Содержание

[Аннотация 4](#_Toc137681535)

[Введение 5](#_Toc137681536)

[1 Описание предметной области 7](#_Toc137681537)

[1.1 Обзор литературы 7](#_Toc137681538)

[1.2 Постановка задачи 8](#_Toc137681539)

[1.3 Обоснование актуальности задачи 8](#_Toc137681540)

[1.4 Обоснование используемых технологий, принципов, методик, решения задачи 9](#_Toc137681541)

[2 Проектирование архитектуры программно-информационной системы 13](#_Toc137681542)

[2.1 Установление требований 13](#_Toc137681543)

[2.2 Составление спецификаций требование 20](#_Toc137681544)

[2.3 Проектирование структуры базы данных и слоя доступа к данным 30](#_Toc137681545)

[3 Реализация программно-информационной системы 35](#_Toc137681546)

[3.1 Описание архитектуры программно-информационной системы 35](#_Toc137681547)

[3.2 Пример использования программно-информационной системы 39](#_Toc137681548)

[4 Результаты реализации программно-информационной системы 39](#_Toc137681549)

[Заключение 50](#_Toc137681550)

[Список использованных источников 51](#_Toc137681551)

[Приложение А (справночное). Код программы 52](#_Toc137681552)

# Аннотация

**на выпускную квалификационную работу**

Разработка АСОИ организации автомобильных и мотоциклетных соревнований

Выпускная квалификационная работа состоит из 6 листов графической части и текстовой части на 70 страницах и 1 приложения. Пояснительная записка состоит из введения, 4 глав, заключения, списка используемых источников и приложения. В пояснительной записке находится 27 рисунков, 22 таблиц. В списке литературы содержится 12 литературных источников.

В главе «Проектирование архитектуры программно-информационной системы» описывается проектирование архитектуры проекта и структуры базы данных. Для отображения возможностей приложения используются диаграмма вариантов использования, таблицы определения вариантов использования и диаграммы последовательности, разработанные в Enterprise Architect.

В главе «Реализация программно-информационной системы» описывается реализация программно-информационной системы, приводится описание интерфейса данного приложения и взаимодействия с ним пользователя.

В главе «Результаты реализации программно-информационной системы» описываются результаты работы и процесс тестирования приложения для обеспечения его корректной работы.

# Введение

Развитие автомобильного или мотоциклетного спорта непосредственно связанно с появлением большого количества автомобилей и мотоциклов на улицах городов Соединенных Штатов Америки в 1930-х годах. Первые автолюбители и мотолюбители того времени соревновались в своих инженерных способностях, а наиболее эффективным способом показать возможности своего творения – это проведения заездов из точки А в точку Б. Но в связи с тем, что автомобильный и мотоциклетный спорт еще не был развит, то заезды происходили по улицам города, что было очень опасным.

Для решения этой проблемы создавались первые автомобильные и мотоциклетные клубы, такие как: Автомобильная Ассоциация Америки (ААА) – автомобильный спорт и Американская Мотоциклетная Ассоциация (АМА) – мотоциклетный спорт. Эти клубы начали проводить свои соревнования на высушенных солевых озерах штата Ютан, на бывших ипподромах и на песчаных пляжах по всей стране. Автомобильные и мотоциклетные клубы начали появляться по всему миру.

С появлением автомобильных и мотоциклетных клубов, автомобильный и мотоциклетный спорт начал становиться одним из популярных видов развлекательных событий. Сотни миллионов людей по всему миру погрузились в мир скорости и безумия. Смотря как любимый гонщик раз за разом выигрывал гонки, люди вдохновлялись и сопереживали гонщикам.

Со временем по всему миру начали появляться гоночные трассы, на которых соревнуются гонщики со всего мира, а посмотреть на соревнования приходят тысячи людей и миллионы смотрят со своих информационных устройств.

Данное приложение создано для того, чтобы упростить процесса поиска автомобильных и мотоциклетных соревнований. Разрабатываемая система будет реализовывать функции помощи при организации автомобильных или мотоциклетных соревнований.

Для реализации указанных задач предполагается разработка web-приложения, включающего клиентскую и серверную части. Главными задачами проекта являются формирование базы данных основных объектов и разработка «дружественного для пользователя» графического интерфейса.

В качестве ожидаемых результатов проекта следует выделить создание адаптированной для ОС web-приложения вёрстки, возможность корректно функционирующего поиска и выбора по категориям, просмотра событий, добавления комментариев и создания событий администратором.

Важным этапом проекта является тестирование системы с целью выявления ошибок, которые могут возникнуть на этапе разработки. Конечная цель проекта заключается в создании удобной и интуитивно понятной информационной системы, которая поможет пользователям быстро и легко найти интересующие их автомобильные и мотоциклетные соревнования.

1. **Описание предметной области**

**1.1 Обзор литературы**

В процессе реализации данного проекта были изучены различные технологии, включая такие как: Java, Spring Boot, Spring Data, JavaScript, React.js и другие.

Для успешного написания проекта были изучены следующие документации:

* https://www.baeldung.com/spring-boot – документация по Spring Boot;
* https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ – документация Spring Data JPA;
* https://h2database.com/html/tutorial.html –документация системы управления базами данных H2;
* https://javascript.ru/manual – справочник по языку программирования JavaScript;
* https://ru.reactjs.org/docs/getting-started.html – документация React.

Аналогам данной разрабатываемой системы являются следующий электронный ресурс:

− новости поехали onliner. [Электронный ресурс]. – Режим доступа: https://auto.onliner.by/poexali. Дата обращения: 03.06.2023;

Рассматриваемое web-приложений не корректно рассматривать как аналог разрабатываемого приложения, т.к. данные системы предназначены не только для автомобильного или мотоциклетного спорта, но и других. Также к нему присущи следующие недостатки:

* обширный информационный контент;
* перегруженность пользовательского интерфейса;
* невозможность комментировать событий;
* отсутствие рейтинговых таблиц;
* отсутствие панели администратора.

Для создания информационной системы было решено использовать передовые технологии разработки веб-приложений, которые постоянно улучшаются и обновляются. Для ознакомления с последними тенденциями и лучшими практиками web-разработки были использованы соответствующие информационные источники.

## 1.2 Постановка задачи

Задачей данного проекта является разработка программно-информационной системы, которое упростит поиск проводимых соревнований.

Web-приложение должно включать такие функции как:

* регистрация и аутентификация пользователя, которые позволяют обеспечить безопасность и функциональность web-приложения;
* просмотр событий, которые позволяют пользователю просмотреть зарегистрированные события, а также дату и место проведения соревнования;
* добавление комментария, позволяет пользователям обсуждать события, оставлять отзывы и делится впечатлениями;
* просмотр спонсоров, позволяет пользователю просмотреть контактную информацию спонсоров того или иного соревнования;
* поиск события, позволяет пользователю найти события по категории или конкретное событие.
* просмотр результата событий и турниров, позволяет пользователю просмотреть информацию о результатах событий или турниров.
* выбор типа событий, позволяет пользователю отфильтровать типы событий для удобства просмотра.

**1.3 Обоснование актуальности задачи**

Автомобильные и мотоциклетные соревнования являются одними из самых популярных спортивных мероприятий в мире. Однако, не всегда возможно узнать о всех проводимых соревнований в одном месте, приходится искать информацию по разным ресурсам. Информационная система, специализированная на мониторинге автомобильных и мотоциклетных соревнований, может быть очень полезной для любителей автомобильного мотоциклетного спорта, которые не хотят тратить свое время на поиск проводимых мероприятий.

В следствие чего, предлагаемый сервис будет полезен тем людям, которые пользуются услугами интернет-площадок, размещающих на своих страницах об проводимых соревнованиях. Так как пользователи будут мониторить автомобильные и мотоциклетные соревнования на одном ресурсе, они смогут экономить свое время при поиске того или иного мероприятия.

## 1.4 Обоснование используемых технологий, принципов, методик, решения задачи

Для проектирования функциональной части проекта был выбран язык программирования Java. Java является одной из наиболее популярных и широко используемых языков программирования в мире. Он обладает множеством преимуществ, таких как кроссплатформенность, высокая производительность, богатая стандартная библиотека и так далее. Кроме того, Java является объектно-ориентированным языком программирования, что позволяет разработчикам создавать более структурированный и модульный код.

Spring Boot – это фреймворк для создания веб-приложений на Java. Он предоставляет множество инструментов и библиотек, которые значительно упрощают процесс разработки и позволяют разработчикам быстро создавать высококачественные приложения. Spring Boot также обеспечивает высокую производительность и масштабируемость приложения.

Spring Data – это фреймворк, который облегчает работу с базами данных. Он предоставляет удобные и простые способы взаимодействия с различными типами баз данных, например, с реляционными и NoSQL базами данных, что позволяет разработчикам сосредоточиться на реализации бизнес-логики приложения, а не на работе с базами данных.

H2 Database – это легковесная встроенная база данных на Java, которая может использоваться в качестве временного хранилища данных для тестирования и разработки приложений. Она обеспечивает быструю и эффективную работу с данными, а также поддерживает SQL-стандарты.

JWT (JSON Web Tokens) – это открытый стандарт для создания токенов доступа, которые могут использоваться для аутентификации и авторизации пользователей. JWT обеспечивает безопасную передачу информации между клиентом и сервером, что делает его идеальным выбором для создания безопасных веб-приложений.

REST (Representational State Transfer) – это архитектурный стиль для создания веб-сервисов. Он определяет протоколы и правила для передачи данных между клиентом и сервером. REST является легковесным и гибким, что позволяет разработчикам создавать масштабируемые и расширяемые приложения.

Для разработки визуальной части проекта был выбран язык программирования JavaScript. JavaScript – это высокоуровневый, динамический язык программирования, который хорошо подходит для объектно-ориентированного и функционального стиля программирования.

В качестве среды выполнения JavaScript-кода был выбрана среда Node.js. Node.js был разработан компанией Google для своего браузера, чтобы запускать JavaScript-код на серверной части проекта. Также были использованы библиотеки JavaScript такие как: React.js, Axios, Bootstrap, Redux, React Router Dom.

React.js – это библиотека JavaScript, созданная компанией Facebook, которая позволяет разработчикам создавать компоненты, которые они могут использовать повторно в различных частях интерфейса. Такой подход по созданию web-приложений является наиболее эффективным и масштабируемым.

Axios – это библиотека JavaScript, основанная на Promise для Node.js, которая позволяет выполнять HTTP-запросы такие как GET, POST, PUT, DELETE и другие, а также дает возможность выявления и обработки ошибок, в том числе и ошибок сети.

Bootstrap – это фреймворк для создания web-приложений или web-сайтов. Он позволяет разработчикам использовать его для упрощения и ускорения в создании современного и отзывчивого интерфейса. Он предлагает широкий ассортимент готовых стилей и компонентов, таких как кнопки, формы, таблицы, модальные окна, навигационное меню и многое другое.

Bootstrap также дает возможность адаптивной верстки, то есть приложения написанные с помощью Bootstrap с легкостью адаптируются к различным устройствам.

Redux – это библиотека для управления состоянием в JavaScript, которая позволяет контролировать состояние приложения независимо от используемого фреймворка или библиотеки для создания пользовательского интерфейса.

Главная концепция Redux заключается в том, что все состояние приложения хранится в единственном объекте, называемом "store". Изменение состояния происходит только через действия (actions), которые являются объектами, описывающими изменение состояния. Действия передаются в "редьюсер" (reducer), который обрабатывает действия и возвращает новое состояние.

React Router DOM (Document Object Model) – это библиотека для навигации в React.js, которая позволяет разработчикам создавать маршруты и связанные с ними компоненты, которые отображаются при переходе по определенным маршрутам. Ключевыми компонентами React Router DOM являются:

* BrowserRouter: компонент, который используется для настройки маршрутизации в приложении;
* Switch: компонент, который используется для выбора первого подходящего маршрута из набора маршрутов;
* Route: компонент, который связывает маршрут с компонентом, который будет отображаться при переходе по этому маршруту;
* Link: компонент, который создает ссылку на другой маршрут в приложении;
* Redirect: компонент, который используется для перенаправления пользователя на другой маршрут;

Эти компоненты делают навигацию в React удобной и понятной для пользователя, что является одним из ключевых способов ускорения написания кода.

**2** **Проектирование архитектуры программно-информационной системы**

**2.1 Установление требований**

2.1.1 Предварительные замечания к проекту

2.1.1.1 Цели и рамки проекта. Целью данного проекта является разработка программно-информационной системы, которая будет позволять пользователям получать информацию о проводимых соревнованиях, о результатах проведенных соревнований или турниров, оставлять комментарии и осуществлять поиск событий, которые будут интересны им.

2.1.1.2 Деловой контекст. Многие любители автомобильного и мотоциклетного спорта ищут информацию о проводимых соревнованиях, а также месте и времени их проведения, и хотели бы оставлять комментарии о них. В зависимости от интересов пользователей, они могут быстро просмотреть информацию о проводимых соревнованиях в автомобильном или мотоциклетном спорте.

2.1.1.3 Участники проекта. Разработчик – Амбаров Олег Владиславович.

2.1.1.4 Идеи в отношении решений. Разработка web-приложения будет выполнена на языке программирования Java Spring REST с использованием СУБД H2 Database. Графическое представление будет реализовано с помощью технологий React.js, JavaScript и Redux.

2.1.1.5 Обзор документа. Раздел «Системные сервисы» описывает рамки работы web-приложения, функциональных требования и требование к данным.

Раздел «Системные ограничения» описывает требования к интерфейсу, производительности, безопасности, эксплуатации.

2.1.2 Системные сервисы

2.1.2.1 Рамки системы. Программно-информационная система «мониторинга автомобильных и мотоциклетных соревнований» получает параметры пользователей, необходимые для работы системы, от сущности «Авторизированный пользователь».

2.1.2.2 Функциональные требования. Диаграмма вариантов использования на рисунке 2.1 отображает функциональные требования к данной программно-информационной системе.

В таблице 2.1 предоставлено определение варианта использования «Регистрация»; в таблице 2.2 предоставлено определение варианта использования «Авторизация»; в таблице 2.3 предоставлено определение варианта использования «Мониторинг и модерация комментариев»; в таблице 2.4 предоставлено определение варианта использования «Управление информации о событиях»; в таблице 2.5 предоставлено определение варианта использования «Добавление модераторов»; в таблице 2.6 предоставлено определение варианта использования «Просмотр событий»; в таблице 2.7 предоставлено определение варианта использования «Добавление комментариев».



Рисунок 2.1 – Диаграмма вариантов использования

Таблица 2.1 – Определение варианта использования «Регистрация»

|  |  |
| --- | --- |
| Характеристика варианта использования | Описание |
| 1 | 2 |
| Идентификатор | UC-01 |
| Наименование | Регистрация |
| Первичное действующее лицо | Анонимный пользователь |
| Другие действующие лица | Нет |
| Алгоритм | Для получения доступа к возможностям веб-приложения пользователь должен зарегистрироваться в системе. |
| Предварительные условия | Приложение запущено, открыта страница регистрации. |
| Нормальный поток развития | UC-01 Регистрация  1) Пользователь вводит имя;  2) Пользователь вводит фамилию;  3) Пользователь вводит номер телефона;  4) Пользователь вводит пароль;  5) Пользователь нажимает кнопку «Зарегистрироваться»;  6) Система регистрирует данные пользователя. |
| Альтернативный поток развития | Нет |
| Исключения | Нет |
| Частота использования | При каждом заходе на несуществующий аккаунт |
| Постусловие | Данные пользователя сохранены в базе данных |

Таблица 2.2 – Определение варианта использования «Авторизация»

|  |  |
| --- | --- |
| Характеристика варианта использования | Описание |
| 1 | 2 |
| Идентификатор | UC-02 |
| Наименование | Авторизация |
| Первичное действующее лицо | Анонимный пользователь |
| Другие действующие лица | Нет |
| Алгоритм | Для получения доступа к возможностям веб-приложения пользователь должен зарегистрироваться в системе. |
| Предварительные условия | Приложение запущено, открыта страница авторизации. |
| Нормальный поток развития | UC-02 Авторизация  1) Пользователь вводит номер телефона;  2) Пользователь вводит пароль;  3) Пользователь нажимает кнопку «Войти»;  6) Система авторизует пользователя. |
| Исключительная ситуация | Нет |
| Частота использования | При каждом заходе на существующий аккаунт. |
| Постусловие | Данные пользователя взяты из базы данных. |

Таблица 2.3 – Определение варианта использования «Мониторинг и модерация комментариев»

|  |  |
| --- | --- |
| Характеристика варианта использования | Описание |
| 1 | 2 |
| Идентификатор | UC-03 |
| Наименование | Мониторинг и модерация комментариев |
| Первичное действующее лицо | Модератор |
| Другие действующие лица | Нет |
| Алгоритм | Модератор может осуществлять мониторинг и модерацию комментариев. |
| Предварительные условия | Приложение запущено, открыта страница события. |
| Нормальный поток развития | UC-03 Мониторинг и модерация комментариев  1) Модератор переходит в отдел комментариев;  2) Модератор просматривает комментарии;  3) Модератор удаляет комментарии по необходимости. |
| Исключительная ситуация | Нет |
| Частота использования | При необходимости просмотра или удаления комментариев. |
| Постусловие | Данные комментариев удаляются из базы данных. |

Таблица 2.4 – Определение варианта использования «Управление информации о событиях»

|  |  |
| --- | --- |
| Характеристика варианта использования | Описание |
| 1 | 2 |
| Идентификатор | UC-04 |
| Наименование | Управление информации о событиях |
| Первичное действующее лицо | Администратор |
| Другие действующие лица | Нет |
| Алгоритм | Администратор может управлять информацией о событиях, о спонсорах и о результатах. |
| Предварительные условия | Приложение запущено, пользователь авторизован как администратор, открыта страница администратора. |
| Нормальный поток развития | UC-04 Управление информации о событиях  1) Администратор заходит на страницу администратора;  2) Администратор добавляет новые соревнования;  3) Администратор добавляет спонсоров;  4) Администратор добавляет турниры. |
| Исключительная ситуация | Администратор может не добавлять спонсоров или не создавать турниры. |
| Частота использования | При организации новых соревнований, новых турниров или новых спонсоров. |
| Постусловие | Данные о соревнованиях, о спонсорах или о турнирах добавляются в базу данных. |

Таблица 2.5 – Определение варианта использования «Добавление модераторов»

|  |  |
| --- | --- |
| Характеристика варианта использования | Описание |
| 1 | 2 |
| Идентификатор | UC-05 |
| Наименование | Добавление модераторов |
| Первичное действующее лицо | Администратор |
| Другие действующие лица | Нет |
| Алгоритм | Администратор может назначать новых модераторов. |
| Предварительные условия | Приложение запущено, пользователь авторизован как администратор, открыта страница администратора. |
| Нормальный поток развития | UC-05 Добавление модераторов  1) Администратор заходит на страницу администратора;  2) Администратор просматривает список пользователей;  3) Администратор назначает пользователя модератором. |
| Исключительная ситуация | Нет |
| Частота использования | При добавлении новых модераторов. |
| Постусловие | У пользователя меняется роль в базе данных |

Таблица 2.6 – Определение варианта использования «Просмотр событий»

|  |  |
| --- | --- |
| Характеристика варианта использования | Описание |
| 1 | 2 |
| Идентификатор | UC-06 |
| Наименование | Просмотр событий |
| Первичное действующее лицо | Авторизованный пользователь |
| Другие действующие лица | Нет |
| Алгоритм | Пользователь может просматривать список событий. |
| Предварительные условия | Приложение запущено, открыта страница события. |
| Нормальный поток развития | UC-06 Просмотр событий  1) Пользователь просматривает события;  2) Пользователь выбирает тип соревнований;  3) Пользователь просматривает спонсоров;  4) Пользователь просматривает турниры;  5) Пользователь просматривает результаты соревнований;  6) Пользователь осуществляет поиск соревнований. |
| Исключительная ситуация | При поиске соревнований не найдено искомое. |
| Частота использования | При открытии страницы соревнований. |
| Постусловие | Осуществляется выборка из базы данных. |

Таблица 2.7 – Определение варианта использования «Добавление комментариев»

|  |  |
| --- | --- |
| Характеристика варианта использования | Описание |
| 1 | 2 |
| Идентификатор | UC-07 |
| Наименование | Добавление комментариев |
| Первичное действующее лицо | Авторизованный пользователь |
| Другие действующие лица | Администратор и модератор |
| Алгоритм | Пользователь может оставлять комментарии под событием. |
| Предварительные условия | Приложение запущено, открыта страница события. |
| Нормальный поток развития | UC-07 Добавление комментариев  1) Пользователь переходит на страницу нужного события;  2) Пользователь добавляет текст комментария;  3) Пользователь добавляет файл по необходимости;  4) Пользователь сохраняет комментарий;  5) Пользователь может просмотреть другие комментарии. |
| Исключительная ситуация | Пользователь не авторизован. |
| Частота использования | При открытии страницы соревнований. |
| Постусловие | Добавляется комментарий в базу данных. |

2.1.2.3 Требования к данным. Для хранения данных программно-информационная система должна использовать реляционную базу данных H2 Database.

2.1.3 Системные ограничения

2.1.3.1 Требования к интерфейсу. Для обеспечения удобства использования программно-информационной системы необходимо создать интуитивно понятный пользовательский интерфейс.

Для этого интерфейс должен включать следующие страницы: главная страница с описанием web-приложения, страницы регистрации и авторизации пользователей, страница навигации пользователя по приложению, страница просмотра списка соревнований, страница просмотра обсуждений, страница просмотра результатов, страница просмотра турниров и страница просмотра спонсоров.

2.1.3.2 Требования к производительности. Приложение должно обеспечивать доступ к системе нескольким пользователям одновременно.

2.1.3.3 Разделение пользователей на 4 группы – авторизованных: пользователь, администратор, модератор и неавторизованных.

2.1.3.4 Эксплуатационные требования. Для обеспечения стабильной работы приложения необходимо, чтобы оно функционировало корректно в современных браузерах (Google Chrome, Microsoft Edge, Yandex Browser и Mozilla Firefox). Минимальные аппаратные требования зависят от операционной системы, а также от минимальных системных требований для корректной работы браузера.

2.1.3.5 Политические и юридические требования. Приложение должно соответствовать действующему законодательству и обеспечивать защиту пользователей.

**2.2 Составление спецификаций требование**

Была разработана диаграмма состояний для предоставления работы web-приложения, а также переходов между страницами, иллюстрация представлена на рисунке 2.2.

Диаграмма последовательности «Регистрация» представлена на рисунке 2.3, а процесс выполнения данной функции описан в таблице 2.8.

Диаграмма последовательности «Навигация по домашней странице» предоставлена в графической части ВКР.09.03.01.100.30.722, а процесс выполнения данной функции описан в таблице 2.9.

Диаграмма последовательности «Отправка комментария» предоставлена на рисунке 2.5, а процесс выполнения данной функции описан в таблице 2.10.

Диаграмма последовательности «Добавление события» предоставлена на рисунке 2.6, а процесс выполнения данной функции описан в таблице 2.11.

Диаграмма последовательности «Добавление спонсора» предоставлена на рисунке 2.7, а процесс выполнения данной функции описан в таблице 2.12.

Диаграмма последовательности «Добавление турнира» предоставлена на рисунке 2.8, а процесс выполнения данной функции описан в таблице 2.13.

Диаграмма последовательности «Модерация комментариев» предоставлена на рисунке 2.8, а процесс выполнения данной функции описан в таблице 2.14.



Рисунок 2.3 – Диаграмма состояний



Рисунок 2.4 – Диаграмма последовательности «Регистрация»

Таблица 2.8 – Описание процесса выполнения функции «Регистрация»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Пользователь | Система | Экранная форма | Условие: последующий шаг |
| 1 | 2 | 3 | 4 | 5 |
| Предусловие: открыта страница «Регистрация». | | | | |
| 1 | Ввести имя |  | «Регистрация» |  |
| 2 | Ввести фамилию |  | «Регистрация» |  |
| 3 | Ввести номер телефона |  | «Регистрация» |  |
| 4 | Ввести пароль |  | «Регистрация» |  |
| 5 | Нажать кнопку «Зарегистрироваться» |  | «Регистрация» |  |
| 6 |  | Выполнение метода register(name, lastname, numberPhone, password) в контроллере UserController |  |  |
| 7 |  | Отправка данных в UserService и выполнение метода registration ( UserRegistrationDto userRegistrationDto ) |  |  |
| 8 |  | Отправка данных в UserRepository при помощи метода save(user) |  |  |
| Постусловие: сохранение данных о пользователе. | | | | |

Таблица 2.9 – Описание процесса выполнения функции «Навигации по домашней странице»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Пользователь | Система | Экранная форма | Условие: последующий шаг |
| 1 | 2 | 3 | 4 | 5 |
| Предусловие: открыта страница «Навигации по домашней странице». | | | | |
| 1 | Просмотреть события |  | «Навигация по домашней странице» |  |
| 2 |  | Вызов метода контроллера EventController getEvent() |  |  |
| 3 |  | Вызов метода контроллера EventService getEvent() |  |  |
| 4 |  | Вызов метода контроллера EventRepository getEvent() |  |  |
| 5 | Нажать кнопку «Поиск» |  | «Навигация по домашней странице» |  |
| 6 |  | Вызов метода контроллера EventController findByName() |  |  |
| 7 |  | Отправка запроса в метод EventService findByName() |  |  |
| 8 |  | Отправка запроса в метод EventRepository findByName() |  |  |
| 9 | Нажать кнопку «Спонсоры» |  | «Навигация по домашней странице» |  |
| 10 |  | get() |  |  |
| 11 | Нажать кнопку «Турниры» |  | «Навигация по домашней странице» |  |
| 12 |  | get() |  |  |
| 13 | Нажать кнопку «Результаты» |  | «Навигация по домашней странице» |  |
| 14 |  | get() |  |  |
| Постусловие: вывод результатов пользователю. | | | | |



Рисунок 2.5 – Диаграмма последовательности «Отправка комментария»

Таблица 2.10 – Описание процесса выполнения функции «Отправка комментария»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Пользователь | Система | Экранная форма | Условие: последующий шаг |
| 1 | 2 | 3 | 4 | 5 |
| Предусловие: открыта страница «Отправка комментария». | | | | |
| 1 | Переход на страницу события |  | Переход на страницу события |  |
| 2 |  | getByID() |  |  |
| 3 | Ввод текста комментария |  | Ввод текста комментария |  |
| 4 | Изменение комментария |  | Изменение комментария |  |
| 5 |  | Вызов метода контроллера CommentController addComment() |  |  |
| 6 |  | Отправка запроса в метод CommentService addComment() |  |  |
| 7 |  | Отправка запроса в метод CommentRepository addComment() |  |  |

Продолжение таблицы 2.10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 8 |  | Вызов метода контроллера CommentController editComment() |  |  |
| 9 |  | Отправка запроса в метод CommentService editComment() |  |  |
| 10 |  | Отправка запроса в метод CommentRepository editComment() |  |  |
| Постусловие: добавление и изменение комментария пользователем. | | | | |



Рисунок 2.6 – Диаграмма последовательности «Добавление события»

Таблица 2.11 – Описание процесса выполнения функции «Добавление события»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Пользователь | Система | Экранная форма | Условие: последующий шаг |
| 1 | 2 | 3 | 4 | 5 |
| Предусловие: открыта страница «Добавление события». | | | | |
| 1 | Переход на страницу администратора |  | «Панель администратора» |  |
| 2 | Выбор типа соревнования |  | «Добавить событие» |  |

Продолжение таблицы 2.11

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 3 | Ввод названия события |  | «Добавить событие» |  |
| 4 | Ввод описания события |  | «Добавить событие» |  |
| 5 | Ввод локации |  | «Добавить событие» |  |
| 6 | Выбор даты соревнования |  | «Добавить событие» |  |
| 7 | Добавление файла |  | «Добавить событие» |  |
| 8 | Нажатие кнопки «Добавить» |  | «Добавить событие» |  |
| 9 |  | Выполнение метода add(name, description, location, picture, date, type) в контроллере EventController |  |  |
| 10 |  | Отправка данных в EventService и выполнение метода add(name, description, location, picture, date, type) |  |  |
| 11 |  | Отправка данных в EventRepository при помощи метода save(event) |  |  |
| Постусловие: сохранение данных о соревновании. | | | | |



Рисунок 2.6 – Диаграмма последовательности «Добавление спонсора»

Таблица 2.12 – Описание процесса выполнения функции «Добавление спонсора»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Пользователь | Система | Экранная форма | Условие: последующий шаг |
| 1 | 2 | 3 | 4 | 5 |
| Предусловие: открыта страница «Добавление спонсора». | | | | |
| 1 | Переход на страницу администратора |  | «Панель администратора» |  |
| 2 | Ввод названия спонсора |  | «Добавить спонсора» |  |
| 3 | Ввод рекламной информации |  | «Добавить спонсора» |  |
| 4 | Ввод контактов |  | «Добавить спонсора» |  |
| 5 | Нажатие кнопки «Добавить» |  | «Добавить спонсора» |  |
| 6 |  | Выполнение метода add(name, advertisingInformation, contacts, logo) в контроллере SponsorController |  |  |
| 7 |  | Отправка данных в SponsorService и выполнение метода add(name, advertisingInformation, contacts, logo) |  |  |
| 8 |  | Отправка данных в SponsorRepository при помощи метода save(sponsor) |  |  |
| Постусловие: сохранение данных о спонсоре. | | | | |



Рисунок 2.7 – Диаграмма последовательности «Добавление турнира»

Таблица 2.13 – Описание процесса выполнения функции «Добавление турнира»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Пользователь | Система | Экранная форма | Условие: последующий шаг |
| 1 | 2 | 3 | 4 | 5 |
| Предусловие: открыта страница «Добавление турнира». | | | | |
| 1 | Переход на страницу администратора |  | «Панель администратора» |  |
| 2 | Ввод названия турнира |  | «Добавить турнир» |  |
| 3 | Нажатие кнопки «Добавить» |  | «Добавить турнир» |  |
| 4 |  | Выполнение метода add(name) в контроллере TournamentController |  |  |
| 5 |  | Отправка данных в TournamentService и выполнение метода add(name) |  |  |
| 6 |  | Отправка данных в TournamentRepository при помощи метода save(tournament) |  |  |
| Постусловие: сохранение данных о турнире. | | | | |



Рисунок 2.8 – Диаграмма последовательности «Модерация комментариев»

Таблица 2.14 – Описание процесса выполнения функции «Модерация комментариев»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Пользователь | Система | Экранная форма | Условие: последующий шаг |
| 1 | 2 | 3 | 4 | 5 |
| Предусловие: открыта страница «Модерация комментариев». | | | | |
| 1 | Открытие страницы событие |  | «Страница событие» |  |
| 2 | Нажатие кнопки «Удалить» |  | «Страница событие» |  |
| 3 |  | Выполнение метода delete(eventide, id) в контроллере CommentController |  |  |
| 4 |  | Отправка данных в CommentService и выполнение метода delete(eventide, id) |  |  |
| 5 |  | Отправка данных в CommentRepository при помощи метода save(comment) |  |  |
| Постусловие: удаление данных комментариев из базы данных. | | | | |

**2.3 Проектирование структуры базы данных и слоя доступа к данным**

Модель базы данных программно-информационной системы представлена на рисунке 2.9.

В базе данных были определены следующие сущности, а также связи между ними:

* «comments» – необходима для хранения комментариев добавленным пользователем;
* «users» – необходима для хранения информации о пользователей;
* «event» – необходима для хранения информации о событиях;
* «sponsors» – необходима для хранения информации о спонсорах;
* «types» – необходима для хранения информации о типах соревнований;
* «racers» – необходима для хранения информации о гонщиках;
* «event\_sponsor» – необходима для хранения информации о спонсорах в событии событий;
* «event\_racer» – необходима для хранения информации о гонщиках в событии событий;
* «tournaments» – необходима для хранения информации о турнирах.



Рисунок 2.9 – Модель базы данных

Связь между сущностями Users и Comments идентифицирующая, так как эти сущности не могут быть созданы без сущности Пользователи. Тип связи 1:М, т.к. один пользователь может иметь множество комментариев.

Связь между сущностями Comments и Events идентифицирующая, так как эти сущности не могут быть созданы без сущности Событие. Тип связи М:1, т.к. одно событие может иметь множество комментариев.

Связь между сущностями Events, Sponsors и Event\_sponsor не идентифицирующая, т.к. сущность Events может быть однозначно идентифицирована, если не задана сущность Sponsors. Тип связи М:М, т.к. одному событию может принадлежать несколько спонсоров, так и 1 спонсор может принадлежать многим событиям.

Связь между сущностями Событие и Тип идентифицирующая, т.к. сущность Тип не может быть однозначно идентифицирована, если не задана сущность Событие. Тип связи М:1, т.к. несколько событий может принадлежать одному типу.

Связь между сущностями Events, Racers и Event\_racer не идентифицирующая, т.к. сущность Events может быть однозначно идентифицирована, если не задана сущность Racers. Тип связи М:М, т.к. одному событию может принадлежать несколько гонщиков, так и 1 гонщик может принадлежать многим событиям.

Связь между сущностями Tournaments и Events не идентифицирующая, так как эти сущности могут быть созданы без сущности Tournaments. Тип связи 1:М, т.к. один турнир может иметь множество событий.

Структуры сущностей баз данных программно-информационной системы представлены в виде таблиц: структура сущности типов «Types» представлена в таблице 2.14; структура сущности пользователей «Users» представлена в таблице 2.15; структура сущности комментариев «Comments» представлена в таблице 2.16; структура сущности события «Events» представлена в таблице 2.17; структура сущности спонсоры «Sponsors» представлена в таблице 2.18; структура сущности гонщики «Racers» представлена в таблице 2.19; структура промежуточной сущности между событиями и гонщиками «Event\_racer» представлена в таблице 2.20; структура промежуточной сущности между событиями и спонсорами «Event\_sponsor» представлена в таблице 2.21; структура сущности турниров «Tournaments» представлена в таблице 2.22.

Таблица 2.14 – Структура таблицы «Types»

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN NAME | DATATYPE | NOT NULL | COMMENTS |
| Id | integer | Yes | PK |
| USER\_ID | bigint | Yes | FK |

Таблица 2.15 – Структура таблицы «Users»

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN NAME | DATATYPE | NOT NULL | COMMENTS |
| id | integer | Yes | PK |
| name | varchar(50) | Yes |  |
| numberphone | varchar(13) | Yes |  |
| password | varchar(50) | Yes |  |
| role | varchar(15) | Yes |  |
| lastname | varchar(15) | Yes |  |

Таблица 2.16 – Структура таблицы «Comments»

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN NAME | DATATYPE | NOT NULL | COMMENTS |
| id | integer | Yes | PK |
| text | text | Yes |  |
| data | timestamp | Yes |  |
| file | bytea | No |  |
| userId | integer | Yes | FK |

Таблица 2.17 – Структура таблицы «Events»

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN NAME | DATATYPE | NOT NULL | COMMENTS |
| id | integer | Yes | PK |
| data | timestamp | Yes |  |
| name | varchar(50) | Yes |  |
| description | text | Yes |  |
| photo | bytea | No |  |
| typeId | integer | Yes | FK |
| sponsorId | integer | Yes | FK |
| commentId | integer | Yes | FK |

Таблица 2.18 – Структура таблицы «Sponsors»

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN NAME | DATATYPE | NOT NULL | COMMENTS |
| id | integer | Yes | PK |
| logo | bytea | No |  |
| name | varchar(25) | Yes |  |
| advesteringInformaton | text | Yes |  |
| contacts | varchar(50) | Yes |  |
| id | integer | Yes | PK |

Таблица 2.19 – Структура таблицы «Racers»

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN NAME | DATATYPE | NOT NULL | COMMENTS |
| id | integer | Yes | PK |
| name | varchar(50) | Yes |  |
| lastname | varchar(50) | Yes |  |

Таблица 2.20 – Структура таблицы «Event\_racer»

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN NAME | DATATYPE | NOT NULL | COMMENTS |
| event\_id | integer | Yes | FK |
| racer\_id | integer | No | FK |

Таблица 2.21 – Структура таблицы «Event\_sponsor»

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN NAME | DATATYPE | NOT NULL | COMMENTS |
| event\_id | integer | Yes | FK |
| sponsor\_id | integer | No | FK |

Таблица 2.22 – Структура таблицы «Tournaments»

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN NAME | DATATYPE | NOT NULL | COMMENTS |
| id | integer | Yes | PK |
| name | varchar(50) | Yes |  |

**3 Реализация программно-информационной системы**

## 3.1 Описание архитектуры программно-информационной системы

Для создания программно-информационной системы была выбрана архитектура, основанной на работе микросервисов, представляет собой подход, при котором приложение разделяется на небольшие автономные сервисы, каждый из которых выполняет свою конкретную функцию. Каждый микросервис может быть разработан и развернут независимо от других сервисов, что упрощает разработку и обновление приложения в целом.

Клиент-серверная архитектура микросервисов состоит из трех уровней:

1. уровень клиента. Клиентское приложение обеспечивает интерфейс пользователя для взаимодействия с системой;
2. уровень микросервисов. Все функции приложения разделены на независимые микросервисы, каждый из которых выполняет свою функцию. Микросервисы взаимодействуют между собой через API, обеспечивая общий функционал системы;
3. уровень базы данных. Все микросервисы имеют свою собственную базу данных, что обеспечивает независимость их работы и устойчивость к сбоям.

Преимущества архитектуры микросервисов заключаются в возможности более гибкой разработки и масштабирования приложения, быстрой доставке новых функций и улучшений, а также увеличении устойчивости системы в целом благодаря независимости микросервисов от других компонентов приложения. Однако, такая архитектура требует более сложного управления и мониторинга работы микросервисов.

Программная архитектура системы представлена на рисунке 3.1.

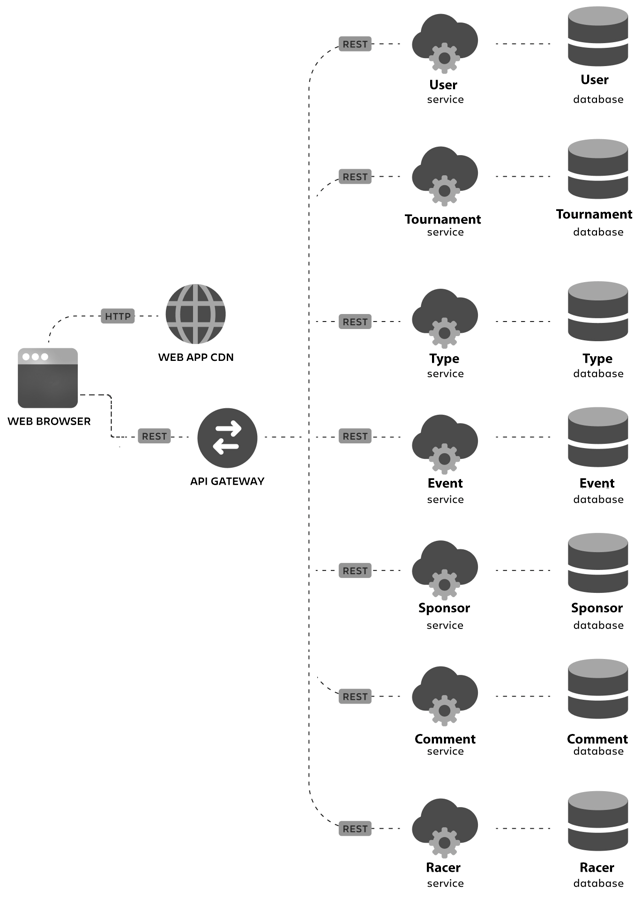


Рисунок 3.1 – Программная архитектура системы

Для реализации серверной части web-приложения был использован фреймворк Spring Boot. Он базируется на архитектуре Spring Framework, что означает, что Spring Boot также имеет многоуровневую архитектуру, при этом он имеет множество преимуществ перед Spring Framework таких как:

* Упрощенная конфигурация: Spring Boot предоставляет механизм автоматической конфигурации, что упрощает создание и настройку приложений.
* Самодостаточность: Spring Boot позволяет создавать самодостаточные приложения, которые содержат все необходимые зависимости и конфигурации.
* Ускоренный запуск: Spring Boot использует встроенный сервер приложений, который позволяет ускорить запуск приложения.
* Легкая масштабируемость: Spring Boot обеспечивает легкую масштабируемость приложений.

## В рамках проекта использовались основные компоненты архитектуры Spring Boot-приложения такие как:

## Модели (Models). Они представляют собой классы-сущности (User, Type, Event, Sponsor, Tournament, Racer) и служат для описания сущностей и логики. В рамках структуры проекта модели располагаются в папке "model" соответствующего микросервиса.

## Контроллеры (Controllers). Они представляют собой классы-контроллеры (UserController, TypeController, EventController, SponsorController, CommentController, TournamentController, RacerController) и служат для обработки HTTP-запросов и возврата ответа клиенту. В рамках структуры проекта классы-контроллеры располагаются в папке "controller" соответствующего микросервиса.

## Сервисы (Services). Они представляют собой классы-сервисы (UserService, TypeService, EvenService, SponsorService, CommentService, TournamentService, RacerService). Сервисы обеспечивают отделение бизнес-логики от контроллеров и слоев репозиториев. В рамках структуры проекта классы-сервисы располагаются в папке "service" соответствующего микросервиса.

## Репозитории (Repositories). Они представляют собой классы-репозитории (UserRepository, TypeRepository, EvenRepository, SponsorRepository, CommentRepository, TournamentRepository, RacerRepository) и обеспечивают доступ к данным в базе данных. В рамках структуры проекта классы-репозитории располагаются в папке "repository" соответствующего микросервиса.

## Для предоставления результатов обработки данных верхний слой, слой бизнес-логики (Business Logic Layer), взаимодействует с базой данных и передает полученные данные в слой представления. Нижний слой, слой доступа к данным (Data Access Layer), отвечает за хранение сущностей и репозиториев для взаимодействия слоя бизнес-логики и базы данных.

## На диаграмме, представленной на рисунке 3.2, показаны компоненты данной программно-информационной системы.

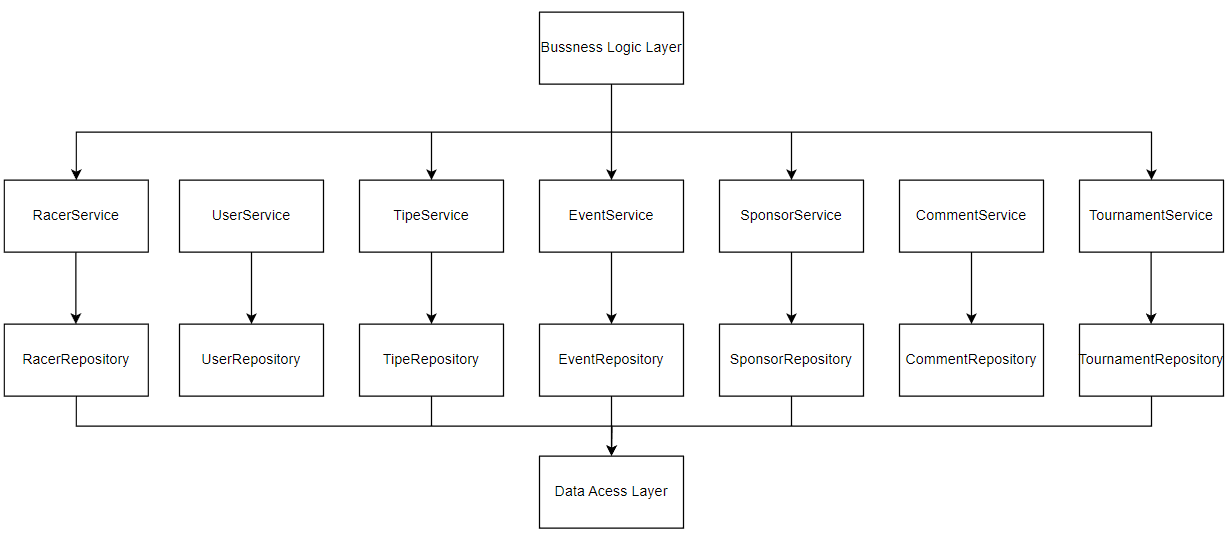


Рисунок 3.2 – Диаграмма компонентов программно-информационной системы

На рисунке 3.3 изображена диаграмма развертывания данной программно-информационной системы.

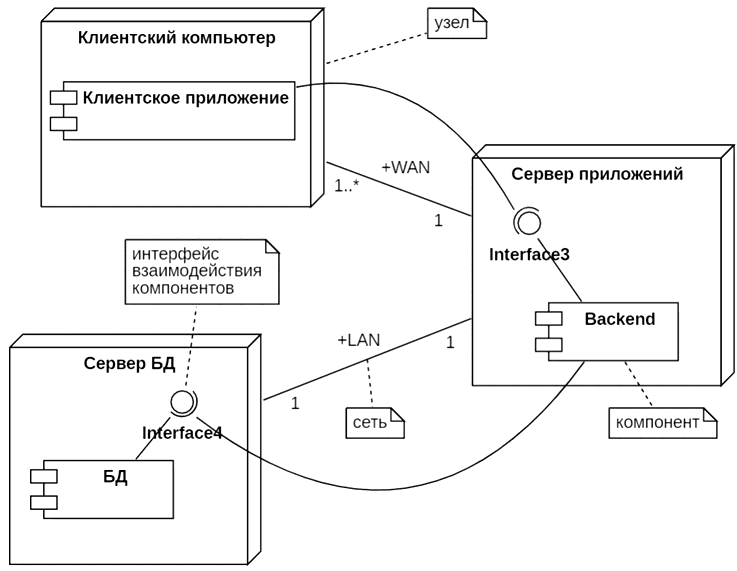


Рисунок 3.3 – Диаграмма развертывания информационной системы

## Диаграмма развертывания информационной системы показывает архитектуру исполнения системы, включая такие узлы, как аппаратные или программные среды исполнения, а также промежуточное программное обеспечение, соединяющее их.

## 3.2 Пример использования программно-информационной системы

При открытии web-приложения, разрабатываемой программно-информационной системы, нас встречает главная страница, на которой располагаются список соревнований и навигационная панель.

На навигационной панели находятся кнопки для перехода между страницами сайта, кнопка выбора типа соревнований, строка для поисковых запросов а также кнопки для входа и регистрации. Главная страница представлена на рисунке 3.4.

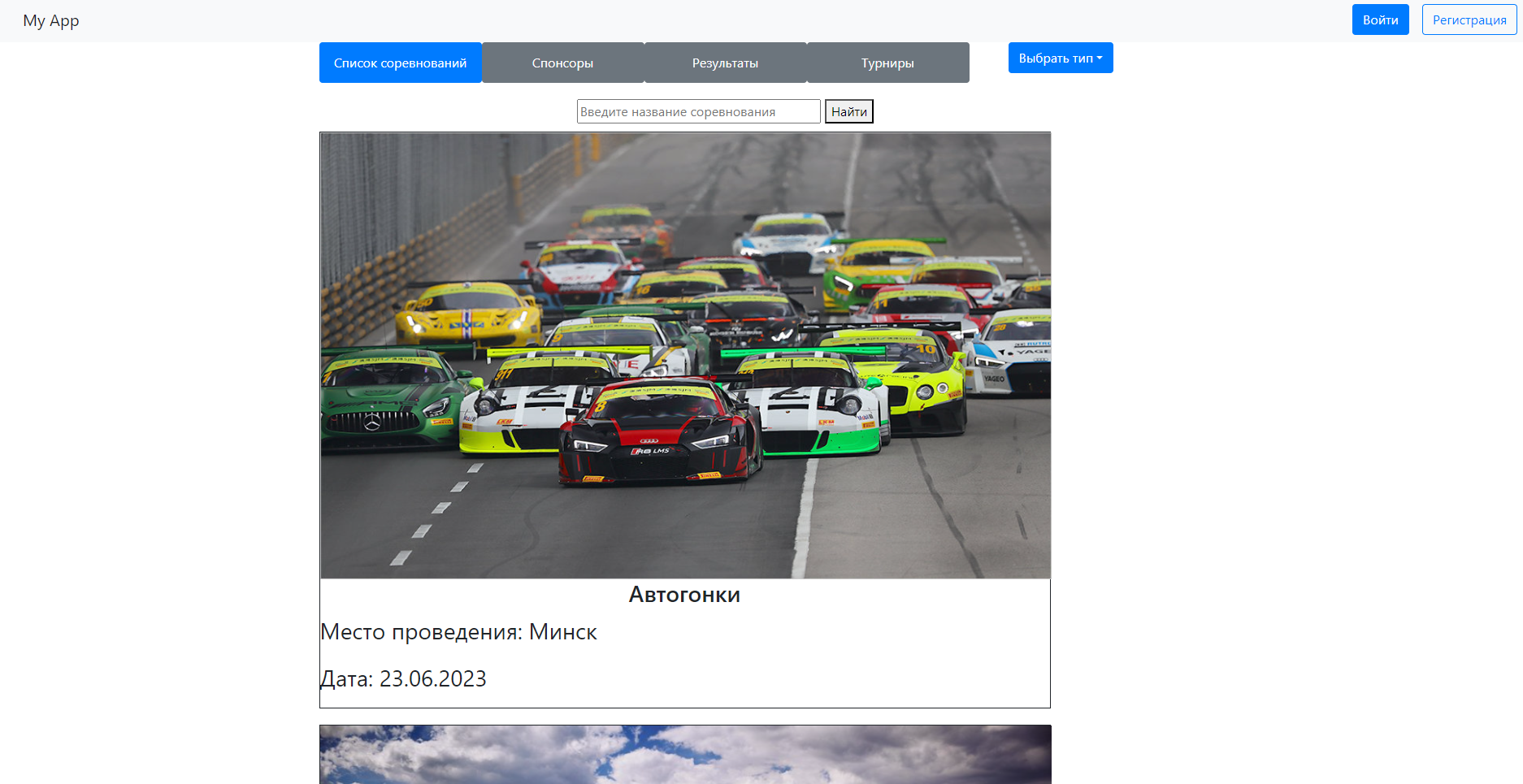


Рисунок 3.4 – Главная страница приложения

Пользователь, просматривая список соревнований, может узнать подробную информацию о любом событии просто нажав на него, после чего ему откроется подробная информация и возможность оставить комментарий. Страница подробной информации о событие представлена на рисунке 3.5.



Рисунок 3.5 – Страница подробной информации о событии

Если пользователь нажмет на кнопку «Спонсоры» на навигационной панели, то ему откроется список всех спонсоров сайта. Результат представлен на рисунке 3.6.

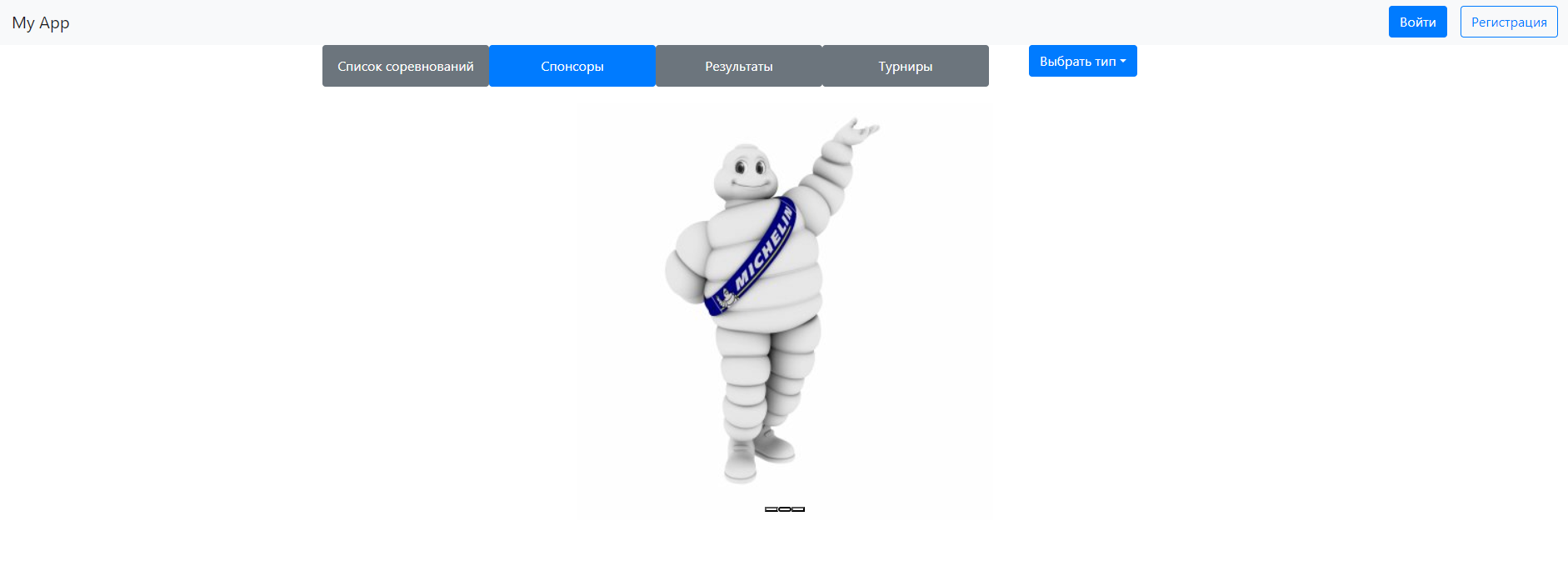


Рисунок 3.6 – Страница спонсоров

Если пользователь нажмет на кнопку «Результаты» на навигационной панели, то ему откроется таблица результатов. Результат представлен на рисунке 3.7.

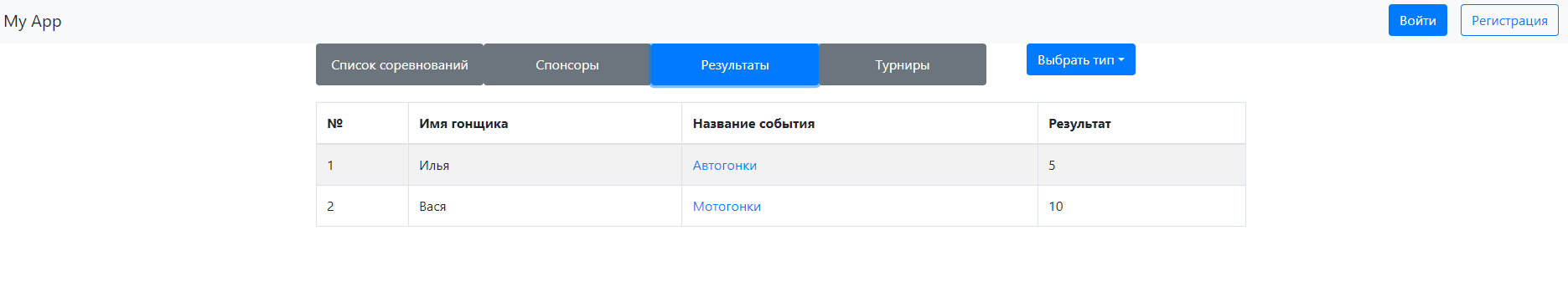


Рисунок 3.7 – Страница результатов

Если пользователь нажмет на кнопку «Турниры» на навигационной панели, то ему откроется список турниров. Результат представлен на рисунке 3.8.

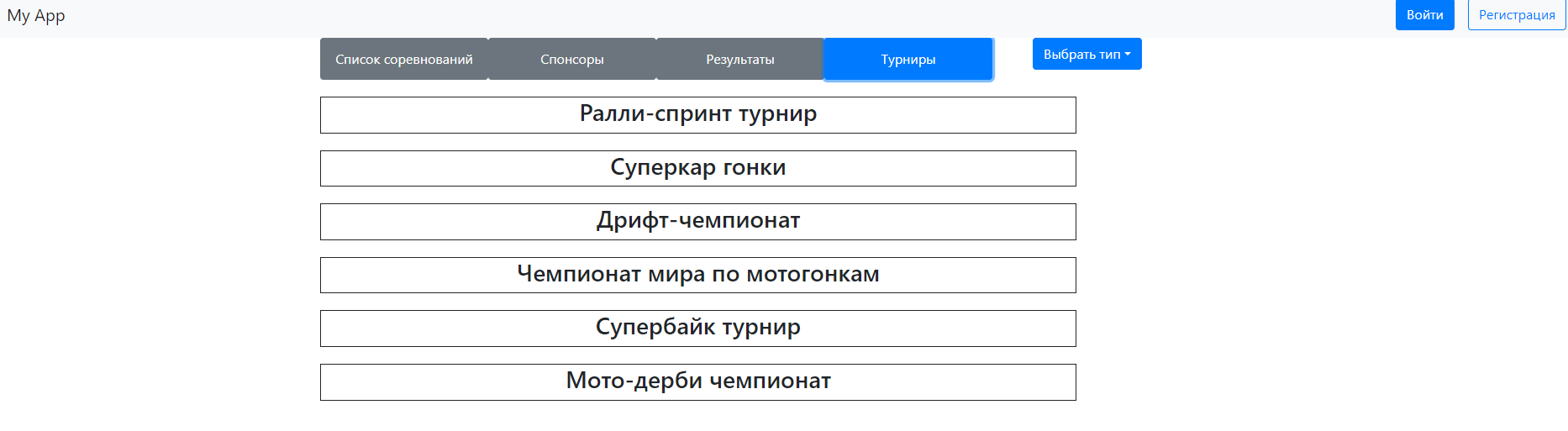


Рисунок 3.8 – Страница турниров

Пользователь может также отсортировать соревнования по типу, нажав на кнопку «Выбрать тип» откроется окно со списком типов соревнований. Результат представлен на рисунке 3.9.

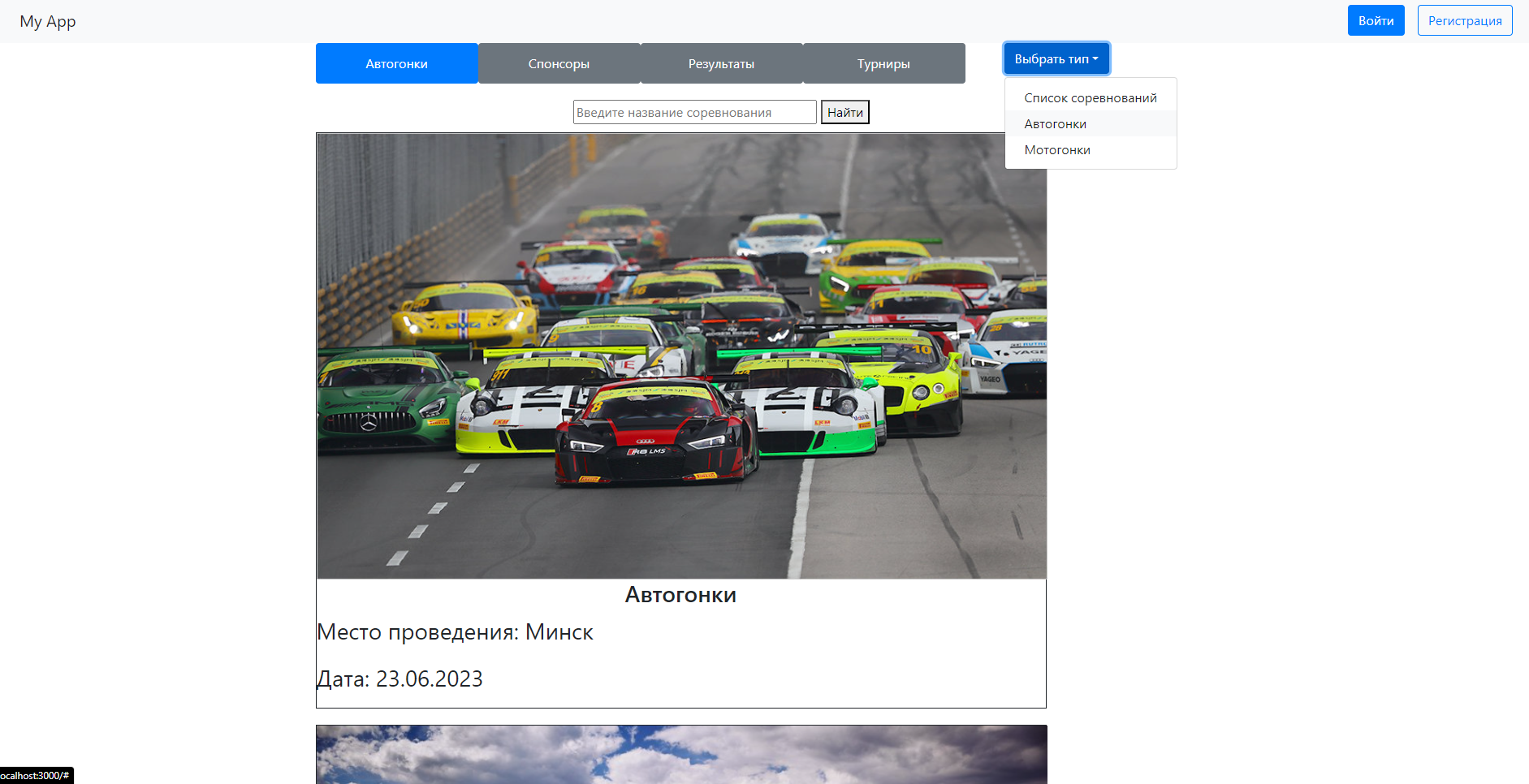


Рисунок 3.8 – Выбор типа соревнований

Пользователь также может авторизоваться, нажав кнопку «Войти», или зарегистрироваться, нажав кнопку «Регистрация». Ему откроется страница регистрации или авторизации в зависимости от выбора. Страницы авторизации и регистрации представлены на рисунках 3.9 и 3.10.

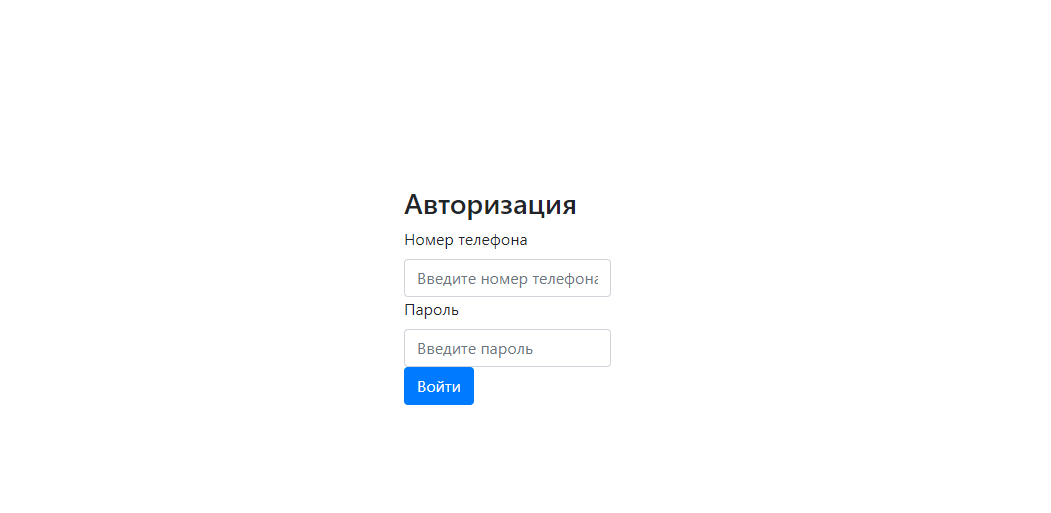


Рисунок 3.9 – Страница авторизации

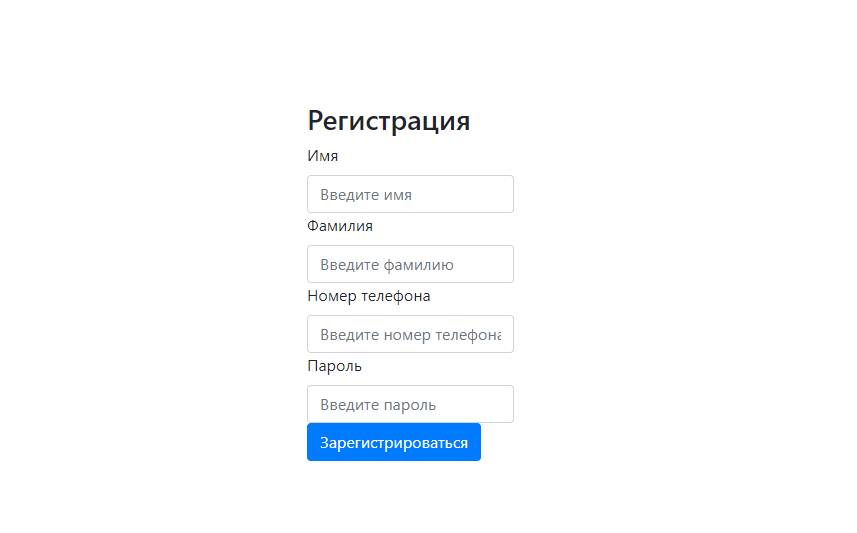


Рисунок 3.10 – Страница регистрации

Если в приложение зайдет Администратор у него появится панель администратора, нажав на которую откроется страница панели администратора. Главная страница с статусом администратора и страница с панелью администратора представлена на рисунке 3.11 и 3.12.

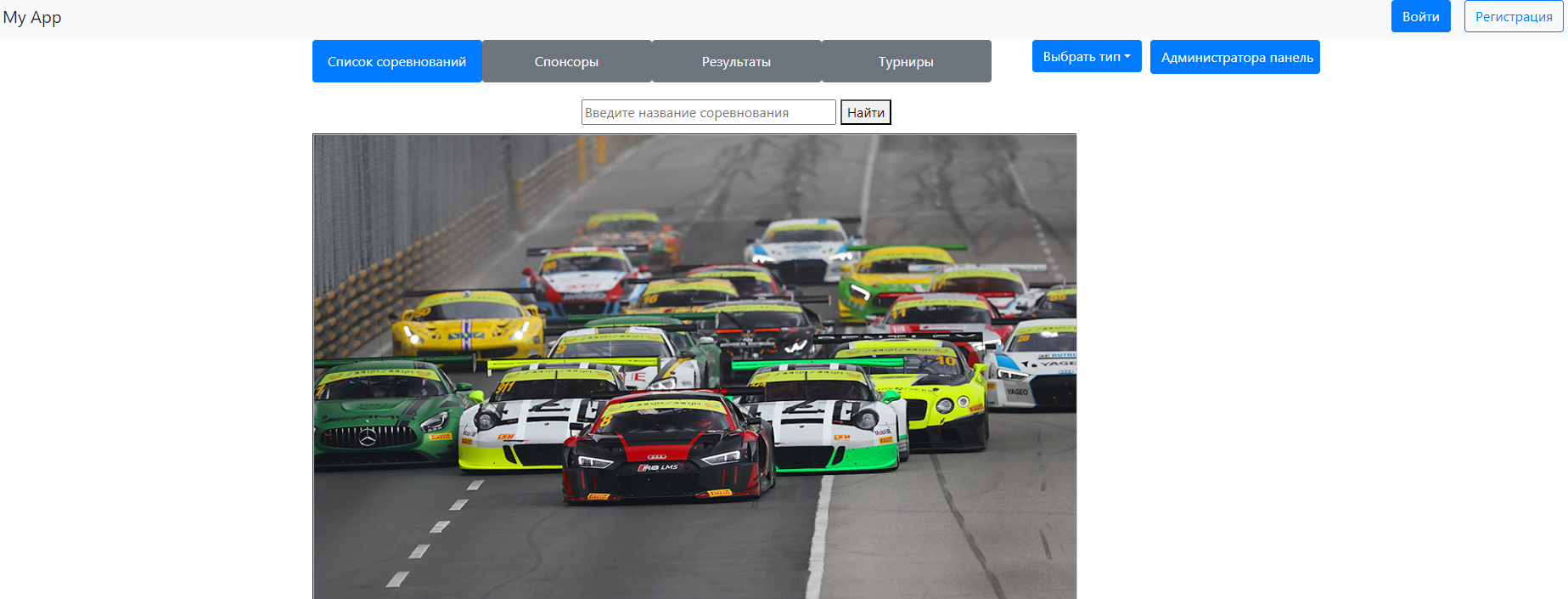


Рисунок 3.11 – Главная страница с правами администратора

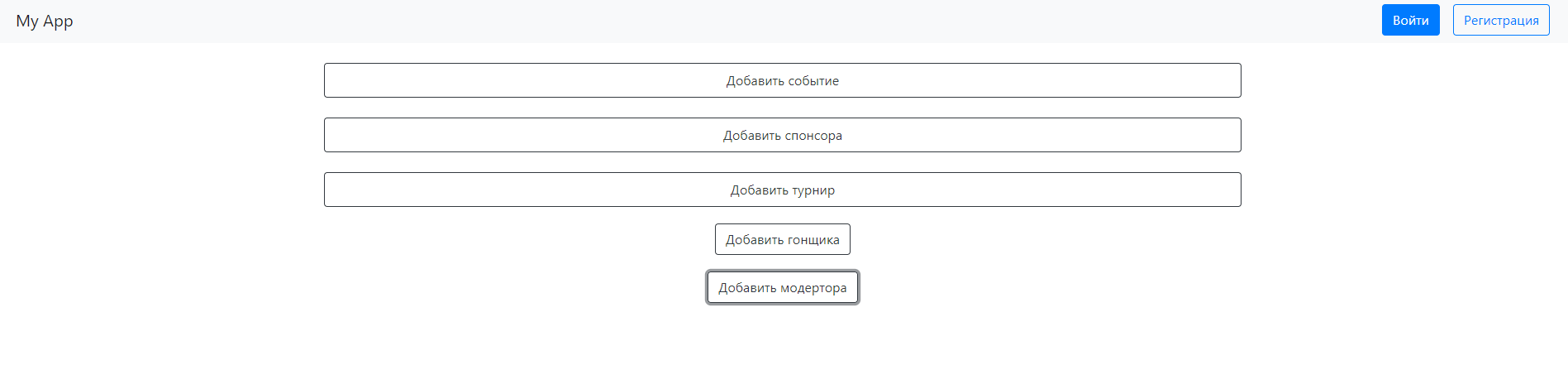


Рисунок 3.12 – Страница панели администратора

Администратор, с помощью этой панель, может добавить событие, добавить спонсора, добавить турнир, добавить гонщика и добавить модератора. Нажав на любой из этих пунктов, откроется окно с функциями. Открытие окна «Добавить событие» представлен на рисунке 3.13, открытие окна «Добавить спонсора» представлена на рисунке 3.14, открытие окна «Добавить турнир» представлена на рисунке 3.15, открытие окна «Добавить гонщика» представлена на рисунке 3.16, открытие окна «Добавить модератора» представлена на рисунке 3.17.

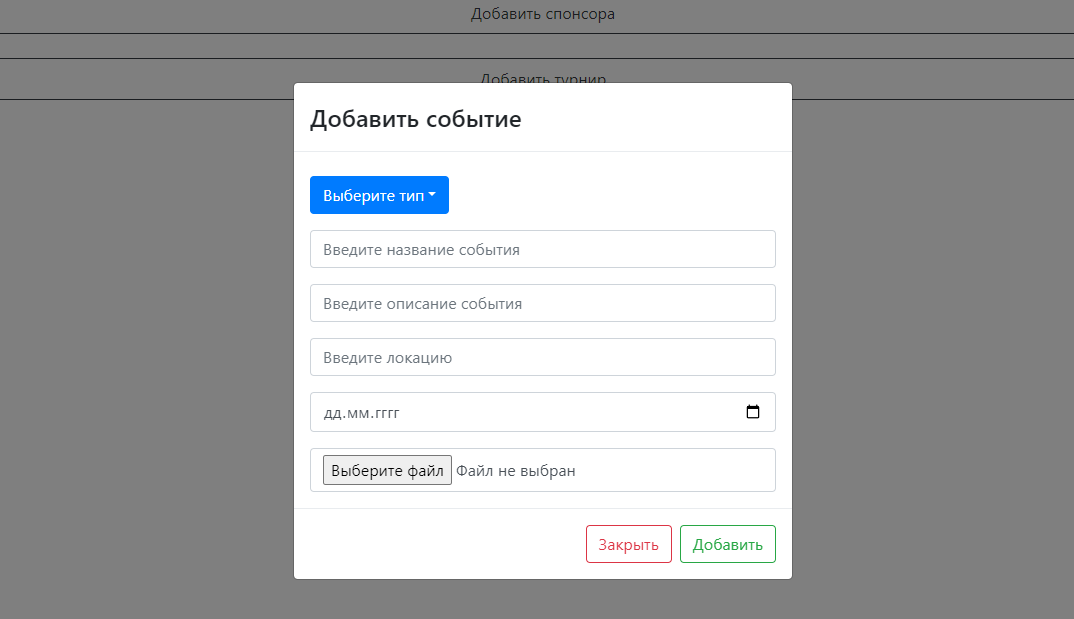


Рисунок 3.13 – Окно добавления события

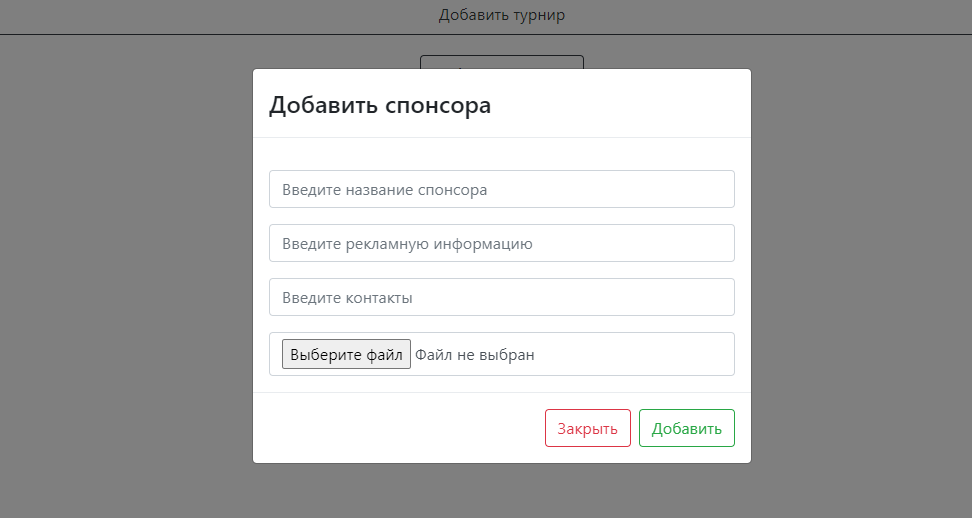


Рисунок 3.14 – Окно добавления спонсора

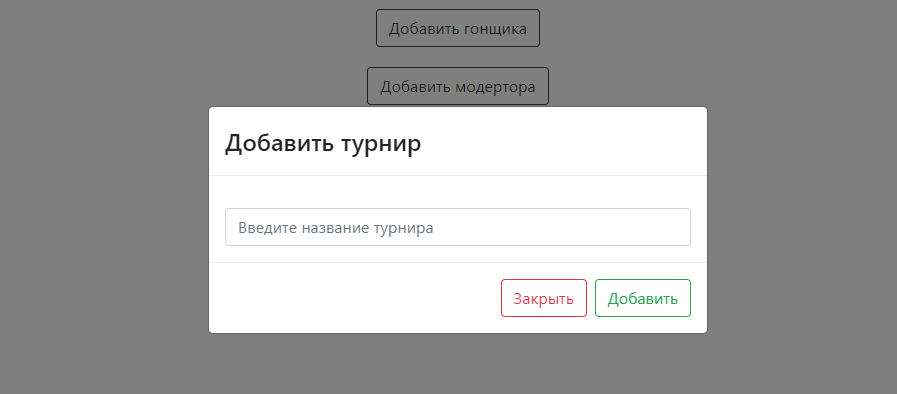


Рисунок 3.15 – Окно добавления турнира

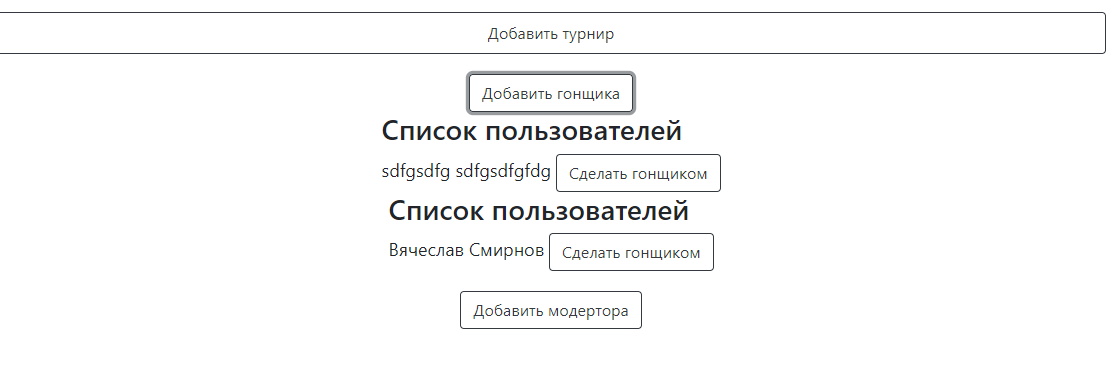


Рисунок 3.16 – Окно добавления гонщика



Рисунок 3.17 – Окно добавления модератора

Если зайдет пользователь с ролью модератора, то он сможет удалять комментарии других пользователей. Для этого ему нужно перейти на страницу соревнования, где будет доступна функция удаления комментариев. Результат представлен на рисунке 3.18.

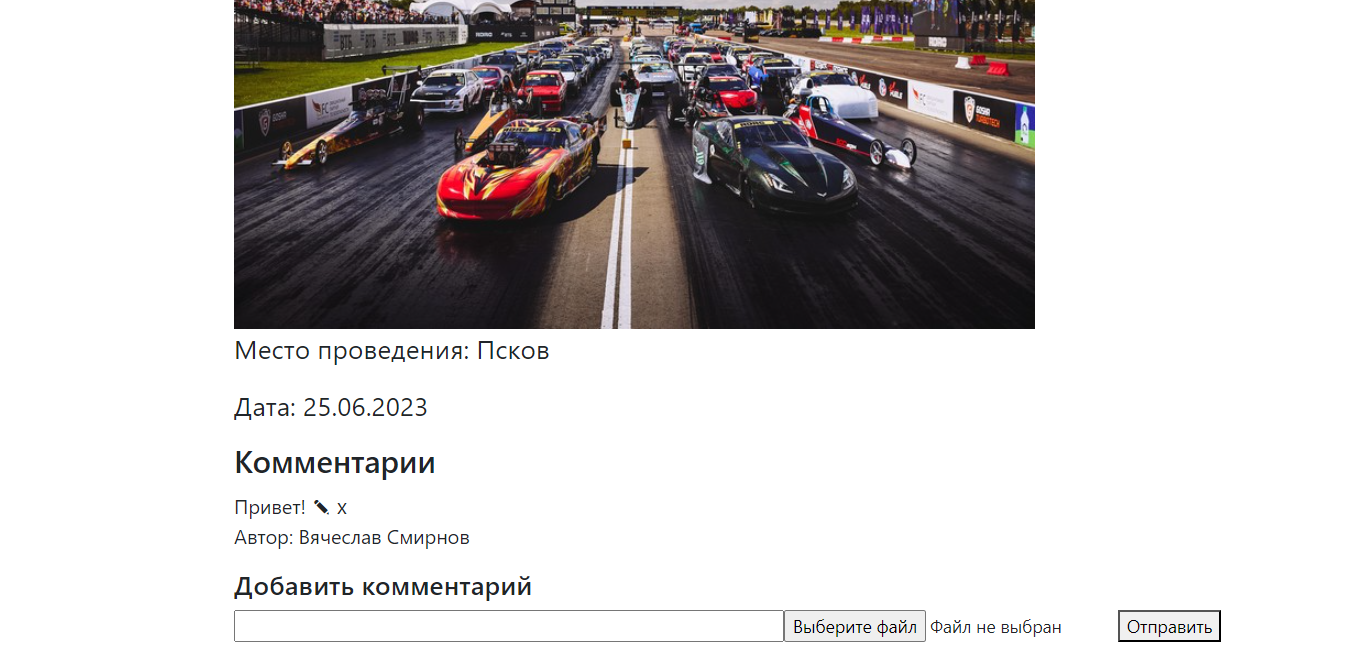


Рисунок 3.18 – Страница соревнования с правами модератора

**4 Результаты реализации программно-информационной системы**

В ходе разработки программно-информационной системы было создано web-приложение, соответствующее описанию архитектуры программно-информационной системы.

Приложение прошло функциональное и системное тестирование на выявление ошибок.

Было выполнено тестирование компонентов, что позволило выявить ошибки и дефекты в отдельных компонентах. Компонентное тестирование упростило отладку и повысило качество конечного web-приложения.

Программно-информационная система прошла все тесты и на данный момент не вызывает замечаний.

# Заключение

В итоге реализации выпускной квалификационной работы была разработана программно-информационная система организации автомобильных и мотоциклетных соревнований.

Разрабатываемое web-приложение ставит целью помочь желающим следить за соревнованиями автомобильного и мотоциклетного спорта. При использовании данной информационной системы пользователи смогут просматривать события автомобильного и мотоциклетного спорта, а также делиться впечатлениями.

Данное приложение будет полезно как обычным пользователям, а также пользователям, которые хотят быть участниками соревнований, размещенных в этом web-приложении.

Результатом разработки стало web-приложение, упрощающие организацию автомобильных и мотоциклетных соревнований.

# Список использованных источников

1. React JavaScript-библиотека для создания пользовательских интерфейсов [Электронный ресурс]. – Режим доступа: https://ru.reactjs.org. Дата доступа: 30.05.2023
2. Компоненты и пропсы React [Электронный ресурс]. – Режим доступа: https://ru.reactjs.org/docs/components-and-props.html. Дата доступа: 31.05.2023
3. REST with Spring Tutorial [Электронный ресурс]. – Режим доступа: https://www.baeldung.com/rest-with-spring-series. Дата доступа: 02.06.2023
4. Введение Axios Docs [Электронный ресурс]. – Режим доступа: https://axios-http.com/ru/docs/intro. Дата доступа: 05.06.2023
5. Веб-сайт аналога приложения Новости поехали onliner [Электронный ресурс]. – Режим доступа: <https://just-eat.by/mogilev>. Дата доступа: 07.06.2023
6. React Router. GeeksForGeeks. [Электронный ресурс]. – Режим доступа: <https://spring.io/guides/gs/rest-service/>. Дата доступа: 05.06.2023
7. Руководство по React [Электронный ресурс]. – Режим доступа: <https://proglib.io/p/json-tokens>. Дата доступа: 05.06.2023
8. **John Carnell.** Spring Microservices in Action. — Кэри.: США, 2017. – 384 c.
9. **Курняван, Б.** Программирование WEB-приложений на языке Java / Б. Курняван. - М.: Лори, 2014. - 880 c
10. **Алан Бьюли.** Изучаем SQL. — СПб.: Москва, 2007. — 301 с.
11. **Хеклер Марк.** Spring Boot по-быстрому. — СПб.: Питер, 2022. — 352с.
12. **Уоллс К.** Spring в действии. 6-е изд./ пер. с англ.А. Н. Киселева. – М.: ДМКПресс, 2022. – 544 с

## Приложение А

## (справочное)

**Код программы**

package edu.bru.commentmicroservice.controller;

import edu.bru.commentmicroservice.dto.CommentDto;

import edu.bru.commentmicroservice.model.Comment;

import edu.bru.commentmicroservice.model.User;

import edu.bru.commentmicroservice.security.JwtUtil;

import edu.bru.commentmicroservice.service.CommentService;

import jakarta.servlet.http.HttpServletRequest;

import lombok.RequiredArgsConstructor;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

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

import java.io.IOException;

import java.util.List;

@RestController

@RequiredArgsConstructor

@RequestMapping("/comment")

public class CommentController {

private final JwtUtil jwtUtil;

private final CommentService commentService;

@GetMapping

public ResponseEntity<List<Comment>> get() {

return ResponseEntity.ok(commentService.get());

}

@GetMapping("/user")

public ResponseEntity<List<Comment>> getByUser(HttpServletRequest httpServletRequest) {

return ResponseEntity.ok(commentService.getByUser(extractUserName(httpServletRequest)));

}

@PostMapping(path = "/{id}", consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)

public ResponseEntity<Void> add(@PathVariable Long id, @ModelAttribute CommentDto comment, HttpServletRequest httpServletRequest) throws IOException {

commentService.add(id, comment, extractUserName(httpServletRequest));

return ResponseEntity.ok().build();

}

@PutMapping(value = "/{id}", consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)

public ResponseEntity<Void> update(@RequestParam Long eventId, @ModelAttribute CommentDto comment, @PathVariable Long id, HttpServletRequest httpServletRequest) throws IOException {

commentService.update(eventId, comment, id, extractUserName(httpServletRequest));

return ResponseEntity.ok().build();

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> delete(@RequestParam Long eventId, @PathVariable Long id, HttpServletRequest httpServletRequest) {

commentService.delete(eventId, id, extractUserName(httpServletRequest));

return ResponseEntity.ok().build();

}

private String extractUserName(HttpServletRequest request) {

final String authorizationHeader = request.getHeader("Authorization");

String username = null;

String jwt = null;

if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {

jwt = authorizationHeader.substring(7);

username = jwtUtil.extractUsername(jwt);

}

return username;

}

}

package edu.bru.commentmicroservice.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

import jakarta.persistence.\*;

import lombok.\*;

import java.time.LocalDateTime;

@Entity

@Table(name = "COMMENTS")

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@ToString

@EqualsAndHashCode

@Builder

public class Comment {

@Id

@GeneratedValue (strategy = GenerationType.SEQUENCE)

@Column (name = "ID")

private Long id;

@Column (name = "TEXT")

private String text;

@Column (name = "DATE")

private LocalDateTime date;

@Lob

private byte[] file;

@Column(name = "CHANGED")

private Boolean changed;

@ManyToOne

private User user;

@JsonIgnore

@JoinColumn(name = "event\_id")

@ManyToOne

private Event event;

}

package edu.bru.eventmicroservice.controller;

import edu.bru.eventmicroservice.dto.EventDto;

import edu.bru.eventmicroservice.model.Comment;

import edu.bru.eventmicroservice.model.Event;

import edu.bru.eventmicroservice.model.User;

import edu.bru.eventmicroservice.security.JwtUtil;

import edu.bru.eventmicroservice.service.EventService;

import jakarta.servlet.http.HttpServletRequest;

import lombok.RequiredArgsConstructor;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

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

import java.io.IOException;

import java.util.List;

@RestController

@RequiredArgsConstructor

@RequestMapping("/event")

public class EventController {

private final EventService eventService;

private final JwtUtil jwtUtil;

@GetMapping

public ResponseEntity<List<Event>> get() {

return ResponseEntity.ok(eventService.get());

}

@GetMapping("/{id}")

public ResponseEntity<Event> get(@PathVariable Long id) {

return ResponseEntity.ok(eventService.getById(id));

}

@GetMapping("/search")

public ResponseEntity<Event> getByName(@RequestParam String name) {

return ResponseEntity.ok(eventService.getByName(name));

}

@PostMapping(consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)

public ResponseEntity<Void> add(@ModelAttribute EventDto event, HttpServletRequest httpServletRequest) throws IOException {

eventService.add(event, extractUserName(httpServletRequest));

return ResponseEntity.ok().build();

}

@PostMapping("/{id}/sponsor")

public ResponseEntity<Void> addSponsor(@PathVariable Long id, @RequestParam Long sponsorId) {

eventService.addSponsor(id, sponsorId);

return ResponseEntity.ok().build();

}

@PostMapping("/{id}/racer")

public ResponseEntity<Void> addRacer(@PathVariable Long id, @RequestParam Long racerId) {

eventService.addRacer(id, racerId);

return ResponseEntity.ok().build();

}

@PostMapping("/{id}/tournament")

public ResponseEntity<Void> addTournament(@PathVariable Long id, @RequestParam Long tournamentId) {

eventService.addTournament(id, tournamentId);

return ResponseEntity.ok().build();

}

@PutMapping(value = "/{id}", consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)

public ResponseEntity<Void> update(@PathVariable Long id, @ModelAttribute EventDto event, HttpServletRequest httpServletRequest) throws IOException {

eventService.update(id, event, extractUserName(httpServletRequest));

return ResponseEntity.ok().build();

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> delete(@PathVariable Long id, HttpServletRequest httpServletRequest) {

eventService.delete(id, extractUserName(httpServletRequest));

return ResponseEntity.ok().build();

}

private String extractUserName(HttpServletRequest request) {

final String authorizationHeader = request.getHeader("Authorization");

String username = null;

String jwt = null;

if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {

jwt = authorizationHeader.substring(7);

username = jwtUtil.extractUsername(jwt);

}

return username;

}

}

package edu.bru.eventmicroservice.model;

import edu.bru.eventmicroservice.model.enums.StateEvent;

import jakarta.persistence.\*;

import lombok.\*;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

@Entity

@Table(name = "EVENTS")

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@ToString

@EqualsAndHashCode

@Builder

public class Event {

@Id

@GeneratedValue(strategy = GenerationType.SEQUENCE)

@Column (name = "ID")

private Long id;

@Column (name = "NAME")

private String name;

@Column (name = "DESCRIPTION")

private String description;

@Column (name = "LOCATION")

private String location;

@Lob

@Column(columnDefinition = "bytea")

private byte[] picture;

@Column (name = "DATA")

private Date date;

@Enumerated(EnumType.STRING)

private StateEvent state;

@OneToOne

private Type type;

@OneToMany(mappedBy = "event", cascade = CascadeType.ALL, orphanRemoval = true)

private List<Comment> comments = new ArrayList<>();

@ManyToMany(fetch = FetchType.LAZY)

@JoinTable(

name = "event\_sponsor",

joinColumns = @JoinColumn(name = "event\_id"),

inverseJoinColumns = @JoinColumn(name = "sponsor\_id")

)

private List<Sponsor> sponsors;

@ManyToMany

@JoinTable(

name = "event\_racer",

joinColumns = @JoinColumn(name = "event\_id"),

inverseJoinColumns = @JoinColumn(name = "racer\_id")

)

private List<Racer> racers;

@OneToOne

private Tournament tournament;

}

package edu.bru.member.microservice.controller;

import edu.bru.member.microservice.model.Result;

import edu.bru.member.microservice.service.ResultService;

import lombok.RequiredArgsConstructor;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/result")

@RequiredArgsConstructor

public class ResultController {

private final ResultService resultService;

@GetMapping

public ResponseEntity<List<Result>> get() {

return ResponseEntity.ok(resultService.get());

}

@GetMapping("/{eventId}")

public ResponseEntity<List<Result>> get(@PathVariable Long eventId) {

return ResponseEntity.ok(resultService.getWinners(eventId));

}

@PostMapping

public ResponseEntity<Void> add(Result sponsor) {

resultService.add(sponsor);

return ResponseEntity.ok().build();

}

@PutMapping

public ResponseEntity<Void> update(Long id, Result sponsor) {

resultService.update(id, sponsor);

return ResponseEntity.ok().build();

}

@DeleteMapping

public ResponseEntity<Void> delete(Long id) {

resultService.delete(id);

return ResponseEntity.ok().build();

}

}

package edu.bru.member.microservice.model;

import jakarta.persistence.\*;

import lombok.\*;

@Entity

@Table(name = "RESULTS")

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

@ToString

@EqualsAndHashCode

@Builder

public class Result {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "ID")

private Long id;

@ManyToOne

private Racer racer;

@ManyToOne

private Event event;

private Integer points;

}

package edu.bru.sponsormicroservce.controller;

import edu.bru.sponsormicroservce.dto.SponsorDto;

import edu.bru.sponsormicroservce.model.Sponsor;

import edu.bru.sponsormicroservce.service.SponsorService;

import lombok.RequiredArgsConstructor;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

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

import java.io.IOException;

import java.util.List;

@RestController

@RequestMapping("/sponsor")

@RequiredArgsConstructor

public class SponsorController {

private final SponsorService sponsorService;

@GetMapping

public ResponseEntity<List<Sponsor>> get() {

return ResponseEntity.ok(sponsorService.get());

}

@PostMapping(consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)

public ResponseEntity<Void> add(SponsorDto sponsor) throws IOException {

sponsorService.add(sponsor);

return ResponseEntity.ok().build();

}

@PutMapping(path = "/{id}", consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)

public ResponseEntity<Void> update(@PathVariable Long id, SponsorDto sponsor) throws IOException {

sponsorService.update(id, sponsor);

return ResponseEntity.ok().build();

}

@DeleteMapping

public ResponseEntity<Void> delete(Long id) {

sponsorService.delete(id);

return ResponseEntity.ok().build();

}

}

package edu.bru.sponsormicroservce.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

import jakarta.persistence.\*;

import lombok.\*;

import java.util.List;

@Entity

@Table(name = "SPONSORS")

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

@ToString

@EqualsAndHashCode

@Builder

public class Sponsor {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "ID")

private Long id;

@Lob

private byte[] logo;

@Column(name = "NAME")

private String name;

@Column(name = "ADVERTISING\_INFORMATION")

private String advertisingInformation;

@Column(name = "CONTACTS")

private String contacts;

@ManyToMany(mappedBy = "sponsors")

@ToString.Exclude

@JsonIgnore

private List<Event> events;

}

package edu.bru.tournamentmicroservice.controller;

import edu.bru.tournamentmicroservice.model.Tournament;

import edu.bru.tournamentmicroservice.service.TournamentService;

import lombok.RequiredArgsConstructor;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/tournament")

@RequiredArgsConstructor

public class TournamentController {

private final TournamentService tournamentService;

@GetMapping

public ResponseEntity<List<Tournament>> get() {

return ResponseEntity.ok(tournamentService.get());

}

@GetMapping("/{id}")

public ResponseEntity<Tournament> getById(@PathVariable Long id) {

return ResponseEntity.ok(tournamentService.getById(id));

}

@PostMapping

public ResponseEntity<Void> add(Tournament tournament) {

tournamentService.add(tournament);

return ResponseEntity.ok().build();

}

@PutMapping("/{id}")

public ResponseEntity<Void> update(@PathVariable Long id, Tournament tournament) {

tournamentService.update(id, tournament);

return ResponseEntity.ok().build();

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> delete(@PathVariable Long id) {

tournamentService.delete(id);

return ResponseEntity.ok().build();

}

}

package edu.bru.tournamentmicroservice.model;

import jakarta.persistence.\*;

import lombok.\*;

@Entity

@Table(name = "TOURNAMENTS")

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@ToString

@EqualsAndHashCode

@Builder

public class Tournament {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "ID")

private Long id;

@Column(name = "name")

private String name;

}

package edu.bru.usermicroservice.controller;

import edu.bru.usermicroservice.dto.UserAuthenticationDto;

import edu.bru.usermicroservice.dto.UserRegistrationDto;

import edu.bru.usermicroservice.model.Racer;

import edu.bru.usermicroservice.model.User;

import edu.bru.usermicroservice.model.enums.UserRole;

import edu.bru.usermicroservice.repository.UserRepository;

import edu.bru.usermicroservice.security.JwtUtil;

import edu.bru.usermicroservice.security.userDetails.JwtUser;

import edu.bru.usermicroservice.service.UserService;

import lombok.RequiredArgsConstructor;

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.BadCredentialsException;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.annotation.AuthenticationPrincipal;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

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

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

import java.util.List;

@RestController

@RequiredArgsConstructor

@RequestMapping("/user")

public class UserController {

private final AuthenticationManager authenticationManager;

private final UserDetailsService userDetailsService;

private final UserService userService;

private final UserRepository userRepository;

private final JwtUtil jwtUtil;

@GetMapping("/{userId}")

public ResponseEntity<Void> addRacer(@PathVariable Long userId) {

userService.addRacer(userId);

return ResponseEntity.ok().build();

}

@GetMapping("/moderator/{userId}")

public ResponseEntity<Void> addModerator(@PathVariable Long userId) {

User user = userRepository.findById(userId).get();

user.setRole(UserRole.MODERATOR);

userRepository.save(user);

return ResponseEntity.ok().build();

}

@GetMapping("/users")

public ResponseEntity<List<User>> getUsers() {

return ResponseEntity.ok(userService.getUsers());

}

@GetMapping("/racers")

public ResponseEntity<List<Racer>> getRacers() {

return ResponseEntity.ok(userService.getRacers());

}

@GetMapping("/check")

public ResponseEntity<UserRole> checkRole(@AuthenticationPrincipal JwtUser jwtUser) {

return ResponseEntity.ok(userService.checkRole(jwtUser));

}

@PostMapping("/authentication")

public ResponseEntity<String> authenticate(@RequestBody UserAuthenticationDto user) {

authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(user.getNumberPhone(), user.getPassword())

);

final UserDetails userDetails = userDetailsService.loadUserByUsername(user.getNumberPhone());

if (userDetails == null) {

throw new BadCredentialsException("Not Valid email or password");

}

return ResponseEntity.ok(jwtUtil.generateToken(userDetails));

}

@PostMapping("/registration")

public ResponseEntity<String> register(@RequestBody UserRegistrationDto userRegistrationDto) {

String password = userRegistrationDto.getPassword();

userRegistrationDto.setPassword(new BCryptPasswordEncoder().encode(userRegistrationDto.getPassword()));

userService.registration(userRegistrationDto);

authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(userRegistrationDto.getNumberPhone(), password)

);

final UserDetails userDetails = userDetailsService.loadUserByUsername(userRegistrationDto.getNumberPhone());

if (userDetails == null) {

throw new BadCredentialsException("Not Valid email or password");

}

return ResponseEntity.ok(jwtUtil.generateToken(userDetails));

}

}