $this-
  >belongsTo(Seat::class); }
7.    public function order() { return $this-
  >belongsTo(Order::class); }
8.}
```

## 5.2. Контроллеры и маршруты

Реализованы ресурсные контроллеры для CRUD: ShowController, ActorController, PerformanceController, TicketController, OrderController. Дополнительно: CartController (корзина/checkout) и Admin\StatsController (статистика).

**Маршруты:** `Route::resource` для основных сущностей, `/shows/search` для поиска, `/cart` и `/cart/checkout` для корзины, `/admin/stats` для статистики.

Листинг 2 — Фрагмент (пример)

```
1.Route::resource('shows', ShowController::class);
2.Route::resource('performances',
  PerformanceController::class);
3.Route::resource('tickets', TicketController::class);
```

4. `Route::resource('orders', OrderController::class);`

**Ключевые операции контроллеров:**

- `store` — создание сущности с валидацией.
- `update` — обновление с валидацией.
- `index` — пагинация и вывод списка.
- `show` — детализация сущност

### 5.3. Представления и формы

UI реализован на Blade. Используется единый layout (`resources/views/layouts/app.blade.php`) и частичные шаблоны (navbar). Формы защищены CSRF и показывают ошибки валидации.

**Пользовательский интерфейс (описание)**

- **Главная страница** — список постановок с постерами и ссылками.
- **Карточка шоу** — описание, длительность, доп. параметры.
- **Список сеансов** — дата/время, фильтрация.
- **Билеты** — просмотр статуса, привязка к заказам.
- **Заказы** — оформление и просмотр деталей.

**Описание бизнес-форм**

- Форма создания шоу: поля `title`, `description`, `duration_minutes`, `language`, `age_rating`, `venue_id`.
- Форма создания заказа: `user_id`, `total_amount`, `status`.
- Форма билета: `performance_id`, `seat_id`, `status`, `price` или `price_tier_id`.

**Описание ключевых экранов**

1. **Список постановок** — отображает карточки спектаклей, позволяет переходить к деталям.
2. **Карточка постановки** — описание спектакля, выбор сеансов.
3. **Список сеансов** — расписание показов с фильтрацией.
4. **Билеты** — список билетов пользователя/администратора с их статусом.
5. **Заказы** — список заказов и детализация по билетам.

### 5.4. Валидация данных

Валидация реализована через FormRequest: StoreShowRequest, StoreOrderRequest, StoreTicketRequest и др.

```
public function rules(): array
{
    return [
        // Для обычного пользователя user_id принудительно задается в контроллере.
        // Админ может указать user_id или оставить пустым.
        'user_id' => 'nullable|exists:users,id',
        'total_amount' => 'required|numeric|min:0',
        'status' => 'nullable|string',
    ];
}
```

**Причина выбора FormRequest:**

- единообразие правил;
- упрощение контроллеров;
- более простое тестирование правил.

### 5.5. Аутентификация и авторизация

Аутентификация: регистрация/логин/логаут. Авторизация реализована через middleware `auth` и `AdminMiddleware`, а также проверками владения данными в контроллерах заказов и билетов.

**RBAC:** администратор управляет сущностями и статистикой; пользователь покупает билеты и видит только свою корзину и свои заказы.

# 6. АЛГОРИТМЫ И БИЗНЕС-ЛОГИКА

**Резервирование билета:** пользователь выбирает сеанс и место, создаётся ticket со статусом `reserved`. Надёжность обеспечивается уникальностью пары `performance_id + seat_id`.

**Покупка (оформление заказа):** создаётся order с суммой `total_amount`, билеты привязываются к заказу и переводятся в `sold`, генерируется QR-код, фиксируется `issued_at`.

**Корзина:** пользователь собирает reserved-билеты, затем выполняет checkout одним действием.

**Check-in:** при проходе устанавливается `checked_in_at`. После операций продажи/проверки обновляется `performance_stats` (продано/проверено/выручка).

### Алгоритм резервирования билета

1. Пользователь выбирает сеанс и место.
2. Система проверяет уникальность пары `performance_id` + `seat_id`.
3. Создается запись в `tickets` со статусом `reserved`.

### Алгоритм покупки билета

1. Создается заказ (`orders`) с суммой `total_amount`.
2. Билет связывается с заказом (`order_id`).
3. При смене статуса на `sold` генерируется QR-код.

### Алгоритм check-in (проверка билета)

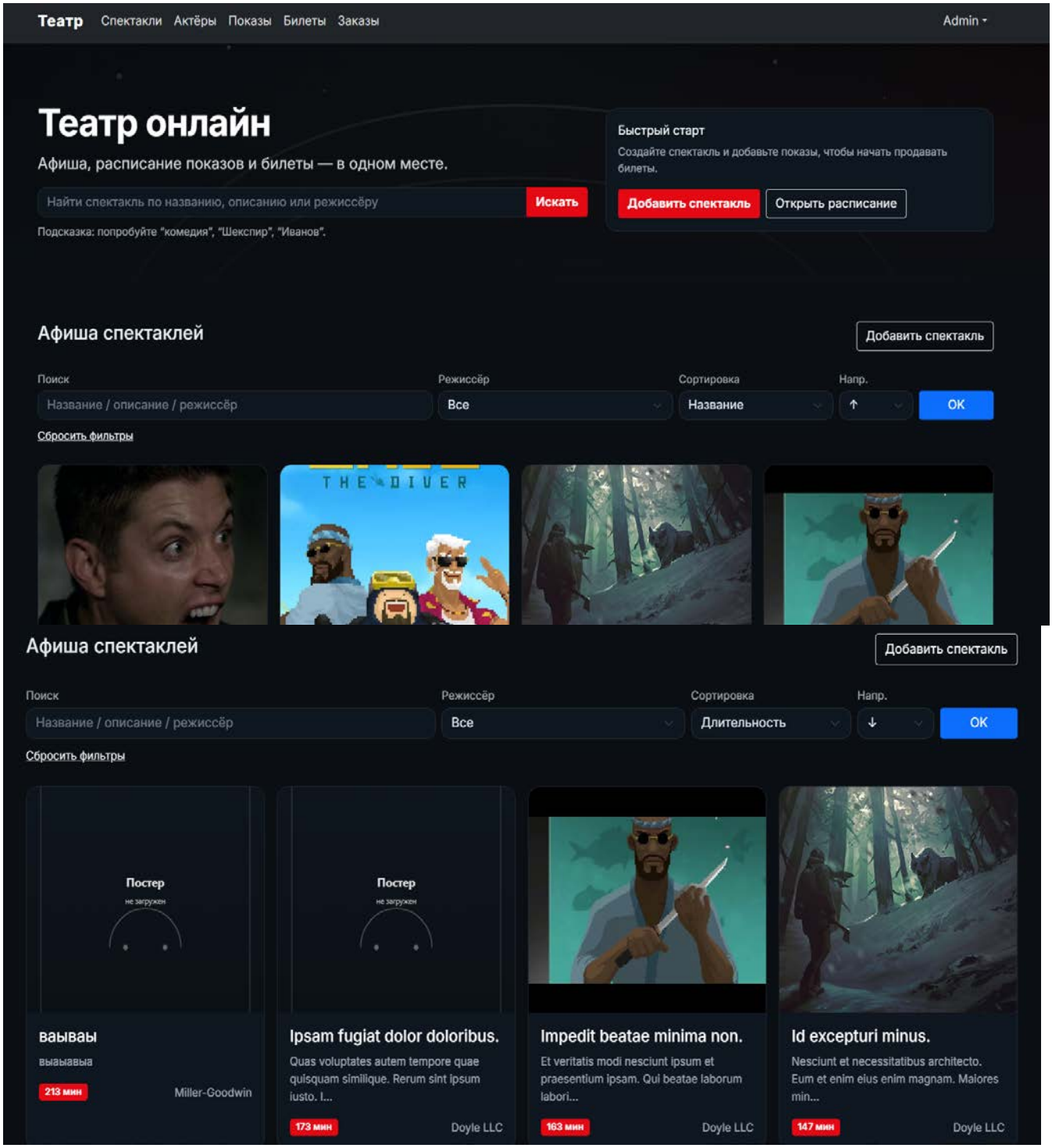
1. Контроллер принимает запрос на обновление билета.
2. Валидация включает дату `checked_in_at`.
3. Время прохода записывается в поле `checked_in_at`.

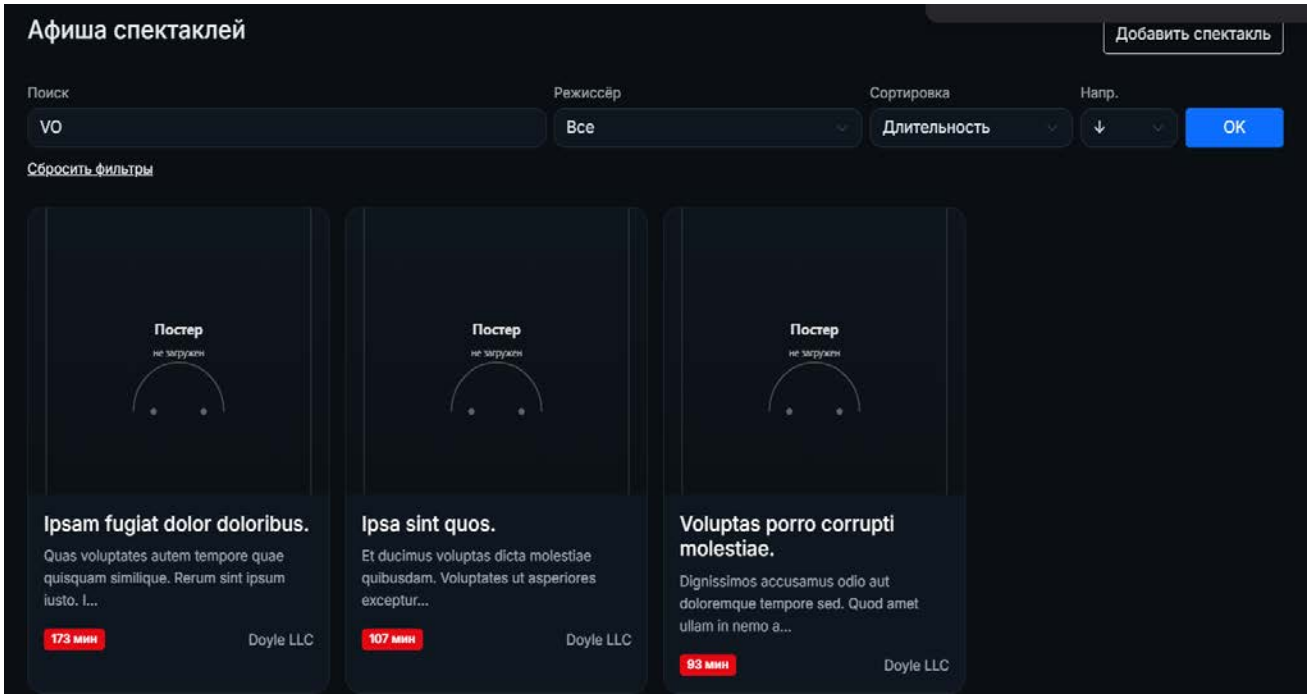
### Обработка цен

Цена хранится в `price_tiers.amount_cents`. При создании билета она переводится в дробный формат:
`price = amount_cents / 100`.

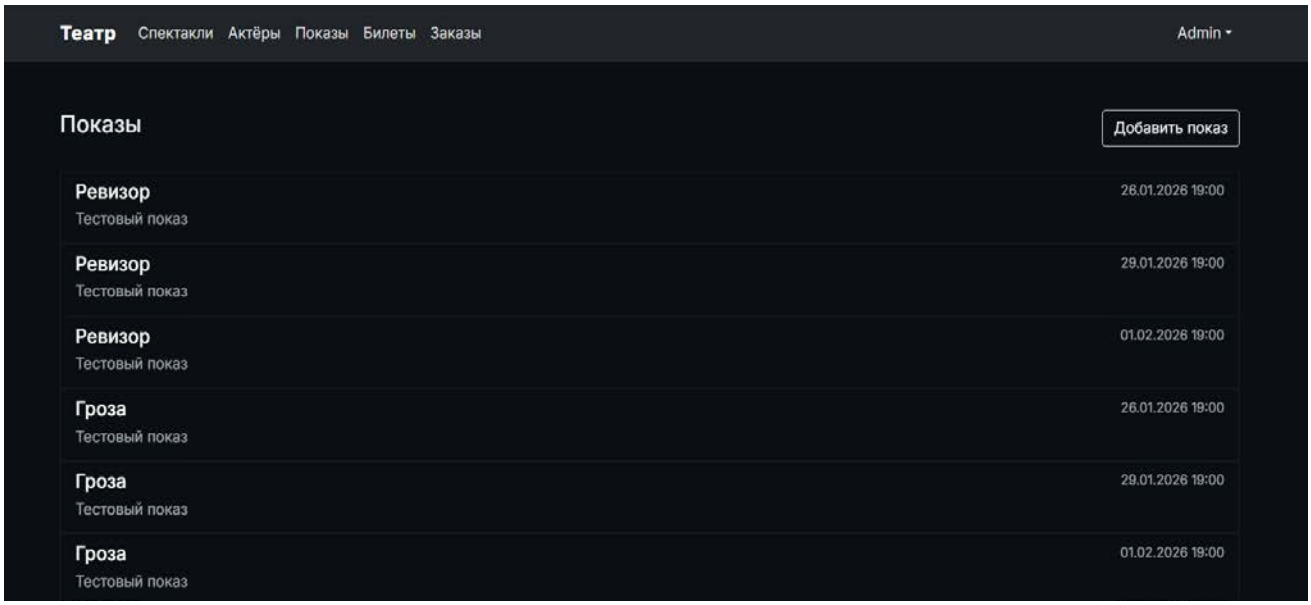
# 7. СКРИНШОТЫ РАБОТЫ ПРИЛОЖЕНИЯ

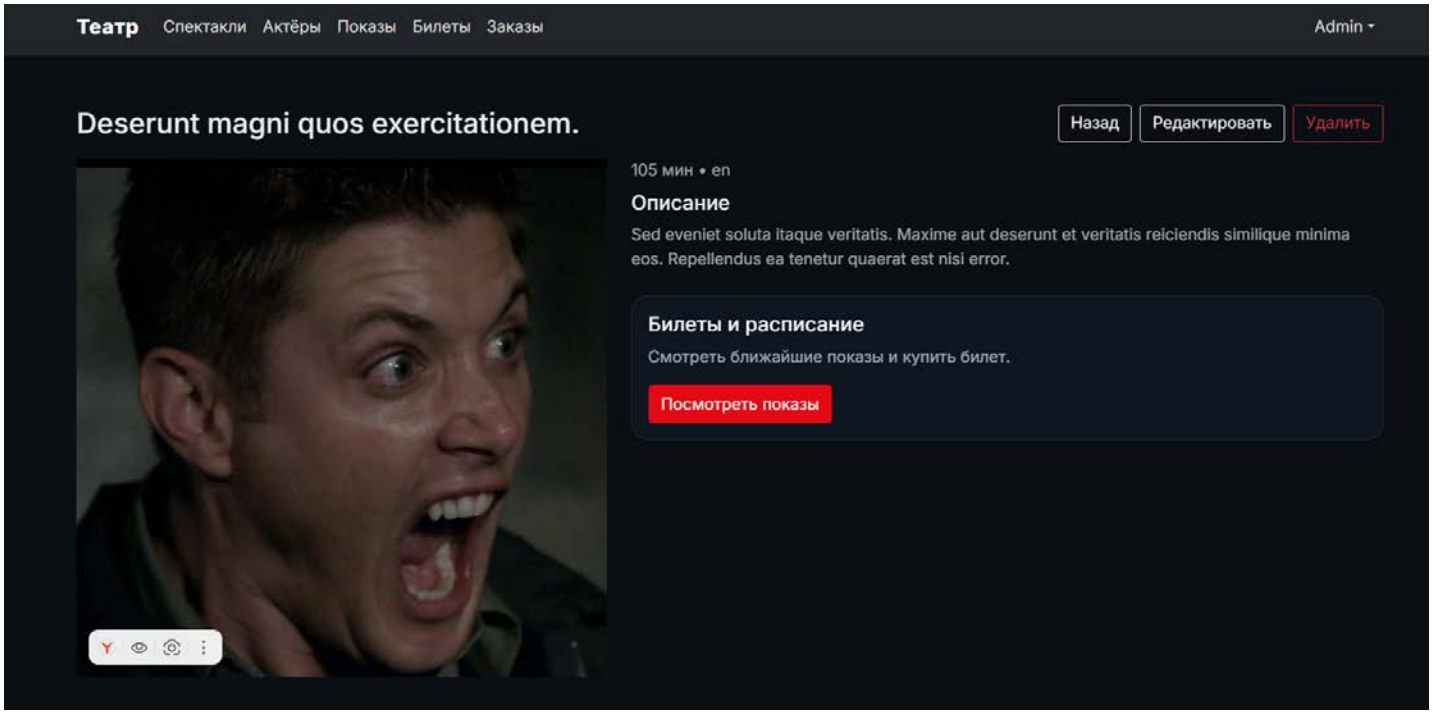
- Афиша (список постановок) и поиск/фильтры.



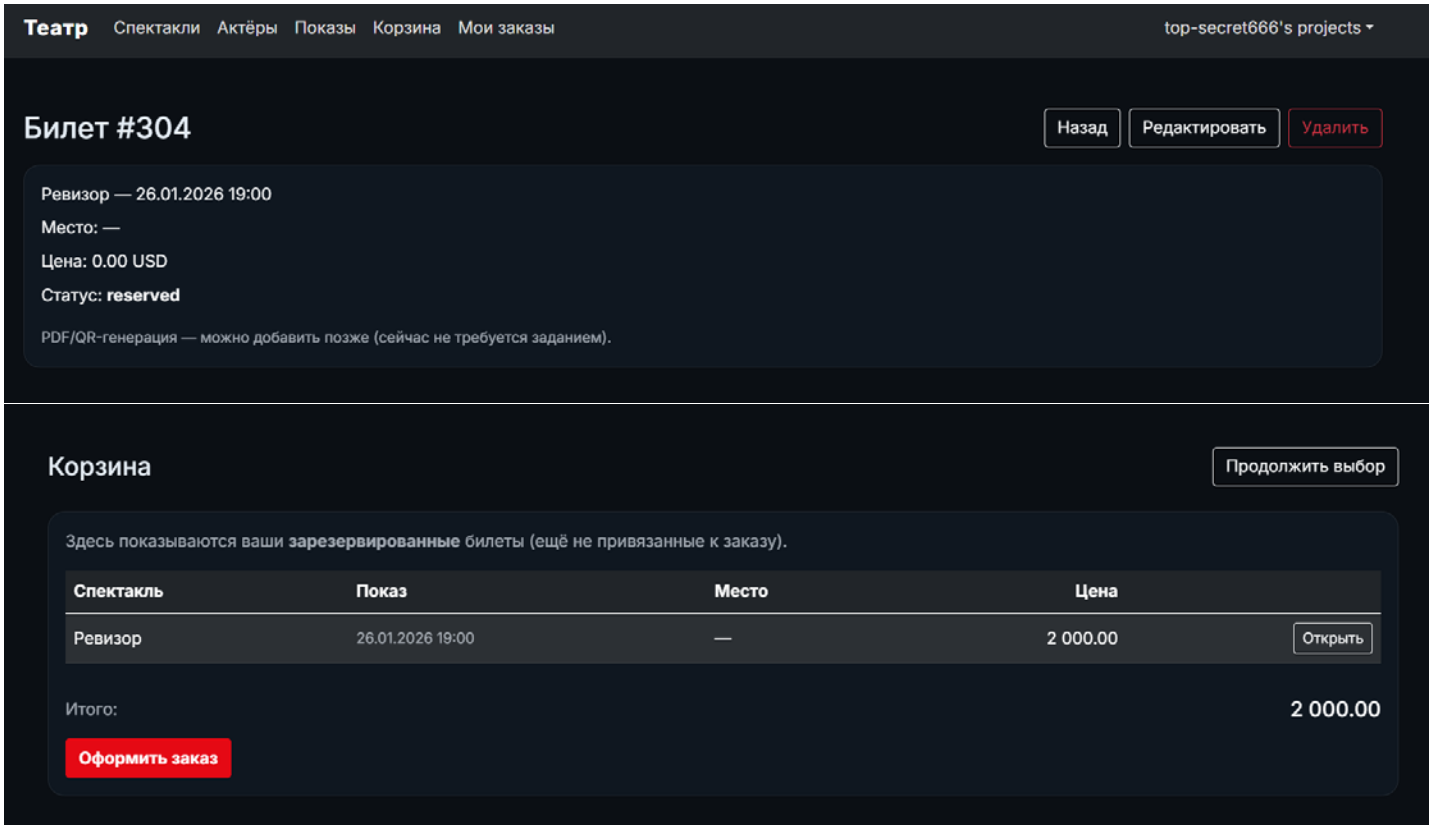



- Карточка постановки и расписание сеансов.

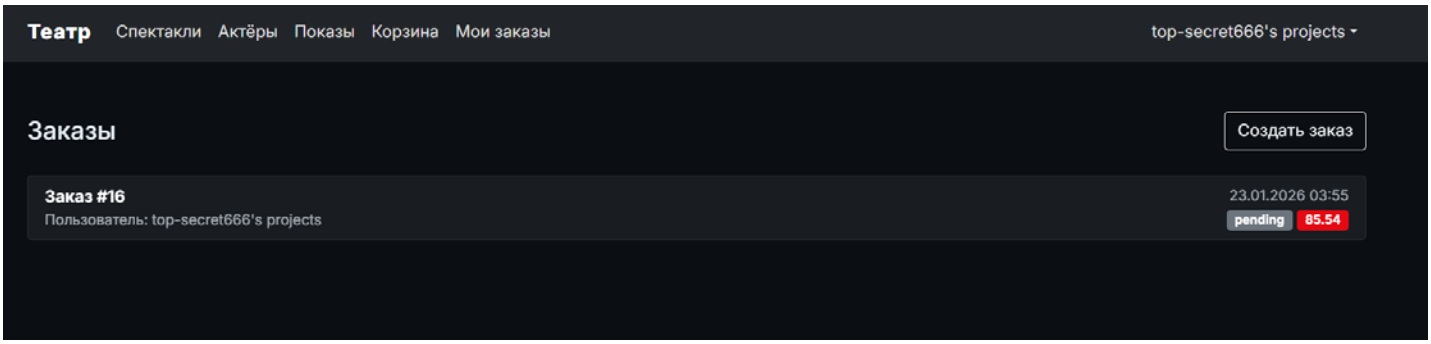




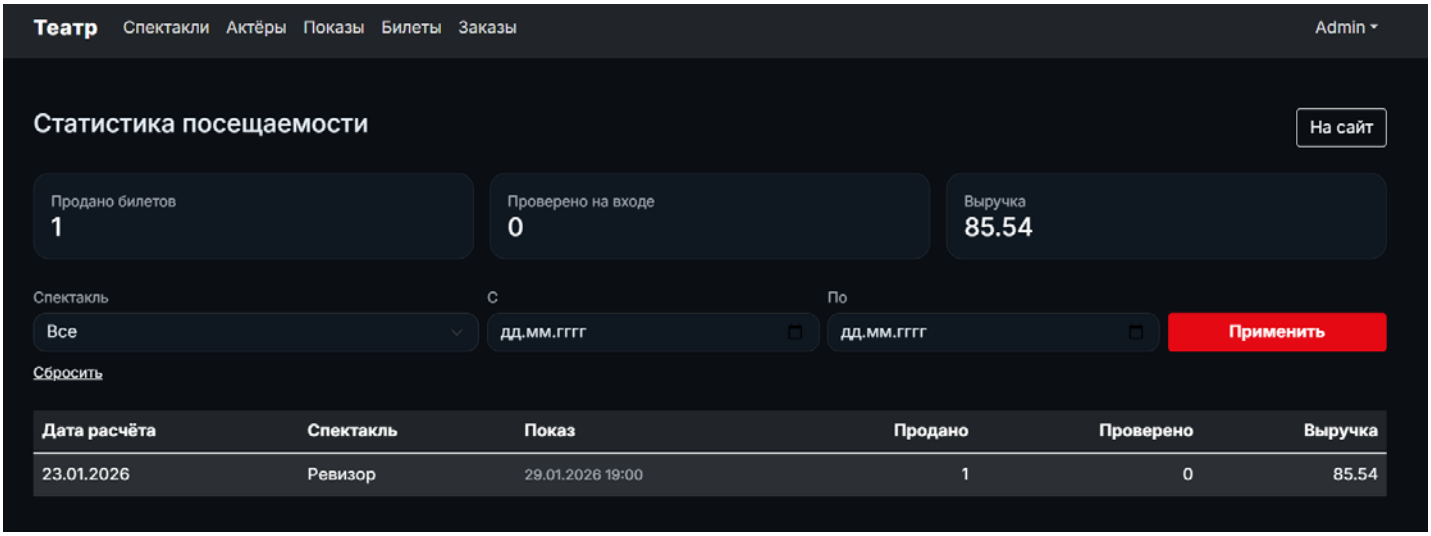

- Покупка/создание  билета, корзина.

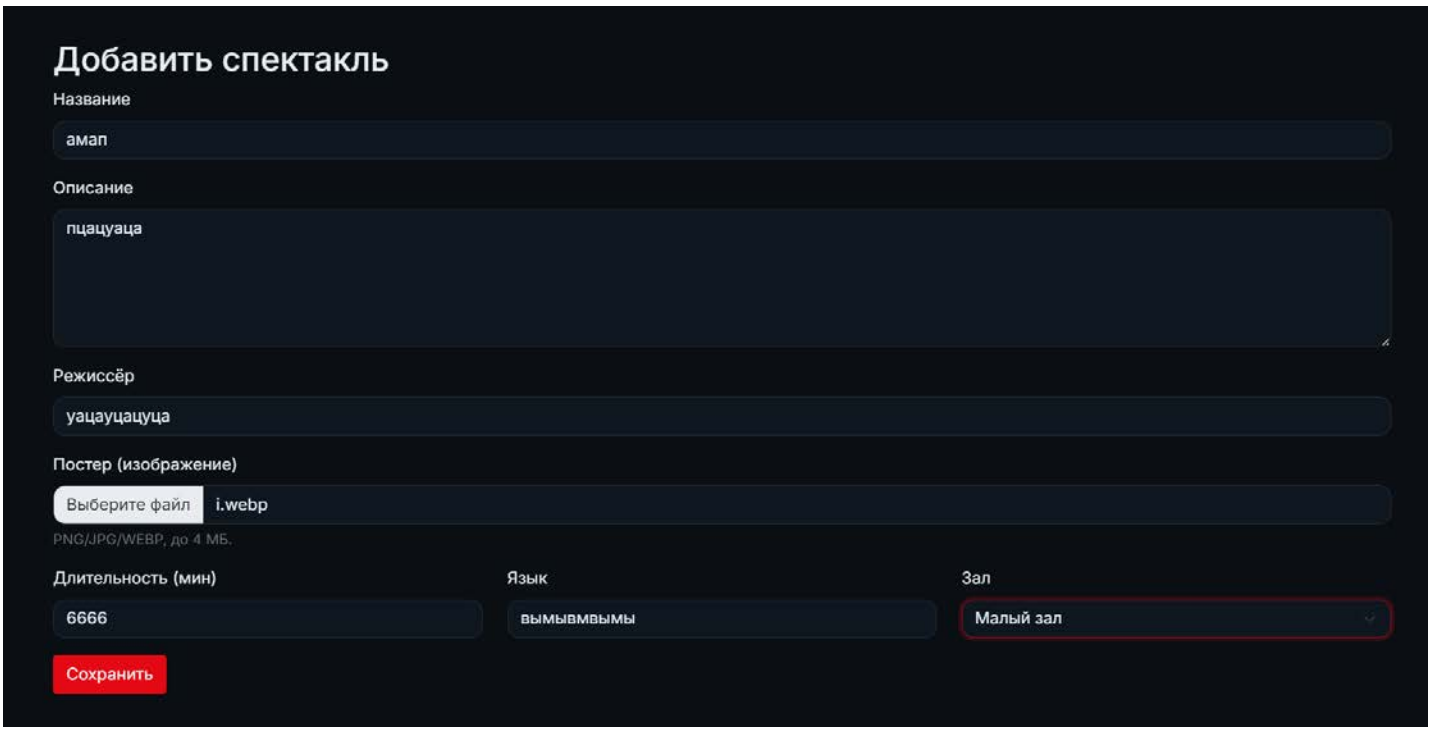


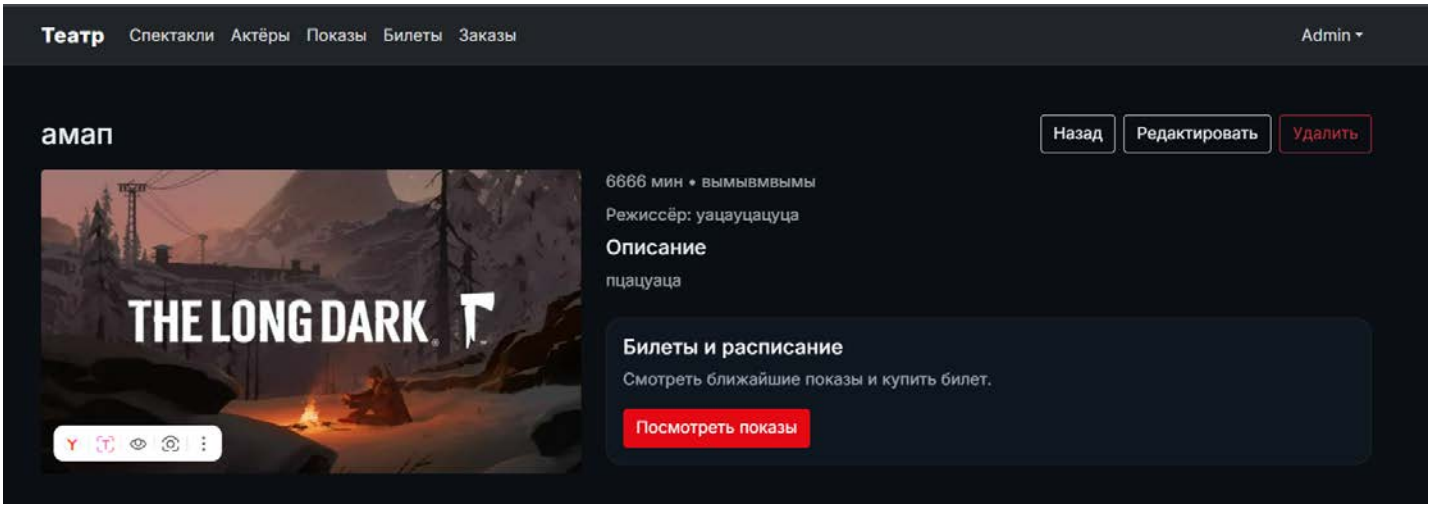

- Мои заказы и просмотр заказа.



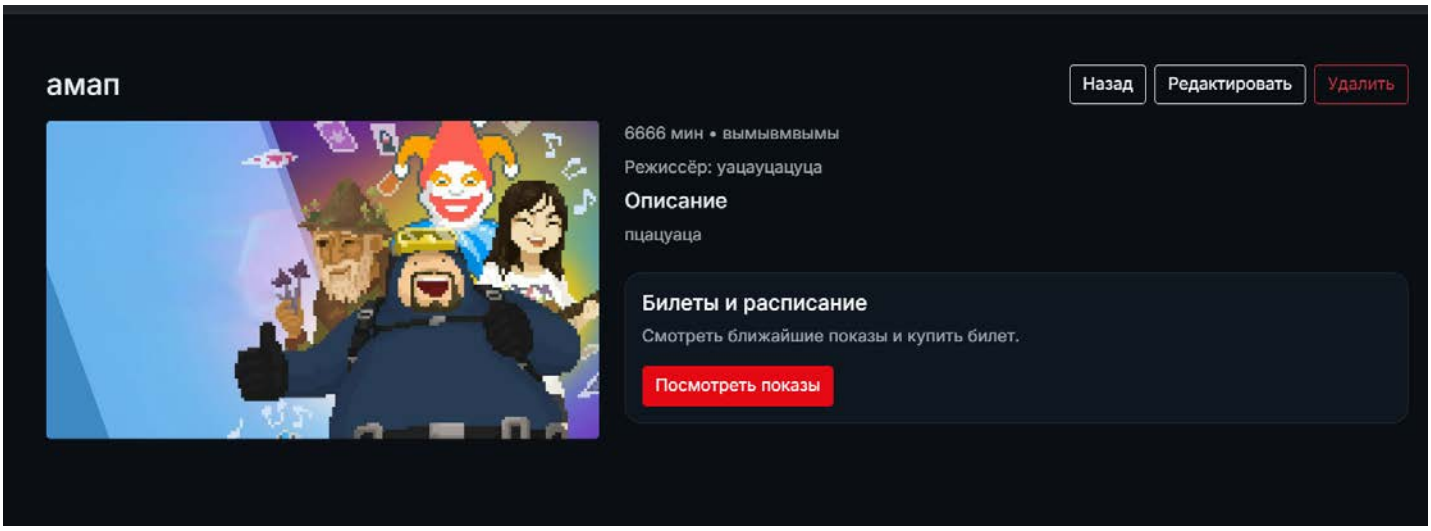

- Админ-страницы: управление сущностями, статистика.

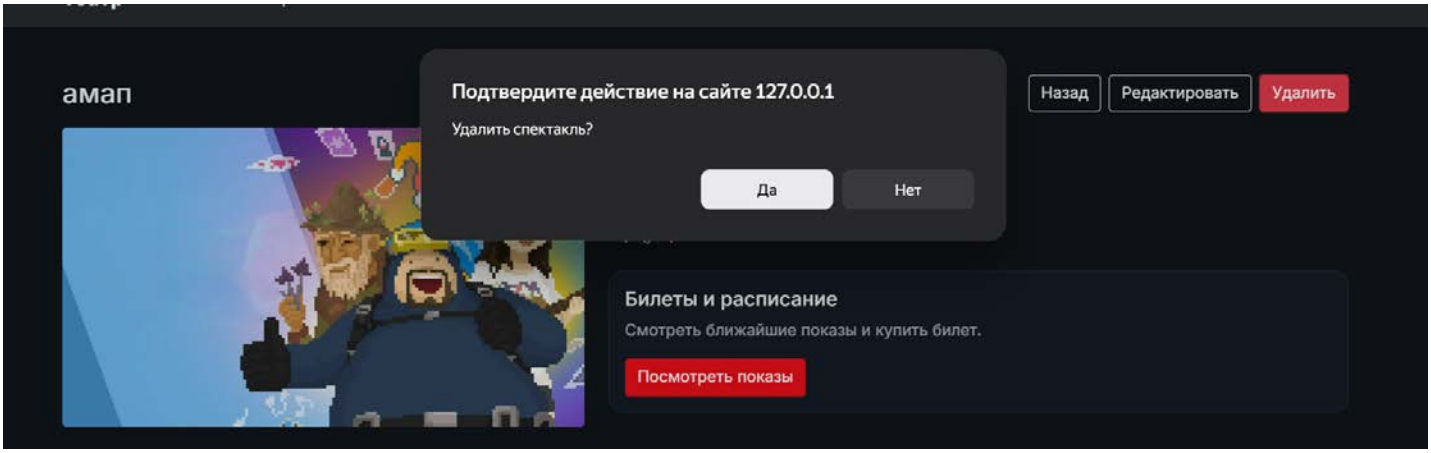

Театр
Спектакли
Актёры
Показы
Билеты
Заказы
Admin
Статистика посещаемости
На сайт
Продано билетов
1
Проверено на входе
0
Выручка
85.54
Спектакль
Все
С
дд.мм.гггг
По
дд.мм.гггг
Применить
Сбросить
Дата расчёта
Спектакль
Показ
Продано
Проверено
Выручка
23.01.2026
Ревизор
29.01.2026 19:00
1
0
85.54

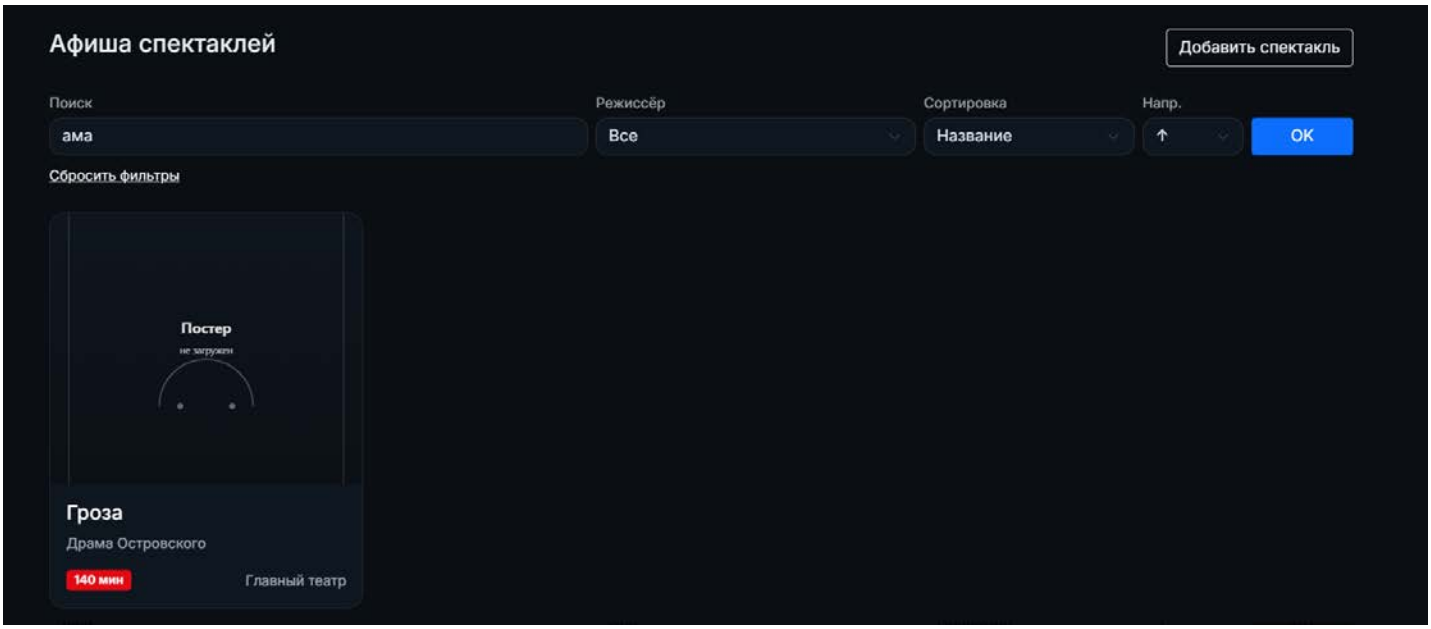

Добавить спектакль
Название
амап
Описание
пцацуаца
Режиссёр
уацауцацуца
Постер (изображение)
Выберите файл
i.webp
PNG/JPG/WEBP, до 4 МБ.
Длительность (мин)
6666
Язык
вымывмвымы
Зал
Малый зал
Сохранить


Театр
Спектакли
Актёры
Показы
Билеты
Заказы
Admin
амап
Назад
Редактировать
Удалить
THE LONG DARK
6666 мин • вымывмвымы
Режиссёр: уацауцацуца
Описание
пцацуаца
Билеты и расписание
Смотреть ближайшие показы и купить билет.
Посмотреть показы


Редактировать спектакль
Название
амап
Описание
пцацуаца
Режиссёр
уацауцацуца


амап
Назад
Редактировать
Удалить
6666 мин • вымывмвымы
Режиссёр: уацауцацуца
Описание
пцацуаца
Билеты и расписание
Смотреть ближайшие показы и купить билет.
Посмотреть показы


амап
Подтвердите действие на сайте 127.0.0.1
Удалить спектакль?
Да
Нет
Назад
Редактировать
Удалить
Билеты и расписание
Смотреть ближайшие показы и купить билет.
Посмотреть показы


Афиша спектаклей
Добавить спектакль
Поиск
ама
Режиссёр
Все
Сортировка
Название
Напр.
↑
OK
Сбросить фильтры
Постер
не загружен
Гроза
Драма Островского
140 мин
Главный театр

# 8. ЛИСТИНГ КЛЮЧЕВЫХ ФРАГМЕНТОВ КОДА

**Генерация QR-кода при продаже:**

```
if (isset($data['status']) && $data['status'] === 'sold' && empty($data['qr_code'])) {
    $data['qr_code'] = bin2hex(random_bytes(8));
}
```

**Привязка цены к ценовому уровню:**

```
$tier = PriceTier::find($data['price_tier_id']);
$data['price'] = $tier->amount_cents / 100;
```

**Фрагмент админ-middleware:**

```
if (!$user || !$user->is_admin) {
    abort(403);
}
```

Листинг 3 — Проверка билета:

```
1.'ticket_id' => 'required|exists:tickets,id'
```

Листинг 4 — Обновление времени проверки билета:

```
1.'checked_in_at' => 'nullable|date'
```

Листинг 5 — Переход билета в статус «продан»:

```
1.$ticket->update([
2.    'status' => 'sold',
3.    'order_id' => $order->id,
4.]);
```

Листинг 6 — Пример теста проверки билета:

```
1.$response = $this->actingAs($user)-
  >put(route('tickets.update', $ticket), [
2.    'performance_id' => $ticket->performance_id,
3.    'seat_id' => $ticket->seat_id,
4.    'status' => 'sold',
5.    'checked_in_at' => now()->toDateTimeString(),
6.]);
```

Листинг 7 — Создание заказа (контроллер):

```
$order = Order::create([
    'user_id' => $request->user_id,
    'total_amount' => $request->total_amount,
    'status' => $request->status ?? 'pending',
]);
```

Листинг 8 — Создание шоу (контроллер):

```
$show = Show::create($request->validated());
```

# 9. ТЕСТИРОВАНИЕ И РЕЗУЛЬТАТЫ

Используются Feature-тесты (RefreshDatabase), фабрики данных и SQLite в тестовой среде. Проверены сценарии регистрации/логина, CRUD (админ), поиск/фильтрация/пагинация, загрузка постера, покупка/оформление заказа, генерация QR, check-in и разграничение доступа.

**Проверка соответствия требованиям задания**

| Требование | Реализация в проекте |
|---|---|
| Laravel 12.x+, PHP 8.1+ | Laravel 12.x в каталоге `theater_full/`, PHP 8.2 в CI (`.github/workflows/ci.yml`). |
| Не менее 3 связанных моделей | `Show`, `Performance`, `Ticket`, `Order`, `Venue` и др. Связи: `Show->performances()`, `Ticket->performance()`, `Order->tickets()` и т.д. |
| Миграции для таблиц | `theater_full/database/migrations/*` (создание `shows`, `performances`, `tickets`, `orders` и др.). |
| Отношения Eloquent | `theater_full/app/Models/*.php` (`hasMany`, `belongsTo`, `belongsToMany`). |
| Resource-контроллеры | `Route::resource(...)` в `theater_full/routes/web.php` для шоу/сеансов/билетов/заказов. |
| CRUD для актёров | `ActorController` + `Route::resource('actors', ...)` + шаблоны `resources/views/actors/*`. Роль и спектакль хранятся в pivot `actor_show.character_name`. |
| Доп. контроллеры/логика | `TicketController` — продажа/QR/статусы, статистика; `AdminMiddleware` — доступ администратора. |
| Представления CRUD | `theater_full/resources/views/*` (например, `shows/*`, `orders/*`, `tickets/*`). |
| Макет (layout) | `theater_full/resources/views/layouts/app.blade.php`. |
| Blade-компоненты | `theater_full/resources/views/components/*` (используются для переиспользуемых UI-фрагментов). |
| Формы создания/редактирования | `theater_full/resources/views/shows/create.blade.php`, `edit.blade.php` и аналоги для сущностей. |
| Серверная валидация | FormRequest: `theater_full/app/Http/Requests/*` (например, `StoreShowRequest`). |

| Требование | Реализация в проекте |
|---|---|
| Отображение ошибок | В формах используется `@error(...)` и `invalid-feedback` (Bootstrap-паттерн). |
| Коллекции | Пример: построение списка режиссёров для фильтра — `ShowController::getFilteredShows()` (работа с `Collection`). |
| Поиск/фильтрация/сортировка | Афиша: `theater_full/resources/views/shows/index.blade.php` + `ShowController` (параметры `q`, `director`, `sort`, `dir`). |
| Пагинация | `paginate(15)` в контроллерах и `{{ $paginator->links() }}` во view (например, афиша). |
| Аутентификация | Логин/регистрация (контроллеры в `theater_full/app/Http/Controllers/Auth/*`, маршруты в `theater_full/routes/web.php`). |
| Авторизация/разграничение | `AdminMiddleware` + защищённые маршруты `admin/*`. |
| Статистика посещаемости | `admin/stats` — просмотр агрегатов из `performance_stats` (продано/проверено/выручка). |
| Загрузка файлов | Постер спектакля: формы `shows/create |

**Поиск, фильтрация, сортировка и пагинация (афиша)**

В приложении реализован список спектаклей (афиша) с обработкой параметров запроса.

**Маршруты:**

- `GET /shows` — список спектаклей;
- `GET /shows/search` — тот же список, но используется как endpoint для формы поиска.

**Параметры фильтрации и сортировки:**

- `q` — строка поиска по полям: `title`, `description`, `director`;
- `director` — точный фильтр по режиссёру;
- `sort` — сортировка (`title`, `duration_minutes`, `created_at`);
- `dir` — направление сортировки (`asc`/`desc`).

**Техническая реализация:**

- логика находится в `ShowController::getFilteredShows()`;

- используется `paginate(15)` и `appends($request->query())`, чтобы параметры сохранялись при переходе по страницам;
- поиск выполнен через `like` (совместимо с SQLite/MySQL/PostgreSQL), чтобы не зависеть от `ilike`.

**Загрузка файлов (постер спектакля)**

Для предметной области театра логично хранить постер спектакля. В проекте реализована загрузка изображения с сохранением в файловое хранилище.

**Где реализовано:**

- формы: `resources/views/shows/create.blade.php`, `resources/views/shows/edit.blade.php` (`enctype="multipart/form-data"`, поле `poster`);
- валидация: `app/Http/Requests/StoreShowRequest.php` (`poster` — `image|max:4096`);
- обработка: `app/Http/Controllers/ShowController.php` (сохранение на диске `public`, запись URL в поле `poster_url`).

**Примечание:** для раздачи файлов используется `Storage::url(...)`. Для локальной разработки требуется создать публичную ссылку на каталог хранения:

- выполнить `php artisan storage:link`;
- после этого файлы, сохранённые в `storage/app/public`, будут доступны через `/storage/...`.

**Использование коллекций (Collections)**

В Laravel коллекции применяются для удобной обработки наборов данных и подготовки их к отображению.

В проекте коллекции используются, например, для построения списка доступных режиссёров в фильтре афиши:

- выбираются уникальные значения `director` из таблицы `shows`;
- результат превращается в `Collection` через `pluck('director')->values()`;
- далее коллекция используется в Blade-шаблоне для формирования `<select>`.

Это демонстрирует применение коллекций для подготовки данных к фильтрации (в UI) без ручной обработки массивов.

## Тестирование

**Подход:**

- Используются `Feature`-тесты на базе `RefreshDatabase`.
- Применяются фабрики данных для генерации тестовых сущностей.

**Ключевые проверенные сценарии:**

- Регистрация пользователя.
- CRUD для шоу.
- Поиск/фильтрация/сортировка афиши.
- Загрузка постера спектакля.
- Просмотр списка сеансов.
- Покупка заказа и выдача билета.
- Проверка билета (установка `checked_in_at`).

**Результаты:**

- Все тесты проходят успешно (45 тестов, 114 утверждений).

# 10. CI/CD И АВТОМАТИЗАЦИЯ

Настроен GitHub Actions workflow, который выполняет установку зависимостей, подготовку окружения, миграции и сиды, после чего запускает тесты. Такой подход обеспечивает воспроизводимость и контроль качества на уровне репозитория.

**Ключевые шаги CI:**

- запуск job в каталоге `theater_full/` (там расположен `composer.json` и `artisan`);
- установка зависимостей через Composer;
- подготовка окружения: копирование `.env.example` → `.env`, создание SQLite файла `database/database.sqlite`;
- генерация ключа приложения (`php artisan key:generate`);
- миграции и сиды (`php artisan migrate:fresh --seed --force`);
- запуск тестов (`php artisan test`).

**Почему это важно:**

- исключает ситуацию, когда CI запускается не из директории Laravel-проекта и не находит `composer.json`;
- обеспечивает воспроизводимость проверок на чистой базе данных (SQLite), что соответствует требованиям задания.

## Охрана труда и безопасность

При разработке программного продукта учитываются базовые требования охраны труда:

- соблюдение регламентов рабочего времени при длительной работе за ПК;
- эргономичная организация рабочего места (освещение, расстояние до монитора);
- регулярные перерывы для профилактики утомления.

В части информационной безопасности:

- защита форм от CSRF;
- валидация входных данных;
- ограничения доступа через middleware.
-

## Экономическое обоснование

Внедрение системы снижает трудозатраты на продажу билетов, уменьшает ошибки учета и повышает доступность сервиса.
Экономический эффект выражается в:

- снижении времени на обработку заказов;
- росте продаж за счет онлайн-канала;
- уменьшении расходов на бумажный документооборот.

# 11. ЗАКЛЮЧЕНИЕ

В результате выполнена разработка веб-приложения «Театральная касса» на Laravel 12.x, реализованы основные сущности и связи, CRUD-интерфейсы (для администратора), пользовательские сценарии покупки (включая корзину), серверная валидация, аутентификация и разграничение ролей. Добавлены загрузка постеров и статистика по сеансам. Проект покрыт функциональными тестами и имеет CI/CD.

В рамках работы создано полноценное веб-приложение «Театральная касса», включающее:

- проектирование структуры БД и реализацию миграций;
- реализацию CRUD-операций для основных сущностей;
- валидацию данных через FormRequest;
- базовую авторизацию;
- функциональные тесты для ключевых сценариев.

Дополнительно реализованы элементы, которые часто являются обязательными при проверке по чек-листу:

- поиск по данным и фильтрация/сортировка (афиша);
- пагинация списков;
- загрузка файлов (постер спектакля);
- автоматическая проверка в GitHub Actions.

Проект можно расширять: добавить интеграцию с платежными системами, роль администратора, генерацию отчетности и advanced-аналитику.

Дополнительные направления развития:

- генерация электронных билетов в PDF;
- интеграция с реальными платежными провайдерами;
- модуль рассадки и интерактивная карта зала;
- административная панель и отчеты по продажам.

# 12. СПИСОК ЛИТЕРАТУРЫ

1. Документация Laravel. URL: https://laravel.com/docs (дата обращения: 23.01.2026).

2. Документация PHP. URL: https://www.php.net/docs.php (дата обращения: 23.01.2026).

3. Документация PHPUnit. URL: https://phpunit.de/documentation.html (дата обращения: 23.01.2026).

4. diagrams.net (draw.io). URL: https://www.diagrams.net/ (дата обращения: 23.01.2026).

5. Date C.J. An Introduction to Database Systems. (общая теория баз данных).