**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕБЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«Рязанский государственный радиотехнический университет  
имени В. Ф. Уткина»**

(ФГБОУ ВО «Рязанский государственный радиотехнический университет им. В.Ф. Уткина»,

ФГБОУ ВО «РГРТУ», РГРТУ)

**Кафедра вычислительной и прикладной математики**

**ПРОЕКТИРОВАНИЕ ПРОГРАММНОЙ СИСТЕМЫ ПОИСКА МЕРОПРИЯТИЙ И ПОКУПКИ БИЛЕТОВ**

Пояснительная записка к курсовому проекту  
по дисциплине  
«Проектирование информационных систем»

Выполнил: студент гр. 044  
Лутиков В. А.

Проверил: д-р техн. наук,  
профессор каф. ВПМ   
Белов В.В.

Рязань 2023

ЗАДАНИЕ

на выполнение курсового проекта по дисциплине

**«Проектирование информационных систем»**

|  |  |  |
| --- | --- | --- |
| ст. гр. | 044 | Лутиков Вадим Александрович |
|  |  | ФИО |

**Тема проекта:**

**ПРОЕКТИРОВАНИЕ ПРОГРАММНОЙ СИСТЕМЫ ПОИСКА МЕРОПРИЯТИЙ И ПОКУПКИ БИЛЕТОВ**

**Замечания:**

1. предметная область, объект и предмет автоматизации выбираются самостоятельно (в качестве основы для выбора следует использовать тему квалификационного исследования, может использоваться и задание на ранее выполненные курсовые проекты, например, по дисциплине «Базы данных», можно использовать и задания из методички Каюмовой);
2. при автоматизации информационных процессов реализуемая система должна реализовывать типичные CRUD-операции с базами данных (Create – ввод, Read – чтение, Update – изменение, Delete – удаление) и решать конкретные прикладные задачи;
3. при реализации относительно самостоятельных аналитических и/или игровых процессов рекомендуется реализовать инфраструктурное информационное обеспечение, например, в виде системы учета пользователей программы;
4. при реализации относительно самостоятельных технических процессов следует тщательно обосновать целесообразность нового программного решения и осуществить тщательное тестирование созданной программы;
5. методология проектирования функционально-ориентированная или объектно-ориентированная выбирается самостоятельно.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Задание выдано | « | 14 | » | сентября | 2023 г. | Профессор. Белов В.В. |

**Оглавление**

[Введение 5](#_heading=h.1fob9te)

[1 Формирование требований к ИС 6](#_heading=h.irqlxv3aum79)

[1.1 Модель предметной области AS IS 6](#_heading=h.3znysh7)

[1.1.1 Функциональный аспект 6](#_heading=h.eqojlte28lyj)

[1.1.2 Информационный аспект 6](#_heading=h.jt8ltxidtzko)

[1.1.3 Структурно архитектурный аспект 7](#_heading=h.i003jpordia1)

[1.1.4 Аспект поведения 9](#_heading=h.a0fradbqvg4q)

[1.2 Обоснование необходимости создания ПрИс 19](#_heading=h.89yzyctxyyff)

[1.3 Модель требований к проектируемой системе 20](#_heading=h.9w85tpl3p1j3)

[1.3.1 Модель TO BE 20](#_heading=h.w6ufpcd199e)

[1.3.2 Функциональные требования 20](#_heading=h.50ugfdiqg36n)

[1.3.3 Требования к информационности 41](#_heading=h.5mr3xkvznulz)

[1.3.4 Архитектурные требования 41](#_heading=h.mfrmlhyjmqux)

[1.3.5 Требования к поведению 42](#_heading=h.8ufbb5vywmhu)

[2 Проектирование системы 51](#_heading=h.1egqt2p)

[2.1 Архитектура данных и процессов 51](#_heading=h.3ygebqi)

[2.1 Модель поведения 72](#_heading=h.1x0gk37)

[2.2 Модель реализации 72](#_heading=h.4h042r0)

[2.2.1 Модель исходного кода 72](#_heading=h.2w5ecyt)

[2.2.3 Модель размещения артефактов 91](#_heading=h.3mzq4wv)

[3 Реализация ПрИС 92](#_heading=h.2250f4o)

[3.1 Реализация бизнес-логики 92](#_heading=h.haapch)

[3.2 Реализация пользовательского интерфейса 93](#_heading=h.319y80a)

[3.3 Реализация базы данных 112](#_heading=h.45jfvxd)

[4 Тестирование ПрИС 114](#_heading=h.zu0gcz)

[5 Состав проекта 118](#_heading=h.qtsvonj0ir3j)

[6 Компиляция и сборка системы 119](#_heading=h.1yyy98l)

[6.1 Правила создания системы 119](#_heading=h.lvxemloomeu8)

[6.2 Состав приложения 119](#_heading=h.2y3w247)

[7 Документация 121](#_heading=h.1d96cc0)

[7.1 Назначения системы и выполняемые ею функции 121](#_heading=h.3x8tuzt)

[7.2 Условия применения системы 121](#_heading=h.lvlh5v4sdub4)

[7.3 Правила инсталляции системы 121](#_heading=h.n0h53yqg5b6y)

[7.4 Сценарий взаимодействия с пользователем 122](#_heading=h.rub5msacgy0k)

[Заключение 124](#_heading=h.8zhtyrsppwbk)

[Список использованных источников 125](#_heading=h.n21wxv2krnbk)

[ПРИЛОЖЕНИЯ 126](#_heading=h.yn149o8yarbc)

[Приложение 1. Классы-сущности 126](#_heading=h.n8z236lenqur)

[Класс User 126](#_heading=h.dng287rmlh0g)

[Класс Event 130](#_heading=h.ge26mo6f8g5q)

[Класс Review 132](#_heading=h.d8qocdwf945a)

[Класс Ticket 135](#_heading=h.7fdbdlnbfs2y)

[Приложение 2. Управляющие классы 139](#_heading=h.1rle1bxf0z0a)

[Класс HomeController 139](#_heading=h.mq5lpba61vjk)

[Класс LogoutController 140](#_heading=h.8w1156w5bedx)

[Класс EventController 142](#_heading=h.nfw04d11ihfo)

[Класс ReviewController 149](#_heading=h.w7gfddwwpyh6)

[Класс TicketController 157](#_heading=h.llpfyq2i3w)

[Класс UserController 162](#_heading=h.4x1sqcl4midr)

[Класс RegistrationController 173](#_heading=h.7i6udybew0rh)

[Приложение 3. HTML-страницы 176](#_heading=h.s60abvpiu5a)

[Страница changePassword 176](#_heading=h.f2ljmvhv9gh8)

[Страница deleteEvent 179](#_heading=h.s0lxv7jjhk19)

[Страница deleteReview 180](#_heading=h.69qftozbx6rh)

[Страница deleteUser 182](#_heading=h.ctnetqq2uy7f)

[Страница editEvent 184](#_heading=h.kepwhjwb4h7j)

[Страница editReview 187](#_heading=h.92k97w4v7pw4)

[Страница editUser 189](#_heading=h.9gxktsbm2rk2)

[Страница editUserProfile 192](#_heading=h.r3f6te6kc0vp)

[Страница events 194](#_heading=h.tuw1xnhb2nnf)

[Страница home 198](#_heading=h.bqovvttquwhw)

[Страница login 200](#_heading=h.2i94aulctixv)

[Страница newEvent 202](#_heading=h.l0jmveafeq3)

[Страница newReview 205](#_heading=h.ibqklle7vlnx)

[Страница newUser 207](#_heading=h.k1n9pgxzhh9b)

[Страница registration 210](#_heading=h.rav1o6xn0ksp)

[Страница reviews 212](#_heading=h.uprcsdyefr87)

[Страница tickets 214](#_heading=h.7hz6vesd31q4)

[Страница userProfile 216](#_heading=h.ljem7rtjmr2w)

[Страница userReviews 218](#_heading=h.mw8inwbrjnui)

[Страница users 220](#_heading=h.wz3ki2aigmmc)

# Введение

В современном мире, где технологии проникают в различные аспекты нашей повседневной жизни, создание эффективных информационных систем становится неотъемлемой частью развития общества. В контексте культурных мероприятий и развлечений, важность легкости доступа к информации о мероприятиях и возможности онлайн-покупки билетов неоспорима. В связи с этим, представляемый курсовой проект направлен на разработку информационной системы, спроектированной для удобного поиска мероприятий и моментальной покупки билетов.

Данный проект будет осуществляться с использованием передовых технологий, что гарантирует его высокую производительность и эффективность. Программное обеспечение будет реализовано на языке программирования Java 17, используя фреймворк Spring для обеспечения надежности и масштабируемости приложения. Фронтенд системы будет создан с использованием HTML и CSS. Для разработки будет использоваться среда IntelliJ Idea CE, обеспечивающая интуитивно понятный пользовательский интерфейс.

Для хранения и управления данными будет использована база данных PostgresQL, обеспечивающая надежность и эффективность операций с информацией о мероприятиях. Для проектирования структуры системы и ее компонентов мы воспользуемся инструментом StarUML 5.1.0, обеспечивающим наглядное и структурированное представление архитектуры приложения.

Таким образом, разрабатываемая информационная система предоставит удобный и эффективный механизм для поиска мероприятий и приобретения билетов, совмещая в себе передовые технологии разработки и интуитивно понятный интерфейс для конечных пользователей.

# 1 Формирование требований к ИС

## 1.1 Модель предметной области AS IS

Модель "AS IS" представляет текущее состояние предметной области, предоставляя описание существующей организации бизнес-процессов. В следующих разделах мы подробно рассмотрим функциональные, информационные, структурно-архитектурные и поведенческие аспекты системы, которую мы планируем проектировать.

### 1.1.1 Функциональный аспект

Определим функции, которые система должна выполнять для удовлетворения потребностей пользователей и достижения целей бизнеса. Этот аспект фокусируется на функциональности системы, то есть на том, как она будет выполнять различные операции, обработку данных, взаимодействие с пользователями и другими системами.

Отразим функциональный аспект деятельности путём перечисления задач, выполняемых программным обеспечением:

1. добавление, изменение, удаление мероприятия в базе данных администратором;
2. добавление, изменение, удаление отзыва в базе данных пользователем;
3. добавление, изменение, удаление билета в базе данных пользователем и (или) администратором;
4. добавление, изменение, удаление пользователя в базе данных администратором и (или) пользователем.

### 1.1.2 Информационный аспект

Информационный аспект при проектировании информационной системы связан с управлением данными и информацией, необходимой для функционирования системы.

Информационный аспект отражен в таблице 1.1 ниже.

* + - 1. 1.1 – Информационный аспект проектируемой системы

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Класс | Носитель | Место хранения |
| Данные с информацией о мероприятии | Входная | Файл | База данных |
| Данные с информацией о билете | Входная | Файл | База данных |
| Данные с информацией об сеансе отзыве | Входная | Файл | База данных |
| Данные о билете | Выходная | Файл формата pdf | Личное хранилище |

### 1.1.3 Структурно архитектурный аспект

Проектируемая система предусматривает две роли для участников бизнес-процессов:

1. пользователь;
2. администратор.

Администратор – пользователь системы, имеющий права доступа к базе данных для внесения изменений в хранящуюся в ней информацию.

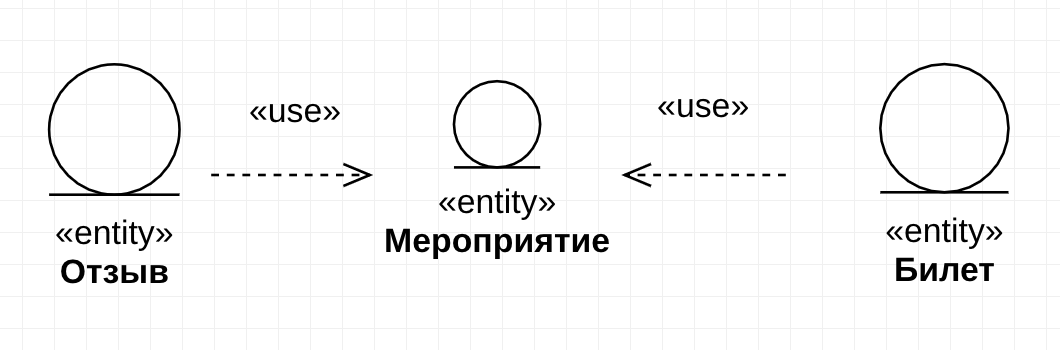
Пользователь – обычный пользователь системы, единственной возможностью которого является выбор и покупка билетов на мероприятия, написание, редактирование, и удаление отзывов, а также возможность редактирования информации о своей учетной записи.

На рисунке 1.1 представлено графическое представление исполнителей бизнес-процессов.



1. 1.1 – Исполнители бизнес-процессов

На рисунке 1.2 представлены исполнители информационности. В базе данных хранится информация о мероприятиях и отзывах на них. На основании мероприятия формируется билет, содержащий информацию о мероприятии и пользователе, который купил билет на него.

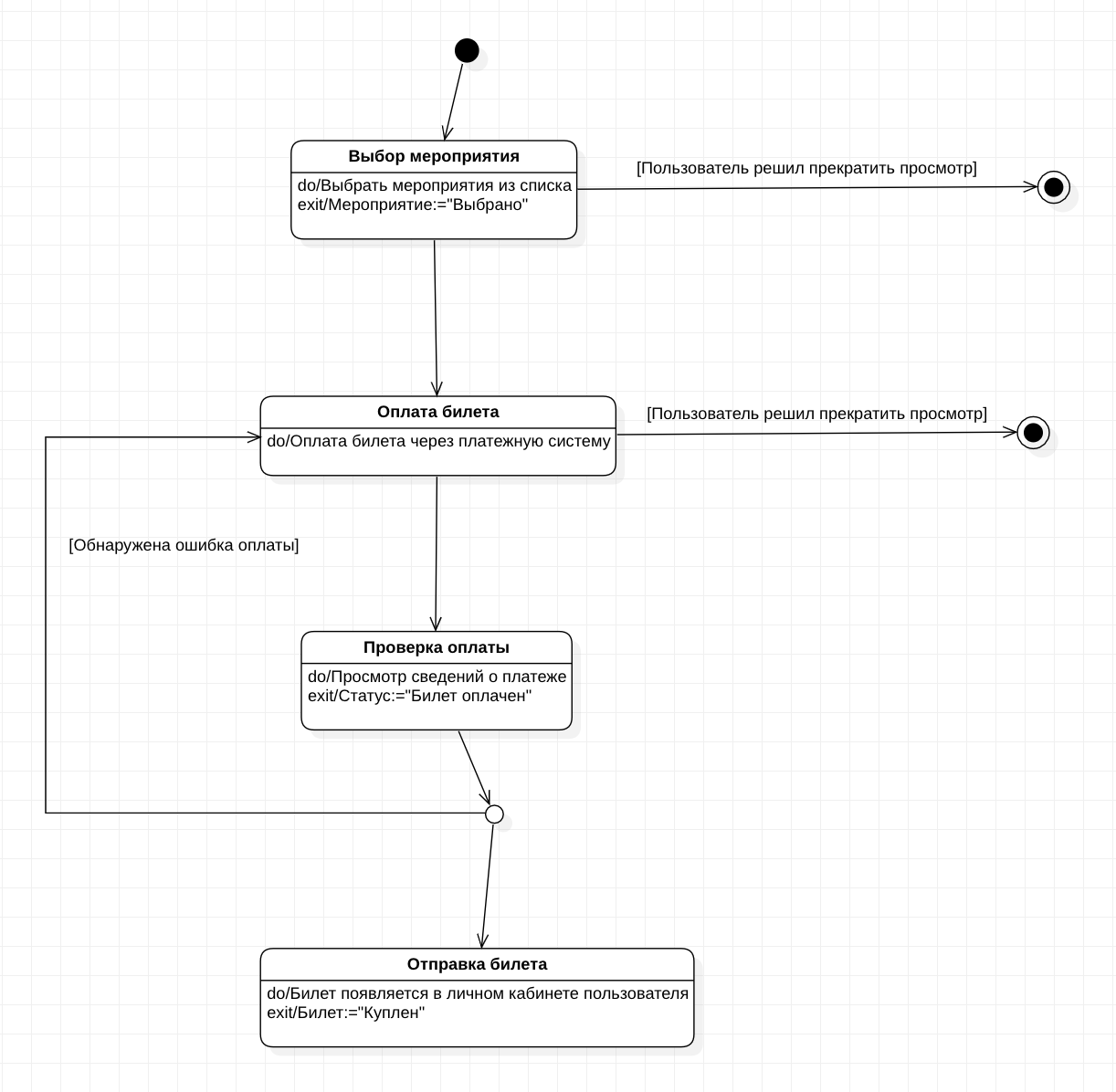


1. 1.2 – Исполнители информационности

### 1.1.4 Аспект поведения

Так как взаимодействие с системой подразумевает под собой возможность бесконечного ожидания, например, ввода названия мероприятия или выбора мероприятия из выпадающего таблицы, для графического представления алгоритмов реализации будут использоваться диаграммы автоматов. Алгоритм покупки билета представлен на рисунке 1.3.

Если над стрелкой есть подпись в квадратных скобках «Пользователь решил прекратить просмотр», переход происходит только при выполнении условия, указанного внутри этих скобок, в данном случае при нажатии пользователя отмены или закрытия формы.



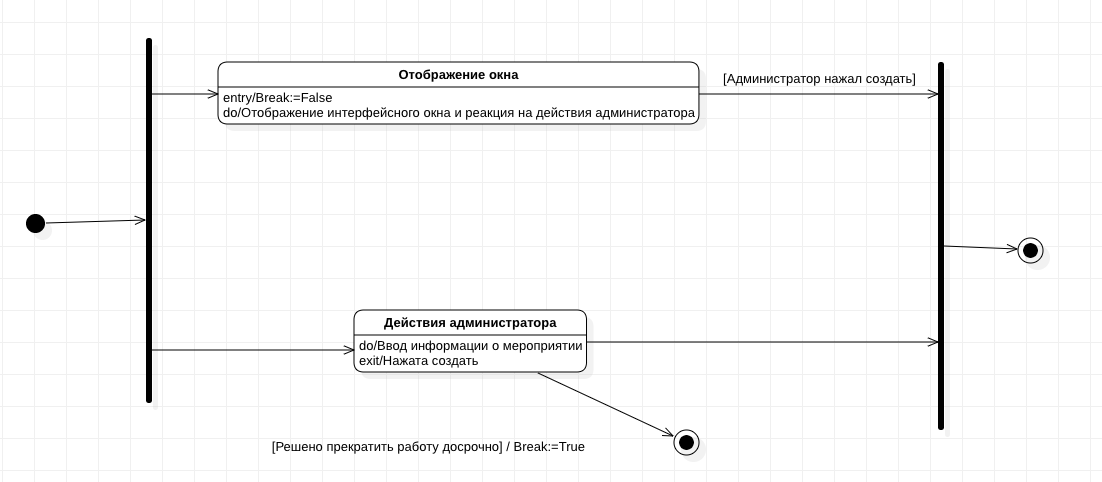
**Ошибка! Здесь и далее. Триггерные события изображаются как условия!**

1. 1.3 – Алгоритм покупки билета

На рисунке 1.4 представлен алгоритм добавления нового мероприятия в базу данных. После ввода всей информации, администратор может инициировать добавление новой записи в базу данных:

1. отображение окна - отображение окна с полями ввода для следующей информации: название мероприятия, дата, описание. И реакция системы на действия администратора.
2. действия администратора - ввод следующей информации о мероприятиее: название мероприятия, год выпуска, возрастное ограничение, афиша, постер, и подтверждение внесения изменений в БД.

Если над стрелкой есть подпись в квадратных скобках «Пользователь решил прекратить просмотр», переход происходит только при выполнении условия, указанного внутри этих скобок, в данном случае при нажатии пользователя отмены или закрытия формы.



**Ошибка! Здесь и до рис. 1.6. Событие «Нажата создать» изображено как действие!**

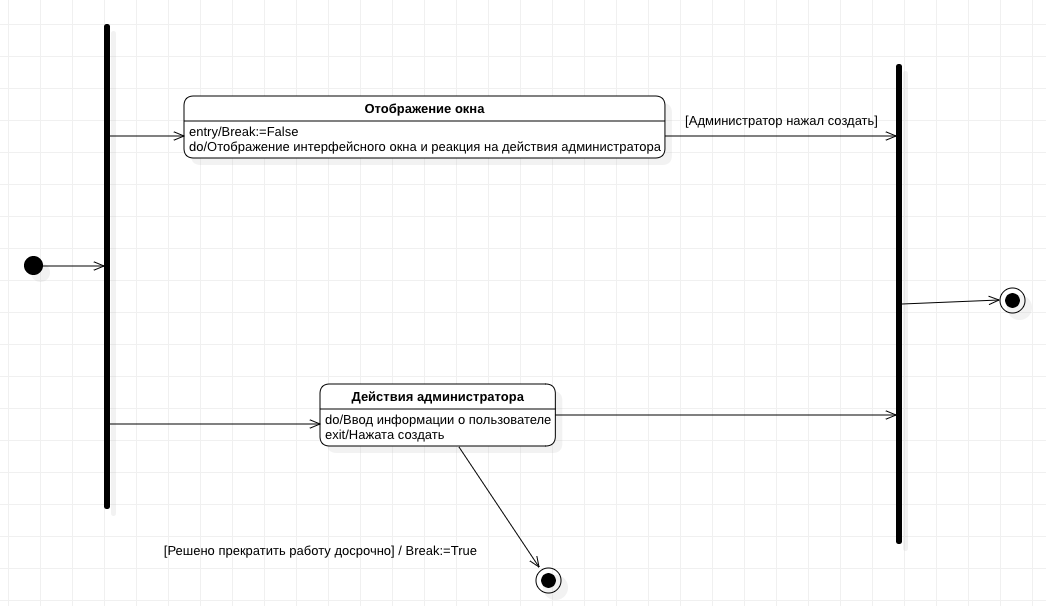
1. 1.4 – Алгоритм добавления нового мероприятия

Аналогично реализован алгоритм добавления нового пользователя.

1. Отображение окна - отображение интерфейсного окна с полями ввода для пользователя и реакция системы на действия администратора.
2. Действия администратора - действия, совершаемые администратором, для создания нового пользователя (ввод данных пользователя), подтверждение внесения изменений в БД.

Администратор может создать как обычного пользователя, так и нового администратора.

Если над стрелкой есть подпись в квадратных скобках «Пользователь решил прекратить просмотр», переход происходит только при выполнении условия, указанного внутри этих скобок, в данном случае при нажатии пользователя отмены или закрытия формы. Алгоритм представлен на рисунке 1.5.



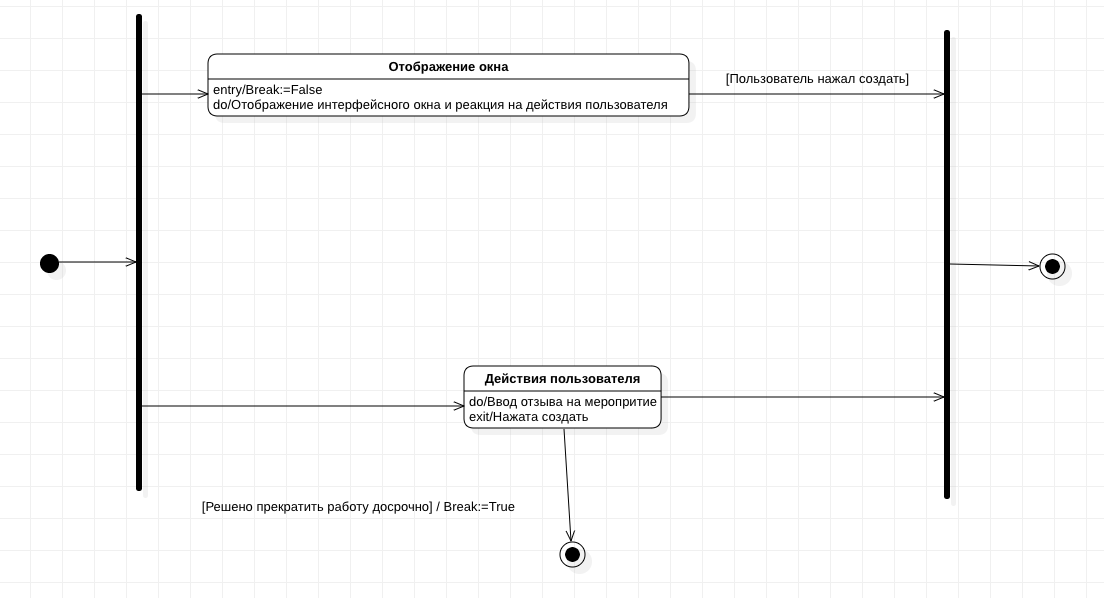
1. 1.5 – Алгоритм добавления нового пользователя

Аналогично реализован алгоритм добавления нового отзыва.

1. Отображение окна - отображение интерфейсного окна с полями ввода для пользователя и реакция системы на действия пользователя.
2. Действия пользователя - действия, совершаемые пользователя, для создания нового отзыва (ввод данных об отзыве), подтверждение внесения изменений в БД.

Если над стрелкой есть подпись в квадратных скобках «Пользователь решил прекратить просмотр», переход происходит только при выполнении условия, указанного внутри этих скобок, в данном случае при нажатии пользователя отмены или закрытия формы.

Алгоритм представлен на рисунке 1.6.



1. 1.6 – Алгоритм добавления нового отзыва

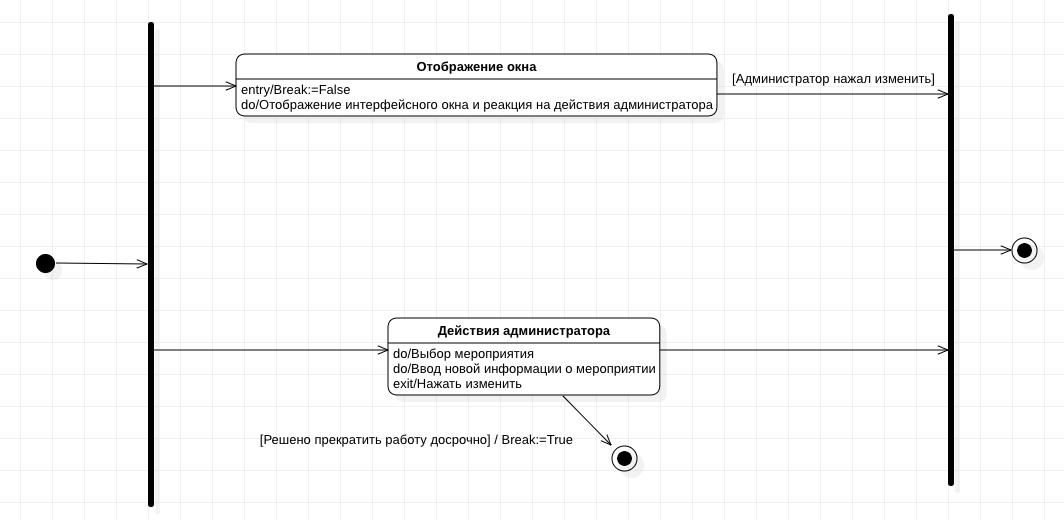
Алгоритмы редактирования схожи с алгоритмами добавления. Только перед внесением данных необходимо сначала выбрать из выпадающего списка нужную сущность. Алгоритмы редактирования мероприятия, пользователя и отзыва представлены на рисунках ниже.

Рассмотрим алгоритм изменения мероприятия:

1. Отображение окна - отображение окна с полями ввода для следующей информации: название мероприятия, дата проведения, описание. И реакция системы на действия администратора.
2. Действия администратора – возможность ввода следующей информации о мероприятии: название мероприятия, дата проведения, описание, и подтверждение внесения изменений в БД, нажатием на кнопку «Сохранить».

Если над стрелкой есть подпись в квадратных скобках «Пользователь решил прекратить просмотр», переход происходит только при выполнении условия, указанного внутри этих скобок, в данном случае при нажатии пользователя отмены или закрытия формы.

Алгоритм представлен на рисунке 1.7.



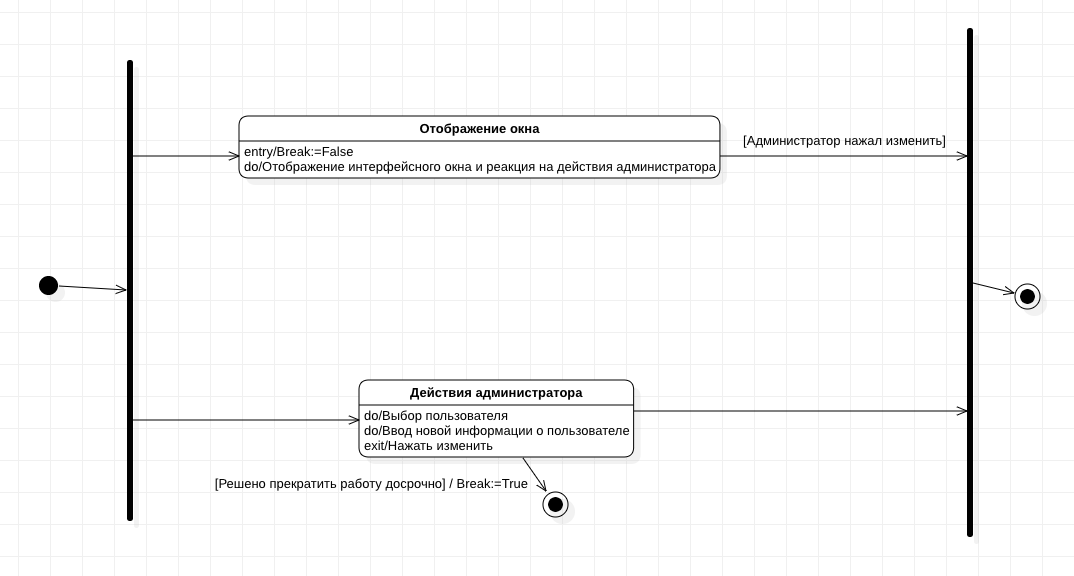
1. 1.7 – Алгоритм изменения мероприятия

Рассмотрим алгоритм изменения пользователя:

1. Отображение окна - отображение окна с полями ввода для следующей информации: имя пользователя, роль, почта. И реакция системы на действия администратора.
2. Действия администратора – возможность ввода следующей информации о пользователе: имя пользователя, роль, почта, и подтверждение внесения изменений в БД, нажатием на кнопку «Сохранить».

Если над стрелкой есть подпись в квадратных скобках «Пользователь решил прекратить просмотр», переход происходит только при выполнении условия, указанного внутри этих скобок, в данном случае при нажатии пользователя отмены или закрытия формы.

Алгоритм представлен на рисунке 1.8.



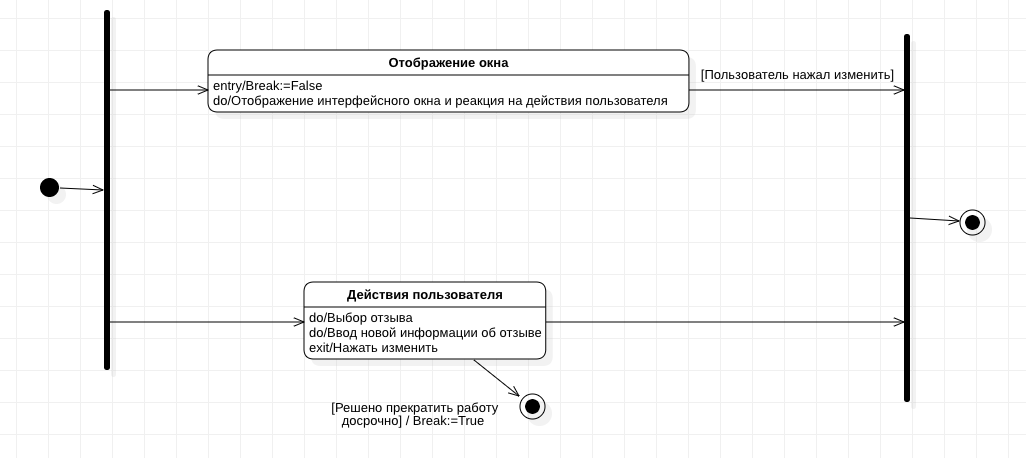
1. 1.8 – Алгоритм изменения пользователя

Рассмотрим алгоритм изменения пользователя:

1. Отображение окна - отображение окна с полями ввода для следующей информации: текст отзыва. И реакция системы на действия пользователя.
2. Действия пользователя – возможность ввода следующей информации о пользователе: текст отзыва, и подтверждение внесения изменений в БД, нажатием на кнопку «Сохранить».

Если над стрелкой есть подпись в квадратных скобках «Пользователь решил прекратить просмотр», переход происходит только при выполнении условия, указанного внутри этих скобок, в данном случае при нажатии пользователя отмены или закрытия формы.

Алгоритм представлен на рисунке 1.9.



1. 1.9 – Алгоритм изменения отзыва

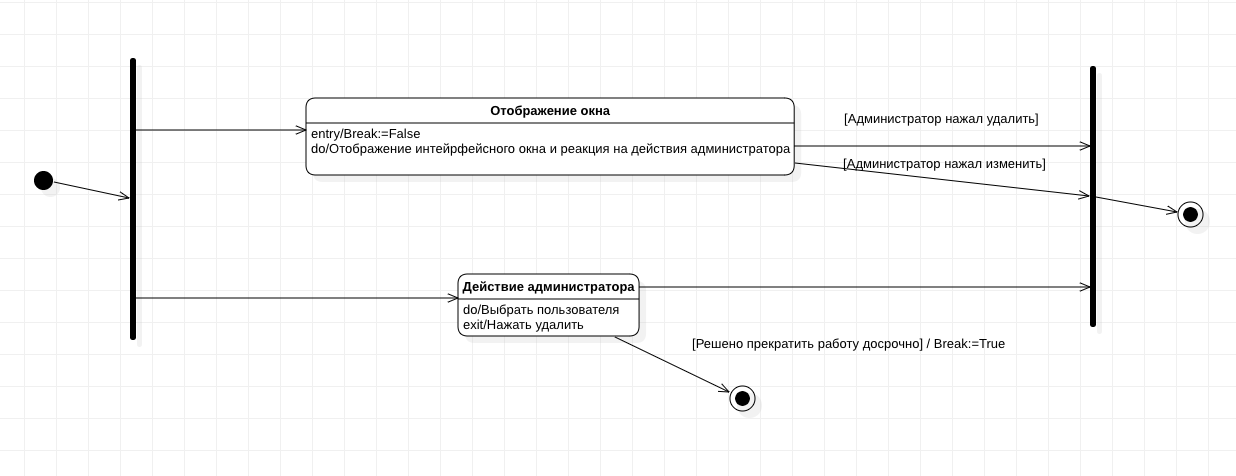
Для удаления сущности из базы данных администратор должен выбрать сущность и подтвердить намерение удалить нажатием на кнопку. На рисунках ниже приведены алгоритмы удаления сущностей из базы данных.

Рассмотрим алгоритм удаления пользователя:

1. Отображение окна - отображение окна списком из пользователей. И реакция системы на действия администратора.
2. Действия администратора – выбор пользователя из выпадающего списка и подтверждение внесения изменений в БД, нажатием на кнопку «Удалить».

Если над стрелкой есть подпись в квадратных скобках «Пользователь решил прекратить просмотр», переход происходит только при выполнении условия, указанного внутри этих скобок, в данном случае при нажатии пользователя отмены или закрытия формы.

Алгоритм представлен на рисунке 1.10.



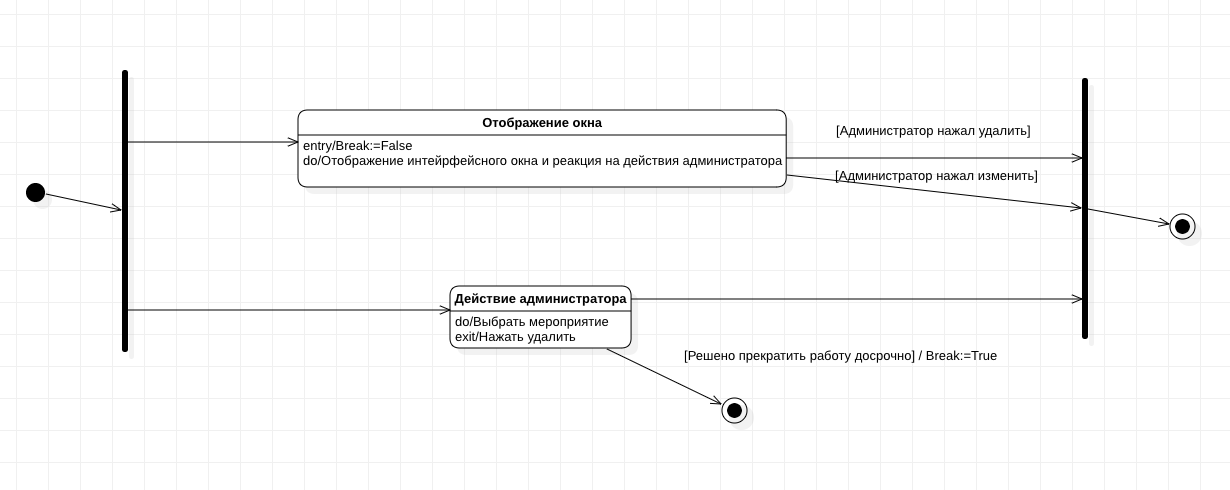
1. 1.10 – Алгоритм удаления пользователя

Рассмотрим алгоритм удаления мероприятия:

1. Отображение окна - отображение окна списком из мероприятий. И реакция системы на действия администратора.
2. Действия администратора – выбор пользователя из выпадающего списка и подтверждение внесения изменений в БД, нажатием на кнопку «Удалить».

Если над стрелкой есть подпись в квадратных скобках «Пользователь решил прекратить просмотр», переход происходит только при выполнении условия, указанного внутри этих скобок, в данном случае при нажатии пользователя отмены или закрытия формы.

Алгоритм представлен на рисунке 1.11.



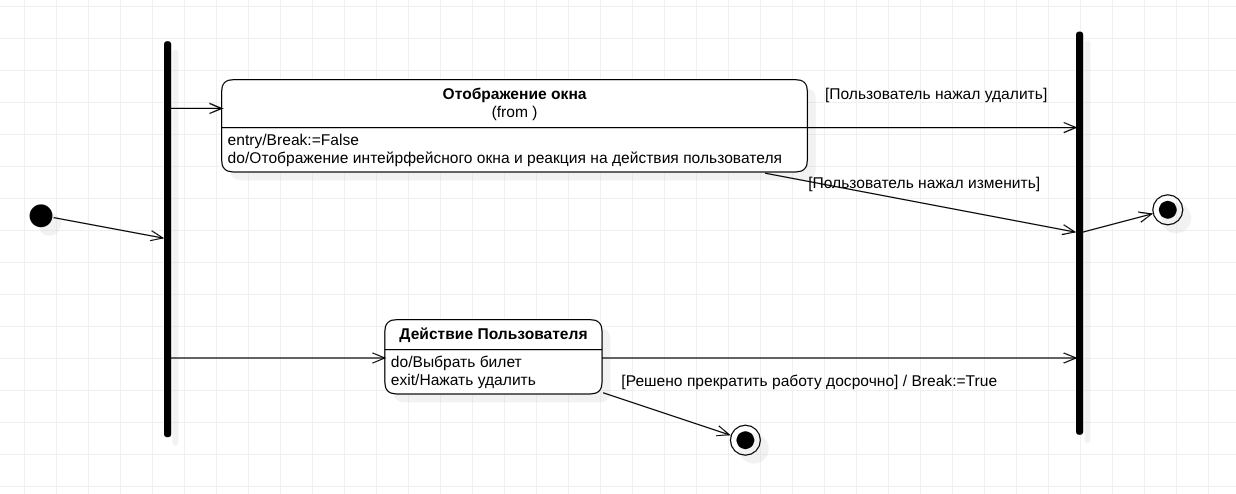
1. 1.11 – Алгоритм удаления мероприятия

Рассмотрим алгоритм удаления билета:

1. Отображение окна - отображение окна списком из билетов. И реакция системы на действия администратора.
2. Действия администратора – выбор пользователя из выпадающего списка и подтверждение внесения изменений в БД, нажатием на кнопку «Удалить».

Если над стрелкой есть подпись в квадратных скобках «Пользователь решил прекратить просмотр», переход происходит только при выполнении условия, указанного внутри этих скобок, в данном случае при нажатии пользователя отмены или закрытия формы.

Алгоритм представлен на рисунке 1.12.



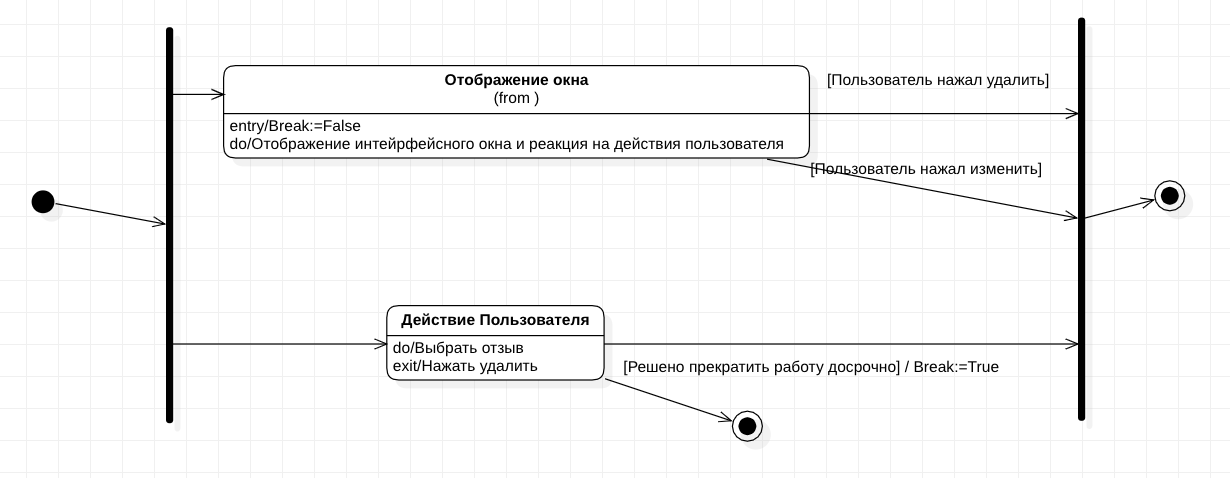
1. 1.12 – Алгоритм удаления билета

Рассмотрим алгоритм удаления мероприятия:

1. Отображение окна - отображение окна списком из мероприятий. И реакция системы на действия администратора.
2. Действия администратора – выбор пользователя из выпадающего списка и подтверждение внесения изменений в БД, нажатием на кнопку «Удалить».

Если над стрелкой есть подпись в квадратных скобках «Пользователь решил прекратить просмотр», переход происходит только при выполнении условия, указанного внутри этих скобок, в данном случае при нажатии пользователя отмены или закрытия формы.

Алгоритм представлен на рисунке 1.13.



1. 1.13 – Алгоритм удаления отзыва

## 1.2 Обоснование необходимости создания ПрИс

Разрабатываемая система создается для широкого круга пользователей. Практически каждый человек хоть раз посещал мероприятия, проводимые в городе. С помощью разрабатываемой системы найти мероприятие и купить билет на него можно будет с помощью пары кликов

На текущий момент существующие аналоги работают через сеть Интернет. Аналоги проектируемого приложения задействованы в основном на столицу. Из-за этого популяризировать мероприятия в других городах не просто. Благодаря приложениям в интернете не требует наличие программы, которую пользователь устанавливает на свое устройство. Это позволяет масштабировать приложение на разные устройства.

Таким образом, проектируемая система позволит покрыть широкий круг пользователей, которые будут использовать данное приложение.

## 

## 1.3 Модель требований к проектируемой системе

### 1.3.1 Модель TO BE

В таблице 1.2 приведена сравнительная характеристика проектируемого сервиса и аналогов.

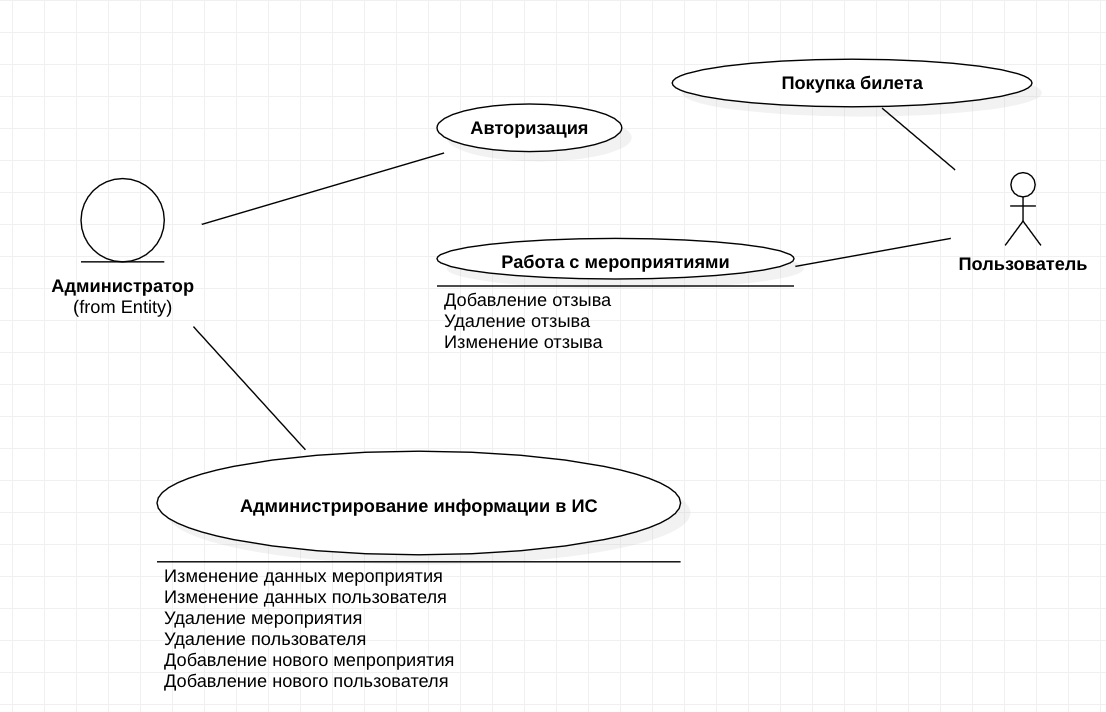
* + - 1. 1.2 – Сравнение бизнес-процессов проектируемой системы и аналогов

|  |  |
| --- | --- |
| **Было** | **Стало** |
| Сервисы покупки билетов на мероприятия направлены на столицу | Сервис покупки билетов на мероприятия доступен во всех городах |
| Сервисы покупки билетов содержать большое количество рекламы | Сервис покупки билетов не содержит мешающей рекламы |
| Сервисы покупки билетов содержат огромное количество избыточной информации | Сервис позволяет купить билет на интересующее мероприятие без избыточной информации |

### 1.3.2 Функциональные требования

Выразим функциональные требования к системе через диаграммы вариантов использования.

Базисные варианты использования представлены на рисунке 1.14.



**Ошибка! Работник не Worker, а сущность!**

1. 1.14 – Базисные варианты использования

Опишем представленные варианты использования:

1. авторизация – авторизация в системе для получения доступа к функционалу администратора;
2. покупка билетов – просмотр информации о мероприятиях, локациих, в которых они идут, доступных сеансов и возможность купить билет на него;
3. администрирование информации в ИС – создание, удаление, изменение данных, созданных в системе;
4. работа с мероприятиями – добавление, изменение и удаление данных, созданных пользователем.

Отношения между актерами и прецедентами выбрано ненаправленным потому актор является активным участником взаимодействия – не только запускает выполнение прецедента, задает исходные данные и получает результат, но и участвует в диалоге с программой, реализующей прецедент.

Декомпозируем вариант использования Авторизация.

Авторизация предусматривает заполнение следующих полей:

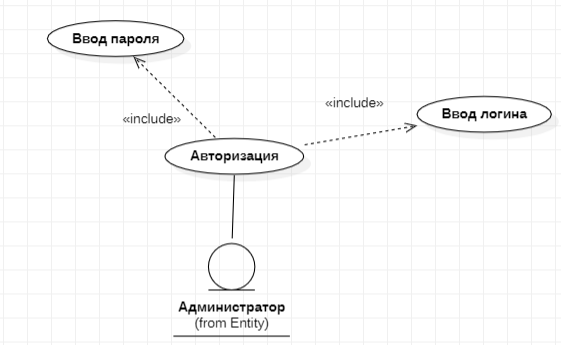
1. ввод логина – ввод логина для авторизации в системе;
2. ввод пароля – ввода пароля от аккаунта для авторизации в системе.

В таблице 1.3 представлено описание алгоритма.

* + - 1. 1.3 – Описание алгоритма варианта использования «Авторизация»

|  |  |
| --- | --- |
| Действующие лица | Администратор |
| Краткое описание | Администратор авторизируется в своем личном кабинете для получения прав на изменение БД |
| Предусловия | Система запущена |
| Постусловия | Администратор получил доступ к функционалу системы |
| Нормальное течение | 1. Прецедент начинается с загрузки администратором страницы авторизации  2. Система открывает форму.  3. Администратор вводит: логин, пароль  4. Система загружает профиль  *5. Е1. [Пароль и логин неверны]*  6. Вариант использования завершается. |
| Альтернативные течения |  |
| Исключительные течения | **Е1. Пароль и логин неверны**  Поток возвращается к этапу 2 основного потока |
| Приоритет | Критично |
| Частота использования | Часто |

На рисунке 1.15 представлена декомпозиция авторизации.



1. 1.15 – Декомпозиция варианта использование «Авторизация»

Тип отношение зависимости между прецедентами Авторизация и Ввод пароля (смотри рисунок 1.15), а также Ввод логина выбран «include», потому что при реализации прецедента Регистрация функциональность прецедента Ввод пароля используется безусловно.

Декомпозируем вариант использования “Администрирование информации в ИС”.

Варианты использования представлены в таблицах 1.4-1.11.

Разделим вариант использования на две составляющие: работа с пользователями и работа с мероприятиями.

Рассмотрим создание пользователя (создание аккаунта пользователя в системе):

1. ввод логина – ввод логина для авторизации в системе;
2. ввод пароля – ввод пароля для авторизации в системе;
3. ввод почты – ввод почты, на которую будет приходит информация об аккаунте;
4. указание роли – указываем, пользователь это или администратор.

Изменение аккаунта пользователя (изменение данных аккаунта пользователя в системе):

1. ввод логина – ввод нового логина для существующего аккаунта;
2. ввод пароля – ввод нового пароля для существующего аккаунта ;
3. ввод почты – ввод новой почты для существующего аккаунта;
4. указание роли – указываем, пользователь это или администратор.

Удаление аккаунта – удаление существующего в системе аккаунта.

Тип отношение зависимости между прецедентом «Работа с аккаунтами пользователей и прецедентами «Изменение аккаунта пользователя, «Удаление аккаунта пользователей, «Создание пользователя был выбран «extend», потому что при реализации прецедентов «Работа с аккаунтами пользователей функциональности прецедентов «Изменение аккаунта пользователя, «Удаление аккаунта пользователя, «Создание пользователя используются только при выборе администратором нужной опции.

* + - 1. 1.4 – Описание алгоритма варианта использования «Работа с аккаунтом пользователя

|  |  |
| --- | --- |
| Действующие лица | Администратор |
| Краткое описание | Администратор вносит изменения в БД |
| Предусловия | Авторизация в кабинете администратора |
| Постусловия | Администратор успешно изменил данные БД |
| Нормальное течение | 1. Прецедент начинается с выбора администратором изменения информации в БД 2. Система открывает форму. 3. *А1. [Администратор решает вернуться на страницу выбора действий]* 4. S1 [Администратор выбирает пункт «Создать пользователя] 5. S2 [Администратор выбирает пункт «Редактировать аккаунт пользователя] 6. S3 [Администратор выбирает пункт «Удалить аккаунт пользователя] 7. Вариант использования завершается. |
| Альтернативные течения | **A1. Пользователь решает вернуться на страницу выбора действий.**   1. Поток возвращается к этапу 1 основного потока |
| Подпотоки | Подпотоки S1-3 описываются отдельными таблицами |
| Исключительные течения | Не определены |
| Подпотоки | Не определены |
| Приоритет | Критично |
| Частота использования | Иногда |

* + - 1. 1.5 – Описание алгоритма варианта использования «Создание пользователя, подпоток S1

|  |  |
| --- | --- |
| Действующие лица | Администратор |
| Краткое описание | Администратор регистрирует аккаунт нового работника-администратора |
| Предусловия | Авторизация в кабинете администратора |
| Постусловия | Администратор был добавлен в базу данных |
| Нормальное течение | 1. Прецедент начинается с выбора администратором режима создания нового администратора  2. Система открывает форму.  *3. А1. [Администратор решает вернуться на страницу выбора действий]*  4. Администратор вводит: логин, пароль, почту, роль  5. Администратор сохраняет изменения на сервер  *6. А2. [ Администратор нажимает кнопку отмена]*  *7. Е1. Логин уже занят*  8. Вариант использования завершается. |
| Альтернативные течения | **A1. Пользователь решает вернуться на страницу выбора действий.**  Поток возвращается к этапу 1 основного потока  **А2. Администратор нажимает кнопку отмена**  1. Поток возвращается к этапу 1 основного потока. |
| Исключительные течения | **Е1. Логин уже занят**  Поток возвращается к этапу 4 основного потока |
| Подпотоки | Не определены |
| Приоритет | Критично |
| Частота использования | Иногда |

* + - 1. 1.6 – Описание алгоритма варианта использования «Редактирование аккаунта пользователя, подпоток S2

|  |  |
| --- | --- |
| Действующие лица | Администратор |
| Краткое описание | Администратор редактирует аккаунт существующего работника-администратора |
| Предусловия | Авторизация в кабинете администратора |
| Постусловия | Данные аккаунта администратора в базе данных были изменены |
| Нормальное течение | 1. Прецедент начинается с выбора администратором режима редактирования аккаунта администратора 2. Система открывает форму. 3. А1. [Администратор решает вернуться на страницу выбора действий] 4. Администратор выбирает аккаунт для редактирования 5. Администратор может изменить: логин, пароль, адрес электронной почты 6. Администратор нажимает на кнопку сохранить 7. А2. [ Администратор нажимает кнопку отмена] 8. Вариант использования завершается. |
| Альтернативные течения | **A1. Администратор решает вернуться на страницу выбора действий.**  1. Поток возвращается к этапу 1 основного потока.  **А2. Администратор нажимает кнопку отмена**  1. Поток возвращается к этапу 1 основного потока. |
| Исключительные течения |  |
| Подпотоки | Не определены |
| Приоритет | Критично |
| Частота использования | Иногда |

* + - 1. 1.7 – Описание алгоритма варианта использования «Удаление аккаунта пользователя, подпоток S3

|  |  |
| --- | --- |
| Действующие лица | Администратор |
| Краткое описание | Администратор удаляет аккаунт существующего пользователя |
| Предусловия | Авторизация в кабинете администратора |
| Постусловия | Данные аккаунта администратора в базе данных были удалены |
| Нормальное течение | 1. Прецедент начинается с выбора администратором режима удаления аккаунта администратора 2. Система открывает форму. 3. *А1. [Администратор решает вернуться на страницу выбора действий]* 4. Администратор выбирает аккаунт для удаления 5. Администратор нажимает на кнопку удалить 6. *А2. [ Администратор нажимает кнопку отмена]* 7. Вариант использования завершается. |
| Альтернативные течения | **A1. Администратор решает вернуться на страницу выбора действий.**  1. Поток возвращается к этапу 7 основного потока.  **А2. Администратор нажимает кнопку отмена**  1. Поток возвращается к этапу 7 основного потока. |
| Исключительные течения |  |
| Подпотоки | Не определены |
| Приоритет | Критично |
| Частота использования | Иногда |

Рассмотрим вариант использования «Создание мероприятия»:

1. ввод названия – ввод названия мероприятия;
2. ввод описания – ввод описания меропрития;
3. ввод даты – ввод даты проведения мероприятия;

Изменение мероприятия (изменение данных о мероприятии в системе):

1. ввод названия – ввод названия мероприятия;
2. ввод описания – ввод описания меропрития;
3. ввод даты – ввод даты проведения мероприятия;

Удаление мероприятия – удаление существующего в системе мероприятия.

Тип отношение зависимости между прецедентом «Администрирование информации в ИС и прецедентами «Изменение мероприятия, «Удаление мероприятия, «Создание мероприятия был выбран «extend», потому что при реализации прецедентов «Администрирование информации в ИС функциональности прецедентов «Изменение мероприятия, «Удаление мероприятия, «Создание мероприятия используются только при выборе администратором нужной опции.

* + - 1. 1.8 – Описание алгоритма варианта использования «Администрирование информации в ИС

|  |  |
| --- | --- |
| Действующие лица | Администратор |
| Краткое описание | Администратор вносит изменения в БД |
| Предусловия | Авторизация в кабинете администратора |
| Постусловия | Администратор успешно изменил данные БД |
| Нормальное течение | 1. Прецедент начинается с выбора администратором изменения информации в БД 2. Система открывает форму. 3. *А1. [Администратор решает вернуться на страницу выбора действий]* 4. S1 [Администратор выбирает пункт «Создать мероприятие] 5. S2 [Администратор выбирает пункт «Редактировать мероприятие] 6. S3 [Администратор выбирает пункт «Удалить мероприятие] 7. Вариант использования завершается. |
| Альтернативные течения | **A1. Пользователь решает вернуться на страницу выбора действий.**   1. Поток возвращается к этапу 1 основного потока |
| Подпотоки | Подпотоки S1-3 описываются отдельными таблицами |
| Исключительные течения | Не определены |
| Подпотоки | Не определены |
| Приоритет | Критично |
| Частота использования | Иногда |

* + - 1. 1.9 – Описание алгоритма варианта использования «Создание мероприятия, подпоток S1

|  |  |
| --- | --- |
| Действующие лица | Администратор |
| Краткое описание | Администратор регистрирует новое мероприятие |
| Предусловия | Авторизация в кабинете администратора |
| Постусловия | Мероприятие было добавлено в базу данных |
| Нормальное течение | 1. Прецедент начинается с выбора администратором режима создания нового мероприятия  2. Система открывает форму.  *3. А1. [Администратор решает вернуться на страницу выбора действий]*  4. Администратор вводит: название, описание, дату  5. Администратор сохраняет изменения на сервер  *6. А2. [ Администратор нажимает кнопку отмена]*  7. Вариант использования завершается. |
| Альтернативные течения | **A1. Пользователь решает вернуться на страницу выбора действий.**  Поток возвращается к этапу 1 основного потока  **А2. Администратор нажимает кнопку отмена**  1. Поток возвращается к этапу 1 основного потока. |
| Исключительные течения |  |
| Подпотоки | Не определены |
| Приоритет | Критично |
| Частота использования | Иногда |

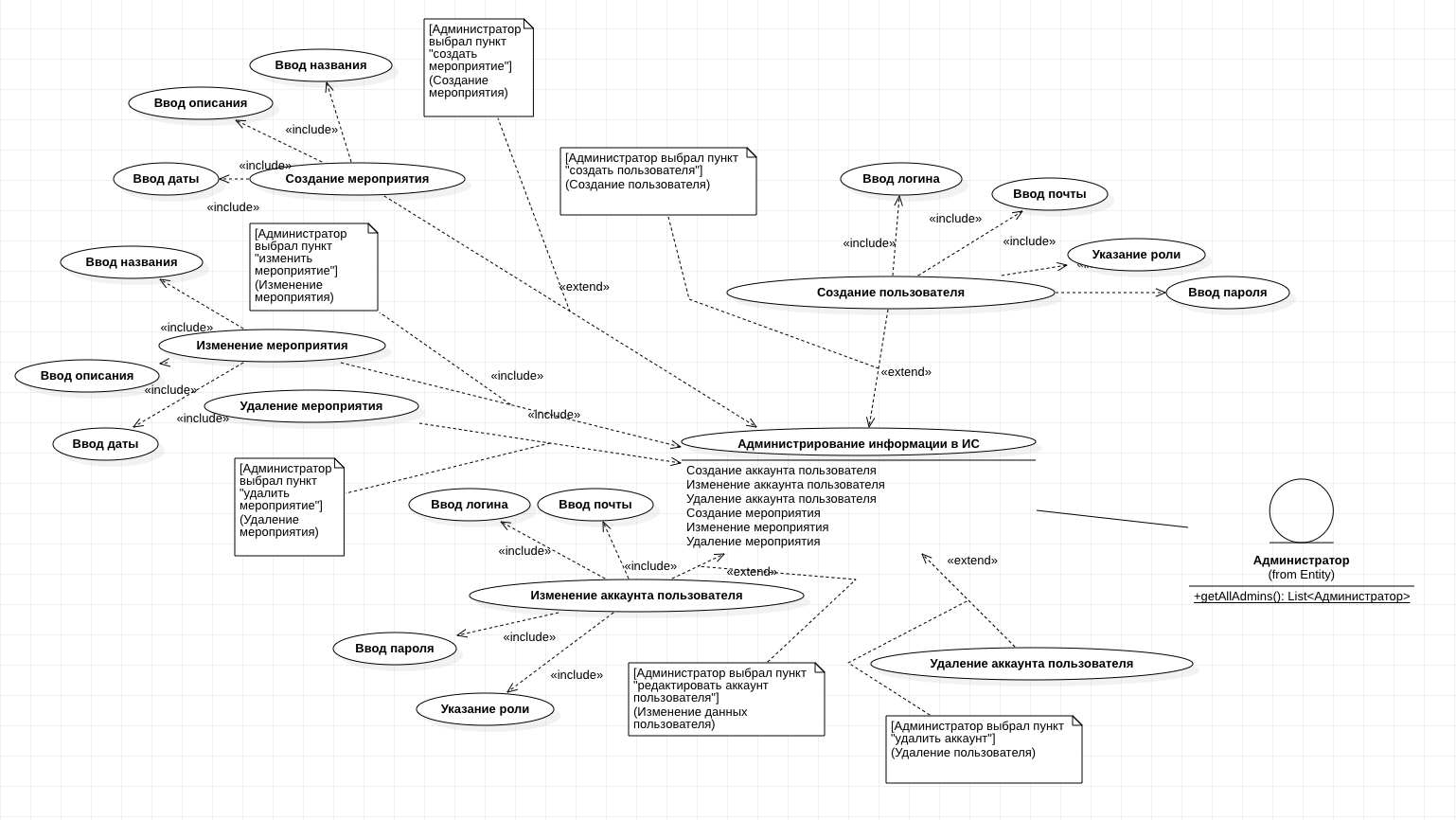
* + - 1. 1.10 – Описание алгоритма варианта использования «Редактирование мероприятия, подпоток S2

|  |  |
| --- | --- |
| Действующие лица | Администратор |
| Краткое описание | Администратор редактирует существующее мероприятие |
| Предусловия | Авторизация в кабинете администратора |
| Постусловия | Данные мероприятия в базе данных были изменены |
| Нормальное течение | 1. Прецедент начинается с выбора администратором режима редактирования мероприятия 2. Система открывает форму. 3. *А1. [Администратор решает вернуться на страницу выбора действий]* 4. Администратор выбирает аккаунт для редактирования 5. Администратор может изменить: название, описание, дату 6. Администратор нажимает на кнопку сохранить 7. *А2. [ Администратор нажимает кнопку отмена]* 8. Вариант использования завершается. |
| Альтернативные течения | **A1. Администратор решает вернуться на страницу выбора действий.**  1. Поток возвращается к этапу 1 основного потока.  **А2. Администратор нажимает кнопку отмена**  1. Поток возвращается к этапу 1 основного потока. |
| Исключительные течения | **Е1. Логин уже занят**  Поток возвращается к этапу 5 основного потока |
| Подпотоки | Не определены |
| Приоритет | Критично |
| Частота использования | Иногда |

* + - 1. 1.11 – Описание алгоритма варианта использования «Удаление мероприятия, подпоток S3

|  |  |
| --- | --- |
| Действующие лица | Администратор |
| Краткое описание | Администратор удаляет аккаунт существующего пользователя |
| Предусловия | Авторизация в кабинете администратора |
| Постусловия | Данные аккаунта администратора в базе данных были удалены |
| Нормальное течение | 1. Прецедент начинается с выбора администратором режима удаления мероприятия 2. Система открывает форму. 3. А1. [Администратор решает вернуться на страницу выбора действий] 4. Администратор выбирает мероприятие для удаления 5. Администратор нажимает на кнопку удалить 6. А2. [ Администратор нажимает кнопку отмена] 7. Вариант использования завершается. |
| Альтернативные течения | **A1. Администратор решает вернуться на страницу выбора действий.**  1. Поток возвращается к этапу 7 основного потока.  **А2. Администратор нажимает кнопку отмена**  1. Поток возвращается к этапу 7 основного потока. |
| Исключительные течения |  |
| Подпотоки | Не определены |
| Приоритет | Критично |
| Частота использования | Иногда |

На рисунке 1.16 представлена декомпозиция варианта использования «Администрирование информации в ИС».



**Ошибка! Две стрелки extend изображены как include**

1. 1.16 – Декомпозиция варианта использование «Администрирование информации в ИС»

Рассмотрим покупку билета пользователем:

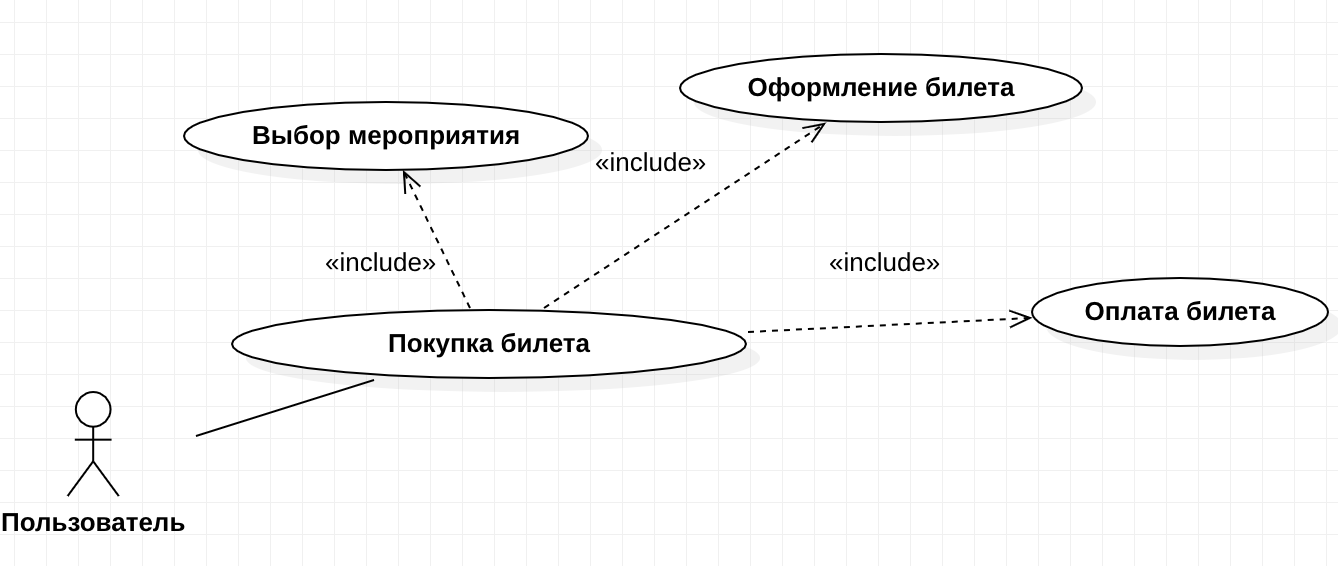
1. выбор мероприятия – выбор мероприятия из предложенного списка;
2. оформление билета – ввод дополнительных данных для покупки билетов (почта для получения электронного билета, ввод данных карты);
3. оплата билета – проведение оплаты билета.

В таблице 1.12 представлено описание алгоритма варианта использования «Покупка билета» в виде текстовых потоков.

* + - 1. 1.12 – Описание алгоритма варианта использования «Покупка билета»

|  |  |
| --- | --- |
| Действующие лица | Пользователь |
| Краткое описание | Пользователь ищем мероприятие и покупает билет |
| Предусловия | Запущена система |
| Постусловия | Пользователь завершил общение с системой |
| Нормальное течение | 1. Пользователь выбирает поиск мероприятий 2. Система открывает каталог. 3. Пользователь выбирает мероприятие. 4. А1. [Пользователь выбирает другое мероприятие] 5. Пользователь нажимает кнопку «Купить билет». 6. Система переходит к первому шагу покупки билета: запрашивает у пользователя личные данные и почту. 7. Пользователь вводит личные данные и почту. 8. Система переходит ко второму шагу оформления билета: показывает информацию о билете и просит подтвердить покупку. 9. Пользователь подтверждает покупку. 10. А2. [Пользователь возвращается к списку мероприятий]. 11. Система переходит к третьему шагу покупки билета: запрашивает тип кредитной карты, ее номер, секретный код, имя владельца и дату завершения срока действия. 12. Пользователь вводит тип кредитной карты, ее номер, секретный код, имя владельца и дату завершения срока действия. 13. Система переходит к четвертому шагу оформления заказа: подтверждает оплату. 14. А3. [Счет пользователя не найден]. 15. А4. [Недостаточно денег на счете]. 16. Е1. [Платежная система недоступна]. 17. Система присваивает заказу номер и отправляет его вместе с подтверждением заказа на электронный адрес покупателя. 18. Вариант использования завершается. |
| Альтернативные течения | **А1. [Пользователь выбирает другое мероприятие.**  Поток возвращается к этапу 2 основного потока  **А2. Пользователь решает к списку мероприятий.**  1. Поток возвращается к этапу 2 основного потока.  **А3. Счет пользователя не найден**  1. Система выводит сообщение о том, что счет пользователя не обнаружен.  2. Поток возвращается к этапу 9 основного потока.  **А4. Недостаточно денег на счете**  1. Система выводит сообщение о том, что на счете пользователя недостаточно денег для совершения операции.  2. Поток возвращается к этапу 16 основного потока. |
| Исключительные течения | **Е1. Платежная система недоступна**  1. Система выводит сообщение о недоступности платежной системы.  2. Поток возвращается к этапу 9 основного потока. |
| Подпотоки | Не определены |
| Приоритет | Критично |
| Частота использования | Часто |

Декомпозиция варианта использования представлена на рисунке 1.17.



1. 1.17 – Декомпозиция варианта использование «Написание отзыва»

Декомпозируем вариант использования “Работа с мероприятиями”:

1. добавление отзыва – ввод информации об отзыве на мероприятие;
2. изменение отзыва – ввод новой информации об отзыве;
3. удаление отзыва – удаление отзыва из ИС.

В таблице 1.13 представлено описание алгоритма варианта использования «Добавление отзыва в виде текстовых потоков.

* + - 1. 1.13 – Описание алгоритма варианта использования «Добавление отзыва»

|  |  |
| --- | --- |
| Действующие лица | Пользователь |
| Краткое описание | Пользователь пишет отзыв на мероприятие |
| Предусловия | Запущена система |
| Постусловия | Пользователь завершил общение с системой |
| Нормальное течение | 1. Пользователь выбирает поиск мероприятий 2. Система открывает каталог. 3. Пользователь выбирает мероприятие. 4. А1. [Пользователь выбирает другое мероприятие] 5. Пользователь нажимает кнопку «Написать отзыв». 6. Пользователь вводит отзыв. 7. А2. [Пользователь возвращается к списку мероприятий]. 8. Вариант использования завершается. |
| Альтернативные течения | **А1. [Пользователь выбирает другое мероприятие.**  Поток возвращается к этапу 2 основного потока  **А2. Пользователь решает к списку мероприятий.**  1. Поток возвращается к этапу 2 основного потока. |
| Исключительные течения |  |
| Подпотоки | Не определены |
| Приоритет | Критично |
| Частота использования | Часто |

В таблице 1.14 представлено описание алгоритма варианта использования «Изменение отзыва» в виде текстовых потоков.

* + - 1. 1.14 – Описание алгоритма варианта использования «Изменение отзыва»

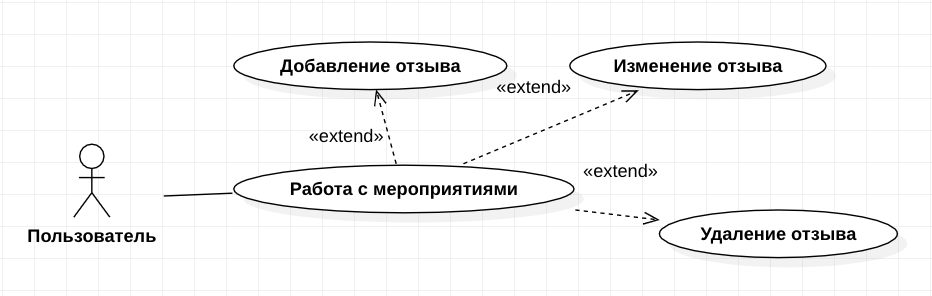
|  |  |
| --- | --- |
| Действующие лица | Пользователь |
| Краткое описание | Пользователь редактирует отзыв |
| Предусловия | Запущена система |
| Постусловия | Пользователь завершил общение с системой |
| Нормальное течение | 1. Пользователь выбирает поиск мероприятий 2. Система открывает каталог. 3. Пользователь выбирает мероприятие. 4. А1. [Пользователь выбирает другое мероприятие] 5. Пользователь нажимает кнопку «Написать отзыв». 6. Пользователь вводит отзыв. 7. А2. [Пользователь возвращается к списку мероприятий]. 8. Вариант использования завершается. |
| Альтернативные течения | **А1. [Пользователь выбирает другое мероприятие.**  Поток возвращается к этапу 2 основного потока  **А2. Пользователь решает к списку мероприятий.**  1. Поток возвращается к этапу 2 основного потока. |
| Исключительные течения |  |
| Подпотоки | Не определены |
| Приоритет | Критично |
| Частота использования | Часто |

В таблице 1.15 представлено описание алгоритма варианта использования «Удаление отзыва» в виде текстовых потоков.

* + - 1. 1.15 – Описание алгоритма варианта использования «Удаление отзыва»

|  |  |
| --- | --- |
| Действующие лица | Пользователь |
| Краткое описание | Пользователь удаляет отзыв |
| Предусловия | Запущена система |
| Постусловия | Пользователь завершил общение с системой |
| Нормальное течение | 1. Пользователь выбирает поиск мероприятий 2. Система открывает каталог. 3. Пользователь выбирает мероприятие. 4. А1. [Пользователь выбирает другое мероприятие] 5. Пользователь нажимает кнопку «Удалить отзыв». 6. А2. [Пользователь возвращается к списку мероприятий]. 7. Вариант использования завершается. |
| Альтернативные течения | **А1. [Пользователь выбирает другое мероприятие.**  Поток возвращается к этапу 2 основного потока  **А2. Пользователь решает к списку мероприятий.**  Поток возвращается к этапу 2 основного потока. |
| Исключительные течения |  |
| Подпотоки | Не определены |
| Приоритет | Критично |
| Частота использования | Часто |

Декомпозиция варианта использования представлена на рисунке 1.18.



1. 1.18 – Декомпозиция варианта использование «Работа с мероприятиями»

### 1.3.3 Требования к информационности

Требования к информационности представляют собой данные, которые хранятся и используются в проектируемой системе. Данные приведены в таблице 1.\* ниже.

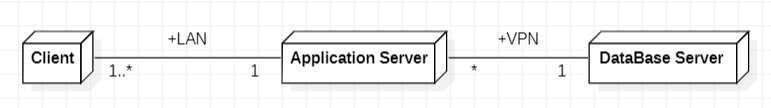
1. 1.19 – Данные для работы проектируемой системы

|  |  |  |
| --- | --- | --- |
| Название | Класс | Место хранения |
| Данные с информацией о мероприятии | Входная | База данных |
| Данные с информацией о билете | Входная | База данных |
| Данные с информацией об сеансе отзыве | Входная | База данных |
| Данные о билете | Выходная | Личное хранилище |

### 1.3.4 Архитектурные требования

Проектируемая система должна иметь клиент-серверную архитектуру, состоящую из трех звеньев: клиентская часть, серверная часть и сервер базы данных. Диаграмма развертывания представлена на рисунке ниже.

На рисунке 1.\*, Client – клиентская часть приложения, ApplicationServer – серверная часть, DataBase Server – сервер базы данных.



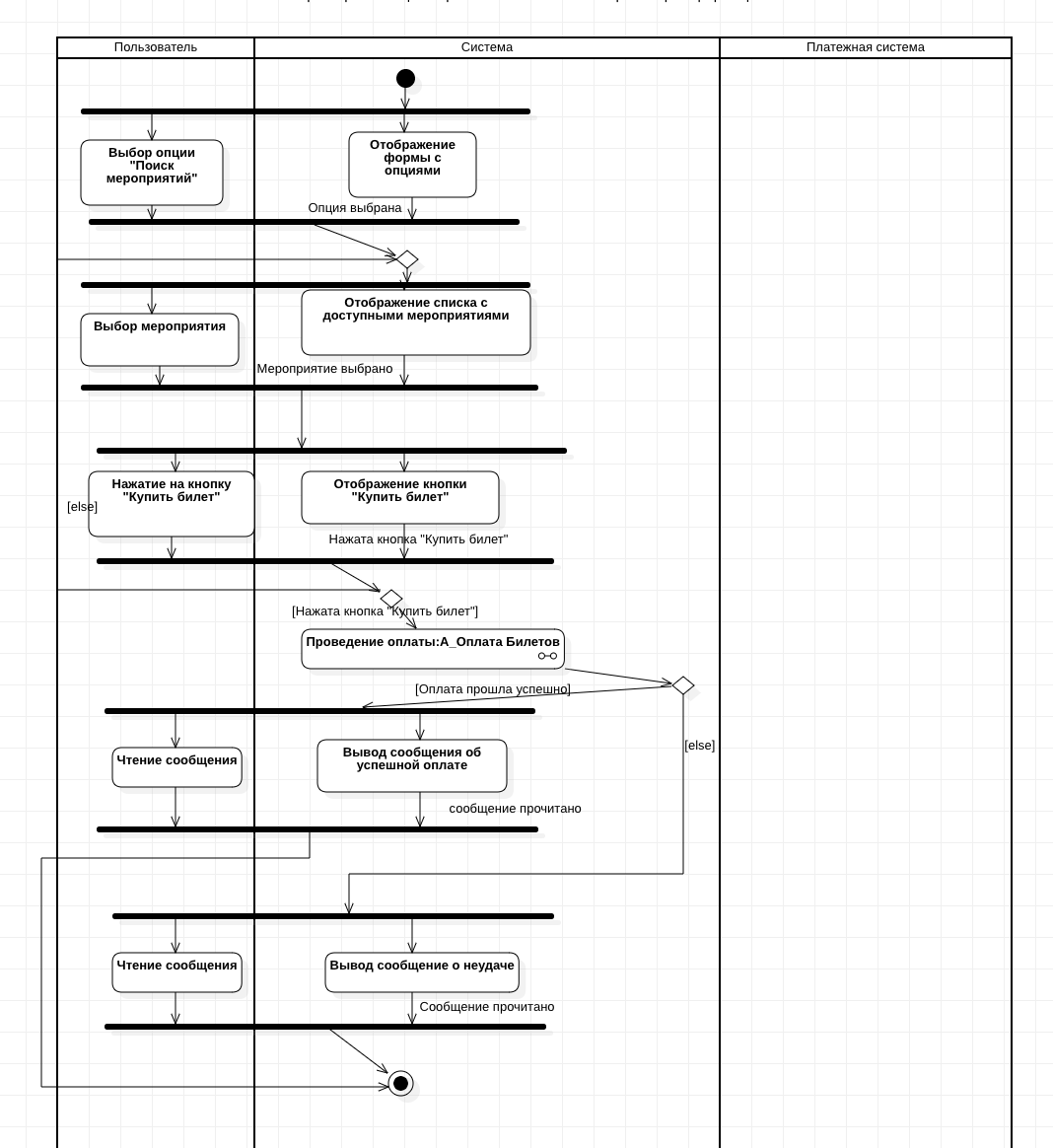
1. 1.21 – Диаграмма развертывания уровня типов с типами коммуникационной связи

Общение между клиентской частью и серверной происходит в локальной вычислительной сети (LAN - local area network), а между серверной частью и базой данных через виртуальную частную сеть (vpn = virtual private network).

### 1.3.5 Требования к поведению

Поведение проектируемой системы должно удовлетворять алгоритмам реализации, представленным в текстовой форме в пункте 1.3.2 Функциональные требования», а также диаграмма автоматов, приведенным в пункте 1.1.4 «Аспект поведения» Приведем диаграммы деятельности для наиболее важных прецедентов.

На рисунке 1.22 представлена диаграмма деятельность прецедента «Покупка билетов».

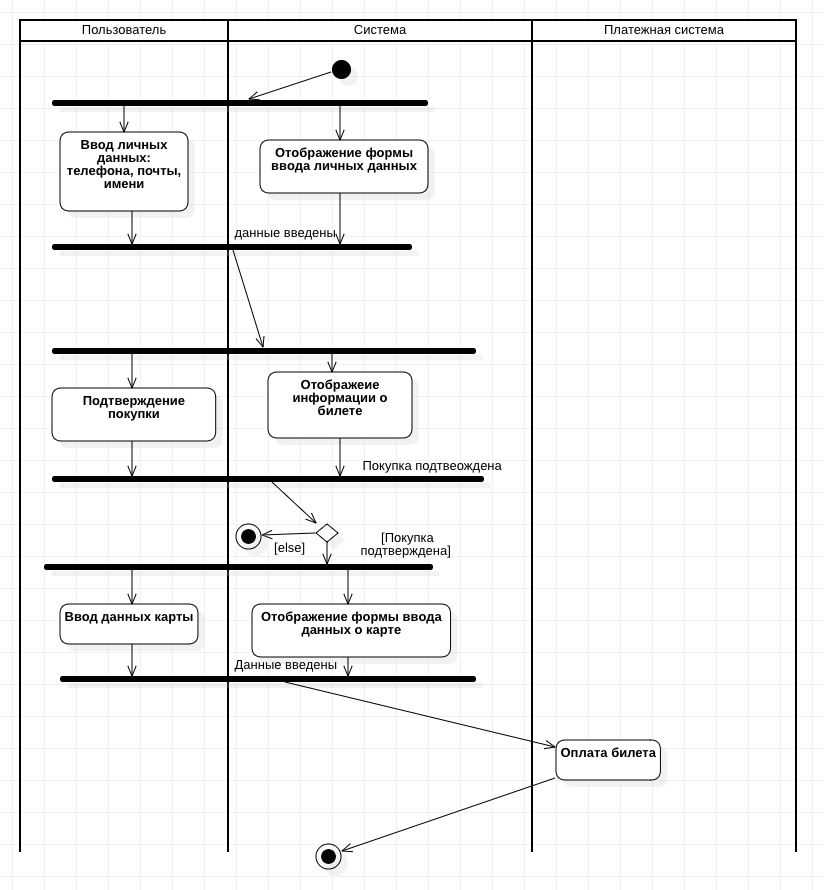


1. 1.22 – Деятельность «Покупка билета»

Ниже представлено описание приведенных действий и деятельностей:

1. выбор опции "Поиск мероприятий" - выбор одной опции из предложенных на отображенной форме. Реализация не приводятся в силу простоты действия;
2. отображение формы с опциями - отображение формы с опцией поиск мероприятий. Реализация не приводятся в силу простоты действия;
3. нажатие на кнопку "Купить билет" – нажатие на кнопку для перехода к следующему этапу покупки билета. Реализация не приводятся в силу простоты действия;
4. отображение кнопки "Купить билет" - Отображение кнопки купить билет для выбранного мероприятия. Реализация не приводятся в силу простоты действия;
5. проведение оплаты - проведение оплаты через платежную систему. Реализация приведена на рисунке 1.23;
6. чтение сообщения - чтение пользователем сообщения о результатах оплаты. Реализация не приводятся в силу простоты действия;
7. вывод сообщения об успешной оплате - вывод в текстовое поле сообщение об удачной оплате. Реализация не приводятся в силу простоты действия;
8. вывод сообщение о неудаче - вывод в текстовое поле сообщение о неудачной оплате. Реализация не приводятся в силу простоты действия.

На рисунке 1.23 представлена деятельность “Оплата билета”.



1. 1.23 – Деятельность «Оплата билета»

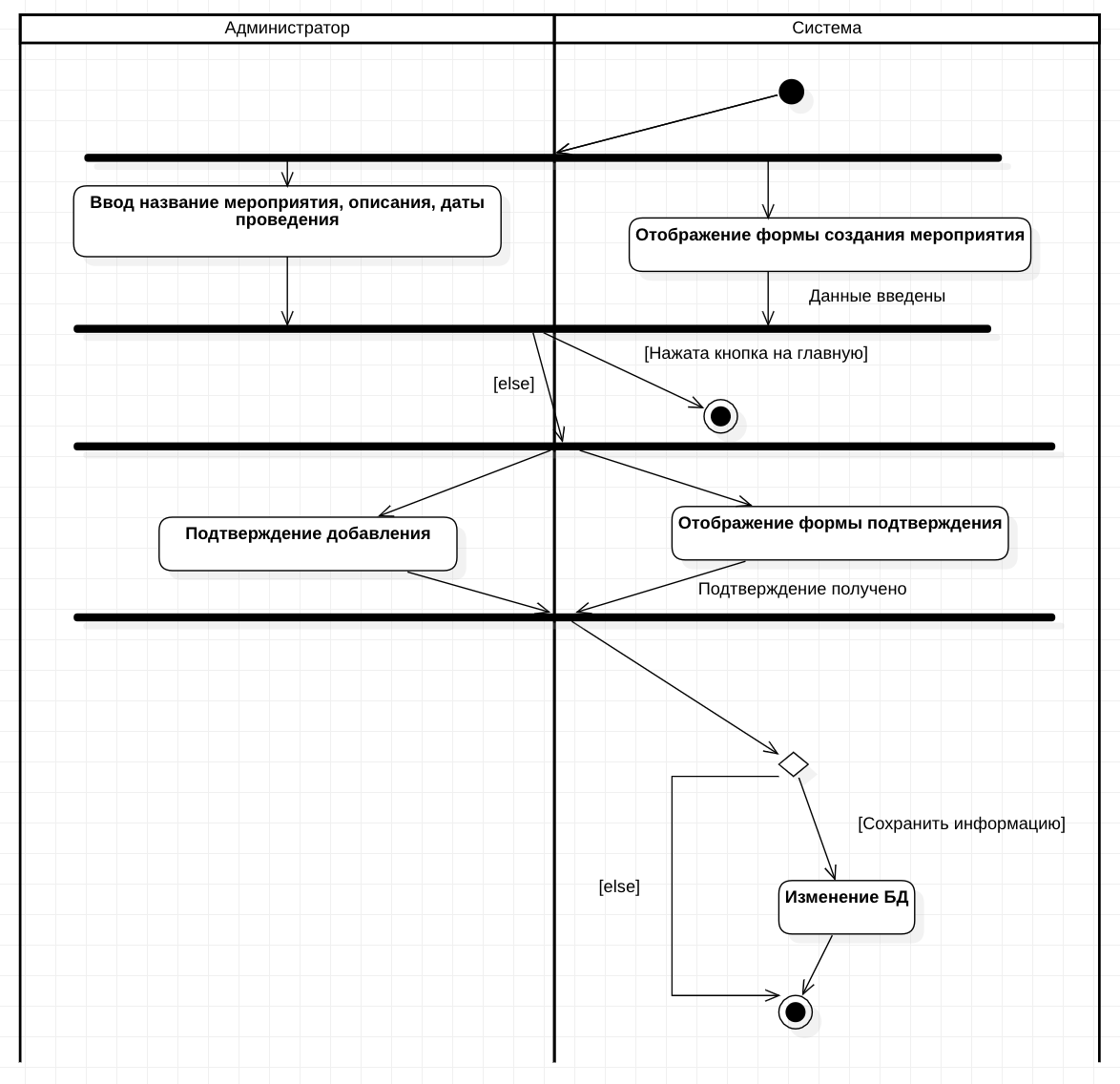
Описание диаграммы деятельности «Оплата билетов», представленной на рисунке 1.23:

1. ввод личных данных: телефона, почты, имени - ввод информации о пользователе, а именно: телефона, почты и имени. Реализация не приводятся в силу простоты действия;
2. отображение формы ввода личных данных - отображение формы для ввода телефона, почты и имени. Реализация не приводятся в силу простоты действия;
3. подтверждение покупки - подтверждение пользователем того, что данные в билет внесены верно. Реализация не приводятся в силу простоты действия;
4. отображение информации о билете - отображение информации о билете (мероприятие, дата) для подтверждения пользователем. Реализация не приводятся в силу простоты действия;
5. ввод данных карты - ввод номера, владельца, срока действия и cvc кода карты. Реализация не приводятся в силу простоты действия;
6. отображение формы ввода данных о карте - отображение формы ввода номера, владельца, срока действия и cvc кода карты. Реализация не приводятся в силу простоты действия;
7. оплата билета - оплата билета в платежной системе. Реализация не приводятся так как деятельность происходит в собственной системе банка.

В случае отсутствия каких-либо надписей над стрелками, переход происходит безальтернативно. Если над стрелкой есть подпись в квадратных скобках, переход происходит только при выполнении условия, указанного внутри этих скобок. В случае подписи без скобок, подпись является триггерным событием, после наступления которого осуществляется переход.

Приведем диаграммы деятельности для заполнения системы информацией: добавление в базу данных мероприятий, пользователей, отзывов.

На рисунке 1.24 представлена Деятельность «Добавление мероприятия»



**Ошибки здесь и далее! 1. Ветвление из линии синхрогизации не осуществимо! Выйти надо было из узла отображения.**

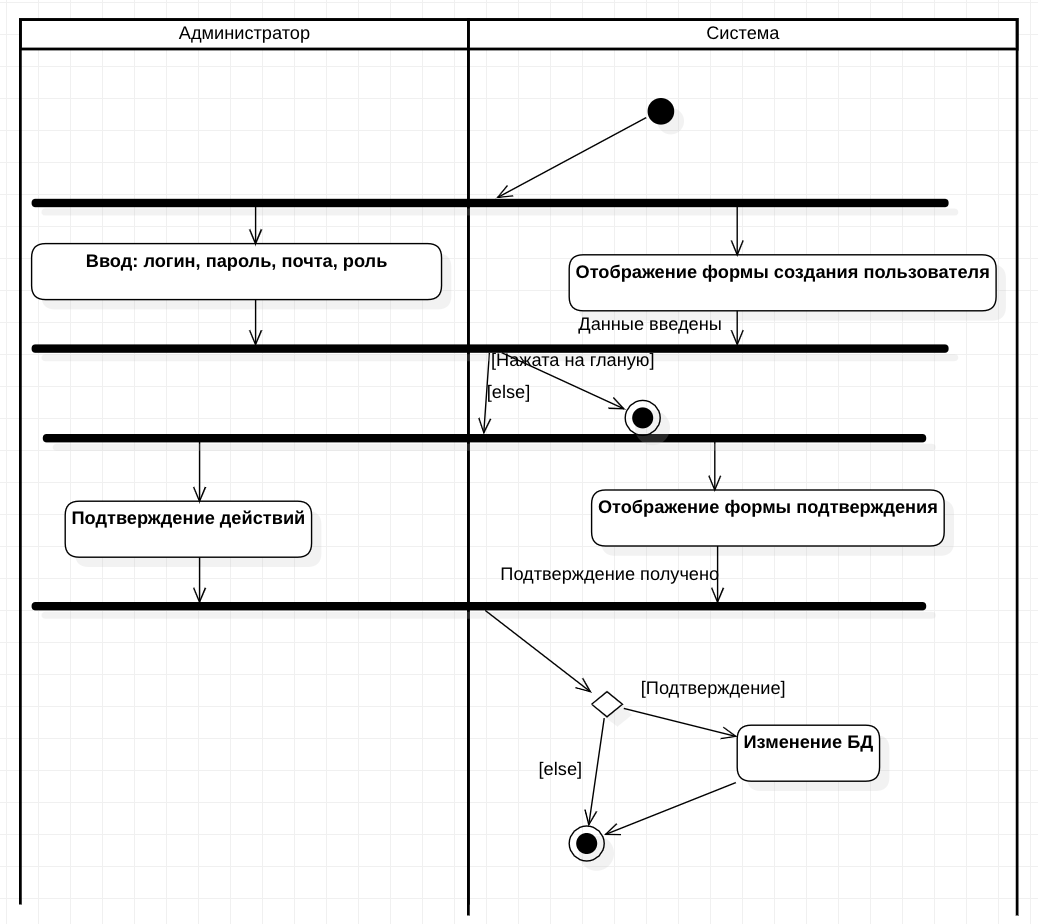
**2. В узел «Подтверждение…» надо добавить «Решение о сохранении информации»**

1. 1.24 – Деятельность «Добавление мероприятия»

На диаграмме, изображенной на 1.24 представлены следующие действия:

1. ввод: название мероприятия, описания, даты проведения и заполнение аналогичных полей. Реализация не приводятся в силу простоты действия;
2. отображение формы создание мероприятия - отображение формы с полями: название мероприятия, описания, даты проведения. Реализация не приводятся в силу простоты действия;
3. подтверждение добавления - нажатие на кнопку подтверждение согласия на внесение изменения в базу данных. Реализация не приводятся в силу простоты действия;
4. отображение формы подтверждения - отображение формы подтверждения изменений в БД;
5. изменение БД - Изменение данных в БД. Реализация не приводятся так как выполняется сторонней системой.

На рисунке 1.25 представлена Деятельность «Добавление пользователя».

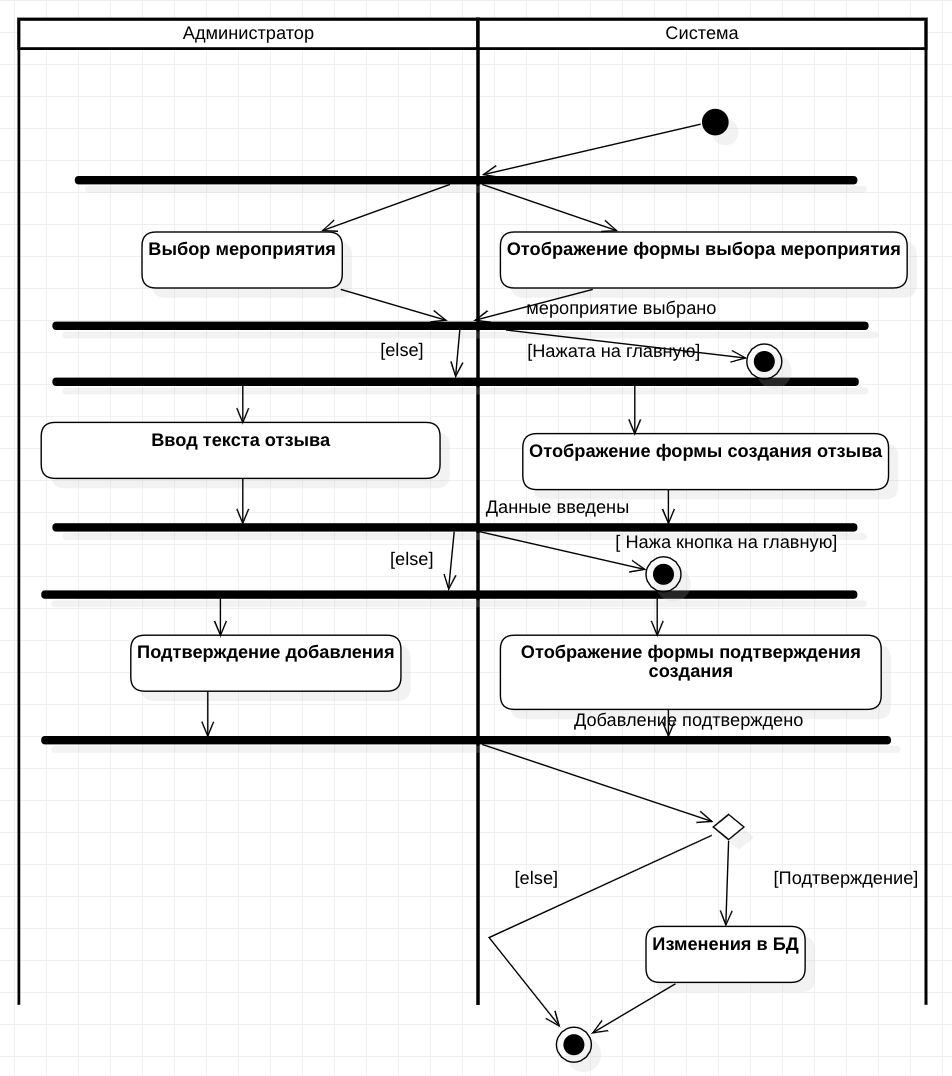


1. 1.25 – Деятельность «Добавление пользователя»

На диаграмме, изображенной на 1.25 представлены следующие действия:

1. ввод: логин, пароль, почта, роль. Реализация не приводятся в силу простоты действия.
2. отображение формы создание мероприятия - отображение формы с полями: логин, пароль, почта, роль. Реализация не приводятся в силу простоты действия.
3. подтверждение добавления - нажатие на кнопку подтверждение согласия на внесение изменения в базу данных. Реализация не приводятся в силу простоты действия.
4. отображение формы подтверждения - отображение формы подтверждения изменений в БД.
5. изменение БД - Изменение данных в БД. Реализация не приводятся так как выполняется сторонней системой.

На рисунке 1.26 представлена Деятельность «Добавление отзыва».



1. 1.26 – Деятельность «Добавление отзыва»

На диаграмме, изображенной на 1.26 представлены следующие действия:

1. выбор мероприятия – выбор мероприятия из списка на форме. Реализация не приводятся в силу простоты действия.
2. отображение формы выбора мероприятия – список из мероприятий.
3. ввод: текст отзыва. Реализация не приводятся в силу простоты действия.
4. отображение формы создание отзыва - отображение формы с текстом отзыва. Реализация не приводятся в силу простоты действия.
5. подтверждение добавления - нажатие на кнопку подтверждение согласия на внесение изменения в базу данных. Реализация не приводятся в силу простоты действия.
6. отображение формы подтверждения - отображение формы подтверждения изменений в БД.
7. изменение БД - Изменение данных в БД. Реализация не приводятся так как выполняется сторонней системой.

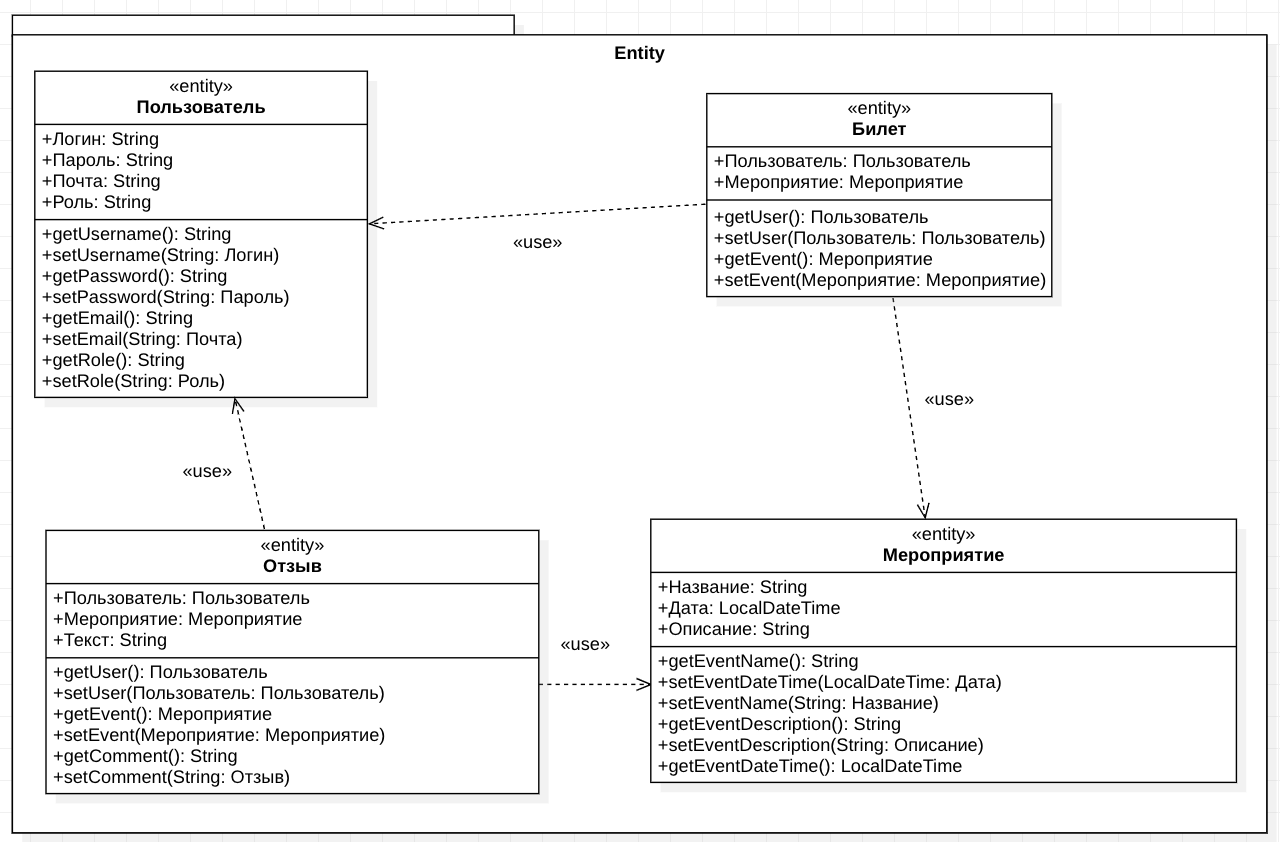
В случае отсутствия каких-либо надписей над стрелками, переход происходит безальтернативно. Если над стрелкой есть подпись в квадратных скобках, переход происходит только при выполнении условия, указанного внутри этих скобок. В случае подписи без скобок, подпись является триггерным событием, после наступления которого осуществляется переход.

# 2 Проектирование системы

## 2.1 Архитектура данных и процессов

Для представления архитектуры данных и процессов реализуем диаграмму классов. Разделим данную диаграмму на пакеты по типам классов: классы-сущности, управляющие классы, граничащие классы.

К классам-сущностям относятся классы Мероприятие, Отзыв, Билет, Пользователь.



1. 2.1 – Классы-сущности

На рисунке 2.1 представлена подробная диаграмма классов пакета Entity (сущности).

**Класс Пользователь**

Рассмотрим поля.

1. Логин – логин для авторизации в системе, режим доступа private, тип строка.
2. Пароль – пароль для авторизации в системе, режим доступа private, тип строка.
3. Почта - почта для отправления информации по аккаунту, режим доступа private, тип строка.
4. Роль - роль пользователя (админ или пользователь), режим доступа private, тип строка.

Рассмотрим методы класса.

1. getUsername(): String – метод для получения имени пользователя, режим доступа public, возвращаемое значение – строка.
2. setUsername(String:Логин) – метод для установки имени пользователя, режим доступа public, входное значение – строка.
3. getPassword(): String – метод для получения пароля пользователя, режим доступа public, возвращаемое значение – строка.
4. setPassword(String:Пароль) – метод для установки пароля пользователя, режим доступа public, входное значение – строка.
5. getEmail(): String – метод для получения почты пользователя, режим доступа public, возвращаемое значение – строка.
6. setEmail(in String:Почта) – метод для установки почты пользователя, режим доступа public, входное значение – строка.
7. getRole(): String – метод для получения роли пользователя, режим доступа public, возвращаемое значение – строка.
8. setRoleString:Роль) – метод для установки роли пользователя, режим доступа public, входное значение – строка.

**Класс Мероприятие**

Рассмотрим поля.

1. Название – название мероприятия, режим доступа private, тип строка.
2. Описание – описание мероприятия, режим доступа private, тип строка.
3. Дата - дата проведения мероприятия, режим доступа private, тип LocalDateTime.

Рассмотрим методы класса.

1. getEventName(): String – метод для получения имени пользователя, режим доступа public, возвращаемое значение – строка.
2. setEventName(String:Название) – метод для установки имени пользователя, режим доступа public, входное значение – строка.
3. getEventDescription(): String – метод для получения пароля пользователя, режим доступа public, возвращаемое значение – строка.
4. setEventDescription(String:Описание) – метод для установки пароля пользователя, режим доступа public, входное значение – строка.
5. getEventDateTime(): String – метод для получения почты пользователя, режим доступа public, возвращаемое значение – LocalDateTime.
6. setEventDateTIme(in String:Дата) – метод для установки почты пользователя, режим доступа public, входное значение – LocalDateTime.

**Класс Билет**

Рассмотрим поля.

1. Пользователь – пользователь, купивший билет, режим доступа private, тип Пользователь.
2. Мероприятие – мероприятие, на которое пользователь купил билет, режим доступа public, тип private.

Рассмотрим методы класса.

1. getUser(): Пользователь – метод для получения имени пользователя, режим доступа public, возвращаемое значение – Пользователь.
2. setUser(Пользователь: Пользователь) – метод для установки имени пользователя, режим доступа public, входное значение – Пользователь.
3. getEvent(): Мероприятие – метод для получения пароля пользователя, режим доступа public, возвращаемое значение – Мероприятие.
4. setEvent(Мероприятие: Мероприятие) – метод для установки пароля пользователя, режим доступа public, входное значение – Мероприятие.

**Класс Отзыв**

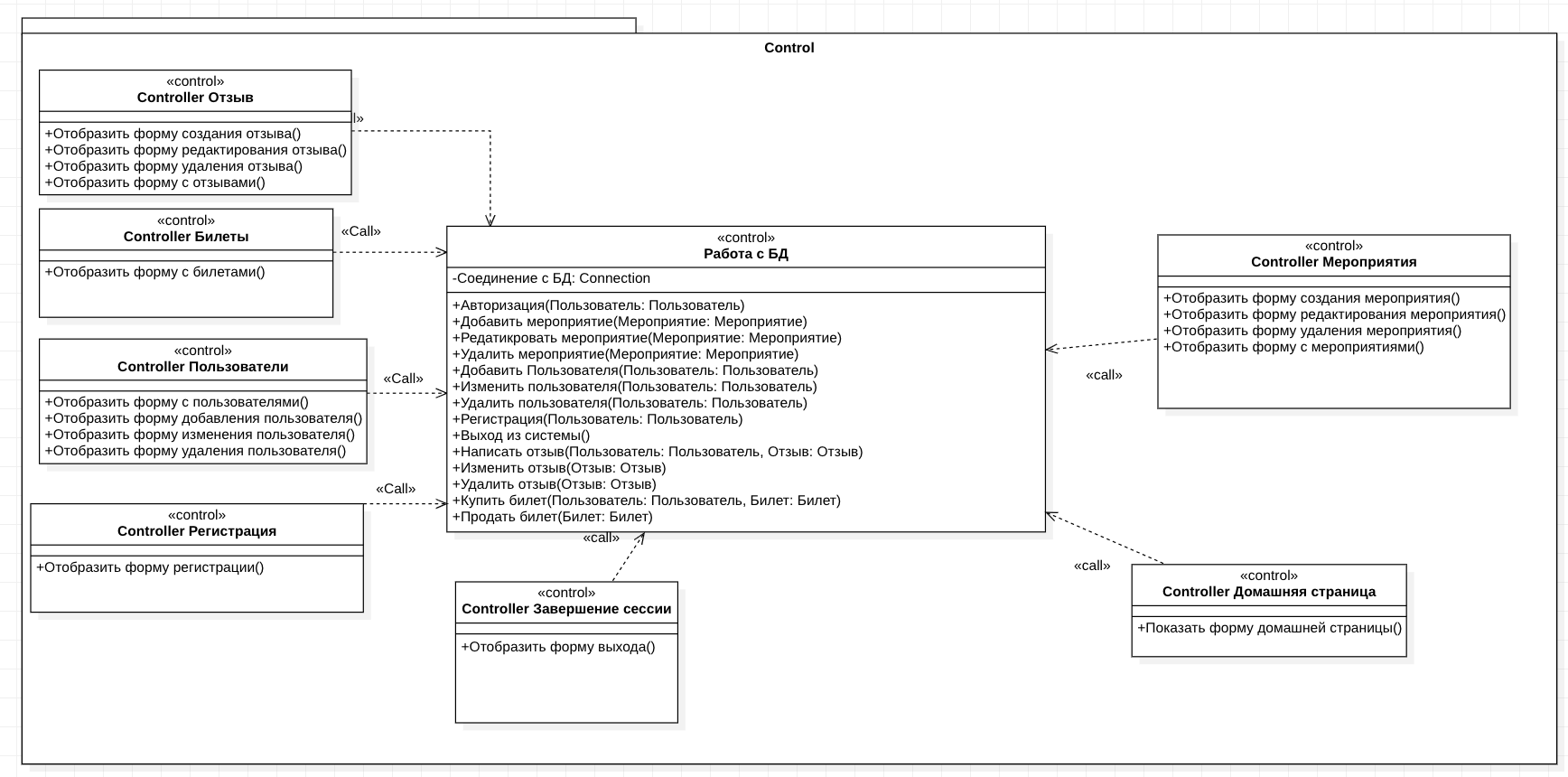
Рассмотрим поля.

1. Пользователь – пользователь, написавший отзыв, режим доступа private, тип Пользователь.
2. Мероприятие – мероприятие, на которое пользователь написал отзыв, режим доступа private, тип Мероприятие.
3. Текст - текст отзыва, режим private, тип строка.

Рассмотрим методы класса.

1. getUser(): Пользователь – метод для имени пользователя, режим доступа public, возвращаемое значение – Пользователь.
2. setUser(Пользователь:Пользователь) – метод для установки пользователя, режим доступа public, входное значение – Пользователь.
3. getEvent(): Мероприятие – метод для получения мероприятия, режим доступа public, возвращаемое значение – Мероприятие.
4. setEvent(Мероприятие:Мероприятие) – метод для установки мероприятия, режим доступа public, входное значение – Мероприятие.
5. getComment(): String – метод для получения текста отзыва, режим доступа public, возвращаемое значение – строка.
6. setComment(String: Текст) – метод для установки текста отзыва, режим доступа public, входное значение – строка.

Ниже представлены управляющие классы (через них меняем данные).



1. 2.2 – Управляющие классы

**Класс-контроллер Отзыв**

Рассмотрим методы класса.

1. createReviewsPage() – отображает список мероприятий, режим доступа public.
2. createAddReviewPage() – отображает форму создания отзыва, вызывает метод для создания отзыва из контроллера Работа с БД, режим доступа public.
3. createEditReviewPage() – отображает форму редактирования отзыва, вызывает метод для редактирования отзыва из контроллера Работа с БД. отзыва из контроллера Работа с БД, режим доступа public.
4. createDeleteReviewPage() – отображает форму создания отзыва, вызывает метод для создания отзыва из контроллера Работа с БД, режим доступа public.

**Класс-контроллер Пользователи**

Рассмотрим методы класса.

1. createUsersPage() – отображает список мероприятий, режим доступа public.
2. createAddUserPage() – отображает форму создания отзыва, вызывает метод для создания отзыва из контроллера Работа с БД, режим доступа public.
3. createEditUserPage() – отображает форму редактирования отзыва, вызывает метод для редактирования отзыва из контроллера Работа с БД. отзыва из контроллера Работа с БД, режим доступа public.
4. createDeleteUserPage() – отображает форму создания отзыва, вызывает метод для создания отзыва из контроллера Работа с БД, режим доступа public.

**Класс-контроллер Мероприятия**

Рассмотрим методы класса.

1. createEventsPage() – отображает список мероприятий, режим доступа public.
2. createAddEventPage() – отображает форму создания отзыва, вызывает метод для создания отзыва из контроллера Работа с БД, режим доступа public.
3. createEditEventPage() – отображает форму редактирования отзыва, вызывает метод для редактирования отзыва из контроллера Работа с БД. отзыва из контроллера Работа с БД, режим доступа public.
4. createDeleteEventPage() – отображает форму создания отзыва, вызывает метод для создания отзыва из контроллера Работа с БД, режим доступа public.

**Класс-контроллер Завершение сессии**

Рассмотрим метод класса.

createLogoutPage() – отображает форму завершения сессии, вызывает метод logout() из контроллера Работа с БД, режим доступа public.

**Класс-контроллер Домашняя страница**

Рассмотрим метод класса.

createHomePage() – отображает форму домашней страницы, режим доступа public.

**Класс-контроллер Билеты**

Рассмотрим метод класса.

createTicketPage() – отображает форму с билетами, режим доступа public.

**Класс-контроллер Регистрация**

Рассмотрим метод класса.

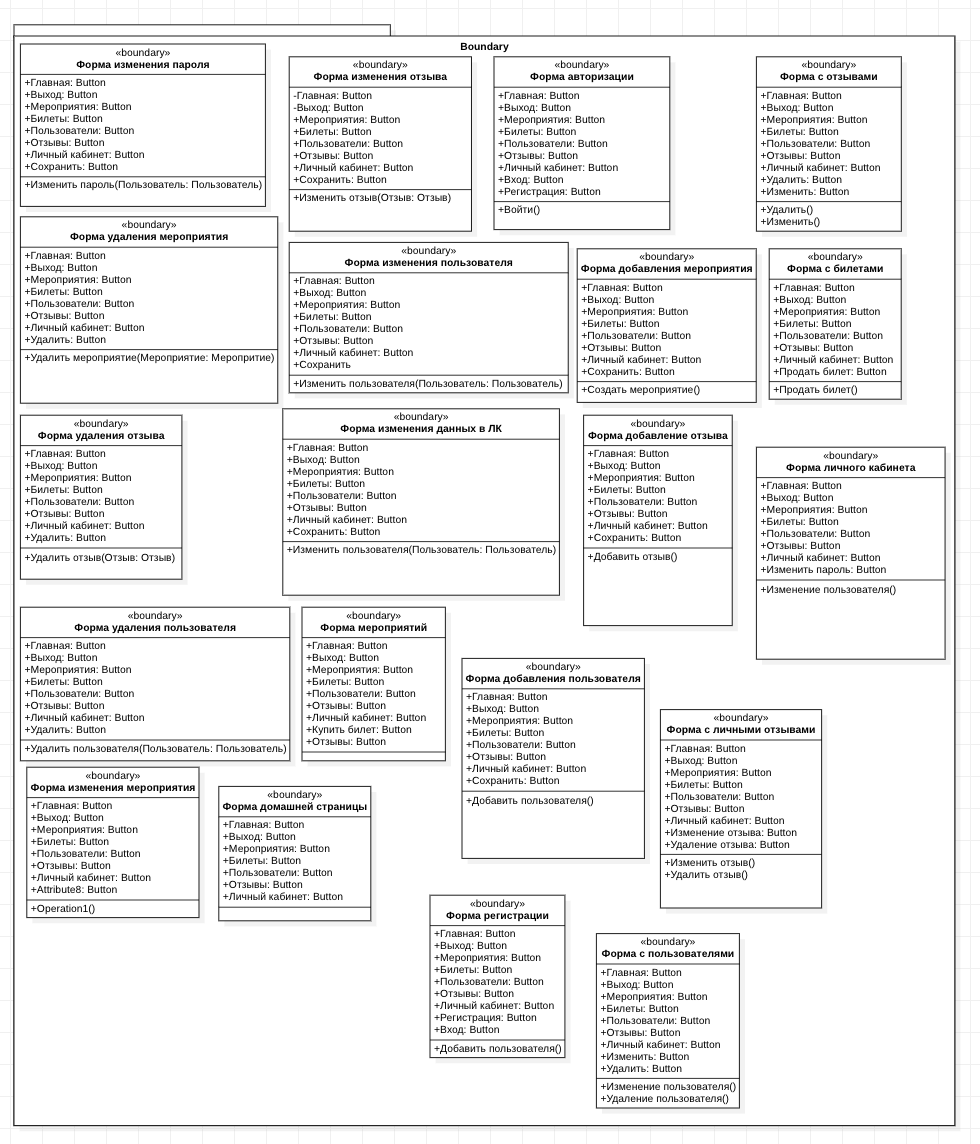
createRegistrationPage() – отображает форму регистрации, режим доступа public.

**Класс-контроллер Работа с БД**

Рассмотрим методы класса.

1. Авторизация(Пользователь:Пользователь) – авторизация пользователя в системе, режим доступа public.
2. Добавить мероприятие(Мероприятие: Мероприятие) – добавление мероприятия в базу данных, входные параметры: мероприятие, режим доступа public.
3. Изменить мероприятие(Мероприятие: Мероприятие)– изменение мероприятия в базе данных, входные параметры, режим доступа public.
4. Удалить мероприятие(Мероприятие: Мероприятие) – удаление мероприятия из базы данных, входные параметры: мероприятие, режим доступа public.
5. Добавить пользователя(Пользователь: Пользователь) – добавление пользователя в базу данных, входные параметры: пользователь, режим доступа public.
6. Изменить пользователя(Пользователь: Пользователь)– изменение пользователя в базе данных, входные параметры: пользователь, режим доступа public.
7. Удалить пользователя(Пользователь: Пользователь) – удаление пользователя из базы данных, входные параметры: пользователь, режим доступа public.
8. Добавить отзыв(Отзыв: Отзыв) – добавление отзыва в базу данных, входные параметры: отзыв, выходные, режим доступа public.
9. Изменить отзыв(Отзыв: Отзыв)– изменение отзыва в базе данных, входные параметры: отзыв, режим доступа public.
10. Удалить отзыв(Отзыв: Отзыв) – удаление отзыва из базы данных, входные параметры: отзыв, выходные, режим доступа public.
11. Купить билет(Пользователь:Пользователь) – добавление билета в базу данных, входные параметры: пользовтель, режим доступа public.
12. Продать билет(Билет: Билет) – удаление билета из базы данных, входные параметры: билет, режим доступа public.
13. Регистрация(Пользователь:Пользователь) – добавление пользователя в базу данных, входные параметры: пользователь, режим доступа public.
14. Выход из системы() – удаление пользователя из сессии, режим доступа public.

На рисунке 2.3 изображена диаграмма граничных классов с полным отображением классов.



1. 2.3 – Граничные классы

На рисунке 2.3 изображена диаграмма граничных классов с полным отображением классов.

**Форма изменения пароля**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Сохранить – кнопка для сохранения изменений, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Изменить пароль (Пользователь: Пользователь) – изменение пароля пользователя, режим доступа public.

**Форма удаления мероприятия**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Удалить – кнопка для удаления мероприятия, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Удалить мероприятие (Мероприятие: Мероприятие) – удаление мероприятия, режим доступа public.

**Форма удаления отзыва**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Удалить – кнопка для удаления отзыва, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Удалить отзыв (Отзыв: Отзыв) – Удаление отзыва пользователя, режим доступа public.

**Форма удаления пользователя**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Удалить – кнопка для удаления пользователя, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Удалить пользователя (Пользователь: Пользователь) – удаление пароля пользователя, режим доступа public.

**Форма изменения мероприятия**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Сохранить – кнопка для сохранения изменений, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Изменить мероприятие (Мероприятие: Мероприятие) – изменение мероприятия, режим доступа public.

**Форма изменения отзыва**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Сохранить – кнопка для сохранения изменений, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Изменить отзыв (Отзыв: Отзыв) – изменение отзыва пользователя, режим доступа public.

**Форма изменения пользователя**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Сохранить – кнопка для сохранения изменений, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Изменить пользователя (Пользователь: Пользователь) – изменение пользователя, режим доступа public.

**Форма изменения данных в ЛК**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Сохранить – кнопка для сохранения изменений, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Изменить (Пользователь: Пользователь) – изменение пользователя, режим доступа public.

**Форма мероприятий**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Купить билет – кнопка для покупки билета, режим доступа public, тип button (кнопка).
8. Отзывы – кнопка для просмотра отзывов о мероприятии, режим доступа public, тип button (кнопка).

**Форма домашней страницы**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).

**Форма авторизации**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Вход – кнопка входа в систему, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Регистрания – кнопка перехода на страницу регистрации, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Вход (Пользователь: Пользователь) – вход в систему, режим доступа public.

**Форма добавления мероприятия**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Сохранить – кнопка для сохранения изменений, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Добавить мероприятие (Мероприятие: Мероприятие) – добавление мероприятиия, режим доступа public.

**Форма добавления отзыва**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Сохранить – кнопка для сохранения изменений, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Добавить отзыв (Отзыв: Отзыв) – добавление мероприятиия, режим доступа public.

**Форма добавления пользователя**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Сохранить – кнопка для сохранения изменений, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Добавить пользователя (Пользователь: Пользователь) – добавление пользователя, режим доступа public.

**Форма регистрации**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Сохранить – кнопка для сохранения изменений, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Добавить пользователя (Пользователь: Пользователь) – добавление пользователя, режим доступа public.

**Форма с отзывами**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Удалить – кнопка для удаления отзыва, режим доступа public, тип button (кнопка).
8. Изменить – кнопка для измения отзыва, режим доступа public, тип button (кнопка).

Рассмотрим методы.

1. Изменить отзыв (Отзыв: Отзыв) – изменение отзыва пользователя, режим доступа public.
2. Удалить отзыв (Отзыв: Отзыв) – удаление отзыва пользователя, режим доступа public.

**Форма с билетами**

Рассмотрим поля.

Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).

Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).

Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).

Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).

Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).

Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).

Продать билет – кнопка для продажи билета, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Продать билет (Билет: Билет) – удаление билета из системы, режим доступа public.

**Форма личного кабинета**

Рассмотрим поля.

Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).

Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).

Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).

Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).

Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).

Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).

Изменить пароль – кнопка для изменения пароля, режим доступа public, тип button (кнопка).

Рассмотрим метод.

Изменить пароль (Пользователь: Пользователь) – изменение пароля пользователя, режим доступа public.

**Форма с личными отзывами**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Удалить – кнопка для удаления отзыва, режим доступа public, тип button (кнопка).
8. Изменить – кнопка для измения отзыва, режим доступа public, тип button (кнопка).

Рассмотрим методы.

Изменить отзыв (Отзыв: Отзыв) – изменение отзыва пользователя, режим доступа public.

Удалить отзыв (Отзыв: Отзыв) – удаление отзыва пользователя, режим доступа public.

**Форма с пользователями**

Рассмотрим поля.

1. Главная – кнопка перехода на главну. страницу, режим доступа public, тип button (кнопка).
2. Выход – кнопка выхода из системы, режим доступа public, тип button (кнопка).
3. Мероприятия – кнопка перехода на страницу с мероприятиями, режим доступа public, тип button (кнопка).
4. Билеты – кнопка перехода на страницу с билетами, режим доступа public, тип button (кнопка).
5. Отзывы – кнопка перехода на страницу с отзывами пользователя, режим доступа public, тип button (кнопка).
6. Личный кабинет – кнопка перехода на страницу личного кабинета, режим доступа public, тип button (кнопка).
7. Изменить – кнопка для изменения пользователя, режим доступа public, тип button (кнопка).
8. Изменить – кнопка для удаления пользователя, режим доступа public, тип button (кнопка).

Рассмотрим методы.

1. Изменить пользователя (Пользователь: Пользователь) – изменение пользователя, режим доступа public.
2. Удалить пользователя (Пользователь: Пользователь) – удаление пользователя, режим доступа public.

Отношения между классами не приводятся ввиду наличия подробной документами зависимостей между классами.

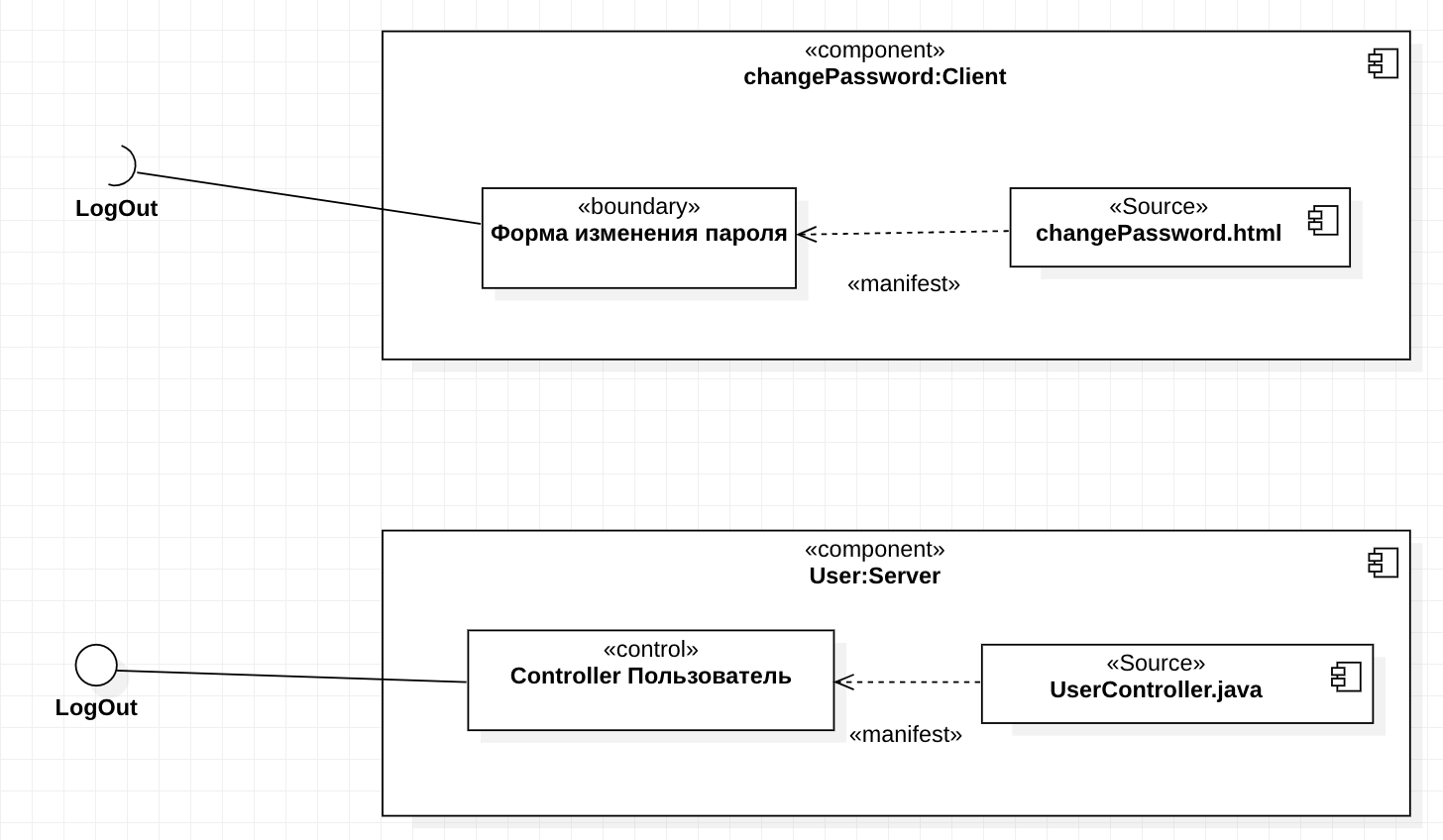
## 2.1 Модель поведения

Алгоритмы поведения представлены на рисунках 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13, 1.14, 1.15. Детальное представление основных алгоритмов в виде диаграмм деятельности представлены на рисунках 1.22, 1.23, 1.24, 1.25, 1.26.

## 2.2 Модель реализации

### 2.2.1 Модель исходного кода

На рисунке 2.19 приведена диаграмма компонентов исходного кода для прецедента Изменение пароля.



1. 2.19 – Модель исходного кода для реализации прецедента «Авторизация»

Прецедент Изменение пароля должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели changePassword:Client;

2) компонент серверной части, имя класса в модели User:Server;

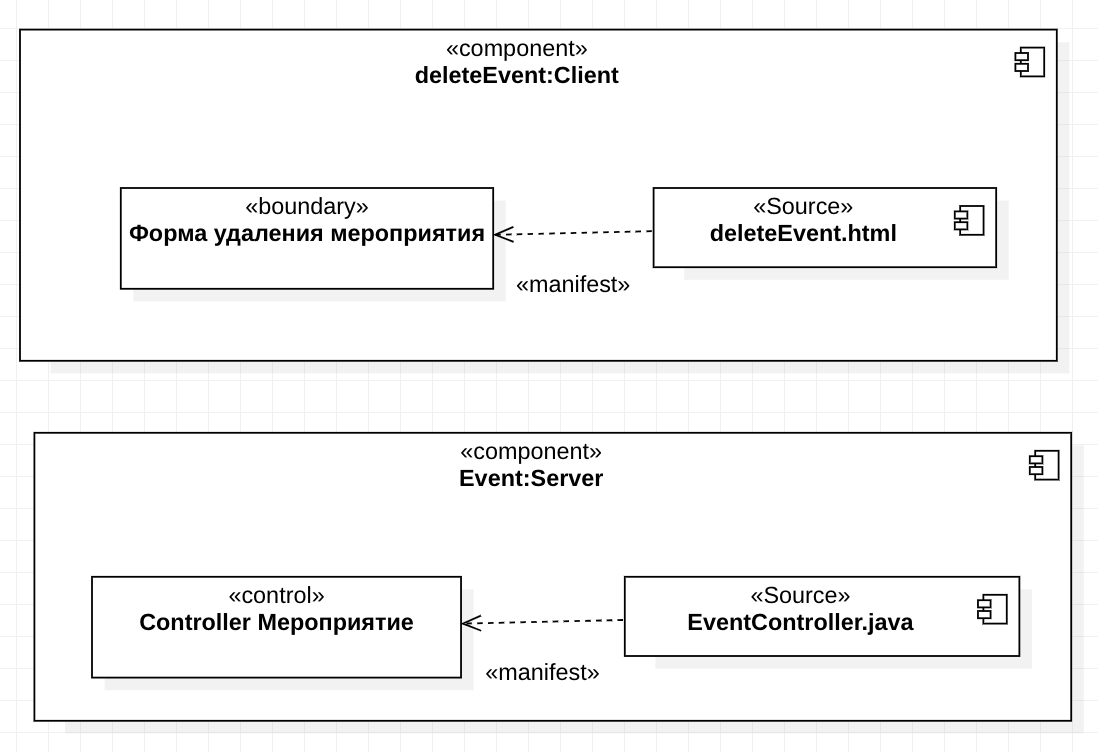
– функциональность компонента changePassword:Client должна быть реализована в виде страницы с именем Форма авторизации;

– исходный программный код страницы changePassword должен быть размещён в файле с именем changePassword.html;

– функциональность компонента User:Server должна быть реализована в виде класса с именем Controller Пользователь;

– исходный программный код класса Controller Пользователь должен быть размещён в файле с именем UserController.java;

На рисунке 2.20 приведена диаграмма компонентов исходного кода для прецедента Удаление мероприятия.



1. 2.20 – Модель исходного кода для реализации прецедента «Удаление мероприятия»

Прецедент Удаление мероприятия должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели deleteEvent:Client;

2) компонент серверной части, имя класса в модели Event:Server;

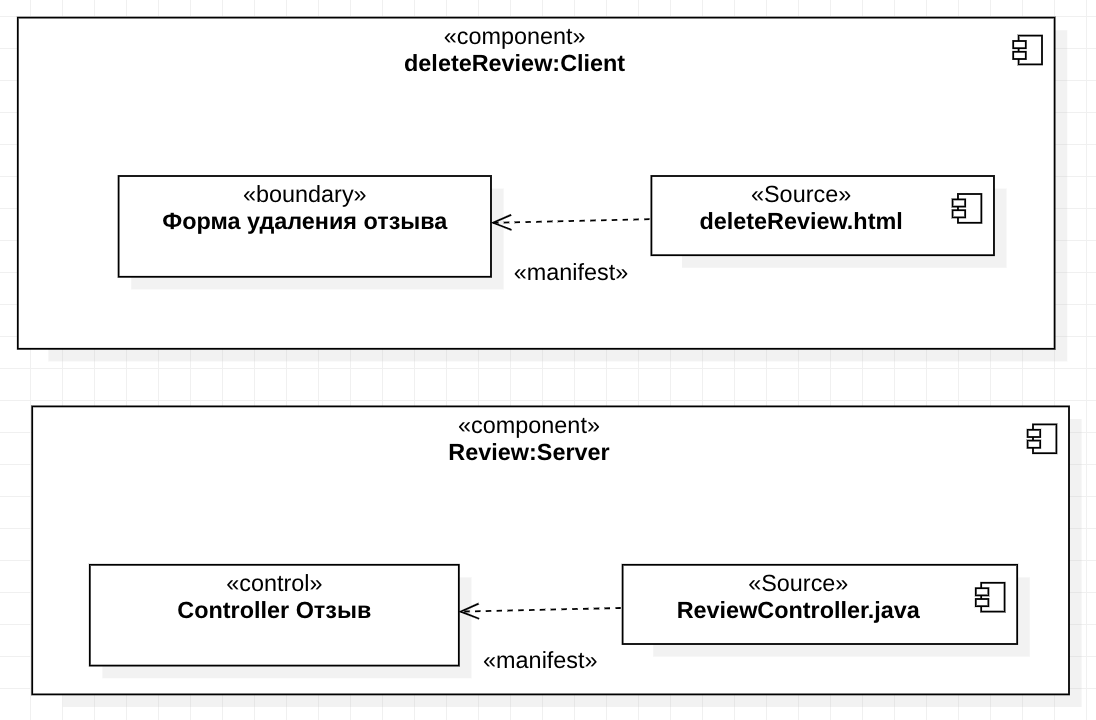
– функциональность компонента deleteEvent:Client должна быть реализована в виде класса с именем «Удаление мероприятия»;

– исходный программный код класса «Удаление мероприятия» должен быть размещён в форме с именем deleteEvent.html;

– функциональность компонента Event:Server должна быть реализована в виде класса с именем «Controller Мероприятия»;

– исходный программный код класса «Controller Меропрития» должен быть размещён в файле с именем EventController.java;

На рисунке 2.21 приведена диаграмма компонентов исходного кода для прецедента «Удаление отзыва».



1. 2.21 – Модель исходного кода для реализации прецедента «Удаление отзыва»

Прецедент Удаление отзыва должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели deleteReview:Client;

2) компонент серверной части, имя класса в модели Review:Server;

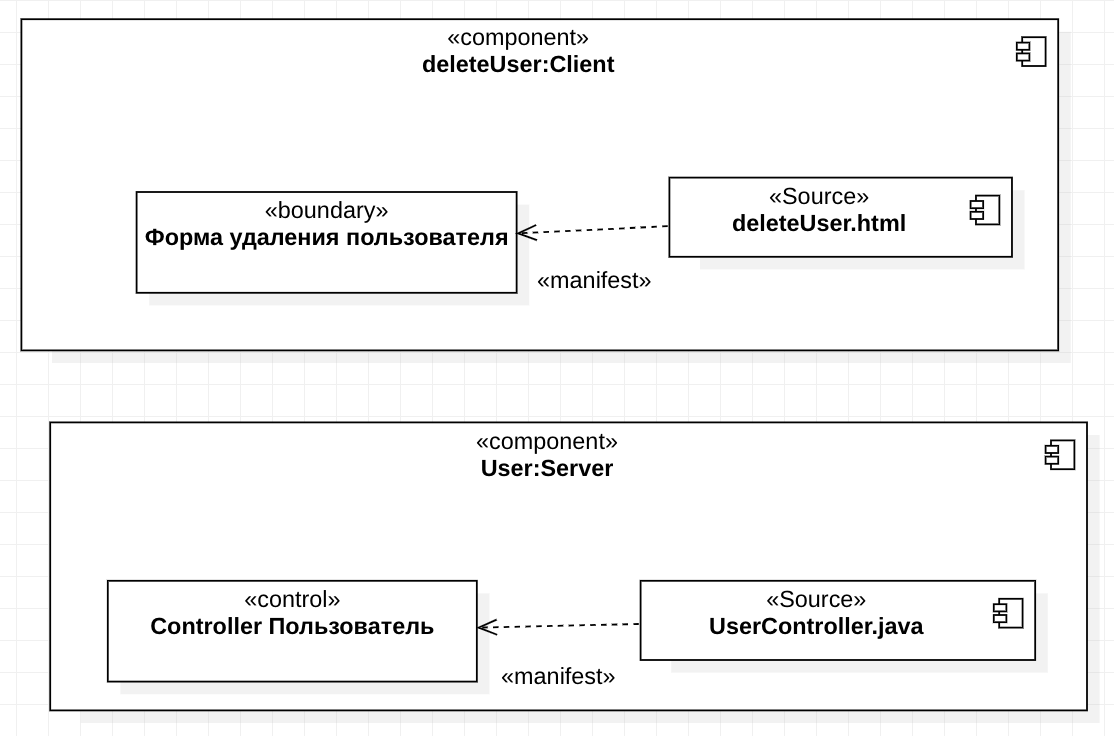
– функциональность компонента deleteReview:Client должна быть реализована в виде класса с именем «Удаление мероприятия»;

– исходный программный код формы Удаление отзыва должен быть размещён в форме с именем deleteReview.html;

– функциональность компонента Event:Server должна быть реализована в виде класса с именем «Controller Мероприятия»;

– исходный программный код класса «Controller Отзывы должен быть размещён в файле с именем ReviewController.java;

На рисунке 2.22 приведена диаграмма компонентов исходного кода для прецедента «Удаление пользователя.



1. 2.22 – Модель исходного кода для реализации прецедента «Удаление отзыва»

Прецедент Удаление отзыва должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели deleteUser:Client;

2) компонент серверной части, имя класса в модели User:Server;

– функциональность компонента deleteUser:Client должна быть реализована в виде класса с именем «Удаление пользователя»;

– исходный программный код формы Удаление отзыва должен быть размещён в форме с именем deleteUser.html;

– функциональность компонента User:Server должна быть реализована в виде класса с именем «Controller Пользователи;

– исходный программный код класса «Controller Пользователи должен быть размещён в файле с именем UserController.java;

На рисунке 2.23 приведена диаграмма компонентов исходного кода для прецедента Изменение пользователя.



1. 2.23 – Модель исходного кода для реализации прецедента «Изменение пользователя»

Прецедент Изменение пользователя должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели editUser:Client;

2) компонент серверной части, имя класса в модели User:Server;

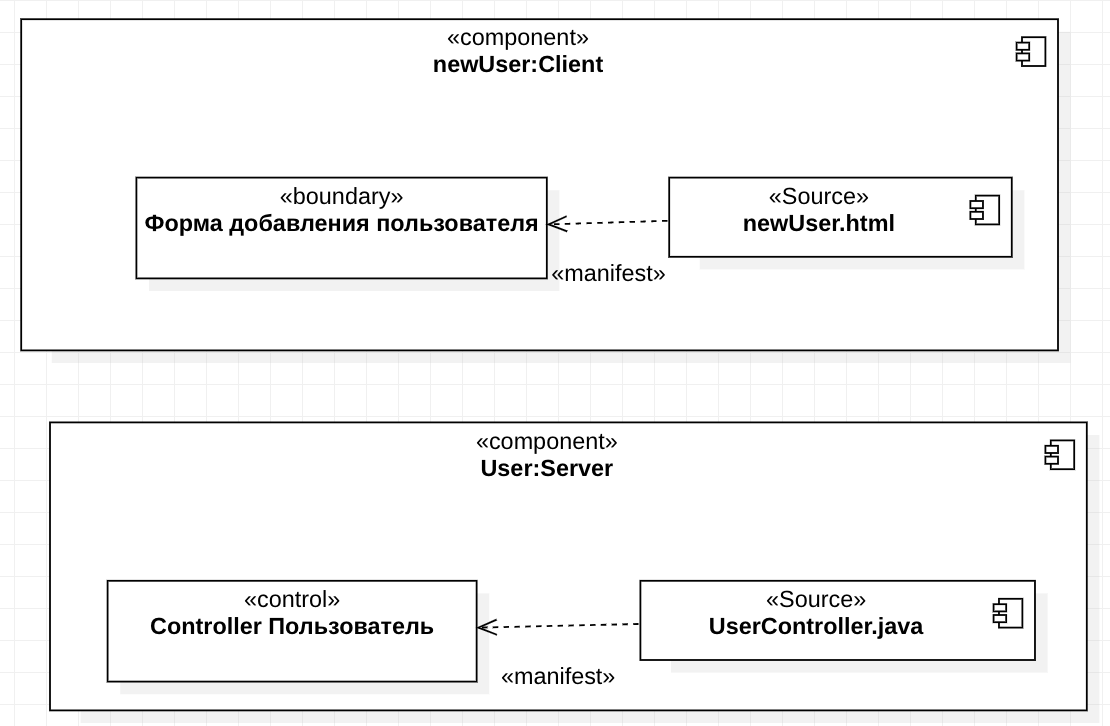
– функциональность компонента editUser:Client должна быть реализована в виде класса с именем «Изменение пользователя»;

– исходный программный код формы Удаление отзыва должен быть размещён в форме с именем editUser.html;

– функциональность компонента User:Server должна быть реализована в виде класса с именем «Controller Пользователи;

– исходный программный код класса «Controller Пользователи должен быть размещён в файле с именем UserController.java;

На рисунке 2.24 приведена диаграмма компонентов исходного кода для прецедента «Добавление пользователя».



1. 2.24 – Модель исходного кода для реализации прецедента «Добавление пользователя»

Прецедент Добавление пользователя должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели deleteReview:Client;

2) компонент серверной части, имя класса в модели Review:Server;

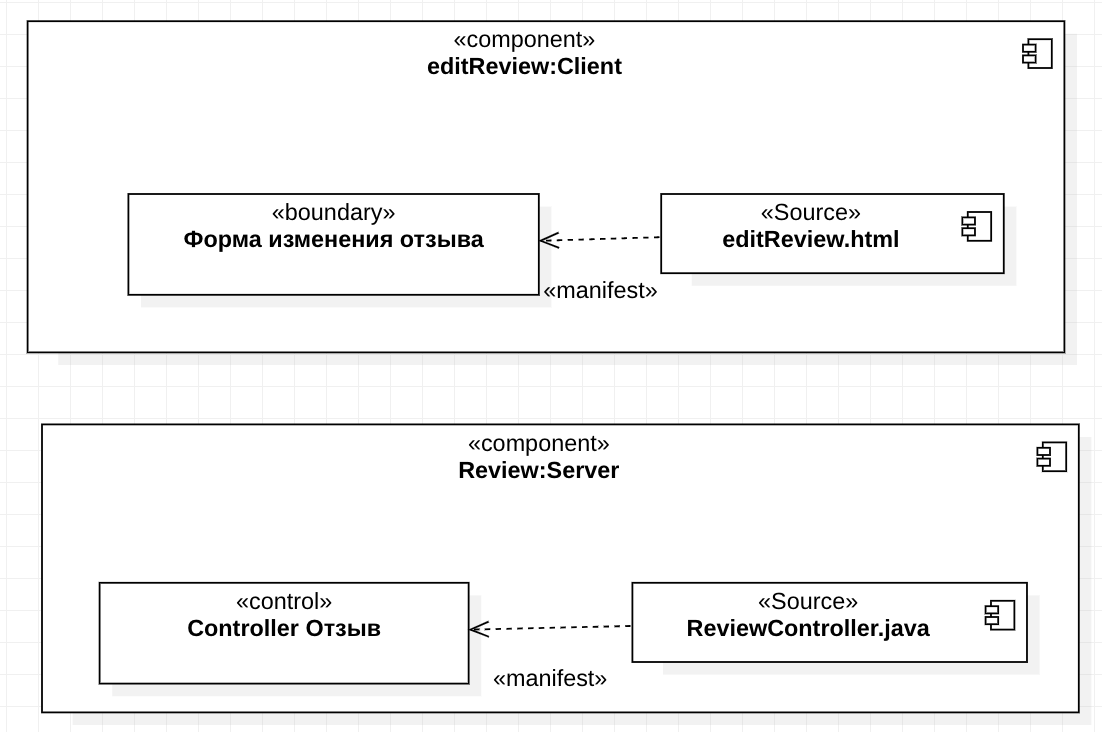
– функциональность компонента newUser:Client должна быть реализована в виде класса с именем «Добавление пользователя»;

– исходный программный код формы Добавление пользователя должен быть размещён в форме с именем newUser.html;

– функциональность компонента User:Server должна быть реализована в виде класса с именем «Controller Мероприятия»;

– исходный программный код класса «Controller Пользователи должен быть размещён в файле с именем UserController.java;

На рисунке 2.25 приведена диаграмма компонентов исходного кода для прецедента Изменение отзыва».



1. 2.25 – Модель исходного кода для реализации прецедента Изменение отзыва»

Прецедент Изменение отзыва должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели editReview:Client;

2) компонент серверной части, имя класса в модели Review:Server;

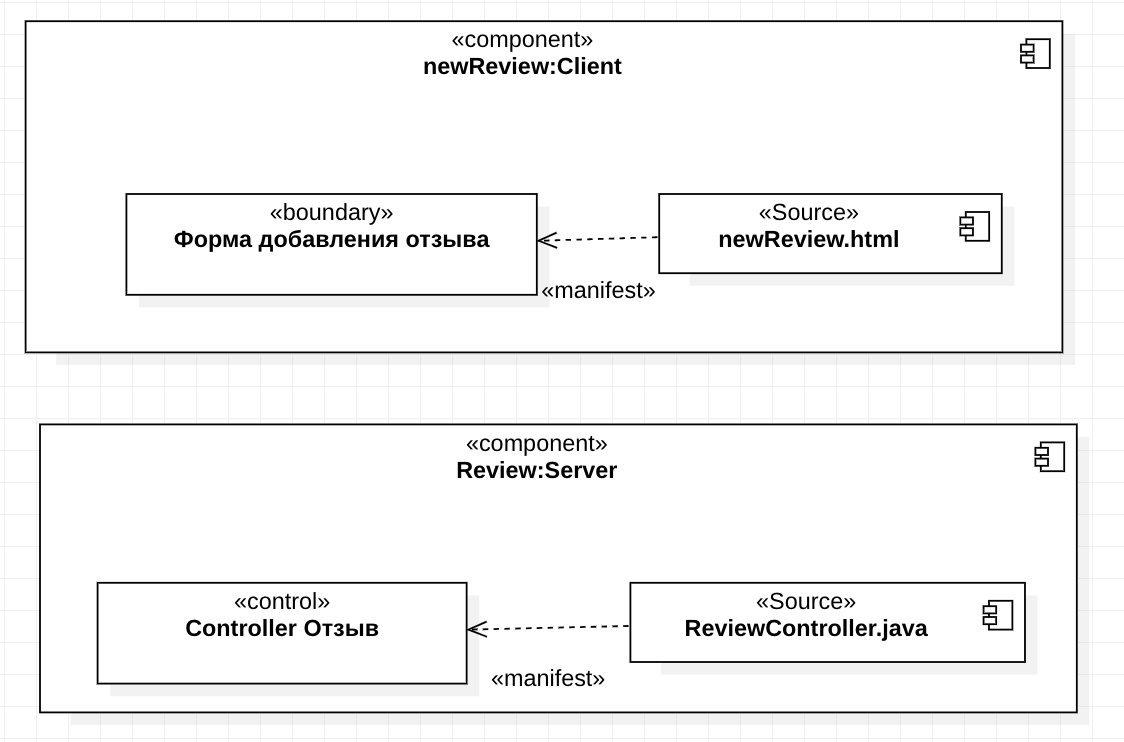
– функциональность компонента editReview:Client должна быть реализована в виде класса с именем Изменение отзыва;

– исходный программный код формы Удаление отзыва должен быть размещён в форме с именем editReview.html;

– функциональность компонента Review:Server должна быть реализована в виде класса с именем «Controller Отзывы;

– исходный программный код класса «Controller Отзывы должен быть размещён в файле с именем ReviewController.java;

На рисунке 2.26 приведена диаграмма компонентов исходного кода для прецедента «Удаление отзыва».



1. 2.26 – Модель исходного кода для реализации прецедента «Добавление отзыва»

Прецедент Добавление отзыва должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели newReview:Client;

2) компонент серверной части, имя класса в модели Review:Server;

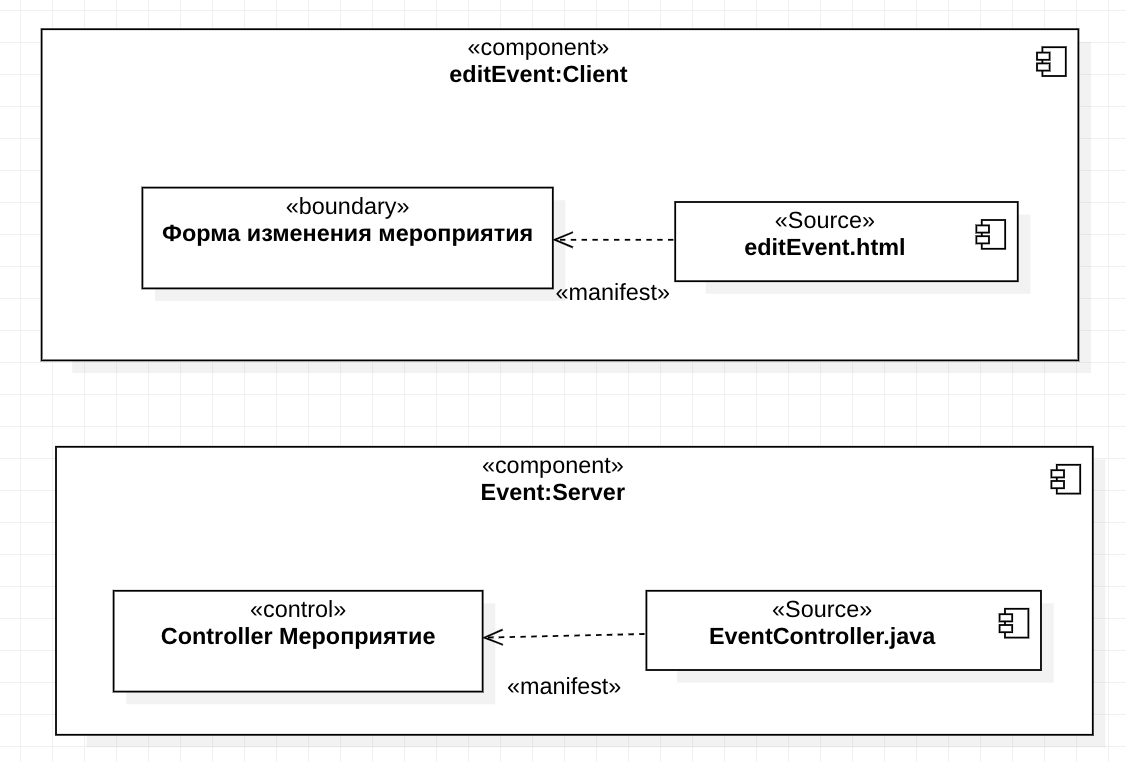
– функциональность компонента newReview:Client должна быть реализована в виде класса с именем Добавление мероприятия»;

– исходный программный код формы Добавление отзыва должен быть размещён в форме с именем newReview.html;

– функциональность компонента Event:Server должна быть реализована в виде класса с именем «Controller Отзывы;

– исходный программный код класса «Controller Отзывы должен быть размещён в файле с именем ReviewController.java;

На рисунке 2.27 приведена диаграмма компонентов исходного кода для прецедента «Изменение мероприятия».



1. 2.27 – Модель исходного кода для реализации прецедента «Изменение мероприятия»

Прецедент Изменение мероприятия должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели editEvent:Client;

2) компонент серверной части, имя класса в модели Review:Server;

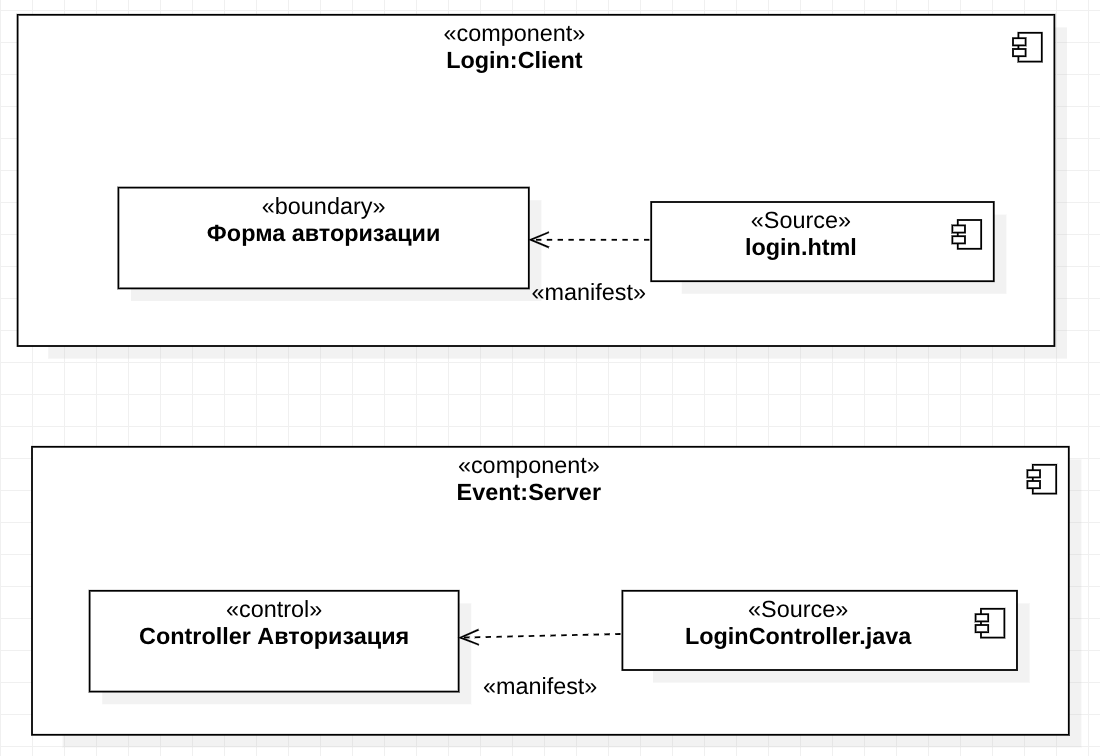
– функциональность компонента editEvent:Client должна быть реализована в виде класса с именем «Изменение мероприятия»;

– исходный программный код формы Изменение мероприяти должен быть размещён в форме с именем editEvent.html;

– функциональность компонента Event:Server должна быть реализована в виде класса с именем «Controller Мероприятия»;

– исходный программный код класса «Controller Мероприятия должен быть размещён в файле с именем EventController.java;

На рисунке 2.28 приведена диаграмма компонентов исходного кода для прецедента «Авторизация».



1. 2.28 – Модель исходного кода для реализации прецедента «Авторизация»

Прецедент авторизация должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели login:Client;

2) компонент серверной части, имя класса в модели Login:Server;

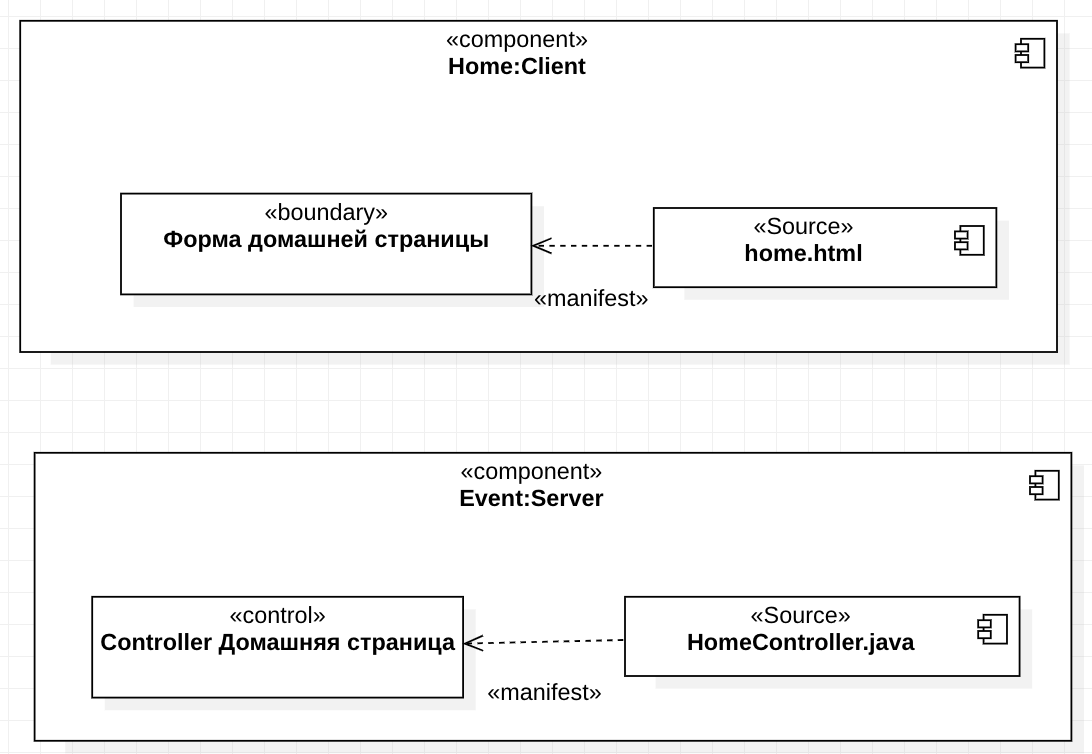
– функциональность компонента login:Client должна быть реализована в виде класса с именем «Авторизация»;

– исходный программный код формы Удаление отзыва должен быть размещён в форме с именем login.html;

– функциональность компонента Login:Server должна быть реализована в виде класса с именем «Controller Авторизация;

– исходный программный код класса «Controller Авторизация должен быть размещён в файле с именем LoginController.java;

На рисунке 2.29 приведена диаграмма компонентов исходного кода для прецедента «Домашняя страница».



1. 2.29 – Модель исходного кода для реализации прецедента «Домашняя страница»

Прецедент Удаление отзыва должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели home:Client;

2) компонент серверной части, имя класса в модели Home:Server;

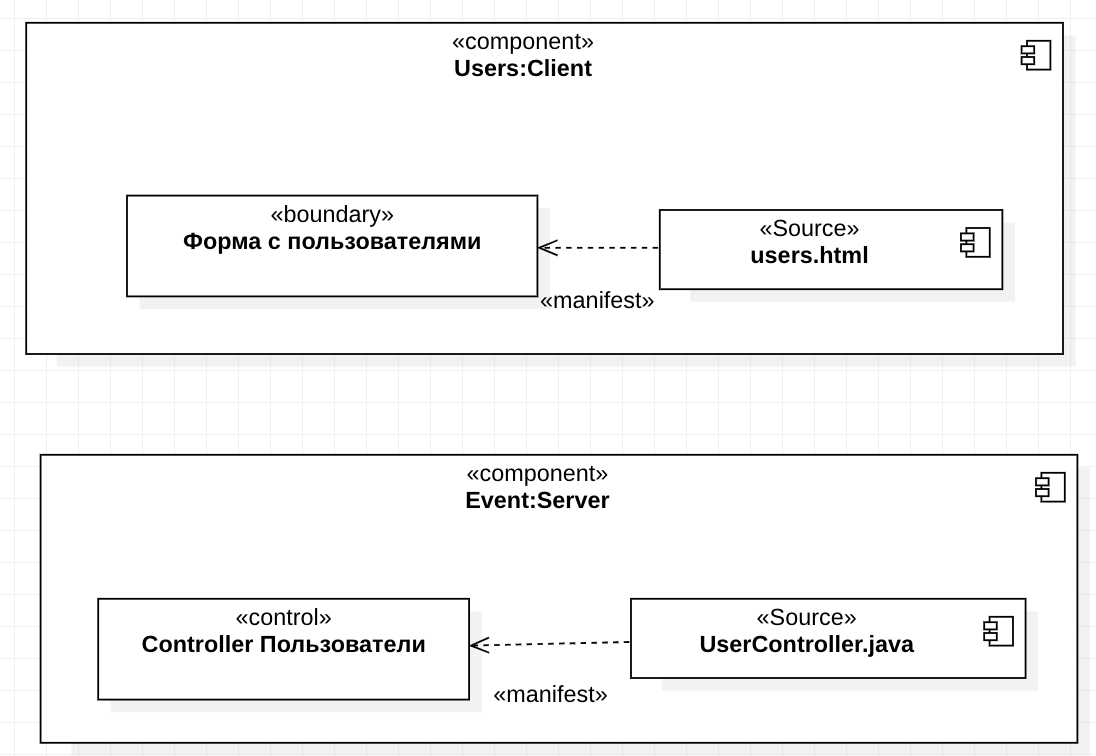
– функциональность компонента home:Client должна быть реализована в виде класса с именем «Домашняя страница»;

– исходный программный код формы Домашняя страница должен быть размещён в форме с именем home.html;

– функциональность компонента Home:Server должна быть реализована в виде класса с именем «Controller Домашняя страница;

– исходный программный код класса «Controller Домашняя страница должен быть размещён в файле с именем HomeController.java;

На рисунке 2.30 приведена диаграмма компонентов исходного кода для прецедента «Пользователи».



1. 2.30 – Модель исходного кода для реализации прецедента «Пользователи»

Прецедент Пользователи должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели users:Client;

2) компонент серверной части, имя класса в модели Review:Server;

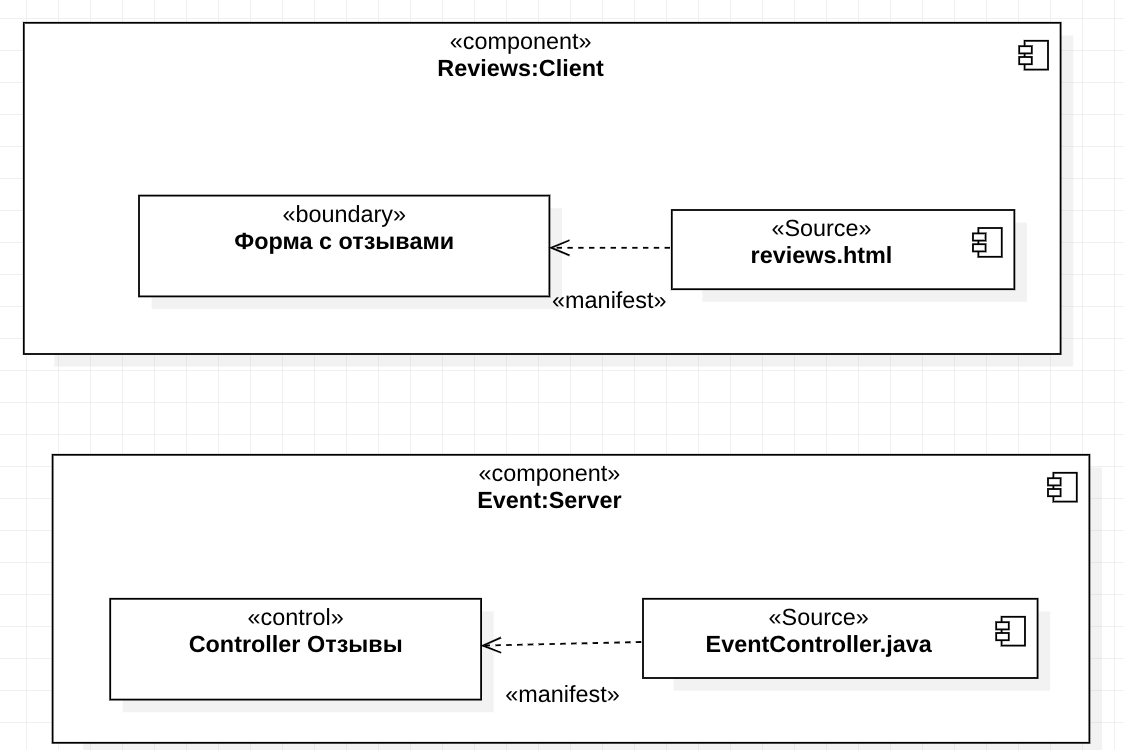
– функциональность компонента Users:Client должна быть реализована в виде класса с именем «Пользователи»;

– исходный программный код формы Пользователи должен быть размещён в форме с именем users.html;

– функциональность компонента Users:Server должна быть реализована в виде класса с именем «Controller Пользователи»;

– исходный программный код класса «Controller Пользователи должен быть размещён в файле с именем UserController.java;

На рисунке 2.31 приведена диаграмма компонентов исходного кода для прецедента «Отзывы».



1. 2.31 – Модель исходного кода для реализации прецедента «Отзывы»

Прецедент Отзывы должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели reviews:Client;

2) компонент серверной части, имя класса в модели Reviews:Server;

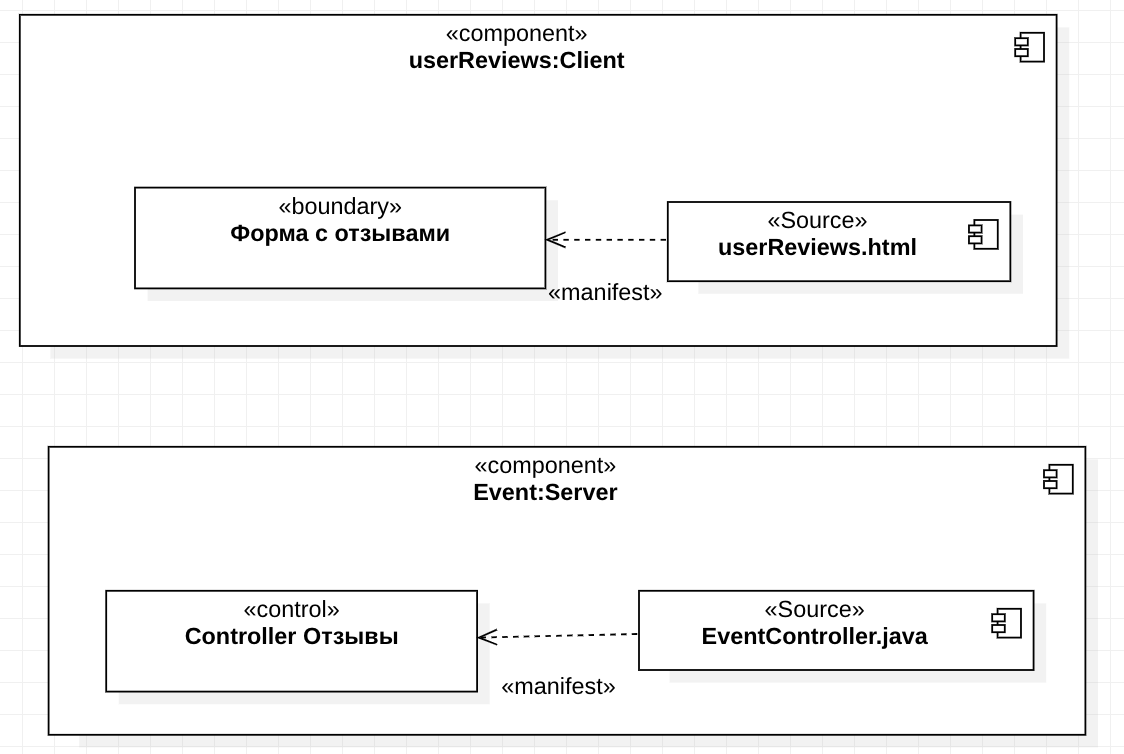
– функциональность компонента reviews:Client должна быть реализована в виде класса с именем «Отзывы»;

– исходный программный код формы Отзывы должен быть размещён в форме с именем reviews.html;

– функциональность компонента Review:Server должна быть реализована в виде класса с именем «Controller Отзывы»;

– исходный программный код класса «Controller Отзывы должен быть размещён в файле с именем ReviewController.java;

На рисунке 2.32 приведена диаграмма компонентов исходного кода для прецедента «Отзывы пользователя».



1. 2.32 – Модель исходного кода для реализации прецедента «Отзывы пользователя»

Прецедент Отзывы должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели reviews:Client;

2) компонент серверной части, имя класса в модели Reviews:Server;

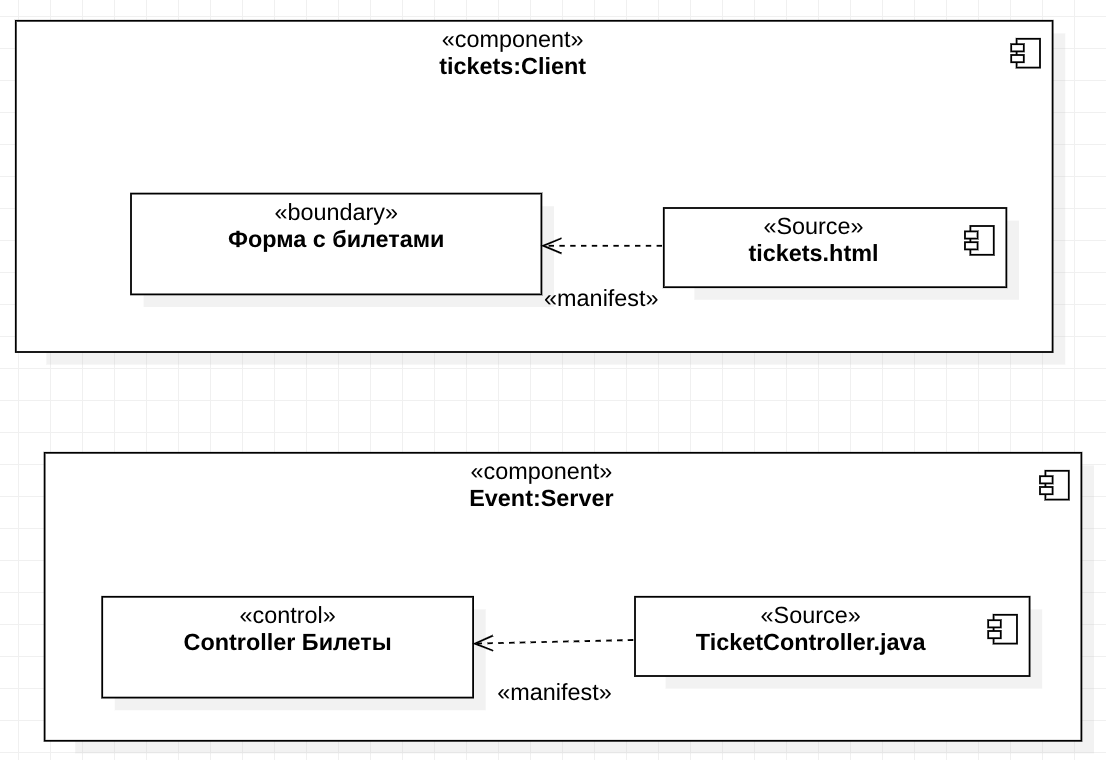
– функциональность компонента reviews:Client должна быть реализована в виде класса с именем «Отзывы»;

– исходный программный код формы Отзывы должен быть размещён в форме с именем reviews.html;

– функциональность компонента Review:Server должна быть реализована в виде класса с именем «Controller Отзывы»;

– исходный программный код класса «Controller Отзывы должен быть размещён в файле с именем ReviewController.java;

На рисунке 2.33 приведена диаграмма компонентов исходного кода для прецедента «Билеты».



1. 2.33 – Модель исходного кода для реализации прецедента «Билеты»

Прецедент Билеты должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели tickets:Client;

2) компонент серверной части, имя класса в модели Tickets:Server;

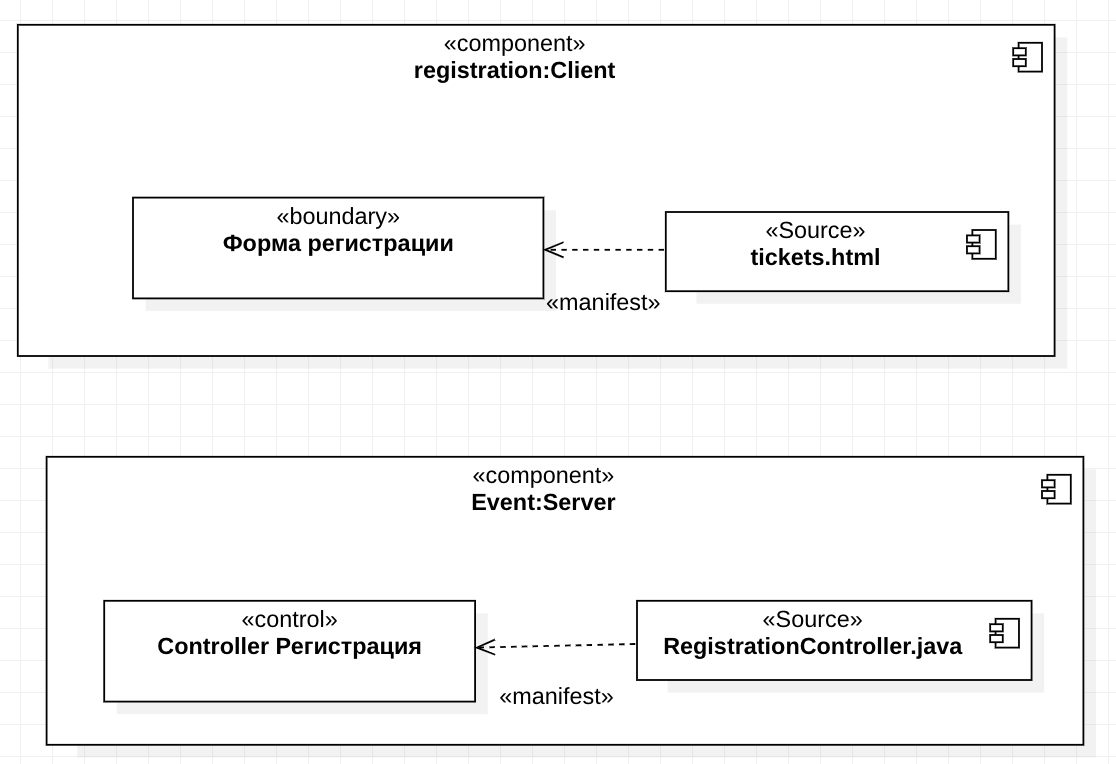
– функциональность компонента tickets:Client должна быть реализована в виде класса с именем «Удаление мероприятия»;

– исходный программный код формы Билеты должен быть размещён в форме с именем tickets.html;

– функциональность компонента Tickets:Server должна быть реализована в виде класса с именем «Controller Билеты;

– исходный программный код класса «Controller Билеты должен быть размещён в файле с именем TicketController.java;

На рисунке 2.34 приведена диаграмма компонентов исходного кода для прецедента «Регистрация».



1. 2.34 – Модель исходного кода для реализации прецедента «Регистрация»

Прецедент Регистрация должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели registration:Client;

2) компонент серверной части, имя класса в модели Registration:Server;

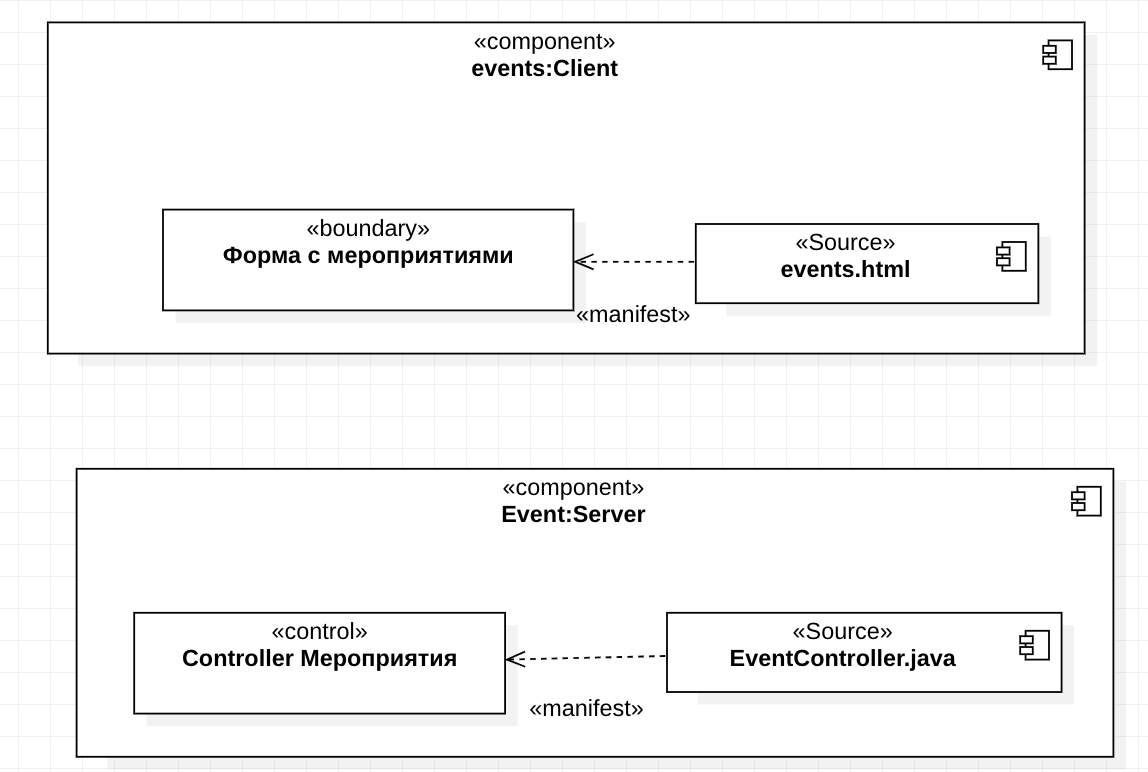
– функциональность компонента registration:Client должна быть реализована в виде класса с именем «Регистрация»;

– исходный программный код формы Регистрация должен быть размещён в форме с именем registration.html;

– функциональность компонента Event:Server должна быть реализована в виде класса с именем «Controller Регистрация;

– исходный программный код класса «Controller Регистрация должен быть размещён в файле с именем RegistrationController.java;

На рисунке 2.35 приведена диаграмма компонентов исходного кода для прецедента «Мероприятия».



1. 2.35 – Модель исходного кода для реализации прецедента «Мероприятия»

Прецедент Удаление отзыва должен быть реализован на языке Java в виде двух компонентов:

1) компонент клиентской части, имя компонента в модели events:Client;

2) компонент серверной части, имя класса в модели Event:Server;

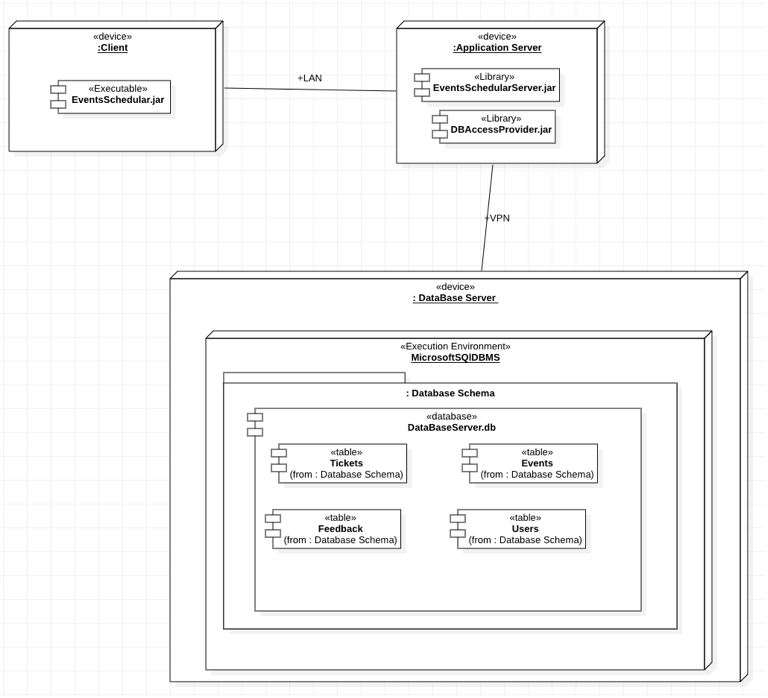
– функциональность компонента events:Client должна быть реализована в виде класса с именем «Мероприятия»;

– исходный программный код формы Удаление отзыва должен быть размещён в форме с именем events.html;

– функциональность компонента Event:Server должна быть реализована в виде класса с именем «Controller Мероприятия;

– исходный программный код класса «Controller Мероприятия должен быть размещён в файле с именем EvetnsController.java;

### 2.2.3 Модель размещения артефактов



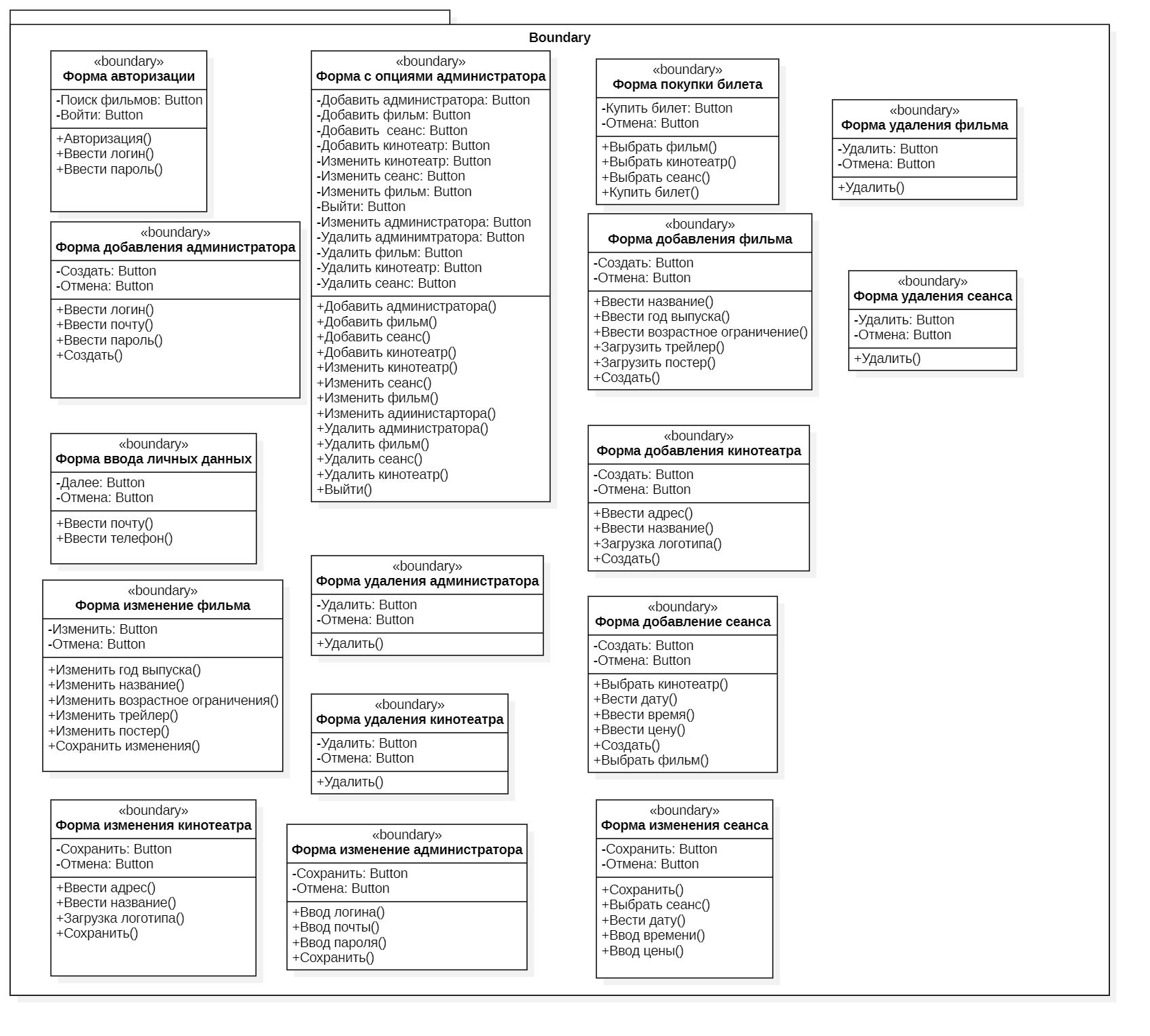
1. 2.26 – Модель размещения артефактов

Система развертывается на трех звеньях, клиентская и серверная части общаются между собой через локальную сеть (LAN), а серверная с базой данных через VPN.

# 3 Реализация ПрИС

## 3.1 Реализация бизнес-логики

Итоговая диаграмма граниничных классов представлена на рисунке ниже.



1. 3.1 – Итоговая диаграмма пакета граничных классов

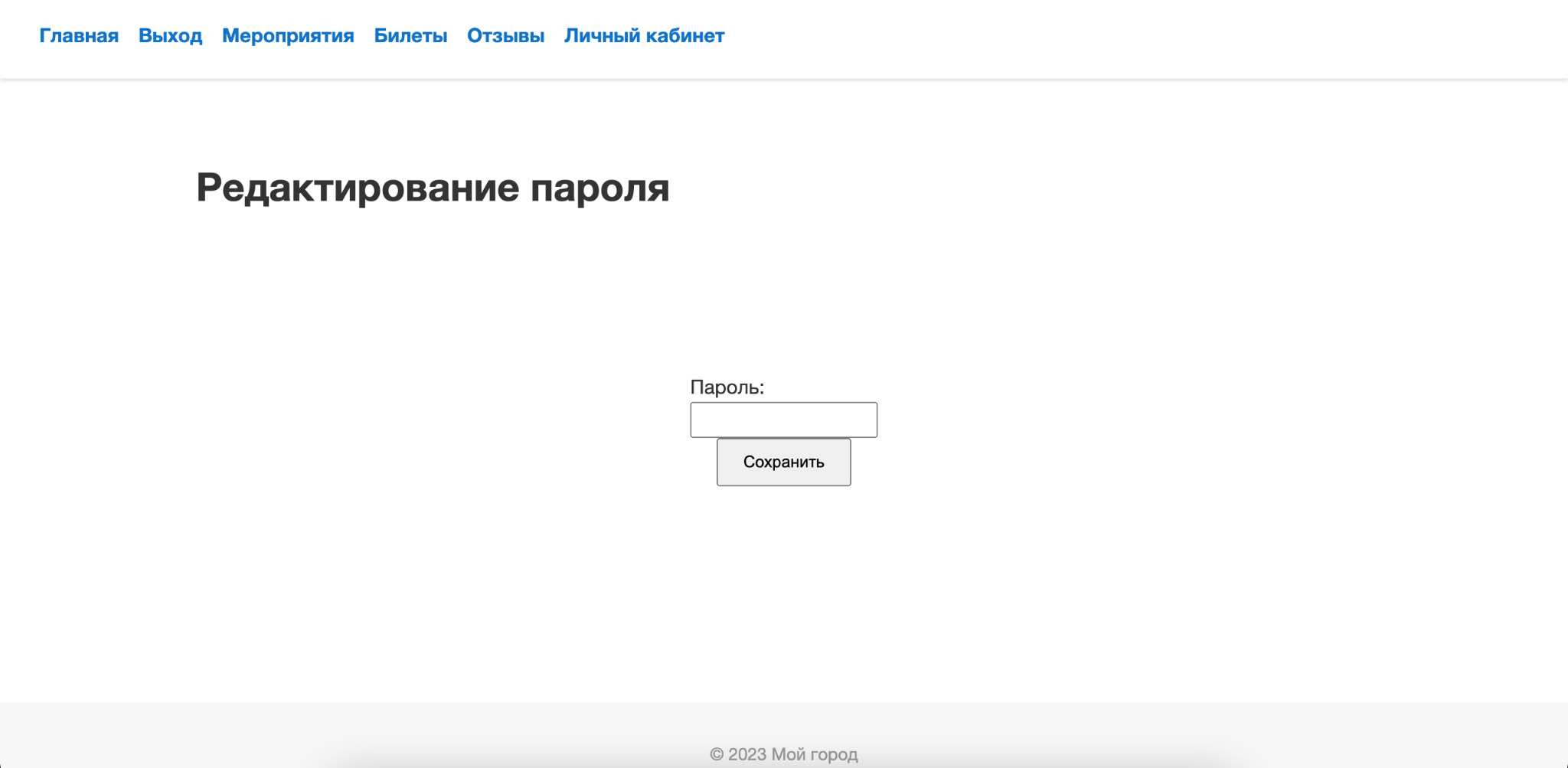
## 

## 3.2 Реализация пользовательского интерфейса

* + - 1. 3.1 – Компоненты формы изменения пароля

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| changePassword | Форма изменения пароля | Text | Изменение пароля |
| labelPassword | Подпись у пароля | Text | Пароль |
| inputPassword | Поле ввода пароля | Text |  |
| labelError | Вывод сообщения о несоответствии пары логин\пароль | Text | Неправильный логин или пароль |
|

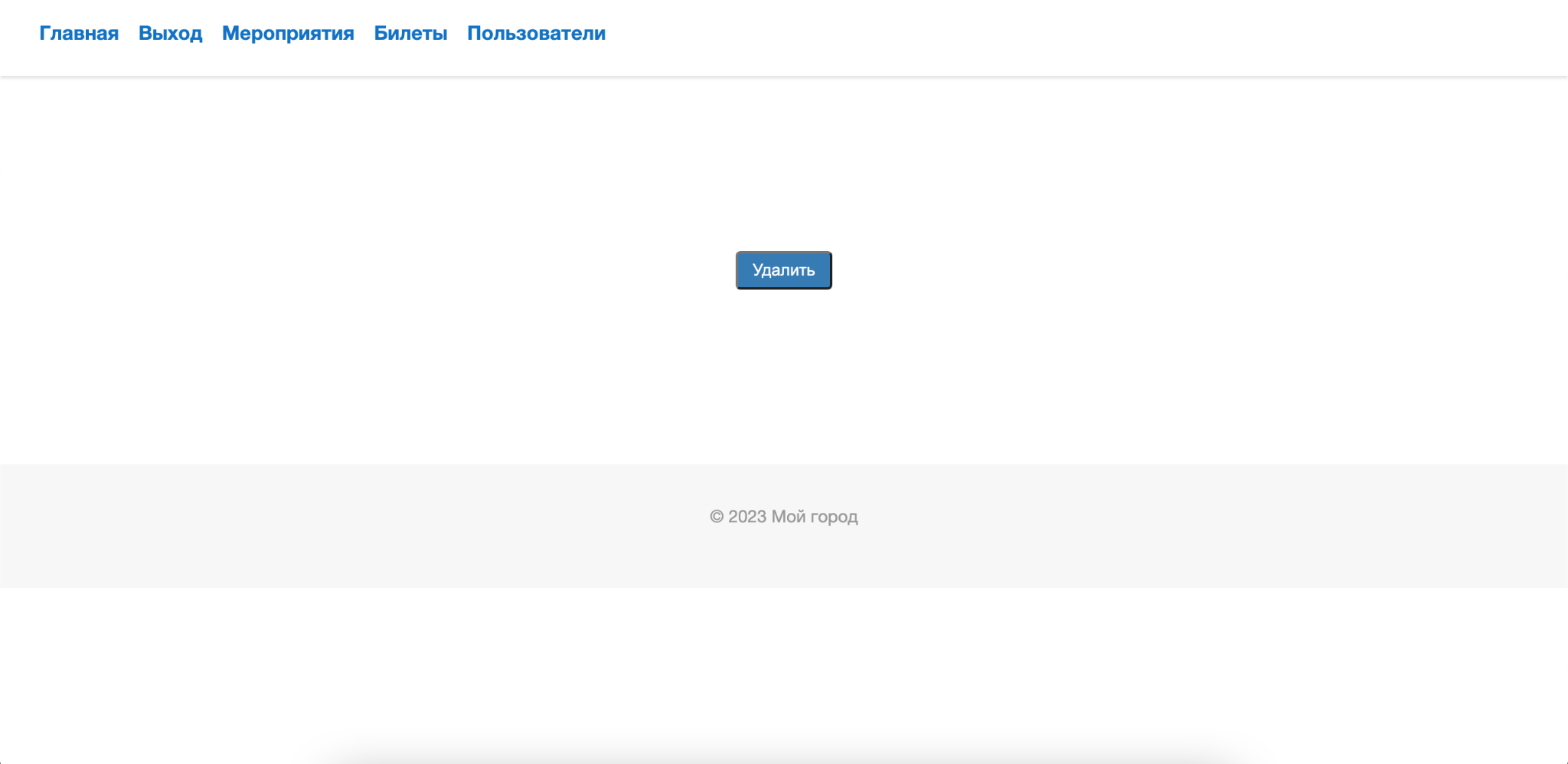
В таблице 3.1 представлен список компонентов формы изменения пароля. Реализация этой формы представлена на рисунке ниже:



1. 3.2 – Изменения пароля
   * + 1. 3.2 – Компоненты формы удаления мероприятия

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| deleteEvent | Форма удаления мероприятия | Text | Удаление мероприятия |
| deleteEventButton | Кнопка удаления мероприятия | Text | Удалить |

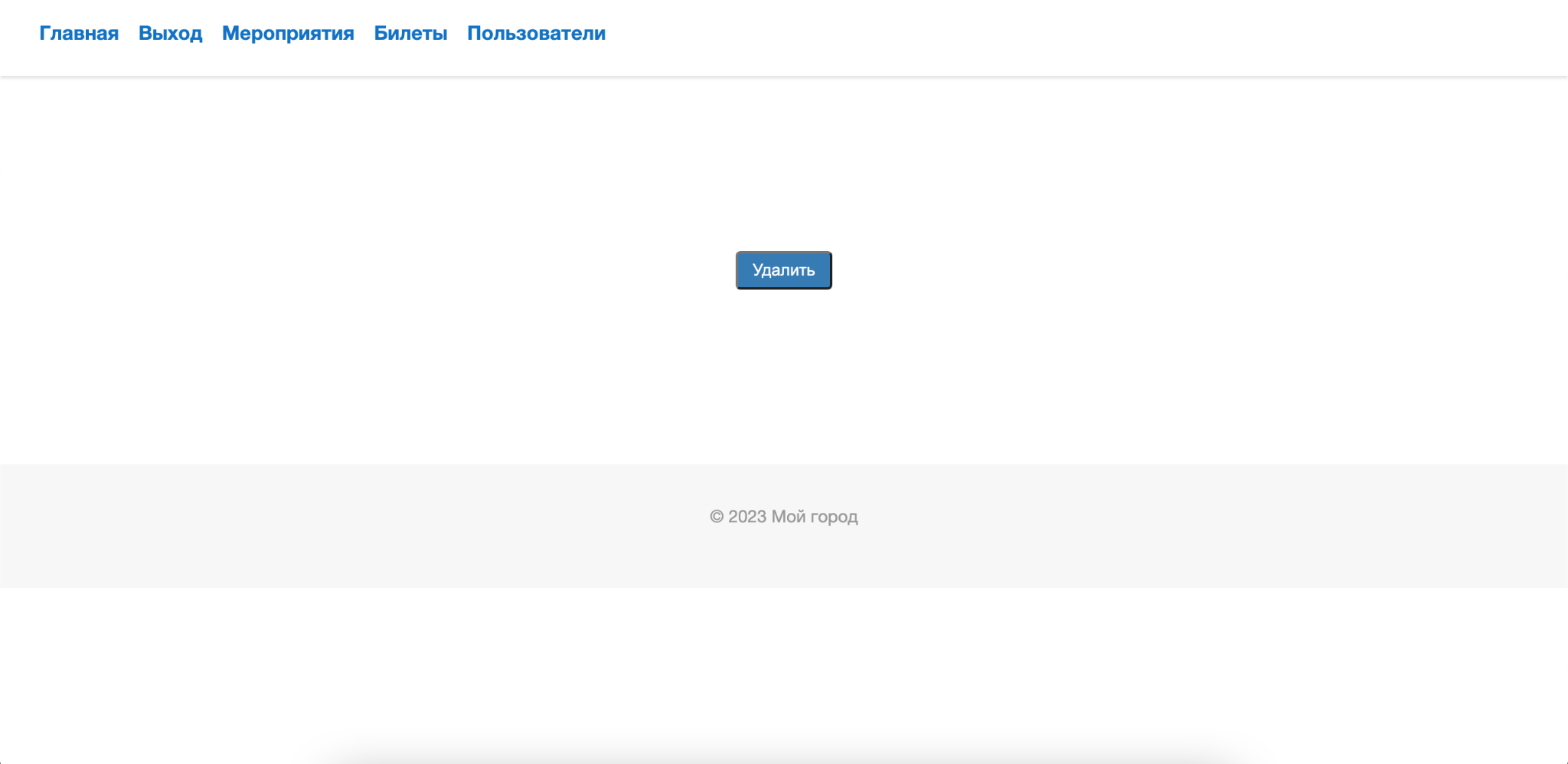
В таблице 3.2 представлен список компонентов формы удаления мероприятия. Реализация этой формы представлена на рисунке ниже:



1. 3.3 – Форма удаления мероприятия
   * + 1. 3.3 – Компоненты формы удаления отзыва

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| deleteReview | Форма удаления мероприятия | Text | Удаление отзыва |
| deleteReviewButton | Кнопка удаления мероприятия | Text | Удалить |

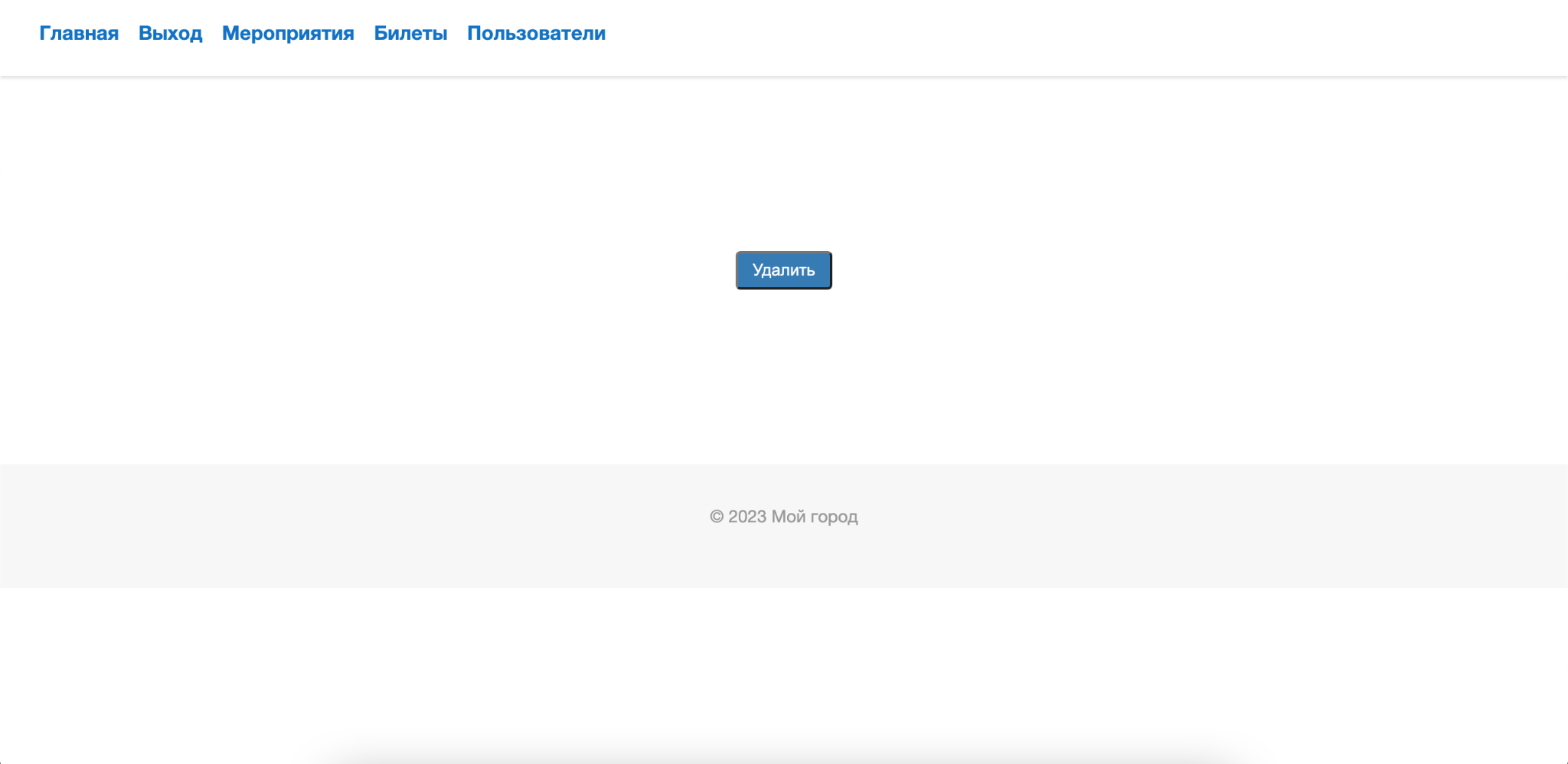
В таблице 3.3 представлен список компонентов формы удаления отзыва. Реализация этой формы представлена на рисунке ниже:



1. 3.4 – Форма удаления отзыва
   * + 1. 3.4 – Компоненты формы удаления пользователя

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| deleteUser | Форма удаления мероприятия | Text | Удаление мероприятия |
| deleteUserButton | Кнопка удаления мероприятия | Text | Удалить |

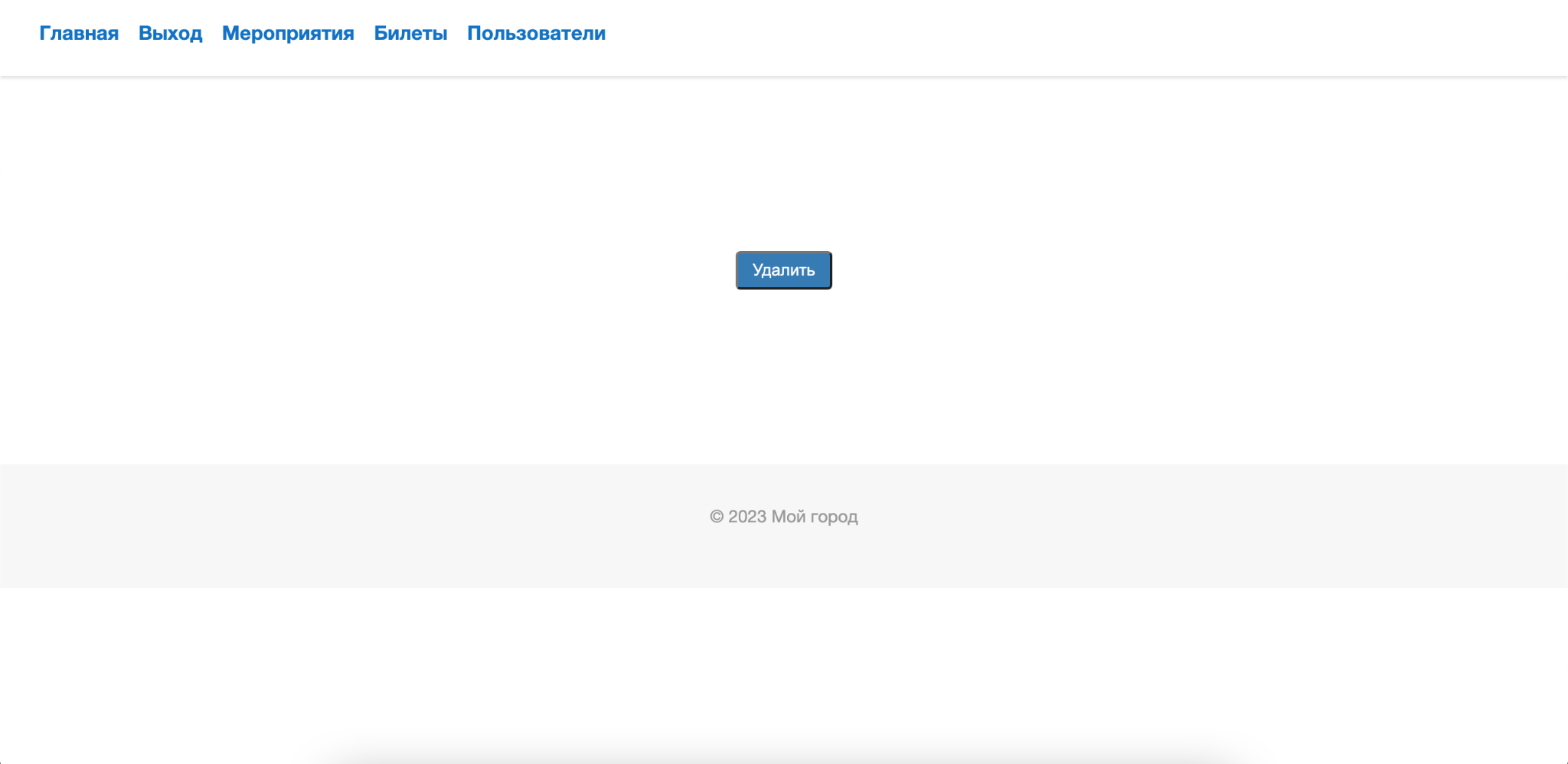
В таблице 3.4 представлен список компонентов формы удаления пользователя. Реализация этой формы представлена на рисунке ниже:



1. 3.5 – Форма удаления пользователя
2. 3.5 – Компоненты формы добавления пользователя

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| deleteUser | Форма удаления мероприятия | Text | Удаление мероприятия |
| deleteUserButton | Кнопка удаления мероприятия | Text | Удалить |

В таблице 3.5 представлен список компонентов формы добавления пользователя. Реализация этой формы представлена на рисунке ниже:



1. 3.6 – Форма добавления пользователя
2. 3.6 – Компоненты формы добавления мероприятия

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| addEvent | Форма удаления мероприятия | Text | Удаление мероприятия |
| nameLabel | Подпись у названия | Текст | Подпись |
| descriptionLabel | Подпись у описания | Текст | Подпись |
| dateLabel | Подпись у даты | Текст | Подпись |
| nameInput | Поле ввода названия | Текст | Ввод названия |
| descriptionInput | Поле ввода описания | Текст | Ввод описания |
| dateInput | Поле ввода даты | Текст | Ввод даты |
| addEvent | Кнопка добавления мероприятия | Text | Добавить мероприятие |

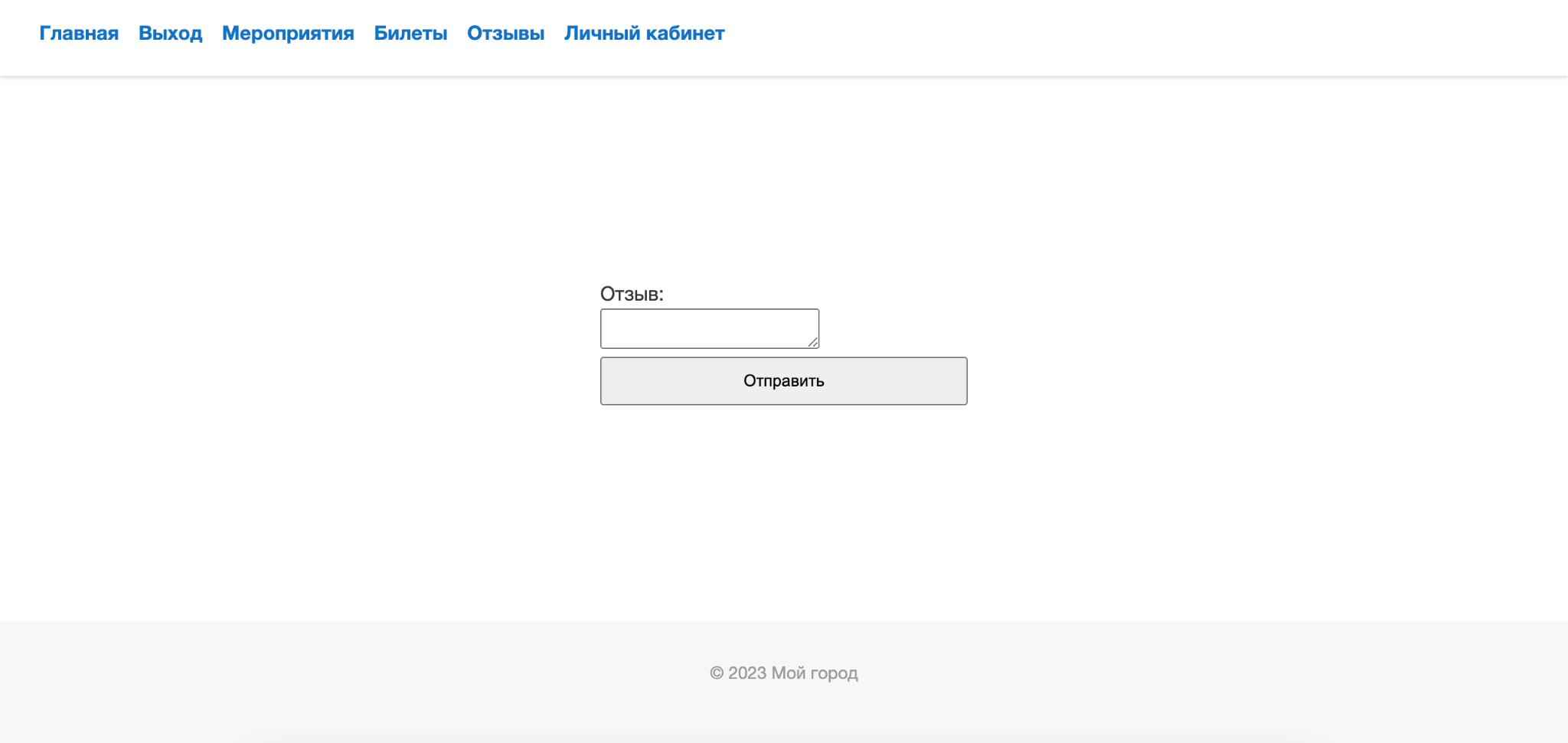
В таблице 3.6 представлен список компонентов формы добавления мероприятия. Реализация этой формы представлена на рисунке ниже:



1. 3.7 – Форма добавления мероприятия
2. 3.7 – Компоненты формы добавления отзыва

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| addReview | Форма удаления отзыва | Text | Добавление отзыва |
| textLabel | Подпись у текста | Текст | Подпись |
| textInput | Поле ввода текста | Текст | Ввод текста |
| addReview | Кнопка добавления отзыва | Text | Добавить отзыв |

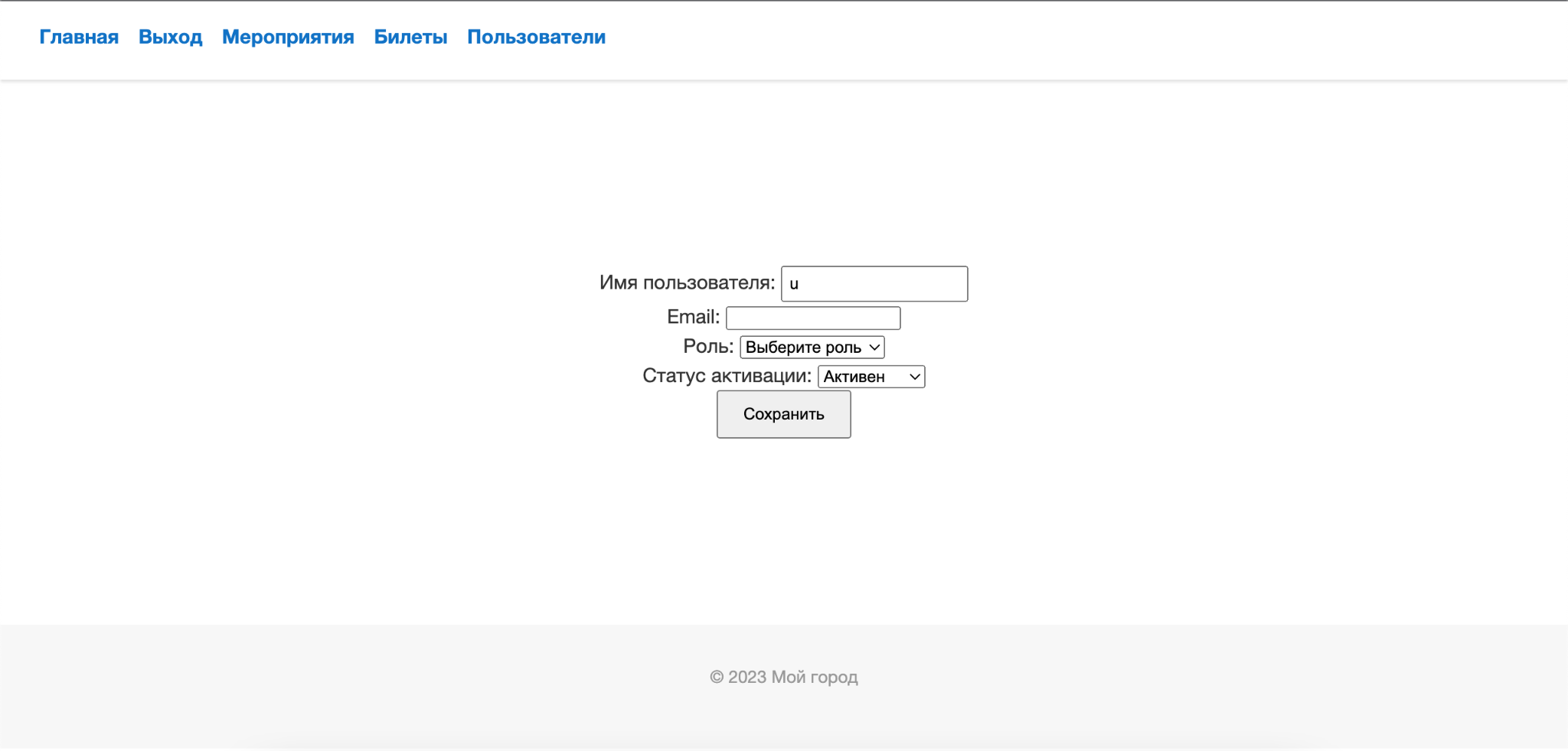
В таблице 3.7 представлен список компонентов формы добавления отзыва. Реализация этой формы представлена на рисунке ниже:



1. 3.8 – Форма добавления отзыва
2. 3.8 – Компоненты формы изменения пользователя

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| editUser | Форма удаления мероприятия | Text | Удаление мероприятия |
| usernameLabel | Подпись у имени пользователя | Текст | Подпись |
| emailLabel | Подпись у почты | Текст | Подпись |
| roleLabel | Подпись у роли |  | Подпись |
| activeLabel | Подпись у активации | Текст | Подпись |
| activeInput | Поле активации |  | Ввод |
| usernameInput | Поле логина | Текст | Ввод имени пользователя |
| emailInput | Поле ввода gjxns | Текст | Ввод почты |
| roleInput | Поле ввода роли | Текст | Ввод роли |
| editUserButton | Кнопка Изменения пользователя | Text | Изменить пользователя |

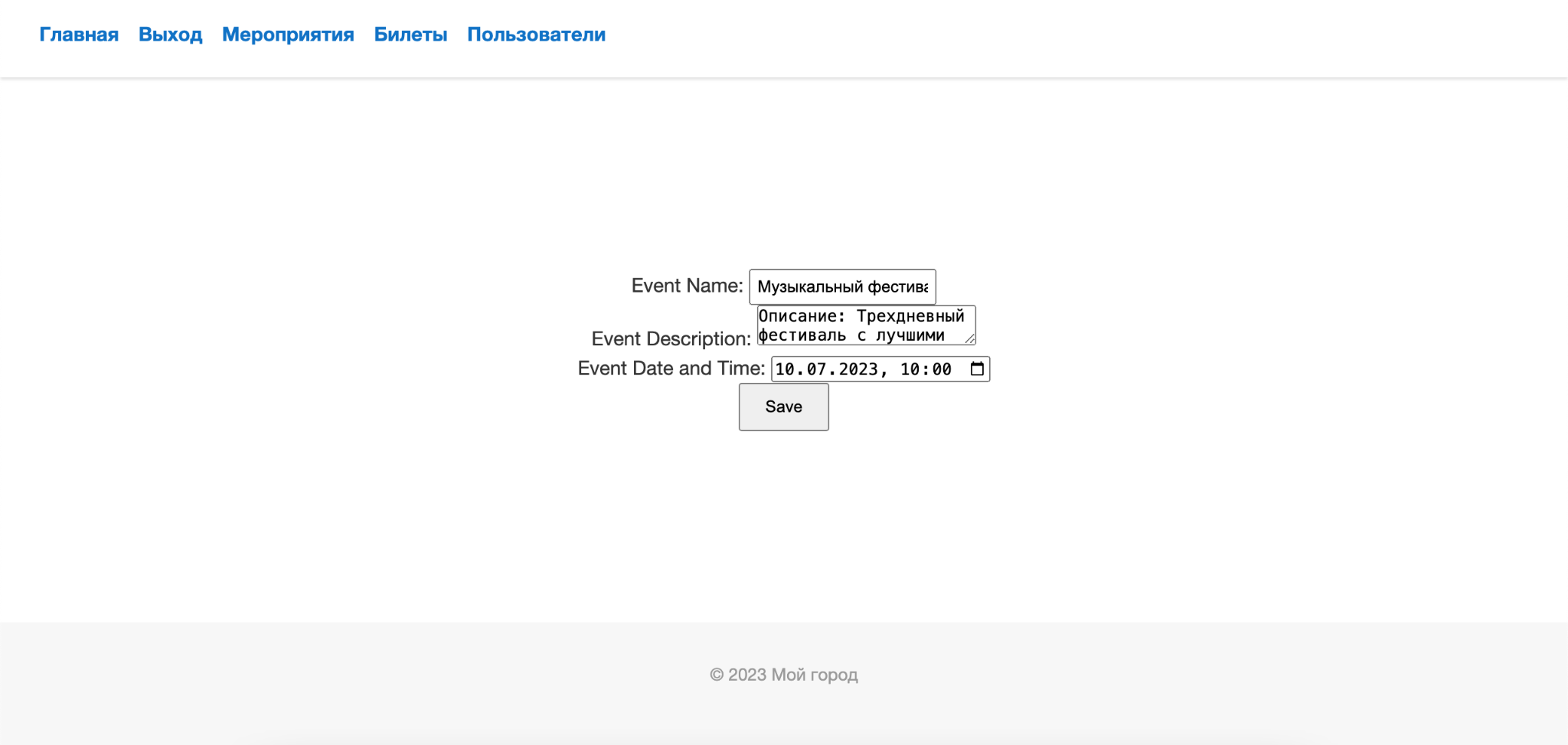
В таблице 3.8 представлен список компонентов формы изменения пользователя. Реализация этой формы представлена на рисунке ниже:



1. 3.9 – Форма изменения пользователя
2. 3.9 – Компоненты формы изменения мероприятия

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| editEvent | Форма удаления мероприятия | Text | Изменение мероприятия |
| nameLabel | Подпись у названия | Текст | Подпись |
| descriptionLabel | Подпись у описания | Текст | Подпись |
| dateLabel | Подпись у даты | Текст | Подпись |
| nameInput | Поле ввода названия | Текст | Ввод названия |
| descriptionInput | Поле ввода описания | Текст | Ввод описания |
| dateInput | Поле ввода даты | Текст | Ввод даты |
| editEvent | Кнопка добавления мероприятия | Text | Изменить мероприятие |

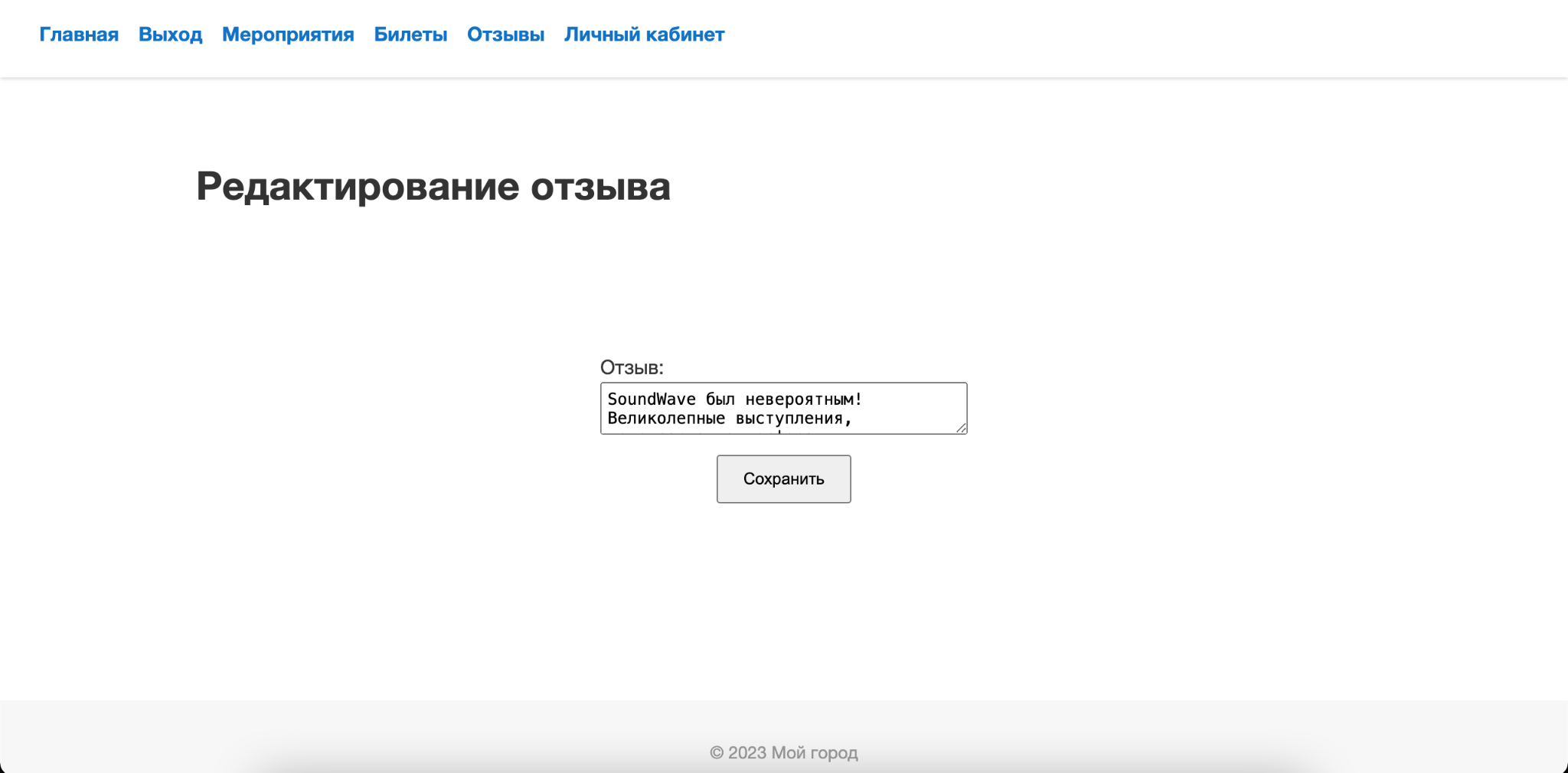
В таблице 3.9 представлен список компонентов формы изменения мероприятия. Реализация этой формы представлена на рисунке ниже:



1. 3.10 – Форма изменения мероприятия
2. 3.10 – Компоненты формы изменения отзыва

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| editReview | Форма удаления отзыва | Text | Изменить отзыва |
| textLabel | Подпись у текста | Текст | Подпись |
| textInput | Поле ввода текста | Текст | Ввод текста |
| editReview | Кнопка добавления отзыва | Text | Изменить отзыв |

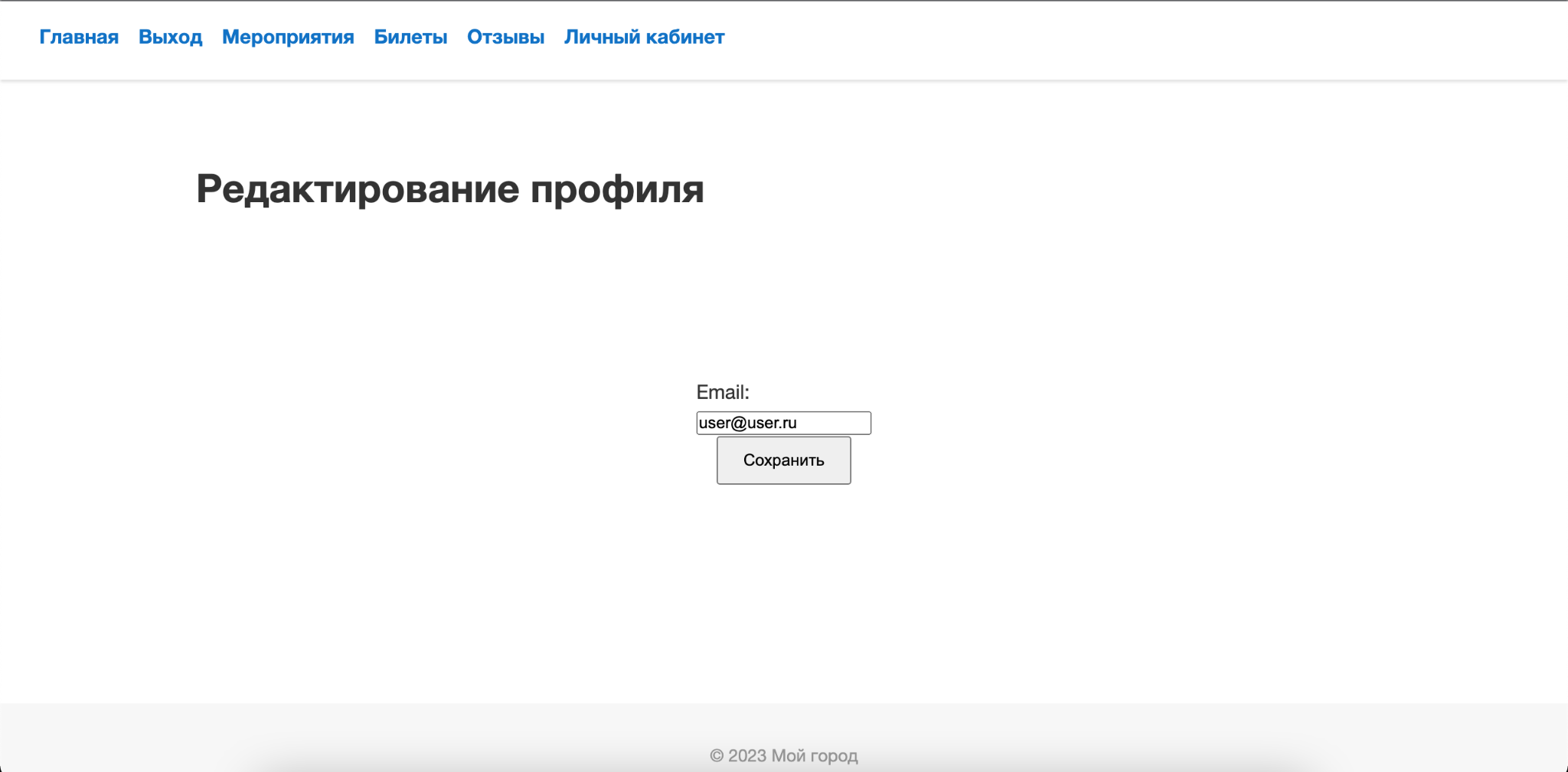
В таблице 3.10 представлен список компонентов формы изменения отзыва. Реализация этой формы представлена на рисунке ниже:



1. 3.11 – Форма изменения отзыва
2. 3.11 – Компоненты формы изменения пользователя в ЛК

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| editUser | Форма удаления отзыва | Text | Изменение пользователя |
| emailLabel | Подпись у текста | Текст | Подпись |
| enailInput | Поле ввода почты | Текст | Ввод почты |
| editUser | Кнопка добавления отзыва | Text | Изменение пользователя |

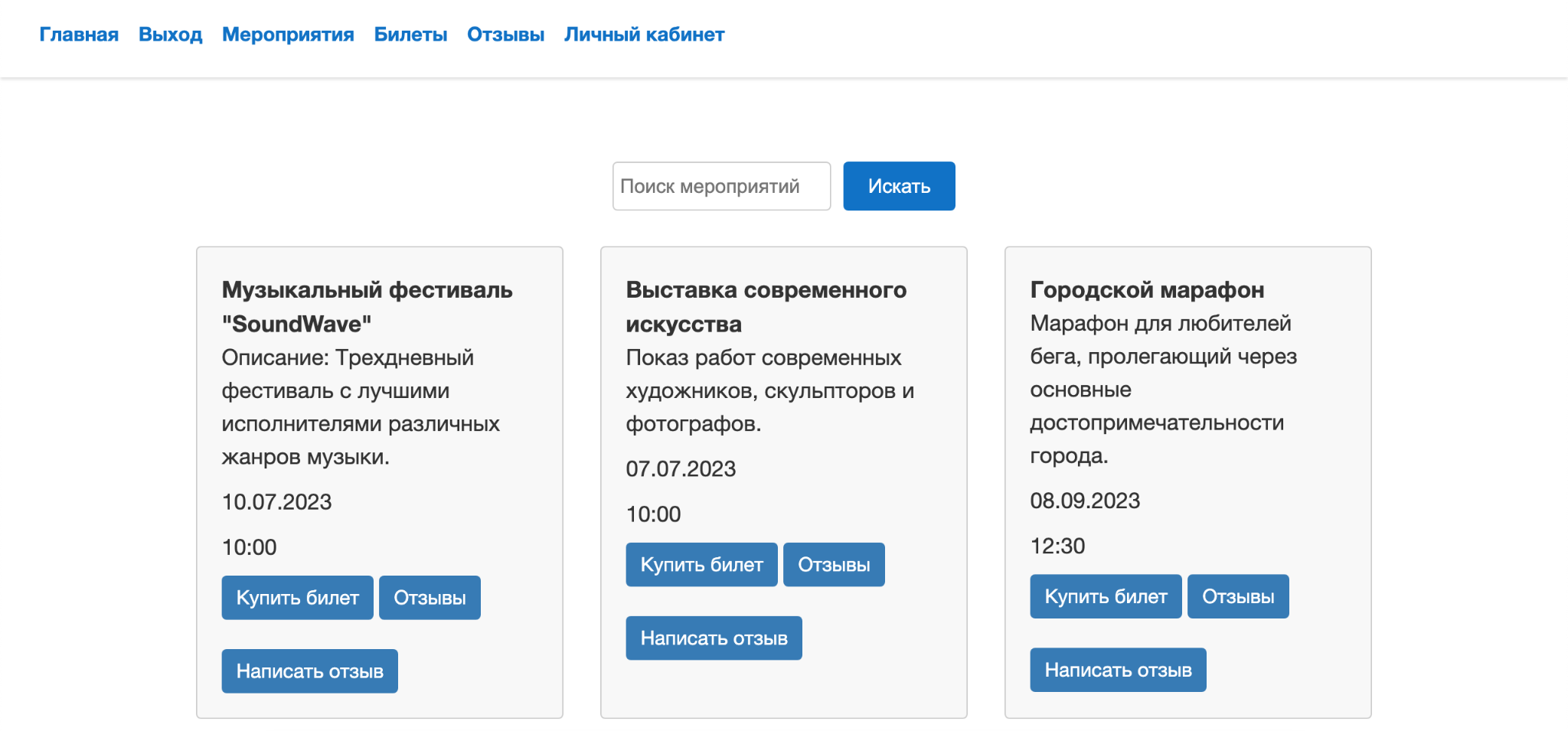
В таблице 3.11 представлен список компонентов формы изменения пользователя в ЛК. Реализация этой формы представлена на рисунке ниже:



1. 3.12 – Форма изменения пользователя в ЛК
2. 3.12 – Компоненты формы мероприятий

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| events | Форма удаления мероприятия | Text | Мероприятия |
| nameLabel | Название | Текст | Подпись |
| descriptionLabel | Описание | Текст | Подпись |
| dateLabel | Дата | Текст | Подпись |
| buyTicket | Кнопка покупки билета | Text | Купить билет |
| reviews | Кнопка отзывов | Text | Посмотреть отзыв |
| addReview | Кнопка написания отзыва | Text | Написать отзыв |

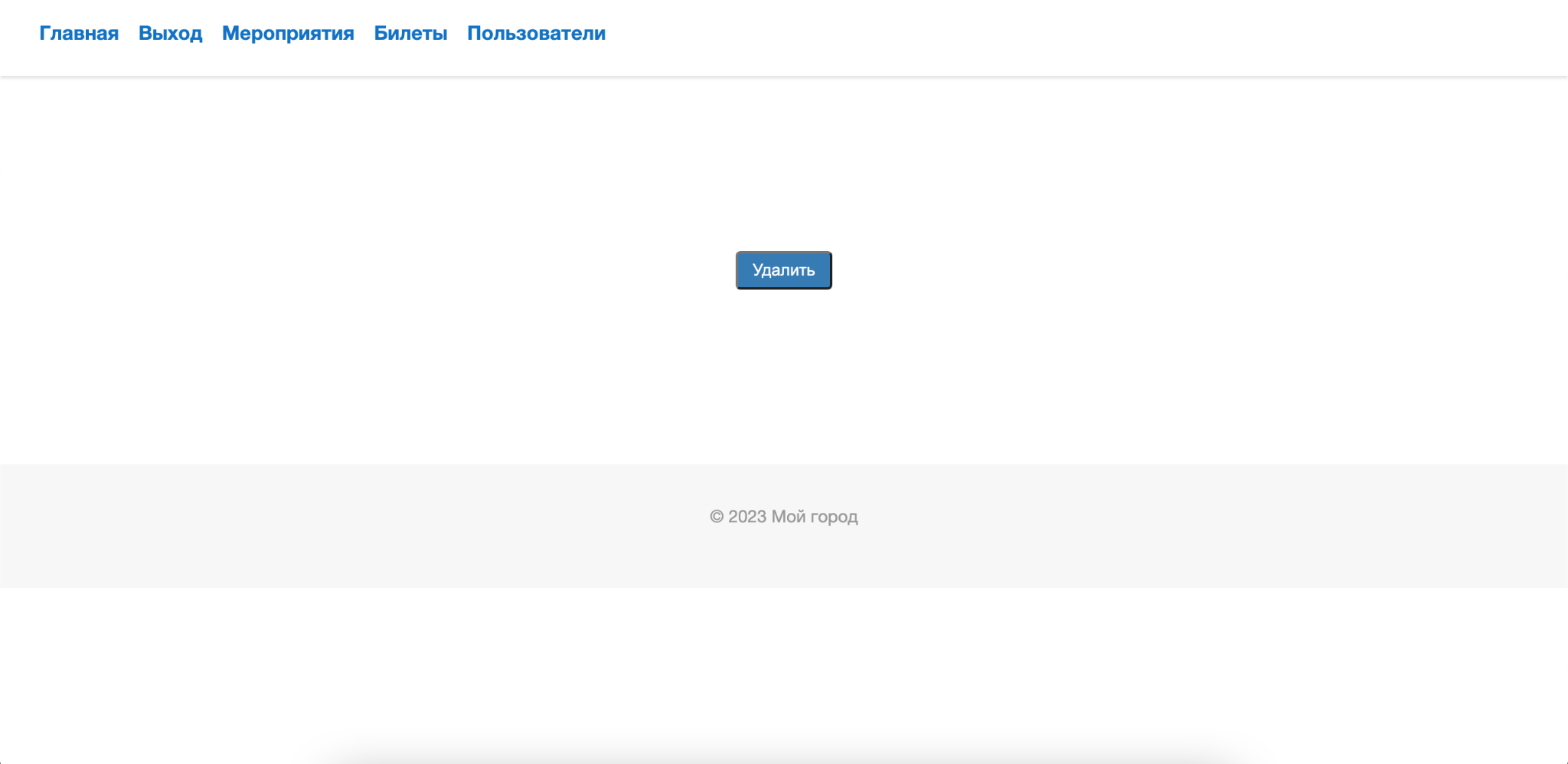
В таблице 3.12 представлен список компонентов формы мероприятий. Реализация этой формы представлена на рисунке ниже:



1. 3.13 – Форма мероприятий
2. 3.13 – Компоненты формы удаления пользователя

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| deleteUser | Форма удаления мероприятия | Text | Удаление мероприятия |
| deleteUserButton | Кнопка удаления мероприятия | Text | Удалить |

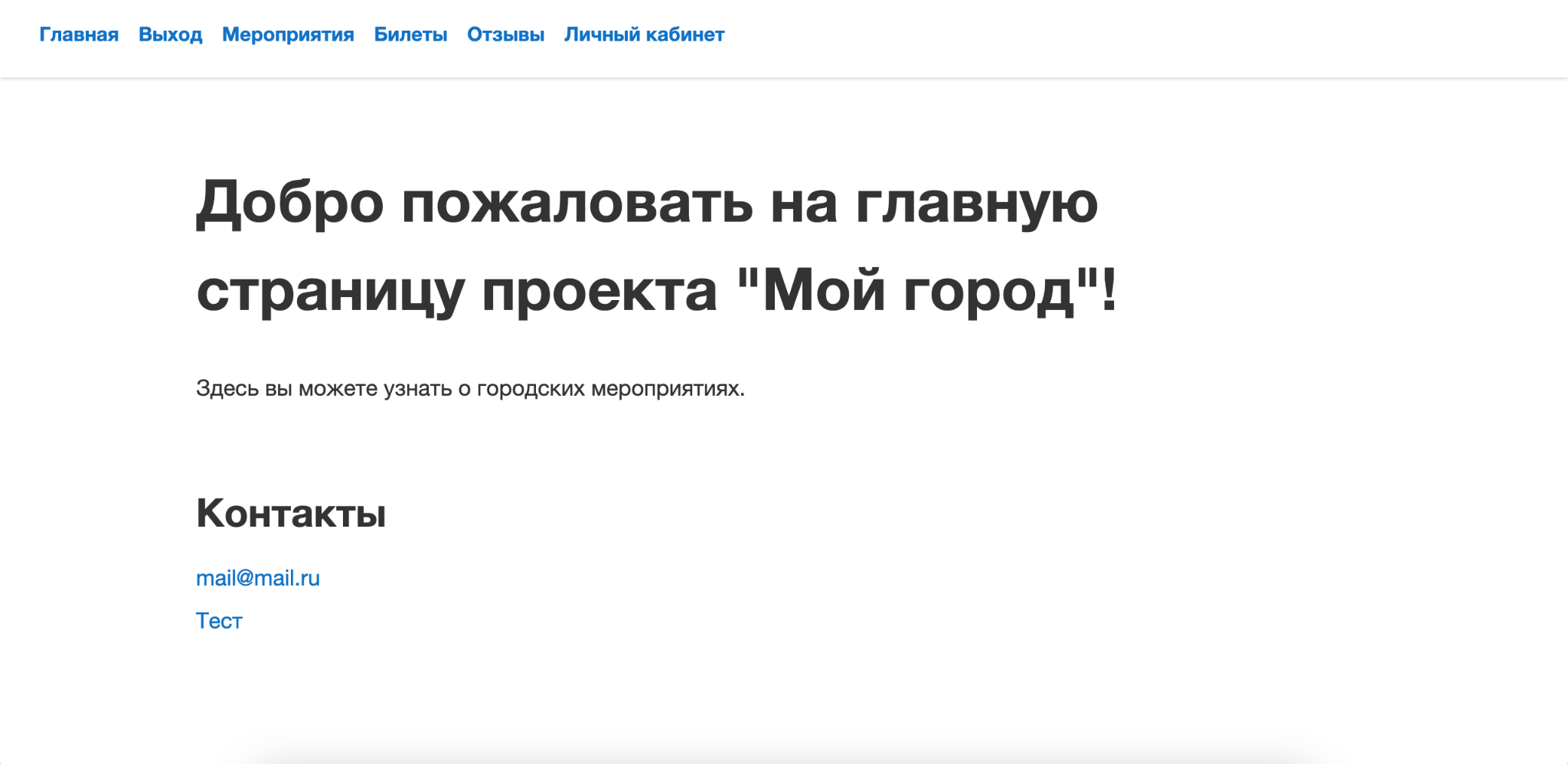
В таблице 3.13 представлен список компонентов формы удаления пользователей. Реализация этой формы представлена на рисунке ниже:



1. 3.14 – Форма удаления пользователей
2. 3.14 – Компоненты формы домашней страницы

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| label | Текст с информацией о приложении | Text | Текст с информацией о приложении |

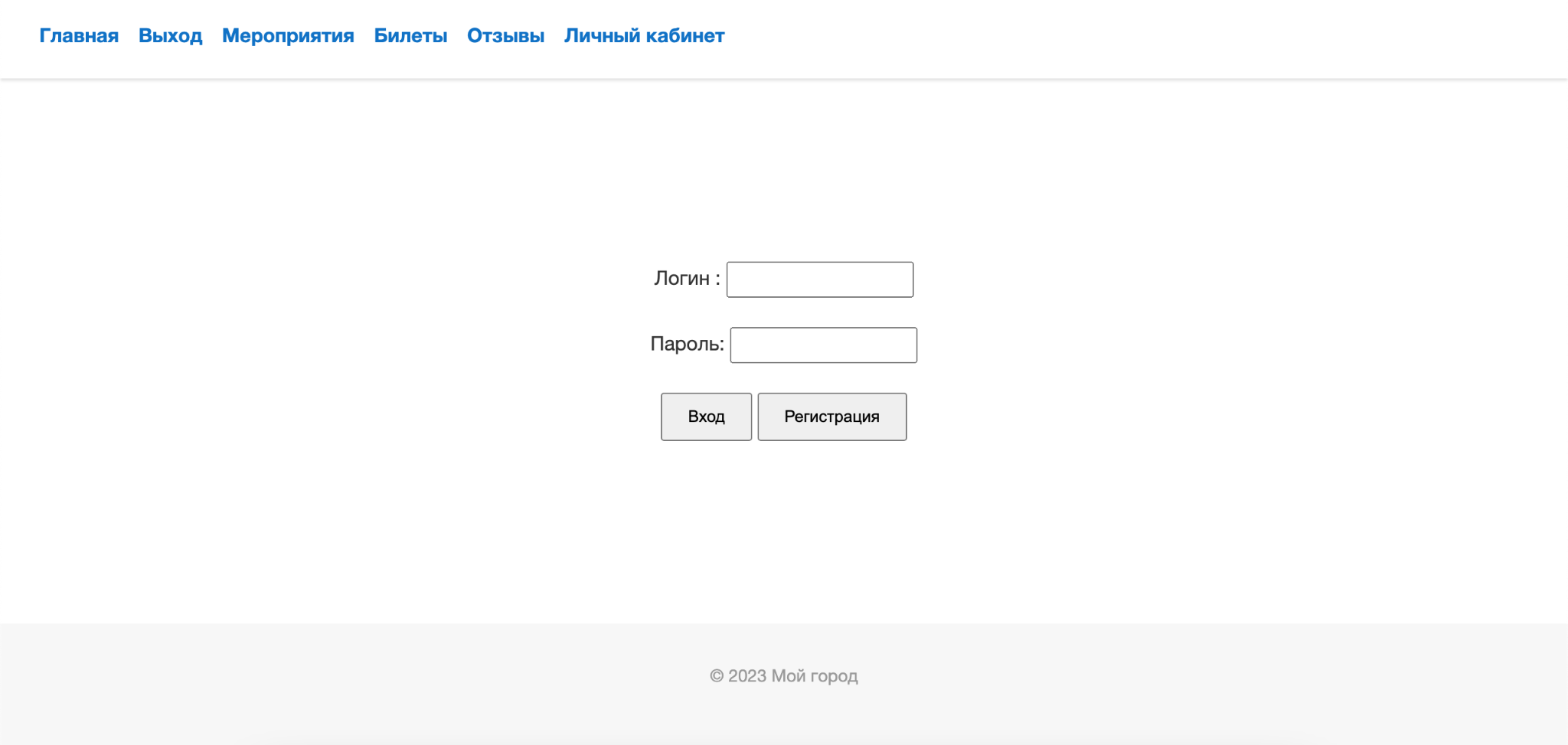
В таблице 3.14 представлен список компонентов формы домашней страницы. Реализация этой формы представлена на рисунке ниже:



1. 3.15 – Форма домашней страницы
2. 3.15 – Компоненты формы авторизации

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| login | Форма удаления мероприятия | Text | Авторизация |
| loginLabel | Подпись у логина | Text | Логин |
| passwordLabel | Подпись у пароля | Text | Пароль |
| loginInput | Поле ввода логина | Text | Логин |
| passwordInput | Поле ввода пароля | Text | Пароль |
| loginButton | Кнопка авторизации | Text | Авторизоваться |
| registrationButton | Кнопка регистрации | Text | Зарегистрироваться |

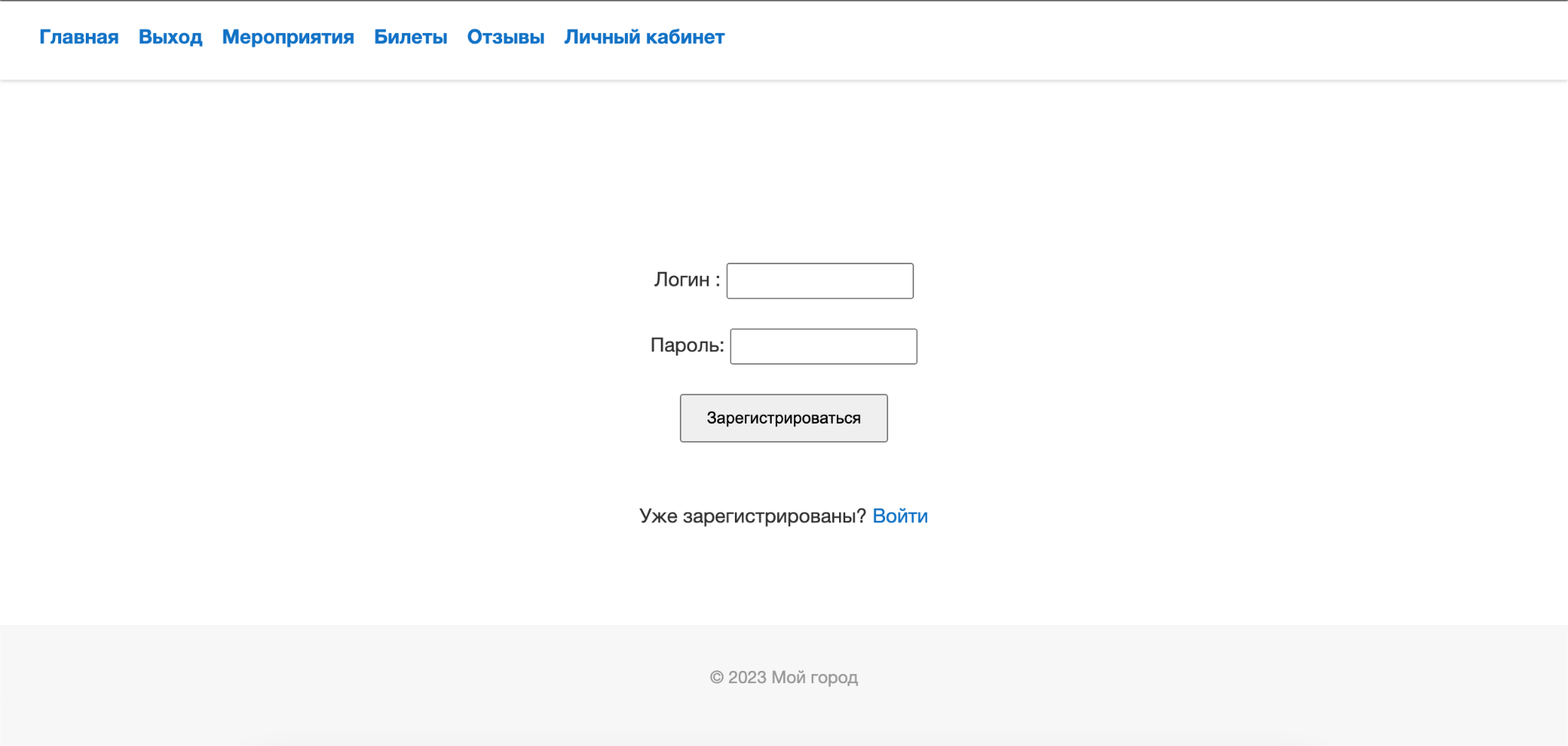
В таблице 3.15 представлен список компонентов формы авторизации. Реализация этой формы представлена на рисунке ниже:



1. 3.16 – Форма авторизации
2. 3.16 – Компоненты формы регистрации

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| registration | Форма удаления мероприятия | Text | Регистрация |
| loginLabel | Подпись у логина | Text | Логин |
| passwordLabel | Подпись у пароля | Text | Пароль |
| loginInput | Поле ввода логина | Text | Логин |
| passwordInput | Поле ввода пароля | Text | Пароль |
| loginButton | Кнопка авторизации | Text | Авторизоваться |
| registrationButton | Кнопка регистрации | Text | Зарегистрироваться |

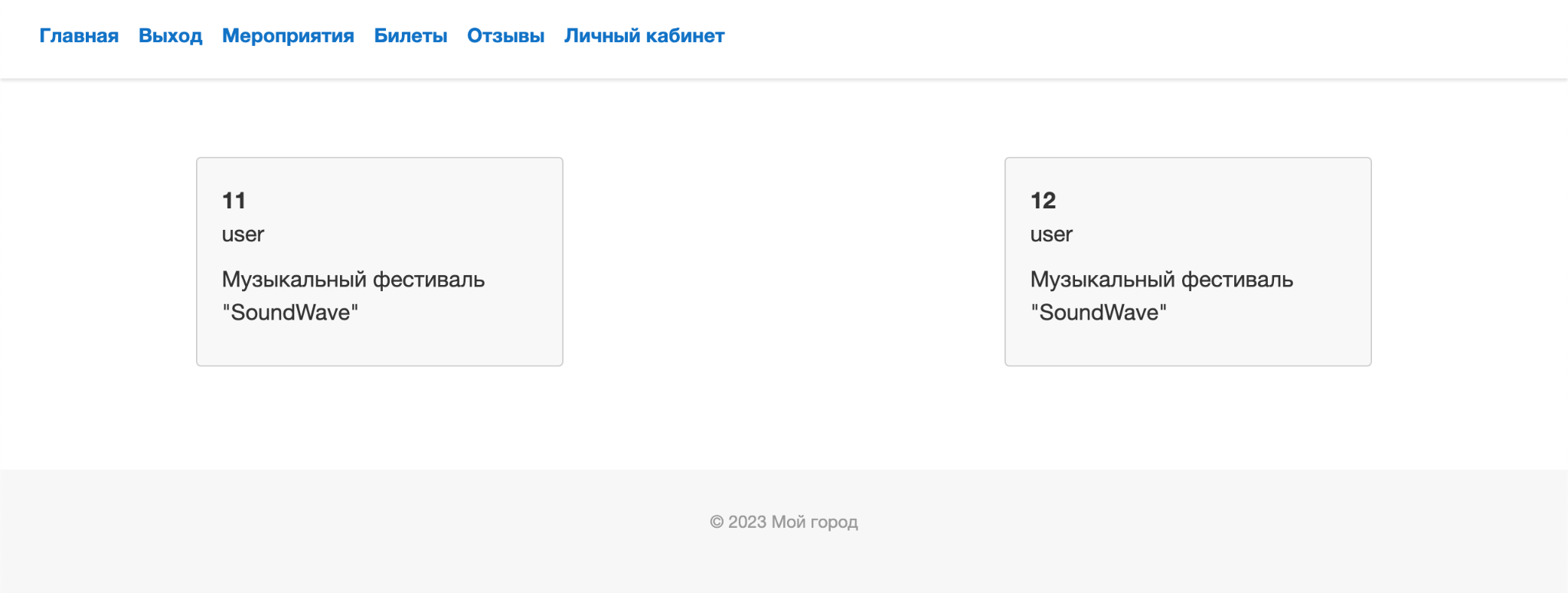
В таблице 3.16 представлен список компонентов формы регистрации. Реализация этой формы представлена на рисунке ниже:



1. 3.17 – Форма регистрации
2. 3.17 – Компоненты формы билетов

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| tickets | Форма с билетами | Text | Билеты |
| idLabel | Подпись номера билета | Text | Номер |
| userLabel | Подпись пользователя | Text | Пользователь |
| eventLabel | Поле мероприятия | Text | Мероприятия |

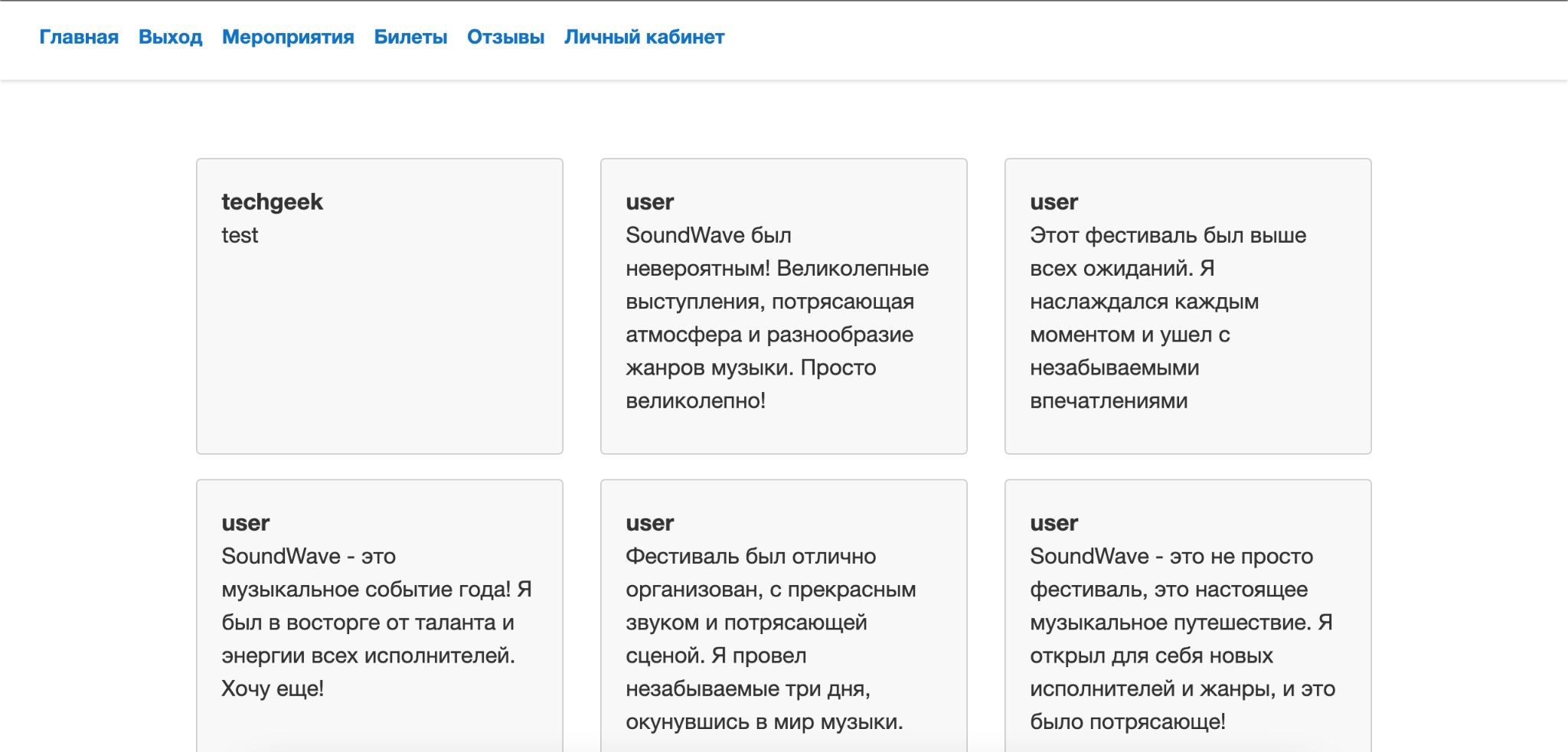
В таблице 3.17 представлен список компонентов формы билетов. Реализация этой формы представлена на рисунке ниже:



1. 3.18 – Форма билетов
2. 3.18 – Компоненты формы отзывов

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| reviews | Отзывы | Text | Отзывы |
| textLabel | Отзыв | Текст | Отзыв |
| userLabel | Пользователь | Текст | Пользователь |

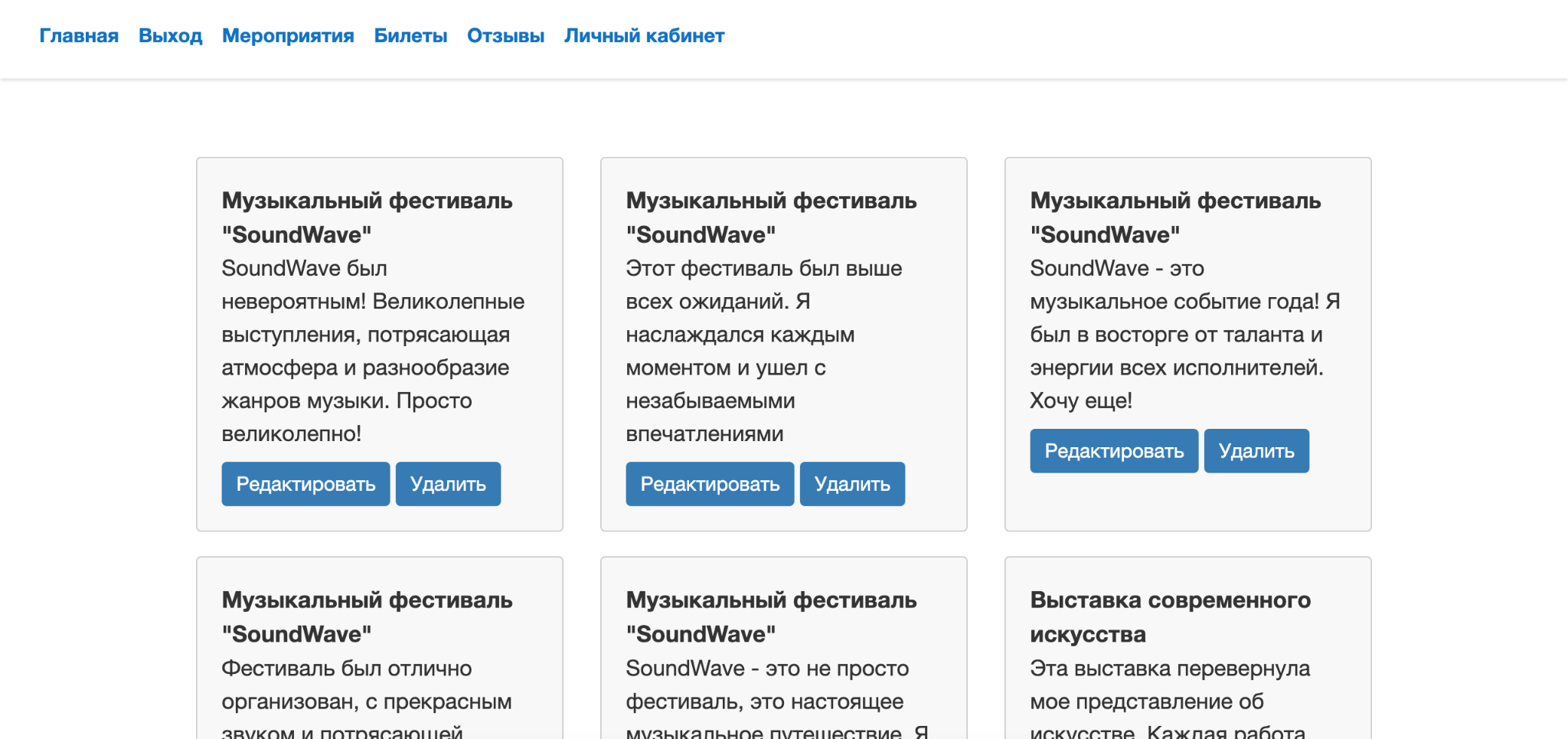
В таблице 3.18 представлен список компонентов формы отзывов. Реализация этой формы представлена на рисунке ниже:



1. 3.19 – Форма отзывов
2. 3.19 – Компоненты формы отзывов пользователя

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| UserReviews | Отзывы | Text | Отзывы |
| textLabel | Отзыв | Текст | Отзыв |
| commentLabel | Отзыв | Текст | Отзыв |
| editReview | Кнопка изменения отзыва | Text | Изменить отзыв |
| deleteReview | Кнопка удаления отзыва | Text | Удалить отзыв |

В таблице 3.19 представлен список компонентов формы отзывов пользователя. Реализация этой формы представлена на рисунке ниже:



1. 3.20 – Форма отзывов пользователя
2. 3.20 – Компоненты формы личного кабинета пользователя

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Назначение | Свойство | Значение |
| userProfile | Форма ЛК пользователя | Text | ЛК пользователя |
| label | Информация об ЛК клиента | Text | Информация об ЛК клиента |
| editData | Изменение данных | Text | Изменение данных |
| editPassword | Изменение пароля | Text | Изменение пароля |

В таблице 3.20 представлен список компонентов формы личного кабинета пользователя. Реализация этой формы представлена на рисунке ниже:

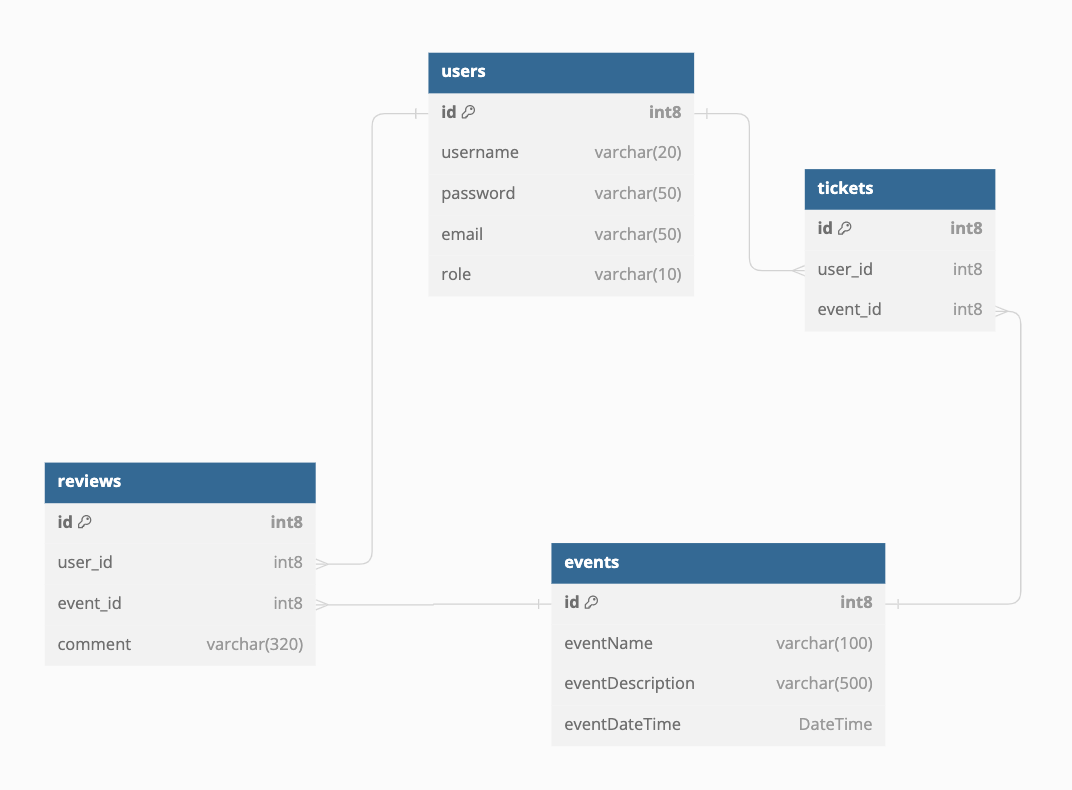


1. 3.21 – Форма личного кабинета

## 

## 3.3 Реализация базы данных

Для реализации базы данных используем PostgreSQL.



1. 3.13 – Схема базы данных

На рисунке 3.13 представлена схема базы данных разрабатываемой системы

**Таблица Users**

1. id – идентификатор пользователя, первичный ключ.
2. username – логин пользователя.
3. password – пароль пользователя.
4. email – почта пользователя.
5. role – роль пользователя.

**Таблица Events**

1. id – идентификатор мероприятия, первичный ключ.
2. eventName – название мероприятия.
3. eventDescription – описание мероприятия.
4. eventDateTime – дата проведения мероприятия.

**Таблица Reviews**

1. id – идентификатор отзыва, первичный ключ.
2. user\_id – пользователь.
3. event\_id – мероприятие.
4. comments – отзыв.

**Таблица Tickets**

1. id – идентификатор билета, первичный ключ.
2. user\_id – пользователь.
3. event\_id – мероприятие.

Таблица Reviews связана с таблицами Users и Events внешним ключом столбцами Reviews.user\_id и Users.id, Reviews.event\_id и Events.id, таблицы Users и Events является главной.

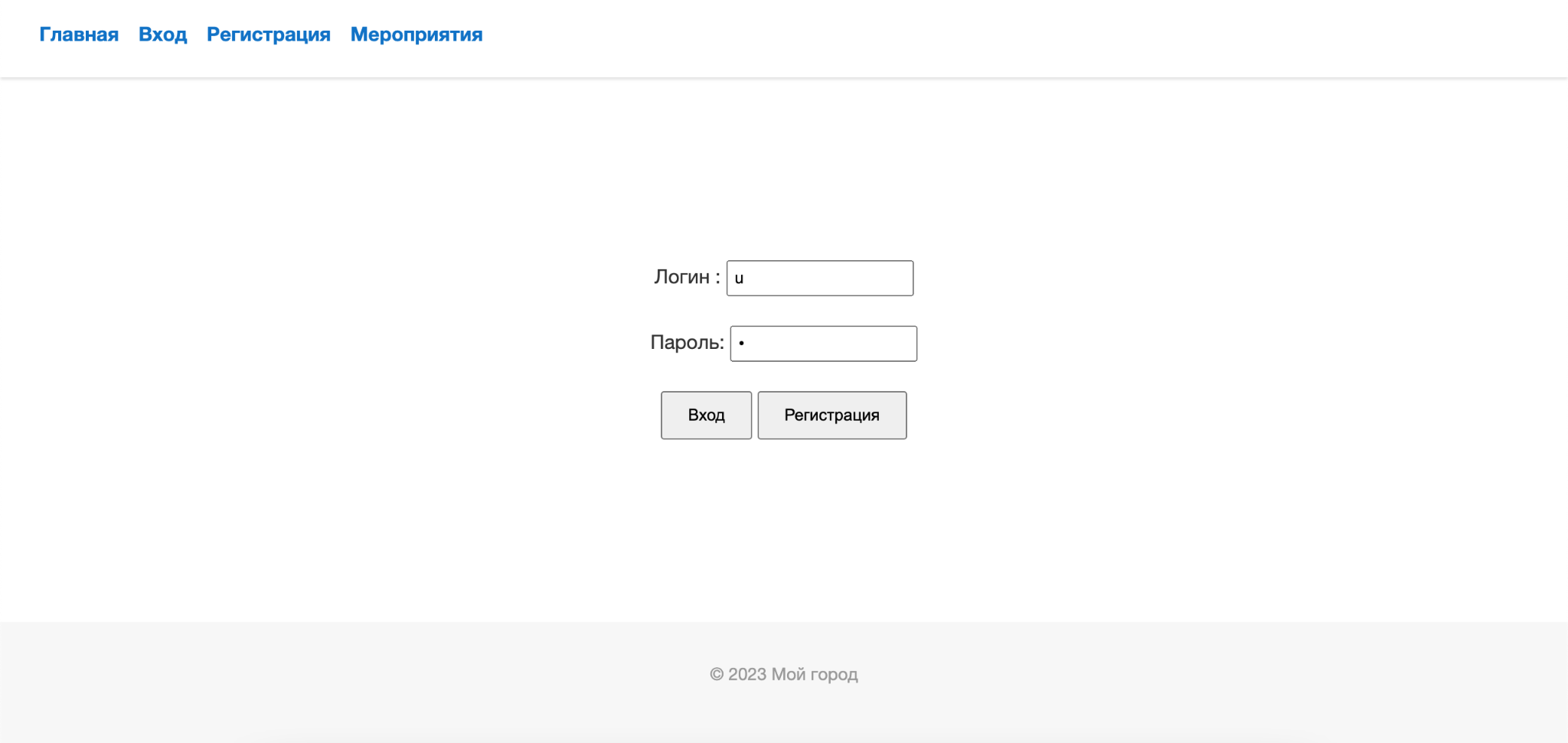
Таблица Tickets связана с таблицами Users и Events внешним ключом столбцами Tickets.user\_id и Users.id, Tickets.event\_id и Events.id, таблицы Users и Events является главной.

# 4 Тестирование ПрИС

Протестируем созданный прототип на тестовых данных.

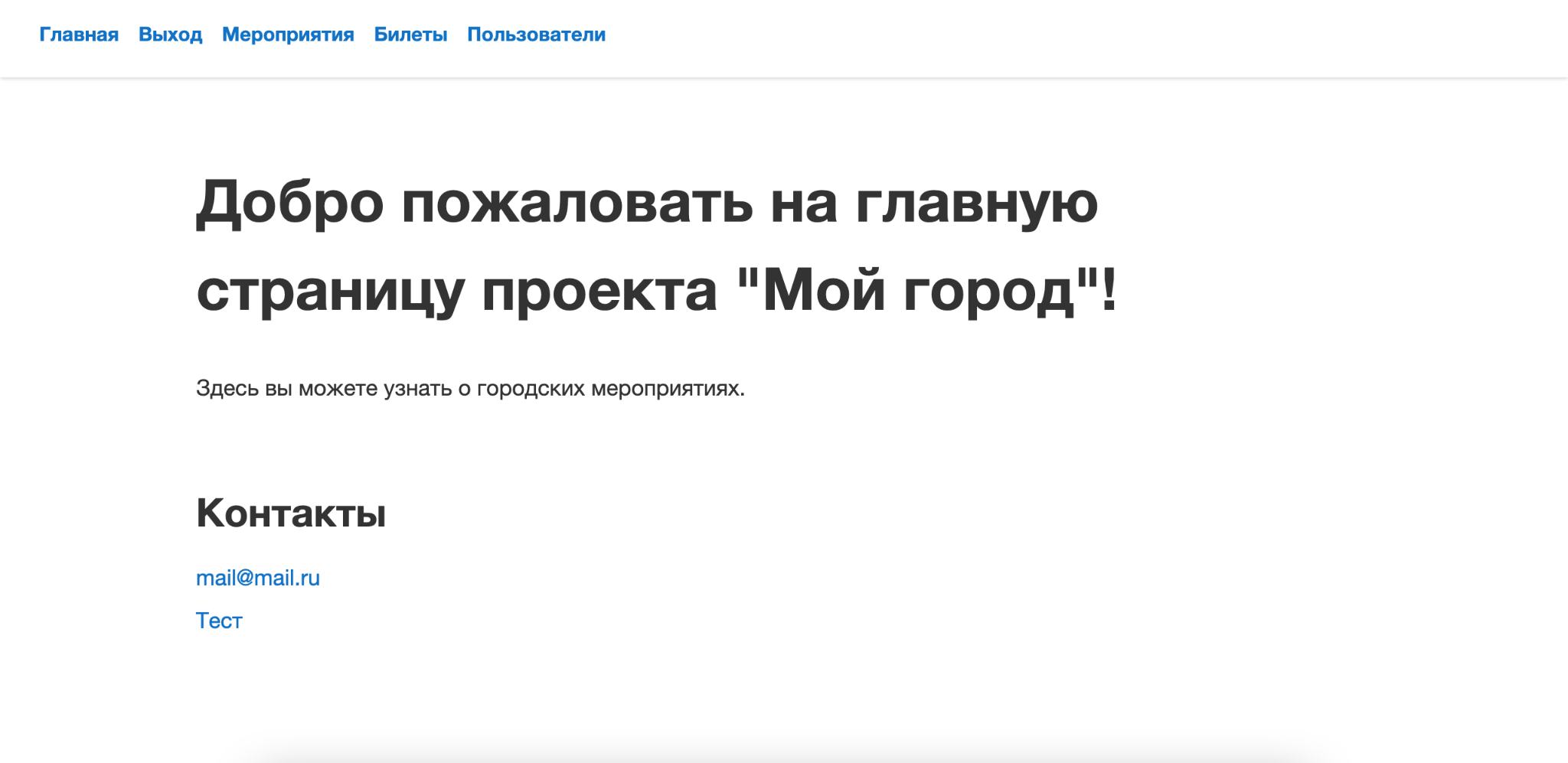
**Авторизация**

Введем авторизационные данные:



1. 4.1 – Ввод логина и пароля

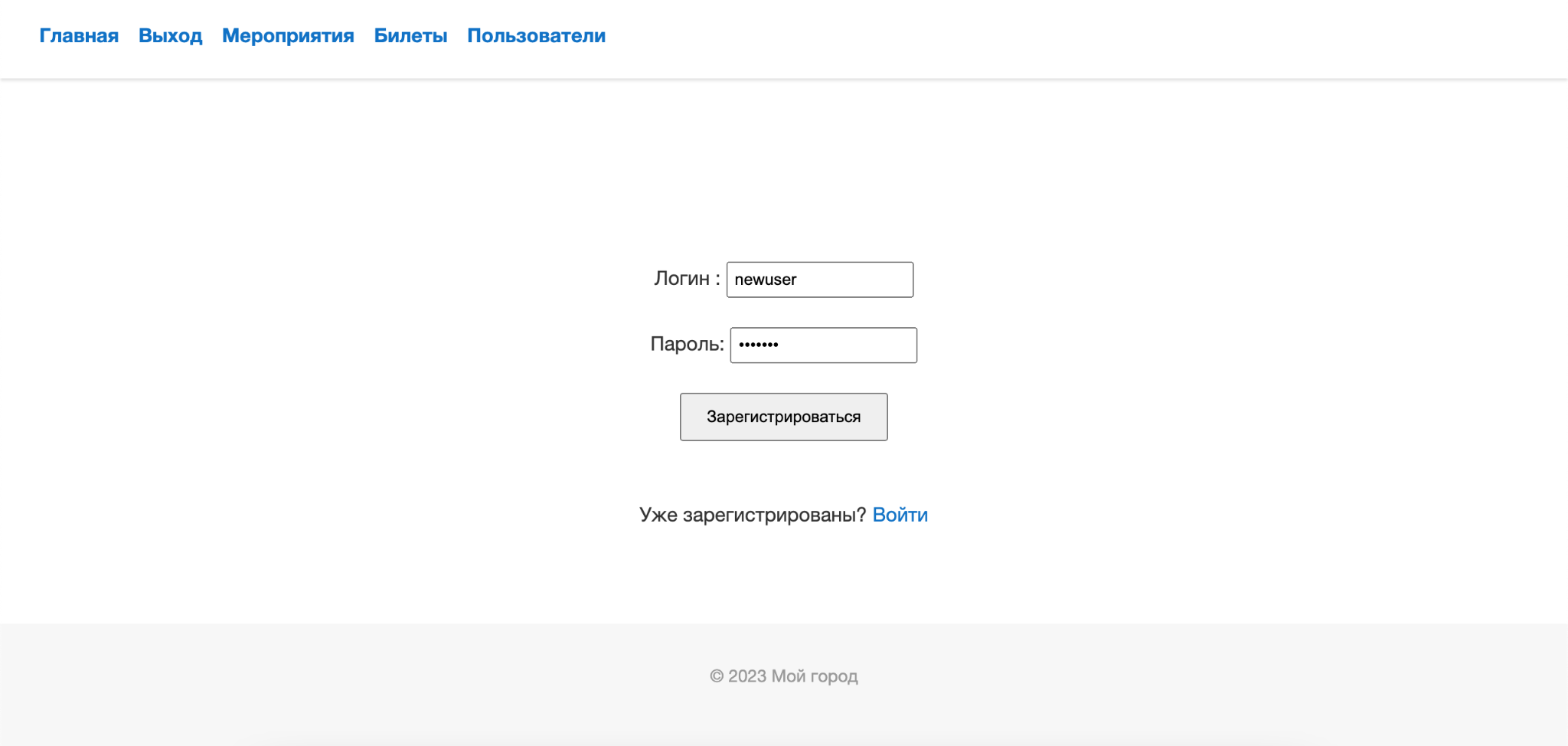
При нажатии на кнопку Войти откроется главная страница:



1. 4.2 – Главная страница

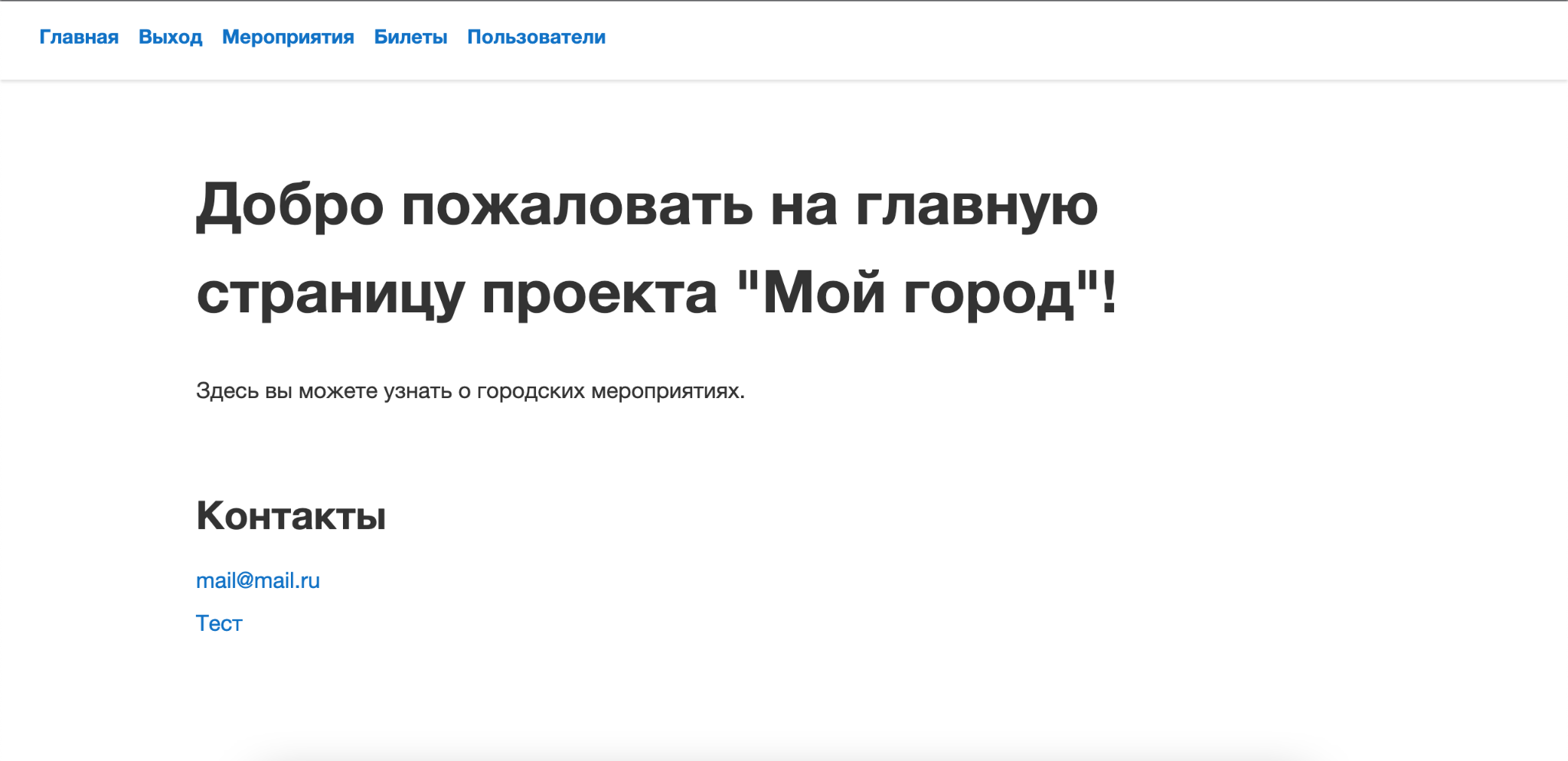
**Регистрация**

Введем регистрационные данные:



1. 4.3 – Проверка формы регистрации

При нажатии на кнопку Зарегистрироваться откроется главная страница:



1. 4.4 – Проверка формы изменения локации

Проверим, что в базу данных добавилась запись о пользователе:

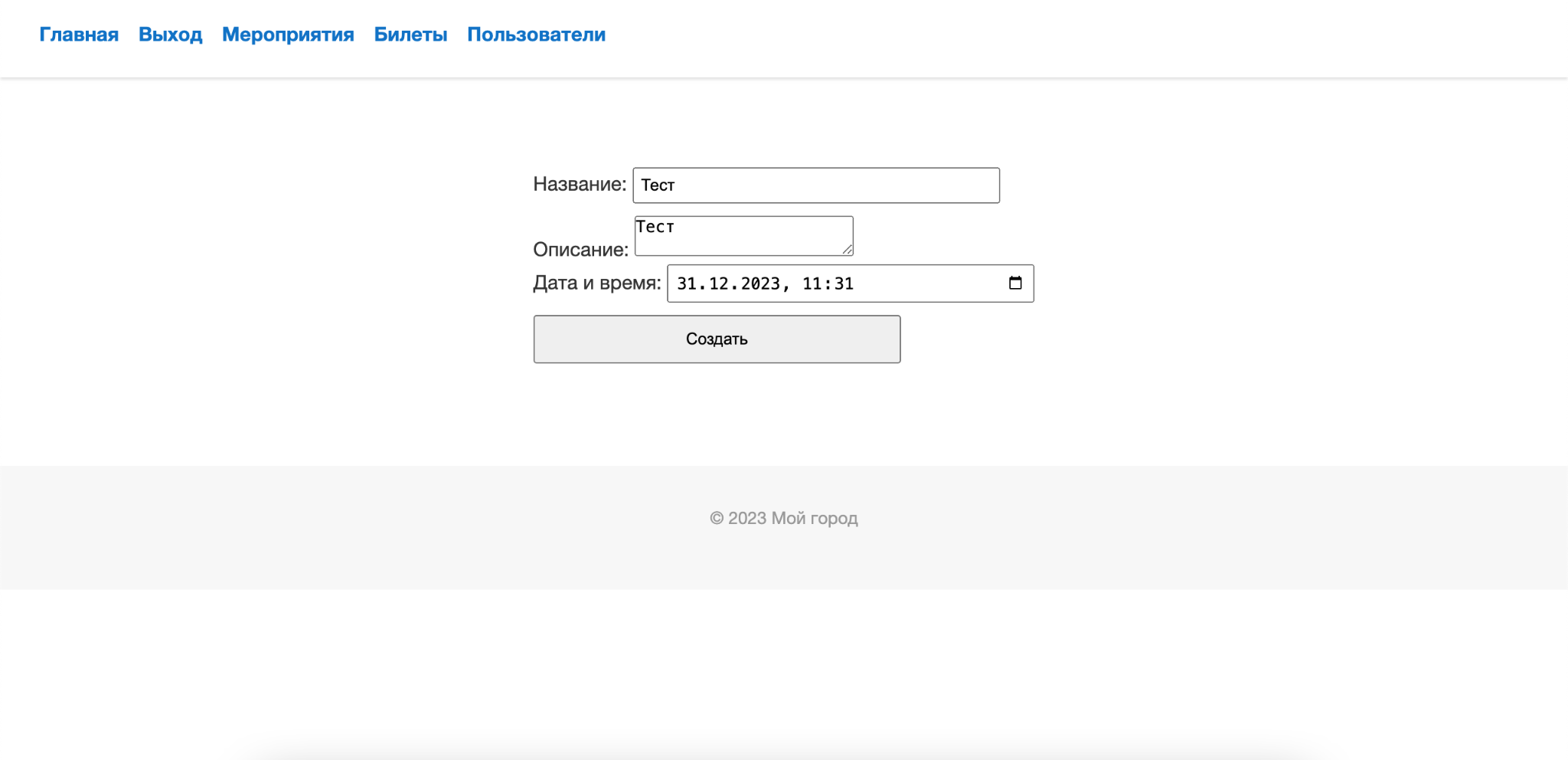


1. 4.5 – Проверка добавления пользователя в БД

Так как на ПК отсутствует интерфейс работы с БД, проверка осуществляется в консоли. На рисунке выше видно, что пользователь в БД был успешно добавлен. Там же можно увидеть, что пароль хранится в зашифрованном виде.

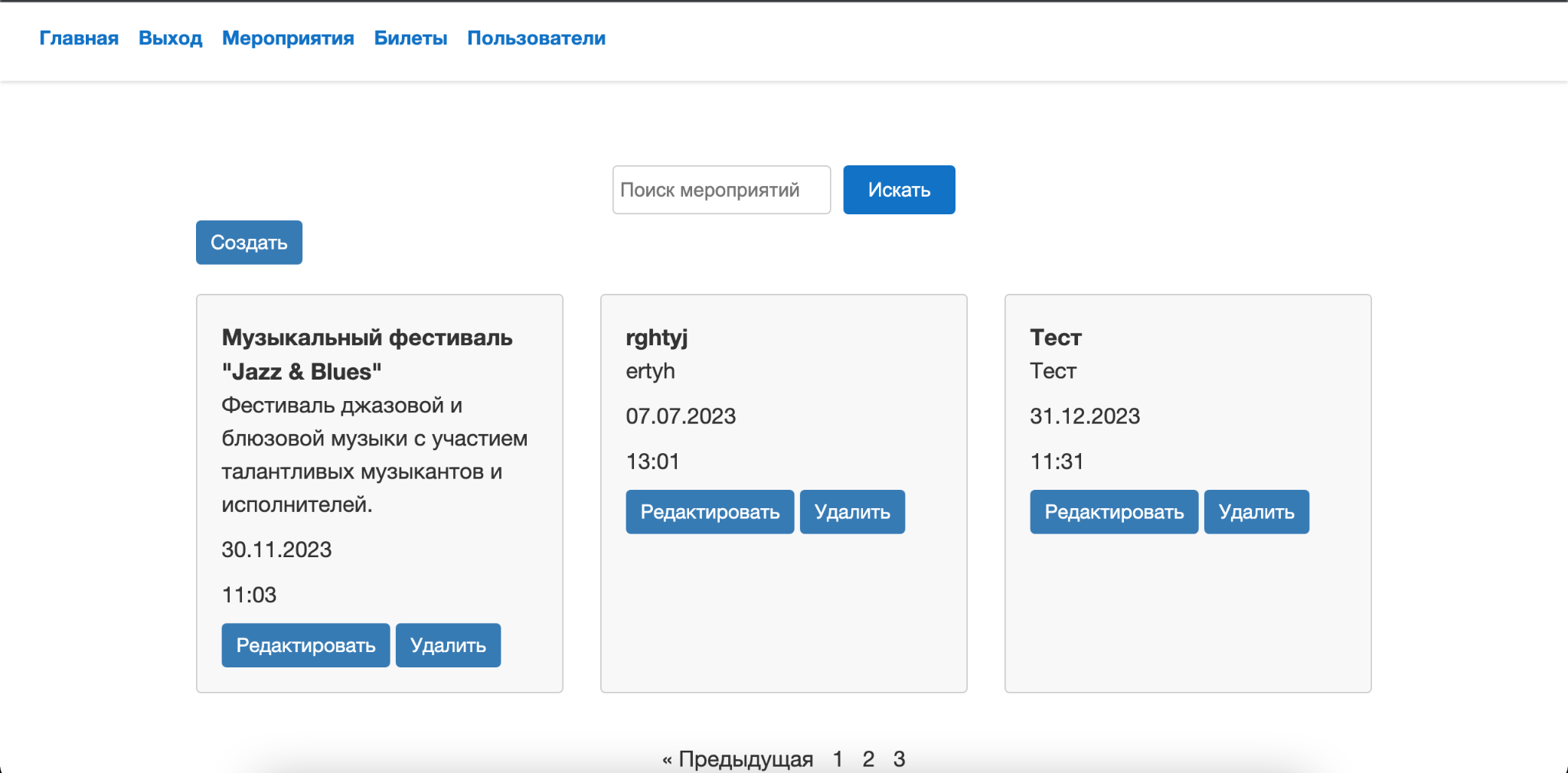
**Создание мероприятия**

При заполнении данных на форме в базу добавляется новое значение:



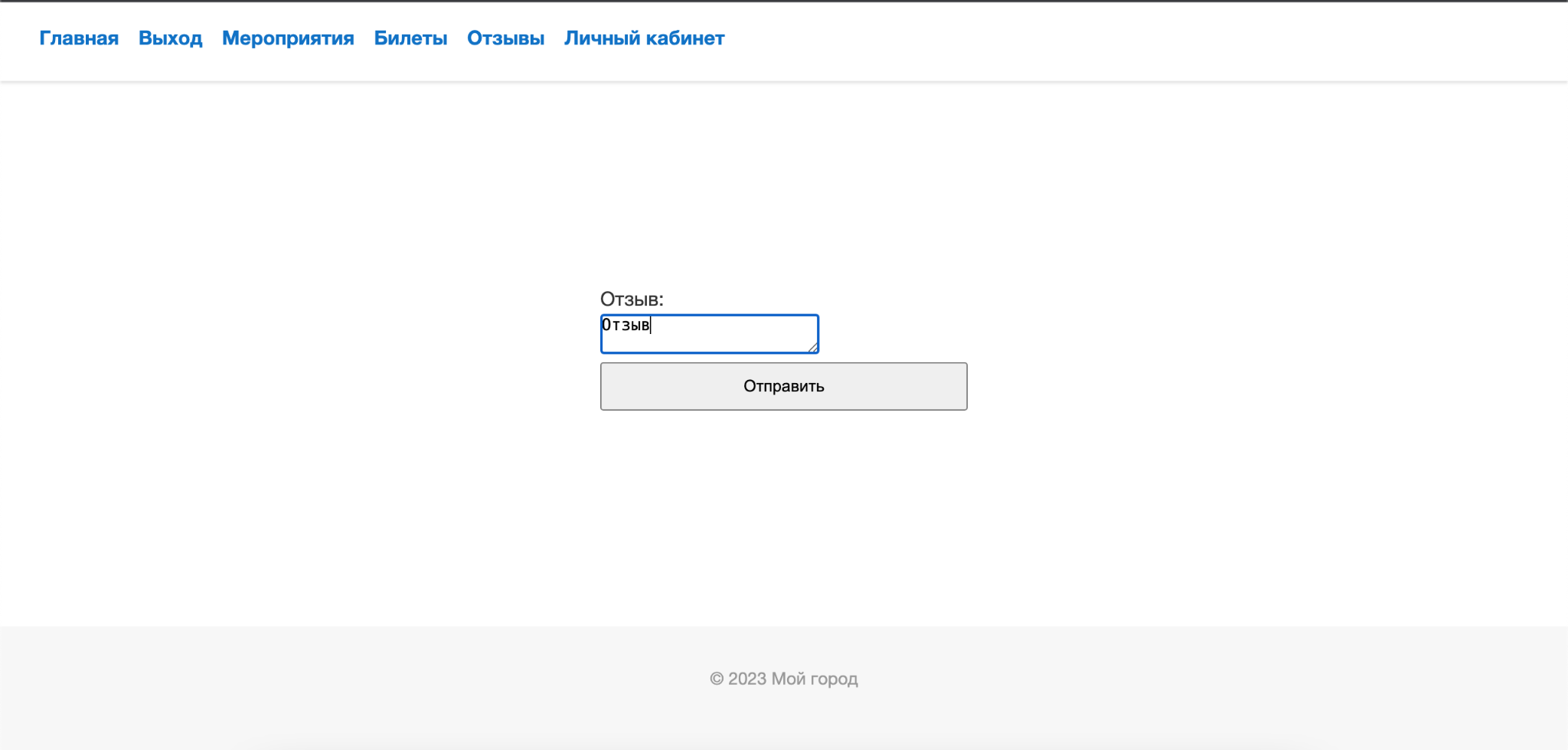
1. 4.6 – Форма создания мероприятия

Проверим, что мероприятие было создано:



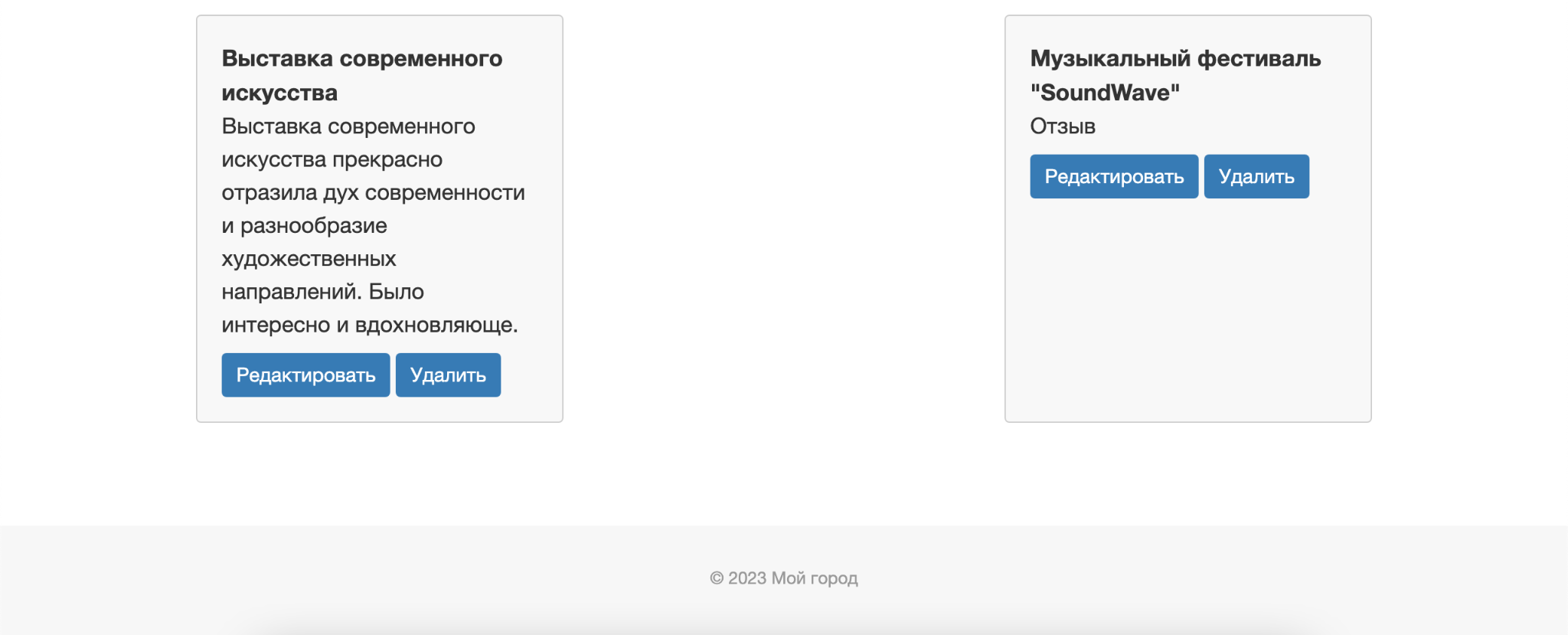
1. 4.7 – Форма с мероприятиями

**Создание отзыва**

При заполнении данных на форме в базу добавляется новое значение:

1. 4.8 – Форма создания мероприятия

Проверим, что отзыв был создан:



1. 4.9 – Форма с отзывами

# 5 Состав проекта

Проект включает в себя следующие файлы:

1. Классы-сущности;
2. Управляющие классы;
3. Класс с методом main;
4. HTML-страницы;
5. Файлы БД.

# 6 Компиляция и сборка системы

## 6.1 Правила создания системы

Для успешной сборки и дальнейшей разработки проекта необходимо установить на компьютере следующее программное обеспечение:

1. Intellij Idea с Java 8
2. СУБД PostgreSQL;

Системные требования, необходимые для запуска системы:

1. операционная система:Windows 10,11 или Linux Ubuntu, Debian;
2. процессор: 1.6 GHz или выше;
3. оперативная память: 2 GB или больше.

В случае использования текущей версии системы с полными правами доступа (доступ к функционалу администратора), среда разработки Microsoft Visual Studio не понадобится, однако СУБД может пригодиться для непосредственной работы с базой данных.

В случае использования текущей версии системы с полными правами доступа (доступ к функционалу администратора), но без необходимости использования непосредственного подключения к базе данных, достаточно установить полный пакет файлов, указанных в пункте 5.

В случае использования текущей версии системы без прав администратора (только для покупки билетов), достаточно установить первые два файла, перечисленные в пункте 5. Однако этот вариант использования системы нежелательный, так как в случае возникновения необходимости получить доступ к расширенным возможностям, могут появиться проблемы при переустановке системы.

## 6.2 Состав приложения

Файлы, предназначенные для дистрибуции в первую очередь, включают в себя: или файл проекта или докер-файл. В случае использования докер-файла требуется лишь запустить контейнер в приложении Docker, все остальное система сделает сама.

# 7 Документация

## 7.1 Назначения системы и выполняемые ею функции

Проектируемая система позволяет пользователю покупать билеты на мероприятия в своем городе. Система реализует функционал:

1. покупка билета;
2. добавление, изменение, удаление мероприятий;
3. добавление, изменение, удаление пользователей;
4. добавление, изменение, удаление отзывов.

## 7.2 Условия применения системы

Минимальные системные требования:

1. операционная система Windows XP Service Pack 2 и выше, Windows Server 2003 и выше;
2. процессор с тактовой частотой не ниже 1,8 ГГц;
3. оперативная память 2 Гбайт;
4. видеоадаптер с минимальным разрешением 720p.

## 7.3 Правила инсталляции системы

Для инсталляции программы необходимо установить на ПК следующее ПО:

1. Intellij Idea с Java 8 и выше с СУБД PostgreSQL.
2. Docker.

После этого открывается файл проекта через Intellij Idea (рисунок 7.1)

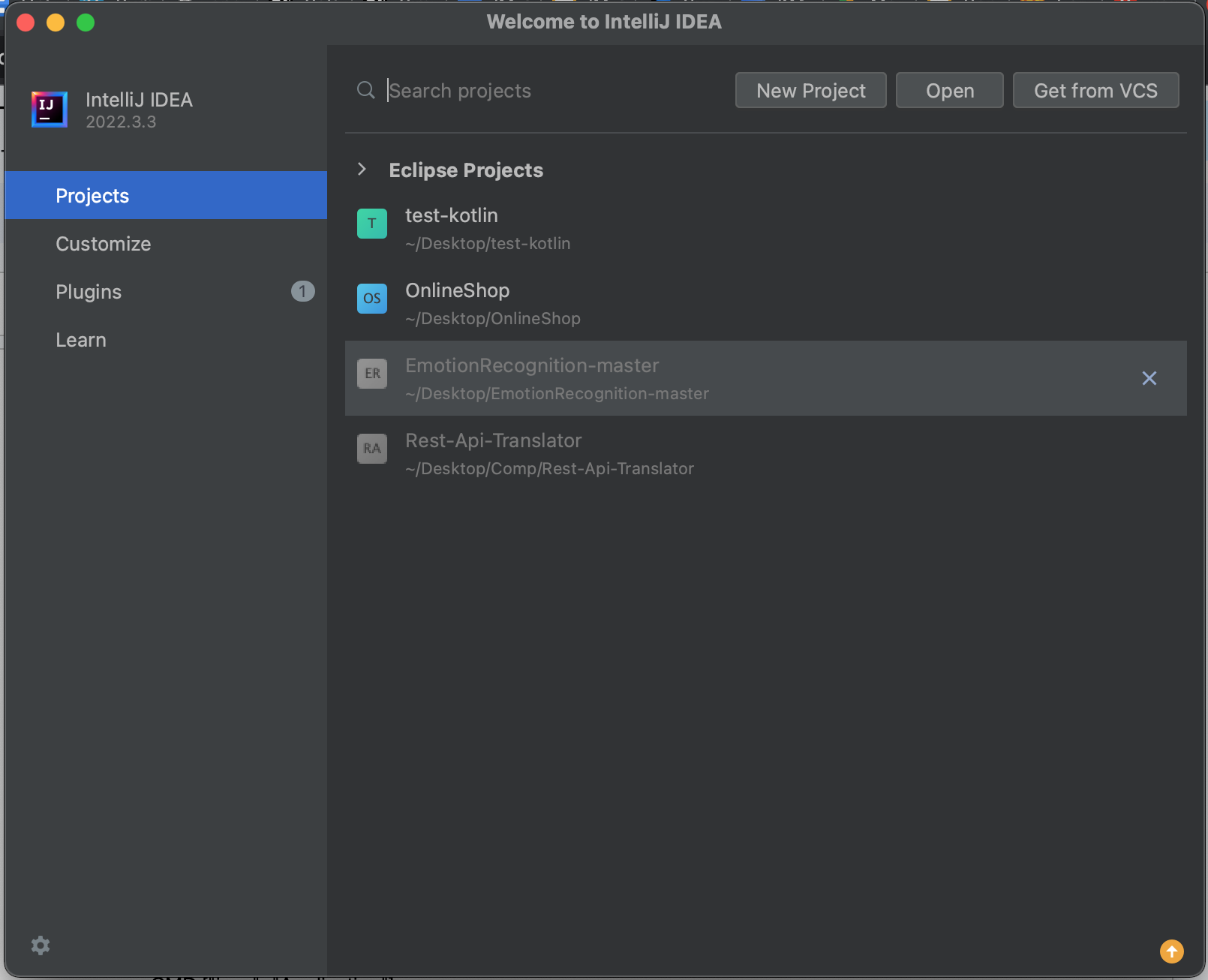


Рисунок 3.1 – Выбор проекта

В корне проекта создается Docker-файл со следующим содержанием:

FROM amazoncorretto:17

COPY . /Users/mac/Desktop/application

WORKDIR /Users/mac/Desktop/application

RUN javac ./src/main/java/ru/rsreu/lutikov/Application.java

CMD ["java", "Application"]

Директории меняются в зависимости от расположения.

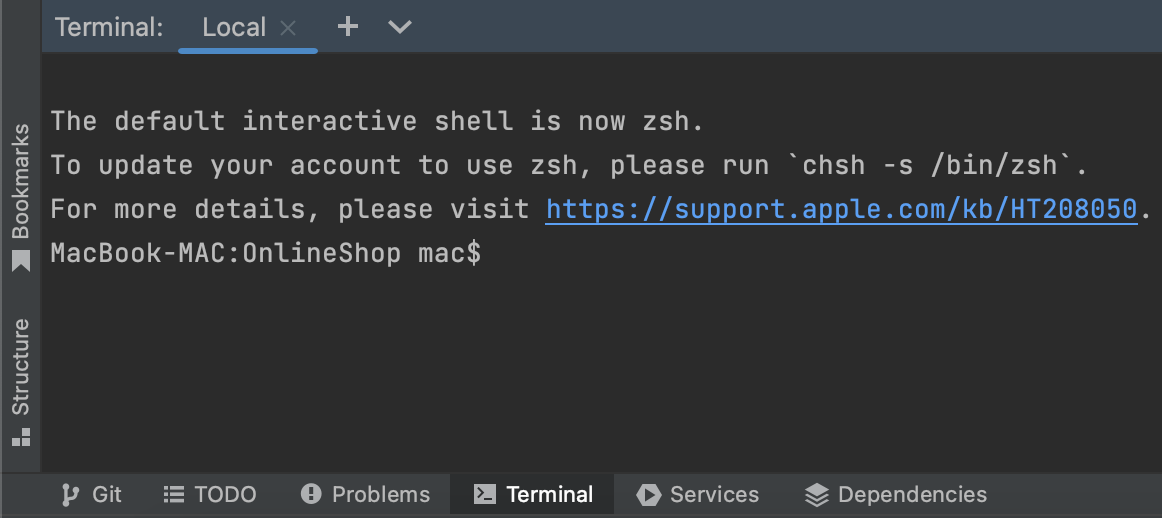
После сохранения необходимо открыть терминал в проекте (рисунок 7.2).

Рисунок 3.2 – Окно терминала

В терминал пишется следующая команда:

./mvnw install

docker build -t application .

Перед запуском необходимо запустить Docker.

Для запуска проекта используется следующая команда:

./mvnw install

docker build -t application .

После запуска заходим в браузер и вводим команду <http://localhost:8080>.

Приложение запущено (рисунок 7.3).

## 7.4 Сценарий взаимодействия с пользователем

Пользователи системы делятся на администраторов и обычных пользователей. Обычный пользователь имеет доступ к авторизации и покупке билетов, а также к редактированию отзывов.

Для покупки билета пользователь на странице авторизации, изображённой на рисунке 3.2, должен нажать на кнопку «Купить билет» после чего откроется новая форма, изображенная на рисунке 3.12. На ней пользователь может на свое усмотрение выбрать мероприятие, после чего пользователь должен нажать на кнопку «Купить билет» для оплаты своего заказа. В случае успешной оплаты пользователю на почту придет сообщение с купленным билетом.

Если пользователь хочет получить доступ к функционалу администратора, ему необходимо авторизироваться в системе. Для этого на форме авторизации (рисунок 3.2) пользователь должен ввести логин, пароль и нажать на кнопку «Войти».

Если комбинация логина и пароля верны (запись такого аккаунта присутствует в базе данных), то пользователь получает права администратора. Ему становится доступна панель с опциями (рисунок 3.3). Администратор может выбрать необходимое ему действие (добавление, удаление, изменение мероприятия и пользователя) после чего откроется форма, советующая выбранному действию.

В случае добавления администратор должен заполнить все поля формы и нажать на кнопку «Добавить».

В случае изменения администратор должен выбрать из списка, как объект он хочет изменить. После он может изменить поля по его желанию. Для сохранения изменений нужно нажать на кнопку «Сохранить».

В случае удаления объекта администратор должен выбрать объект из списка и нажать на кнопку «Удалить».

# Заключение

В результате проведенного анализа и проектирования информационной системы для поиска мероприятий и покупки билетов, мы выявили неотъемлемую роль современных технологий в упрощении и улучшении нашей повседневной жизни. Наш проект стремится соответствовать этим тенденциям, предлагая эффективное и удобное приложение, спроектированное с использованием передовых технологий.

Используя язык программирования Java 17 и фреймворк Spring, мы строим надежное и масштабируемое приложение. Frontend разрабатывается на HTML и CSS в среде разработки IntelliJ Idea CE, обеспечивая привлекательный и интуитивно понятный пользовательский интерфейс. База данных PostgresQL обеспечивает надежное хранение и управление данными о мероприятиях, билетах и отзывах на них.

Предложенная информационная система позволит пользователям с легкостью находить мероприятия, оставлять на них отзывы и покупать билеты, тем самым внося вклад в современные тенденции цифровизации культурной жизни.

# Список использованных источников

Ниже представлен список использованных источников.

1. Белов В.В., Чистякова В.И. Проектирование информационных систем: учебник – М.: КУРС, 2018. – 400 с. ISBN 978-5-906923-53-0 (КУРС) (45 экз. в БФ РГРТУ).

2. Проектирование информационных систем: Методические указания к курсовому проектированию / Рязан. гос. радиотехн. ун-т. им. В.Ф. Уткина; Сост.: В.В. Белов, В.И. Чистякова. Рязань, 2022. 40 с

3. Иванов, Денис Юрьевич. Унифицированный язык моделирования UML [Электронный ресурс]: учебное пособие для вузов по направлению подготовки "Системный анализ и управление" / Д.Ю. Иванов, Ф.А. Новиков; Санкт-Петербургский государственный политехн. унт. – Электрон. текстовые дан. (1 файл: 1,83 Мб). – Санкт-Петербург, 2011. – Загл. с титул. экрана. – Электронная версия печатной публикации. – Свободный доступ из сети Интернет (чтение, печать, копирование). – Текстовый документ. – Adobe Acrobat Reader 7.0. URL: http://elib.spbstu.ru/dl/2962.pdf/download/2962.pdf

4. OMG® Unified Modeling Language® (OMG UML®) Version 2.5.1 December 2017. URL: https/www.omg.org/spec/UML/

5. Маклаков, С. В. Bpwin и Erwin: CASE средства разработки информационных систем / С. В. Маклаков. – М.: Диалог-МИФИ, 2000. 256 с.

6. Каюмова А.В. Визуальное моделирование систем в StarUML: Учебное пособие/ А.В. Каюмова. Казань. – Казанский федеральный университет, 2013. – 104 с

# ПРИЛОЖЕНИЯ

## Приложение 1. Классы-сущности

### Класс User

package ru.rsreu.lutikov.sber.domain;

import javax.persistence.\*;

import java.util.Iterator;

import java.util.Objects;

import java.util.Set;

@Entity

@Table(name = "users")

public class User {

@OneToMany(mappedBy = "id")

private Set<Ticket> ticket;

@OneToMany(mappedBy = "id")

private Set<RefreshToken> refreshToken;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(unique = true)

private String username;

private String password;

@Column(unique = true)

private String email;

private boolean active;

@ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)

@CollectionTable(name = "user\_role", joinColumns = @JoinColumn(name = "user\_id"))

@Enumerated(EnumType.STRING)

private Set <Role> roles;

public User() {

}

public User (String username, String password, String email, boolean active, Set<Role> role) {

this.username = username;

this.password = password;

this.email = email;

this.active = active;

this.roles = role;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public Set<Role> getRole() {

return roles;

}

public void setRole(Set<Role> role) {

this.roles = role;

}

// Методы equals и hashCode

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

User user = (User) o;

return Objects.equals(id, user.id);

}

@Override

public int hashCode() {

return Objects.hash(id);

}

public boolean isActive() {

return active;

}

public void setActive(boolean active) {

this.active = active;

}

public String[] getRoles() {

Iterator<Role> iterator = roles.iterator();

String[] roleArray = new String[roles.size()];

int i = 0;

while(iterator.hasNext()) {

roleArray[i] = iterator.next().name();

i++;

}

return roleArray;

}

// Дополнительные методы

}

### Класс Event

package ru.rsreu.lutikov.sber.domain;

import org.springframework.format.annotation.DateTimeFormat;

import javax.persistence.\*;

import java.time.LocalDateTime;

import java.util.Set;

@Entity

@Table(name = "events")

public class Event {

@OneToMany(mappedBy = "id")

private Set<Ticket> ticket;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String eventName;

private String eventDescription;

@DateTimeFormat(iso = DateTimeFormat.ISO.DATE\_TIME)

private LocalDateTime eventDateTime;

public Event() {

}

public Event(String eventName, String eventDescription, LocalDateTime eventDateTime) {

this.eventName = eventName;

this.eventDescription = eventDescription;

this.eventDateTime = eventDateTime;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getEventName() {

return eventName;

}

public void setEventName(String eventName) {

this.eventName = eventName;

}

public String getEventDescription() {

return eventDescription;

}

public void setEventDescription(String eventDescription) {

this.eventDescription = eventDescription;

}

public LocalDateTime getEventDateTime() {

return eventDateTime;

}

public void setEventDateTime(LocalDateTime eventDateTime) {

this.eventDateTime = eventDateTime;

}

// Дополнительные методы

}

### Класс Review

package ru.rsreu.lutikov.sber.domain;

import javax.persistence.\*;

import java.util.Objects;

@Entity

@Table(name = "reviews")

public class Review {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "user\_id")

private User user;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "event\_id")

private Event event;

private String comment;

public Review() {

}

public Review (User user, Event event, String comment) {

this.user = user;

this.event = event;

this.comment = comment;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public Event getEvent() {

return event;

}

public void setEvent(Event event) {

this.event = event;

}

public String getComment() {

return comment;

}

public void setComment(String comment) {

this.comment = comment;

}

// Методы equals и hashCode

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Review review = (Review) o;

return Objects.equals(id, review.id);

}

@Override

public int hashCode() {

return Objects.hash(id);

}

// Дополнительные методы

}

### Класс Ticket

package ru.rsreu.lutikov.sber.domain;

import com.fasterxml.jackson.annotation.\*;

import javax.persistence.\*;

import java.util.Objects;

@Entity

@Table(name = "tickets")

public class Ticket {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne(fetch = FetchType.EAGER)

@JoinColumn(name = "user\_id")

@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")

@JsonIdentityReference(alwaysAsId = true)

@JsonProperty("user\_id")

private User user;

@ManyToOne(fetch = FetchType.EAGER)

@JoinColumn(name = "event\_id")

@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")

@JsonIdentityReference(alwaysAsId = true)

@JsonProperty("event\_id")

private Event event;

public Ticket() {

}

public Ticket(User user, Event event) {

this.user = user;

this.event = event;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public Event getEvent() {

return event;

}

public void setEvent(Event event) {

this.event = event;

}

// Методы equals и hashCode

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Ticket ticket = (Ticket) o;

return Objects.equals(id, ticket.id);

}

@Override

public int hashCode() {

return Objects.hash(id);

}

// Дополнительные методы

}

## Приложение 2. Управляющие классы

### Класс HomeController

/\*\*

\* Controller class for handling home-related requests.

\*

\* <p>

\* This class is responsible for handling requests to the root path ("/").

\* </p>

\*

\* <p>

\* <strong>Author:</strong> Vadim

\* <br>

\* <strong>Email:</strong> blinvadik@mail.ru

\* </p>

\*

\* @see org.springframework.stereotype.Controller

\* @see org.springframework.web.bind.annotation.GetMapping

\*

\* @since 2023

\*/

package ru.rsreu.lutikov.sber.controllers;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

@Controller

public class HomeController {

/\*\*

\* Handles requests to the root path ("/").

\*

\* @return the name of the view to be rendered

\*/

@GetMapping("/")

public String home() {

return "home.html";

}

}

### Класс LogoutController

/\*\*

\* Controller class for handling logout-related requests.

\*

\* <p>

\* This class is responsible for handling requests to the "/logout" path and performing logout functionality.

\* </p>

\*

\* <p>

\* <strong>Author:</strong> Vadim

\* <br>

\* <strong>Email:</strong> blinvadik@mail.ru

\* </p>

\*

\* @see org.springframework.stereotype.Controller

\* @see org.springframework.web.bind.annotation.GetMapping

\* @see javax.servlet.http.HttpServletRequest

\* @see javax.servlet.http.HttpSession

\*

\* @since 2023

\*/

package ru.rsreu.lutikov.sber.controllers;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpSession;

@Controller

public class LogoutController {

/\*\*

\* Handles requests to the "/logout" path.

\*

\* @param request the HttpServletRequest object to access the current session

\* @return the redirect view name after logout

\*/

@GetMapping("/logout")

public String logout(HttpServletRequest request) {

HttpSession session = request.getSession(false);

if (session != null) {

// Invalidating the session

session.invalidate();

}

// Redirecting to the page after logout

return "redirect:/login?logout";

}

}

### Класс EventController

/\*\*

\* Controller class for handling event-related requests.

\*

\* <p>

\* This class is responsible for handling requests related to events, such as displaying events,

\* creating new events, editing events, and deleting events.

\* </p>

\*

\* <p>

\* <strong>Author:</strong> Vadim

\* <br>

\* <strong>Email:</strong> blinvadik@mail.ru

\* </p>

\*

\* @see org.springframework.stereotype.Controller

\* @see org.springframework.beans.factory.annotation.Autowired

\* @see org.springframework.data.domain.Page

\* @see org.springframework.data.domain.PageRequest

\* @see org.springframework.ui.Model

\* @see org.springframework.validation.BindingResult

\* @see org.springframework.web.bind.annotation

\* @see ru.rsreu.lutikov.sber.domain.Event

\* @see ru.rsreu.lutikov.sber.repositories.EventRepository

\* @see ru.rsreu.lutikov.sber.services.EventService

\*

\* @since 2023

\*/

package ru.rsreu.lutikov.sber.controllers;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

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

import ru.rsreu.lutikov.sber.domain.Event;

import ru.rsreu.lutikov.sber.repositories.EventRepository;

import ru.rsreu.lutikov.sber.services.EventService;

import java.util.List;

@Controller

public class NewEventController {

@Autowired

private EventService eventService;

@Autowired

private EventRepository eventRepository;

/\*\*

\* Handles requests to the "/events" path.

\*

\* @param page the page number

\* @param search the search query string (optional)

\* @param model the Model object for passing data to the view

\* @return the name of the view to be rendered

\*/

@GetMapping("/events")

public String events(@RequestParam(value = "page", defaultValue = "1") int page,

@RequestParam(value = "search", required = false) String search,

Model model) {

int pageSize = 9; // Number of cards per page

Page<Event> eventPage;

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

eventPage = eventRepository.findByEventNameContainingIgnoreCase(search, PageRequest.of(page - 1, pageSize));

} else {

eventPage = eventRepository.findAll(PageRequest.of(page - 1, pageSize));

}

List<Event> yourDataList = eventPage.getContent(); // Get JSON data

model.addAttribute("yourDataList", yourDataList); // Pass data to the view model

model.addAttribute("currentPage", page);

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

return "events"; // Return the view name

}

/\*\*

\* Handles requests to the "/events/new" path.

\*

\* @param model the Model object for passing data to the view

\* @return the name of the view to be rendered

\*/

@GetMapping("/events/new")

public String createEventForm(Model model) {

model.addAttribute("event", new Event());

return "newEvent";

}

/\*\*

\* Handles requests to create a new event.

\*

\* @param event the Event object containing the event data

\* @param bindingResult the BindingResult object for validation errors

\* @return the redirect view name after creating the event

\*/

@PostMapping("/events/new")

public String createEvent(@ModelAttribute("event") Event event, BindingResult bindingResult) {

if (bindingResult.hasErrors()) {

System.out.println("ERROR");

return "newEvent";

}

eventRepository.save(event);

return "redirect:/events";

}

/\*\*

\* Handles requests to the "/events/{eventId}/edit" path.

\*

\* @param eventId the ID of the event to be edited

\* @param model the Model object for passing data to the view

\* @return the name of the view to be

rendered

\*/

@GetMapping("/events/{eventId}/edit")

public String editEventForm(@PathVariable Long eventId, Model model) {

Event event = eventService.getEventById(eventId);

model.addAttribute("event", event);

return "editEvent";

}

/\*\*

\* Handles requests to edit an event.

\*

\* @param eventId the ID of the event to be edited

\* @param event the Event object containing the updated event data

\* @param bindingResult the BindingResult object for validation errors

\* @return the redirect view name after editing the event

\*/

@PostMapping("/events/{eventId}/edit")

public String editEvent(@PathVariable String eventId, @ModelAttribute("event") Event event,

BindingResult bindingResult) {

if (bindingResult.hasErrors()) {

return "editEvent";

}

Long eventIdLong = Long.parseLong(eventId);

event.setId(eventIdLong);

eventService.updateEvent(event.getId(), event);

return "redirect:/events";

}

/\*\*

\* Handles requests to the "/events/{eventId}/delete" path.

\*

\* @param eventId the ID of the event to be deleted

\* @param model the Model object for passing data to the view

\* @return the name of the view to be rendered

\*/

@GetMapping("/events/{eventId}/delete")

public String deleteEventForm(@PathVariable Long eventId, Model model) {

Event event = eventService.getEventById(eventId);

model.addAttribute("event", event);

return "deleteEvent";

}

/\*\*

\* Handles requests to delete an event.

\*

\* @param eventId the ID of the event to be deleted

\* @return the redirect view name after deleting the event

\*/

@PostMapping("/events/{eventId}/delete")

public String deleteEventForm(@PathVariable("eventId") Long eventId) {

eventService.deleteEvent(eventId);

return "redirect:/events"; // Redirect to the events page after deleting

}

}

### Класс ReviewController

/\*\*

\* Controller class for handling review-related requests.

\*

\* <p>

\* This class is responsible for handling requests related to reviews, such as viewing reviews,

\* creating new reviews, editing reviews, and deleting reviews.

\* </p>

\*

\* <p>

\* <strong>Author:</strong> Vadim

\* <br>

\* <strong>Email:</strong> blinvadik@mail.ru

\* </p>

\*

\* @see org.springframework.stereotype.Controller

\* @see org.springframework.beans.factory.annotation.Autowired

\* @see org.springframework.ui.Model

\* @see org.springframework.web.bind.annotation.GetMapping

\* @see org.springframework.web.bind.annotation.ModelAttribute

\* @see org.springframework.web.bind.annotation.PathVariable

\* @see org.springframework.web.bind.annotation.PostMapping

\* @see ru.rsreu.lutikov.sber.domain.Event

\* @see ru.rsreu.lutikov.sber.domain.Review

\* @see ru.rsreu.lutikov.sber.domain.User

\* @see ru.rsreu.lutikov.sber.services.EventService

\* @see ru.rsreu.lutikov.sber.services.ReviewService

\* @see ru.rsreu.lutikov.sber.services.UserService

\* @since 2023-07-04

\*/

package ru.rsreu.lutikov.sber.controllers;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import ru.rsreu.lutikov.sber.domain.Event;

import ru.rsreu.lutikov.sber.domain.Review;

import ru.rsreu.lutikov.sber.domain.User;

import ru.rsreu.lutikov.sber.services.EventService;

import ru.rsreu.lutikov.sber.services.ReviewService;

import ru.rsreu.lutikov.sber.services.UserService;

import java.security.Principal;

import java.util.List;

@Controller

public class NewReviewController {

@Autowired

EventService eventService;

@Autowired

UserService userService;

@Autowired

ReviewService reviewService;

/\*\*

\* Handles requests to the "/reviews/{eventId}" path.

\*

\* @param eventId the ID of the event to view reviews for

\* @param model the Model object for passing data to the view

\* @return the name of the view to be rendered

\*/

@GetMapping("/reviews/{eventId}")

public String viewReviewForm(@PathVariable Long eventId, Model model) {

List<Review> yourDataList = reviewService.getReviewsByEventId(eventId);

model.addAttribute("yourDataList", yourDataList); // Pass data to the view model

return "reviews";

}

/\*\*

\* Handles requests to the "/reviews/{eventId}/create" path.

\*

\* @param eventId the ID of the event to create a review for

\* @param model the Model object for passing data to the view

\* @return the name of the view to be rendered

\*/

@GetMapping("/reviews/{eventId}/create")

public String createReviewForm(@PathVariable Long eventId, Model model) {

model.addAttribute("review", new Review());

return "newReview";

}

/\*\*

\* Handles requests to create a new review.

\*

\* @param review the Review object containing the review data

\* @param eventId the ID of the event the review belongs to

\* @param principal the Principal object representing the current authenticated user

\* @return the redirect view name after creating the review

\*/

@PostMapping("/reviews/{eventId}/create")

public String createReview(@ModelAttribute("review") Review review, @PathVariable("eventId") Long eventId, Principal principal) {

System.out.println("Error!");

String username = principal.getName();

Long userId = userService.getUserIdByUsername(username);

User user = userService.getUserById(userId);

Event event = eventService.getEventById(eventId);

review.setEvent(event);

review.setUser(user);

reviewService.createReview(review);

// Redirect the user to the tickets page or another page

return "redirect:/events";

}

/\*\*

\* Handles requests to the "/user/reviews" path.

\*

\* @param model the Model object for passing data to the view

\*

@param principal the Principal object representing the current authenticated user

\* @return the name of the view to be rendered

\*/

@GetMapping("/user/reviews")

public String viewUserReviews(Model model, Principal principal) {

String username = principal.getName();

Long id = userService.getUserIdByUsername(username);

List<Review> userReviews = reviewService.getReviewsByUserId(id);

model.addAttribute("userReviews", userReviews);

return "userReviews";

}

/\*\*

\* Handles requests to the "/user/reviews/{reviewId}/edit" path.

\*

\* @param reviewId the ID of the review to be edited

\* @param model the Model object for passing data to the view

\* @return the name of the view to be rendered

\*/

@GetMapping("/user/reviews/{reviewId}/edit")

public String editReviewForm(@PathVariable Long reviewId, Model model, Principal principal) {

Review review = reviewService.getReviewById(reviewId);

if (review != null && review.getUser().getUsername().equals(principal.getName())) {

model.addAttribute("review", review);

return "editReview";

} else {

return "redirect:/user/reviews";

}

}

/\*\*

\* Handles requests to edit a review.

\*

\* @param editedReview the Review object containing the edited review data

\* @param reviewId the ID of the review to be edited

\* @return the redirect view name after editing the review

\*/

@PostMapping("/user/reviews/{reviewId}/edit")

public String editReview(@ModelAttribute("review") Review editedReview, @PathVariable("reviewId") Long reviewId) {

Review review = reviewService.getReviewById(reviewId);

if (review != null) {

review.setComment(editedReview.getComment());

reviewService.updateReview(reviewId, review);

}

return "redirect:/user/reviews";

}

/\*\*

\* Handles requests to the "/user/reviews/{reviewId}/delete" path.

\*

\* @param reviewId the ID of the review to be deleted

\* @param model the Model object for passing data to the view

\* @return the name of the view to be rendered

\*/

@GetMapping("/user/reviews/{reviewId}/delete")

public String deleteReviewForm(@PathVariable Long reviewId, Model model) {

Review review = reviewService.getReviewById(reviewId);

model.addAttribute("review", review);

return "deleteReview";

}

/\*\*

\* Handles requests to delete a review.

\*

\* @param reviewId the ID of the review to be deleted

\* @return the redirect view name after deleting the review

\*/

@PostMapping("/user/reviews/{reviewId}/delete")

public String deleteReview(@PathVariable("reviewId") Long reviewId) {

reviewService.deleteReview(reviewId);

return "redirect:/user/reviews"; // Redirect to the users page after deleting

}

}

### Класс TicketController

/\*\*

\* Controller class for handling ticket-related requests.

\*

\* <p>

\* This class is responsible for handling requests related to tickets, such as viewing tickets and buying tickets.

\* </p>

\*

\* <p>

\* <strong>Author:</strong> Vadim

\* <br>

\* <strong>Email:</strong> blinvadik@mail.ru

\* </p>

\*

\* @see org.springframework.stereotype.Controller

\* @see org.springframework.beans.factory.annotation.Autowired

\* @see org.springframework.security.core.Authentication

\* @see org.springframework.ui.Model

\* @see org.springframework.web.bind.annotation.GetMapping

\* @see org.springframework.web.bind.annotation.PathVariable

\* @see ru.rsreu.lutikov.sber.domain.Event

\* @see ru.rsreu.lutikov.sber.domain.Ticket

\* @see ru.rsreu.lutikov.sber.domain.User

\* @see ru.rsreu.lutikov.sber.services.EventService

\* @see ru.rsreu.lutikov.sber.services.TicketService

\* @see ru.rsreu.lutikov.sber.services.UserService

\* @since 2023

\*/

package ru.rsreu.lutikov.sber.controllers;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.Authentication;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import ru.rsreu.lutikov.sber.domain.Event;

import ru.rsreu.lutikov.sber.domain.Ticket;

import ru.rsreu.lutikov.sber.domain.User;

import ru.rsreu.lutikov.sber.services.EventService;

import ru.rsreu.lutikov.sber.services.TicketService;

import ru.rsreu.lutikov.sber.services.UserService;

import java.security.Principal;

import java.util.Collections;

import java.util.List;

@Controller

public class NewTicketController {

@Autowired

private TicketService ticketService;

@Autowired

private UserService userService;

@Autowired

private EventService eventService;

/\*\*

\* Handles requests to the "/tickets" path.

\*

\* @param model the Model object for passing data to the view

\* @param authentication the Authentication object representing the current authentication

\* @return the name of the view to be rendered

\*/

@GetMapping("/tickets")

public String tickets(Model model, Authentication authentication) {

List<Ticket> yourDataList;

if (authentication != null && authentication.getAuthorities().stream()

.anyMatch(a -> a.getAuthority().equals("ROLE\_ADMIN"))) {

// If the user is an admin, retrieve all tickets

yourDataList = ticketService.getAllTickets();

} else if (authentication != null && authentication.getAuthorities().stream()

.anyMatch(a -> a.getAuthority().equals("ROLE\_USER"))) {

// If the user is a regular user, retrieve only their own tickets

String user = authentication.getName();

Long userId = userService.getUserIdByUsername(user);

System.out.println(userId);

yourDataList = ticketService.getTicketsByUserId(userId);

} else {

// If the user is not authenticated, handle this situation as desired

yourDataList = Collections.emptyList();

}

model.addAttribute("yourDataList", yourDataList); // Pass data to the view model

return "tickets"; // Return the name of the view to be rendered

}

/\*\*

\* Handles requests to buy a ticket for a specific event.

\*

\* @param eventId the ID of the event to buy a ticket for

\* @param principal the Principal object representing the current authenticated user

\* @return the redirect view name after buying the ticket

\*/

@GetMapping("/tickets/{eventId}/buy")

public String buyTicket(@PathVariable("eventId") Long eventId, Principal principal) {

// Get the name of the authenticated user from Principal

String username = principal.getName();

// Get the user ID by their username

Long userId = userService.getUserIdByUsername(username);

User user = userService.getUserById(userId);

Event event = eventService.getEventById(eventId);

Ticket ticket = new Ticket(user, event);

ticketService.createTicket(ticket);

// Redirect the user to the tickets page or another page

return "redirect:/tickets";

}

}

### Класс UserController

/\*\*

\* Controller class for handling user-related requests.

\*

\* <p>

\* This class is responsible for handling requests related to users, such as viewing users, creating users, editing user profiles, and changing passwords.

\* </p>

\*

\* <p>

\* <strong>Author:</strong> Vadim

\* <br>

\* <strong>Email:</strong> blinvadik@mail.ru

\* </p>

\*

\* @see org.springframework.stereotype.Controller

\* @see org.springframework.beans.factory.annotation.Autowired

\* @see org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder

\* @see org.springframework.security.crypto.password.PasswordEncoder

\* @see org.springframework.ui.Model

\* @see org.springframework.validation.BindingResult

\* @see org.springframework.web.bind.annotation.GetMapping

\* @see org.springframework.web.bind.annotation.ModelAttribute

\* @see org.springframework.web.bind.annotation.PathVariable

\* @see org.springframework.web.bind.annotation.PostMapping

\* @see ru.rsreu.lutikov.sber.domain.Event

\* @see ru.rsreu.lutikov.sber.domain.User

\* @see ru.rsreu.lutikov.sber.services.UserService

\* @see java.security.Principal

\* @see java.util.List

\* @since 2023-07-04

\*/

package ru.rsreu.lutikov.sber.controllers;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import ru.rsreu.lutikov.sber.domain.User;

import ru.rsreu.lutikov.sber.services.UserService;

import java.security.Principal;

import java.util.List;

@Controller

public class NewUserController {

@Autowired

UserService userService;

private PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

/\*\*

\* Handles requests to the "/users" path.

\*

\* @param model the Model object for passing data to the view

\* @return the name of the view to be rendered

\*/

@GetMapping("/users")

public String users(Model model) {

List<User> yourDataList = userService.getAllUsers(); // Retrieve JSON data

model.addAttribute("yourDataList", yourDataList); // Pass data to the view model

return "users"; // Return the name of the view to be rendered

}

/\*\*

\* Handles requests to create a new user.

\*

\* @param model the Model object for passing data to the view

\* @return the name of the view to be rendered

\*/

@GetMapping("/users/new")

public String createUserForm(Model model) {

model.addAttribute("user", new User());

return "newUser";

}

/\*\*

\* Handles requests to create a new user.

\*

\* @param user the User object containing the user data

\* @param result the BindingResult object for validation result

\* @return the redirect view name after creating the user

\*/

@PostMapping("/users/new")

public String createUser(@ModelAttribute("user") User user, BindingResult result) {

if (result.hasErrors()) {

return "newUser";

}

String encodedPassword = passwordEncoder.encode(user.getPassword());

user.setPassword(encodedPassword);

userService.createUser(user);

return "redirect:/users";

}

/\*\*

\* Handles requests to edit a user.

\*

\* @param userId the ID of the user to edit

\* @param model the Model object for passing data to the view

\* @return the name of the view to be rendered

\*/

@GetMapping("/users/{userId}/edit")

public String editUserForm(@PathVariable("userId") Long userId, Model model) {

User user = userService.getUserById(userId);

model.addAttribute("user", user);

return "editUser";

}

/\*\*

\* Handles requests to edit a user.

\*

\* @param userId the ID of the user to edit

\* @param updatedUser the User object containing the updated user data

\* @return the redirect

view name after updating the user

\*/

@PostMapping("/users/{userId}/edit")

public String updateUser(@PathVariable("userId") Long userId, @ModelAttribute User updatedUser) {

// Update the user with the specified userId using the UserService

String password = userService.getUserById(userId).getPassword();

updatedUser.setPassword(password);

userService.updateUser(userId, updatedUser);

return "redirect:/users";

}

/\*\*

\* Handles requests to delete a user.

\*

\* @param userId the ID of the user to delete

\* @param model the Model object for passing data to the view

\* @return the name of the view to be rendered

\*/

@GetMapping("/users/{userId}/delete")

public String deleteUserForm(@PathVariable Long userId, Model model) {

User user = userService.getUserById(userId);

model.addAttribute("user", user);

return "deleteUser";

}

/\*\*

\* Handles requests to delete a user.

\*

\* @param userId the ID of the user to delete

\* @return the redirect view name after deleting the user

\*/

@PostMapping("/users/{userId}/delete")

public String deleteUser(@PathVariable("userId") Long userId) {

userService.deleteUser(userId);

return "redirect:/users"; // Redirect to the users page after deletion

}

/\*\*

\* Handles requests to view the user profile.

\*

\* @param model the Model object for passing data to the view

\* @param principal the Principal object representing the current authenticated user

\* @return the name of the view to be rendered

\*/

@GetMapping("/user/profile")

public String userProfile(Model model, Principal principal) {

String username = principal.getName();

Long userId = userService.getUserIdByUsername(username);

User user = userService.getUserById(userId);

model.addAttribute("user", user);

return "userProfile";

}

/\*\*

\* Handles requests to edit the user profile.

\*

\* @param model the Model object for passing data to the view

\* @param principal the Principal object representing the current authenticated user

\* @return the name of the view to be rendered

\*/

@GetMapping("/user/profile/edit")

public String editUserProfileForm(Model model, Principal principal) {

String username = principal.getName();

Long userId = userService.getUserIdByUsername(username);

User user = userService.getUserById(userId);

model.addAttribute("user", user);

return "editUserProfile";

}

/\*\*

\* Handles requests to edit the user profile.

\*

\* @param editedUser the User object containing the edited user data

\* @param principal the Principal object representing the current authenticated user

\* @return the redirect view name after updating the user profile

\*/

@PostMapping("/user/profile/edit")

public String editUserProfile(@ModelAttribute("user") User editedUser, Principal principal) {

String username = principal.getName();

Long userId = userService.getUserIdByUsername(username);

User user = userService.getUserById(userId);

user.setEmail(editedUser.getEmail());

userService.updateUser(userId, user);

return "redirect:/user/profile";

}

/\*\*

\* Handles requests to change the user password.

\*

\* @param model the Model object for passing data to the view

\* @param principal the Principal object representing the current authenticated user

\* @return the name of the view to be rendered

\*/

@GetMapping("/user/profile/password")

public String changeUserPasswordForm(Model model, Principal principal) {

String username = principal.getName();

Long userId = userService.getUserIdByUsername(username);

User user = userService.getUserById(userId

);

model.addAttribute("user", user);

return "changePassword";

}

/\*\*

\* Handles requests to change the user password.

\*

\* @param editedUser the User object containing the edited user data

\* @param principal the Principal object representing the current authenticated user

\* @return the redirect view name after changing the password

\*/

@PostMapping("/user/profile/password")

public String changePassword(@ModelAttribute("user") User editedUser, Principal principal) {

String username = principal.getName();

Long userId = userService.getUserIdByUsername(username);

User user = userService.getUserById(userId);

String encodedPassword = passwordEncoder.encode(editedUser.getPassword());

user.setPassword(encodedPassword);

userService.updateUser(userId, user);

return "redirect:/user/profile";

}

}

### Класс RegistrationController

/\*\*

\* Controller class for handling user registration.

\*

\* <p>

\* This class is responsible for handling requests related to user registration, such as displaying the registration form

\* and adding a new user to the system.

\* </p>

\*

\* <p>

\* <strong>Author:</strong> Vadim

\* <br>

\* <strong>Email:</strong> blinvadik@mail.ru

\* </p>

\*

\* @see org.springframework.stereotype.Controller

\* @see org.springframework.beans.factory.annotation.Autowired

\* @see org.springframework.web.bind.annotation.GetMapping

\* @see org.springframework.web.bind.annotation.PostMapping

\* @see ru.rsreu.lutikov.sber.domain.Role

\* @see ru.rsreu.lutikov.sber.domain.User

\* @see ru.rsreu.lutikov.sber.repositories.UserRepository

\* @see org.springframework.security.crypto.password.PasswordEncoder

\* @see org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder

\* @since 2023

\*/

package ru.rsreu.lutikov.sber.controllers;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import ru.rsreu.lutikov.sber.domain.Role;

import ru.rsreu.lutikov.sber.domain.User;

import ru.rsreu.lutikov.sber.repositories.UserRepository;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import java.util.Collections;

import java.util.Map;

@Controller

public class RegistrationController {

@Autowired

private UserRepository userRepository;

private PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

/\*\*

\* Handles requests to the "/registration" path.

\*

\* @return the name of the view to be rendered

\*/

@GetMapping("/registration")

public String registration() {

return "registration";

}

/\*\*

\* Handles requests to add a new user.

\*

\* @param user the User object representing the new user

\* @param model the Map object for passing data to the view

\* @return the redirect view name after adding the user

\*/

@PostMapping("/registration")

public String addUser(User user, Map<String, Object> model) {

User userFromDb = userRepository.findByUsername(user.getUsername());

if (userFromDb != null) {

model.put("message", "User exists!");

return "registration";

}

user.setActive(true);

user.setRole(Collections.singleton(Role.USER));

String encodedPassword = passwordEncoder.encode(user.getPassword());

user.setPassword(encodedPassword);

userRepository.save(user);

return "redirect:/login";

}

}

## Приложение 3. HTML-страницы

### Страница changePassword

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<title>Мой город | Редактирование пароля</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<div class="form-container">

<h2>Редактирование пароля</h2>

<form th:object="${user}" th:action="@{/user/profile/password}" method="post">

<div>

<label>Пароль:</label>

<br>

<input type="password" th:field="\*{password}">

<span th:if="${#fields.hasErrors('password')}" th:errors="\*{password}"></span>

</div>

<div>

<input type="submit" value="Сохранить"/>

</div>

</form>

</div>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница deleteEvent

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<title>Мой город | Удаление мероприятия</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<form th:action="@{/events/{eventId}/delete(eventId=${eventId})}" method="post" class="delete-form">

<input type="hidden" name="\_method" value="DELETE">

<button type="submit" class="edit-button">Удалить</button>

</form>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница deleteReview

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<title>Мой город | Удаление отзыва</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<form th:action="@{/user/reviews/{reviewId}/delete(reviewId=${reviewId})}" method="post" class="delete-form">

<input type="hidden" name="\_method" value="DELETE">

<button type="submit" class="edit-button">Удалить</button>

</form>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница deleteUser

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<title>Мой город | Удаление пользователя</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<form th:action="@{/users/{userId}/delete(userId=${userId})}" method="post" class="delete-form">

<input type="hidden" name="\_method" value="DELETE">

<button type="submit" class="edit-button">Удалить</button>

</form>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница editEvent

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<title>Мой город | Редактирование мероприятия</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<form th:object="${event}" th:action="@{/events/{eventId}/edit(eventId=${eventId})}" method="post">

<input type="hidden" th:field="\*{id}"/>

<div>

<label>Event Name:</label>

<input type="text" th:field="\*{eventName}"/>

<span th:if="${#fields.hasErrors('eventName')}" th:errors="\*{eventName}"></span>

</div>

<div>

<label>Event Description:</label>

<textarea th:field="\*{eventDescription}"></textarea>

<span th:if="${#fields.hasErrors('eventDescription')}" th:errors="\*{eventDescription}"></span>

</div>

<div>

<label>Event Date and Time:</label>

<input type="datetime-local" th:field="\*{eventDateTime}"/>

<span th:if="${#fields.hasErrors('eventDateTime')}" th:errors="\*{eventDateTime}"></span>

</div>

<div>

<input type="submit" value="Save"/>

</div>

</form>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница editReview

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<title>Мой город | Редактирование отзыва</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<div class="form-container">

<h2>Редактирование отзыва</h2>

<form th:object="${review}" th:action="@{/user/reviews/{reviewId}/edit(reviewId=${review.id})}" method="post">

<div>

<label>Отзыв:</label>

<br>

<textarea th:field="\*{comment}"></textarea>

<span th:if="${#fields.hasErrors('comment')}" th:errors="\*{comment}"></span>

</div>

<div>

<input type="submit" value="Сохранить"/>

</div>

</form>

</div>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница editUser

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Мой город | Редактирование пользователя</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<form th:action="@{'/users/' + ${userId} + '/edit'}" method="post">

<input type="hidden" name="id" th:value="${user.id}"/>

<div>

<label>Имя пользователя:</label>

<input type="text" name="username" th:value="${user.username}" required/>

</div>

<!-- <div>-->

<!-- <label>Пароль:</label>-->

<!-- <input type="password" name="password" required/>-->

<!-- </div>-->

<div>

<label>Email:</label>

<input type="email" name="email" th:value="${user.email}" required/>

</div>

<div>

<label>Роль:</label>

<select name="role" required>

<option value="" disabled selected>Выберите роль</option>

<option value="ADMIN" th:selected="${user.role == 'ADMIN'}">Администратор</option>

<option value="USER" th:selected="${user.role == 'USER'}">Пользователь</option>

</select>

</div>

<div>

<label>Статус активации:</label>

<select name="active" required>

<option value="true" th:selected="${user.active}">Активен</option>

<option value="false" th:selected="${!user.active}">Неактивен</option>

</select>

</div>

<div>

<input type="submit" value="Сохранить"/>

</div>

</form>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница editUserProfile

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<title>Мой город | Редактирование профиля</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<div class="form-container">

<h2>Редактирование профиля</h2>

<form th:object="${user}" th:action="@{/user/profile/edit}" method="post">

<div>

<label>Email:</label>

<br>

<input type="email" th:field="\*{email}">

<span th:if="${#fields.hasErrors('email')}" th:errors="\*{email}"></span>

</div>

<div>

<input type="submit" value="Сохранить"/>

</div>

</form>

</div>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница events

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<title>Мой город | Мероприятия</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a

th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a>

</li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a>

</li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a>

</li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<form class="search-form" th:action="@{/events}" method="get">

<input type="text" name="search" placeholder="Поиск мероприятий">

<button type="submit">Искать</button>

</form>

<div th:if="${#request.isUserInRole('ADMIN')}">

<a class="edit-button" th:href="@{'/events/new'}">Создать</a>

</div>

<br>

<div class="card-container">

<!-- Генерация карточек на основе данных JSON -->

<div th:each="yourData : ${yourDataList}" class="card">

<h3 th:text="${yourData.eventName}"></h3>

<p th:text="${yourData.eventDescription}"></p>

<!-- <p th:text="${yourData.eventDateTime}"></p>-->

<p th:text="${#temporals.format(yourData.eventDateTime, 'dd.MM.yyyy')}"></p>

<p th:text="${#temporals.format(yourData.eventDateTime, 'HH:mm')}"></p>

<div th:if="${#request.isUserInRole('ADMIN')}">

<a class="edit-button" th:href="@{'/events/' + ${yourData.id} + '/edit'}">Редактировать</a>

<a class="edit-button" th:href="@{'/events/' + ${yourData.id} + '/delete'}">Удалить</a>

</div>

<div th:if="${#request.isUserInRole('USER')}">

<a class="edit-button" th:href="@{'/tickets/' + ${yourData.id} + '/buy'}">Купить билет</a>

<a class="edit-button" th:href="@{'/reviews/' + ${yourData.id}}">Отзывы</a>

</div>

<div th:if="${#request.isUserInRole('USER')}">

<br>

<a class="edit-button" th:href="@{'/reviews/' + ${yourData.id} + '/create'}">Написать отзыв</a>

</div>

</div>

</div>

<div class="pagination">

<ul>

<li th:if="${currentPage > 1}">

<a th:href="@{/events(page=${currentPage - 1})}">&laquo; Предыдущая</a>

</li>

<li th:each="pageNumber : ${#numbers.sequence(1, totalPages)}"

th:class="${pageNumber == currentPage} ? 'active' : ''">

<a th:href="@{/events(page=${pageNumber})}" th:text="${pageNumber}"></a>

</li>

<li th:if="${currentPage < totalPages}">

<a th:href="@{/events(page=${currentPage + 1})}">Следующая &raquo;</a>

</li>

</ul>

</div>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница home

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="utf-8">

<title>Мой город</title>

<link rel="stylesheet" href="style.css">

<link rel="icon" href="favicon.ico">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<section>

<h1>Добро пожаловать на главную страницу проекта "Мой город"!</h1>

<p>Здесь вы можете узнать о городских мероприятиях.</p>

</section>

<section>

<h2>Контакты</h2>

<ul>

<li><a href="mailto:mail@mail.ru">mail@mail.ru</a></li>

<li><a href="https://test">Тест</a></li>

</ul>

</section>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница login

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org">

<head>

<link rel="stylesheet" href="style.css">

<title>Мой город | Авторизация</title>

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<div th:if="${param.error}" align="center">

Некорректный логин или пароль!

</div>

<div th:if="${param.logout}" align="center">

Успешный выход!

</div>

<form th:action="@{/login}" method="post">

<div><label> Логин : <input type="text" name="username"/> </label></div>

<br>

<div><label> Пароль: <input type="password" name="password"/> </label></div>

<br>

<div>

<input type="submit" value="Вход"/>

<input type="submit" value="Регистрация" formaction="/registration"/>

</div>

</form>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница newEvent

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<title>Мой город | Создание мероприятия</title>

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<form th:object="${event}" th:action="@{/events/new}" method="post">

<div class="form-container-edit">

<div>

<label>Название:</label>

<input type="text" th:field="\*{eventName}"/>

<span th:if="${#fields.hasErrors('eventName')}" th:errors="\*{eventName}"></span>

</div>

<div>

<label>Описание:</label>

<textarea th:field="\*{eventDescription}"></textarea>

<span th:if="${#fields.hasErrors('eventDescription')}" th:errors="\*{eventDescription}"></span>

</div>

<div>

<label>Дата и врeмя:</label>

<input type="datetime-local" th:field="\*{eventDateTime}"/>

<span th:if="${#fields.hasErrors('eventDateTime')}" th:errors="\*{eventDateTime}"></span>

</div>

<div>

<input type="submit" value="Создать"/>

</div>

</div>

</form>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница newReview

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Мой город | Написание отзыва</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<form th:object="${review}" th:action="@{'/reviews/' + ${eventId} + '/create'}" method="post">

<input type="hidden" th:field="\*{event.id}"/> <div class="form-container-edit">

<div>

<label>Отзыв:</label>

<br>

<textarea th:field="\*{comment}"></textarea>

<span th:if="${#fields.hasErrors('comment')}" th:errors="\*{comment}"></span>

</div>

<div>

<input type="submit" value="Отправить"/>

</div>

</div>

</form>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница newUser

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/html">

<head>

<meta charset="UTF-8">

<title>Мой город | Добавление пользователя</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<form th:object="${user}" th:action="@{/users/new}" method="post">

<div>

<label>Имя пользователя:</label>

<input type="text" name="username" required/>

</div>

<div>

<label>Пароль:</label>

<input type="password" name="password" required/>

</div>

<div>

<label>Email:</label>

<input type="email" name="email" required/>

</div>

<div>

<label>Роль:</label>

<select name="role" required>

<option value="" disabled selected>Выберите роль</option>

<option value="ADMIN">Администратор</option>

<option value="USER">Пользователь</option>

</select>

</div>

<div>

<label>Статус активации:</label>

<select name="active" required>

<option value="true">Активен</option>

<option value="false">Неактивен</option>

</select>

</div>

<div>

<input type="submit" value="Добавить"/>

</div>

</form>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница registration

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<link rel="stylesheet" href="style.css">

<title>Мой город | Регистрация</title>

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<div>

<form th:action="@{/registration}" method="post">

<br>

<br>

<br>

<div><label> Логин : <input type="text" name="username"/> </label></div>

<br>

<div><label> Пароль: <input type="password" name="password"/> </label></div>

<br>

<div>

<div class="button-container">

<input type="submit" value="Зарегистрироваться"/>

</div>

<br><br>

Уже зарегистрированы? <a href="/login"> Войти </a>

</div>

</form>

</div>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница reviews

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Мой город | Отзывы</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<div class="card-container">

<!-- Генерация карточек на основе данных JSON -->

<div th:each="yourData : ${yourDataList}" class="card">

<h3 th:text="${yourData.user.getUsername()}"></h3>

<p th:text="${yourData.comment}"></p>

</div>

</div>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница tickets

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<title>Мой город | Билеты</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<div class="card-container">

<!-- Генерация карточек на основе данных JSON -->

<div th:each="yourData : ${yourDataList}" class="card">

<h3 th:text="${yourData.id}"></h3>

<p th:text="${yourData.user.getUsername()}"></p>

<p th:text="${yourData.event.getEventName()}"></p>

</div>

</div>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница userProfile

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<title>Мой город | Личный кабинет</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<div class="profile-container">

<h2>Личный кабинет</h2>

<div>

<p><strong>Имя пользователя:</strong> <span th:text="${user.username}"></span></p>

<p><strong>Email:</strong> <span th:text="${user.getEmail()}"></span></p></div>

<a class="edit-button" href="/user/profile/edit">Редактировать данные</a>

<a class="edit-button" href="/user/profile/password">Редактировать пароль</a>

</div>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница userReviews

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<title>Мой город | Отзывы</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" th:href="@{/style.css}" href="../style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<div class="card-container">

<!-- Генерация карточек на основе данных JSON -->

<div th:each="review : ${userReviews}" class="card">

<h3 th:text="${review.event.eventName}"></h3>

<p th:text="${review.comment}"></p>

<a class="edit-button" th:href="@{'/user/reviews/' + ${review.id} + '/edit'}">Редактировать</a>

<a class="edit-button" th:href="@{'/user/reviews/' + ${review.id} + '/delete'}">Удалить</a>

</div>

</div>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>

### Страница users

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:th="http://www.thymeleaf.org">

<head>

<title>Мой город | Пользователи</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<nav>

<ul>

<li><a href="/">Главная</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a th:href="@{/logout}">Выход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/login">Вход</a></li>

<li th:unless="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/registration">Регистрация</a></li>

<li><a href="/events">Мероприятия</a></li>

<li th:if="${#request.isUserInRole('ADMIN') || #request.isUserInRole('USER')}"><a href="/tickets">Билеты</a></li>

<li th:if="${#request.isUserInRole('ADMIN')}"><a href="/users">Пользователи</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/reviews">Отзывы</a></li>

<li th:if="${#request.isUserInRole('USER')}"><a href="/user/profile">Личный кабинет</a></li>

</ul>

</nav>

</header>

<main>

<div th:if="${#request.isUserInRole('ADMIN')}">

<a class="edit-button" th:href="@{'/users/new'}">Добавить</a>

</div>

<br>

<div class="card-container">

<!-- Генерация карточек на основе данных JSON -->

<div th:each="yourData : ${yourDataList}" class="card">

<h3 th:text="${yourData.username}"></h3>

<p th:text="${yourData.email}"></p>

<p th:text="${yourData.active}"></p>

<p th:text="${yourData.role}"></p>

<div th:if="${#request.isUserInRole('ADMIN')}">

<a class="edit-button" th:href="@{'/users/' + ${yourData.id} + '/edit'}">Редактировать</a>

<a class="edit-button" th:href="@{'/users/' + ${yourData.id} + '/delete'}">Удалить</a>

</div>

</div>

</div>

</main>

<footer>

<p>&copy; 2023 Мой город</p>

</footer>

</body>

</html>