Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

|  |  |
| --- | --- |
|  | Номер зачетной книжки 15350096 |
|  | Практика зачтена с оценкой  *\_\_\_\_\_* (*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*)  (цифрой) (прописью) |
|  | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  (подпись руководителя практики от БГУИР)  *\_\_\_\_\_\_\_\_\_\_\_\_\_*.2025 |

**ОТЧЕТ**

**по практике**

Место прохождения практики: ЗАО «Международный деловой альянс», г. Минск

Сроки прохождения практики: с 10.02.2025 по 23.03.2025

|  |  |  |
| --- | --- | --- |
| Руководитель практики от предприятия:  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_*А.В.Макацкий  (подпись руководителя)  М.П. |  | Студент группы 153501  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*П.С.Павлюткин  (подпись студента)  Руководитель практики от БГУИР  Давыдчик А.В. – ассистент кафедры информатики |

Минск 2025

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc70968530)

[1 Анализ литературных источников, аналагов и формирование требований к проектируемому программному средству 4](#_Toc70968531)

[1.1 Аналитический обзор литературных источников 6](#_Toc70968532)

[1.2 Анализ существующих аналогов 13](#_Toc70968533)

[1.3 Требования к проектируемому программному средству 18](#_Toc70968534)

[2 Моделирование программного средства и разработка функциональных требований 21](#_Toc70968535)

[2.1 Функциональная модель программного средства 21](#_Toc70968536)

[2.2 Разработка спецификации функциональных требований 23](#_Toc70968537)

[3 Технико-экономическое обоснование разработки и реализации на рынке веб-приложения для организации спортивных мероприятий и турниров 26](#_Toc192495178)

[3.1 Характеристика программного средства, разрабатываемого для реализации на рынке 26](#_Toc192495179)

[3.2 Расчет инвестиций в разработку программного средства для   
      его реализации на рынке 28](#_Toc192495180)

[3.3 Расчет экономического эффекта от реализации программного средства   
      на рынке 30](#_Toc192495181)

[3.4 Расчет показателей экономической эффективности разработки и   
      реализации программного средства на рынке 31](#_Toc192495182)

[Заключение 34](#_Toc70968549)

[Список использованной литературы 35](#_Toc70968550)

ВВЕДЕНИЕ

Спортивные мероприятия и турниры занимают важное место в жизни общества, объединяя людей, мотивируя к достижению высоких результатов и способствуя развитию здорового образа жизни. Однако процесс организации таких мероприятий требует значительных временных и человеческих ресурсов, а также координации множества участников. Современные цифровые технологии позволяют автоматизировать эти процессы, повысить их эффективность и удобство как для организаторов, так и для участников. В рамках данной работы рассматривается разработка программного средства, предназначенного для управления процессами организации спортивных мероприятий и турниров.

Выбор веб-приложения в качестве основы проекта обусловлен его универсальностью и доступностью. Использование технологий Spring Boot для серверной части и React для клиентской позволяет создать масштабируемую и высокопроизводительную систему, поддерживающую удобный интерфейс взаимодействия с пользователями. В качестве базы данных применяется PostgreSQL, что обеспечивает надежное хранение данных и возможность обработки сложных запросов. Благодаря REST API обеспечивается гибкость интеграции с другими системами, а также возможность расширения функционала в будущем.

Вовлечение пользователей будет достигаться за счет ряда ключевых механизмов. Важную роль играет система персонализированных уведомлений, информирующая участников о предстоящих матчах, изменениях в расписании и других важных событиях. Встроенная система коммуникации позволит пользователям оперативно взаимодействовать друг с другом и организаторами. Геймификация, включая начисление баллов за участие и достижения, также будет способствовать повышению интереса пользователей к платформе.

Целью данного проекта является создание удобного и эффективного инструмента для автоматизации процессов организации спортивных мероприятий. Программное средство должно обеспечить удобную регистрацию команд и участников, автоматическое составление расписания соревнований, ведение турнирных таблиц, контроль за проведением матчей и возможность ведения аналитики по различным показателям. В процессе разработки также будет проведено исследование современных методик автоматизации подобных процессов, изучены подходы к обеспечению безопасности данных и организации удобного пользовательского интерфейса.

В результате проект должен предоставить комплексное решение, которое поможет организаторам существенно сократить временные затраты на планирование и проведение спортивных мероприятий, а участникам – получить удобный инструмент для взаимодействия и отслеживания прогресса в турнирах.

**1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ, АНАЛАГОВ И ФОРМИРОВАНИЕ ТРЕБОВНИЙ К ПРОГРАММНОМУ СРЕДСТВУ**

Итоговый успех разработки и реализации программного проекта напрямую зависит от этапа подготовки, на котором необходимо как можно более детально и тщательно определить все нюансы и особенности проекта. Чем качественнее будет проведена подготовительная работа, тем меньше возникнет проблем в процессе проектирования и реализации, а также тем выше вероятность успешного завершения проекта в срок и в рамках бюджета.

Одним из ключевых факторов, определяющих успешность разработки программного обеспечения, является ясное и четкое формулирование проблемы, которую система должна решать. Для проекта, направленного на разработку программного средства для управления процессами организации спортивных мероприятий и турниров, важно на раннем этапе точно понять, какие именно функции и процессы должны быть автоматизированы, какие задачи будут решаться с помощью системы и какие ожидания у конечных пользователей. Это позволит не только выработать четкие и обоснованные требования, но и избежать недоразумений на последующих этапах разработки.

Цель подготовительного этапа – минимизация рисков. Если на этапе подготовки удастся предусмотреть возможные проблемы и заранее спланировать все действия, то основную часть работы можно будет выполнить эффективно и без значительных затрат времени и ресурсов. Важно понимать, что проблемы, связанные с недостаточной подготовленностью на старте, могут привести к существенным задержкам, увеличению бюджета и снижению качества конечного продукта.

Один из главных факторов риска при разработке программного обеспечения – это неудачная и неполная выработка требований. Требования к системе должны быть максимально подробными и конкретными, четко описывая, что должна делать система, какие функции она должна выполнять и как должны взаимодействовать различные компоненты. Внимание к деталям на этапе формирования требований поможет свести к минимуму изменения в проекте после начала разработки, что значительно улучшит качество и срок выполнения проекта.

Для того чтобы сформировать правильные и реалистичные требования к программному обеспечению, необходимо провести предварительный анализ различных источников, таких как литература, исследования аналогичных решений и прототипы существующих систем. Это позволит выявить лучшие практики и определить, какие функции и характеристики программного средства могут быть полезны в контексте конкретной задачи.

Важно проанализировать уже существующие программные продукты и системы для управления спортивными мероприятиями и турнирами. Это включает в себя не только изучение их функциональных возможностей, но и выявление их недостатков, которые можно учесть при разработке нового продукта.

Прототипы и минимально жизнеспособные продукты (MVP) могут служить полезным ориентиром на этапе разработки. Их использование поможет понять, какие функции и возможности наиболее важны для конечных пользователей, а какие можно отложить на более поздние этапы или исключить из первого релиза.

Не менее важным аспектом является выбор платформы и архитектуры программного средства. Для успешной реализации проекта необходимо определить, какие технологии, языки программирования и архитектурные решения будут наиболее эффективными для решения поставленных задач. Это решение должно основываться на анализе требуемых характеристик системы, таких как масштабируемость, производительность, совместимость с различными устройствами и операционными системами.

После анализа существующих решений и выбора архитектуры можно переходить к составлению технического задания (ТЗ). ТЗ станет основой для разработки функциональных требований, описывающих, как должна работать система, какие процессы она должна поддерживать и какие данные обрабатывать.

В ходе подготовки проекта и формирования требований важно помнить, что проектирование программного средства должно быть гибким процессом, готовым к адаптации в зависимости от изменений в исходных данных, технологий и требований конечных пользователей. Правильное сочетание технической подготовки, анализа аналогичных решений и четкого определения требований создаст прочную основу для успешной реализации проекта и удовлетворения потребностей пользователей.

Таким образом, предварительный этап подготовки к разработке программного средства для управления процессами организации спортивных мероприятий и турниров включает в себя не только анализ литературных источников и существующих аналогов, но и тщательно продуманное формирование требований. Это позволит избежать многих рисков, повысит эффективность разработки и обеспечит конечный успех проекта.

* 1. **Аналитический обзор литературных источников**

Далее приводится анализ сведений, которые влияют на формулирование требований, выбор архитектуры и дальнейшее проектирование и разработку программного средства.

**1.1.1** Обзор целевой платформы

Разработка программного средства для управления процессами организации спортивных мероприятий и турниров предусматривает создание веб-приложения с клиентской частью, реализованной с использованием HTML5, CSS3, JavaScript с применением React. Выбор данной платформы обусловлен рядом факторов, связанных с удобством развертывания, кроссплатформенностью, простотой обновления и масштабируемостью. В данной главе будет проведен обзор ключевых преимуществ веб-приложений, а также рассмотрены их недостатки и потенциальные трудности, возникающие при разработке и эксплуатации.

Одним из главных преимуществ веб-приложений является их доступность. В отличие от нативных решений, требующих установки на каждое пользовательское устройство, веб-приложение развертывается на сервере и доступно через браузер. Это означает, что пользователи могут работать с системой из любой точки мира, имея лишь доступ в интернет и браузер, поддерживающий современные веб-стандарты (HTML5, CSS3, JavaScript) [1].

Еще одним важным преимуществом является централизованное управление обновлениями. В традиционных настольных и мобильных приложениях обновления требуют вмешательства со стороны пользователя: необходимо загружать, устанавливать и настраивать новую версию. В веб-приложениях этот процесс автоматизирован – обновления вносятся непосредственно на сервере, после чего все пользователи получают доступ к последней версии системы без необходимости выполнять дополнительные действия.

Кроме того, веб-приложения обеспечивают гибкость и кроссплатформенность. Они не привязаны к конкретной операционной системе или устройству – для работы достаточно браузера, будь то на Windows, macOS, Linux или мобильных платформах (Android, iOS). Это делает систему более универсальной и снижает затраты на разработку и поддержку, поскольку нет необходимости разрабатывать отдельные версии для каждой платформы.

Еще одним значимым плюсом является распределение нагрузки. Поскольку большая часть бизнес-логики и вычислений выполняется на сервере, клиентские устройства могут быть менее мощными. Это особенно важно, если предполагается работа с ресурсоемкими вычислениями, сложными алгоритмами или обработкой больших объемов данных. Такой подход позволяет снизить требования к клиентским устройствам и повысить общую производительность системы.

Несмотря на значительные преимущества, веб-приложения имеют и свои недостатки, которые необходимо учитывать при разработке.

Один из главных минусов – зависимость от интернет-соединения. В отличие от нативных приложений, которые могут работать в офлайн-режиме, веб-приложение требует постоянного доступа к сети для полноценного функционирования. Если у пользователя слабое или нестабильное соединение, это может привести к ухудшению пользовательского опыта, задержкам в загрузке страниц и некорректной работе функционала.

Важным фактором является также нагрузка на сервер. Поскольку все основные вычисления и обработка данных выполняются на стороне сервера, требуется мощное серверное оборудование и продуманная архитектура масштабирования. Если система будет одновременно обслуживать большое количество пользователей, необходимо предусмотреть балансировку нагрузки, резервные серверы и другие механизмы для обеспечения бесперебойной работы. Это увеличивает затраты на инфраструктуру и требует постоянного мониторинга и оптимизации [2].

Еще одна сложность связана с безопасностью. Веб-приложения, работающие через браузер, подвержены различным угрозам, таким как SQL-инъекции, XSS-атаки (межсайтовый скриптинг), CSRF-атаки (подделка межсайтовых запросов) и другие уязвимости. Необходимо внедрять надежные механизмы защиты, включая шифрование данных, аутентификацию пользователей, защиту API и регулярное тестирование безопасности.

Также стоит учитывать особенности работы в браузере. В отличие от нативных приложений, веб-приложения зависят от браузерных технологий, которые могут отличаться в разных версиях и на разных устройствах. Несмотря на унификацию стандартов, возможны проблемы совместимости, различия в отображении интерфейса и производительности. Для решения этих проблем требуется тестирование на различных устройствах и браузерах, что усложняет процесс разработки.

Выбор веб-приложения в качестве целевой платформы для разрабатываемого программного средства обусловлен его универсальностью, простотой обновления, доступностью и снижением требований к клиентским устройствам. Однако при этом необходимо учитывать риски, связанные с зависимостью от интернет-соединения, нагрузкой на серверную инфраструктуру, безопасностью данных и кроссбраузерной совместимостью.

Понимание преимуществ и недостатков данной платформы позволяет заранее предусмотреть возможные сложности и разработать стратегию их минимизации, что обеспечит успешную реализацию проекта и высокий уровень удовлетворенности пользователей.

**1.1.2** Обзор архитектурных стилей

Разработка программного средства для управления процессами организации спортивных мероприятий и турниров требует тщательно продуманного архитектурного подхода, который обеспечит надежность, масштабируемость и удобство поддержки системы. В рамках данного проекта используются следующие архитектурные стили:

1. Монолитная архитектура – для серверной части.
2. Клиент-серверная архитектура – для взаимодействия клиентской и
3. Объектно-ориентированная архитектура – для построения серверной части с применением принципов ООП.
4. Компонентная и декларативная архитектура – для фронтенд-разработки на React [3].

Монолитная архитектура представляет собой традиционный подход, при котором все компоненты системы – пользовательский интерфейс, бизнес-логика и работа с данными – объединены в один единственный исполняемый модуль.

Преимущества монолитной архитектуры:

1. Простота разработки и развертывания – отсутствие необходимости координировать множество отдельных сервисов упрощает процесс написания и отладки кода.
2. Целостность данных – доступ к данным осуществляется локально, без необходимости использовать сложные механизмы межсервисного взаимодействия.
3. Минимальные накладные расходы – отсутствие необходимости управлять коммуникацией между микросервисами снижает нагрузку на сервер.
4. Единый стек технологий – позволяет избежать проблем совместимости между разными языками программирования и фреймворками.
5. Более простое тестирование – удобнее проводить интеграционное и сквозное тестирование, так как вся система находится в одном месте.

Недостатки монолитной архитектуры:

1. Сложность масштабирования – вертикальное масштабирование (увеличение мощности одного сервера) имеет ограничения, в отличие от микросервисного горизонтального масштабирования.
2. Сложность внесения изменений в больших системах – при значительном росте кода могут появляться сложности с внесением новых функций.
3. Низкая гибкость в выборе технологий – все модули должны использовать одни и те же инструменты и библиотеки.

Для данного проекта выбрана монолитная архитектура, так как она лучше всего подходит для относительно небольшого веб-приложения, не требующего высокой степени модульности и частого изменения технологий. Также такой подход позволит быстрее разрабатывать и тестировать систему без необходимости управлять множеством отдельных сервисов [4].

Клиент-серверная архитектура определяет принцип взаимодействия двух основных компонентов системы. Клиент – фронтенд-приложение, запущенное в браузере пользователя. Сервер – бэкенд-приложение, обрабатывающее запросы и управляющее базой данных.

Преимущества клиент-серверной архитектуры:

1. Централизованное хранение данных – информация хранится на сервере, что повышает безопасность и целостность данных.
2. Обновляемость системы – клиенты автоматически получают актуальную версию без необходимости обновления ПО на стороне пользователя.
3. Минимальные требования к клиентскому устройству – вся сложная обработка выполняется на сервере, что позволяет работать даже на слабых устройствах.
4. Гибкость в масштабировании – серверную часть можно масштабировать независимо от клиентской.

Недостатки клиент-серверной архитектуры:

1. Зависимость от сети – веб-приложение не будет работать без подключения к интернету.
2. Нагрузки на сервер – при большом числе пользователей требуется мощная серверная инфраструктура.

Для данного проекта клиент-серверная архитектура является естественным выбором, так как приложение представляет собой веб-сервис, работающий в браузере и взаимодействующий с сервером через REST API.

Объектно-ориентированная архитектура применяется для построения серверной части с использованием принципов ООП (объектно-ориентированного программирования).

Преимущества объектно-ориентированной архитектуры:

1. Структурированность кода – разбиение системы на объекты, классы и модули облегчает поддержку и рефакторинг.
2. Инкапсуляция – скрытие внутренней логики объектов повышает безопасность и снижает риск ошибок.
3. Повторное использование кода – полиморфизм и наследование позволяют легко расширять функциональность.
4. Легкость тестировании – тестирование отдельных классов и компонентов упрощает отладку и ускоряет разработку.

Недостатки объектно-ориентированной архитектуры:

1. Высокая сложность при неправильном проектировании – запутанные связи между классами могут затруднять разработку.
2. Перегруженность кода – чрезмерное использование классов и наследования может привести к избыточности [5].

Серверная часть приложения будет разрабатываться с использованием Java и Spring Boot, что естественно соответствует объектно-ориентированному подходу. Классы будут представлять сущности предметной области (турниры, команды, пользователи и т. д.), а бизнес-логика будет реализована через сервисные слои [6].

Фронтенд-приложение разрабатывается с использованием React, который поддерживает несколько архитектурных подходов. React-приложение будет построено на основе модульных компонентов, которые можно переиспользовать и объединять в более сложные структуры. Преимущества:

1. Повторное использование кода.
2. Упрощенная отладка.
3. Легкость тестирования и поддержки.

React использует декларативный подход, который упрощает работу с UI. Разработчик описывает, что нужно отобразить, а React сам обновляет DOM-дерево при изменении состояния. Преимущества:

1. Минимизация работы с DOM, что улучшает производительность.
2. Прозрачное управление состоянием.
3. Уменьшение количества ошибок при рендеринге.

В процессе разработки программного средства для управления спортивными мероприятиями и турнирами применяются различные архитектурные стили, каждый из которых отвечает за свою задачу:

1. Монолитная архитектура упрощает разработку серверной части.
2. Клиент-серверная архитектура обеспечивает разделение фронтенда и бэкенда.
3. Объектно-ориентированная архитектура структурирует код серверной части.
4. Компонентная и декларативная архитектура упрощает разработку фронтенда на React.

Такой подход позволяет достичь оптимального баланса между удобством разработки, производительностью и масштабируемостью.

**1.1.3** Проектирование базы данных

В современных информационных системах, особенно в тех, которые ориентированы на управление процессами и взаимодействие множества пользователей, надежная и продуманная база данных является фундаментом всей архитектуры. В программном средстве, разрабатываемом для управления процессами организации спортивных мероприятий и турниров, база данных играет ключевую роль в систематизации и структурировании данных, обеспечивая их целостность, доступность и эффективность обработки.

Для реализации хранения данных в разрабатываемом веб-приложении выбрана реляционная система управления базами данных (СУБД) PostgreSQL. Этот выбор обусловлен особенностями предметной области, требованиями к надежности, консистентности данных, а также необходимостью поддержки сложных связей между сущностями, характерных для спортивных соревнований. В турнирах и спортивных мероприятиях задействованы различные участники: команды, игроки, судьи, организаторы, болельщики. Все они взаимодействуют в рамках четко определенных регламентов, что требует строгой структурированности данных и возможности их оперативной обработки.

Спортивные соревнования подразумевают работу с большим количеством взаимосвязанных данных. Например, при проведении турнира необходимо учитывать состав команд, расписание матчей, результаты игр, турнирные таблицы, статистику отдельных игроков, регламент соревнования, а также финансовые и административные аспекты, такие как взносы, призовые фонды и выплаты судьям. PostgreSQL, как объектно-реляционная СУБД, идеально подходит для такого рода задач, так как позволяет четко структурировать данные, обеспечивать их логическую связанность и предотвращать избыточность за счет нормализации [7].

В проектируемой базе данных ключевыми сущностями будут:

1. Турниры, содержащие информацию о названии, типе (одиночный, командный), формате проведения (групповой этап, плей-офф, смешанный), организаторе и временных рамках.
2. Команды и игроки, объединенные отношением многие-ко-многим, так как один игрок может участвовать в нескольких турнирах за разные команды.
3. Матчи, включающие данные о составе команд, счете, судьях и результатах.
4. Судьи, назначаемые на конкретные матчи и обладающие возможностью фиксировать результаты.
5. Финансовые операции, связанные с регистрационными взносами, оплатой судей и распределением призового фонда.

Наличие сложных взаимосвязей между объектами требует использования внешних ключей, индексов и каскадных операций для поддержания целостности данных. PostgreSQL предоставляет все необходимые механизмы для этих задач, обеспечивая надежность и скорость обработки информации.

Спортивные турниры часто подразумевают финансовые вложения и обязательства, что делает критически важным обеспечение защиты данных от потерь и несанкционированного доступа. PostgreSQL поддерживает ACID-транзакции, что гарантирует атомарность, консистентность, изолированность и долговечность операций. Это особенно важно при обновлении результатов матчей и изменении турнирных таблиц, где даже малейшая ошибка может привести к некорректным итогам соревнований.

База данных будет содержать уровни доступа, позволяя разграничить права пользователей. Например, администраторы смогут управлять турнирами и финансовыми операциями, организаторы — редактировать состав команд, судьи — фиксировать результаты матчей, а игроки и зрители — просматривать расписание и таблицы. PostgreSQL предоставляет гибкую систему управления ролями и привилегиями, что позволяет эффективно реализовать такую модель.

Система должна быть способной обрабатывать запросы в реальном времени, так как пользователи, особенно зрители и участники турниров, заинтересованы в оперативном обновлении информации о ходе соревнований. PostgreSQL предоставляет широкий набор инструментов для оптимизации запросов, включая индексацию, анализ выполнения SQL-запросов (EXPLAIN ANALYZE) и использование кэширования.

Возможны сценарии резкого увеличения нагрузки, например, в моменты регистрации большого количества пользователей или массового обновления результатов матчей. В таких случаях могут быть задействованы механизмы горизонтального и вертикального масштабирования. PostgreSQL поддерживает репликацию данных, позволяя распределять нагрузку между несколькими серверами, а также шардирование, что делает ее пригодной для использования в масштабных системах.

В процессе работы системы могут возникать ситуации, когда часть информации не укладывается в строго определенную схему. Например, организаторы турниров могут добавлять специфические параметры, такие как особые правила, дополнительные категории участников или бонусные очки. Для хранения таких данных в PostgreSQL можно использовать JSONB, который позволяет комбинировать реляционную модель с элементами NoSQL.

Дополнительно в системе могут использоваться географические данные, например, для определения местоположения спортивных объектов и построения карт маршрутов. PostgreSQL предоставляет расширение PostGIS, которое позволяет эффективно работать с координатами и геоданными, что может быть полезно при интеграции с картографическими сервисами.

Одной из возможных альтернатив PostgreSQL мог бы быть NoSQL-подход, например, Firebase, который предлагает гибкость и удобство работы с неструктурированными данными. Однако для системы управления спортивными соревнованиями критически важно строгое соблюдение связей, транзакционная целостность и возможность сложных аналитических запросов, что делает реляционные базы данных более предпочтительным выбором.

В отличие от NoSQL, PostgreSQL позволяет использовать строго типизированные данные, что снижает вероятность ошибок и упрощает валидацию. Также SQL-запросы обеспечивают возможность составления сложных аналитических выборок, например, для построения статистики по турнирам, вычисления рейтингов игроков и анализа исторических данных.

Таким образом, использование PostgreSQL в качестве базы данных для разрабатываемого программного средства является обоснованным решением, соответствующим потребностям системы и требованиям предметной области. Реляционная структура данных позволит четко моделировать связи между объектами, поддерживать целостность информации и обеспечивать удобство работы пользователей с системой. Встроенные механизмы безопасности, масштабируемости и оптимизации позволят справляться с высокой нагрузкой и обеспечивать стабильную работу приложения даже при интенсивном использовании.

Выбранная архитектура базы данных закладывает надежный фундамент для последующего развития системы, позволяя в будущем расширять функционал, интегрироваться с внешними сервисами и обеспечивать высокое качество обслуживания пользователей [8].

* 1. Анализ существующих аналогов

Разработка программного обеспечения для управления процессами организации спортивных мероприятий и турниров требует изучения уже существующих решений. Это позволит определить их ключевые возможности, а также выявить недостатки, которые следует учесть при создании собственного продукта. На рынке представлено несколько платформ, предлагающих функциональность, схожую с требуемой для управления спортивными мероприятиями. Однако большинство из них ориентированы либо на управление спортивными командами, либо на бронирование и организацию логистики для мероприятий, но не совмещают все необходимые функции. Рассмотрим основные решения.

EventPipe — облачная платформа, предназначенная для управления бронированием и размещением участников мероприятий, включая спортивные турниры (см. рисунок 1.1). Она фокусируется на логистике и обеспечении комфортного размещения спортсменов и организаторов. Ключевые функции:

1. Бронирование отелей: позволяет организаторам выбирать отели для участников, управлять условиями размещения и предоставлять скидки.
2. Настраиваемый сайт мероприятия: создание брендированного веб-приложения для бронирования без необходимости программирования.
3. Управление групповыми блоками: контроль за доступностью номеров, назначение менеджеров для управления группами гостей.
4. Аналитика и отчеты: сбор статистики по бронированию, финансовым потокам и заполняемости отелей.

Недостатки данной облачной платформы:

1. Отсутствие инструментов для управления самим турниром: EventPipe ориентирован на бронирование отелей, но не включает функционал для составления расписаний игр, регистрации команд и учета результатов матчей.
2. Отсутствие мобильного приложения: доступ осуществляется только через веб-интерфейс, что может быть неудобно для участников, желающих быстро внести изменения.
3. Зависимость от партнерских отелей: функциональность бронирования сильно привязана к базе данных отелей, что делает продукт менее универсальным для спортивных мероприятий, проходящих в нестандартных локациях. Также просматривается специфика данного сервиса именно для организации комфортного проведения, но никак не самого мероприятия.

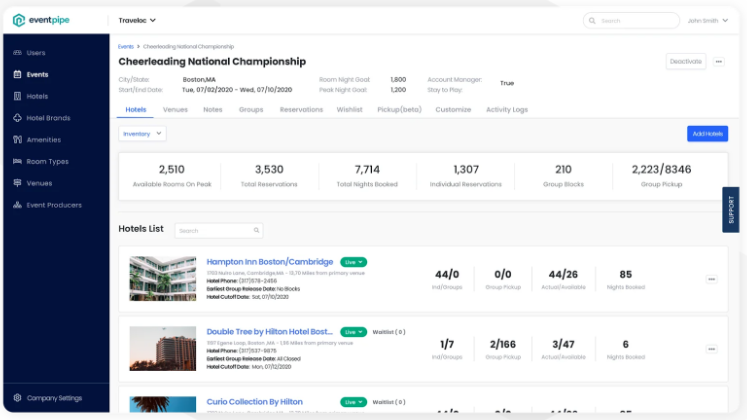


Рисунок 1.1 – облачная платформа EventPipe

Stack Team App — платформа для управления спортивными командами, позволяющая создать собственное мобильное приложение без навыков программирования (см. рисунок 1.2). Ключевые функции:

1. Коммуникация: чат для команды с возможностью отправки объявлений и обновлений.
2. Управление командой: инструменты для отслеживания посещаемости, создания расписания игр и организации мероприятий.
3. Опросы и голосования: возможность проводить внутренние голосования среди участников.
4. Интеграция с социальными сетями: поддержка публикации новостей команды в Facebook и Twitter.

Основные недостатки данной платформы:

1. Отсутствие полноценного управления турнирами: приложение больше ориентировано на отдельные команды, чем на организацию многокомандных соревнований.
2. Ограниченный функционал в бесплатной версии: многие ключевые функции, включая аналитику и брендирование, доступны только в платной подписке.
3. Нет инструментов для бронирования и финансового управления: приложение не поддерживает организацию проживания участников и не включает модули для работы с платежами.

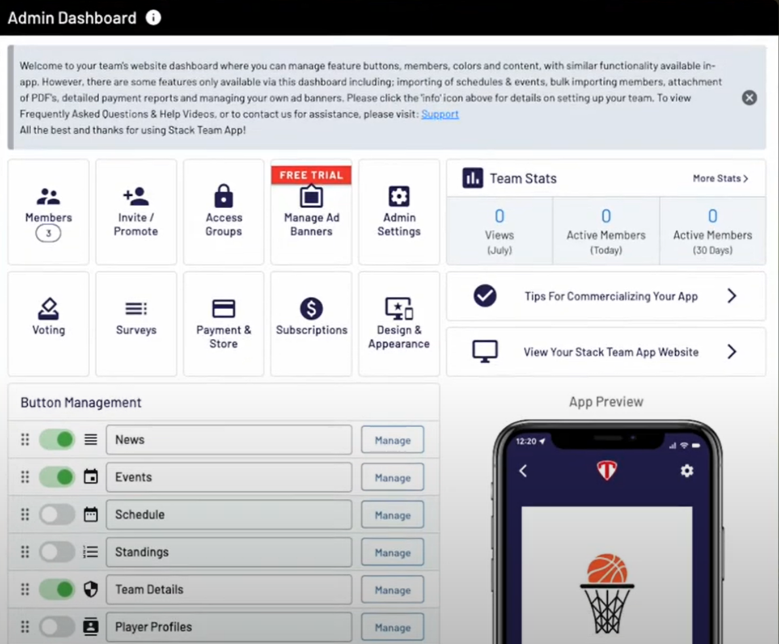


Рисунок 1.2 – платформа для управления спортивными командами Stack Team App

[SportCRM](https://sportcrm.ru/) — система для автоматизации процессов в спортивных клубах и школах. Она помогает управлять клиентами, тренерами и финансами, но может быть адаптирована для работы с турнирами. Ключевые функции:

1. Учет спортсменов и тренеров: управление данными клиентов, тренерским составом, расписанием занятий.
2. Финансовый контроль: учет платежей, анализ доходов и расходов, управление скидками и подписками.
3. CRM-функционал: ведение клиентской базы, управление обращениями, контроль сроков действия документов.
4. Интеграция с системами страхования: автоматический учет медицинских страховок участников.

Недостатки:

1. Не адаптирован для управления турнирами: SportCRM создан для спортивных клубов, а не для организации соревнований, что требует значительной доработки для использования в турнирах.
2. Нет поддержки многокомандных матчей и сетки турниров: приложение не предоставляет инструменты для создания расписания игр и отслеживания результатов соревнований.
3. Сложность настройки: система ориентирована на бизнес-процессы спортивных школ, что делает ее сложной для быстрой интеграции в процессы организации турниров.

Анализ существующих решений показывает, что на рынке нет единого программного продукта, который бы полностью соответствовал требованиям для управления процессами организации спортивных мероприятий и турниров. Каждое рассмотренное ПО обладает преимуществами, но при этом имеет и существенные недостатки, которые могут ограничивать его применение в данной сфере.

EventPipe представляет собой удобный инструмент для бронирования отелей и размещения участников мероприятий. Его ключевое преимущество – автоматизация логистики и возможность создания брендированного веб-приложения для управления бронированием. Однако платформа не предоставляет функций для управления самим турниром, включая регистрацию команд, составление расписаний матчей и учет результатов. Кроме того, отсутствие мобильного приложения делает использование сервиса менее удобным, а зависимость от базы партнерских отелей ограничивает возможности для проведения соревнований в нестандартных локациях, таких как стадионы без развитой гостиничной инфраструктуры.

Stack Team App ориентирован на управление спортивными командами и предоставляет удобные инструменты для коммуникации, создания расписаний игр и проведения внутренних опросов среди участников. Его ключевым преимуществом является наличие мобильного приложения и интеграция с социальными сетями, что облегчает обмен информацией внутри команды. Однако приложение не поддерживает полноценное управление турнирами, так как фокусируется на отдельных командах, а не на соревнованиях с несколькими участниками. Кроме того, значительная часть полезного функционала доступна только в платной версии, а отсутствие инструментов для финансового учета и бронирования делает его неполноценным решением для комплексного управления спортивными мероприятиями.

SportCRM ориентирован на спортивные клубы и школы, предоставляя широкий спектр возможностей для учета спортсменов, тренеров и финансовых потоков. Он удобен для ведения клиентской базы и управления расписанием тренировок, а также включает CRM-функционал для работы с клиентами. Однако, несмотря на развитые финансовые инструменты и поддержку интеграции с системами страхования, SportCRM не адаптирован для организации турниров. В нем отсутствует функционал для создания расписаний матчей, ведения турнирных таблиц и учета результатов соревнований. Кроме того, сложность настройки системы может стать препятствием для быстрой интеграции в процессы организации спортивных мероприятий [9].

Таким образом, существующие решения не охватывают всех аспектов управления спортивными мероприятиями и турнирами. Одни системы ориентированы на бронирование и логистику, но не поддерживают управление играми, другие предназначены для работы с командами, но не учитывают организационные задачи турниров, а третьи сосредоточены на спортивных школах и клубах, не имея инструментов для многокомандных соревнований.

С учетом выявленных недостатков, разрабатываемое программное средство должно объединять функции управления турнирами и организацией логистики, что позволит охватить весь процесс подготовки и проведения спортивного мероприятия. Оно должно быть гибким и адаптивным, чтобы поддерживать различные форматы соревнований, а также иметь удобный и интуитивно понятный интерфейс. Немаловажным фактором является наличие мобильного приложения или адаптивного веб-интерфейса, что обеспечит удобство работы для организаторов и участников. Важным преимуществом станет интеграция с платежными системами для автоматизированного сбора взносов и оплаты услуг, что позволит значительно упростить финансовый учет и снизить административную нагрузку.

С учетом этих аспектов, разрабатываемое программное средство сможет занять свою нишу среди аналогичных решений, предоставив организаторам турниров удобный инструмент для комплексного управления спортивными мероприятиями.

**1.3** **Требования к проектируемому программному средству**

После анализа литературных источников, а также обзора уже существующих аналогов необходимо сформулировать требования к проектируемому программному средству

**1.3.1** Назначение проекта

Назначением проекта является разработка программного средства для управления процессами организации спортивных мероприятий и турниров. Оно должно обеспечивать планирование, регистрацию участников, автоматизацию судейства, расчет результатов и интеграцию с платежными системами.

**1.3.2** Основные функции

Программное средство должно поддерживать следующие основные функции:

* регистрация, аутентификация, авторизация пользователей;
* управление профилями пользователей (спортсмены, тренеры, организаторы, судьи);
* создание и настройка спортивных мероприятий и турниров;
* управление расписанием соревнований;
* автоматизация фиксации результатов матчей;
* интеграция с платежными системами для оплаты регистрационных взносов и подписок;
* генерация отчетов, таблиц и диаграмм о соревнованиях и рейтингов спортсменов;
* поддержка push-уведомлений и email-рассылки для участников.

**1.3.3** Дополнительные функции

После анализа аналогичных систем были определены дополнительные функции:

* модуль блога, новостей и чатов для обмена пользовательским опытом;
* возможность ведения подробного учета соревнований;
* автоматизированное формирование турнирных сеток и расписаний;
* API для интеграции с внешними сервисами (картами, спортивными организациями, сервисами по составлению продвинутой статистики, социальными сетями);
* поддержка многопользовательского режима для организаторов.

**1.3.4** Требования к входным данным

Входные данные должны включать информацию, вводимую пользователями через веб-интерфейс или мобильное приложение, файлы соответствующих форматов (CSV, JSON, XML) для загрузки данных о спортсменах и соревнованиях, а также данные, полученные через API от внешних сервисов [10].

Программное средство должно обеспечивать валидацию входных данных, включая проверку корректности введенных имен, контактных данных, документов и финансовых транзакций.

**1.3.5** Требования к выходным данным

Выходные данные должны включать:

* отображение информации через пользовательский интерфейс (веб приложение);
* экспорт данных в PDF, Excel, JSON и другие форматы;
* автоматизированную рассылку уведомлений о статусе соревнований;
* визуализацию турнирных таблиц и рейтингов.

**1.3.6** Требования к надежности

Для обеспечения надежной работы программного средства должны быть реализованы следующие меры:

* резервное копирование данных;
* поддержка отказоустойчивых кластеров баз данных;
* защита от несанкционированного доступа;
* логирование всех значимых событий системы;
* использование механизма транзакций в БД для обеспечения целостности данных.

**1.3.7** Требования к аппаратному обеспечению клиентской части

Клиентская часть должна корректно функционировать на устройствах со следующими минимальными характеристиками:

* процессор Intel Core i5 или аналогичный;
* 8 ГБ оперативной памяти;
* доступ в интернет;
* поддержка современных браузеров: Google Chrome, Mozilla Firefox, Microsoft Edge (актуальные версии).

**1.3.8** Выбор инструментов разработки

Для реализации программного средства выбраны следующие технологии:

* Backend: Spring Boot, PostgreSQL;
* Frontend: React;
* API: RESTful с использованием OpenAPI;
* Система контроля версий: Git, хостинг кода – GitHub;
* CI/CD: GitHub Actions;
* Хостинг и развертывание: облачные сервисы (AWS, Notifly);
* Инструменты тестирования: JUnit, Mockito для backend, Jest для frontend.

Сформулированные требования позволят осуществить успешное проектирование и разработку программного средства для управления спортивными мероприятиями и турнирами.

1. МОДЕЛИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ
   1. Функциональная модель программного средства

Функциональная модель программного средства представлена в виде диаграммы вариантов использования и информационной модели предметной области. Варианты использования отражают функциональность системы в ответ на внешние воздействия с точки зрения получения значимого результата для пользователей.

**2.1.1** Варианты использования программного средства

Исходя из выдвинутых в подразделе 1.3 требований, проектируемое программное средство предполагает поддержку системы ролей, т.е. разделение пользователей на различные роли, а следовательно, и выдачу различных прав нескольким категориям пользователей: администратору, организатору мероприятия или турнира, участнику и гостю.

Возможности пользователей системы представлены на рисунке 2.1 в виде диаграммы вариантов использования, разработанной с использованием нотации UML 2.1 [11].

Регистрация, аутентификация, авторизация, просмотр общедоступных результатов и новостей – функции, доступные для роли «Гость» (не зарегистрированный пользователь). Аутентификация выполняется через логин и пароль или внешние сервисы авторизации.

После регистрации пользователю назначается одна из следующих ролей: «Участник», «Организатор» или «Администратор».

Функции участника:

* Редактирование личной информации (ФИО, дата рождения, контактные данные);
* Регистрация на турнир или спортивное мероприятие;
* Просмотр информации о соревнованиях;
* Получение уведомлений о турнирах и изменениях в расписании;
* Просмотр своих результатов и достижений.

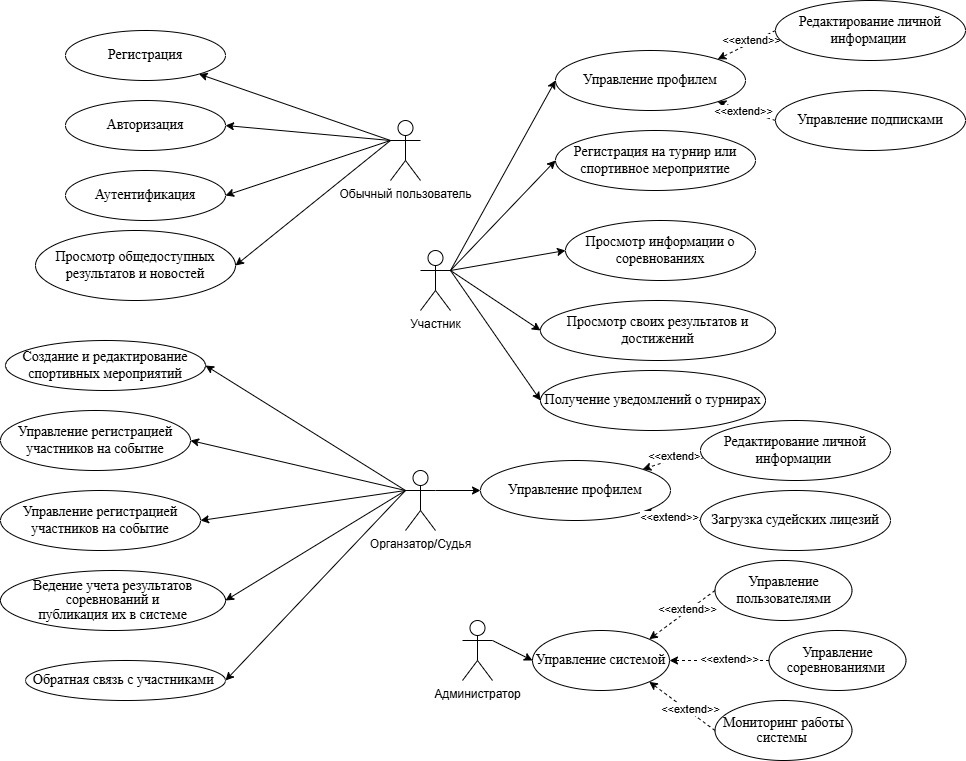


Рисунок 2.1 – Диаграмма вариантов использования ПС

Функции организатора:

* Создание и редактирование спортивных мероприятий;
* Управление регистрацией участников (подтверждение, отклонение заявок);
* Формирование турнирных сеток и расписания соревнований;
* Ведение учета результатов соревнований и публикация их в системе;
* Обратная связь с участниками (уведомления, комментарии, новости).

Функции администратора:

* Управление пользователями системы (блокировка, изменение ролей);
* Управление соревнованиями (удаление, изменение параметров);
* Мониторинг работы системы и логирование событий.

**2.1.2** Разработка инфологической модели базы данных

Предметная область программного средства включает в себя следующие сущности и их атрибуты:

а) Пользователь:

* ID пользователя;
* Имя;
* Фамилия;
* Электронная почта;
* Роль (Участник, Организатор, Администратор);
* Дата регистрации;

б) Соревнование:

* ID соревнования;
* Описание;
* Дата и время проведения;
* Место проведения;
* Организатор;
* Список участников.

в) Турнирная сетка:

* ID турнира;
* Тип сетки (одинарная, двойная);
* Список матчей;
* Результаты матчей.

г) Матч:

* ID матча;
* Соревнование;
* Участники;
* Дата и время;
* Результат.

**2.2 Разработка спецификации функциональных требований**

Спецификация функциональных требований описывает основные функции программного средства с учетом внутренней организации системы и взаимодействия с пользователями.

**2.2.1** Функция регистрации

Функция регистрации должна быть реализована с учетом следующих требований:

1. Регистрация доступна для новых пользователей;
2. Требуется ввод электронной почты, пароля и имени;
3. Возможность авторизации через внешние сервисы;
4. Подтверждение почты для активации аккаунта.

**2.2.2** Функция аутентификации

Функция аутентификации должна быть реализована с учетом следующих требований:

1. Вход в систему с использованием логина и пароля;
2. Поддержка OAuth 2.0 для входа через соцсети;
3. Восстановление пароля через почту.

**2.2.3** Функция создания соревнования

Данная функция должна быть реализована с учетом следующих требований:

1. Доступна пользователям с ролью «Организатор»;
2. Заполнение информации: название, дата, место, тип соревнования;
3. Добавление участников вручную или по заявкам;
4. Определение правил и формата соревнования.

**2.2.4** Функция регистрации на соревнование

Данная функция должна быть реализована с учетом следующих требований:

1. Доступна пользователям с ролью «Участник»;
2. Выбор доступного соревнования из каталога;
3. Заполнение заявки на участие;
4. Ожидание подтверждения от организатора.

**2.2.5** Функция формирования турнирной сетки

Данная функция должна быть реализована с учетом следующих требований:

1. Автоматическая генерация турнирной сетки по заданным параметрам;
2. Возможность ручного редактирования сетки организатором;
3. Отображение расписания матчей.

**2.2.6** Функция внесения результатов

Данная функция должна быть реализована с учетом следующих требований:

1. Доступна организатору и судьям;
2. Ввод результатов матчей в систему;
3. Автоматическое обновление турнирной таблицы;
4. Отправка уведомлений участникам о результатах.

**2.2.7** Функция оповещений

Данная функция должна быть реализована с учетом следующих требований:

1. Отправка уведомлений об изменениях в расписании;
2. Уведомления о старте соревнования;
3. Напоминания о предстоящих матчах.

Разработанная функциональная модель и спецификация требований обеспечивают всестороннюю поддержку организации спортивных мероприятий, включая управление пользователями, регистрацией, соревнованиями и результатами. Благодаря четкому разделению ролей и продуманной архитектуре системы, программное средство позволит эффективно проводить турниры, повышая вовлеченность участников и организаторов.

3 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Разработка архитектуры программного средства

Как было указано в пункте 1.3, на основании проведенного анализа требований и особенностей взаимодействия участников спортивных мероприятий, а также обзора существующих решений, было принято решение использовать в качестве платформы реализации веб-приложение с клиент-серверной архитектурой.

Для создания надежного, расширяемого и легко сопровождаемого решения была выбрана трехуровневая архитектура, разделяющая приложение на:

* клиентский уровень (frontend);
* серверный уровень (backend);
* уровень хранения данных (база данных).

В качестве основы клиентской части было выбрано одностраничное приложение (SPA), реализованное с помощью библиотеки React. Это позволяет достичь высокой отзывчивости пользовательского интерфейса и предоставить удобный UX, аналогичный нативным desktop-приложениям. SPA-приложение взаимодействует с серверной частью через REST API, что соответствует современным стандартам веб-разработки.

Серверная часть реализована на платформе Spring Boot с использованием JWT-аутентификации (JSON Web Token) для обеспечения безопасности и разграничения прав доступа пользователей. Сервер отвечает за:

* обработку бизнес-логики;
* управление ролями (администратор, организатор, участник);
* выдачу токенов и проверку аутентификации;
* обработку и валидацию запросов;
* формирование ответов клиенту.

Хранение и управление структурированными данными осуществляется в реляционной базе данных PostgreSQL, которая обеспечивает надежность, поддерживает сложные связи между сущностями (например, между турнирами, командами, расписаниями матчей и результатами), а также масштабируется при увеличении объема данных.

На рисунке 3.1 представлена схема архитектуры программного средства:

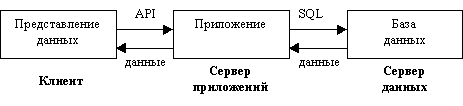


Рисунок 3.1 – Общая схема архитектуры программного средства

Использование SPA с отделением клиентской логики от серверной позволяет достичь следующих преимуществ:

* загрузка данных с сервера только по необходимости, что снижает объем трафика;
* быстрая реакция интерфейса без перезагрузки страниц;
* независимая разработка frontend и backend;
* расширяемость архитектуры за счет четкого разграничения слоев;
* высокая безопасность при использовании JWT (токены не передаются в открытом виде, можно легко реализовать механизмы авторизации и блокировки доступа).

Кроме того, при необходимости, систему можно легко масштабировать, например, заменив монолитную серверную архитектуру на микросервисную или добавив внешние хранилища для мультимедийных данных, таких как фотографии команд и видеозаписи матчей. Таким образом, архитектурное решение было основано на лучших практиках современной веб-разработки и соответствует целям и задачам, поставленным в рамках дипломного проекта.

**3.2 Разработка даталогической и физической базы данных**

Проектируемая система для организации и сопровождения спортивных мероприятий и турниров должна обеспечивать хранение большого количества структурированных данных, включающих пользователей, команды, турниры, расписания матчей, чаты, форумы, а также вспомогательную информацию: комментарии, новости, результаты матчей, магазины болельщиков и т.д.

**3.2.1** Выбор системы управления базами данных

В качестве системы управления базами данных была выбрана PostgreSQL — объектно-реляционная СУБД с открытым исходным кодом, которая обеспечивает высокую степень совместимости с SQL-стандартом, поддержку транзакций, индексов, ограничений целостности, триггеров и богатый набор типов данных. PostgreSQL обладает широкими возможностями для масштабируемых веб-приложений и отлично сочетается со стеком Spring Boot, используемым на серверной части приложения.

**3.2.2** Даталогическая модель

На этапе даталогического проектирования были определены основные сущности системы, а также связи между ними. Основные таблицы базы данных можно сгруппировать по следующим функциональным блокам:

Пользователи и безопасность

1. users — хранит базовую информацию о пользователях системы: имя, фамилия, логин, email, дата регистрации.
2. user\_profiles — расширенная информация о пользователе: биография, аватар, местоположение и т.п.
3. security\_credentials — данные для аутентификации и авторизации, включая логин, хешированный пароль, роль пользователя и флаги верификации.

Команды и игроки

1. teams — информация о спортивных командах: название, страна, город, достижения, тип команды.
2. players — данные об игроках команд, включая имя, номер, титулы.
3. coaches — тренеры команд, с привязкой к team\_id, и с биографией.
4. l\_player\_coach — таблица для связей «многие ко многим» между игроками и тренерами (один игрок может тренироваться у нескольких тренеров, и наоборот).

Турниры и матчи

1. match\_schedule — расписание матчей с указанием даты, локации, домашних и гостевых команд, количества доступных билетов.
2. match\_results — результаты матчей, победитель, описание и финальный счёт.
3. stadiums — стадионы, где проводятся мероприятия, с указанием вместимости и местоположения.

Социальное взаимодействие

1. chats, messages, l\_users\_chats — система общения пользователей: чаты, сообщения (типизированные: JOIN, LEAVE, CHAT), привязка пользователей к чатам.
2. forum, post, comments — форум для общения и публикации материалов. Пользователи могут создавать посты и комментировать их.
3. fan\_club\_members — таблица, фиксирующая участие пользователей в фан-клубах команд.

Новости и магазин

1. news — новости, связанные с командами, событиями, турнирами.
2. shop — корзина болельщика, используется для оформления заявок на матчи или приобретения товаров.
3. l\_users\_teams — таблица «многие ко многим» между пользователями и командами, определяет участников или болельщиков команды.

Таким образом, даталогическая модель построена по нормализованному принципу с минимизацией избыточности и с соблюдением связей между сущностями. На рисунке 3.3 представлена схема взаимосвязей основных таблиц базы данных (ER-диаграмма).

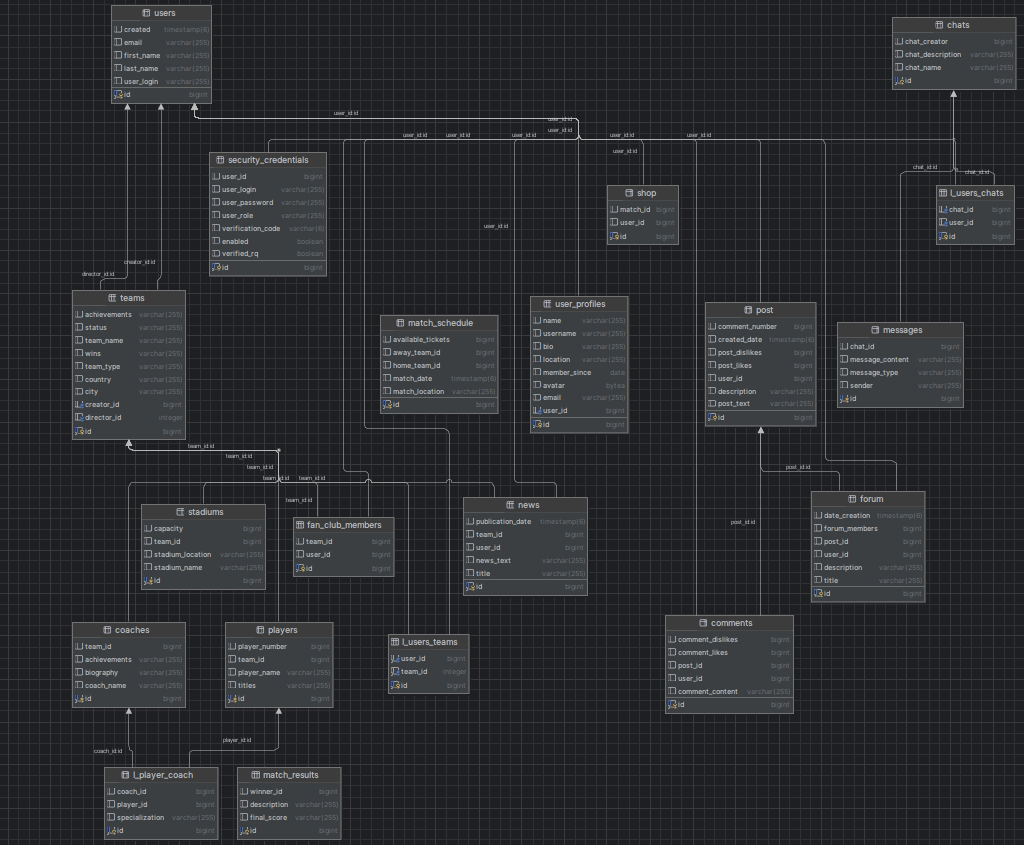


Рисунок 3.3 – ER-диаграмма базы данных системы спортивных мероприятий

**3.2.3** Физическая реализация базы данных

Физическая реализация модели выполнена с использованием SQL-скриптов создания таблиц (DDL). Ниже приведены основные особенности реализации:

1. Тип идентификаторов: все первичные ключи имеют тип bigint, что позволяет масштабировать систему до миллионов записей.
2. Связи: для организации связей используются внешние ключи с явными ограничениями (foreign key), обеспечивая логическую целостность данных.
3. Значения некоторых полей ограничены перечислениями, например, user\_role может принимать только значения ADMIN или USER, а message\_type — только CHAT, JOIN, LEAVE.
4. Хранилище бинарных данных: аватары пользователей хранятся в поле avatar типа bytea, что характерно для PostgreSQL.
5. Поля времени: даты создания записей, публикаций и т.п. представлены типом timestamp(6), обеспечивающим точность до микросекунд.

Пример создания таблицы users:

create table if not exists public.users(

created timestamp(6),

id bigint not null primary key,

email varchar(255),

first\_name varchar(255),

last\_name varchar(255),

user\_login varchar(255)

);

В DDL-коде обеспечена консистентность всех сущностей, продумана структура с возможностью масштабирования и будущего расширения: например, благодаря связующим таблицам можно расширять связи (многие-ко-многим), а также добавлять бизнес-логику без необходимости реорганизации всей базы.

**3.2.4** Особенности использования PostgreSQL

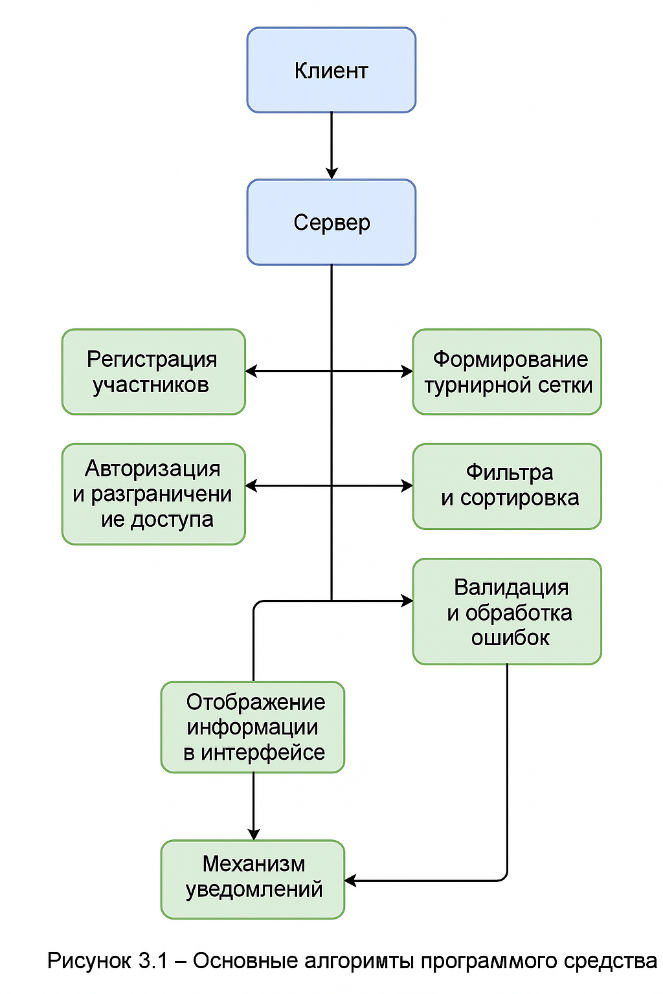
Использование PostgreSQL предоставляет следующие преимущества:

1. Надёжность и стабильность работы при высоких нагрузках.
2. Поддержка расширений, например, PostGIS (географические координаты) или полнотекстовый поиск.
3. Возможность реализации хранимых процедур и триггеров для бизнес-логики, если потребуется её вынести на уровень базы данных.
4. Нативная поддержка транзакций и ограничений целостности данных.

Разработанная даталогическая и физическая модель базы данных обеспечивает полное покрытие требований к хранилищу информации для приложения по организации спортивных мероприятий. Структура базы данных легко расширяется, масштабируется и безопасна для хранения пользовательских и операционных данных, полностью интегрируясь с backend-частью приложения на Spring Boot.

3.3 Проектирование и разработка алгоритмов программного средства

Разработка программного средства организации спортивных мероприятий и турниров требует чётко структурированной архитектуры и эффективной реализации ключевых алгоритмов, обеспечивающих корректную работу системы. В данном разделе рассматриваются алгоритмы, реализованные как на клиентской части (React), так и на серверной части (Spring Boot), а также их взаимодействие. Особое внимание уделено алгоритмам управления турнирами, регистрацией участников, расчёту результатов, а также безопасному доступу к данным и проверке ролей пользователей.



**3.3.1** Архитектура взаимодействия клиента и сервера

Программное средство построено по принципу клиент-серверной архитектуры, где фронтенд (React) обращается к REST API, реализованному на Spring Boot. Между клиентом и сервером происходит обмен данными в формате JSON.

Ключевые принципы взаимодействия:

1. Каждый запрос с фронтенда сопровождается JWT-токеном, удостоверяющим пользователя.
2. Сервер реализует слои контроллеров, сервисов и репозиториев (по шаблону Service/Repository).
3. Сервер обрабатывает бизнес-логику (регистрация турниров, формирование турнирной сетки, подсчёт результатов), а клиент отображает полученные данные и предоставляет интерфейс взаимодействия.

**3.3.2** Алгоритм регистрации участников на турнир

Регистрация реализована с учетом валидации уникальности пользователя в турнире и проверки текущего статуса турнира.

Пошаговый алгоритм регистрации:

1. Пользователь авторизуется в системе и получает JWT.
2. При открытии страницы турнира клиент отправляет запрос на /api/tournaments/{id}/details.
3. Сервер возвращает подробности о турнире, включая текущий статус (OPEN, IN\_PROGRESS, FINISHED).
4. Если статус OPEN, клиент отображает кнопку «Зарегистрироваться».
5. При нажатии кнопки отправляется POST-запрос /api/tournaments/{id}/register.
6. Сервер выполняет проверку:
7. Авторизован ли пользователь.
8. Участвовал ли он ранее.
9. Не превышен ли лимит участников.
10. Если все условия выполнены, создаётся запись участника и возвращается сообщение об успешной регистрации.

**3.3.3** Алгоритм формирования турнирной сетки

После завершения регистрации администратор может инициировать автоматическое формирование сетки матчей. Алгоритм зависит от выбранного формата турнира (круговая система или олимпийская система).

Пример алгоритма для олимпийской системы:

1. Из базы данных извлекается список зарегистрированных участников.
2. Участники случайным образом перемешиваются.
3. Создаются пары (матчи) по следующему принципу: Участник 1 против Участника 2, Участник 3 против Участника 4 и т.д.
4. В случае нечётного количества участников добавляется bye (свободный проход в следующий раунд).
5. Созданные матчи сохраняются в таблицу matches с полем round = 1.
6. По завершении каждого раунда победители автоматически формируют новые пары, пока не останется один победитель.

**3.3.4** Алгоритм подсчёта результатов

Подсчёт результатов проводится по завершении каждого матча. Результаты вводятся судьёй или администратором, после чего сервер обновляет состояние матча и участника.

Алгоритм обработки результата:

1. Пользователь с ролью ADMIN или REFEREE вводит счёт матча.
2. С фронтенда отправляется PATCH-запрос /api/matches/{id}/result.

Сервер:

1. Проверяет наличие прав у отправителя.
2. Устанавливает результат (победитель, очки и т.д.).
3. Обновляет статус матча на FINISHED.
4. Если все матчи текущего раунда завершены — запускается генерация следующего раунда.

**3.3.5** Алгоритм фильтрации и сортировки турниров

Пользователю предоставляется интерфейс с фильтрами по дате, типу турнира, статусу. Запросы с фильтрами передаются на сервер через query-параметры. Пример запроса:

GET /api/tournaments?status=OPEN&type=football&sortBy=date

Алгоритм обработки:

1. Контроллер получает параметры фильтрации.
2. Сервис формирует спецификацию (criteria query) на основе полученных параметров.
3. Репозиторий выполняет запрос к базе данных.
4. Результаты возвращаются клиенту в виде JSON и отображаются в таблице с пагинацией.

**3.3.6** Алгоритм авторизации и разграничения прав

Система использует Spring Security с JWT-авторизацией. На каждый защищённый запрос проверяется JWT, и на его основе определяется роль пользователя (ADMIN, PLAYER, REFEREE).

Пример алгоритма проверки доступа:

1. Пользователь логинится и получает JWT.
2. JWT хранится в localStorage и передаётся в заголовке Authorization: Bearer.
3. На сервере фильтр безопасности (JwtAuthenticationFilter) извлекает токен, проверяет его валидность и аутентифицирует пользователя.
4. Аннотации типа @PreAuthorize("hasRole('ADMIN')") обеспечивают доступ к соответствующим API.

**3.3.7** Алгоритм отображения информации в клиентском интерфейсе

Клиентская часть построена на компонентах React. Каждая страница (например, список турниров, детали турнира, результаты матчей) загружается асинхронно через fetch или axios, и управляется внутренним состоянием компонентов.

Алгоритм отображения списка турниров:

1. Компонент TournamentList монтируется.
2. Выполняется useEffect(() => fetchTournaments(), []).
3. Полученные турниры сохраняются в useState.
4. Таблица заполняется строками, каждая из которых является дочерним компонентом.

**3.3.8** Валидация и обработка ошибок

Для обеспечения корректности данных реализована двухуровневая валидация:

1. На клиенте — через формы (Formik, Yup) и интерфейсные ограничения.
2. На сервере — через аннотации @Valid, @NotNull, @Size и кастомные валидаторы.
3. Ошибки от сервера отображаются в удобном виде на клиенте, а сообщения локализованы для поддержки мультиязычности.

**3.3.9** Механизмы уведомлений

Для уведомления участников о предстоящих матчах и изменениях реализован механизм push-уведомлений через WebSocket.

Алгоритм уведомления:

1. Клиент устанавливает WebSocket-соединение с /ws/notifications.
2. Сервер по событию (например, назначение матча) отправляет сообщение через SimpMessagingTemplate.
3. Клиент принимает сообщение и отображает уведомление с помощью Toast.

Таким образом, алгоритмы, реализованные в программном средстве, охватывают полный цикл работы с турнирами — от регистрации до подсчёта результатов. Каждый алгоритм разрабатывался с учётом масштабируемости, безопасности и удобства пользователя. Использование современных технологий (React, Spring Boot, JWT, WebSocket) позволило создать устойчивую и гибкую архитектуру, удовлетворяющую требованиям автоматизации спортивных мероприятий.

**3.4 Развертывание программного средства**

После завершения проектирования всех компонентов программного средства возникает задача его развертывания в производственной среде. Для обеспечения корректной и стабильной работы системы необходимо определить аппаратно-программные комплексы, используемые для хостинга различных компонентов приложения, а также способы их взаимодействия.

Визуализация структуры развертывания представлена на диаграмме развертывания (рисунок 3.12).

Диаграмма отражает следующие ключевые особенности:

1. Клиентская часть программного средства разрабатывалась с использованием технологии React и предназначена для запуска в любом современном браузере. Это позволяет конечному пользователю взаимодействовать с системой с различных типов устройств — ПК, ноутбуков, планшетов и смартфонов, не устанавливая дополнительное программное обеспечение.
2. Серверная часть, реализованная с помощью фреймворка Spring Boot, размещается на отдельном облачном сервере и предоставляет REST API для клиентской части. Коммуникация между фронтендом и бэкендом осуществляется по протоколу HTTPS, что обеспечивает безопасность передачи данных.
3. Система управления базами данных PostgreSQL развёрнута отдельно и подключена к серверной части приложения. Это позволяет централизованно хранить и обрабатывать информацию о турнирах, командах, матчах, пользователях и других объектах предметной области.
4. Сетевое взаимодействие между клиентской частью и серверной частью осуществляется через глобальную сеть Интернет. Учитывая распределённую природу приложения, предполагается, что пользователи могут находиться на значительном удалении от серверов системы.

Для размещения и обслуживания компонентов программного средства планируется использовать облачные технологии. В частности:

1. Фронтенд развёртывается с использованием сервиса Firebase Hosting, обеспечивающего стабильную доставку клиентской части пользователям по всему миру с помощью CDN;
2. Бэкенд и база данных размещаются на облачных виртуальных машинах, например, на платформе Heroku, Render, Railway или аналогичных, поддерживающих развёртывание Spring-приложений и хостинг PostgreSQL.

Для корректной работы системы в связке с Firebase необходимо зарегистрировать новый проект в Firebase Console и получить параметры конфигурации:

* projectId – уникальный идентификатор проекта;
* apiKey – ключ для обращения к API Firebase;
* authDomain – домен для авторизации пользователей;
* databaseURL – URL-адрес для обращения к базам данных (при использовании Realtime Database);
* messagingSenderId и appId – параметры, необходимые для облачных уведомлений и инициализации клиента.

Эти данные прописываются в конфигурационном файле клиентской части (например, firebase-config.js), что обеспечивает интеграцию фронтенда с соответствующими сервисами Firebase, включая хостинг и, при необходимости, авторизацию.

Таким образом, структура развертывания программного средства обеспечивает его масштабируемость, доступность и надёжность при минимальных затратах на обслуживание инфраструктуры.

**4** **ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА**

Тестирование программного обеспечения — это процесс анализа программного продукта с целью выявления дефектов, проверки соответствия функциональности требованиям и обеспечения качества работы системы. Этот процесс является неотъемлемой частью жизненного цикла разработки и начинается с ранних этапов проектирования. Опыт показывает, что чем позже обнаружена ошибка, тем дороже её устранение, поэтому своевременное тестирование играет ключевую роль в создании надежного программного продукта.

В рамках разработки программного средства для организации спортивных мероприятий и турниров проводилось как ручное, так и автоматизированное тестирование. Особое внимание уделялось функциональному тестированию, направленному на проверку корректной работы пользовательских сценариев: регистрация и авторизация пользователей, создание и управление спортивными мероприятиями, просмотр расписания, участие в турнирах, а также взаимодействие между организаторами и участниками.

Для проверки соответствия реализованной функциональности поставленным требованиям был составлен набор тест-кейсов, охватывающих основные пользовательские сценарии. Также были реализованы юнит-тесты для серверной логики на стороне backend-приложения (Spring Boot), а для frontend-компонентов на React проведено динамическое тестирование поведения интерфейсов в реальных условиях использования.

Помимо модульного и интеграционного тестирования, была проведена ручная проверка устойчивости системы к ошибочным входным данным (например, пустые поля, некорректные форматы дат, несуществующие ID мероприятий), а также реализованы автоматизированные UI-тесты с использованием библиотеки Selenium WebDriver для основных сценариев взаимодействия пользователя с интерфейсом приложения.

Ниже представлены ключевые тест-кейсы, проведенные в процессе тестирования.

**Таблица 4.1 – Тест-кейс регистрации с некорректными данными**

| **Тест** | **Ожидаемый результат** | **Результат** |
| --- | --- | --- |
| Попытка регистрации с несовпадающими паролями | Появляется сообщение об ошибке "Пароли не совпадают" | Успех |

**Таблица 4.2 – Тест-кейс успешной регистрации пользователя**

| **Тест** | **Ожидаемый результат** | **Результат** |
| --- | --- | --- |
| Регистрация нового пользователя | Появляется сообщение "Регистрация прошла успешно" | Успех |

**Таблица 4.3 – Тест-кейс авторизации несуществующего пользователя**

| **Тест** | **Ожидаемый результат** | **Результат** |
| --- | --- | --- |
| Вход с неверными данными | Появляется сообщение "Пользователь не найден" | Успех |

**Таблица 4.4 – Тест-кейс успешной авторизации**

| **Тест** | **Ожидаемый результат** | **Результат** |
| --- | --- | --- |
| Вход существующего пользователя с правильным паролем | Переход в личный кабинет, приветственное сообщение | Успех |

**Таблица 4.5 – Тест-кейс создания нового турнира**

| **Тест** | **Ожидаемый результат** | **Результат** |
| --- | --- | --- |
| Создание турнира с заполненными полями | Турнир отображается в списке мероприятий | Успех |

**Таблица 4.6 – Тест-кейс регистрации участника на турнир**

| **Тест** | **Ожидаемый результат** | **Результат** |
| --- | --- | --- |
| Регистрация на выбранный турнир | Отображается сообщение "Вы зарегистрированы на турнир" | Успех |

**Таблица 4.7 – Тест-кейс редактирования информации о событии**

| **Тест** | **Ожидаемый результат** | **Результат** |
| --- | --- | --- |
| Изменение названия турнира | Новое название отображается в списке турниров | Успех |

**Таблица 4.8 – Тест-кейс отображения расписания турниров**

| **Тест** | **Ожидаемый результат** | **Результат** |
| --- | --- | --- |
| Просмотр расписания в личном кабинете | Расписание отображается корректно, с разбивкой по датам и времени | Успех |

**Таблица 4.9 – Тест-кейс поиска турниров по фильтрам**

| **Тест** | **Ожидаемый результат** | **Результат** |
| --- | --- | --- |
| Поиск по виду спорта и дате | Отображаются только соответствующие турниры | Успех |

**Таблица 4.10 – Тест-кейс проверки стабильности при недоступности сервера**

| **Тест** | **Ожидаемый результат** | **Результат** |
| --- | --- | --- |
| Попытка авторизации при отключенном сервере | Сообщение "Ошибка соединения с сервером" | Успех |

Эффективное тестирование серверной части является важным этапом при разработке программного средства для организации спортивных мероприятий и турниров. Тестирование позволяет убедиться в корректности бизнес-логики, обработке исключений, взаимодействии с базой данных, контроллерами REST API и другими компонентами.

В процессе тестирования серверной части данного программного средства использовалась библиотека JUnit 5, являющаяся современным и гибким фреймворком для модульного тестирования на платформе Java. JUnit 5 был выбран по следующим причинам:

1. Поддержка современных возможностей Java (включая аннотации, лямбды, динамическое выполнение тестов).
2. Расширяемость за счёт механизма плагинов (Extension API).
3. Улучшенная архитектура по сравнению с предыдущими версиями (JUnit 4).
4. Интеграция с Maven, Gradle и CI/CD-инструментами (например, GitHub Actions, Jenkins).
5. Поддержка Spring Boot и аннотаций @SpringBootTest, @DataJpaTest и т.д.

JUnit 5 состоит из следующих основных модулей:

1. JUnit Platform – обеспечивает запуск тестов и взаимодействие с внешними инструментами (IDE, Gradle, Maven и др.).

2. JUnit Jupiter – основная API и механизм аннотаций для написания тестов.

3. JUnit Vintage – обеспечивает обратную совместимость с тестами, написанными на JUnit 3 и JUnit 4.

В проекте использовался модуль JUnit Jupiter, позволяющий писать тесты с помощью таких аннотаций, как @Test, @BeforeEach, @AfterEach, @BeforeAll, @AfterAll, @DisplayName, @Nested, @ParameterizedTest и др.

Тестирование было разделено на следующие уровни:

1. Модульное тестирование – проверка отдельных компонентов: сервисов, репозиториев, утилит.
2. Интеграционное тестирование – проверка взаимодействия между слоями: REST API ↔ Сервис ↔ БД.
3. Тестирование с использованием Spring Boot – запуск приложения в тестовом окружении с аннотацией @SpringBootTest.

В ходе тестирования применялись ключевые аннотации JUnit 5:

@Test — основной маркер для объявления тестовых методов.

@DisplayName("...")— используется для задания читаемого имени теста.

@BeforeEach и @AfterEach — выполняются перед и после каждого теста соответственно.

@BeforeAll и @AfterAll — выполняются один раз до начала и после завершения всех тестов в классе (должны быть static).

@Nested — позволяет группировать тесты внутри вложенных классов.

@Disabled — временно отключает тесты.

@DisplayName("Тесты для сервиса управления турнирами")

class TournamentServiceTest {

@Autowired

private TournamentService tournamentService;

@BeforeEach

void setup() {

// инициализация моков или предварительная настройка

}

@Test

@DisplayName("Создание нового турнира")

void testCreateTournament() {

TournamentDto dto = new TournamentDto("Spring Cup", LocalDate.now(), "Минск");

Tournament result = tournamentService.createTournament(dto);

assertNotNull(result.getId());

assertEquals("Spring Cup", result.getName());

}

@Test

@DisplayName("Проверка исключения при передаче null")

void testCreateTournamentNull() {

assertThrows(IllegalArgumentException.class, () -> {

tournamentService.createTournament(null);

});

}

}

Для интеграционного тестирования использовалась аннотация @SpringBootTest, которая поднимает контекст Spring-приложения полностью, включая все компоненты, репозитории и автоконфигурации. Такой подход позволял проводить полноценные end-to-end проверки:

@SpringBootTest

class TournamentIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Test

void testGetAllTournaments() throws Exception {

mockMvc.perform(get("/api/tournaments"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.length()").isNotEmpty());

}

}

Также применялись:

@AutoConfigureMockMvc – для инициализации MockMvc без запуска веб-сервера.

@DataJpaTest – для тестирования слоя JPA с использованием H2-базы данных.

@DataJpaTest

class TournamentRepositoryTest {

@Autowired

private TournamentRepository repository;

@Test

void testSaveAndFindTournament() {

Tournament tournament = new Tournament("Winter Cup", LocalDate.of(2025, 2, 1), "Гомель");

repository.save(tournament);

Optional<Tournament> found = repository.findByName("Winter Cup");

assertTrue(found.isPresent());

assertEquals("Гомель", found.get().getLocation());

}

}

Для тестов использовалась встроенная база данных H2. Это позволило:

1. Изолировать тесты от основной базы данных PostgreSQL;
2. Обеспечить быстрое выполнение тестов;
3. Легко конфигурировать схемы через schema.sql и data.sql.
4. H2 подключалась автоматически при использовании @DataJpaTest или с помощью настройки профиля application-test.yml.

JUnit 5 предоставляет расширенный набор методов для проверки:

* assertEquals(expected, actual);
* assertTrue(condition);
* assertFalse(condition);
* assertThrows(exception.class, lambda);
* assertAll(...) – для группировки проверок;
* assertTimeout(...) – для проверки времени выполнения.

JUnit 5 позволяет запускать тест с разными входными данными с помощью @ParameterizedTest:

@ParameterizedTest

@ValueSource(strings = {"Spring Cup", "Summer League", "Autumn Games"})

void testTournamentNameNotEmpty(String name) {

assertFalse(name.isEmpty());

}

Также доступны источники данных:

@CsvSource, @CsvFileSource;

@MethodSource – генерация из метода;

@EnumSource – тестирование с перечислениям.

JUnit 5 предоставляет механизм расширений, позволяющий реализовать:

* логирование;
* установку окружения;
* автоматическую инициализацию моков (например, с помощью Mockito);
* перехват исключений и состояния тестов.

Наиболее часто применяемые расширения:

* @ExtendWith(MockitoExtension.class)
* @TestInstance(Lifecycle.PER\_CLASS)

JUnit 5 подключался в проект с помощью Maven:

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

В случае интеграции с Spring Boot, дополнительные зависимости подключались автоматически через spring-boot-starter-test.

Тесты автоматически запускались при каждом коммите в репозиторий. Для этого использовался GitHub Actions, где был настроен рабочий процесс с фазами:

Сборка проекта (mvn clean install)

Запуск тестов (mvn test)

Использование JUnit 5 обеспечило гибкость, масштабируемость и читаемость тестов серверной части. Благодаря продуманной архитектуре и поддержке Spring Boot, тесты позволили на ранней стадии выявлять ошибки, снижать технический долг и повышать надёжность разрабатываемого программного средства.

Результаты проведенного тестирования подтверждают корректную реализацию основных функций программного средства, устойчивость к ошибочным данным, а также удобство использования интерфейса для различных ролей пользователей. Отсутствие критических ошибок и успешное прохождение ключевых сценариев позволяют говорить о работоспособности и готовности программного продукта к использованию в реальных условиях.

5 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

Разработанное программное средство представляет собой современное веб-приложение, предназначенное для организации, управления и проведения спортивных мероприятий и турниров различного масштаба. Оно реализовано на основе клиент-серверной архитектуры, где клиентская часть функционирует в веб-браузере, обеспечивая кроссплатформенность и доступность без необходимости установки, а серверная часть обрабатывает бизнес-логику, хранение и обработку данных.

Клиентская часть приложения реализована с использованием JavaScript-библиотеки React, что обеспечивает высокую интерактивность интерфейса, поддержку компонентного подхода и масштабируемость. Для хостинга фронтенда применяется облачный сервис Firebase Hosting, благодаря которому пользователи могут получить быстрый и безопасный доступ к приложению в любой точке мира. Регистрация проекта на Firebase производится через веб-интерфейс, где после создания проекта автоматически генерируются параметры конфигурации: ключ API, URL-адрес проекта, параметры доступа к базе данных и ключи авторизации. Эти параметры необходимо скопировать и вставить в файл конфигурации проекта (firebase-config.js), чтобы обеспечить корректную интеграцию с облачной платформой.

Для локального запуска фронтенда необходимо предварительно установить среду Node.js, после чего с помощью встроенного пакетного менеджера npm выполняется установка всех зависимостей. После этого можно запустить приложение в режиме разработки командой npm start. На экране пользователя отобразится стартовая страница приложения, с которой начинается взаимодействие с системой.

При первом входе в приложение пользователь попадает на стартовую страницу, на которой отображается краткое описание платформы, список ближайших турниров и кнопки для регистрации и авторизации. Чтобы создать учётную запись, пользователь нажимает кнопку «Регистрация» и переходит на соответствующую страницу, представленную на рисунке 5.1. Здесь требуется ввести имя пользователя, адрес электронной почты и придумать пароль.

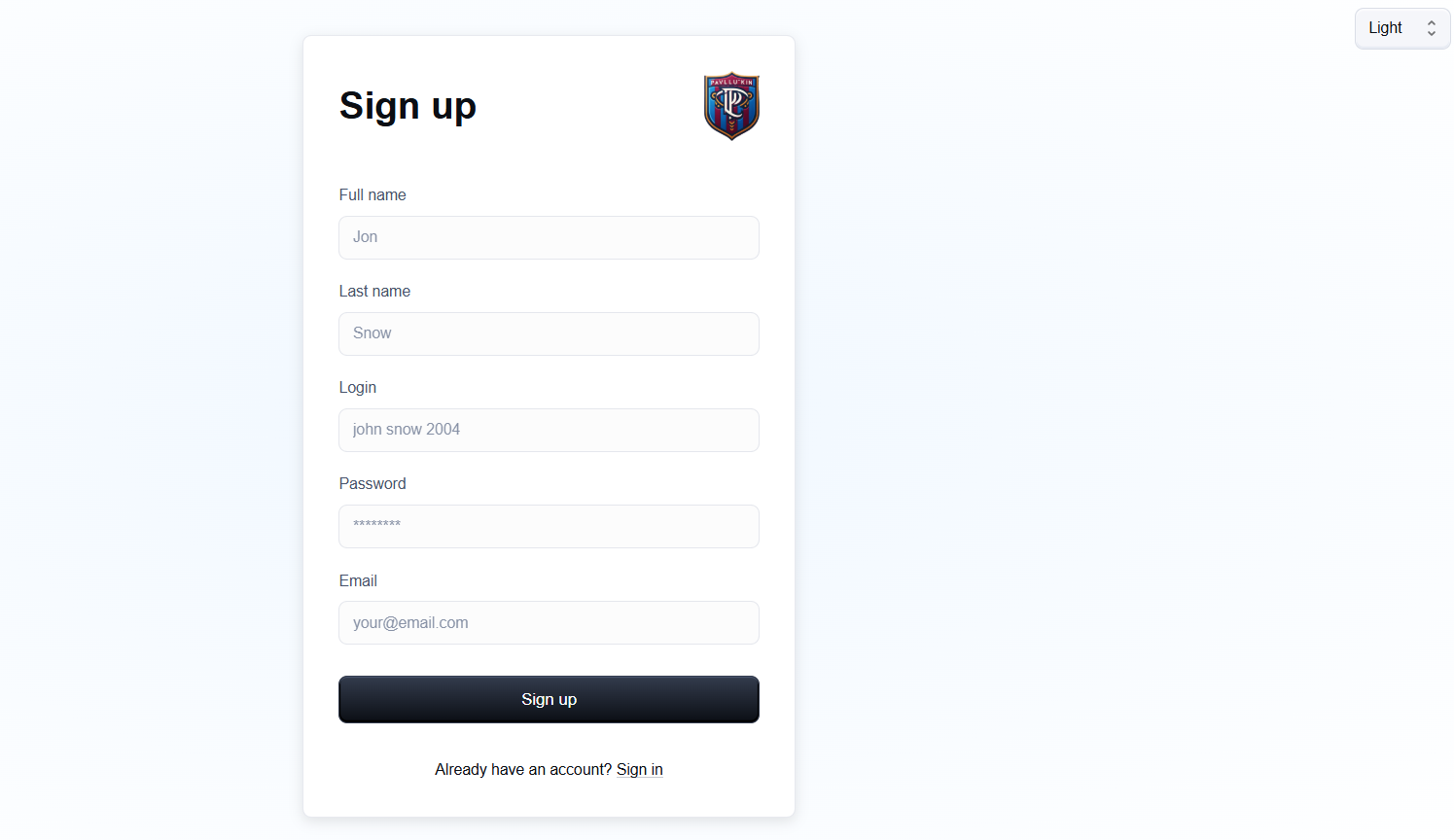


Рисунок 5.1 –– Страница регистрации нового пользователя

После подтверждения формы на указанный email отправляется письмо с кодом верификации. Этот код пользователь должен ввести в отдельное поле на странице подтверждения, что гарантирует подлинность email-адреса и предотвращает автоматические регистрации, изображенной на рисунок 5.2.

После успешной регистрации пользователь перенаправляется на страницу авторизации, представленную на рисунке 5.3, где ему необходимо ввести логин (или email) и пароль. В случае корректных данных пользователь попадает в своё персональное пространство.

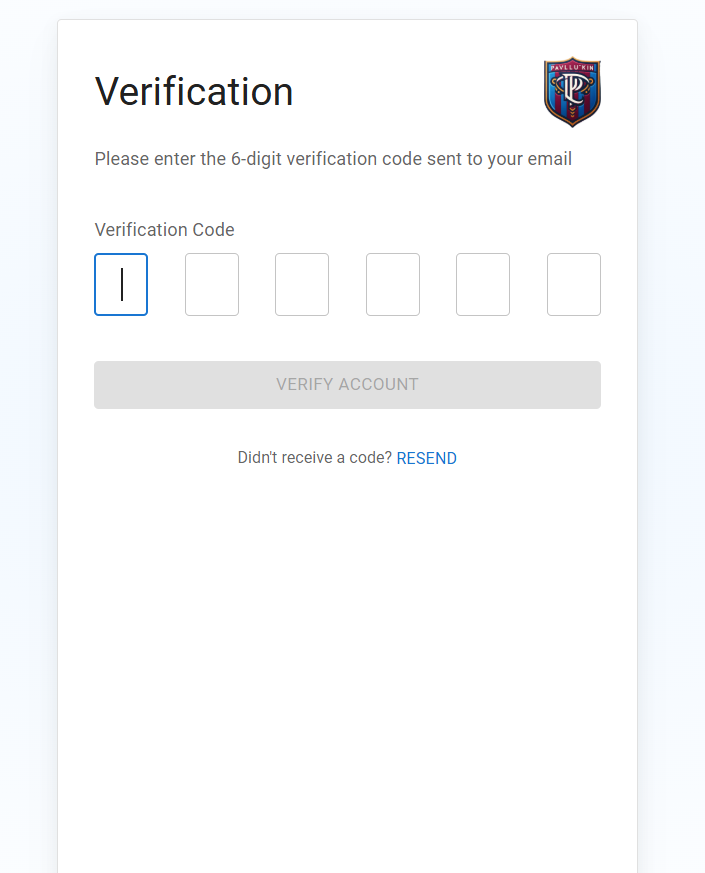


Рисунок 5.2 – Подтверждение регистрации через email-код

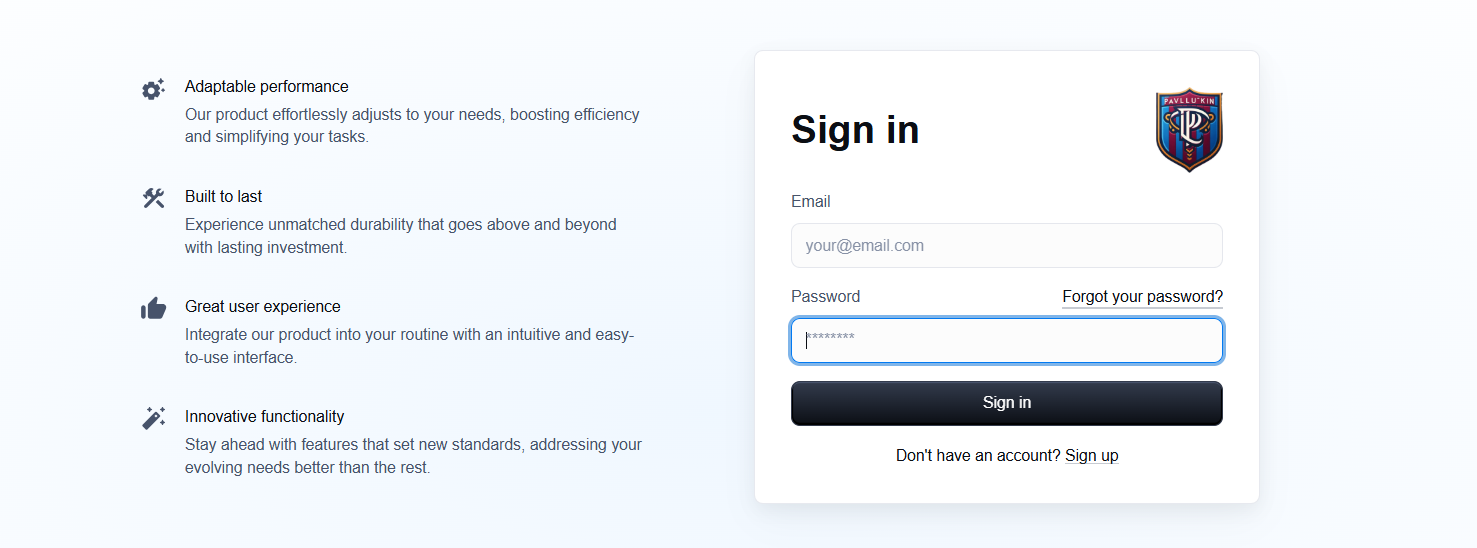


Рисунок 5.3 – Страница авторизации пользователя

После входа в систему пользователь видит главную страницу личного кабинета. Здесь отображается краткая информация о пользователе и список турниров, в которых он участвует или может принять участие. При первом посещении рекомендуется перейти на страницу профиля, представленную на рисунке 5.4, чтобы настроить персональные данные.

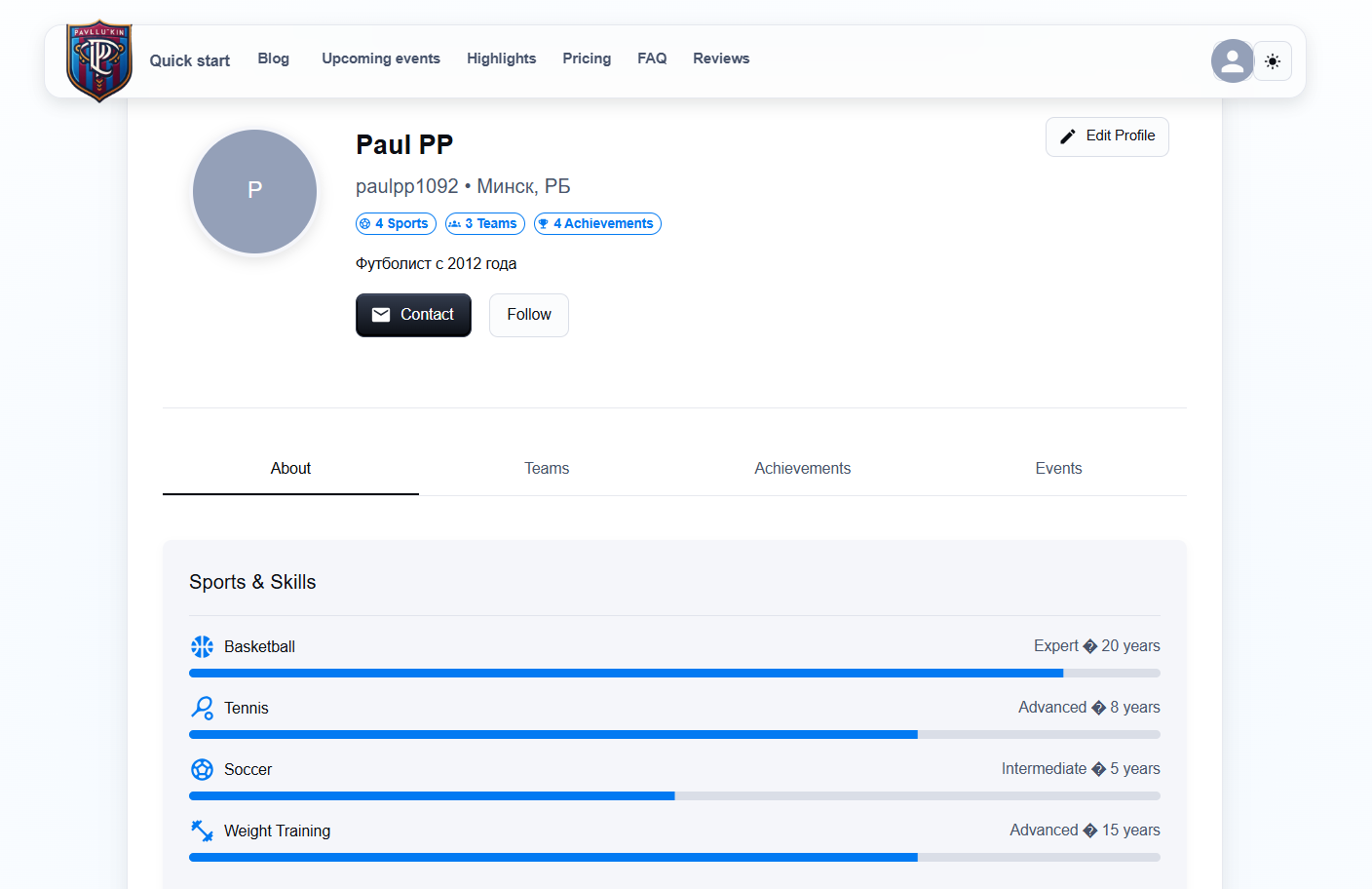


Рисунок 5.4 – Просмотр профиля пользователя

На странице профиля можно изменить имя, загрузить фотографию, указать спортивную категорию, контактную информацию и социальные сети. Все изменения вступают в силу после сохранения настроек. Редактирование профиля представлено на рисунке 5.5.Фотография загружается с локального устройства и сохраняется в Firebase Storage.

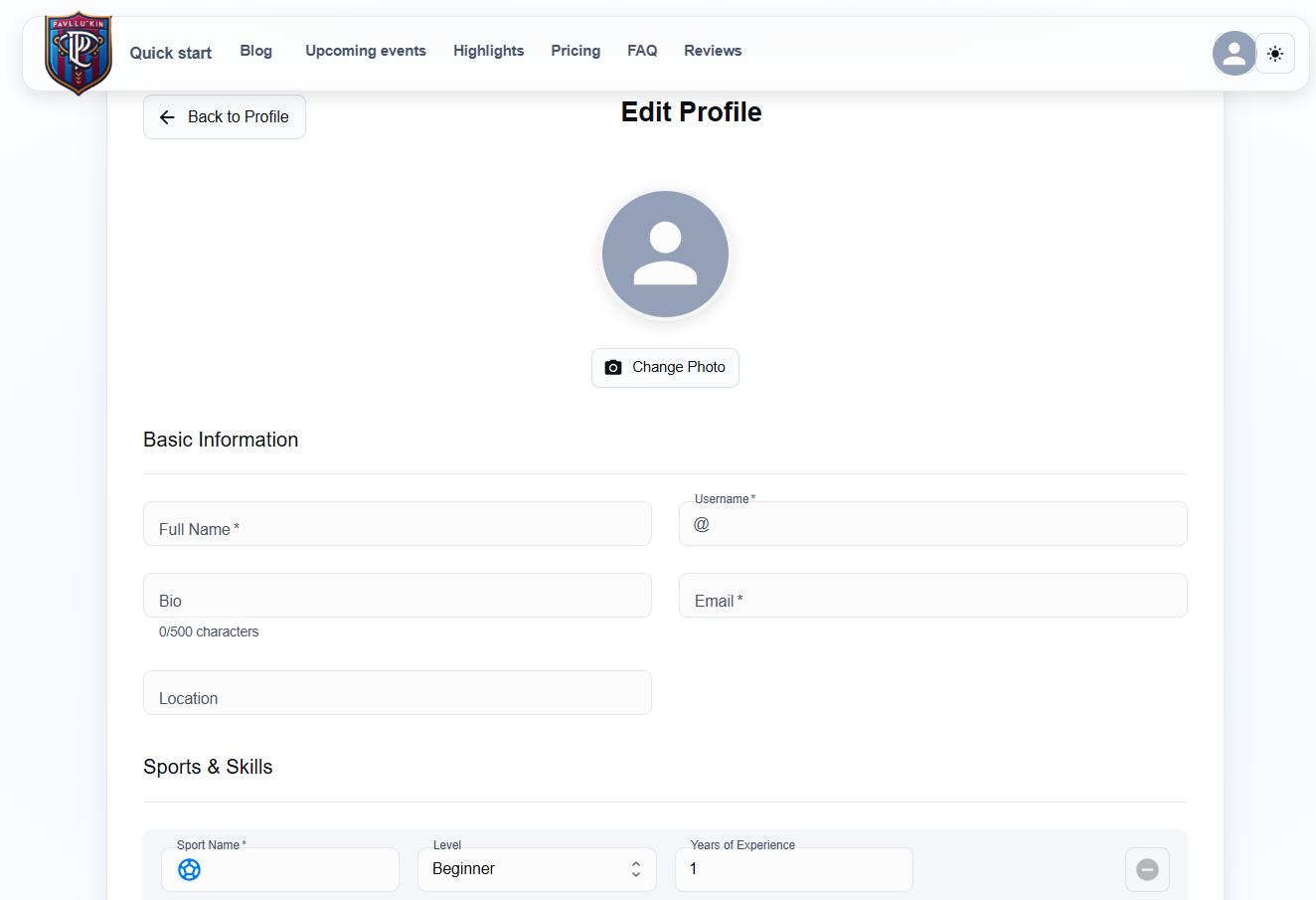


Рисунок 5.5 – Управление профилем пользователя

Особое внимание в приложении уделено ролевой модели. В зависимости от выбранной роли (участник, организатор или администратор), пользователь получает доступ к разному функционалу. Например, участник может просматривать турниры, подавать заявки, отслеживать своё расписание и результаты. Организатор имеет доступ к интерфейсу создания и управления турнирами.

Для создания новой команды организатор или представитель команды переходит на соответствующую вкладку и заполняет форму с названием команды, видом спорта, контактными данными и списком игроков. Также есть возможность загрузить логотип команды и назначить капитана (см. рисунки 5.6 и 5.7). После сохранения команда становится доступной для участия в турнирах.

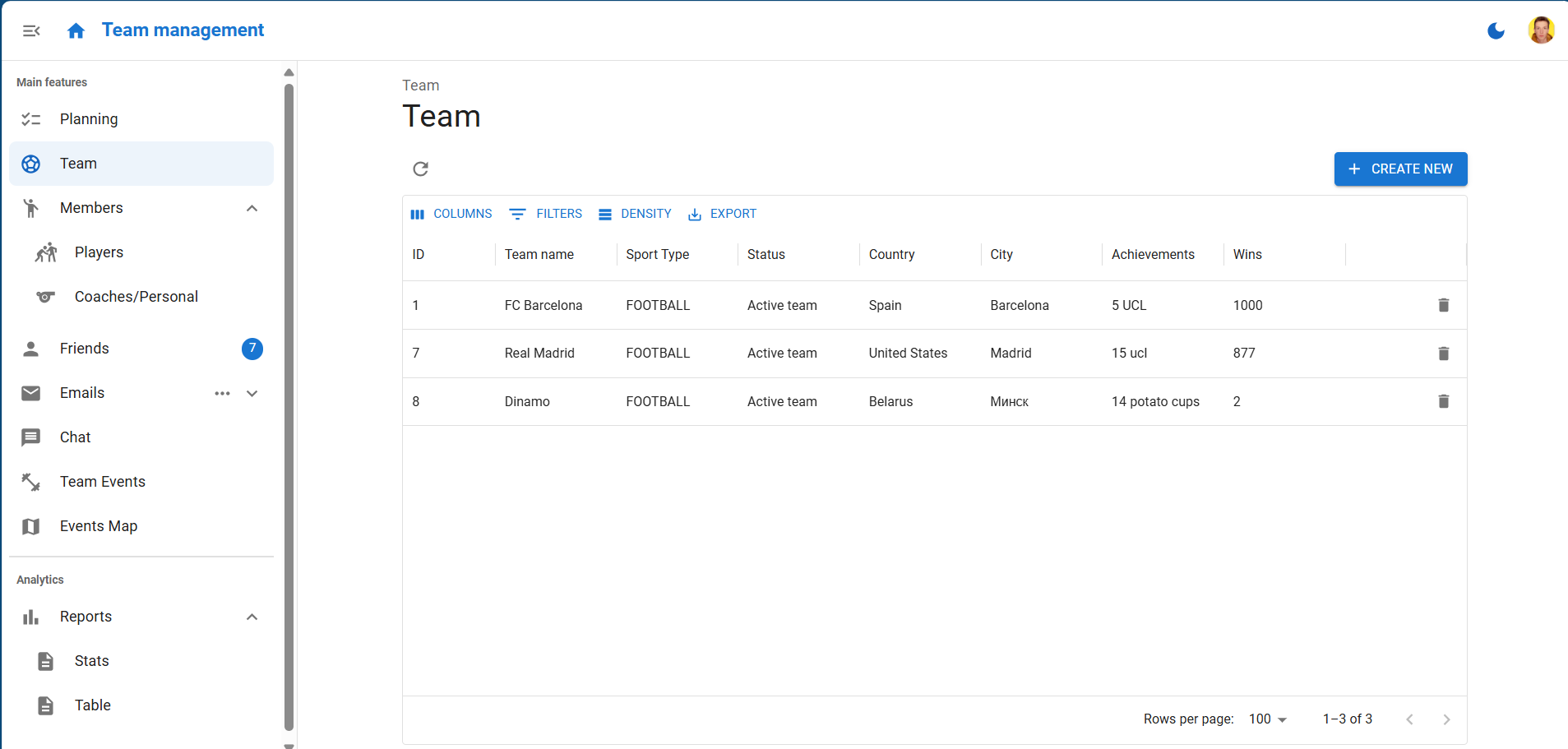


Рисунок 5.6 – Интерфейс управления командами

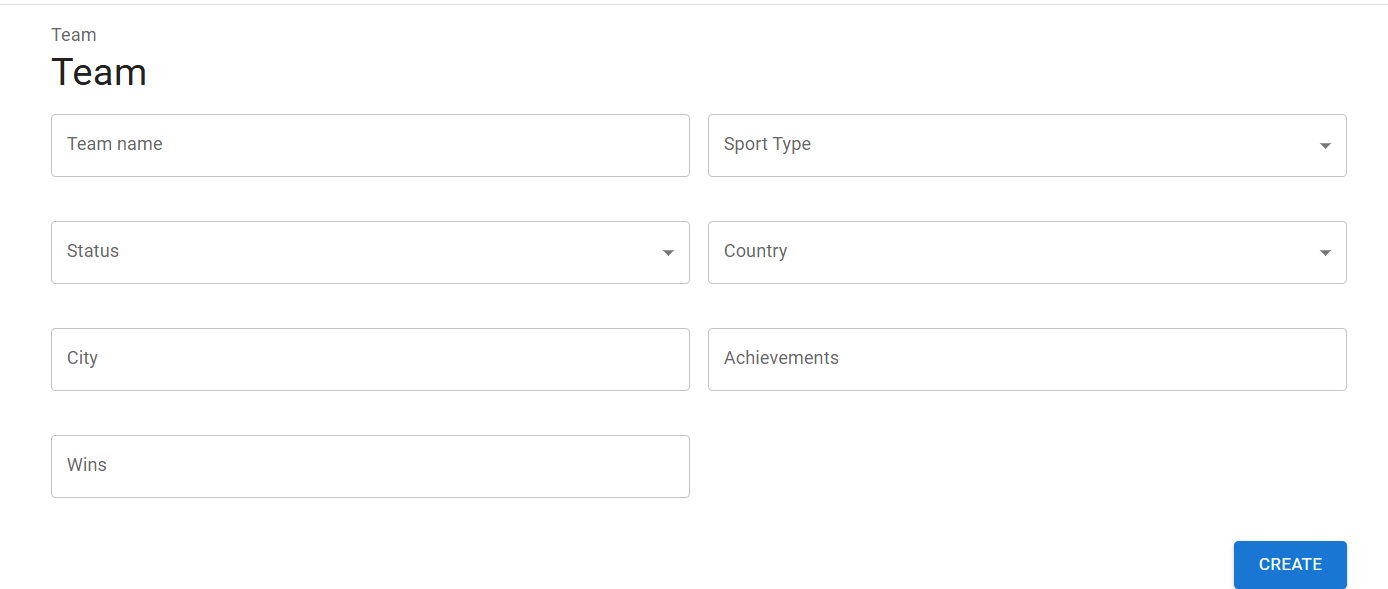


Рисунок 5.7 – Интерфейс создания команды

Создание турнира происходит через отдельную страницу, доступную только организаторам. Здесь необходимо указать название турнира, вид спорта, дату начала и окончания, регламент, правила, условия участия и количество команд. Дополнительно можно прикрепить документы, добавить информацию о судейской коллегии и создать этапы турнира (групповой, плей-офф и т.д.). После подтверждения форма передаётся на модерацию (если включено ручное одобрение) либо сразу становится доступной для подачи заявок (см. рисунок 5.8).

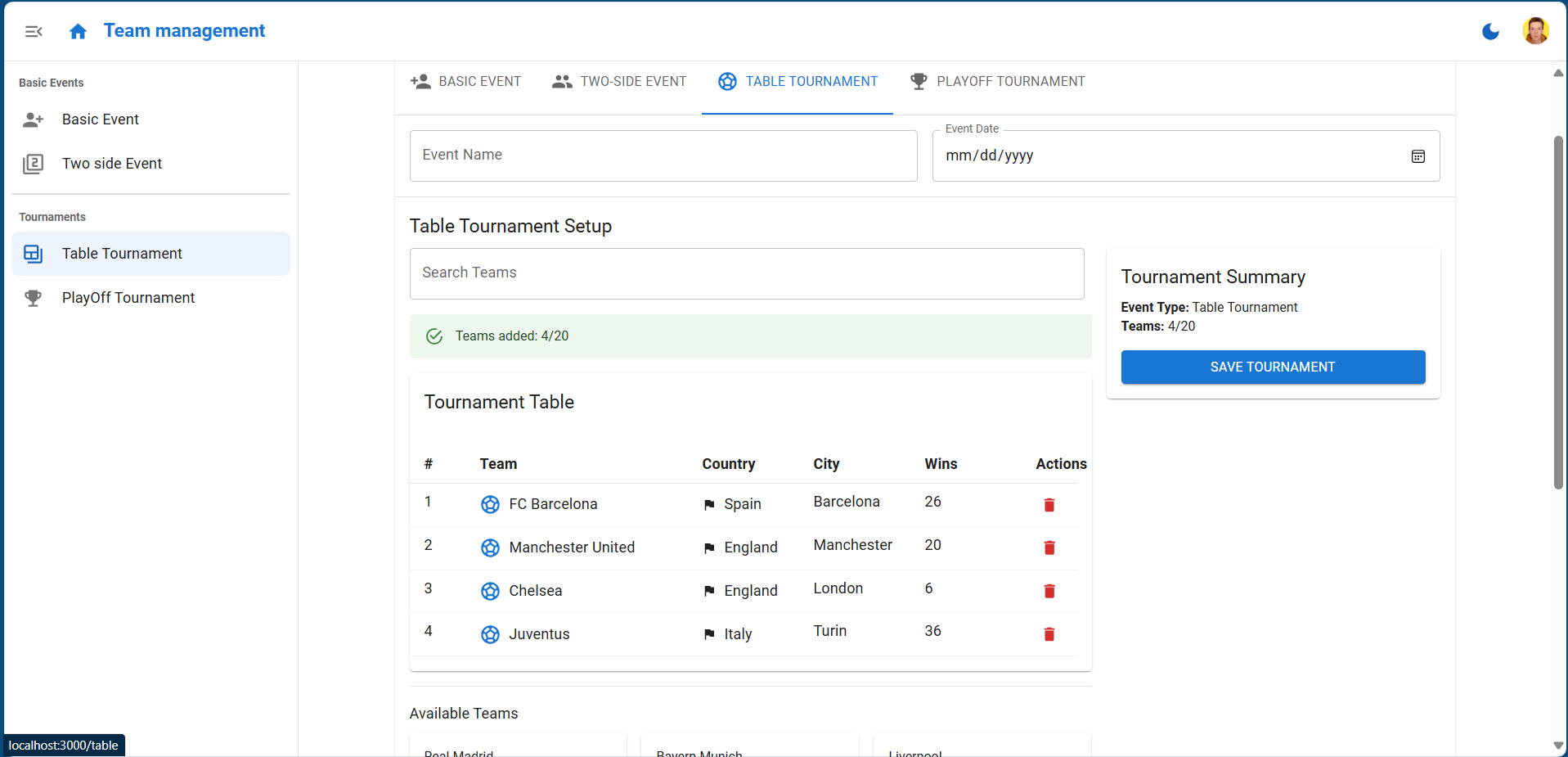


Рисунок 5.8 – Интерфейс создания турнира-таблицы

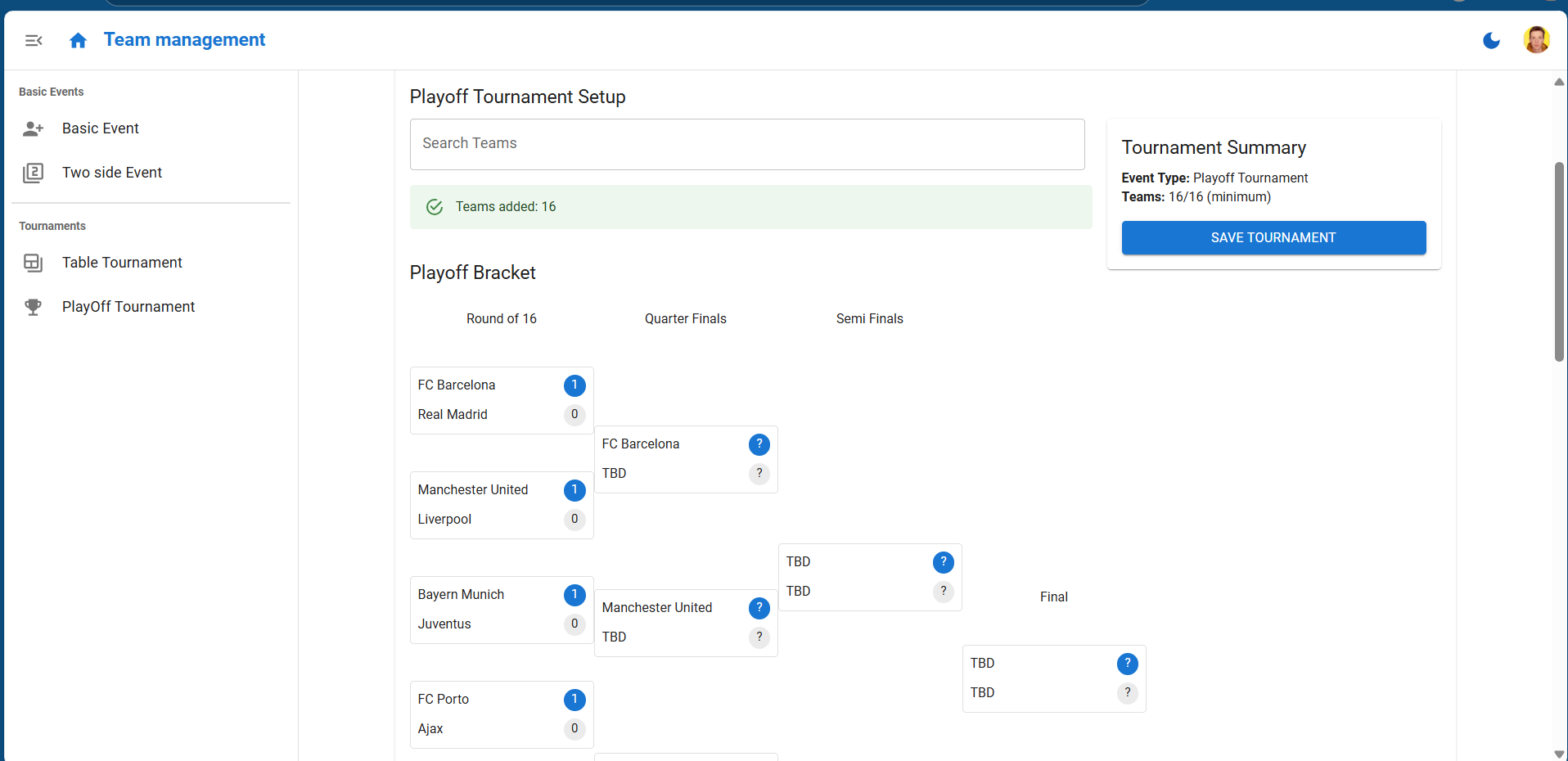


Рисунок 5.9 – Интерфейс создания турнира плей-офф навылет

Турниры можно фильтровать по видам спорта, регионам и датам проведения. Пользователь может открыть страницу турнира, где отображается подробная информация, таблица участников, расписание и результаты. Страницы турниров обновляются в реальном времени, что особенно полезно при проведении соревнований в формате live.

Серверная часть реализована с использованием Java Spring Boot, что позволяет выстраивать логически обособленные REST-сервисы. Сервер обрабатывает входящие запросы от клиента, выполняет логику приложения, взаимодействует с базой данных PostgreSQL и возвращает результаты в виде JSON-ответов. Все параметры подключения к базе данных настраиваются в файле application.properties, включая URL, имя пользователя и пароль.

Для проверки надёжности и корректности работы серверной части применяется тестирование с использованием JUnit 5. Это современный фреймворк, поддерживающий модульное, параметризованное и интеграционное тестирование. Каждый сервис, репозиторий и контроллер снабжён собственными тестами, которые выполняются в изолированной среде. Интеграционные тесты проверяют работу всех компонентов в связке, включая взаимодействие с базой данных. Для имитации зависимостей используется Mockito, позволяющий создавать поддельные объекты и контролировать поведение классов. Также применяется Spring Boot Test, расширяющий возможности тестирования, включая работу с REST-контроллерами и WebApplicationContext.

Процесс тестирования автоматизирован: все тесты запускаются через Maven с помощью команды mvn test. Это позволяет регулярно проверять целостность системы, выявлять регрессию и уверенно вносить изменения в код. В отчёте о выполнении тестов фиксируются как успешные, так и провальные сценарии, включая ошибки аутентификации, некорректные данные и обработку исключений.

Развёртывание серверной части может осуществляться как на локальном сервере Tomcat, так и в контейнере Docker. При использовании Docker создаётся образ приложения с уже настроенным окружением, который можно быстро развернуть на любом сервере, что упрощает процесс установки и сопровождения.

Таким образом, программное средство сочетает в себе интуитивно понятный интерфейс, гибкую архитектуру, мощные средства тестирования и надёжную серверную инфраструктуру. Это делает его универсальным решением как для проведения локальных соревнований, так и для масштабных турниров с участием множества команд и пользователей.

# **6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ НА РЫНКЕ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ОРГАНИЗАЦИИ СПОРТИВНЫХ МЕРОПРИЯТИЙ И ТУРНИРОВ**

## **6.1****Характеристика программного средства, разрабатываемого для реализации на рынке**

Разрабатываемое веб-приложение представляет собой интуитивно понятную платформу для управления процессами организации спортивных мероприятий и турниров. Приложение будет доступно через веб-браузер и ориентировано на упрощение всех этапов планирования: от создания событий и координации участников до анализа результатов. Платформа предложит гибкие и мощные инструменты, которые подходят как для профессиональных спортивных организаций, так и для частных пользователей, стремящихся организовать свои мероприятия с минимальными усилиями.

Приложение предложит пользователям широкий спектр возможностей для управления спортивными событиями:

1. Веб-приложение позволит пользователям создавать мероприятия с индивидуальными настройками, включая указание даты, времени, места проведения (физического или виртуального), а также специфики турнира (например, формат матчей или правила).
2. Оно обеспечит регистрацию участников, как индивидуальных, так и команд. Пользователи смогут управлять списком игроков, подтверждать участие, а также делить их на группы или пары в зависимости от формата турнира.
3. Для удобства планирования приложение будет интегрировано с сервисами календарей, такими как Google Calendar и Apple Calendar. Автоматическое добавление событий и напоминаний исключит возможность забыть о важной встрече.
4. Инструменты анализа результатов и построения рейтингов позволят пользователям отслеживать прогресс участников и команд. После завершения событий будут предоставлены подробные отчеты, в которых учтены все результаты.
5. Система уведомлений будет информировать участников о предстоящих событиях, изменениях в расписании и других важных обновлениях.
6. Приложение будет включать премиум-функции, такие как доступ к расширенной аналитике, брендирование интерфейса для профессиональных пользователей и приоритетное размещение мероприятий.

Целью разработки является обеспечение пользователей доступным и функциональным решением, способным оптимизировать процессы координации событий, улучшить взаимодействие между участниками и дать возможность эффективно управлять временем. Благодаря продуманной архитектуре и интеграции с популярными сервисами, данное программное средство идеально подойдет как для организации локальных мероприятий, так и для более сложных турниров и лиг.

Данное приложение предназначено для владельцев ПК, ноутбуков, а также телефонных устройств, которые имеют операционную систему Android или iOS с доступом к сети Интернет. Программное средство будет востребовано различными категориями пользователей. В первую очередь, это спортивные организации и клубы, нуждающиеся в простых и надежных инструментах для управления лигами и турнирами. Также к числу пользователей можно отнести любителей спорта, которые хотят организовывать тренировки, матчи и дружеские соревнования. Частные организаторы смогут использовать платформу для проведения локальных мероприятий, таких как мастер-классы и любительские соревнования.

Веб-приложение будет развернуто на облачном хостинге с использованием домена в зоне «.by». Такой подход гарантирует стабильную работу платформы, ее высокую производительность и масштабируемость. Для хранения данных о мероприятиях, участниках и статистике будет использоваться надежная и безопасная архитектура баз данных.

Приложение будет предоставляться бесплатно с возможностью покупки премиум-функций. Основным способом монетизации станет встроенная реклама, которая будет адаптирована под спортивные и игровые тематики. Премиум-пользователи смогут отключить рекламу и получить доступ к дополнительным функциям для более глубокого анализа и персонализации интерфейса.

## **6.2 Расчет инвестиций в разработку программного средства для                  его реализации на рынке**

Инвестиции включают расходы, связанные с созданием веб-платформы. Эти затраты формируются из состава и численности проектной команды, уровня ежемесячной заработной платы каждого из участников, а также из трудоемкости разработки.

Для выполнения проекта «Программное средство для управления процессами организации спортивных мероприятий и турниров» потребуется один месяц работы команды, состоящей из программиста, тестировщика, дизайнера UI/UX и маркетолога.

Основная заработная плата участников команды рассчитывается по определенной формуле:

где – это коэффициент премий; – это категории исполнителей, занятых разработкой программного средства; – часовой оклад исполнителя i-й категории, р; – трудоемкость работ, выполняемых исполнителем i-й категории, ч.

Заработные платы членов команды определены на основании средних значений, взятых из вакансий на сайте rabota.by и hh.ru. Часовая оплата труда рассчитывалась путем деления месячной заработной платы на стандартное количество рабочих часов в месяц, равное 168 часам. Объем работы установлен в 168 часов для инженера-программиста, 168 часов для тестировщика, 60 часов для дизайнера и 72 часа для маркетолога. Премии составляют 50% от базовой заработной платы. Расходы на основную заработную плату приведены в таблице 1.

Таблица 1 – Расчет затрат на основную заработную плату команды разработчиков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория исполнителя | Месячный оклад, р. | Часовой оклад, р. | Трудоемкость работ, ч. | Итого, р. |
| Программист | 5 200,00 | 30,95 | 168 | 5 200,00 |
| Тестировщик | 2 500,00 | 14,88 | 168 | 2 500,00 |
| UI/UX-дизайнер | 4 000,00 | 23,80 | 60 | 1 428,00 |
| Маркетолог | 2 000,00 | 11,90 | 72 | 857,00 |
| *Итого* | | | | 9 985,00 |
| Премия и иные стимулирующие выплаты (по данным предприятия или 50-100%) | | | | 4 992,50 |
| *Всего* затрат на основную заработную плату разработчиков | | | | 14 977,50 |

Расчёт дополнительной заработной платы осуществлён исходя из норматива дополнительной заработной платы, который принимаем за 15%. Затраты на дополнительную заработную плату произведены по следующей формуле:

где – это норматив дополнительной заработной платы.

Подставляем полученные значения в формулу 7.2:

Отчисления на социальные нужды рассчитываются по формуле:

где – это норматив отчислений в ФСЗН и Белгострах (по состоянию на январь 2025 г. норматив отчислений в ФСЗН и Белгострах составляет 34,6%).

Подставим полученные значения в формулу 7.3:

Затраты на прочие расходы рассчитываются с помощью норматива прочих расходов, который принят равным 30%, по следующей формуле:

где – это норматив прочих расходов.

Подставим полученные значения в формулу 7.4:

Общая сумма инвестиций на разработку находится по следующей формуле:

Подставим в формулу 7.5 полученные значения:

Таким образом получили общую сумму инвестиций на разработку и реализацию.

## **6.3 Расчет экономического эффекта от реализации программного                  средства на рынке**

Для расчета экономического эффекта внедрения программного средства для управления процессами организации спортивных мероприятий и турниров целесообразно проанализировать целевую аудиторию, рынок, потенциальные доходы и методы монетизации.

Основными пользователями программного средства станут организаторы спортивных мероприятий и турниров: спортивные клубы, школы, федерации, индивидуальные организаторы и любители спорта. Также инструмент будет интересен спортсменам и участникам мероприятий, которые смогут использовать его для регистрации и получения информации о турнирах.

В Республике Беларусь активно функционируют спортивные клубы и федерации, охватывающие значительное количество спортивных направлений. По приблизительным данным, ежегодно проводится около 10-15 тысяч мероприятий различного масштаба. Также значительное число активных спортсменов и любителей спорта участвуют в локальных и региональных турнирах, а также любительский мероприятиях.

Основными методами монетизации для данного программного средства будут:

1. Показ рекламы. Пользователи платформы будут видеть рекламу, например, спортивных товаров или услуг. Стоимость одного показа рекламы может составить от 0,002 до 0,004 белорусских рубля (в зависимости от бренда товара или услуги).
2. Платный функционал, например, возможность подключения дополнительных опций для организаторов, таких как автоматизированная система формирования турнирных таблиц или расширенная аналитика.
3. Комиссия с регистрации: Небольшая комиссия с каждого зарегистрированного участника мероприятия.

При предполагаемой аудитории в 70,000 пользователей и активной рекламе годовой доход от рекламы и платного функционала может составить порядка 65,000 белорусских рублей. Дополнительный доход может быть получен за счет платного функционала и комиссии, что увеличивает общую прибыль.

Чистую прибыль, получаемую от реализации программного средства, можно определить по формуле, учитывающей общий доход, себестоимость разработки, эксплуатационные расходы и другие затраты:

где – предполагаемый доход от рекламы за год, р.; НДС – сумма налога на добавленную стоимость, р.; – ставка налога на прибыль согласно действующему законодательству, % (по состоянию на февраль 2025 г. – 25%)

Налог на добавленную стоимость определяется по формуле

где – ставка налога на добавленную стоимость в соответствии с действующим законодательством, % (по состоянию на февраль 2025 г. – 20%).

Расчет НДС (расчёт налога на прибыль):

Зная налог на добавленную стоимость, можно провести расчет прироста чистой прибыли, которую может получить разработчик от продажи программного продукта, с помощью следующей формулы:

где – отпускная цена копии (лицензии) программного средства, р.

– количество копий (лицензий) программного средства, шт.

– сумма налога на добавленную стоимость.

– рентабельность продаж копий (лицензий).

– ставка налога на прибыль, %.

Зная размер налога на добавленную стоимость, можно провести расчет прироста численной прибыли, которую получит команда разработки. Важно подсчитывать с учётом того, что заработок от рекламы, а не от продажи лицензий. Соответственно уменьшаемое () р. [заработок от рекламы]. Ставка налога на прибыль по состоянию на 03.03.25 составляет 20 %. Используя имеющиеся данные, прирост чистой прибыли команды разработки составит:

Таким образом прирост чистой прибыли от реализации программного средства на массовом рынке составит рублей в год.

## **6.4 Расчет показателей экономической эффективности разработки и                 реализации программного средства на рынке**

Оценка экономической эффективности разработки и реализации программного средства для управления процессами организации спортивных мероприятий и турниров базируется на сравнении первоначальных инвестиций в разработку с предполагаемым годовым приростом чистой прибыли. Экономическая эффективность считается достигнутой, если годовой прирост чистой прибыли превышает затраты на создание платформы.

Для анализа экономической целесообразности используется показатель рентабельности инвестиций, рассчитываемый по формуле:

где – прирост чистой прибыли, полученной от реализации программного средства на рынке, р.

– затраты на разработку программного средства, р.

Экономическая эффективность разработки программного средства для управления процессами организации спортивных мероприятий и турниров определяется не только показателями рентабельности, но и стратегическим вкладом в улучшение рынка спортивных событий. При оценке эффективности учитываются первоначальные инвестиции, прогнозируемые доходы, а также возможность создания устойчивой бизнес-модели, адаптированной к условиям современного рынка.

Разработка платформы показывает превосходные результаты с точки зрения рентабельности, достигая 56,57%, что значительно превышает уровень ставки рефинансирования в 11,5%. Однако достижение поставленных целей требует привлечения обширной пользовательской базы, насчитывающей около 70 000 активных пользователей, ежедневно посещающих платформу и взаимодействующих с её функционалом. Хотя этот сценарий предусматривает значительные возможности для роста, существует риск недостаточного охвата целевой аудитории или низкого уровня вовлеченности пользователей, например, из-за того, что не все аналоги имеют большой успех в Республике Беларусь.

Для минимизации этих рисков предусмотрены дополнительные механизмы монетизации. В частности, внедрение платного функционала для организаторов спортивных мероприятий, включая автоматизацию управления заявками, интеграцию с системами онлайн-оплаты, и предоставление аналитики для оценки результатов. Эти функции не только увеличат доходность проекта, но и сделают платформу привлекательнее для организаторов и участников турниров, повышая её конкурентоспособность на рынке.

Долгосрочный потенциал платформы заключается в её способности адаптироваться к меняющимся потребностям пользователей и рынков. Например, можно рассмотреть развитие инструментов для автоматического составления расписаний соревнований, управление билетами на мероприятия, а также интеграцию с платформами для видеотрансляций турниров, что откроет новые возможности для монетизации и увеличения доходов.

Однако реализация подобных улучшений требует дополнительных инвестиций, включая расширение команды разработчиков, увеличение бюджета на маркетинг, а также приобретение и внедрение новых технологий. Эти расходы необходимо учитывать при пересмотре экономической модели проекта.

Успешная реализация платформы и её дальнейшее развитие обеспечат устойчивый рост проекта и расширение его рыночной доли. Это создаст значительные преимущества для всей экосистемы спортивных событий.

ЗАКЛЮЧЕНИЕ

Анализ текущих тенденций в области управления спортивными мероприятиями и турнирами показал возрастающую потребность в цифровизации процессов организации и проведения соревнований. Рост популярности онлайн-платформ, а также активное использование мобильных и веб-технологий в спорте подтверждают актуальность создания специализированных информационных систем. Развитие облачных технологий, REST API и современных баз данных, таких как PostgreSQL, открывает широкие возможности для автоматизации работы организаторов и участников соревнований.

В рамках дипломного проекта было разработано программное средство, предназначенное для управления процессами организации спортивных мероприятий и турниров. Разработанная система предоставляет организаторам удобный инструмент для планирования событий, управления заявками, составления расписаний и сбора статистики. Участники получают возможность регистрироваться на соревнования, отслеживать расписание, взаимодействовать с организаторами и получать актуальную информацию в режиме реального времени. В основе системы лежит архитектура клиент-сервер с использованием Spring Boot на стороне бэкенда и React для фронтенда, что обеспечивает высокую производительность и удобство работы с приложением.

В ходе работы над проектом были изучены и применены современные методики проектирования программных систем, реализована надежная архитектура приложения, обеспечивающая гибкость и возможность дальнейшего масштабирования. Разработанный продукт интегрирован с базой данных PostgreSQL, что позволяет эффективно управлять большим объемом информации о турнирах, участниках и их результатах.

Перспективы развития системы включают расширение функционала, внедрение дополнительных аналитических инструментов, автоматизированной генерации турнирных сеток, интеграции с мобильными приложениями и социальными платформами. Благодаря модульной архитектуре и использованию передовых технологий, разработанное программное средство может быть адаптировано для различных видов спортивных мероприятий и использовано в широком спектре спортивных организаций.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Sheriff, Paul D. Designing for Web or Desktop? [Электронный ресурс]. – Электронные данные. – Режим доступа: https://msdn.microsoft.com/en-us/library/ms973831.aspx. – Дата доступа: 18.02.2025.
2. Паттерны проектирования для веб-разработки [Электронный ресурс]. – Электронные данные. – Режим доступа: https://refactoring.guru/ru/design-patterns. – Дата доступа: 18.02.2025.
3. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гама, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб.: Питер, 2020. – 368 с.
4. Архитектура программного обеспечения в действии / Пер. с англ. – М.: ДМК Пресс, 2022. – 432 с.
5. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч. – М.: Бином, 2019. – 624 с.
6. Spring Framework Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.spring.io/spring-framework/reference/index.html. – Дата доступа: 20.02.2025.
7. Документация по PostgreSQL [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.postgresql.org/docs/. – Дата доступа: 02.04.2025.
8. REST API Design Guide [Электронный ресурс]. – Электронные данные. – Режим доступа: https://restfulapi.net/. – Дата доступа: 02.04.2025.
9. React Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: https://react.dev/. – Дата доступа: 02.04.2025.
10. Разработка спортивных онлайн-платформ [Электронный ресурс]. – Электронные данные. – Режим доступа: https://medium.com/@sports-tech. – Дата доступа: 02.04.2025.
11. 11 Great Sports Team Management Software Platforms & Apps [2024] [Электронный ресурс]. – Электронные данные. – Режим доступа: https://eventpipe.com/blog/sports-team-management-software. – Дата доступа: 02.04.2025.
12. Национальный статистический комитет Республики Беларусь [Электронный ресурс]. – Режим доступа: https://www.belstat.gov.by. – Дата доступа: 14.02.2025.
13. Парк высоких технологий (ПВТ). Официальный сайт. [Электронный ресурс]. – Режим доступа: https://park.by – Дата доступа: 04.03.2025.
14. HH.ru [Электронный ресурс]. – Режим доступа:  https://hh.ru/?hhtmFrom=main . – Дата доступа: 10.03.2025.
15. Rabota.by [Электронный ресурс]. – Режим доступа:  https://rabota.by/?hhtmFrom=vacancy\_search\_list. – Дата доступа: 12.03.2025.
16. Налог на добавленную стоимость [Электронный ресурс]. – Режим доступа: https://myfin.by/wiki/term/nalog-na-dobavlennuyu-stoimost. – Дата доступа: 14.03.2025.
17. Что такое ROI? [Электронный ресурс]. – Режим доступа: https://soldimarketing.ru/analitika/roi/. – Дата доступа: 15.03.2025.