МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ВЕБ - ПРИЛОЖЕНИЕ
«САЙТ ФУТБОЛЬНОЙ КОМАНДЫ ПГУ»

Студент 30.05.2025 Хабибулин Айрат Маратович
(подпись, дата)

Руководитель 30.05.2025 Юрова Ольга Викторовна
(подпись, дата)

Нормоконтролёр 30.05.2025 Бычков Андрей Станиславович
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 28.05.25г. № 16)

Заведующий кафедрой _____ М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 16 июня 2025 г. № 41)

Секретарь ГЭК _____ Е. И. Гурин
(подпись) (фамилия, инициалы)

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 06 » мая 2025г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Хабибулин Айрат Маратович гр.21ВВП1 факультета вычислительной техники

2.Тема работы: веб - приложение «Сайт футбольной команды ПГУ»

Тема утверждена приказом ПГУ № 601/о от "06" мая 2025 г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Юрова О.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать веб - приложение «Сайт футбольной команды ПГУ». Приложение должно позволять пользователям выполнять просмотр новостей команды, расписания матчей, результатов, турнирной таблицы. Предоставлять информацию о статистике, составе команды, а также о персонале клуба и каждом игроке по отдельности. Приложение должно позволять осуществлять регистрацию и авторизацию пользователя, редактировать информацию о пользователе. Приложение должно быть спроектировано с использованием клиент-серверной архитектуры. Клиентская часть программы должна обладать понятным графическим интерфейсом. Сервер должен включать в себя сохранение данных в базу данных, передачу, редактирование и удаление их из нее. Клиентская программа должна позволять пользователю добавлять комментарии под новостями команды, просматривать профиль игроков, персонала, получать данные о календаре матчей, турнирной таблицы, статистики игроков и команд.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки веб - приложения. Также необходимо изучить технологий для разработки приложения, такие как SpringFramework, PostgreSQL, SpringSecurity и т.д.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	40	06.05.2025-10.05.2025	
2.	Анализ требований	16	11.05.2025-12.05.2025	
3.	Выбор технологий, применяемых при проектировании и разработке	24	13.05.2025-15.05.2025	
4.	Описание алгоритмов	24	16.05.2025-18.05.2025	
5.	Разработка веб-приложения «Сайт футбольной команды ПГУ»	80	19.05.2025-28.05.2025	
6.	Оформление пояснительной записки	16	29.05.2023-30.05.2025	

Дата выдачи задания "06" мая 2025 г.

Руководитель бакалаврской работы 06.05.25 Юрова О.В.
(подпись, дата)

Задание к исполнению принял студент Хабибулин А.М.
(подпись, дата)

Работу к защите допустить 16.06.2025
Декан факультета Фионова Л.Р.

Реферат

Бакалаврская работа содержит 129 листов, 76 рисунков, 10 таблиц, 6 источников и 3 приложения.

ВЕБ-ПРИЛОЖЕНИЕ «САЙТ ФУТБОЛЬНОЙ КОМАНДЫ ПГУ», SPRING FRAMEWORK, INTELLIJ IDEA, SPRING SECUTITY, POSTGRESQL, PGADMIN4, HTML, CSS, JAVA

Цель работы – разработать веб-приложение, позволяющее получать всю необходимую информацию о команде: расписание игр, новости, состав, турнирная таблица, результаты матчей, статистика игроков.

Разработанное приложение должно позволять пользователям выполнять просмотр новостей команды, расписания матчей, результатов, турнирной таблицы. Предоставлять информацию о статистике, составе команды, а также о персонале клуба и каждом игроке по отдельности. Приложение должно позволять осуществлять регистрацию и авторизацию пользователя, редактировать информацию о пользователе.

Результатом разработки является проект, удовлетворяющий поставленной задаче. А именно, разработано, реализовано и протестировано приложение сайта футбольной команды ПГУ. Разработка и реализация производились с использованием современных информационных технологий, к которым относятся: язык программирования Java, язык структурированных запросов SQL, Spring Framework, Spring Security, язык гипертекстовой разметки HTML, язык каскадных таблиц стилей CSS, среда разработки IntelliJIDEA. Для хранения данных выбрана СУБД PostgreSQL, а для её администрирования применялся инструмент PGAdmin4.

Содержание

Перечень сокращений и обозначений 5
Введение 6
1 Анализ предметной области 7
1.1 Анализ актуальности темы 7
1.2 Анализ аналогов веб-сайтов футбольных команд 8
1.3 Постановка задачи 14
2 Выбор технологий и средств разработки приложения 17
2.1 JAVA 17
2.2 Spring Framework 17
2.3 Spring JPA 18
2.4 Spring Security 19
2.5 IntelliJ IDEA 20
2.6 Apache Tomcat 20
2.7 Bootstrap 21
2.8 PostgreSQL 21
2.9 pgAdmin4 22
3 Архитектура веб-приложения «Сайт футбольной команды» 23
4 Реализация приложения 26
4.1 Описание работы приложения 26
4.2 Структура БД 27
4.3 Функционал приложения 36
4.4 Основные компоненты 37
4.4.1 Controller 38
4.4.2 Service 41
4.4.3 Model 43
4.4.4 Основные алгоритмы взаимодействия клиента и сервера 47
5 Тестирование приложения 53

5.1 Тестирование серверной части 53
5.1.1 Тестирование post-методов 53
5.1.2 Тестирование put-методов 66
5.1.3 Тестирование delete-методов 71
6 Тестирование клиентской части 73
6.1 Регистрация пользователя 73
6.2 Авторизация пользователя 77
6.3 Тестирование функционала просмотра информации 80
6.4 Тестирование функционала выход из аккаунта 85
7 Безопасность жизнедеятельности 88
7.1 Требования к помещению и рабочему месту 88
7.2 Требования к режиму труда и отдыха 89
Заключение 91
Список использованных источников 92
Приложение А. UML-диаграммы 93
Приложение Б. Листинг прогаммы 98
Приложение В. Презентация 99

Перечень сокращений и обозначений

БД – база данных

ПО – программное обеспечение

СУБД – система управления базой данных

HTTP – (от англ. Hypertext Transfer Protocol) – протокол прикладного уровня передачи данных

CSS – (от англ. Cascading Style Sheets) – формальный язык описания внешнего вида документа.

HTML (отангл. Hyper Text Markup Language) – язык гипертекстовой разметки

SQL – (англ. Structured Query Language) - язык структурированных запросов

JPA – (от англ. Java Persistence API) – спецификация, описывающая систему управления сохранением java-объектов в таблицы реляционных баз данных в удобном виде

MVC (отангл. Model-View-Controller) — шаблон (паттерн) программирования, разделяющий архитектуру приложения на три модуля

XML – (от англ. eXtensible Markup Language) – расширяемый язык разметки

Введение

В современном цифровом мире веб - сайты спортивных команд играют важную роль в формировании имиджа клуба и являются одним из каналов коммуникации с болельщиками. Веб - сайт - многофункциональный инструмент, призванный информировать болельщиков о последних новостях команды, обеспечивать доступ к эксклюзивному контенту, предоставлять информацию об игроках, предстоящих матча, результатах. С его помощью болельщики имеют возможность приобретения билетов и атрибутики.

В условиях растущей конкуренции в спортивной индустрии наличие современного и эффективного веб - сайта становится одним из важных факторов успеха для спортивной команды.

Таким образом, успешная реализация этих задач требует стратегического подхода и применения современных веб - технологий. В данной работе мы рассмотрим ключевые аспекты создания эффективного веб - сайта для спортивной команды, учитывая особенности спортивного мира и потребности современных болельщиков.

1 Анализ предметной области

Разработка веб - сайта футбольной команды, как и любого программного продукта, начинается с тщательного анализа предметной области. Ключевым является определение целевой аудитории. На основе этого анализа необходимо выбрать актуальные и перспективные технологии, которые обеспечат удобство использования, масштабируемость и возможность дальнейшего развития ресурса. Важнейшим аспектом является выбор подходящей клиент-серверной архитектуры, определяющей способ взаимодействия между клиентами и сервером, хранящим данные и обеспечивающим функционирование сайта. В данном разделе мы рассмотрим теоретические аспекты проектирования и реализации современного веб-сайта футбольной команды, включая вопросы информационной архитектуры, пользовательского интерфейса, серверной инфраструктуры, и, конечно же, принципы построения эффективной клиент-серверной архитектуры.

1.1 Анализ актуальности темы

В современном университетском сообществе, где цифровые технологии пронизывают все аспекты жизни, наличие активного и информативного веб-сайта для институтской футбольной команды становится не просто желательным, а необходимым условием для поддержания и развития команды, привлечения новых болельщиков, спонсоров и повышения престижа университета.

Актуальность выбранной темы обусловлена рядом факторов.

Во-первых, веб-сайт является платформой для информирования студентов о расписании игр, результатах, новостях команды, а также для организации фан-клубов и мероприятий поддержки. Он позволяет создать чувство общности и причастности к команде, что особенно важно для студентов и тех, кто только начинает знакомиться с университетской жизнью.

Во-вторых, успешный веб-сайт может привлечь внимание к команде со стороны потенциальных игроков, спонсоров и выпускников, что способствует

укреплению позиций университета в спортивной сфере и повышению его имиджа. Он также является важным инструментом для привлечения новых студентов и создания позитивного образа университета в целом.

В-третьих, участие студентов в разработке и поддержке веб-сайта команды предоставляет им ценный опыт работы с современными веб-технологиями, развивает их навыки в области веб-дизайна, контент-менеджмента, SMM и интернет-маркетинга, что является важным преимуществом при трудоустройстве после окончания университета.

Таким образом, создание и активное поддержание современного веб-сайта для институтской футбольной команды представляется актуальной и значимой задачей, способствующей достижению целого ряда взаимосвязанных целей, включая расширение аудитории болельщиков, привлечение внебюджетного финансирования и укрепление репутации учебного заведения. Однако, для обеспечения высокой производительности, надежности данного ресурса, а также для гарантии его безопасности и удобства использования, необходимо уделить пристальное внимание проектированию и реализации эффективной клиент-серверной архитектуры. Поэтому в следующем разделе мы подробно рассмотрим основные принципы и подходы к построению клиент-серверных приложений, чтобы определить оптимальную структуру для веб-сайта институтской футбольной команды.

1.2 Анализ аналогов веб-сайтов футбольных команд

С развитием спорта каждый год становится все больше команд, требующих к себе пристального внимания. Для того чтобы привлечь внимание со стороны, создаются новые веб – сервисы и онлайн инструменты. Одним из таких инстументов является веб – сайты футбольных клубов, которые позволяют пользователям узнавать последние новости из стана команды, изучать статистику, покупать билеты и атрибутику, а также участвовать в жизни клуба онлайн.

Для успешной реализации такого приложения, необходимо провести тщательный анализ аналогов. Мы сможем выявить сильные и слабые стороны конкурентов и на их фоне определить необходимый нам функционал, а также требования предъявляемые к разрабатываемому сайту.

В этом разделе мы рассмотрим несколько популярных веб – сайтов футбольных клубов:

– веб-сайт футбольного клуба «Урал» г.Екатеринбург
– веб-сайт футбольного клуба «Зенит» г.Санкт – Петербург
– веб-сайт футбольного клуба «Зенит» г.Пенза

1.2.1 «Урал» г.Екатеринбург

Интерфейс онлайн сервиса «*Урал*» представлен на рисунке 1 и 2.

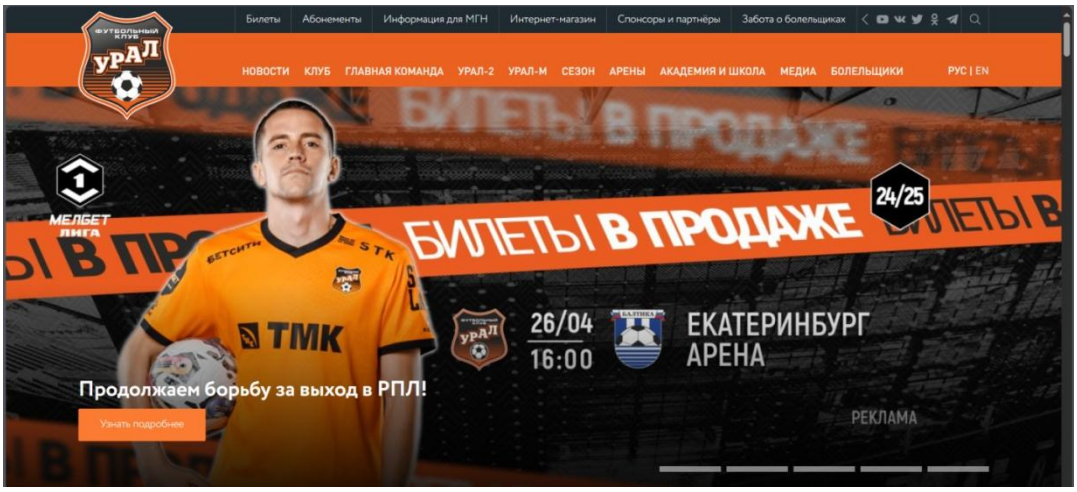


Рисунок 1–Главная страница сервиса

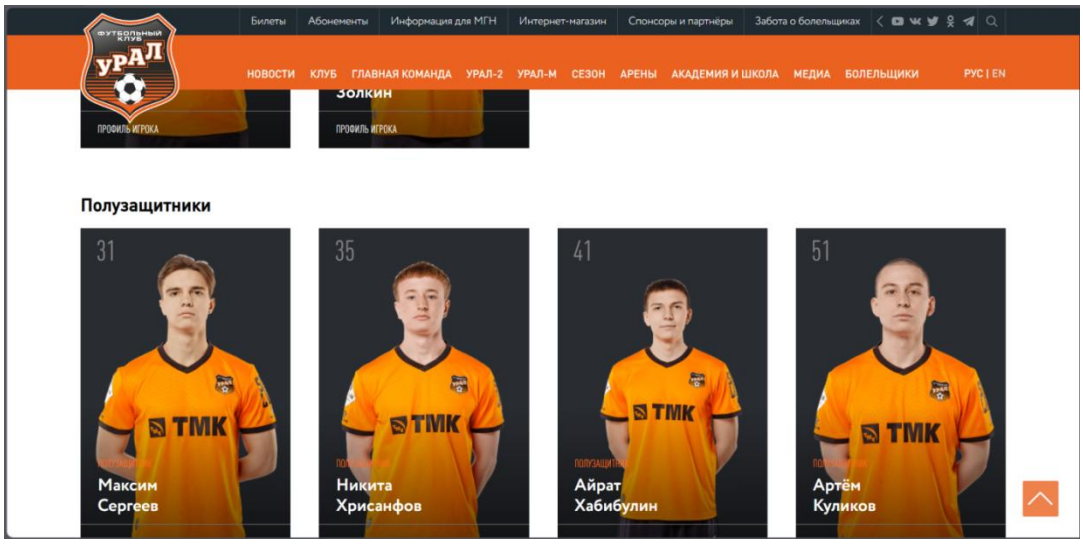


Рисунок 2–Выбор клиента, просмотр состава команды

Сервис веб-сайт «*Урал*» предоставляет своим пользователям ряд услуг для получения подробной информации о команде. Ниже перечислены основные функции и возможности, которые доступны на сайте:

– последние новости о команде, игроках, тренерском штабе, трансферах;

– календарь игр с указанием дат, времени, соперников, места проведения;

– информация о прошедших играх с подробной статистикой, обзорами и видео, турнирная таблица;

– возможность оставлять комментарии к новостям и другим материалам сайта для зарегистрированных пользователей;

– поиск различной информации по сайту;

– фотографии с матчей, тренировок, мероприятий клуба;

– информация о продаже билетов на матчи, возможность онлайн-покупки;

– онлайн-магазин клубной атрибутики, возможность покупки для зарегистрированных пользователей;

– ссылки на страницы клуба в социальных сетях [1].

1.2.2 «Зенит» г.Санкт-Петербург

Интерфейс онлайн сервиса «Зенит» г. Санкт-Петербург представлен на рисунке 3 и 4.

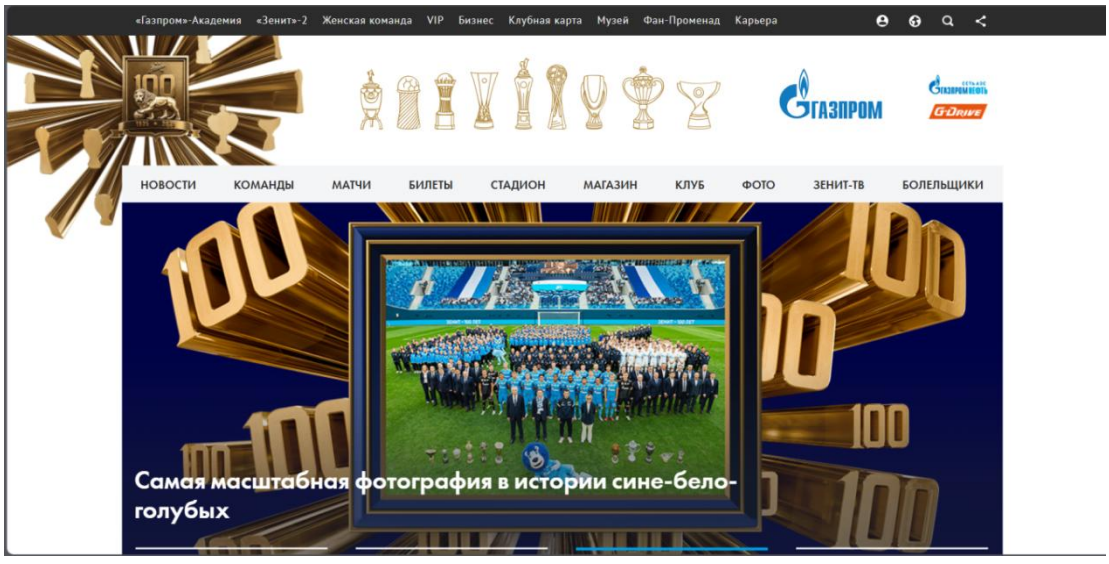


Рисунок 3–Главная страница сервиса

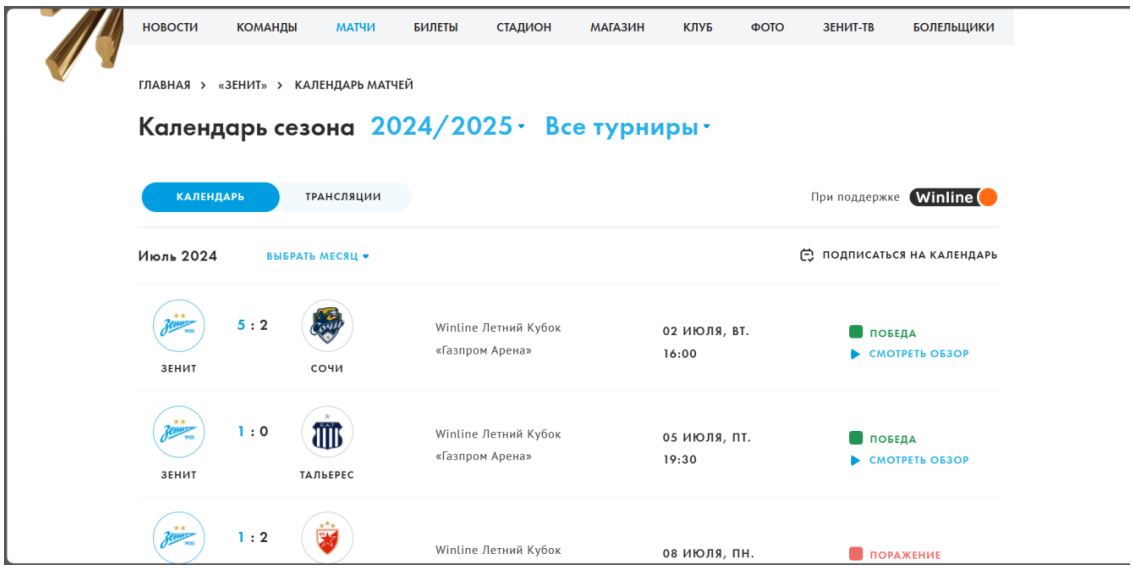


Рисунок 4–Выбор клиентом календаря матчей

Сервис веб-сайт «Зенит» предоставляет своим пользователям ряд услуг для получения подробной информации о команде. Ниже перечислены основные функции и возможности, которые доступны на сайте:

– последние новости о команде, игроках, тренерском штабе, трансферах;

– календарь игр с указанием дат, времени, соперников, места проведения;

– информация о прошедших играх с подробной статистикой, обзорами и видео, турнирная таблица;

– возможность просмотра прямых видеотрансляций матчей для пользователей с платной подпиской;

– раздел для зарегистрированных пользователей с возможностью управления билетами, подписками, участием в конкурсах и акциях, получением персональных предложений;

– наличие версий сайта на других языках для иностранных пользователей;

– поиск информации по сайту;

фотографии с матчей, тренировок, мероприятий клуба;

– онлайн-магазин клубной атрибутики, возможность покупки для зарегистрированных пользователей;

– ссылки на страницы клуба в социальных сетях [2].

1.2.3 «Зенит» г.Пенза

Интерфейс онлайн сервиса «Зенит» г.Пенза представлен на рисунке 5 и 6.

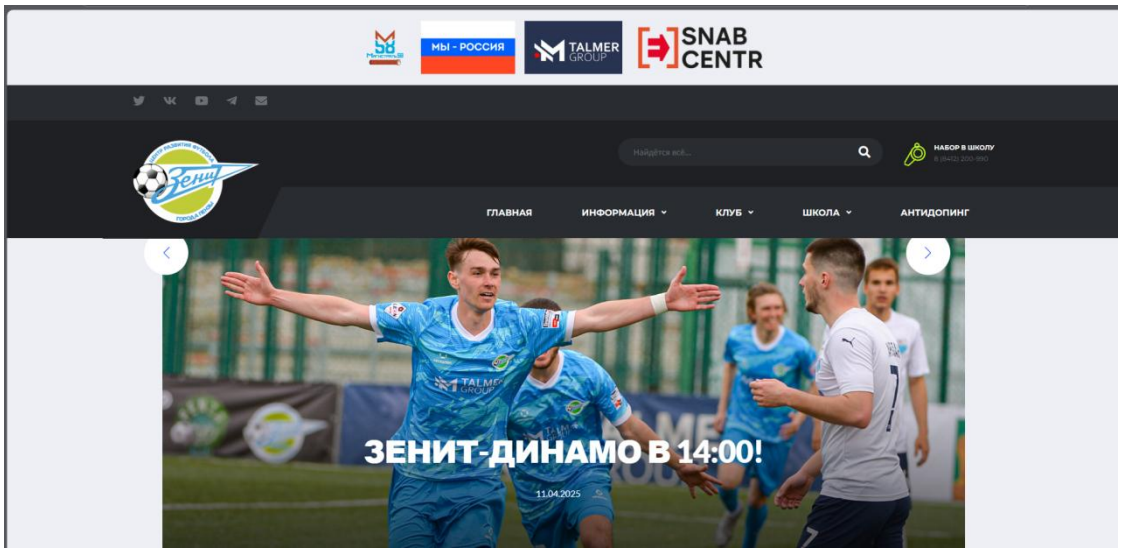


Рисунок 5–Главная страница сервиса

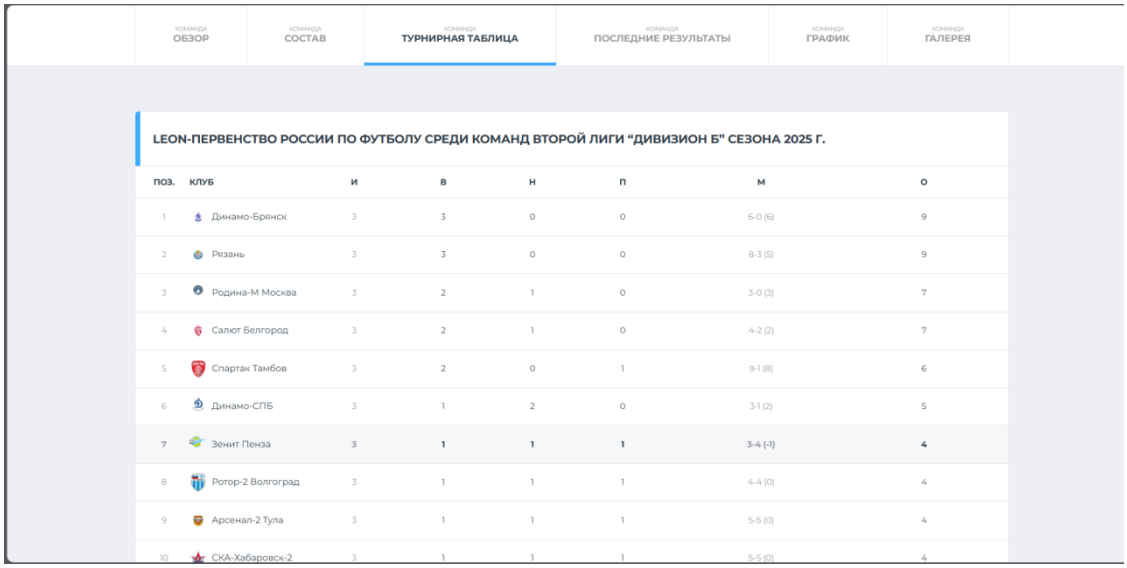


Рисунок 6–Выбор клиентом турнирной таблицы

Сервис веб-сайт «Зенит» предоставляет своим пользователям ряд услуг для получения подробной информации о команде. Самый главный фактор. Пензенский «Зенит» - команда гораздо более скромного уровня, поэтому и сайт будет гораздо проще. Ниже перечислены основные функции и возможности, которые доступны на сайте:

– последние новости о команде, игроках, тренерском штабе, трансферах;

– календарь игр с указанием дат, времени, соперников, места проведения;

– поиск информации по сайту;

– фотографии с матчей, тренировок, мероприятий клуба;

– ссылки на страницы клуба в социальных сетях [3].

1.2.4 Выводы

Исходя из результатов анализа имеющихся аналогов можно составить таблицу 1, показывающую минусы и плюсы веб-сервисов.

Благодаря сравнению и ознакомлению с существующими сайтами футбольных команд, было решено разработать веб-приложение сайта институтской футбольной команды. Оно будет предоставлять пользователям удобный интерфейс и весь функционал, необходимый для комфортного использования.

Таблица 1 –Достоинства и недостатки сайтов футбольных клубов

Функционал	Зенит Санкт-Петербург	Урал Екатеринбург	Зенит Пенза
Поиск по сайту	+	+	+
Новости, расписание, результаты	+	+	+
Фото, видео	+	+	+
Состав и персонал клуба	+	+	+
Турнирная таблица	+	+	+
Личный кабинет	+	+	-
Покупка билетов онлайн	+	-	-
Понятный интерфейс	+	+	+
Ссылки на социальные сети клуба	+	+	+
Онлайн-магазин	+	+	-

клубной атрибутики			

1.3 Постановка задачи

Веб-приложение «Сайт футбольной команды ПГУ» должно иметь функционал представленный ниже.

Со стороны сервера:

– предоставление данных: об игроках, команде, расписании матчей, результатах матчей, новостей клуба, статистики игрока, персонале клуба, турнире, пользователях;

– проверка корректности вводимых данных в базу данных;

– редактирование, добавление и удаление пользователя, игроках, команде, расписания матчей, результатах матчей, статистики игрока, персонале клуба, турнире в базе данных;

– обеспечение целостности данных;

– шифрование паролей;

– обеспечение безопасности данных, редактировать базу данных могут только авторизованные пользователи с ролью администратора.

Со стороны клиента:

– регистрация и авторизация с удобным графическим интерфейсом;

– редактирование, добавление и удаление данных об игроках, команде, расписания матчей, результатах матчей, статистики игроков, персонале клуба, турнире, новостей, пользователях;

– просмотр профиля игрока, статистики игрока, информации о команде, расписания матчей, результатов матчей, информации о персонале клуба, турнирной таблицы, личного кабинета пользователя;

– управление всеми данными через удобный интерфейс административной панели;

– добавление новостей администратором.

Функционал системы представлен на диаграмме вариантов использования (рисунок 7).

Рисунок 7– UML-диаграмма вариантов использования

Минимальные системные требования:

– операционная система: Windows (7, 8, 10);

– браузер: Google Chrome, Mozilla Firefox, Opera (с поддержкой HTML5, CSS3);

– минимальная скорость интернета: 2-3 Мбит/сек;

– минимальный объем оперативной памяти: 1 ГБ (рекомендуется 2 ГБ и выше);

– минимальный объем свободного места на жестком диске: 150 МБ.

2 Выбор технологий и средств разработки приложения

В этом разделе мы рассмотрим технологии, язык программирования и среду разработки, которые были выбраны для написания веб-сайта футбольной команды ПГУ.

Технологии и средства разработки:

- Java;
- Spring framework;
- Spring JPA;
- Spring Security;
- Intellij IDEA;
- Apache Tomcat;
- Bootstrap
- PostgreSQL;
- pgAdmin4;

2.1 JAVA

Java – это широко распространённый объектно-ориентированный язык программирования, предназначенный для создания сетевых и кроссплатформенных веб-приложений. Это быстрый, безопасный и надежный язык программирования. *Java* представляет собой не только язык, но и полноценную платформу для разработки *Java*-приложений. Язык предоставляет богатую стандартную библиотеку классов и методов, обеспечивающую поддержку сетевых протоколов, баз данных, графики и других распространённых задач программирования.

2.2 Spring Framework

Spring Framework – универсальный фреймворк с открытым исходным кодом для Java-платформы. Его уникальность в том, что он имеет огромный функционал

возможностей и значительно упрощает и ускоряет процесс разработки, а также отвечает за качество и надежность приложения.

Spring Framework включает:

– внедрение зависимостей: Оно позволяет объединять несколько частей приложения, а также значительно облегчает управление ими. Зависимости передаются внутрь объектов извне. Код становится гибким, удобным для тестирования и имеет возможность расширения.

– управление зависимостями: Это позволяет сделать процесс разработки гораздо проще и значительно повысить качество. Необходимые зависимости предоставляются контейнеру, а из него автоматически подставляются в те места, где они требуются.

– управление транзакциями: Этот механизм значительно позволяет управлять транзакциями в приложении. С его помощью упрощается работа с базами данных и повышаются меры защиты и безопасности информации.

Spring Framework также поддерживает различные стандарты и спецификации, такие как Java EE, JPA и другие. Таким образом, Spring Framework предлагает различные инструменты и модули, которые упрощают разработку приложений, делая ее более эффективной и гибкой [4].

2.3 Spring JPA

Spring JPA – это модуль Spring Framework, с помощью которого работа в приложениях на Java с базами данных становится проще и удобнее. Он предоставляет возможность работать с различными СУБД и облегчает процесс работы с данными.

Возможности Spring JPA:

– репозитории: Позволяет определить интерфейсы, представляющие уровень доступа к данным. Интерфейсы автоматически реализуются Spring JPA во время выполнения. Это позволяет упростить процесс работы с данными и повысить расширяемость;

– автоматическая генерация запросов: Spring JPA может автоматически генерировать запросы на основе названий методов в репозиториях. Это значительно сокращает объем кода при выполнении запроса к базе данных.

– транзакции: Spring JPA предоставляет механизмы для управления транзакциями в приложении. Это позволяет повысить надежность и безопасность;

– аннотации: Spring JPA использует аннотации для отображения Java-объектов на таблицы базы данных, что уменьшает код.

2.4 Spring Security

Spring security – это фреймворк для обеспечения безопасности веб-приложений. Он предоставляет механизмы для аутентификации и авторизации пользователей, защиты от атак и обработки ошибок безопасности.

Возможности Spring Security:

– аутентификация: Spring Security предоставляет механизмы для аутентификации пользователей, включая поддержку различных методов аутентификации, таких как базовая аутентификация, форма входа и OAuth;

– авторизация: Spring Security предоставляет механизмы для авторизации пользователей, включая поддержку различных методов авторизации, таких как ролевая авторизация и правила авторизации;

– защита от атак: Spring Security предоставляет механизмы для защиты приложения от различных типов атак, включая атаки на сессии, CSRF-атаки и XSS-атаки.

Основные компоненты Spring Security включают:

– authentication: компонент, который предоставляет механизмы для аутентификации пользователей;

– authorization: компонент, который предоставляет механизмы для авторизации пользователей;

– web Security: компонент, который предоставляет механизмы для защиты веб-приложения;

– config: компонент, который предоставляет механизмы для настройки Spring Security.

2.5 IntelliJ IDEA

IntelliJ IDEA от JetBrains является интегрированной средой разработки для Java и других JVM-языков, таких как Kotlin, Groovy и Scala. Она предоставляет множество инструментов и функций для разработки программного обеспечения.

Её особенность – инструменты рефакторинга, которые позволяют быстро и безопасно изменять структуру кода: переименовывать переменные и методы, извлекать методы, преобразовывать анонимные классы. Это помогает упростить код и повысить его читабельность. Также она интегрируется с популярными сборщиками проектов, такими как Maven, Gradle и Ant, что позволяет облегчить процесс сборки и управления зависимостями проектов.

IntelliJ интегрируется с популярными серверами приложений, такими как Tomcat, GlassFish, JBoss и другими.

2.6 Apache Tomcat

Apache Tomcat — это контейнер сервлетов с открытым исходным кодом, разрабатываемый Apache Software Foundation. Он предназначен для тестирования, отладки и исполнения веб-приложений на основе Java.

Tomcat может работать как самостоятельный веб-сервер со статическими страницами и с динамическими запросами. Также он используется в сочетании с традиционными веб-серверами, такими как Apache HTTP Server или Nginx.

Основные функции Apache Tomcat:

– создаёт среду для запуска сервлетов;

– задаёт параметры сессий;

– обеспечивает обмен данными между сервлетами и клиентами;

– проводит идентификацию и авторизацию клиентов;

– управляет метаданными.

Tomcat является легким и простым в использовании сервлет-контейнером, который может работать на различных операционных системах, включая Windows, Linux и macOS.

2.7 Bootstrap

Bootstrap – это бесплатный фреймворк для разработки веб-сайтов, созданный Twitter. Он использует HTML, CSS и JavaScript для создания веб-сайтов.

Данный фреймворк предоставляет разработчикам готовые компоненты, таких как кнопки, формы, таблицы, навигационные панели, модальные окна и т.д. Кроме того, он предоставляет множество классов CSS, которые позволяют быстро и легко настроить стиль веб-сайта.

Одним из основных преимуществ Bootstrap является его адаптивный дизайн. Это означает, что веб-сайт будет автоматически адаптироваться к различным экранам и устройствам, таким как мобильные телефоны и планшеты.

2.8 PostgreSQL

PostgreSQL – это мощная объектно-реляционная система управления базами данных (СУБД). Она предоставляет широкий спектр функций и возможностей для эффективной работы с данными, включая хранение, организацию, управление и анализ. Эта база данных является клиент-серверной системой. Клиентское приложение подключается к серверу PostgreSQL для выполнения операций с базой данных. Сервер PostgreSQL состоит из нескольких процессов: процесс главного сервера, процесс управления соединениями и несколько рабочих процессов, которые обрабатывают запросы на операции с базой данных.

Описание основных компонентов и функций PostgreSQL:

– язык запросов. Поддерживает множество языков запросов, включая SQL, которая является стандартным языком для взаимодействия с большинством СУБД. PostgreSQL также поддерживает Procedural Language/SQL (PL/SQL),

который предоставляет возможность написания процедур и функций на SQL, а также другие языки, такие как Perl, Python, Tcl, Java и C;

– хранение данных. Используется объектно-реляционную модель хранения данных, что означает, что данные хранятся в таблицах, которые могут содержать объекты, такие как строки, столбцы и индексы. PostgreSQL также поддерживает хранение данных в виде JSON-документов, что позволяет хранить данные с более сложной структурой;

транзакции. PostgreSQL поддерживает транзакции, которые позволяют выполнять группу операций с базой данных как единое целое. Если одна из операций в транзакции завершается неудачно, все операции отменяются.

2.9 pgAdmin4

PgAdmin 4 – это бесплатное кросс-платформенное приложение для управления базами данных PostgreSQL. Оно предоставляет удобный и интуитивно понятный интерфейс для работы с базами данных, администрирования и мониторинга.

3 Архитектура веб-приложения «Сайт футбольной команды»

В этом разделе рассмотрим архитектуру нашего веб-приложения и взаимодействие его частей между собой. Для нашего приложения выбрана архитектура MVC (Model-View-Controller).

MVC – это способ разработки приложения, при котором само приложение делится на три компонента: модель (Model), представление (View), контроллер (Controller). Архитектура приложения представлена на рисунке 8.

Модель – это ряд классов, который реализует всю бизнес-логику приложения. Они обрабатывают данные (сущности), размещают их в базе данных, читают из базы данных и позволяют взаимодействовать объектам, составляющим такие данные. В нашем случае модель отвечает за хранение информации об игроках, тренерах, персонале, пользователях, расписании, турнирной таблицы. Также благодаря модели будет проводиться проверка прав доступа пользователя к определенным данным. В приложении в слое Model отдельно выделен сервисный слой (в него входят все классы Service, Handler, Convertor, Contract).

Класс Service – вызывает Repository для обращения к БД. Использует Entity – представление таблиц в БД, чтобы передавать данные в БД нужной структуры, а также получать данные из БД такой же структуры. В классе Service реализованы методы которые получают, изменяют, создают данные в БД, все методы используют транзакции (@Trasactional) для предотвращения ошибок при записи в БД, если происходит ошибка в процессе выполнения записи, то процесс останавливается и состояние БД не изменяется.

Класс Handler – вызывает методы Service класса, использует методы класса Convertor и Contract. Здесь прописана основная бизнес логика, оркистровка вызова нескольких Service классов для разных сущностей, преобразование данных перед передачей в Service и также преобразование данных перед отправкой в Controller через классы Convertor и Contract.

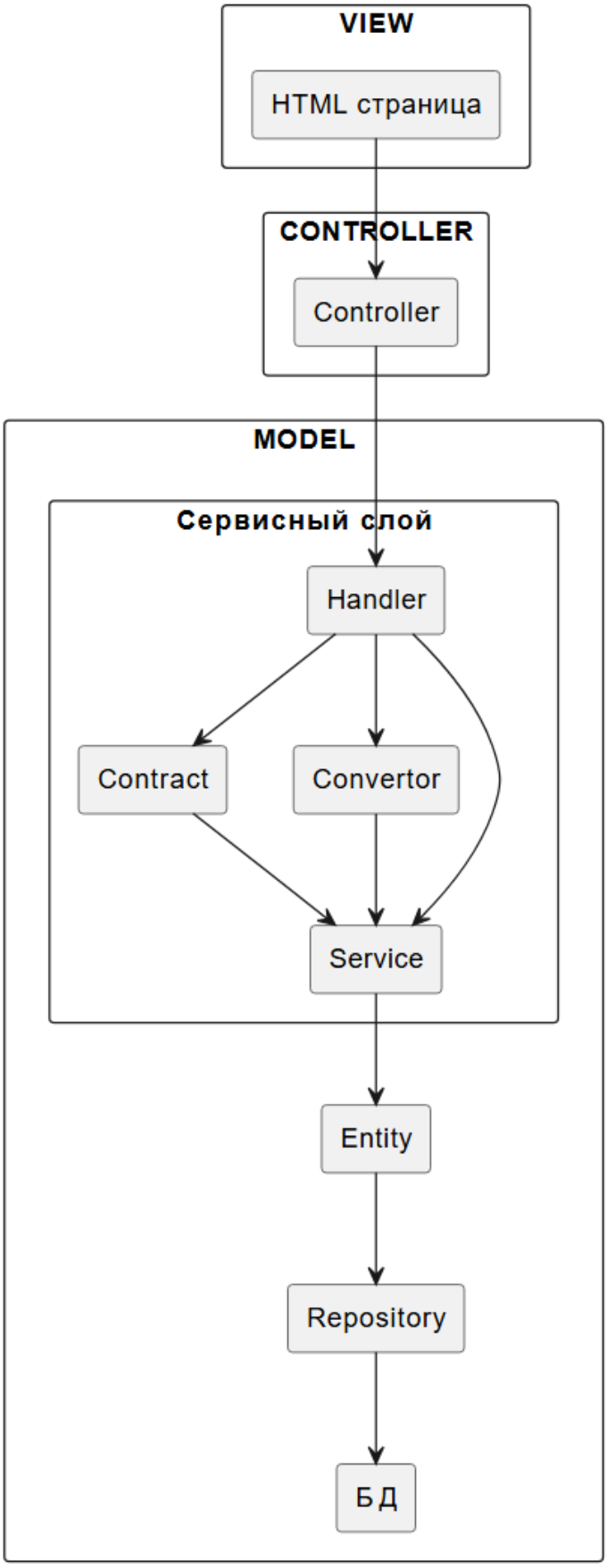


Рисунок 8 – Архитектура приложения

Класс Contract – это класс, технических объектов передачи данных, он не содержит логики, но участвуют в передаче между слоями. В нем могут содержаться такие классы, как PlayerView – ответ который будет приходить от сервера к клиенту, и PlayerReq – запрос от клиента к серверу.

Класс Convertor служит для преобразование Entity в ответ от сервера (PlayerView), чтобы клиент получал только необходимые данные, а не все которые могут содержаться в таблицах БД.

Представление – все классы и шаблоны, которые отвечают за интерфейс взаимодействия с пользователями (User Interface, UI). В нем находится HTML - код, с помощью которого будет происходить взаимодействие с пользователем.

Контроллер связывает модель и представление между собой, принимает HTTP-запросы от клиента (браузера), обрабатывает их и возвращает ответ. Осуществляет маршрутизацию (@GetMapping, @PostMapping и т.д.), валидацию входных данных (с помощью @Valid и BindingResult), вызов соответствующего Handler.

Такая структура дает возможность для дальнейшего масштабирования приложения и разделения его на отдельные микросервисы.

4 Реализация приложения

В этой части будет представлен процесс реализации нашего веб-приложения. Оно представлено для пользователей, интересующихся футбольной институтской командой Пензенского государственного университета. Пользователи могут просматривать информацию о клубе, узнавать новости о команде, об игроках, просматривать турнирную таблицу, следить за ходом соревнований.

В данном разделе будут рассмотрены основные компоненты, функционал веб-приложения, сущности, которые находятся в базе данных. Будет проведено тестирование приложения, а также будет предоставлена полная информация о его разработке и реализации.

4.1 Описание работы приложения

Порядок работы приложения такой: пользователь выбирает, что ему сделать на клиентской части приложения. Она написана на HTML с использованием CSS стилей и Thymeleaf – механизм шаблонов Java. По результату действий клиента сервер получает некий запрос на выполнение задачи. Если возникает ошибка, то сервер возвращает клиенту информацию об ошибке, операция прерывается. В случае, если запрос успешно принят, сервер обращается к базе данных и в зависимости от выбранной операции может вернуть пользователю данные, создать, редактировать или удалить их. В случае успешного запроса сервер вернет пользователю данные, либо выполнит нужную операцию. Если в ходе обращения к базе данных возникнет ошибка, сервер выполнит обработку данной ошибки и вернет в клиентскую часть информацию об ошибке.

Порядок выполняемых действий изображен на рисунке 9.

Развертывание системы и распределение ее компонентов по физическим узлам представлены на рисунке 10.

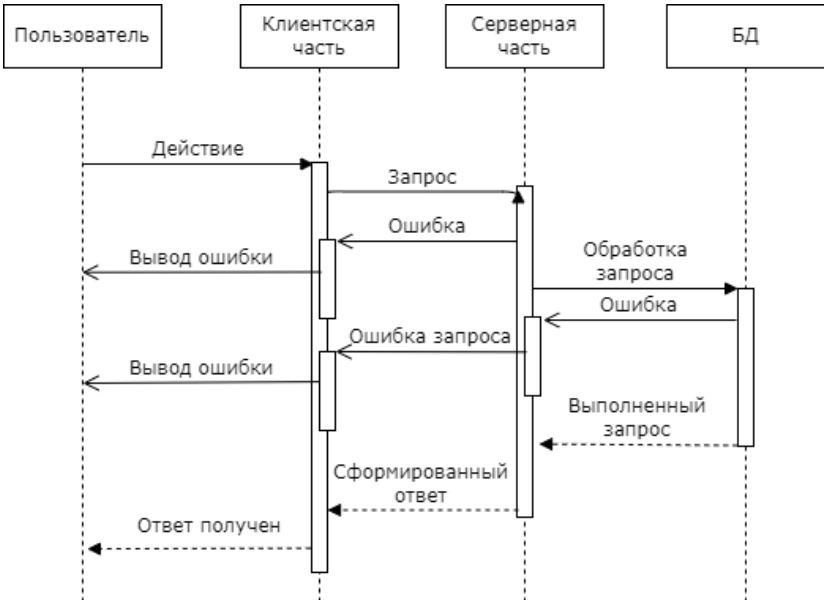


Рисунок 9–*UML* диаграмма последовательности функционирования приложения

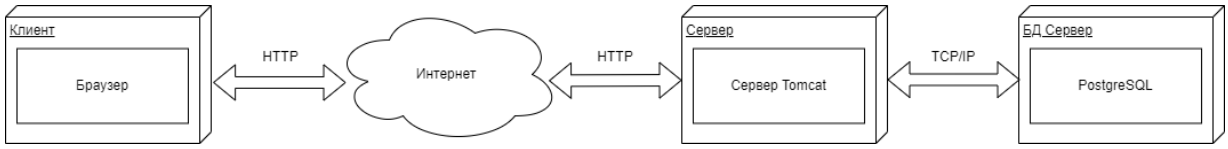


Рисунок 10–*UML* диаграмма развертывания системы

4.2 Структура БД

Хранение данных веб-приложения, это очень важный аспект на этапе его реализации. Необходимо разобраться, какие сущности и связи будут присутствовать в базе данных. После тщательного анализа была разработана структура для оптимального функционирования нашей базы данных. Ниже рассматриваются основные таблицы, атрибуты и связи между ними, которые были разработаны в рамках данной работы [5].

Для реализации веб-приложения и хранения его данных используются следующие таблицы:

– таблица *fc_team* из таблицы 2. В ней будет содержатся информация о командах: ее название, домашний стадион, логотип;

Таблица 2–атрибуты сущности fc_team

Поле	Тип данных	Описание	Ключ	Пример
id	*SERIAL*	Идентификатор команды	PK	1
team_name	*VARCHAR(100)*	Название команды		ФК «ПГУ»
team_short_name	*VARCHAR(10)*	Краткое название команды		ПГУ
stadium	*VARCHAR(100)*	Стадион		Запрудный
logo_url	*VARCHAR(255)*	URL-адрес логотипа команды		Путь к файлу

– таблица *fc_tournament* из таблицы 3. Она будет содержать всю информацию о проходящем соревновании;

Таблица 3 – атрибуты сущности *fc_tournament*

Поле	Тип данных	Описание	Ключ	Пример
id	*SERIAL*	Идентификатор турнира	PK	1
tournament_name	*VARCHAR(100)*	Название турнира		Чемпионат России по футболу
season	*VARCHAR(20)*	Сезон турнира		2024/2025

– таблица *fc_user* из таблицы 4. Будет содержать информацию о пользователях, которая будет использоваться для авторизации и идентификации пользователя в приложении;

Таблица 4–атрибуты таблицы fc_user

Поле	Тип данных	Описание	Ключ	Пример
id	*SERIAL*	Идентификатор пользователя	PK	1
username	*VARCHAR(50)*	Имя пользователя		Дмитрий Королёв
login	*VARCHAR(50)*	Логин пользователя (уникальный)		kerol.dim
email	*VARCHAR(100)*	Email пользователя (уникальный)		kerol29@mail.ru
password	*VARCHAR(255)*	Пароль пользователя		хешированный пароль
user_role_id	*INTEGER*	ID роли пользователя (FK)	FK	ссылка на fc_role_user
created_at	*TIMESTAMP*	Дата создания пользователя		2024-10-27 10:00:00
updated_at	*TIMESTAMP*	Дата обновления пользователя		2024-10-27 10:00:00

– таблица *fc_role_user* из таблицы 5. Здесь будут содержаться роли, которые выдаются пользователям при регистрации, для определения их прав доступа в приложении;

Таблица 5–атрибуты сущности fc_user_role

Поле	Тип данных	Описание	Ключ	Пример
id	*SERIAL*	Идентификатор роли	PK	1
role_name	*VARCHAR(50)*	Название роли (уникальное)		Администратор
discription	*VARCHAR(255)*	Описание роли (уникальное)		Полный доступ

– таблица *fc_match* из таблицы 6. Будет хранить всю информацию о прошедших и предстоящих матчах команд;

Таблица 6–атрибуты таблицы fc_match

Поле	Тип данных	Описание	Ключ	Пример
id	*SERIAL*	Идентификатор матча	PK	1
home_team_id	*INTEGER*	ID домашней команды (FK)	FK	1 (ссылка на fc_team)
away_team_id	*INTEGER*	ID гостевой команды (FK)	FK	2 (ссылка на fc_team)
score	*VARCHAR(10)*	Счет матча		2:1
location	*VARCHAR(100)*	Место проведения		Запрудный
match_status	*VARCHAR(20)*	Статус матча		Завершен
date_time	*TIMESTAMP*	Дата и время матча		2024-10-27 17:00:00

Поле	Тип данных	Описание	Ключ	Пример
tournament_id	*INTEGER*	ID турнира (FK)	FK	1 (ссылка на fc_tournament)

– таблица *fc_player* из таблицы 7. Будет содержать подробную информацию об игроках;

Таблица 7–атрибуты сущности fc_player

Поле	Тип данных	Описание	Ключ	Пример
id	*SERIAL*	Идентификатор игрока	PK	1
surname	*VARCHAR(50)*	Фамилия игрока		Щеглов
name	*VARCHAR(50)*	Имя игрока		Дмитрий
patronymic	*VARCHAR(50)*	Отчество игрока		Алексеевич
birth_date	*DATE*	Дата рождения игрока		2003-07-26
nationality	*VARCHAR(50)*	Национальность игрока		Россия
player_number	*INTEGER*	Игровой номер игрока		35
position	*VARCHAR(20)*	Позиция игрока		Вратарь
photo_url	*VARCHAR(255)*	URL-адрес фотографии игрока		Путь к файлу
team_id	*INTEGER*	ID команды игрока	FK	1 (ссылка на fc_team)

– таблица *fc_staff* из таблицы 8. Будет использоваться для хранения данных персонала футбольной команды;

Таблица 8–атрибуты таблицы fc_staff

Поле	Тип данных	Описание	Ключ	Пример
id	*SERIAL*	Идентификатор сотрудника	PK	1
surname	*VARCHAR(50)*	Фамилия сотрудника		Муромцев
name	*VARCHAR(50)*	Имя сотрудника		Кирилл
patronymic	*VARCHAR(50)*	Отчество сотрудника		Валерьевич
birth_date	*DATE*	Дата рождения сотрудника		2003-07-26
staff_role	*VARCHAR(20)*	Роль сотрудника (тренер, администратор, врач…)		Тренер
photo_url	*VARCHAR(255)*	URL-адрес фотографии сотрудника		Путь к файлу
team_id	*INTEGER*	ID команды сотрудника	FK	1 (ссылка на fc_team)

– таблица *fc_results* из таблицы 9. В ней будет содержаться информация о результатах сыгранных матчей;

Таблица 9–атрибуты сущности fc_results

Поле	Тип данных	Описание	Ключ	Пример
id	*SERIAL*	Идентификатор результатов	PK	1
position	*INTEGER*	Позиция команды в турнире		1
played	*INTEGER*	Количество сыгранных матчей		30
wins	*INTEGER*	Количество побед		19
draws	*INTEGER*	Количество ничьих		6
losses	*INTEGER*	Количество поражений		5
points	*INTEGER*	Количество набранных очков		63
team_id	*INTEGER*	ID команды	FK	1 (ссылка на fc_team)
tournament_id	*INTEGER*	ID турнира	FK	1(ссылка на fc_tournament)

– таблица *fc_player_statistics* из таблицы 10. Будет хранить индивидуальную статистику каждого игрока.

– таблица *fc_news* из таблицы 11. Будет использоваться для хранения новостей команды.

– таблица *fc_comments* из таблицы 12. Будет хранить комментарии пользователей.

Таблица 10–атрибуты сущности fc_player_statistics

Поле	Тип данных	Описание	Ключ	Пример
id	*SERIAL*	Идентификатор статистики	PK	1
player_id	*INTEGER*	ID игрока	FK	5
match_id	*INTEGER*	ID матча	FK	4
goals	*INTEGER*	Количество голов		1
assists	*INTEGER*	Количество голевых передач		2
yellow_cards	*INTEGER*	Количество желтых карточек		0
red_cards	*INTEGER*	Количество красных карточек		0
played_minutes	*INTEGER*	Количество сыгранных минут		90
shots	*INTEGER*	Количество ударов		2
passes	*INTEGER*	Количество передач		41
season	*DATE*	Сезон		2024-08-01

На рисунках 11, 12 изображена схема базы данных, разрабатываемого приложения, для наглядного рассмотрения взаимодействия таблиц.

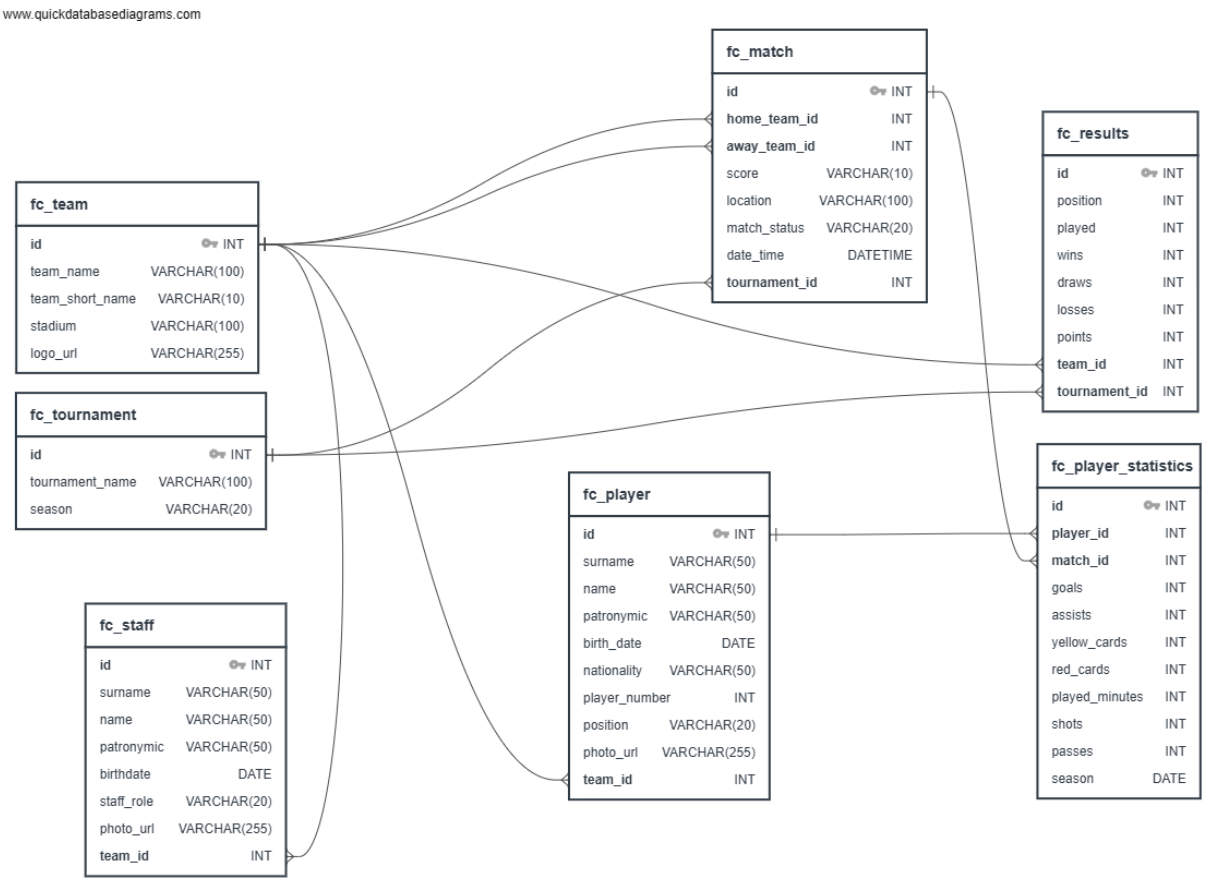


Рисунок 11 – Схема БД

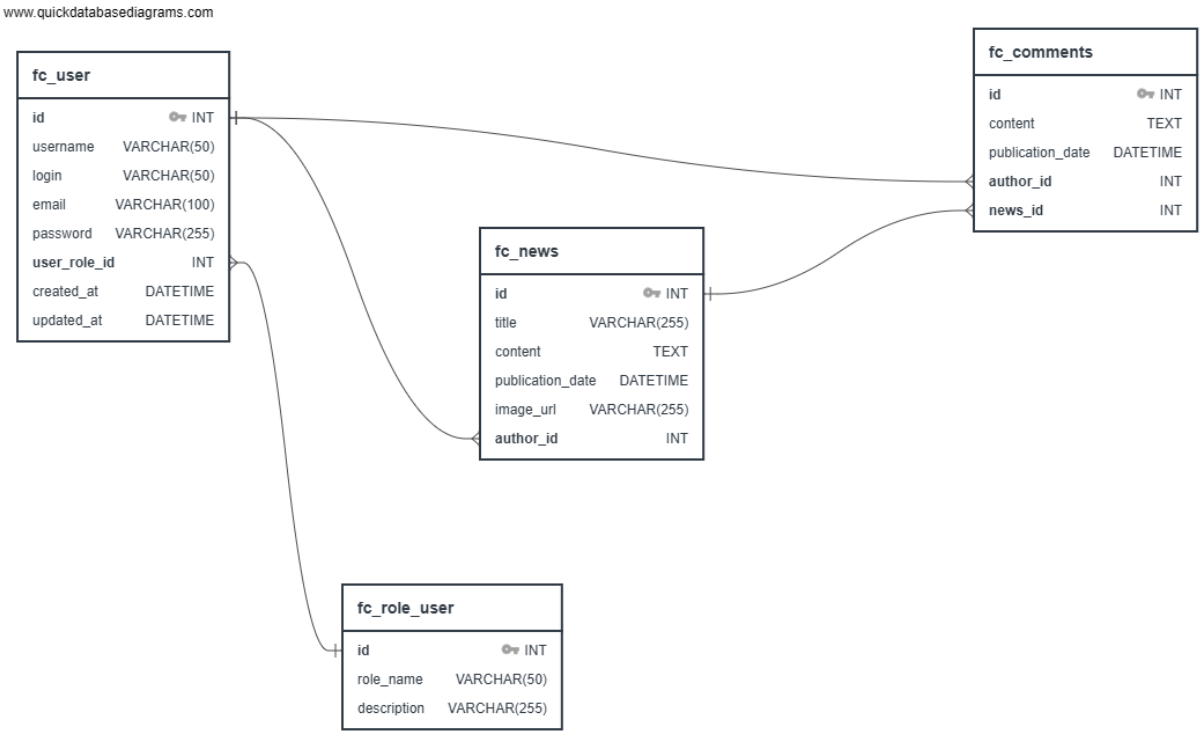


Рисунок 12 – Схема БД (продолжение)

4.3 Функционал приложения

Функции разрабатываемого веб-приложения можно разделить на следующие группы:

– неавторизованный пользователь;

– авторизованный пользователь;

– администратор приложения.

Неавторизованный пользователь обладает правами только на просмотр данных, которые предоставляются сервером. Такой пользователь не может передавать данные на сервер.

Доступный функционал:

– просмотр новостей, расписания матчей, персонала, состава команды, турнирной таблицы, результатов матчей, информацию об игроке, статистику игрока, общую информацию о команде;

– регистрация на сайте;

– авторизация.

Авторизованный пользователь будет обладать тем же функционалом что и неавторизованный пользователь, но с дополнительной возможностью оставлять комментарии под новостями команды и просматривать комментарии, которые написали другие пользователи, также имеет возможность редактировать информацию о себе.

Доступный функционал:

– все права на просмотр, как у неавторизованного пользователя;

– написание комментариев под новостями команды;

– просмотр комментариев других пользователей;

– редактирование данных учетной записи;

– удаление аккаунта;

– выход из аккаунта.

Администратор сайта обладает полным набором прав для управления веб-приложением. Также, как и пользователи, имеет право на просмотр всей

информации о команде. Может просматривать информацию о зарегистрированных пользователях, удалять аккаунты. Есть возможность выдать права администратора пользователю. Создавать, редактировать и удалять информацию о команде, игроках, персонале, турнире, матчах, результатах, статистике. Может добавлять, редактировать и удалять новости, комментарии пользователей.

Доступный функционал:

- просмотр всей информации на сайте;
- создание, редактирование и удаление всей информации о команде;
- просмотр и удаление всей информации о пользователях;
- выдача прав администратора;
- редактирование данных своей учетной записи;
- создание, редактирование и удаление новостей;
- удаление комментариев пользователей;
- выход из аккаунта.

4.4 Основные компоненты

В реализуемом клиент-серверном приложение можно выделить три основных компонента:

- controller;
- service;
- model.

Controller - это связующее звено между service и model. Классы controller осуществляют маршрутизацию, принимают HTTP-запросы от клиента (браузера) к серверу и передают их в модель для обработки и сохранения, и возвращают ответ клиенту.

Service - набор классов, методы которых выполняют различные операции, такие как чтение, запись или изменение данных в базе данных, обработка и анализ данных. Полученные данные service слой передает в model.

Model - набор классов, реализующих всю бизнес-логику приложения. Эти классы отвечают за обработку данных (сущностей), размещение их в БД, чтение из БД, а также за взаимодействие между самими объектами, составляющими такие данные.

4.4.1 Controller

Рассмотрим один из классов контроллеров на примере AuthorizationController. Данный класс отвечает за регистрацию и авторизацию пользователя, работает с моделью User, которая получает и сохраняет данные о пользователях в базе данных. В зависимости от HTTP-запроса клиента контроллер выбирает к какому service классу обратиться, для взаимодействия с БД. Также при обращении к контроллеру производится первичная валидация данных от клиента. Для этого в нем реализованы следующие методы:

— registrationPage();

— registrationUser();

— loginPage().

Метод registrationPage() обрабатывает get-запрос, для перехода на страницу с формой регистрации. Контроллер формирует HTML-страницу с переданной в нее пустой моделью запроса UserCreateReq.

Листинг 1. *registrationPage():*

```
@ResponseStatus(HttpStatus.OK)
@GetMapping("/singup")
public String registrationPage(Model model){

    model.addAttribute("userCreateReq",
          new UserCreateReq("", "", ""));

  return "singup";
}
```

Метод registrationUser() обрабатывает post-запрос, который добавляет пользователя в базу данных, после успешной пройденной валидации данных, которые переданы через модель запроса UserCreateReq. Если были переданы

некорректные данные, то метод вернется на страницу регистрации для повторного ввода данных. В противном случае все полученные данные, контроллер передает в сервисный слой UserHandler, где производится их обработка. Затем обработанные данные передаются в сервисный слой UserService, в нем они, используя модель User, сохраняются в базу данных. После успешного выполнения происходит перенаправление на get-метод loginPage().

Листинг 2. registrationUser ():

```
@ResponseStatus(HttpStatus.CREATED)
@PostMapping("/singup")
public String registrationUser(
          @Valid @ModelAttribute @NotNull UserCreateReq req,
          BindingResult bindingResult,
          Model model) {
    if (bindingResult.hasErrors()) {return "singup";}

  UserView user = handler.handlerCreateUser(req);

  if(user == null){
      model.addAttribute("error", "Пользователь с email: " +
      req.email() + " уже существует");

      return "singup";
  }

    return "redirect:/auth/login";
}
```

Метод loginPage() представляет собой обработчик get-запроса, который отвечает за переход на страницу с формой авторизации. Логика авторизации, представленная ниже в листинге 4, реализована через стандартные методы SpringSecurity. Если пользователь отправил корректные данные через post-запрос будет произведена авторизация и перенаправление на главную страницу приложения, а также сформирован объект Principal, с его помощью система будет определять какой пользователь был авторизован и выданные ему права. В

противном случае пользователь остается на странице с формой авторизации, где будет показана ошибка.

Листинг 3. loginPage ():

```
@ResponseStatus(HttpStatus.OK)
@GetMapping("/login")
public String loginPage(
          @RequestParam(value = "error", required = false)
          boolean error,
          Model model) {
    if (error) {
        model.addAttribute("errorMsg", "Неправильный логин или
          пароль");
  }
  return "login";
}
```

Листинг 4. Авторизация через *SpringSecurity*:

```
@Bean
public SecurityFilterChain securityFilterChain(
          HttpSecurity http) throws Exception {
    http
          .csrf(csrf -> csrf.disable())
          .formLogin((form) -> form
                    .defaultSuccessUrl("/")
                    .loginPage("/auth/login")
                    .usernameParameter("email")
                    .passwordParameter("password")
                    .failureUrl("/auth/login?error=true")
                    .permitAll()
               )
               .logout((logout) -> logout
                    .logoutUrl("/logout")
                    .logoutSuccessUrl("/")
                    .invalidateHttpSession(true)
                    .deleteCookies("JSESSIONID")
```

```
                        .permitAll());
        return http.build();
    }
```

4.4.2 Service

Сервисный слой является посредниками между контроллерами и репозиториями, обрабатывая данные и выполняя операции над ними. В данном пункте будут рассмотрены классы UserHandler, UserService, UserConverter.

Рассмотрим процесс регистрации пользователя в системе. Из контроллера вызывается метод handlerCreateUser(), которой находится в классе UserHandler. Он принимает один параметр - UserCreateReq req, модель данных запроса, в котором заданы обязательные поля (username, email, password) для создания пользователя. Первоначально, используя метод getUserByEmail (String email), который находится в классе UserService, ищем пользователя в БД с переданным email. Если такой пользователь уже существует в базе данных, метод возвращает null. В таком случае клиенту будет показана ошибка регистрации. Если пользователь с таким email не найден, процесс создания продолжается. Создается модель User, куда записываются данные из req, зашифровывается пароль, и присваивается роль по умолчанию (пользователь).

Получившаяся модель передается в метод saveUser(User user), который находится в классе UserService. В этом методе идет обращение к репозиторию, для сохранения данных в БД. Метод помечен аннотацией @Transactional, которая необходима для обеспечения целостности базы данных, если в момент изменения или записи данных в БД произойдет ошибка, то все примененные изменения будут отменены, БД вернется в исходный вид.

После успешного добавления нового пользователя в БД, объект User передается в класс UserConverter. Этот класс нужен для передачи данных клиенту только необходимой информации об объекте. Для этого используем метод toView(), который принимает объект User и возвращает объект UserView. Это позволяет ограничить доступ клиенту к полным данным сущности, тем самым обеспечивает конфиденциальность данных. Все описанные методы классов

представлены в листинге ниже.

Листинг 5. *UserHandler.handlerCreateUser* ():

```
public UserView handlerCreateUser(@NonNull UserCreateReq req) {
        if (service.getUserByEmail(req.email()).isPresent()){
            return null;
        }

        final Role defaultRole = roleService
                .getRoleByDescription("Пользователь")
                .orElseThrow(() -> new
                      NotFoundException(messageUtil.
                            getMessage(
                            "role.description.not-found",
                            "Пользователь")));
        final User user = new User();

        user.setNameUser(req.username());
        user.setEmail(req.email());
        user.setPassword(encoder.encode(req.password()));
        user.setUserRole(defaultRole);

        return converter.toView(
                service.saveUser(user)
        );
}
```

Листинг 6. *UserCreateReq*:

```
@Schema(description = "Создание пользователя")
public record UserCreateReq(
        @NotBlank
        String username,
        @NotBlank
        String email,
        @NotBlank
        String password
) {
```

```
}
```

Листинг 7. *UserService. getUserByEmail*():

```
@Transactional(readOnly = true)
public Optional<User> getUserByEmail(@NonNull String email){
        return userRepo.findUserByEmail(email);
}
```

Листинг 8. *UserService. saveUser*():

```
@Transactional
public User saveUser(@NonNull User user){
        try{
            return userRepo.save(user);
        }catch (DataIntegrityViolationException e){
            throw new
                ConflictResourceException(e.getMessage());
        }
}
```

Листинг 9. *UserConverter. toView*():

```
public UserView toView(@NonNull User user){
        return toUserView.convert(user);
}

@Override
    public UserView convert(User source) {
        return new UserView(
                source.getId(),
                source.getNameUser(),
                source.getEmail(),
                source.getPassword()
        );
    }
```

4.4.3 Model

Модель представляет собой сущность, которая содержит данные, необходимые для отображения на пользовательском интерфейсе. Она отвечает за хранение и управление данными, используемыми в приложении.

Например, для сохранения данных модели пользователя в базе данных был создан класс User, который отображает таблицу fc_user в Postgresql. Аннотация @Entity указывает, что класс является сущностью базы данных, а аннотация @Table указывает имя таблицы, которая отображается этой сущностью. Аннотация @Id указывает, что поле id является первичным ключом таблицы, @Column указывают имена столбцов таблицы, которые отображаются соответствующими полями класса. Аннотация @ManyToOne и @OneToMany указывает на отношение между сущностями, в нашем случае - связь многие-к-одному между сущностями User и Role, а также связь один-ко-многим между сущностями User и News, User и Comment.

Листинг 9. *User.java*:

```
@Getter
@Setter
@Entity
@Table(name = "fc_user")
public class User implements UserDetails {
    @Id
    @SequenceGenerator(name = "seq_gen_user", sequenceName = "seq_user", allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "seq_gen_user")
    @Column(name = "id")
    private Integer id;

    @Column(name = "username")
    private String nameUser;

    @Column(name = "email")
    private String email;

    @Column(name = "password")
    private String password;
```

```
        @ManyToOne
        @JoinColumn(name = "user_role_id")
        private Role userRole;

        @OneToMany(mappedBy = "author")
        private List<News> news = new ArrayList<>();

        @OneToMany(mappedBy = "author")
        private List<Comment> comments = new ArrayList<>();

        @CreationTimestamp
        @Column(name = "created_at")
        private LocalDateTime createdAt;

        @UpdateTimestamp
        @Column(name = "updated_at")
        private LocalDateTime updatedAt;

        @Override
        public Collection<? extends GrantedAuthority>
getAuthorities() {
            return Collections.singleton(userRole);
        }

        @Override
        public String getUsername() {
            return email;
        }

        @Override
        public boolean isAccountNonExpired() {
            return true;
        }

        @Override
```

```
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    public boolean hasRole(String roleName) {
        if (roleName == null || roleName.isEmpty()) {
            return false;
        }
        return getAuthorities().stream()
                .map(GrantedAuthority::getAuthority)
                .anyMatch(roleName::equals);
    }
}
```

Для работы с базой данных и сущностью User используется класс UserRepo. Этот класс представляет собой инструмент Spring Data JPA, который дает готовые методы для взаимодействия с базой данных, позволяет добавлять кастомные запросы к БД, также уменьшает количество шаблонного кода. Основная аннотация, которой помечен класс, @Repository отвечает за обработку исключений от JPA и преобразование их в Spring-исключения, также помечает интерфейс, как компонент репозитория (специализированный @Component). Класс наследует интерфейс JpaRepository<User, Integer>, с его помощью используем стандартные методы для работы с БД, он принимает в себя сущность

с которой производится взаимодействие, и тип первичного ключа этой сущности.

Листинг 10. *UserRepo.java*:

```
@Repository
public interface UserRepo extends JpaRepository<User, Integer>
{
    @Query("select u from User u")
    Page<User> findAllUser(Pageable pageable);

    @Query("select u from User u where u.id = :id")
    Optional<User> findById(@Param("id") Integer id);

    @Query("select u from User u where u.nameUser like
%:searchTerm%")
    Page<User> findUserByUsername(@Param("searchTerm") String
searchTerm, Pageable pageable);

    @Query("select u from User u where u.email = :email")
    Optional<User> findUserByEmail(@Param("email") String
email);

    @Query("select u from User u join u.userRole r where
r.description = :description")
    Page<User> findUsersByRole(@Param("description") String
description, Pageable pageable);

}
```

4.4.4 Основные алгоритмы взаимодействия клиента и сервера

Для наглядности алгоритмов взаимодействий были разработаны *UML*-диаграммы деятельностей, которые отображают последовательность действий между клиентской частью и серверной.

Первым рассматриваемым алгоритмом будет регистрация и авторизация. Для использования всех возможностей, предлагаемых приложением, необходимо выполнить регистрацию и дальнейший вход в систему. При попытки

пользователем, не осуществившем вход, использовать функцию, которая доступна только авторизованным пользователям, ему будет предложено войти в приложение. Если клиент не имеет аккаунта в приложении, он может выполнить регистрацию, перейдя на соответствующую страницу и заполнив необходимые поля. Веденные данные отправляются на сервер, где происходит обработка и проверка уникальности информации, если такие данные уже занесены с БД пользователю будет отправлено соответствующее оповещение об этом. В противном случае данные будут сохранены в БД и пользователь будет перенаправлен на страницу авторизации.

Если при выполнении авторизации данные, введенные пользователем, будут отличны от тех, что хранятся в БД, пользователю будет выведено сообщение о некорректных данных. В случае правильно отправленных данных, будет выполнен вход в приложении, после которого у пользователя будет полный доступ ко всем функциям приложения.

Последовательность описанных действий представлены на рисунке 13.

Вторым алгоритмом, который будет рассмотрен, это процесс просмотра информации на сайте, а также возможность комментировать новости. Просмотр информации, в которая интересна пользователю, осуществляется путем перехода на соответствующую страницу сайта. Для просмотра информации пользователю не обязательно выполнять вход в приложение. Если пользователь перешел на страницу отдельной новости и попытается оставить комментарий, системой проходит проверка авторизации пользователя в приложении, если он не выполнил вход, система перенаправляет его на страницу входа и после завершения авторизации, он может отправить свой комментарий.

Последовательность действий по просмотру информации и комментировании новостей представлены на рисунке 14.

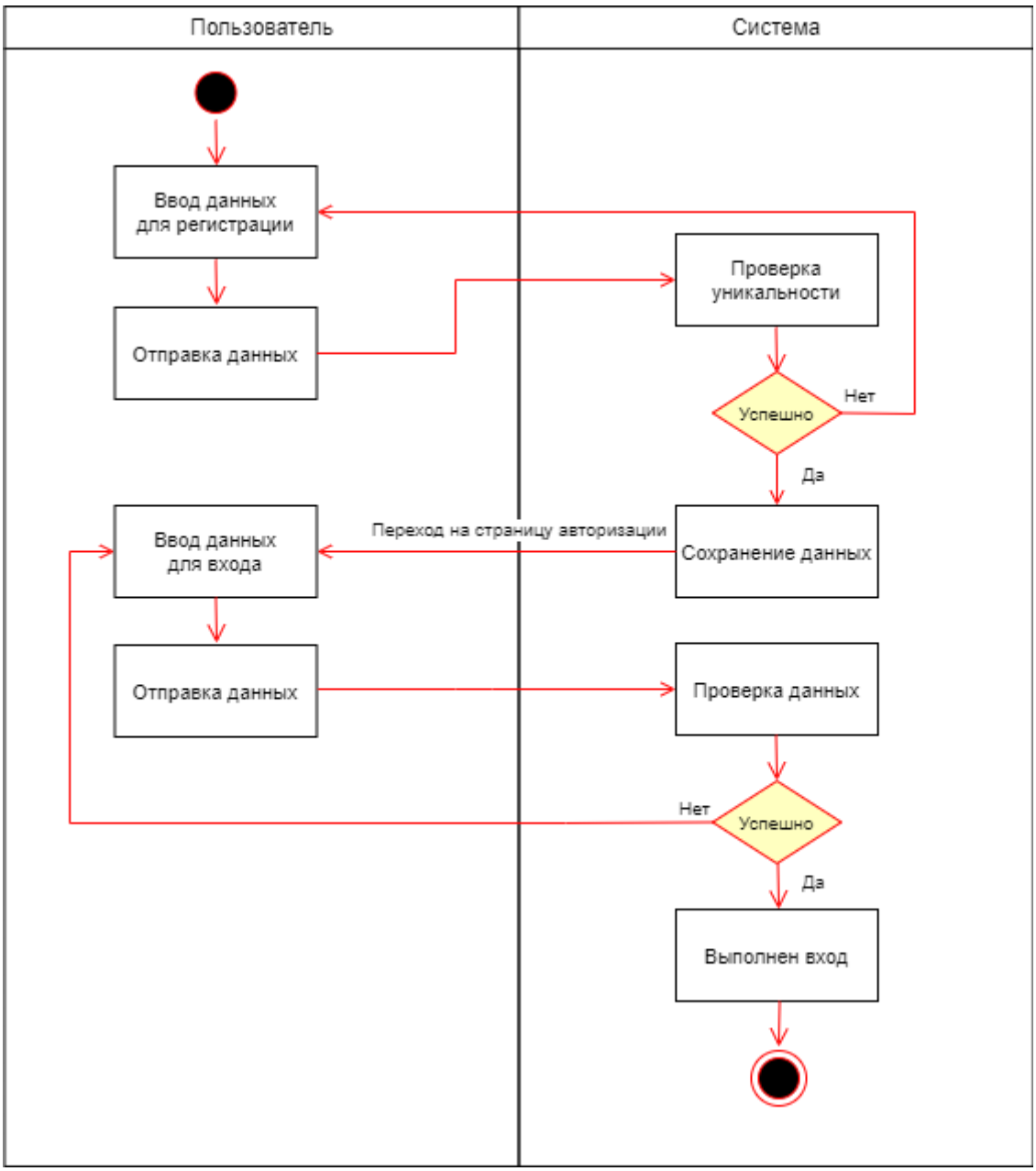


Рисунок 13 – UML диаграмма деятельности регистрации и авторизации

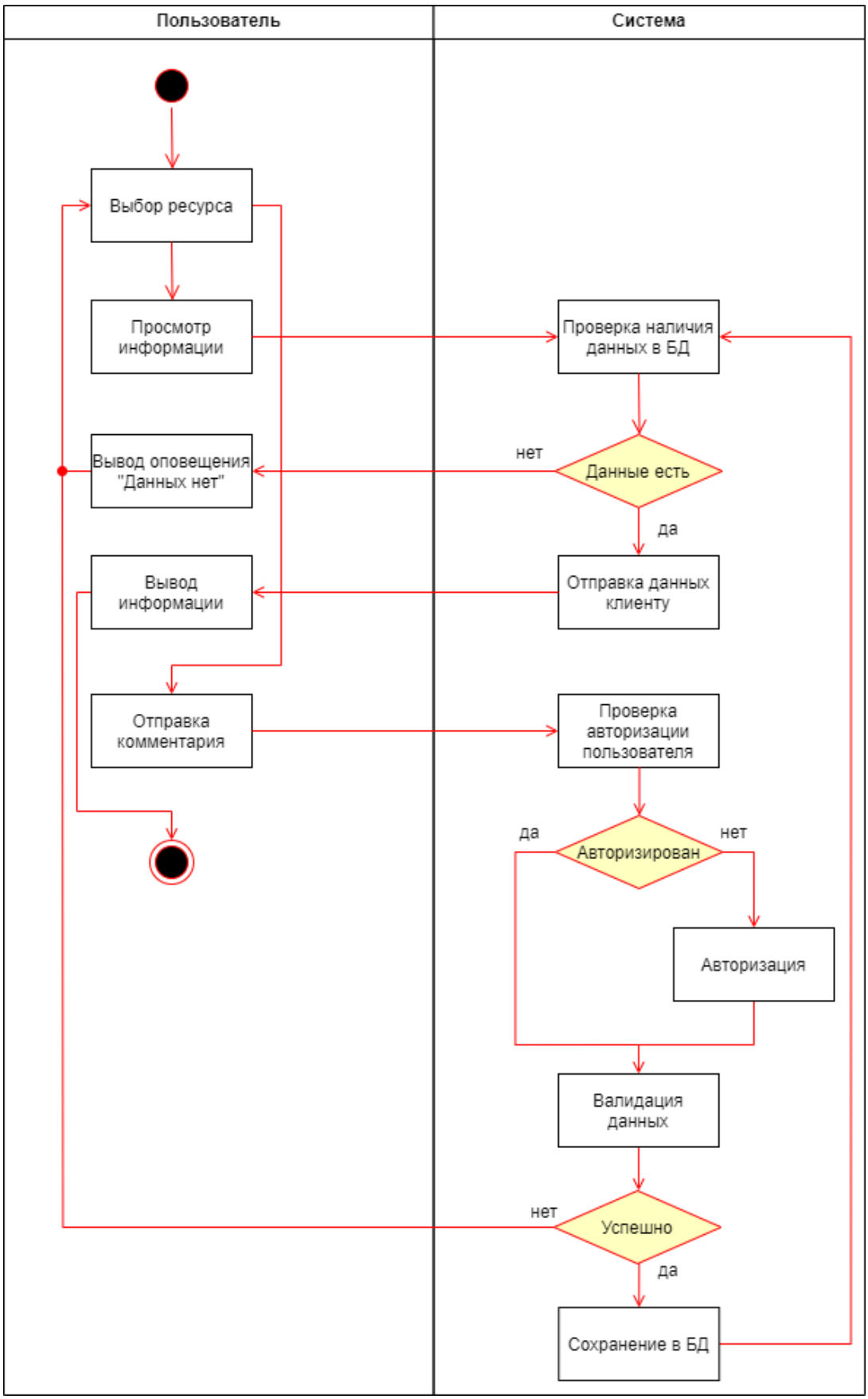


Рисунок 14 – UML диаграмма деятельности просмотра информации на сайте

Еще один из основных алгоритмов, который будет описан в этом пункте, это алгоритм действий, выполняемых администратором системы над информацией содержащийся на сайте. После того, администратор выполнил вход в аккаунт приложения, он может производить такие действия, как просмотр всей информации на сайте, создание новой информации, а также её редактирование и удаление.

Для просмотра, администратору необходимо выбрать нужный ресурс, после чего в системе будет произведен поиск, выбранной информации. Если данные найдены, будет сформирован ответ, который будет показан в браузере. В противном случае пользователь получит сообщение, что данные не найдены в БД.

При создании информации для выбранного ресурса, необходимо заполнить соответствующие поля формы и отправить запрос на создание. После выполнения этих действий в системе происходит валидация данных и если проверка проходит успешно, то информация сохраняется в БД, а пользователя перенаправляет обратно на выбранный им ранее ресурс, где снова проходит проверка на наличие информации и будет показаны добавленные им данные. В том случае если данные не прошли валидацию, администратор должен ввести корректную информацию и повторить попытку добавления.

Для редактирования информации выбранного ресурса, администратор должен заполнить поля формы новыми данными и отправить запрос на изменение. После этого в системе происходят действия, как при добавлении данных, описанные выше.

Чтобы удалить данные из БД, администратор должен выбрать ресурс, подтвердить свои действия и в систему будет направлен запрос на удаление. В системе выполняются действия по удалению данных, после чего на стороне клиента будет показан обновленный список информации.

На рисунке 15 представлена диаграмма деятельности администратора приложения.

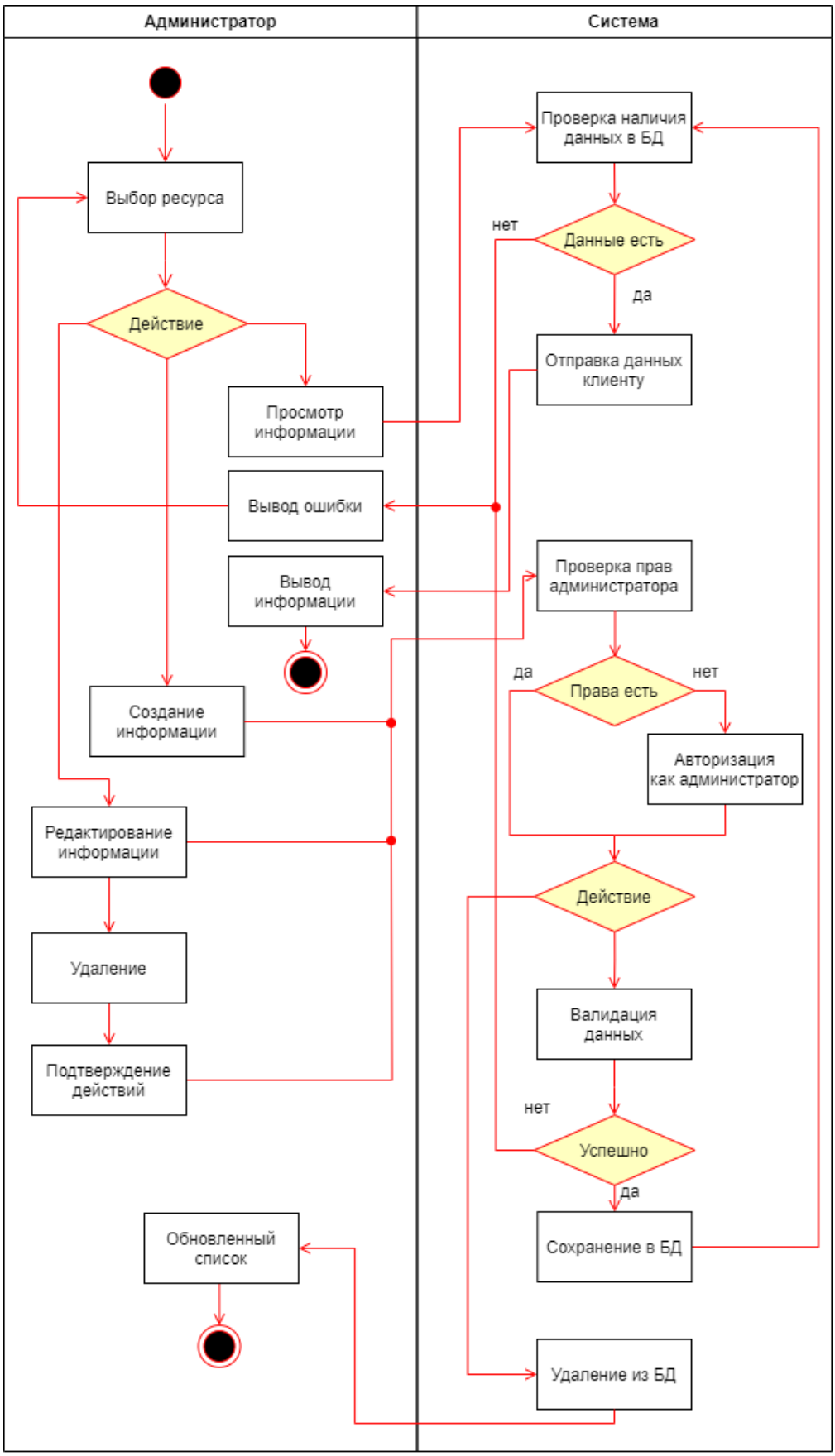


Рисунок 15 – UML диаграмма деятельности администратора

5 Тестирование приложения

Для проверки работоспособности клиент-серверного приложения необходимо провести его тестирование. Тестирование будет состоять из двух групп: тестирование серверной части, тестирование клиентской части.

Тестирование серверной части включает в себя проверку api-методов на сервере. Для этого будет использован инструмент swagger. Тестирование состоит из следующих действий:

– тестирование post-методов контролеров;

– тестирование put-методов контролеров;

– тестирование delete-методов контролеров.

Тестирование клиентской части включает в себя проверку интерфейса приложения и отображения информации в браузере. Для проведения тестирования будет использован браузер chrome. Тестирование состоит из следующих действий:

– регистрация пользователя;

– авторизация пользователя;

– просмотр информации;

– выход из аккаунта пользователя.

5.1 Тестирование серверной части

5.1.1 Тестирование post-методов

Для проведения тестирования методов добавления данных в БД, будут созданы новые записи для каждой из сущностей в проекте. Для тестирования api-методов с использованием инструмента *swagger* была описана структура *json* запросов для каждого метода.

Для начала создадим пользователя в БД. При создании необходимо заполнить поля: имя пользователя, электронная почта и пароль. После отправки данных на сервер, будет проведена валидация данных, если данные не валидны в

ответе сервера будет показана ошибка. Пароль шифруется и хранится в БД в хешированном виде. Также при создании пользователя, на сервере автоматически фиксируется дата создания.

Проверка начального состояния сущности *fc_user* в базе данных представлена на рисунке 16.

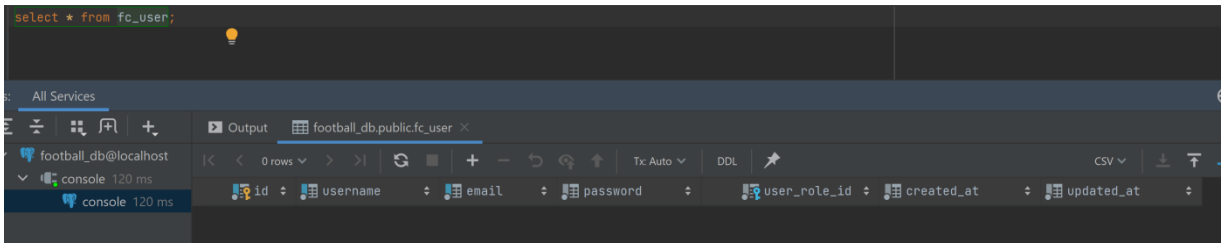


Рисунок 16 – База данных до добавления пользователя

Ввод данных для создания приведен на рисунке 17. Ответ сервера на рисунке 18.

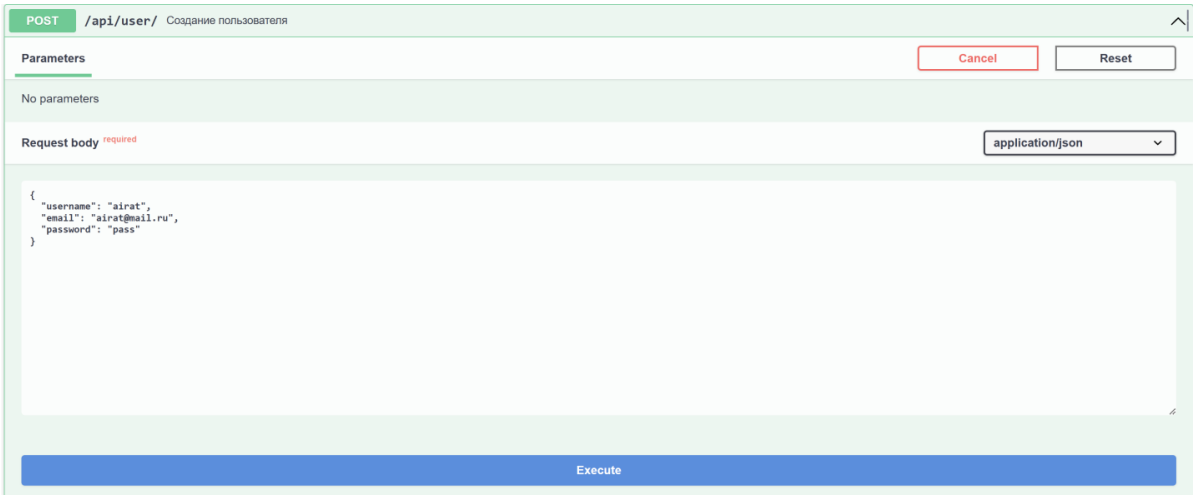


Рисунок 17 – Создание пользователя

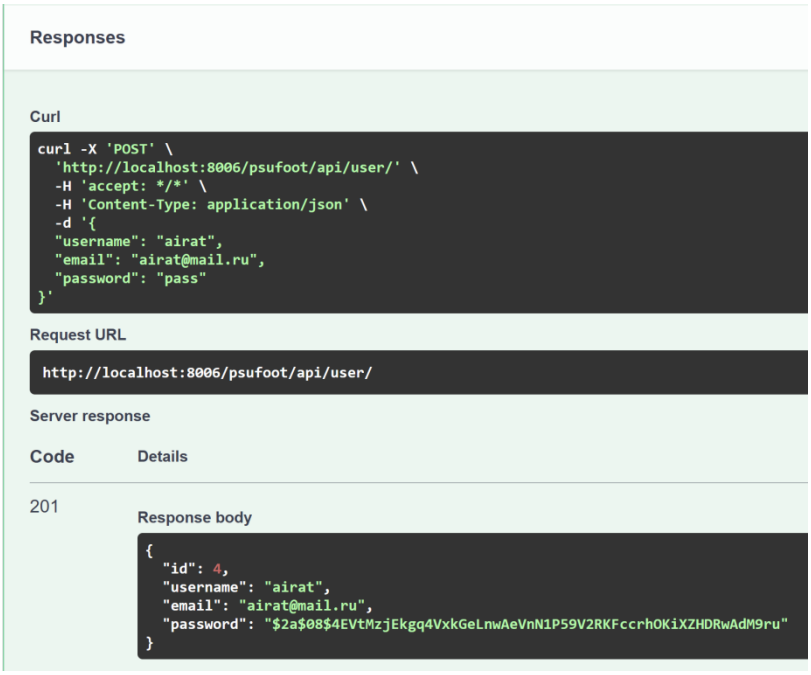


Рисунок 18 – Ответ от сервера

В ответе от сервера отображаются поля согласно модели данных *UserView*, которая представлена на рисунке 19.

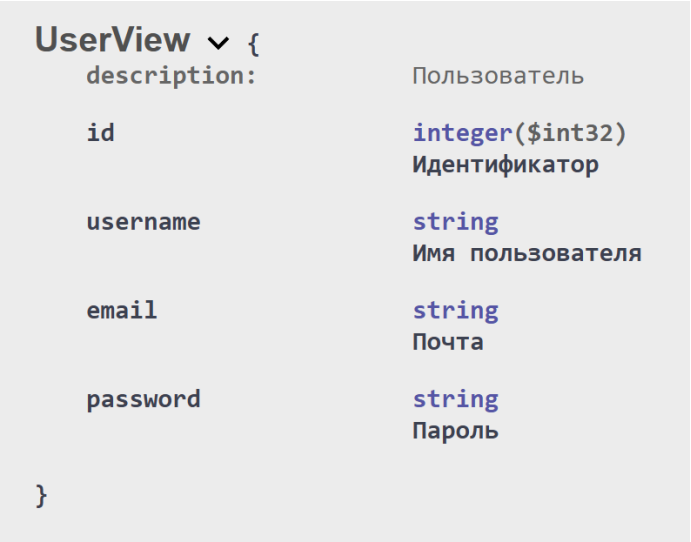


Рисунок 19 – Модель данных UserView

Проверка состояния сущности *fc_user* в базе данных после создания представлена на рисунке 20.

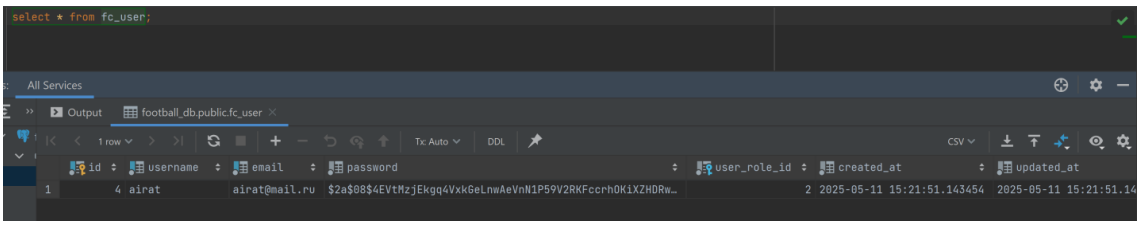


Рисунок 20 – База данных после добавления пользователя

После создания видно, что данные введенные в запросе были успешно сохранены в БД.

Если попытаться создать пользователя с электронной почтой, которая уже существует в БД, пользователь получит сообщение об ошибке.

Попытка создания пользователя с электронной почтой, которая уже есть в БД и вывод соответствующего сообщения представлен на рисунке 21 и 22.

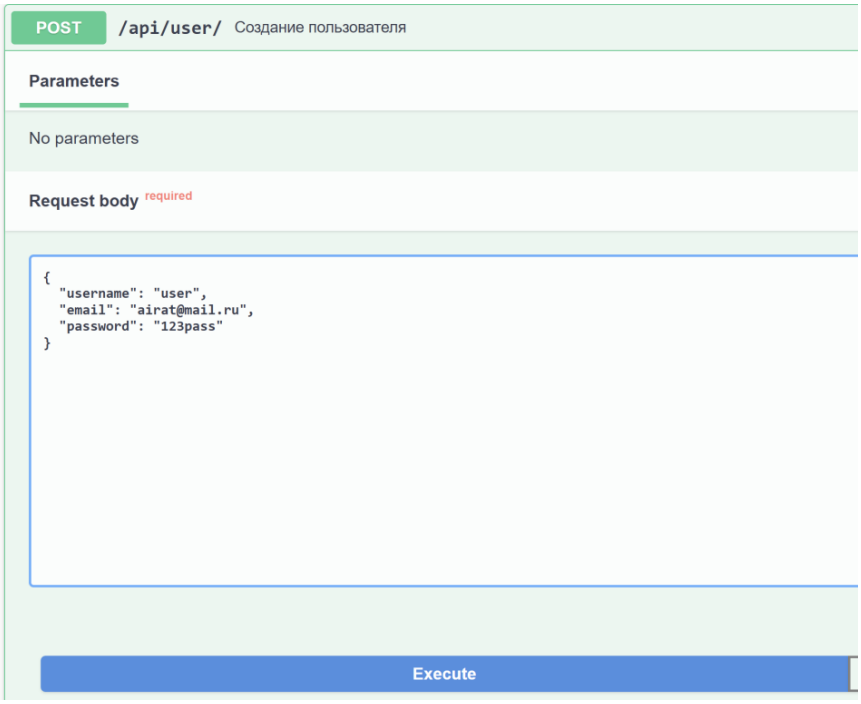


Рисунок 21 – Создание пользователя с существующей почтой в БД

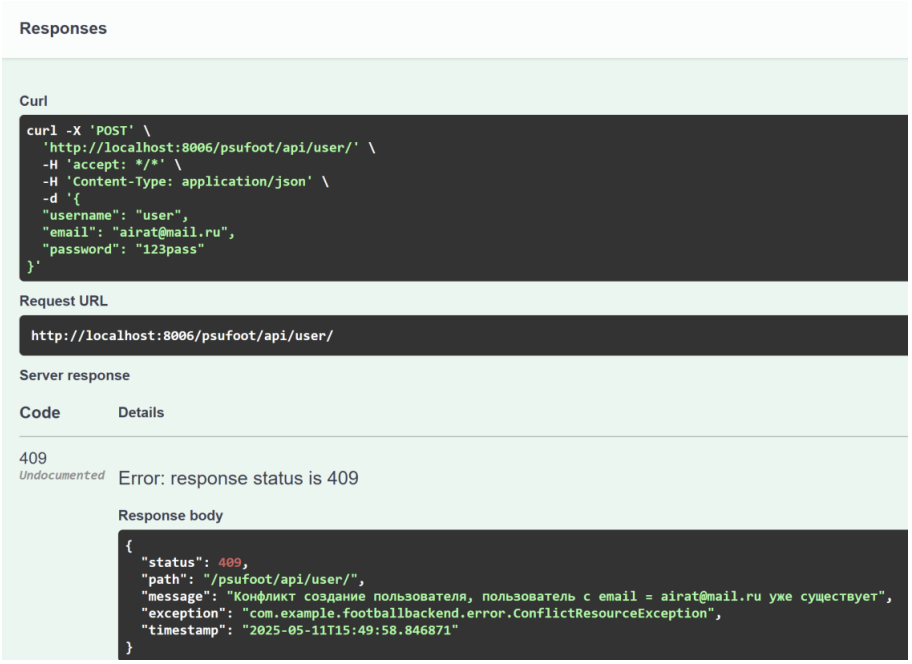


Рисунок 22 – Вывод сообщения

Далее создадим новую запись для сущности *fc_team*. Для добавления новой записи необходимо заполнить следующие поля: название команды, сокращенное имя команды, стадион, относится ли команда к клубу, ссылку на логотип команды. После отправки данные проверяются на валидность, после чего происходит добавление в БД. Если произошла ошибка, то будет выведено соответствующие сообщение.

Проверка начального состояния сущности *fc_team* в базе данных представлена на рисунке 23.

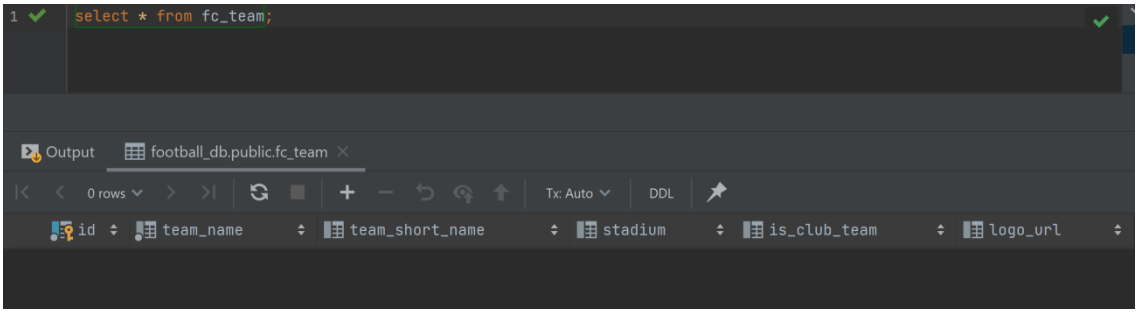


Рисунок 23 – База данных до добавления команды

Ввод данных для создания приведен на рисунке 24. Ответ сервера на рисунке 25.

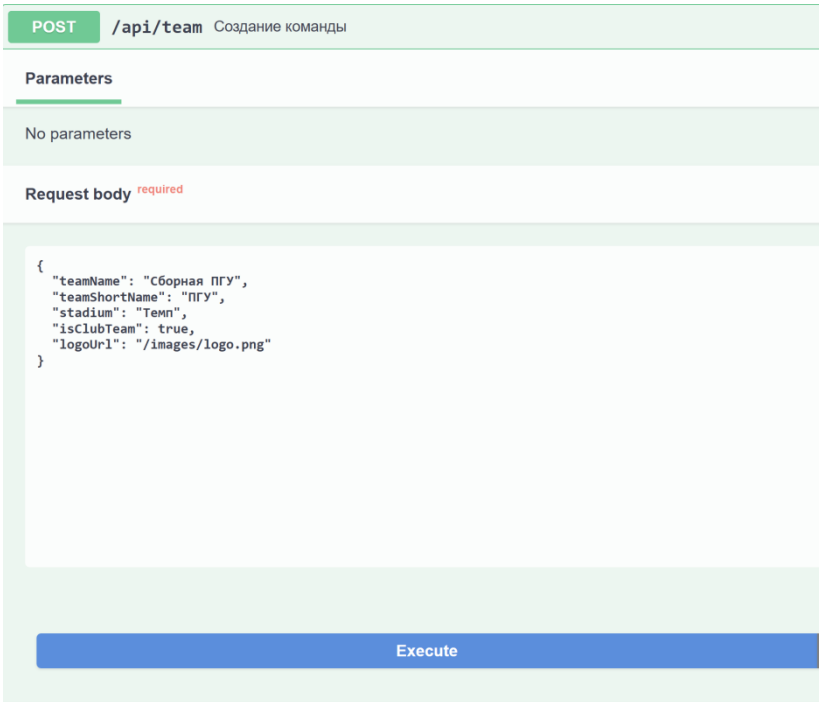


Рисунок 24 – Создание команды

Рисунок 25 – Ответ от сервера

В ответе от сервера отображаются поля согласно модели данных *TeamView*, которая представлена на рисунке 26.

```
TeamView ⌄ {
    description:            Команда

    id                      integer($int32)
                            Идентификатор

    teamName                string
                            Название

    teamShortName           string
                            Короткое название

    stadium                 string
                            Стадион

    isClubTeam              boolean
                            Это команда клуба

    logoUrl                 string
                            Логотип

}
```

Рисунок 26 – Модель данных TeamView

Проверка состояния сущности *fc_team* в базе данных после создания представлена на рисунке 27.

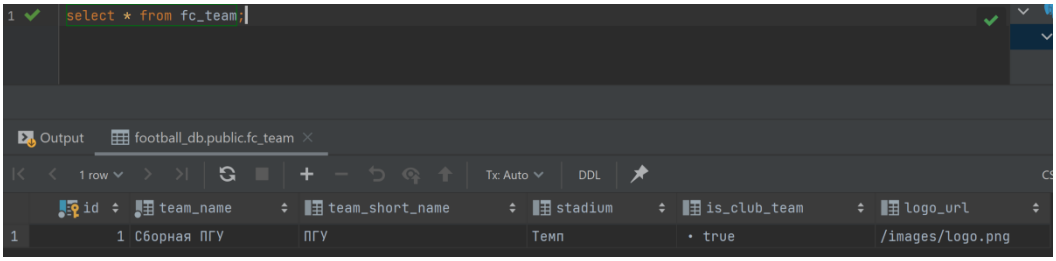


Рисунок 27 – База данных после добавления команды

После создания, мы видим, что данные введенные в запросе были успешно сохранены в БД.

При попытке создать команду с невалидными данными, например, если обязательные поля не заполнены, пользователь получит сообщение об ошибке. Попытка создания команды с невалидными данными и вывод соответствующего сообщения представлен на рисунке 28 и 29.

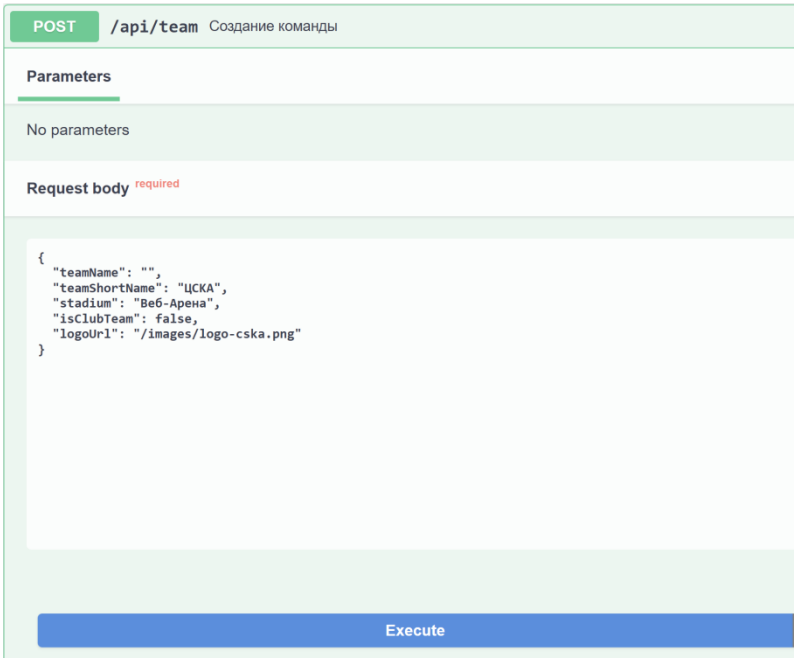


Рисунок 28 – Создание команды без обязательного поля

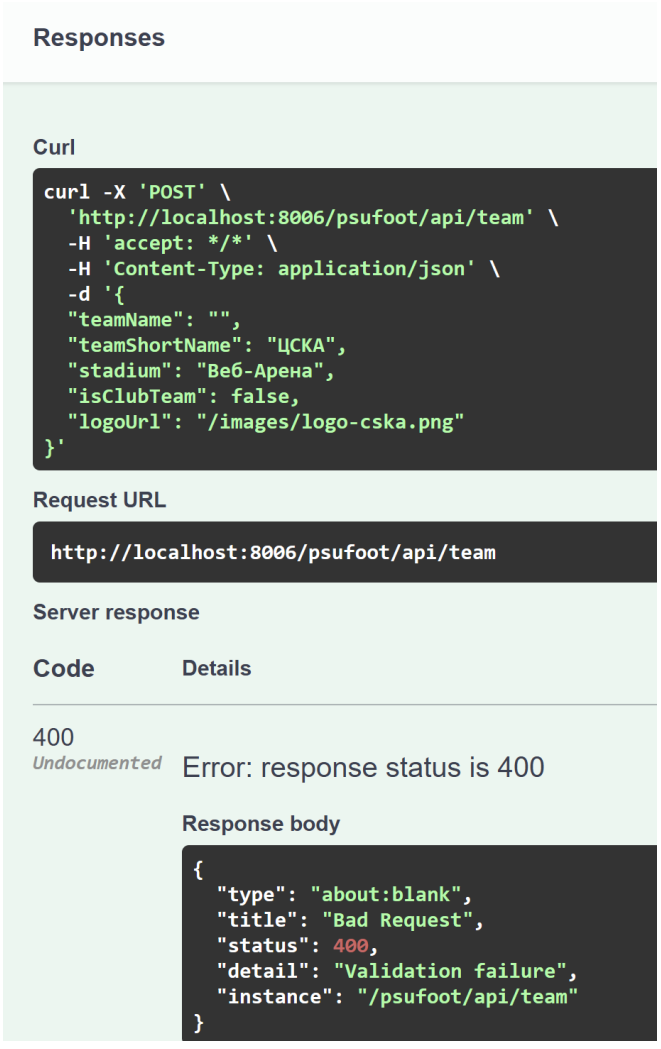


Рисунок 29 – Вывод сообщения

Затем создадим игрока футбольной команды. Для этого необходимо заполнить поля: фамилия, имя, отчество, дату рождения, гражданство, номер игрока, позиция, ссылка на фото. Добавление игрока в команду происходит в put-методе, при создании значение команды будет присвоено null.

Проверка начального состояния сущности *fc_player* в базе данных представлена на рисунке 30.

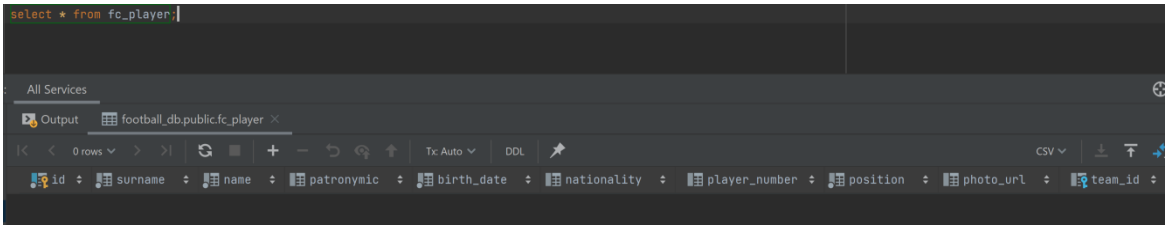


Рисунок 30 – База данных перед добавлением игрока

Ввод данных для создания приведен на рисунке 31. Ответ сервера представлен на рисунке 32.

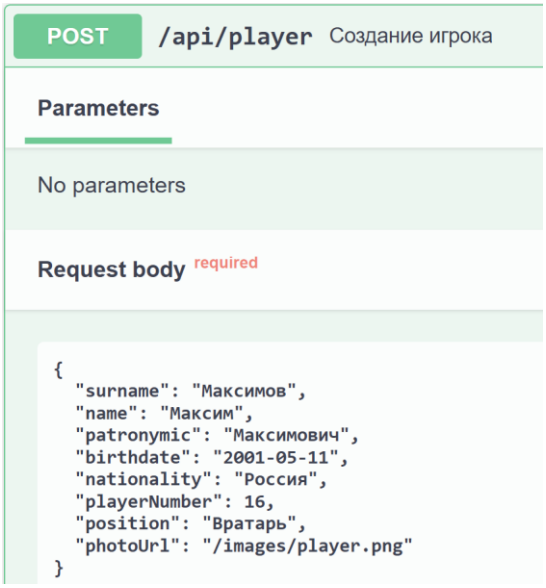


Рисунок 31 – Создание игрока

Рисунок 32 – Ответ от сервера

В ответе от сервера отображаются поля согласно модели данных PlayerView, которая представлена на рисунке 33.

PlayerView ⌄ {

description:	Игрок
id	integer($int32) Идентификатор
surname	string Фамилия
name	string Имя
patronymic	string Отчество
birthdate	string($date) Дата рождения
nationality	string Национальность
position	string Позиция
playerNumber	integer($int32) Номер игрока
teamName	string Команда
photoUrl	string Фото игрока

}

Рисунок 33 – Модель данных PlayerView

Проверка состояния сущности *fc_player* в базе данных после создания представлена на рисунке 34.

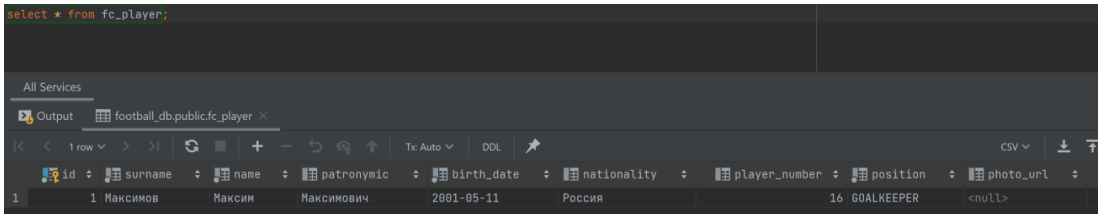


Рисунок 34 – База данных после добавления команды

После создания, мы видим, что данные введенные в запросе были успешно сохранены в БД.

При попытке создать команду с невалидными данными, пользователь получит сообщение об ошибке, как было показано выше.

Чтобы корректно использовать приложение нужно создать турнир, в котором будет участвовать команда. Для этого необходимо заполнить поля: название турнира и сезон проведения турнира. Это необходимо, чтобы в дальнейшем добавлять матчи, турнирную таблицу, статистику игрока.

Проверка начального состояния сущности *fc_tournament* в базе данных представлена на рисунке 35.

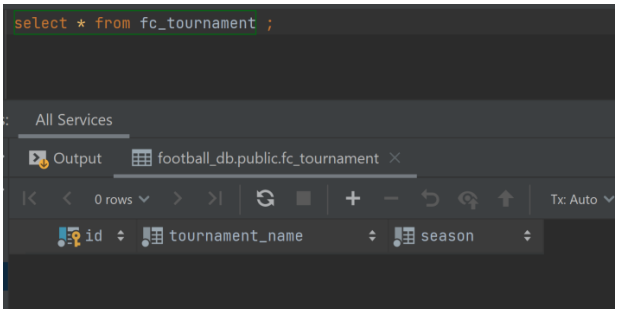


Рисунок 35 – База данных до добавления турнира

Ввод данных для создания приведен на рисунке 36. Ответ сервера представлен на рисунке 37.

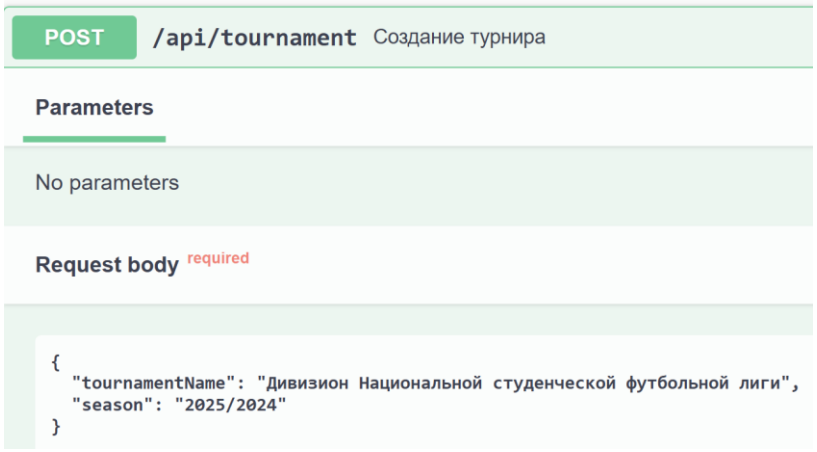


Рисунок 36 – Создание турнира

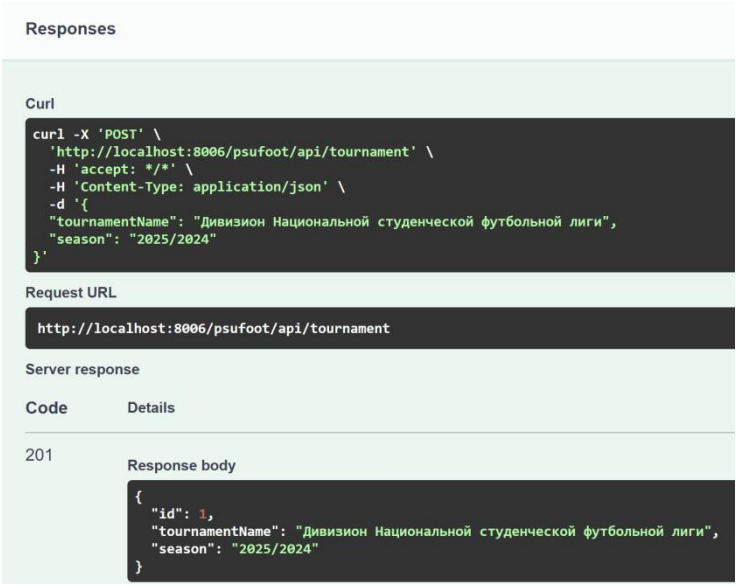


Рисунок 37 – Ответ от сервера

В ответе от сервера отображаются поля согласно модели данных *TournamentView*, которая представлена на рисунке 38.

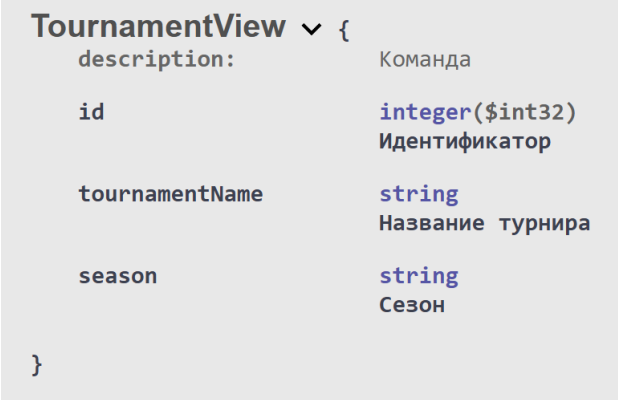


Рисунок 38 – Модель данных TournamentView

Проверка состояния сущности *fc_tournament* в базе данных после создания представлена на рисунке 39.

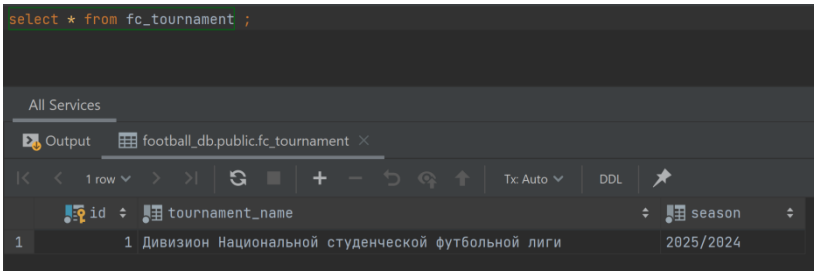


Рисунок 39 – База данных после добавления турнира

После создания, мы видим, что данные введенные в запросе были успешно сохранены в БД.

При попытке ввода невалидных данных, пользователь получит сообщение об ошибке, как было показано выше.

В результате тестирования были успешно созданы основные данные для отображения информации о футбольном клубе клиенту. Также было проверено поведение системы при попытке ввода некорректных данных.

5.1.2 Тестирование put-методов

Для возможности редактирование информации в БД, нужно использовать put-методы контролеров. В ходе тестирования проверим некоторые методы изменения данных в сущностях.

Проверим редактирование данных пользователя. В предыдущем разделе был создан пользователь с именем пользователя – airat, для изменения имени необходимо ввести id пользователя и новое имя пользователя.

На рисунке 40 представлен сценарий редактирования данных пользователя, результат редактирования показан на рисунке 41.

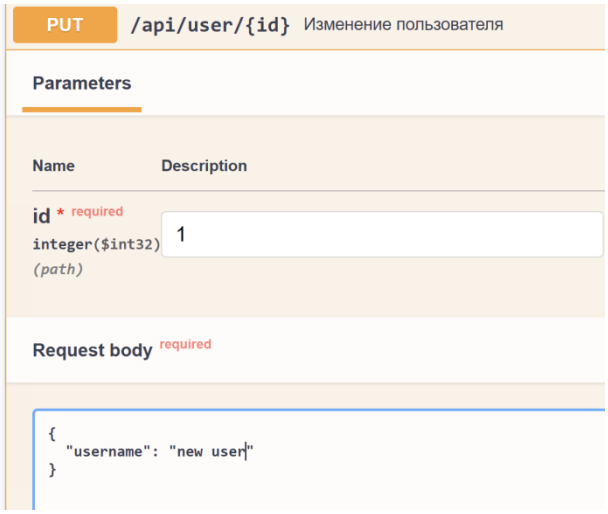


Рисунок 40 – Редактирование пользователя

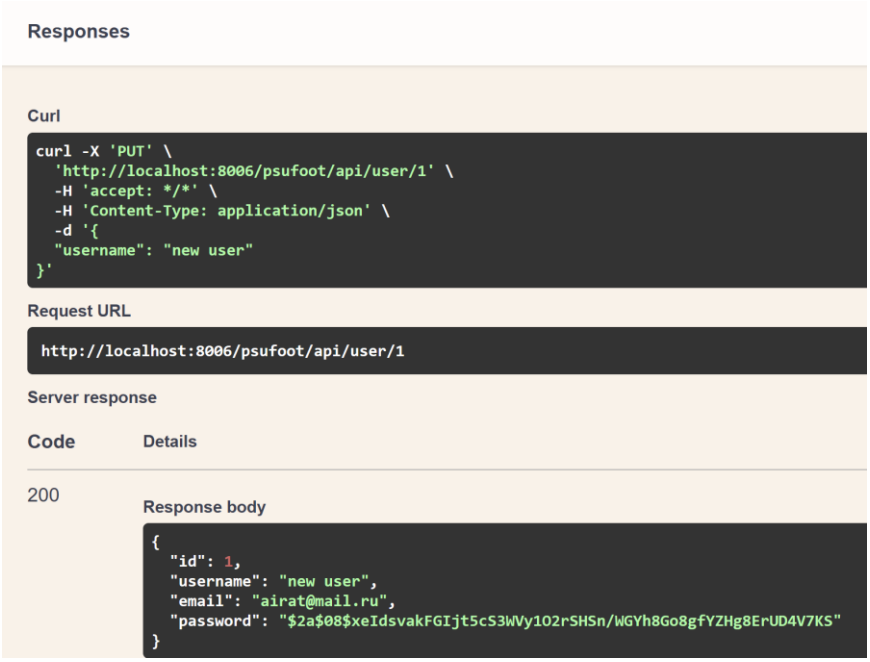


Рисунок 41 – Ответ от сервера

Ниже на рисунке 42 представлен результат обновления данных в БД для сущности *fc_user*. Также видно, что автоматически было обновлено дата и время редактирования данных.

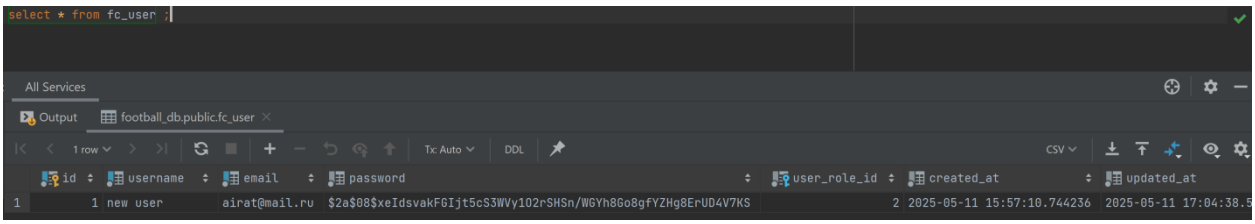


Рисунок 42 – База данных после обновления пользователя

Далее проверим изменения данных для ранее созданной команды. Структура запроса на изменение такая же, как и при создании, такое решение было реализовано для дальнейшей работы с браузером. Когда клиент будет отправлять запрос на изменение с UI-интрефейса, старые данные будут введены автоматически, и пользователь сможет изменить нужные поля, без необходимости вводить их заново. Для изменения нужной записи в БД необходимо ввести id команды.

В этом сценарии будут изменены название команды и сокращенное название, остальные поля останутся без изменения. Процесс редактирование представлен на рисунках 43 и 44.

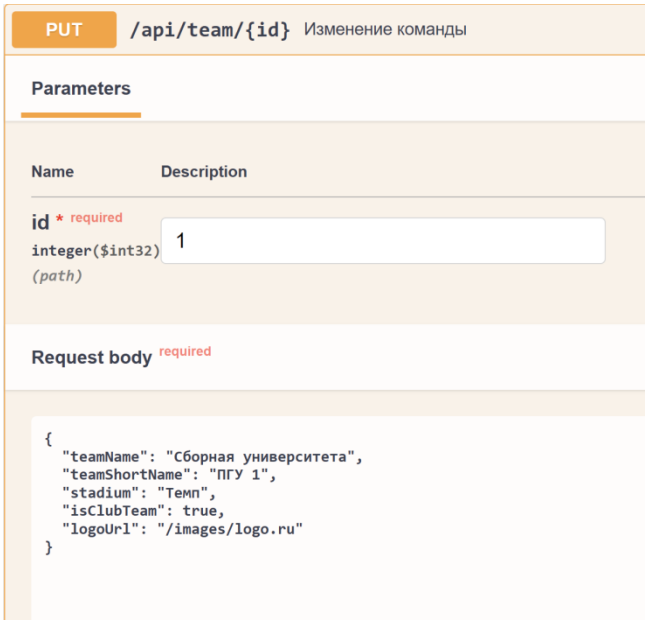


Рисунок 43 – Редактирование пользователя

Рисунок 44 – Ответ от сервера

Ниже на рисунке 45 представлен результат обновления данных в БД для сущности *fc_team*. На рисунке видно, что были изменены только поля с названием и сокращенным названием команды.

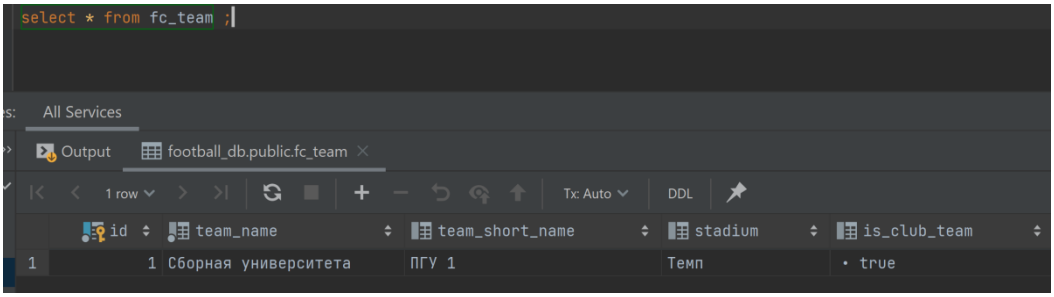


Рисунок 45 – База данных после обновления команды

Выше было описано, что при создании игрока, он не добавляется сразу в футбольную команду, вместо этого присваивается значение null. Проведем тестирование на добавление созданного игрока в команду с помощью метода put. Для этого необходимо ввести id игрока и id команды, куда будет добавлен игрок.

Процесс добавления игрока в команду представлен на рисунках 46 и 47.

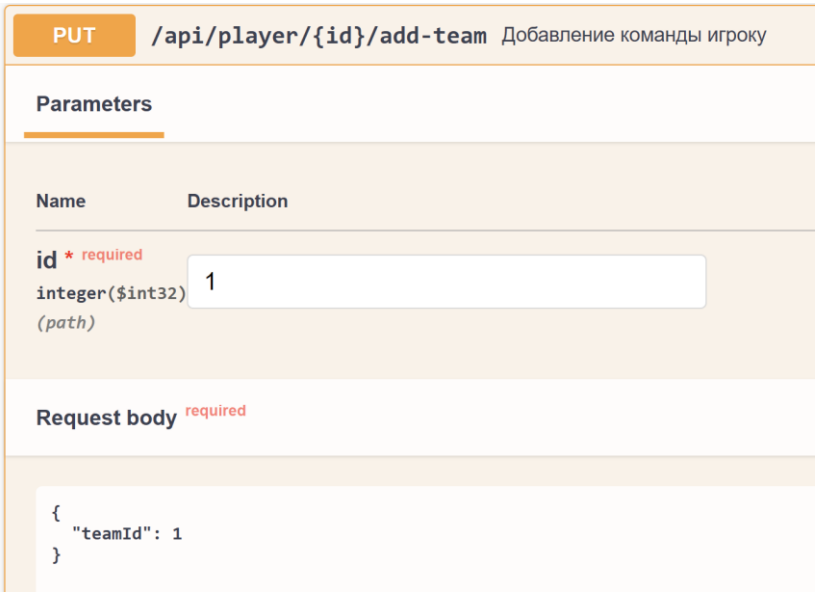


Рисунок 46 – Добавление игрока в команду

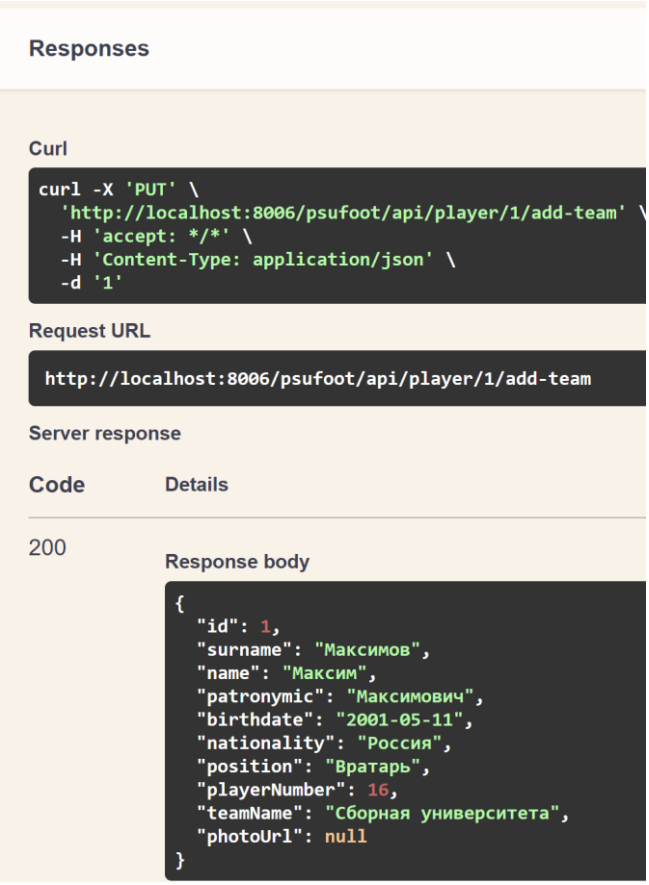


Рисунок 47 – Ответ от сервера

Ниже на рисунке 48 представлен результат добавление игрока. На рисунке видно, что в колонке *team_id*, вместо значение null, записана ссылка на команду.

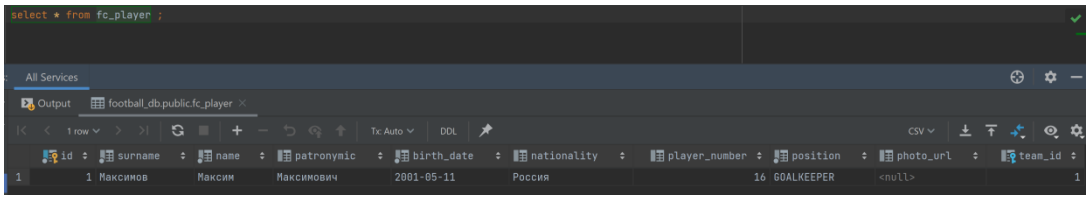


Рисунок 48 – База данных после добавления игрока в команду

В ходе тестирования были проверены основные сценарии, которые прошли успешно. Поведение остальных контролеров при изменении данных аналогично, приведенным выше примерам. При вводе невалидных значений также, как и при создании данных, система отправит ошибку и не даст отредактировать записи в БД.

5.1.3 Тестирование delete-методов

Методы delete предназначены для удаления информации из БД. Для того, чтобы удалить данные необходимо ввести id записи и отправить запрос на сервер. После удаление если сервер пришлет ответ с кодом 204, то все прошло успешно.

Сценарий с удалением данных начнем с удаления пользователя из системы. Вводим его id, как параметр и отправляем запрос на сервер.

На рисунке 49 и 50 показан процесс удаления пользователя.

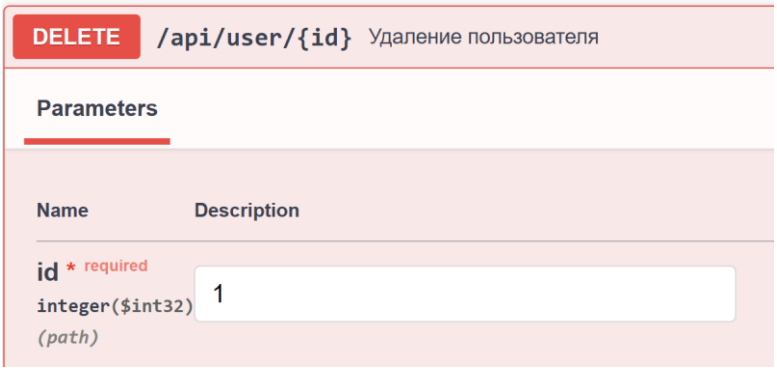


Рисунок 49 – Удаления пользователя из БД

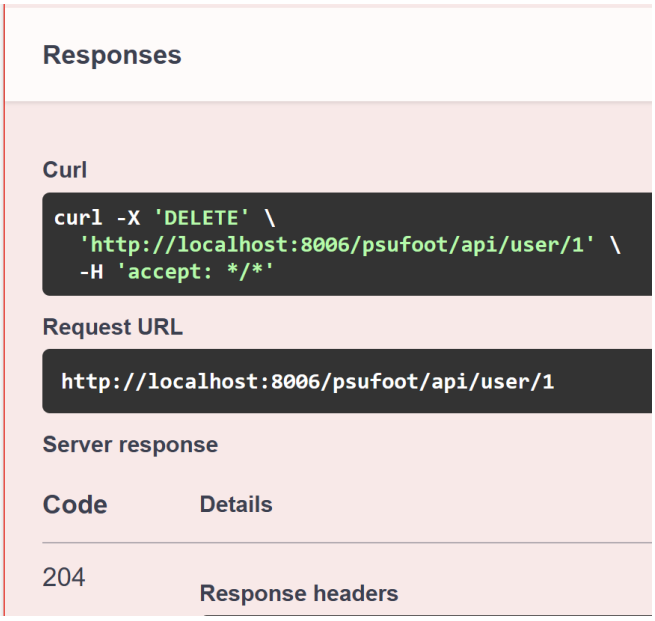


Рисунок 50 – Ответ от сервера

На рисунке 51 представлен результат удаления пользователя из базы данных. При запросе на выборку данных сущности *fc_user*, результатов не найдено.

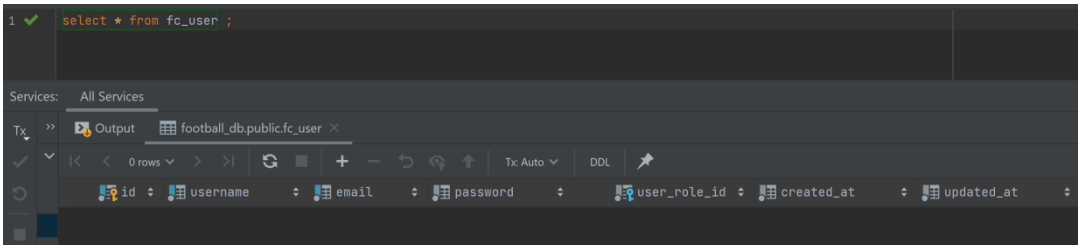


Рисунок 51 – Результат пользователя из базы данных

Для остальных контролеров поведение системы аналогично выше описанному примеру.

6 Тестирование клиентской части

6.1 Регистрация пользователя

Для проведения тестирования регистрации пользователя на клиентской части, нужно перейти на страницу с формой регистрацией, и ввести следующие запрашиваемые данные: имя пользователя, логин и пароль. После введения данных и выполнения запроса на добавление пользователя, в базе данных должна появится запись о новом пользователе.

Проверка начального состояния сущности *fc_user* в базе данных представлена на рисунке 52.

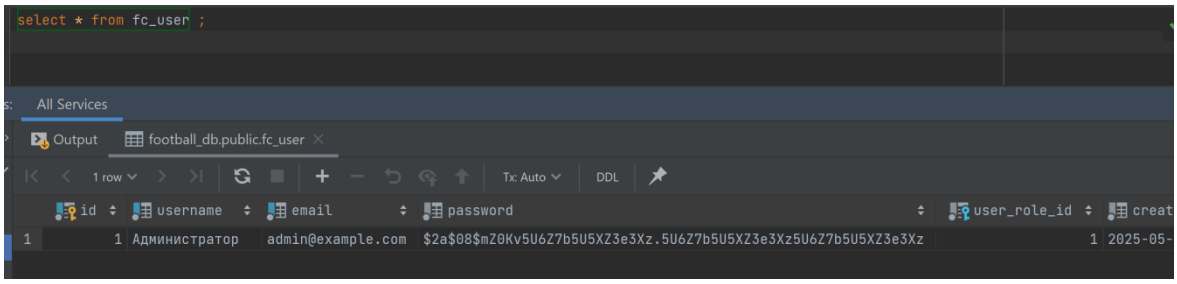


Рисунок 52 – База данных до добавления пользователя

Ввод данных в форму для регистрации и отправка запроса на сервер приведен на рисунке 53 и 54.

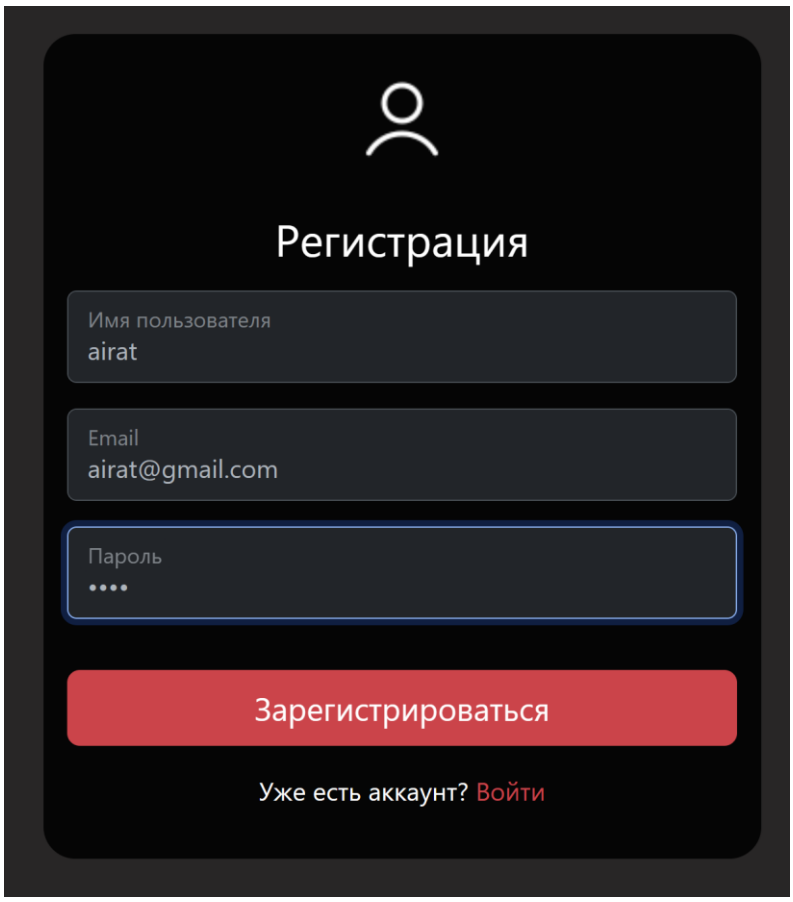


Рисунок 53 – Регистрация пользователя

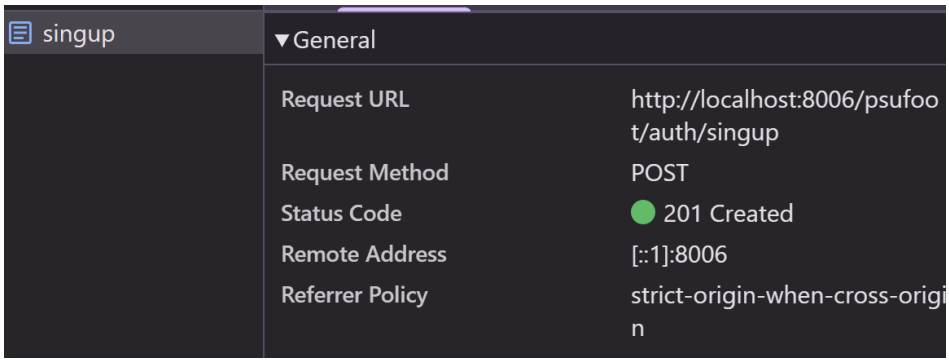


Рисунок 54 – Запрос на регистрацию пользователя

Проверка состояния сущности *fc_user* в базе данных после регистрации представлена на рисунке 55.

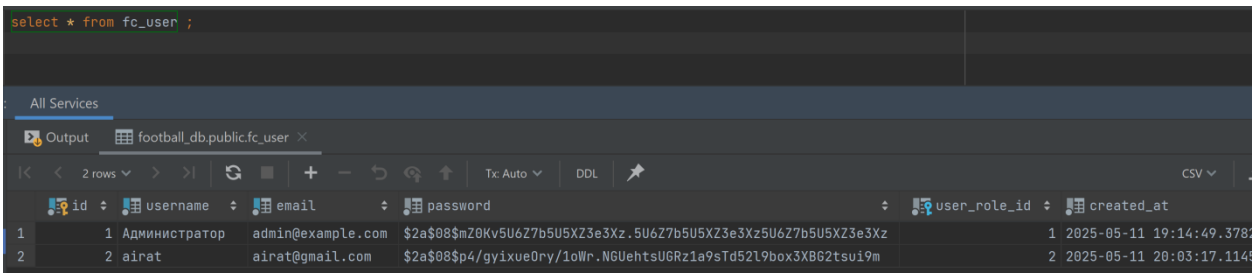


Рисунок 55 – База данных после добавления пользователя

После проведения регистрации, мы видим, что данные введенные на сайте успешно были сохранены в БД.

Если попытаться зарегистрировать пользователя с электронной почтой, которая уже существует в БД, пользователь получит сообщение об этом и будет предложено ввести корректные данные. Попытка регистрации с такими данными и вывод соответствующего сообщения представлен на рисунке 56 и 57. На рисунке 58 представлен запрос с некорректными данными.

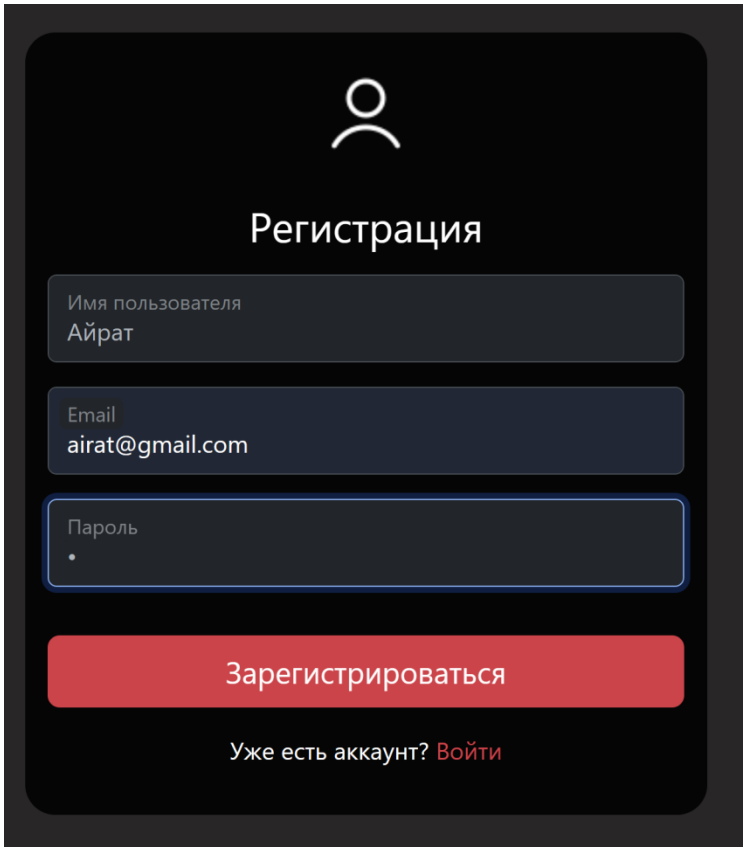


Рисунок 56 – Регистрация пользователя с существующим логином в БД

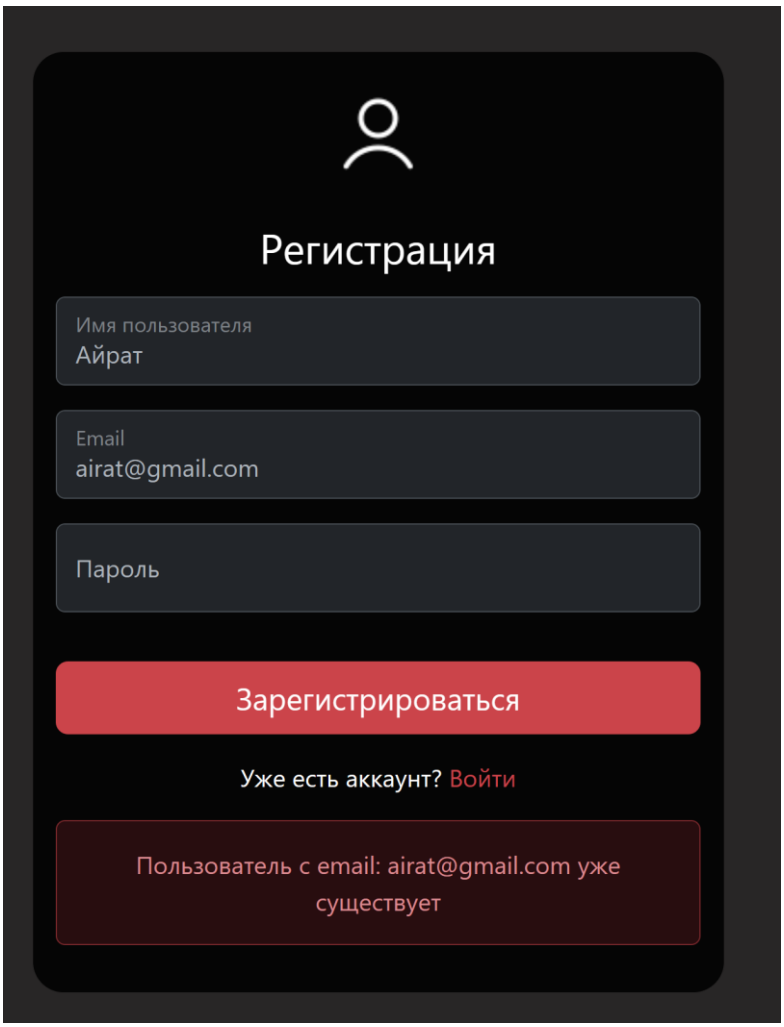


Рисунок 57 – Вывод сообщения

Рисунок 58 – Запрос с невалидными данными

Тестирование прошло успешно, система ожидаемо отреагировала на различные ситуации использования.

6.2 Авторизация пользователя

При авторизации пользователь вводит ранее добавленные данные в БД. Если введенные данные и данные, которые хранятся в базе данных совпадают, авторизация проходит успешно. В противном случае пользователю будет показано сообщение об ошибке.

На рисунке 59 представлен сценарий тестирования с ошибочными данными, результат входа с неправильными данными показан на рисунке 60.

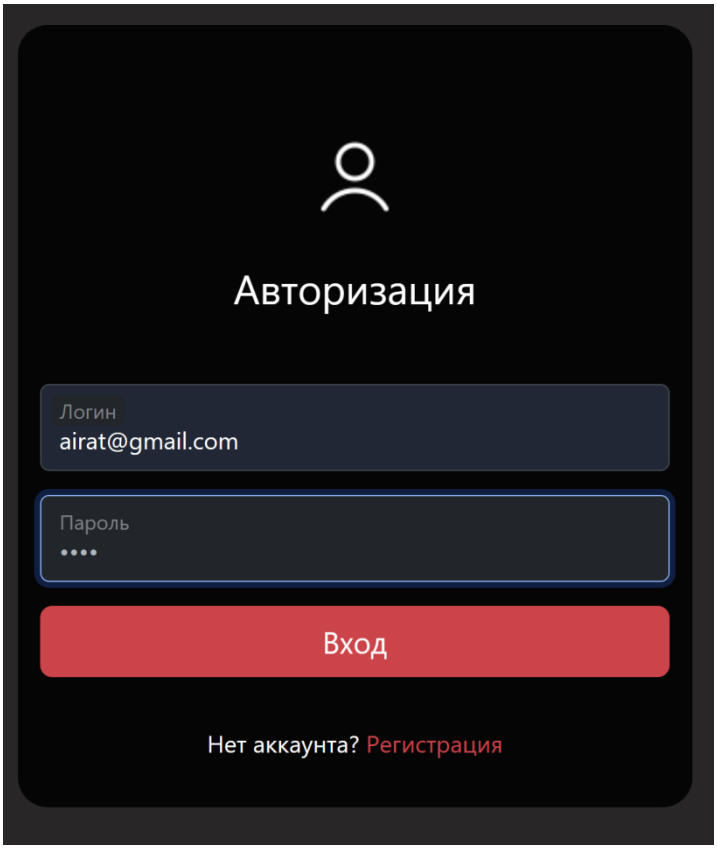


Рисунок 59 – Ввод неправильных данных

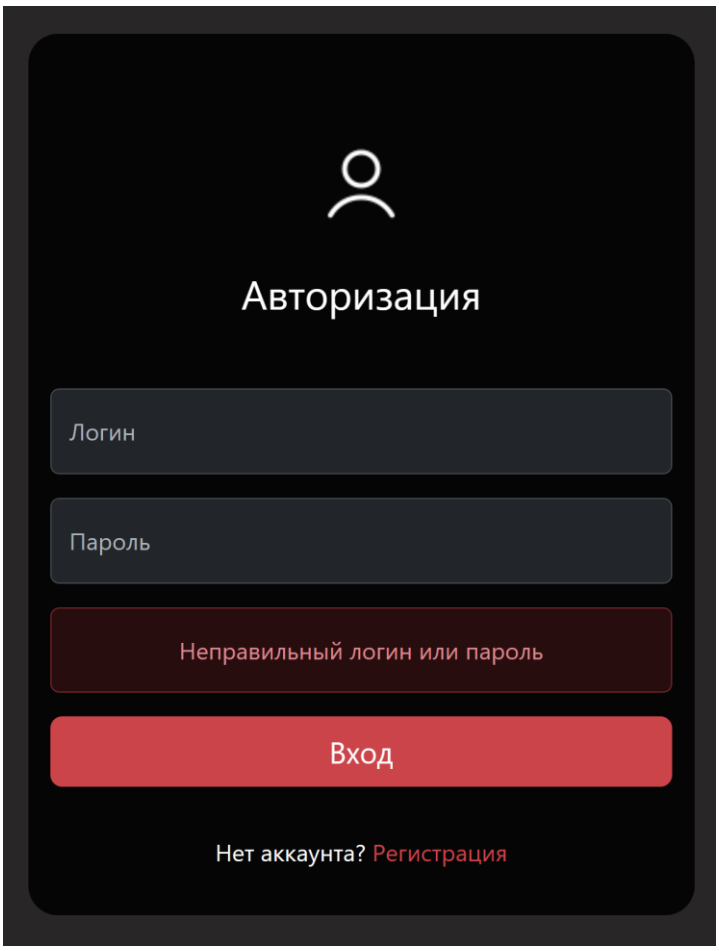


Рисунок 60 – Сообщение об ошибке

При отправке запроса с неправильными данными, система устанавливает параметр error = true, после чего страница рендерится снова с отображением ошибки. Запрос представлен на рисунке 61.

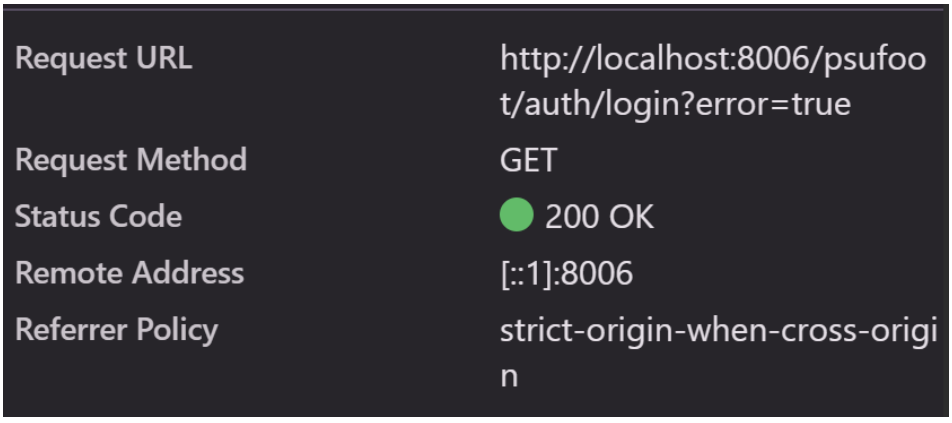


Рисунок 61 – Запрос с параметром ошибки

Далее введем достоверные данные. На рисунке 62 и 63 представлен сценарий тестирования с достоверными данными, результат входа с верными данными показан на рисунке 64.

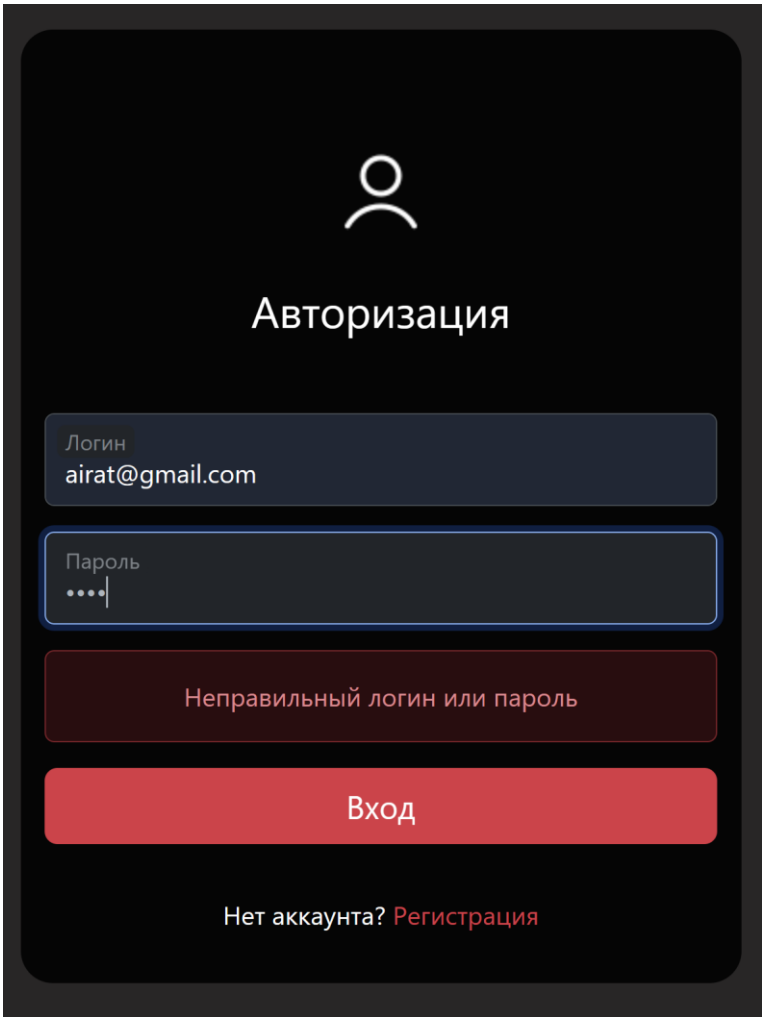


Рисунок 62 – Ввод верных данных

Рисунок 63 – Запрос на перенаправление после успешной авторизации

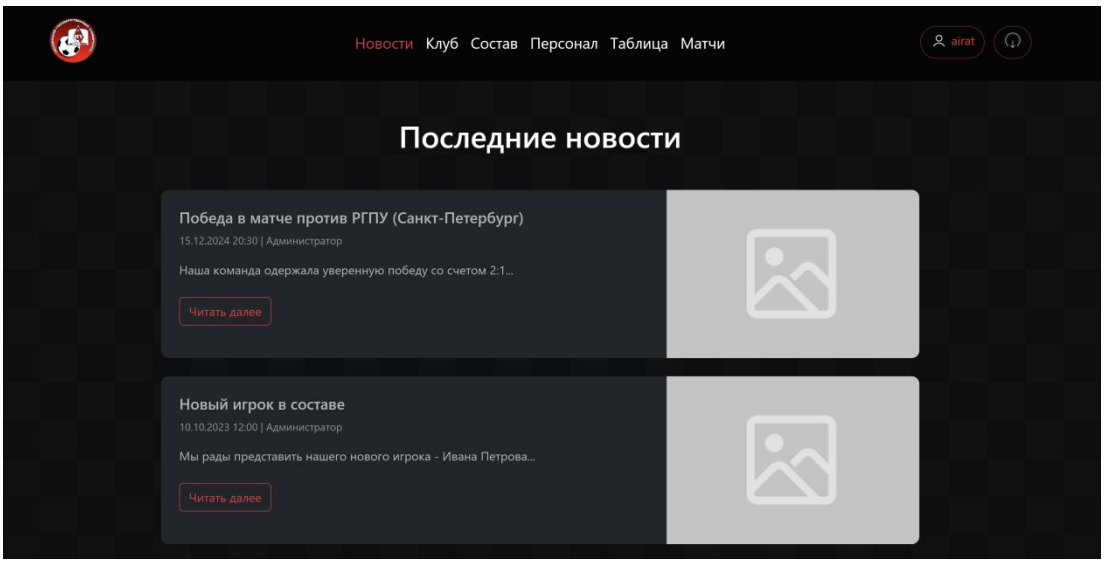


Рисунок 64 – Результат авторизации

В ходе тестирования все сценарии прошли успешно. При неправильных данных пользователю была выведена ошибка. С верно введенными данными произошел переход на главную страницу где отображено имя пользователя, который выполнил вход, а также последние новости команды.

6.3 Тестирование функционала просмотра информации

Чтобы провести это тестирование, предварительно была добавлена информация по всем основным ресурсам. Пользователю доступен функционал просмотра всей информации, находящийся на сайте.

Тестовый сценарий состоит из перехода на станицы «Клуб», «Состав», «Персонал», «Таблица», «Матчи» где отображены данные, хранящиеся в БД.

На рисунках 65 и 66 изображена страница «Клуб». Здесь можно ознакомится со всей основной информацией о футбольном клубе: полное название команды, сокращенное название команды, стадион клуба, главный тренер, первые четыре игрока команды, краткая информация о клубе. Также при нажатии на кнопку «все игроки», пользователь может перейти к полному составу команды.

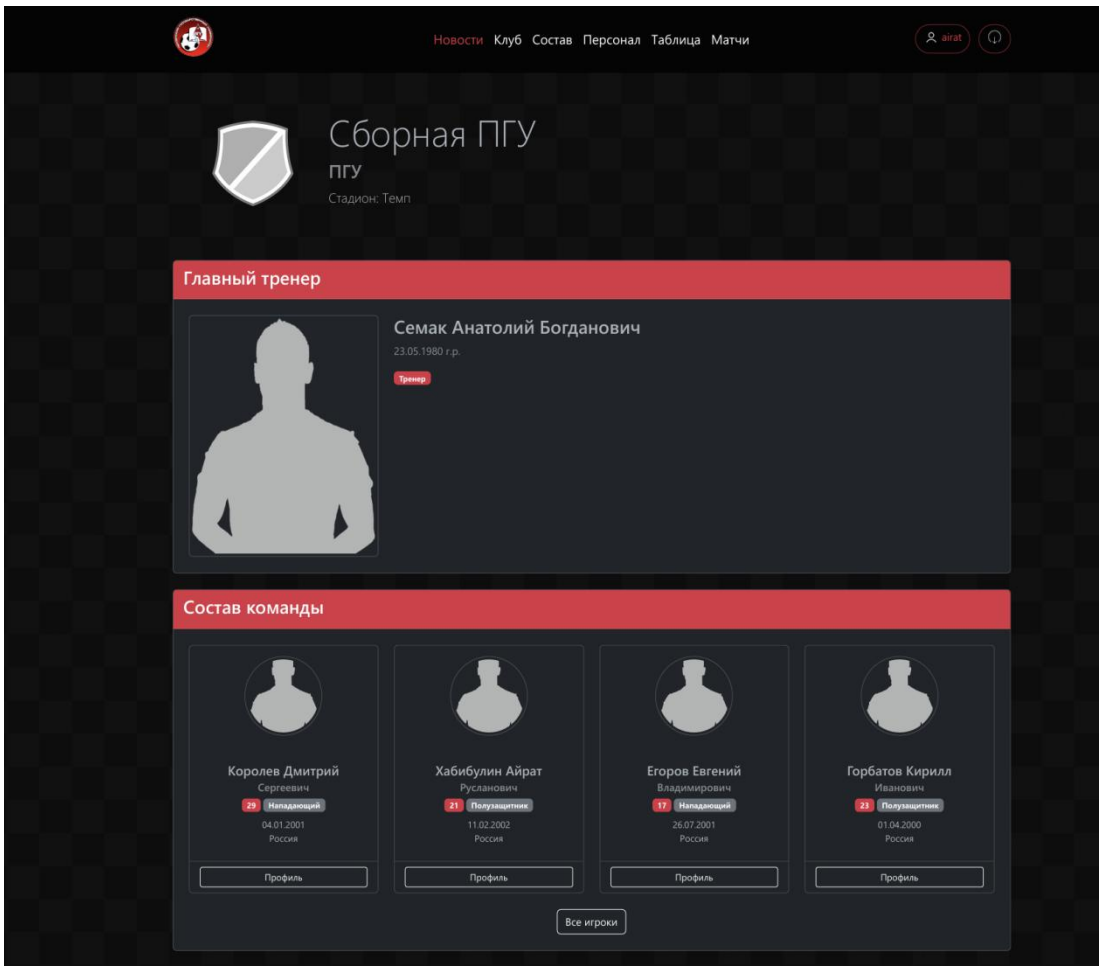


Рисунок 65 – Отображение страницы «Клуб»

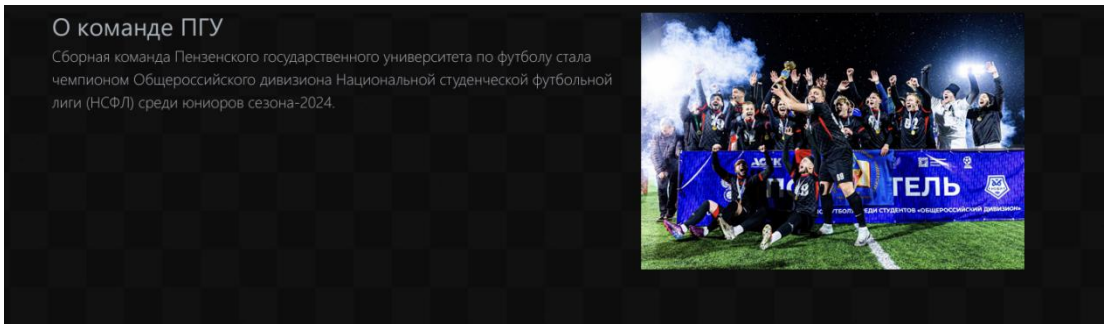


Рисунок 66 – Отображение страницы «Клуб»

На рисунке 67 представлена страница «Состав». На ней пользователь может увидеть название команды, тактическую схему команды, карточки всех игроков команды, с возможностью перейти в личный профиль игрока.

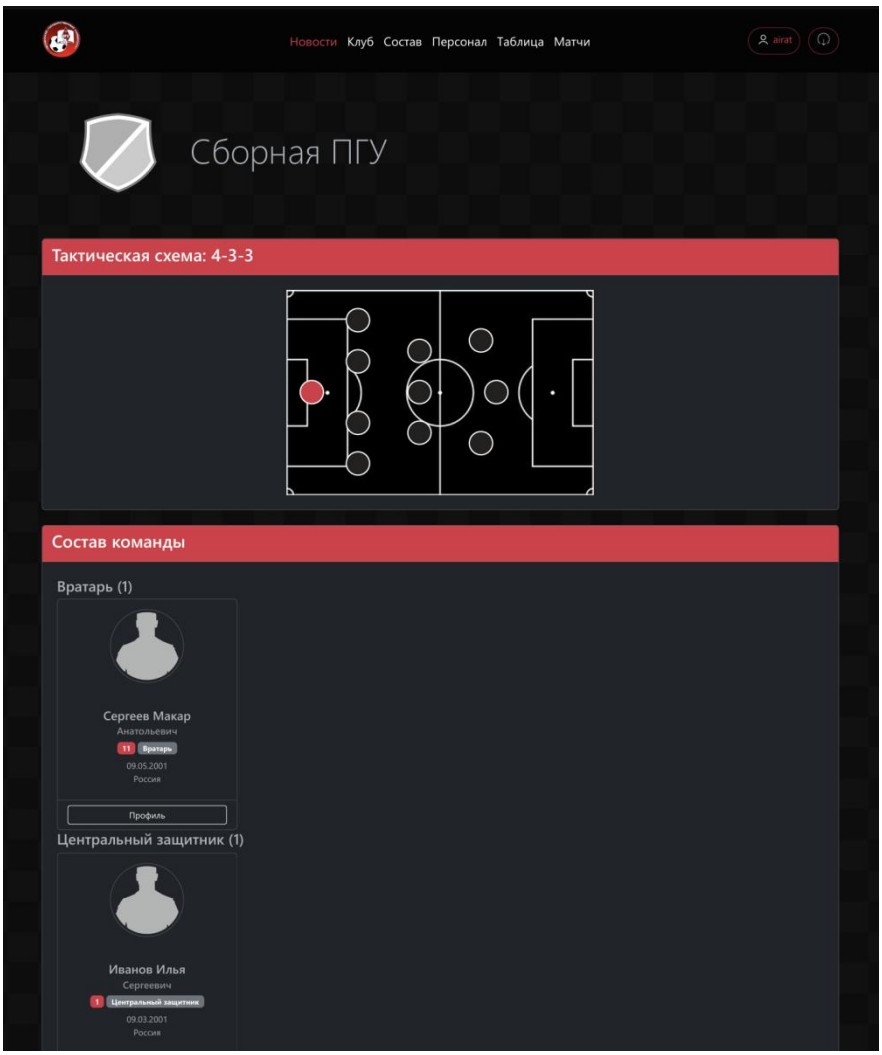


Рисунок 67 – Отображение страницы «Состав»

При переходе на страницу «Персонал» пользователю предоставляется информация о руководстве и тренерском штабе команды, их должности, ФИО, дате рождении. На рисунке 68 показана страница «Персонал».

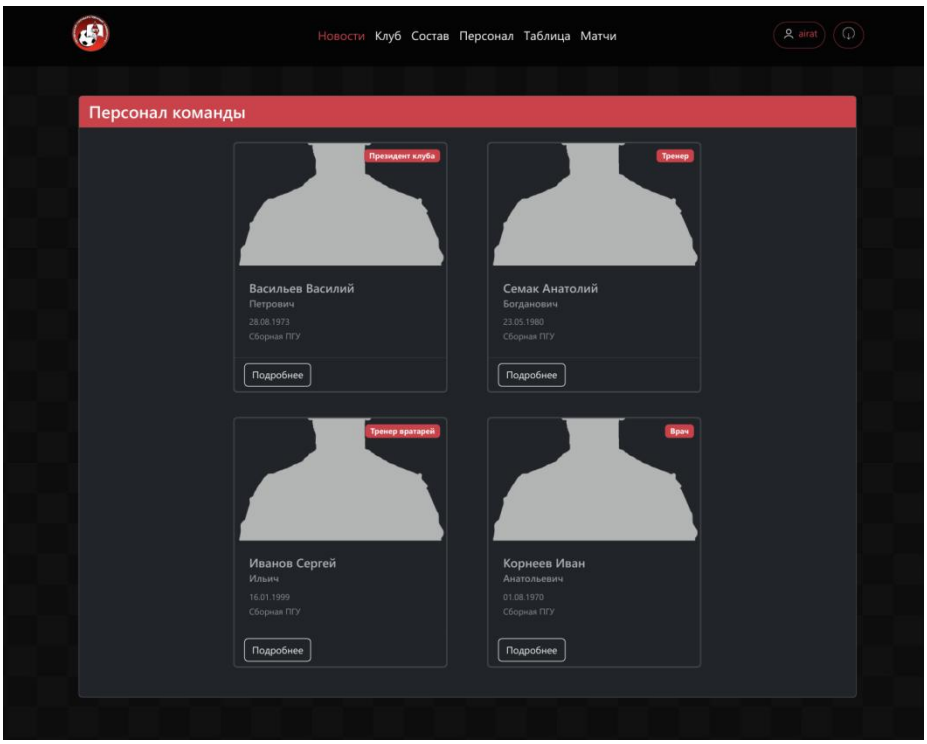


Рисунок 68 – Отображение страницы «Персонал»

При переходе на страницу «Таблица» пользователь имеет возможность выбрать турнир с помощью выпадающего списка, в которой участвует команда, после выбора будет отображена турнирная таблица. Команды в таблице сортируются по количеству очков, набранных в турнире. Если у турнира не заполнена таблица, пользователь увидит уведомление, в котором ему будет предложено выбрать другой турнир. На рисунках 69, 70 и 71 показан функционал страницы.

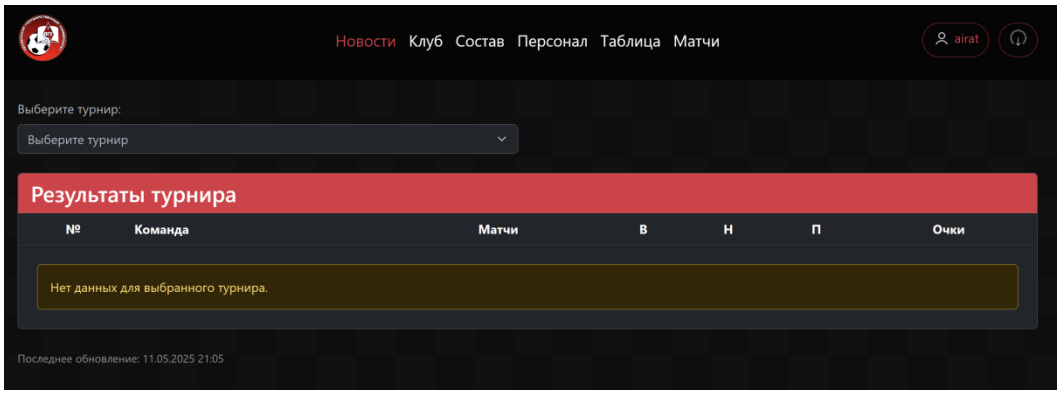


Рисунок 69 – Выбор турнира.

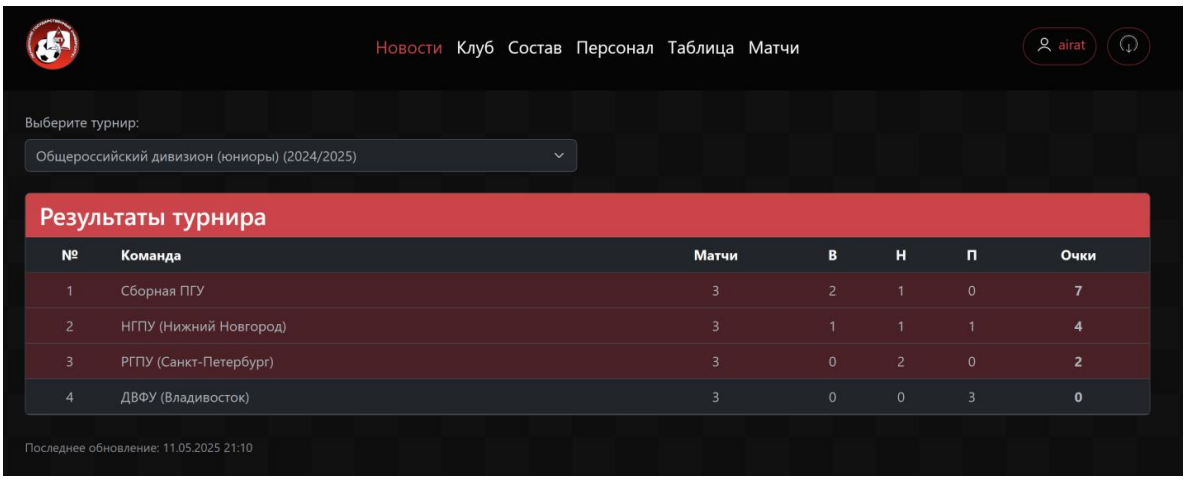


Рисунок 70 – Турнирная таблица.

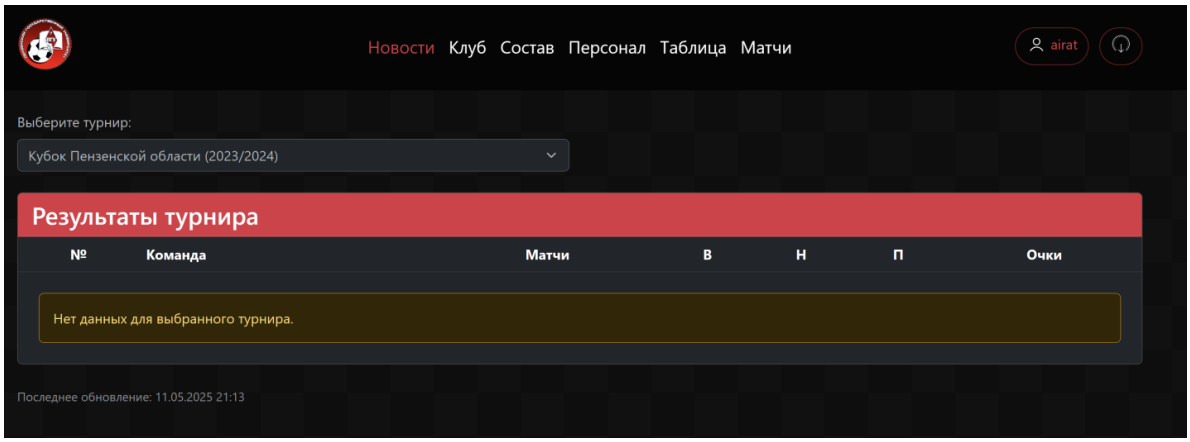


Рисунок 71 – Турнир с незаполненной таблицей.

Для просмотра матчей команды, необходимо перейти на страницу «Матчи». Пользователю также будет предложено выбрать турнир, в котором проходили матчи. Если турнир не выбран, отображается уведомление, без карточек матчей. После выбора будет отображены карточки матчей, где есть информации о будущих матчах, результатах прошедших матчей, место, дата и время проведение. Результат отображения матчей представлен на рисунке 72 и 73.

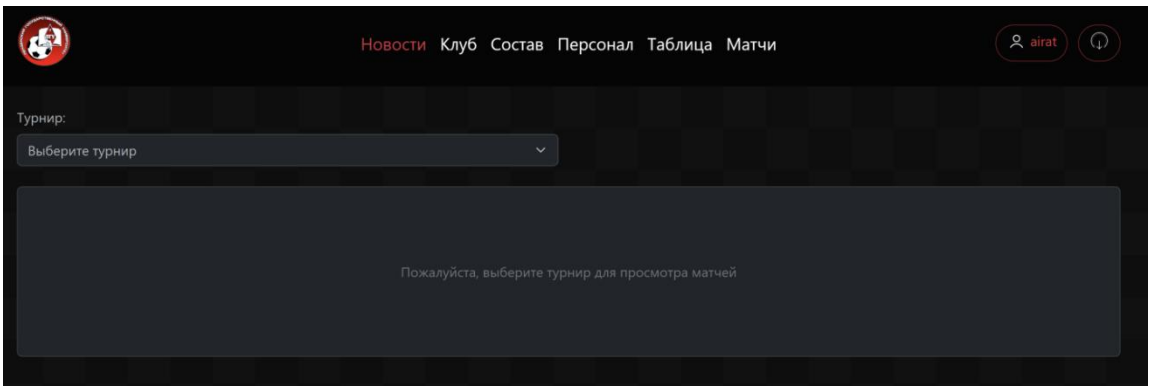


Рисунок 72 – Страница «Матчи» без выбора турнира

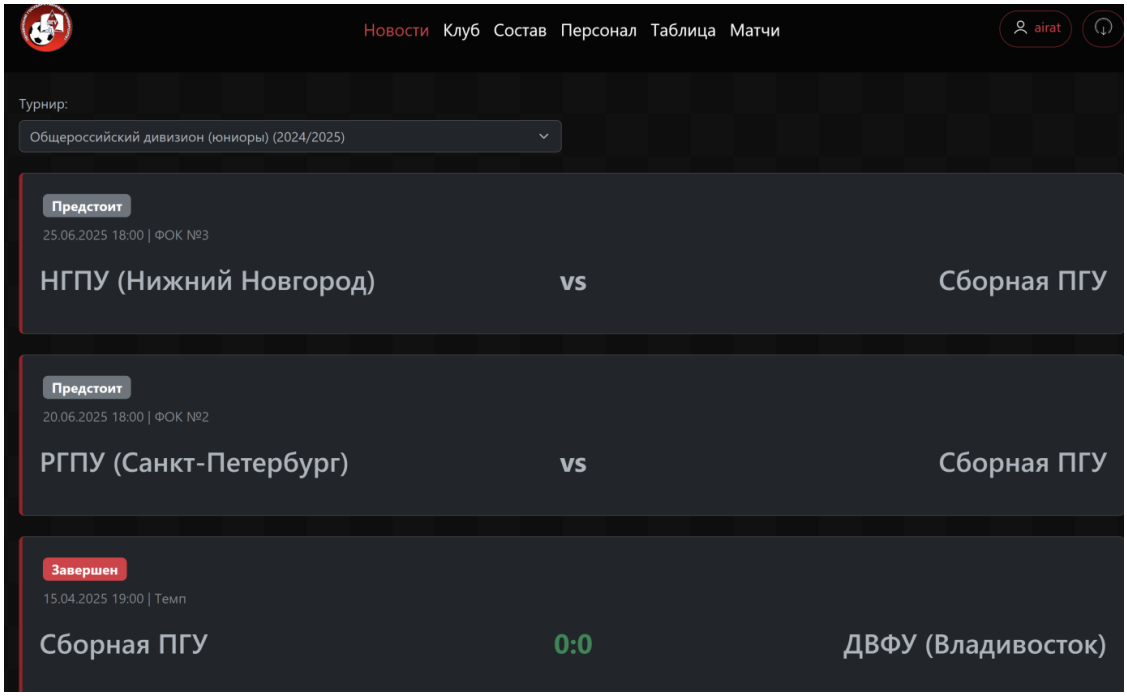


Рисунок 73 – Страница «Матчи» с выбором турнира

После проделанных действий, ожидаемый результат совпал с действительным, тестирование отображения данных прошли успешно.

6.4 Тестирование функционала выход из аккаунта

Для выполнения выхода из аккаунта пользователь должен нажать на кнопку в заголовке страницы. После нажатия, на сервер отправляется запрос, и с помощью *SpringSecurity* очищаются все файлы cookie, данные о сессии, и происходит перенаправление на главную страницу. В заголовке страницы кнопки

профиль и выход, меняются на регистрацию и вход. Процесс тестового сценария показан на рисунках 74, 75 и 76.

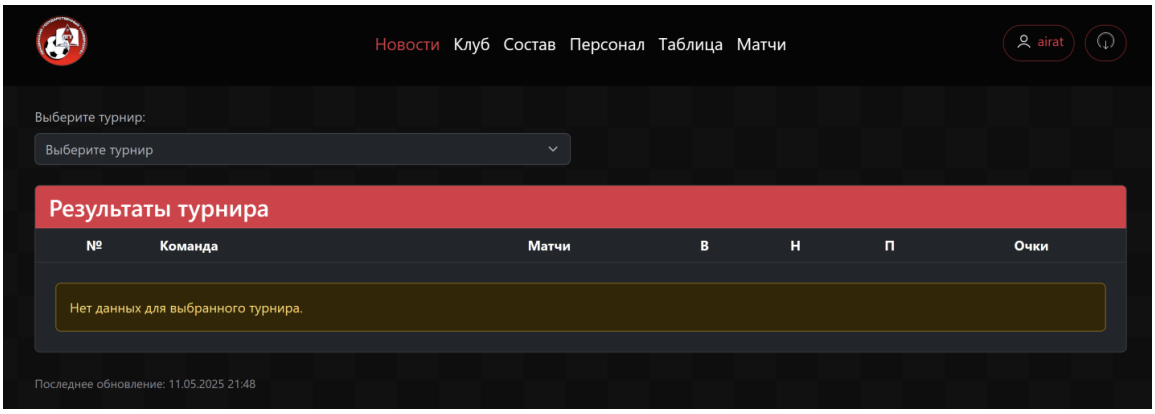


Рисунок 74 – Страница с которой выполняется выход

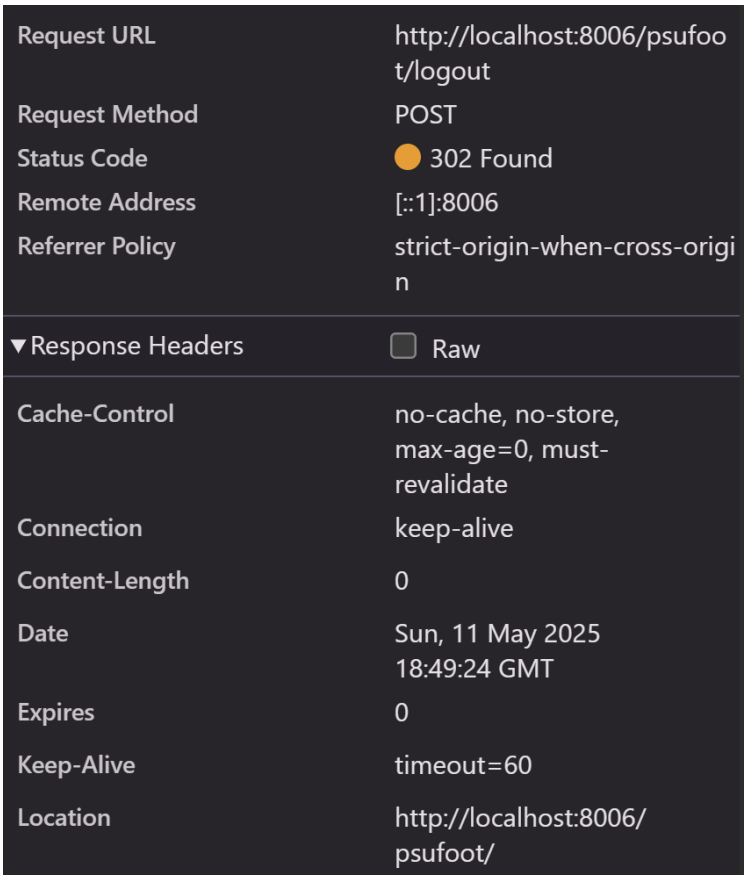


Рисунок 75 – Запрос выхода из аккаунта

Исходя из результата запроса, что при выходе происходит перенаправление на главную страницу сайта.

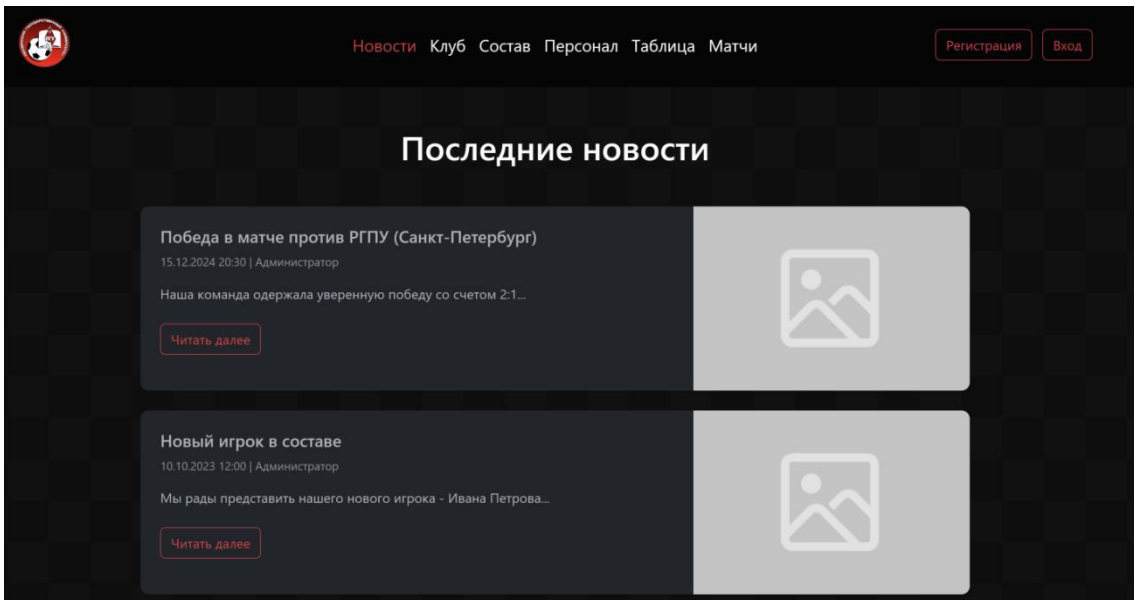


Рисунок 76 – Результат выхода из аккаунта

7 Безопасность жизнедеятельности

Безопасность и сохранность здоровья для людей, проводящих огромное количество за компьютером, требует пристального внимания. Чтобы не подвергать организм сильному напряжению, необходимо знать ряд правил и мер безопасности при работе за компьютером. Несоблюдение этих мер безопасности может привести к различным проблемам со здоровьем.

В данном разделе будут рассмотрены основные аспекты безопасности жизнедеятельности и рекомендации при работе за компьютером для программистов [6].

7.1 Требования к помещению и рабочему месту

Место, в котором программист проводит свое рабочее время, должно соответствовать определенным нормам. Площадь на одно рабочее место, оборудованное компьютером, должна составлять не менее - 4,5 м. кв. Человеку должно быть комфортно на своем месте работы, где он проводит длительное время. Немаловажный фактор - цвет стен и мебели. Это может влиять на состояние, зрительное восприятие и настроение человека. Рекомендуется, чтобы цвет стен в помещении был приближен к цвету аппаратуры. Помещения, предусмотренные для работы с ПК, обязательно должны иметь искусственное и естественное освещение. Рабочее место программиста должно быть освещено равномерно. Освещение соответствует не только общей яркости, но и равномерности распределения света в помещении. Если свет распределен неправильно, то это приводит к неравномерной яркости на поверхности рабочего стола, бликам на экране монитора и другим проблемам. Значит, что степень освещения в комнате и яркость экрана должны быть примерно одинаковыми. Правильно выбранное освещение повышает работоспособность, концентрацию, а также уменьшает зрительное напряжение и снижает утомляемость организма. Важно следить за микроклиматом и поддерживать определённую температуру воздуха, поскольку во время работы компьютер выделяет тепло. Это может

привести к повышению температуры и снижению влажности в помещении. Есть санитарные нормы по поддержанию правильного микроклимата в помещениях с компьютерной техникой. Они подбираются в зависимости от времени года, размера помещения и характера трудового процесса.
Для обеспечения комфортной работы необходимо учитывать уровень звука. Шум может спровоцировать физиологические расстройства, повышенную утомляемость и уменьшить работоспособность. Рекомендуемый уровень шума в помещении не должен превышать 50 дБ. Также должны соблюдаться следующие требования:

- температура примерно 22 °С (допустимая – 20-24 °С);
- относительная влажность – 40-60% (допускается – не более 75%);
- скорость движения воздуха не более 0.1м/с;
- яркость на поверхности рабочего стола должна быть равномерной и не превышать 500 лк;
- общая освещенность должна быть на уровне 300 лк;
- рабочее место следует располагать возле окна, чтобы использовать естественное освещение;
- искусственное освещение должно быть в диапазоне от 4000 до 6500 К (приближенное к естественному).

7.2 Требования к режиму труда и отдыха

Во время работы за ПК необходимо правильно соблюдать режим труда и отдыха. Режим труда и отдыха при работе за ПК – это ряд действий, направленных на сохранность здоровья и работоспособности программистов. Соблюдение этих действий снижает утомляемость глаз и организма в целом, предотвращает развитие различных заболеваний, связанных с сидячим образом жизни. Ниже представлен список основных требований:

— важно сохранять правильную осанку тела при работе за компьютером;

— рекомендуется делать перерывы во время работы 10-15 минут каждый час;

— во время перерыва стоит выполнять гимнастику глаз, рук и спины;

— необходимо иметь правильное освещение в комнате без бликов на экране монитора;

— в течение дня стоит больше двигаться, заниматься спортом;

— важно контролировать свое психологическое состояние и вовремя делать перерывы, чтобы не допустить напряжения и стрессового состояния;

— требуется хорошо проветривать помещение, в котором находится ваше рабочее место;

— немаловажный фактор – планирование рабочего графика и выходных дней;

Соблюдение этих требований поможет сохранить здоровье, снизить риск возникновения усталости и уменьшить уровень стресса. Правильный режим труда и отдыха может также повысить производительность и эффективность работы.

Заключение

Во время написания данной выпускной квалификационной работы были выполнены следующие задания: было проведено сравнение аналогов, анализ и выбор среды разработки, представлена архитектура приложения, выбрана база данных, разработано и протестировано веб – приложение «Сайт футбольной команды ПГУ».

Также был получен определенный опыт разработки *WEB*-приложений с использованием фреймворка *Spring boot* на языке *Java*. Улучшены навыки работы с системой управления базами данных *PostgreSQL*. Проведенное тестирование показало, что веб – приложение «Сайт футбольной команды ПГУ» полностью соответствует всем поставленным требованиям и работает корректно.

В дальнейшем приложение можно усовершенствовать, расширив его функционал, а также улучшить интерфейс клиентской части приложения.

Во время проделанной работы был получен ценный опыт в разработке веб – приложений, который несомненно поможет в дальнейшем развитии профессиональных навыков.

Список использованных источников

1. Футбольный клуб «Урал»[Электронный ресурс]/Режим доступа: *https://fc-ural.ru/?ysclid=mbgotl2flb707811044,свободный*(дата обращения 06.05.2025).

2. Футбольный клуб «Зенит Санкт-Петербург».[Электронный ресурс]/Режим доступа:https://fc-zenit.ru/?ysclid=mbi1tl27qn261112117, свободный(дата обращения 07.05.2025).

3. Футбольный клуб «Зенит Пенза»[Электронный ресурс]/Режим доступа: https://zenitpenza.ru/home/?ysclid=mbi1vsaf8g183008711, свободный (дата обращения 07.05.2025).

4. Документация к фреймворку *Spring Framework* [Электронный ресурс]/Режим доступа: *https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html*,свободный (датаобращения 04.05.2023).

5. Ржеуцкая С.Ю. Базы данных. Язык *SQL*: учеб. пособие / С.Ю. Ржеуцкая. – Вологда: ВоГТУ, 2010. – 17-22 с.

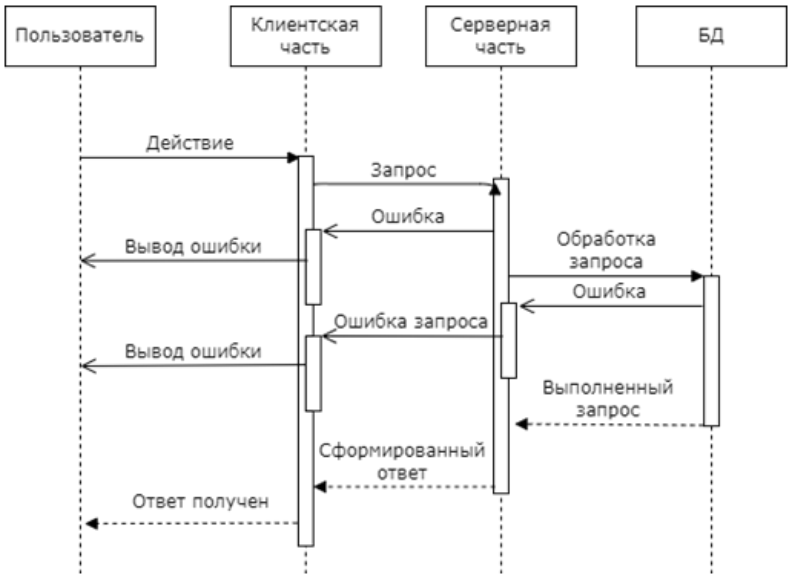
6. Санитарно-гигиенические рекомендации для организации рабочего места пользователя персональным компьютером [Электронный ресурс] / Режим доступа: *https://sakhgig.ru/main/info/664-sanitarno-gigienicheskie-rekomendacii-dlya-organizacii-rabochego-mesta-polzovatelya-personalnym-kompyuterom.html*, свободный (дата обращения 29.05.2025).

Приложение А

(Обязательное)
Веб-приложение «Сайт футбольной команды ПГУ»

UML-диаграммы

ПриложениеА.1-Диаграмма последовательностей

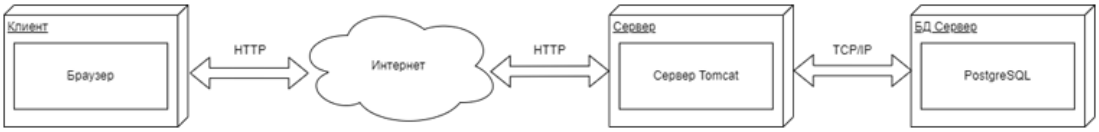


ПГУ 1.090301.23.002 Д14

UML-диаграмма последовательностей Веб-приложение «Сайт футбольной команды ПГУ»

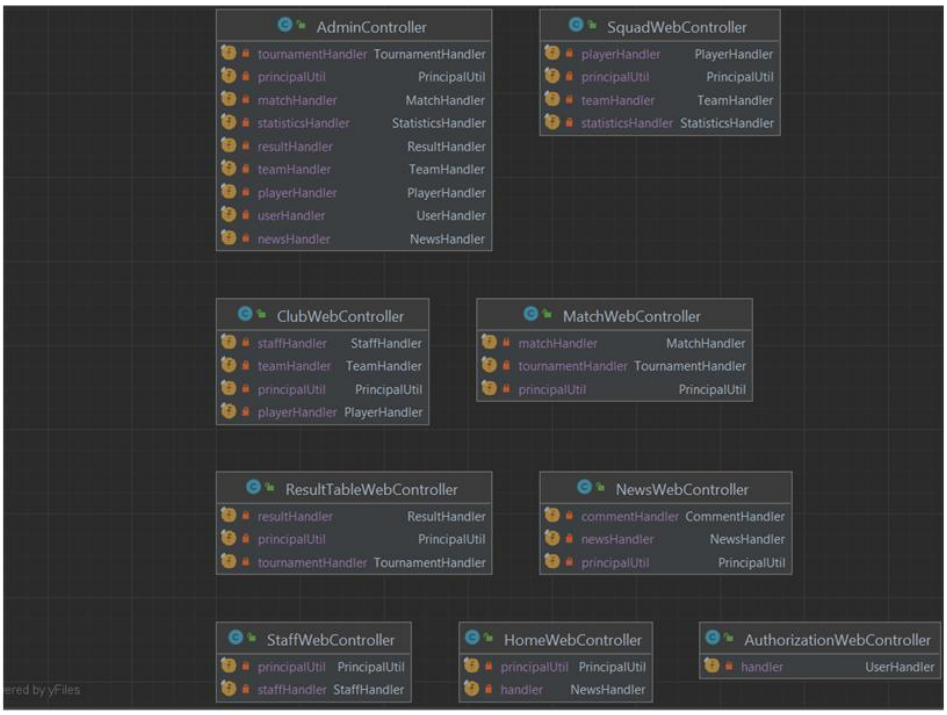
ФВТ, гр. 21ВВП1

ПриложениеА.2-Диаграмма развертывания



								ПГУ 1.090301.23.002 Д19	
Изм.	Лист	№ докум.	Подпись	Дата					
Разраб.		Хабибулин А.М.						UML-диаграмма развертывания Веб-приложение «Сайт футбольной команды ПГУ»	Лит. / Масса / Масштаб
Пров.		Юрина О.Ф.							
Т. контр.									Лист 1 / Листов1
Н. контр.		Анисов А.С.							ФВТ, гр. 21ВВП1
Утв.		Матвеенко М.А.							

ПриложениеА.3-Диаграмма классов



					ПГУ 1.090301.23.002 Д11	
Изм.	Лист	№ докум.	Подпись	Дата	UML-диаграмма классов Веб-приложение «Сайт футбольной команды ПГУ»	Лит. / Масса / Масштаб
Разраб.		Хабибулин А.М.				Лист / Листов
Пров.		Юрова О.В.				
Т. контр.						
Н. контр.		Антонов А. С.				ФВТ, гр. 21ВВП1
Утв.		Митрошин М.А.				

ПриложениеА.4-Диаграмма прецедентов

					ПГУ 1.090301.23.002 Д13	
Изм.	Лист	№ докум.	Подпись	Дата		
Разраб.		Хабибуллин А.М.			UML-диаграмма прецедентов Веб-приложение «Сайт футбольной команды ПГУ»	Лит. / Масса / Масштаб
Пров.		Юрина О.В.				
Т. контр.						Лист / Листов
Н. контр.		Быков А. С.			ФВТ, гр. 21ВВП1	
Утв.		Молчанов М.А.				

Приложение Б

(Обязательное)
Веб-приложение «Сайт футбольной команды ПГУ»

Листинг программы

Файл User.java

```
@Getter
@Setter
@Entity
@Table(name = "fc_user")
public class User implements UserDetails {
    @Id
    @SequenceGenerator(name = "seq_gen_user", sequenceName = "seq_user",
allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"seq_gen_user")
    @Column(name = "id")
    private Integer id;
    @Column(name = "username")
    private String nameUser;
    @Column(name = "email")
    private String email;
    @Column(name = "password")
    private String password;
    @ManyToOne
    @JoinColumn(name = "user_role_id")
    private Role userRole;
    @OneToMany(mappedBy = "author")
    private List<News> news = new ArrayList<>();
    @OneToMany(mappedBy = "author")
    private List<Comment> comments = new ArrayList<>();
    @CreationTimestamp
    @Column(name = "created_at")
    private LocalDateTime createdAt;
    @UpdateTimestamp
    @Column(name = "updated_at")
    private LocalDateTime updatedAt;
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return Collections.singleton(userRole);
    }
    @Override
    public String getUsername() {
        return email;
    }
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    @Override
    public boolean isEnabled() {
        return true;
    }
    public boolean hasRole(String roleName) {
        if (roleName == null || roleName.isEmpty()) {
            return false;
 }
        return getAuthorities().stream()
                .map(GrantedAuthority::getAuthority)
                .anyMatch(roleName::equals);
```

```
    }

}
```

Файл Tournament.java

```
@Getter
@Setter
@Entity
@Table(name = "fc_tournament")
public class Tournament {
    @Id
    @SequenceGenerator(name = "seq_gen_tournament", sequenceName =
"seq_tournament", allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"seq_gen_tournament")
    @Column(name = "id")
    private Integer id;

    @Column(name = "tournament_name")
    private String tournamentName;

    @Column(name = "season")
    private String season;

    @OneToMany(mappedBy = "tournament")
    private List<Match> matches;

    @OneToMany(mappedBy = "tournament")
    private List<Result> resultTables;
}
```

Файл Team.java

```
@Getter
@Setter
@Entity
@Table(name = "fc_team")
public class Team {
    @Id
    @SequenceGenerator(name = "seq_gen_team", sequenceName = "seq_team",
allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"seq_gen_team")
    @Column(name = "id")
    private Integer id;

    @Column(name = "team_name")
    private String teamName;

    @Column(name = "team_short_name")
    private String teamShortName;

    @Column(name = "stadium")
    private String stadium;

    @Column(name = "is_club_team")
    private Boolean isClubTeam;

    @Column(name = "logo_url")
    private String logoUrl;

    @OneToMany(mappedBy = "team")
    private List<Staff> staff;
```

```
    @OneToMany(mappedBy = "team")
    private List<Player> players;
}
```

Файл Statistics.java

```
@Getter
@Setter
@Entity
@Table(name = "fc_player_statistics")
public class Statistics {
    @Id
    @SequenceGenerator(name = "seq_gen_statistics", sequenceName =
"seq_statistics", allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"seq_gen_statistics")
    @Column(name = "id")
    private Integer id;

    @ManyToOne
    @JoinColumn(name = "player_id", referencedColumnName = "id")
    private Player player;

    @ManyToOne
    @JoinColumn(name = "match_id", referencedColumnName = "id")
    private Match match;

    @Column(name = "goals")
    private int goals;

    @Column(name = "assists")
    private int assists;

    @Column(name = "yellow_cards")
    private int yellowCards;

    @Column(name = "red_cards")
    private int redCards;

    @Column(name = "played_minutes")
    private int playedMinutes;

    @Column(name = "shots")
    private int shots;

    @Column(name = "passes")
    private int passes;

    @Column(name = "season")
    private LocalDate season;
}
```

Файл Role.java

```
@Getter
@Setter
@Entity
@Table(name = "fc_role_user", uniqueConstraints = {
        @UniqueConstraint(columnNames = {"role_name"}),
        @UniqueConstraint(columnNames = {"description"})
})
public class Role implements GrantedAuthority {
    @Id
```

```
    @SequenceGenerator(name = "seq_gen_role", sequenceName = "seq_role",
allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"seq_gen_role")
    @Column(name = "id")
    private Integer id;
    @Column(name = "role_name")
    private String roleName;
    @Column(name = "description")
    private String description;
    @Override
    public String getAuthority() {
        return roleName;
    }
}
```

Файл Result.java

```
@Getter
@Setter
@Entity
@Table(name = "fc_results",
        uniqueConstraints = {@UniqueConstraint(columnNames = {"team_id",
"tournament_id"})})
public class Result {
    @Id
    @SequenceGenerator(name = "seq_gen_results", sequenceName =
"seq_results", allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"seq_gen_results")
    @Column(name = "id")
    private Integer id;

    @Column(name = "position")
    private int position;

    @Column(name = "played")
    private int played;

    @Column(name = "wins")
    private int wins;

    @Column(name = "draws")
    private int draws;

    @Column(name = "losses")
    private int losses;

    @Column(name = "points")
    private int points;

    @ManyToOne
    @JoinColumn(name = "team_id")
    private Team team;

    @ManyToOne
    @JoinColumn(name = "tournament_id")
    private Tournament tournament;
}
```

Файл Player.java

```
@Getter
@Setter
@Entity
@Table(name = "fc_player")
public class Player {
    @Id
    @SequenceGenerator(name = "seq_gen_player", sequenceName = "seq_player",
allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"seq_gen_player")
    @Column(name = "id")
    private Integer id;

    @Column(name = "surname")
    private String surname;

    @Column(name = "name")
    private String name;

    @Column(name = "patronymic")
    private String patronymic;

    @Column(name = "birth_date")
    private LocalDate birthdate;

    @Column(name = "nationality")
    private String nationality;

    @Column(name = "player_number")
    private Integer playerNumber;

    @Enumerated(EnumType.STRING)
    @Column(name = "position")
    private PositionEnum position;

    @Column(name = "photo_url")
    private String photoUrl;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "team_id")
    private Team team;

    @OneToMany(mappedBy = "player")
    private List<Statistics> playerStats;
}
```

Файл News.java

```
@Getter
@Setter
@Entity
@Table(name = "fc_news")
public class News {
    @Id
    @SequenceGenerator(name = "seq_gen_news", sequenceName = "seq_news",
allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"seq_gen_news")
    @Column(name = "id")
    private Integer id;
```

```
    @Column(name = "title")
    private String title;

    @Column(name = "content", columnDefinition = "TEXT")
    private String content;

    @Column(name = "publication_date")
    private LocalDateTime publicationDate;

    @Column(name = "image_url")
    private String imageUrl;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "author_id", referencedColumnName = "id")
    private User author;

    @OneToMany(mappedBy = "news", cascade = CascadeType.ALL)
    private List<Comment> comments = new ArrayList<>();
}
```

Файл Match.java

```
@Getter
@Setter
@Entity
@Table(name = "fc_match")
public class Match {
    @Id
    @SequenceGenerator(name = "seq_gen_match", sequenceName = "seq_match",
allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"seq_gen_match")
    @Column(name = "id")
    private Integer id;

    @ManyToOne
    @JoinColumn(name = "home_team_id", referencedColumnName = "id")
    private Team homeTeam;

    @ManyToOne
    @JoinColumn(name = "away_team_id", referencedColumnName = "id")
    private Team awayTeam;

    @Column(name = "score")
    private String score;

    @Column(name = "location")
    private String location;

    @Enumerated(EnumType.STRING)
    @Column(name = "match_status")
    private MatchStatusEnum status;

    @Column(name = "date_time")
    private LocalDateTime dateTime;

    @ManyToOne
    @JoinColumn(name = "tournament_id")
    private Tournament tournament;

    @OneToMany(mappedBy = "match")
    private List<Statistics> playerStats;
}
```

Файл UserRepo.java

```
@Repository
public interface UserRepo extends JpaRepository<User, Integer> {
    @Query("select u from User u")
    Page<User> findAllUser(Pageable pageable);

    @Query("select u from User u where u.id = :id")
    Optional<User> findById(@Param("id") Integer id);

    @Query("select u from User u where u.nameUser like %:searchTerm%")
    Page<User> findUserByUsername(@Param("searchTerm") String searchTerm,
Pageable pageable);

    @Query("select u from User u where u.email = :email")
    Optional<User> findUserByEmail(@Param("email") String email);

    @Query("select u from User u join u.userRole r where r.description =
:description")
    Page<User> findUsersByRole(@Param("description") String description,
Pageable pageable);

}
```

Файл TournamentRepo.java

```
@Repository
public interface TournamentRepo extends JpaRepository<Tournament, Integer> {
    @Query("select t from Tournament t")
    List<Tournament> findAllTournament();

    @Query("select t from Tournament t where t.id = :id")
    Optional<Tournament> findById(@Param("id") Integer id);
}
```

Файл TeamRepo.java

```
@Repository
public interface TeamRepo extends JpaRepository<Team, Integer> {
    @Query("select t from Team t")
    Page<Team> findAllTeam(Pageable pageable);

    @Query("select t from Team t where t.id = :id")
    Optional<Team> findById(@Param("id") Integer id);

    @Query("select t from Team t where t.isClubTeam = true")
    Optional<Team> findClubTeam();

    @Query("select t.id from Team t where t.isClubTeam = true")
    Integer findClubTeamId();
}
```

Файл StatisticsRepo.java

```
public interface StatisticsRepo extends JpaRepository<Statistics, Integer> {
    @EntityGraph(attributePaths = {"player"})
    @Query("select s from Statistics s")
    Page<Statistics> findAllStatistics(Pageable pageable);

    @Query("select s from Statistics s where s.player.id = :id")
    List<Statistics> findStatisticsByPlayerId(@Param("id") Integer id);

    @EntityGraph(attributePaths = {"player"})
    @Query("select s from Statistics s where s.id = :id")
```

```
    Optional<Statistics> findById(@Param("id") Integer id);
}
```

Файл RoleRepo.java

```
public interface RoleRepo extends JpaRepository<Role, Integer> {
    @Query("select r from Role r")
    Page<Role> findAllRole(Pageable pageable);

    @Query("select r from Role r where r.id = :id")
    Optional<Role> findById(@Param("id") Integer id);

    @Query("select r from Role r where r.description = :description")
    Optional<Role> findRoleByDescription(@Param("description") String
description);

    @Query("select case when count(r) > 0 then true else false end from Role
r where r.roleName = :roleName or r.description = :description")
    Boolean existsByRoleNameOrDescription(@Param("roleName") String roleName,
@Param("description") String description);

}
```

Файл ResultRepo.java

```
public interface ResultRepo extends JpaRepository<Result, Integer> {
    @Query("select r from Result r order by r.points desc")
    List<Result> findAllResult();

    @Query("select r from Result r where r.id = :id")
    Optional<Result> findById(@Param("id") Integer id);

    @Query("select r from Result r where r.tournament.id = :id order by
r.points desc")
    List<Result> findResultByTournamentId(@Param("id") Integer id);

    @Query("select r from Result r where r.team.id = :teamId and
r.tournament.id = :tournamentId")
    Optional<Result> findByTeamAndTournamentId(@Param("teamId") Integer
teamId, @Param("tournamentId") Integer tournamentId);
}
```

Файл PlayerRepo.java

```
public interface PlayerRepo extends JpaRepository<Player, Integer> {
    @Query("select p from Player p")
    Page<Player> findAllPlayer(Pageable pageable);

    @Query("select p from Player p where p.id = :id")
    Optional<Player> findById(@Param("id") Integer id);

    @Query("select p from Player p where p.team.id = :id")
    List<Player> findByTeamId(@Param("id") Integer id);

    @Query("select p from Player p where " +
            "p.surname like %:searchTerm% or " +
            "p.name like %:searchTerm% or " +
            "p.patronymic like %:searchTerm%")
    Page<Player> findPlayerByName(@Param("searchTerm") String searchTerm,
Pageable pageable);
}
```

Файл NewsRepo.java

```
public interface NewsRepo extends JpaRepository<News, Integer> {
```

```
    @Query("select n from News n")
    Page<News> findAllNews(Pageable pageable);

    @Query("select n from News n where n.id = :id")
    Optional<News> findById(@Param("id") Integer id);

}
```

Файл MatchRepo.java

```
public interface MatchRepo extends JpaRepository<Match, Integer> {

    @EntityGraph(attributePaths = {"homeTeam", "awayTeam"})
    @Query("select m from Match m order by m.dateTime desc")
    Page<Match> findAllMatch(Pageable pageable);

    @EntityGraph(attributePaths = {"homeTeam", "awayTeam"})
    @Query("select m from Match m where m.tournament.id = :id order by
m.dateTime desc")
    Page<Match> findMatchByTournament(@Param("id") Integer id, Pageable
pageable);

    @EntityGraph(attributePaths = {"homeTeam", "awayTeam"})
    @Query("select m from Match m where m.id = :id")
    Optional<Match> findById(@Param("id") Integer id);
}
```

Файл UserService.java

```
@Service
public class UserService {
    private final UserRepo userRepo;
    private final MessageUtil messageUtil;

    public UserService(UserRepo userRepo,
                       MessageUtil messageUtil) {
        this.userRepo = userRepo;
        this.messageUtil = messageUtil;
    }

    @Transactional(readOnly = true)
    public Page<User> getAllUser(Pageable pageable){
        return userRepo.findAllUser(pageable);
    }

    @Transactional(readOnly = true)
    public Optional<User> getUserById(@NonNull Integer id){
        return userRepo.findById(id);
    }

    @Transactional(readOnly = true)
    public Optional<User> getUserByEmail(@NonNull String email){
        return userRepo.findUserByEmail(email);
    }


    @Transactional(readOnly = true)
    public Page<User> getUserByUsername(@NonNull String username, Pageable
pageable){
        return userRepo.findUserByUsername(username, pageable);
    }

    @Transactional(readOnly = true)
```

```
    public Page<User> getUserByRole(@NonNull String description, Pageable
pageable){
        return userRepo.findUsersByRole(description, pageable);
    }

    @Transactional
    public User saveUser(@NonNull User user){
        try{
            return userRepo.save(user);
        }catch (DataIntegrityViolationException e){
            throw new ConflictResourceException(e.getMessage());
        }
    }

    @Transactional
    public User getReferenceOrNew(@Nullable Integer id){
        return id == null ? new User() : userRepo.getReferenceById(id);
    }

    @Transactional
    public void deleteUserById(@NonNull Integer id){
        User user = getUserById(id)
                .orElseThrow(()->new
NotFoundException(messageUtil.getMessage("user.id.not-found", id)));

        userRepo.delete(user);
    }
}
```

Файл TournamentService.java

```
@Service
public class TournamentService {
    private final TournamentRepo tournamentRepo;

    public TournamentService(TournamentRepo tournamentRepo,
                             MessageUtil messageUtil) {
        this.tournamentRepo = tournamentRepo;
    }

    @Transactional(readOnly = true)
    public List<Tournament> getAllTournament() {
        return tournamentRepo.findAllTournament();
    }

    @Transactional(readOnly = true)
    public Optional<Tournament> getTournamentById(@NonNull Integer id) {
        return tournamentRepo.findById(id);
    }

    @Transactional
    public Tournament saveTournament(@NonNull Tournament tournament) {
        try {
            return tournamentRepo.save(tournament);
        } catch (DataIntegrityViolationException e) {
            throw new ConflictResourceException(e.getMessage());
        }
    }

    @Transactional
    public Tournament getReferenceOrNew(@Nullable Integer id) {
        return id == null ? new Tournament() :
tournamentRepo.getReferenceById(id);
    }
```

```
    @Transactional
    public void deleteTournamentById(@NonNull Integer id) {
        tournamentRepo.deleteById(id);
    }
}
```

Файл TeamService.java

```
@Service
public class TeamService {
    private final TeamRepo teamRepo;

    public TeamService(TeamRepo teamRepo,
                       MessageUtil messageUtil) {
        this.teamRepo = teamRepo;
    }

    @Transactional(readOnly = true)
    public Page<Team> getAllTeam(Pageable pageable) {
        return teamRepo.findAllTeam(pageable);
    }

    @Transactional(readOnly = true)
    public Optional<Team> getTeamById(@NonNull Integer id) {
        return teamRepo.findById(id);
    }

    @Transactional(readOnly = true)
    public Optional<Team> getClubTeam() {
        return teamRepo.findClubTeam();
    }

    @Transactional(readOnly = true)
    public Integer getClubTeamId() {
        return teamRepo.findClubTeamId();
    }

    @Transactional
    public Team saveTeam(@NonNull Team team) {
        try {
            return teamRepo.save(team);
        } catch (DataIntegrityViolationException e) {
            throw new ConflictResourceException(e.getMessage());
        }
    }

    @Transactional
    public Team getReferenceOrNew(@Nullable Integer id) {
        return id == null ? new Team() : teamRepo.getReferenceById(id);
    }

    @Transactional
    public void deleteTeamById(@NonNull Integer id) {
        teamRepo.deleteById(id);
    }
}
```

Файл StatisticsService.java

```
@Service
public class StatisticsService {
    private final StatisticsRepo statisticsRepo;
    private final MessageUtil messageUtil;
```

```
    public StatisticsService(StatisticsRepo statisticsRepo,
                             MessageUtil messageUtil){
        this.statisticsRepo = statisticsRepo;
        this.messageUtil = messageUtil;
    }

    @Transactional(readOnly = true)
    public Page<Statistics> getAllStatistics(Pageable pageable) {
        return statisticsRepo.findAllStatistics(pageable);
    }

    @Transactional(readOnly = true)
    public List<Statistics> getStatisticsByPlayerId(@NonNull Integer
playerId) {
        return statisticsRepo.findStatisticsByPlayerId(playerId);
    }

    @Transactional(readOnly = true)
    public Optional<Statistics> getStatisticsById(@NonNull Integer id) {
        return statisticsRepo.findById(id);
    }

    @Transactional
    public Statistics saveStatistics(@NonNull Statistics match) {
        try {
            return statisticsRepo.save(match);
        } catch (DataIntegrityViolationException e) {
            throw new ConflictResourceException(e.getMessage());
        }
    }

    @Transactional
    public Statistics getReferenceOrNew(@Nullable Integer id) {
        return id == null ? new Statistics() :
statisticsRepo.getReferenceById(id);
    }

    @Transactional
    public void deleteStatisticsById(@NonNull Integer id) {
        statisticsRepo.deleteById(id);
    }
}
```

Файл RoleService.java

```
@Service
public class RoleService {
    private final RoleRepo roleRepo;
    private final MessageUtil messageUtil;

    public RoleService (RoleRepo roleRepo,
                        MessageUtil messageUtil){
        this.roleRepo = roleRepo;
        this.messageUtil = messageUtil;
    }

    @Transactional(readOnly = true)
    public Page<Role> getAllRole(Pageable pageable){
        return roleRepo.findAllRole(pageable);
    }

    @Transactional(readOnly = true)
    public Optional<Role> getRoleById(@NonNull Integer id){
```

```
        return roleRepo.findById(id);
    }

    @Transactional(readOnly = true)
    public Optional<Role> getRoleByDescription(@NonNull String description){
        return roleRepo.findRoleByDescription(description);
    }

    @Transactional
    public Role saveRole(@NonNull Role role){
        if(roleRepo.existsByRoleNameOrDescription(role.getRoleName(),
role.getDescription())){
            throw new
ConflictResourceException(messageUtil.getMessage("role.exist",
role.getRoleName(), role.getDescription()));
        }
        try{
            return roleRepo.save(role);
        }catch (DataIntegrityViolationException e){
            throw new ConflictResourceException(e.getMessage());
        }
    }

    @Transactional
    public Role getReferenceOrNew(@Nullable Integer id){
        return id == null ? new Role() : roleRepo.getReferenceById(id);
    }

    @Transactional
    public void deleteRoleById(@NonNull Integer id){
        Role role = getRoleById(id)
                .orElseThrow(()->new
NotFoundException(messageUtil.getMessage("role.id.not-found", id)));

        roleRepo.delete(role);
    }
}
```

Файл ResultService.java

```
@Service
public class ResultService {
    private final ResultRepo resultRepo;

    public ResultService(ResultRepo resultRepo) {
        this.resultRepo = resultRepo;
    }

    @Transactional(readOnly = true)
    public List<Result> getAllResult() {
        return resultRepo.findAllResult();
    }

    @Transactional(readOnly = true)
    public List<Result> getResultByTournament(@NonNull Integer id) {
        return resultRepo.findResultByTournamentId(id);
    }

    @Transactional(readOnly = true)
    public Optional<Result> getResultById(@NonNull Integer id) {
        return resultRepo.findById(id);
    }

    @Transactional(readOnly = true)
```

```
    public Optional<Result> getResultByTeamAndTournamentId(@NonNull Integer
teamId, @NonNull Integer tournamentId) {
        return resultRepo.findByTeamAndTournamentId(teamId, tournamentId);
    }

    @Transactional
    public Result saveResult(@NonNull Result result) {
        try {
            return resultRepo.save(result);
        } catch (DataIntegrityViolationException e) {
            throw new ConflictResourceException(e.getMessage());
        }
    }

    @Transactional
    public Result getReferenceOrNew(@Nullable Integer id) {
        return id == null ? new Result() : resultRepo.getReferenceById(id);
    }

    @Transactional
    public void deleteResultById(@NonNull Integer id) {
        resultRepo.deleteById(id);
    }
}
```

Файл PlayerService.java

```
@Service
public class PlayerService {
    private final PlayerRepo playerRepo;
    private final MessageUtil messageUtil;

    public PlayerService(PlayerRepo playerRepo,
                         MessageUtil messageUtil) {
        this.playerRepo = playerRepo;
        this.messageUtil = messageUtil;
    }

    @Transactional(readOnly = true)
    public Page<Player> getAllPlayer(Pageable pageable) {
        return playerRepo.findAllPlayer(pageable);
    }

    @Transactional(readOnly = true)
    public Optional<Player> getPlayerById(@NonNull Integer id) {
        return playerRepo.findById(id);
    }

    @Transactional(readOnly = true)
    public List<Player> getPlayerByTeamId(@NonNull Integer id) {
        return playerRepo.findByTeamId(id);
    }

    @Transactional(readOnly = true)
    public Page<Player> getPlayerByName(@NonNull String name, Pageable
pageable) {
        return playerRepo.findPlayerByName(name, pageable);
    }

    @Transactional
    public Player savePlayer(@NonNull Player player) {
        try {
            return playerRepo.save(player);
        } catch (DataIntegrityViolationException e) {
```

```
            throw new ConflictResourceException(e.getMessage());
        }
    }

    @Transactional
    public Player getReferenceOrNew(@Nullable Integer id) {
        return id == null ? new Player() : playerRepo.getReferenceById(id);
    }

    @Transactional
    public void deletePlayerById(@NonNull Integer id) {
        playerRepo.deleteById(id);
    }
}
```

Файл NewsService.java

```
@Service
public class NewsService {
    private final NewsRepo newsRepo;
    private final MessageUtil messageUtil;

    public NewsService(NewsRepo newsRepo,
                       MessageUtil messageUtil) {
        this.newsRepo = newsRepo;
        this.messageUtil = messageUtil;
    }

    @Transactional(readOnly = true)
    public Page<News> getAllNews(Pageable pageable){
        return newsRepo.findAllNews(pageable);
    }

    @Transactional(readOnly = true)
    public Optional<News> getNewsById(@NonNull Integer id){
        return newsRepo.findById(id);
    }

    @Transactional
    public News saveNews(@NonNull News news){
        try{
            return newsRepo.save(news);
        }catch (DataIntegrityViolationException e){
            throw new ConflictResourceException(e.getMessage());
        }
    }

    @Transactional
    public News getReferenceOrNew(@Nullable Integer id){
        return id == null ? new News() : newsRepo.getReferenceById(id);
    }

    @Transactional
    public void deleteNewsById(@NonNull Integer id){
        newsRepo.deleteById(id);
    }
}
```

Файл MatchService.java

```
@Service
public class MatchService {
    private final MatchRepo matchRepo;
    private final MessageUtil messageUtil;
```

```
    public MatchService(MatchRepo matchRepo,
                        MessageUtil messageUtil){
        this.matchRepo = matchRepo;
        this.messageUtil = messageUtil;
    }

    @Transactional(readOnly = true)
    public Page<Match> getAllMatch(Pageable pageable) {
        return matchRepo.findAllMatch(pageable);
    }

    @Transactional(readOnly = true)
    public Page<Match> getMatchByTournament(Pageable pageable,
                                            @NonNull Integer id) {
        return matchRepo.findMatchByTournament(id, pageable);
    }

    @Transactional(readOnly = true)
    public Optional<Match> getMatchById(@NonNull Integer id) {
        return matchRepo.findById(id);
    }

    @Transactional
    public Match saveMatch(@NonNull Match match) {
        try {
            return matchRepo.save(match);
        } catch (DataIntegrityViolationException e) {
            throw new ConflictResourceException(e.getMessage());
        }
    }

    @Transactional
    public Match getReferenceOrNew(@Nullable Integer id) {
        return id == null ? new Match() : matchRepo.getReferenceById(id);
    }

    @Transactional
    public void deleteMatchById(@NonNull Integer id) {
        Match match = getMatchById(id)
                .orElseThrow(() -> new
NotFoundException(messageUtil.getMessage("match.id.not-found", id)));

        matchRepo.delete(match);
    }
}
```

Файл AdminController.java

```
@Controller
@RequestMapping("/admin")
public class AdminController {
    private final UserHandler userHandler;
    private final PlayerHandler playerHandler;
    private final TeamHandler teamHandler;
    private final StatisticsHandler statisticsHandler;
    private final ResultHandler resultHandler;
    private final TournamentHandler tournamentHandler;
    private final MatchHandler matchHandler;
    private final NewsHandler newsHandler;
    private final PrincipalUtil principalUtil;

    public AdminController(UserHandler userHandler,
                           PlayerHandler playerHandler,
                           TeamHandler teamHandler,
```

```
                            StatisticsHandler statisticsHandler,
                            ResultHandler resultHandler,
                            TournamentHandler tournamentHandler,
                            MatchHandler matchHandler,
                            NewsHandler newsHandler,
                            PrincipalUtil principalUtil) {
        this.userHandler = userHandler;
        this.playerHandler = playerHandler;
        this.teamHandler = teamHandler;
        this.statisticsHandler = statisticsHandler;
        this.resultHandler = resultHandler;
        this.tournamentHandler = tournamentHandler;
        this.matchHandler = matchHandler;
        this.newsHandler = newsHandler;
        this.principalUtil = principalUtil;
    }

    // Пользователи
    @ResponseStatus(HttpStatus.OK)
    @GetMapping("/user")
    public String getUsers(Principal principal,
                           @RequestParam(name = "size", required = false,
defaultValue = "10") Integer size,
                           @RequestParam(name = "page", required = false,
defaultValue = "0") Integer page,
                           Model model) {
        Pageable pageable = PageRequest.of(page, size,
Sort.by("nameUser").ascending());
        Page<UserWithRoleView> userWithRoleView =
userHandler.handlerGetAllUserWithRole(pageable);

        model.addAttribute("currentUser",
principalUtil.getCurrentUser(principal));
        model.addAttribute("usersList", userWithRoleView.getContent());
        model.addAttribute("currentPage", page);
        model.addAttribute("totalPages", userWithRoleView.getTotalPages());
        return "user";
    }

    @GetMapping("/user/search")
    public String searchUsersByName(@RequestParam String username,
                                    @RequestParam(name = "size", defaultValue
= "10") int size,
                                    @RequestParam(name = "page", defaultValue
= "0") int page,
                                    Model model,
                                    Principal principal) {
        Pageable pageable = PageRequest.of(page, size,
Sort.by("nameUser").ascending());
        Page<UserWithRoleView> pageResult =
userHandler.handlerGetUserByUsername(username, pageable);

        model.addAttribute("currentUser",
principalUtil.getCurrentUser(principal));
        model.addAttribute("usersList", pageResult.getContent());
        model.addAttribute("totalPages", pageResult.getTotalPages());
        model.addAttribute("currentPage", page);
        model.addAttribute("pageSize", size);
        model.addAttribute("searchQuery", username);
        return "user";
    }

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

```
    public String editUserById(@RequestParam("userId") Integer userId,
                               @Valid @ModelAttribute @NotNull UserUpdateReq
req,
                               RedirectAttributes redirectAttributes) {
        try {
            userHandler.handlerUpdateUserById(userId, req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/user";
        }
        return "redirect:/admin/user";
    }

    @PostMapping("/user/delete")
    public String deleteUserById(@RequestParam("userId") Integer userId) {
        userHandler.handlerDeleteUserById(userId);
        return "redirect:/admin/user";
    }


    //Игроки
    @ResponseStatus(HttpStatus.OK)
    @GetMapping("/players")
    public String getPlayers(Principal principal,
                             @RequestParam(name = "size", required = false,
defaultValue = "10") Integer size,
                             @RequestParam(name = "page", required = false,
defaultValue = "0") Integer page,
                             Model model) {
        Pageable pageable = PageRequest.of(page, size,
Sort.by("surname").ascending());
        Page<PlayerView> playerViews =
playerHandler.handlerGetAllPlayer(pageable);
        Page<TeamView> teamViews =
teamHandler.handlerGetAllTeam(PageRequest.of(page, 100));

        model.addAttribute("currentUser",
principalUtil.getCurrentUser(principal));
        model.addAttribute("playersList", playerViews.getContent());
        model.addAttribute("teamsList", teamViews.getContent());
        model.addAttribute("currentPage", page);
        model.addAttribute("totalPages", playerViews.getTotalPages());
        return "player-admin";
    }

    @PostMapping("/players/edit")
    public String editPlayerById(@RequestParam("playerId") Integer playerId,
                                 @RequestParam(value = "teamId", required =
false) Integer teamId,
                                 @Valid @ModelAttribute @NotNull PlayerReq
req,
                                 RedirectAttributes redirectAttributes) {
        try {
            playerHandler.handlerUpdatePlayerById(playerId, req);
            if (teamId != null) {
                playerHandler.handlerAddToTeam(playerId, teamId);
            }
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/players";
        }
        return "redirect:/admin/players";
    }
```

```
    @PostMapping("/players/add")
    public String addPlayer(@Valid @ModelAttribute @NotNull PlayerReq req,
                            RedirectAttributes redirectAttributes) {
        try {
            playerHandler.handlerCreatePlayer(req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/players";
        }
        return "redirect:/admin/players";
    }

    @PostMapping("/players/delete")
    public String deletePlayerById(@RequestParam("playerId") Integer
playerId) {
        playerHandler.handlerDeletePlayerById(playerId);
        return "redirect:/admin/players";
    }

    @ResponseStatus(HttpStatus.OK)
    @GetMapping("/teams")
    public String getTeams(Principal principal,
                           @RequestParam(name = "size", required = false,
defaultValue = "10") Integer size,
                           @RequestParam(name = "page", required = false,
defaultValue = "0") Integer page,
                           Model model) {
        Pageable pageable = PageRequest.of(page, size,
Sort.by("teamName").ascending());
        Page<TeamView> teamViews = teamHandler.handlerGetAllTeam(pageable);

        model.addAttribute("currentUser",
principalUtil.getCurrentUser(principal));
        model.addAttribute("teamsList", teamViews.getContent());
        model.addAttribute("currentPage", page);
        model.addAttribute("totalPages", teamViews.getTotalPages());
        return "team-admin";
    }

    @PostMapping("/teams/edit")
    public String editTeamById(@RequestParam("teamId") Integer teamId,
                               @Valid @ModelAttribute @NotNull TeamReq req,
                               RedirectAttributes redirectAttributes) {
        try {
            teamHandler.handlerUpdateTeamById(teamId, req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/teams";
        }
        return "redirect:/admin/teams";
    }

    @PostMapping("/teams/add")
    public String addTeam(@Valid @ModelAttribute @NotNull TeamReq req,
                          RedirectAttributes redirectAttributes) {
        try {
            teamHandler.handlerCreateTeam(req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/teams";
        }
        return "redirect:/admin/teams";
```

```
    }

    @PostMapping("/teams/delete")
    public String deleteTeamById(@RequestParam("teamId") Integer teamId) {
        teamHandler.handlerDeleteTeamById(teamId);
        return "redirect:/admin/teams";
    }

    @ResponseStatus(HttpStatus.OK)
    @GetMapping("/stats")
    public String getStats(Principal principal,
                           @RequestParam(name = "size", required = false,
defaultValue = "10") Integer size,
                           @RequestParam(name = "page", required = false,
defaultValue = "0") Integer page,
                           Model model) {
        Pageable pageable = PageRequest.of(page, size);
        Page<StatisticsView> statisticsViews =
statisticsHandler.handlerGetAllStatistics(pageable);

        model.addAttribute("currentUser",
principalUtil.getCurrentUser(principal));
        model.addAttribute("statsList", statisticsViews.getContent());
        model.addAttribute("currentPage", page);
        model.addAttribute("totalPages", statisticsViews.getTotalPages());
        return "stats-admin";
    }

    @PostMapping("/stats/edit")
    public String editStatsById(@RequestParam("statsId") Integer statsId,
                                @Valid @ModelAttribute @NotNull StatisticsReq
req,
                                RedirectAttributes redirectAttributes) {
        try {
            statisticsHandler.handlerUpdateStatisticsById(statsId, req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/stats";
        }
        return "redirect:/admin/stats";
    }

    @PostMapping("/stats/add")
    public String addStats(@Valid @ModelAttribute @NotNull StatisticsReq req,
                           RedirectAttributes redirectAttributes) {
        try {
            statisticsHandler.handlerCreateStatistics(req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/stats";
        }
        return "redirect:/admin/stats";
    }

    @PostMapping("/stats/delete")
    public String deleteStatsById(@RequestParam("statId") Integer statId) {
        statisticsHandler.handlerDeleteStatisticsById(statId);
        return "redirect:/admin/stats";
    }

    @GetMapping("/tables")
    public String getTables(Principal principal,
```

```
                                @RequestParam(required = false) Integer
tournamentId,
                                Model model) {
        List<TournamentView> allTournaments =
tournamentHandler.handlerGetAllTournament();

        if (tournamentId != null) {
            List<ResultView> resultsByTournament =
resultHandler.handlerGetResultByTournament(tournamentId);
            model.addAttribute("resultsByTournament", resultsByTournament);
            model.addAttribute("selectedTournamentId", tournamentId);
        }

        model.addAttribute("allTournaments", allTournaments);
        model.addAttribute("currentUser",
principalUtil.getCurrentUser(principal));
        return "result-admin";
    }

    @PostMapping("/tables/edit")
    public String editTable(@Valid @ModelAttribute @NotNull ResultReq req,
                            RedirectAttributes redirectAttributes) {
        try {
            resultHandler.handlerUpdateResultById(req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/tables";
        }
        return "redirect:/admin/tables";
    }

    @PostMapping("/tables/add")
    public String addTable(@Valid @ModelAttribute @NotNull ResultReq req,
                           RedirectAttributes redirectAttributes) {
        try {
            resultHandler.handlerCreateResult(req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/tables";
        }
        return "redirect:/admin/tables";
    }

    @PostMapping("/tables/delete")
    public String deleteTableById(@RequestParam("resultId") Integer resultId)
{
        resultHandler.handlerDeleteResultById(resultId);
        return "redirect:/admin/tables";
    }

    @GetMapping("/tour")
    public String getTours(Principal principal,
                           Model model) {
        List<TournamentView> allTournaments =
tournamentHandler.handlerGetAllTournament();

        model.addAttribute("allTournaments", allTournaments);
        model.addAttribute("currentUser",
principalUtil.getCurrentUser(principal));
        return "tour-admin";
    }

    @PostMapping("/tour/edit")
```

```
    public String editTour(@RequestParam("tourId") Integer tourId,
                           @Valid @ModelAttribute @NotNull TournamentReq req,
                           RedirectAttributes redirectAttributes) {
        try {
            tournamentHandler.handlerUpdateTournamentById(tourId, req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/tour";
        }
        return "redirect:/admin/tour";
    }

    @PostMapping("/tour/add")
    public String addTour(@Valid @ModelAttribute @NotNull TournamentReq req,
                          RedirectAttributes redirectAttributes) {
        try {
            tournamentHandler.handlerCreateTournament(req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/tour";
        }
        return "redirect:/admin/tour";
    }

    @PostMapping("/tour/delete")
    public String deleteTourById(@RequestParam("tourId") Integer tourId) {
        tournamentHandler.handlerDeleteTournamentById(tourId);
        return "redirect:/admin/tour";
    }

    @GetMapping("/matches")
    public String getMatches(Principal principal,
                             @RequestParam(name = "size", required = false,
defaultValue = "10") Integer size,
                             @RequestParam(name = "page", required = false,
defaultValue = "0") Integer page,
                             Model model) {
        Pageable pageable = PageRequest.of(page, size,
Sort.by("dateTime").ascending());
        Page<MatchView> matchViews =
matchHandler.handlerGetAllMatch(pageable);

        model.addAttribute("matches", matchViews.getContent());
        model.addAttribute("currentPage", page);
        model.addAttribute("totalPages", matchViews.getTotalPages());
        model.addAttribute("currentUser",
principalUtil.getCurrentUser(principal));
        return "match-admin";
    }

    @PostMapping("/matches/edit")
    public String editMatch(@RequestParam("matchId") Integer matchId,
                            @Valid @ModelAttribute @NotNull MatchReq req,
                            RedirectAttributes redirectAttributes) {
        try {
            matchHandler.handlerUpdateMatchById(matchId, req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/matches";
        }
        return "redirect:/admin/matches";
    }
```

```
    @PostMapping("/matches/add")
    public String addMatch(@Valid @ModelAttribute @NotNull MatchReq req,
                           RedirectAttributes redirectAttributes) {
        try {
            matchHandler.handlerCreateMatch(req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/matches";
        }
        return "redirect:/admin/matches";
    }

    @PostMapping("/matches/delete")
    public String deleteMatchById(@RequestParam("matchId") Integer matchId) {
        matchHandler.handlerDeleteMatchById(matchId);
        return "redirect:/admin/matches";
    }

    @GetMapping("/news")
    public String getNews(Principal principal,
                          @RequestParam(name = "size", required = false,
defaultValue = "10") Integer size,
                          @RequestParam(name = "page", required = false,
defaultValue = "0") Integer page,
                          Model model) {
        Pageable pageable = PageRequest.of(page, size,
Sort.by("publicationDate").ascending());
        Page<NewsView> newsViews = newsHandler.handlerGetAllNews(pageable);

        model.addAttribute("newsList", newsViews.getContent());
        model.addAttribute("currentPage", page);
        model.addAttribute("totalPages", newsViews.getTotalPages());
        model.addAttribute("currentUser",
principalUtil.getCurrentUser(principal));
        return "news-admin";
    }

    @PostMapping("/news/add")
    public String addNews(@Valid @ModelAttribute @NotNull NewsReq req,
                          RedirectAttributes redirectAttributes) {
        try {
            newsHandler.handlerCreateNews(req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/admin/news";
        }
        return "redirect:/admin/news";
    }

    @PostMapping("/news/delete")
    public String deleteNewsById(@RequestParam("matchId") Integer matchId) {
        newsHandler.handlerDeleteNewsById(matchId);
        return "redirect:/admin/news";
    }
}
```

Файл AuthorizationWebController.java

```
@Controller
@RequestMapping("/auth")
public class AuthorizationWebController {
    private final UserHandler handler;

    public AuthorizationWebController(UserHandler handler){
```

```
        this.handler = handler;
    }

    @ResponseStatus(HttpStatus.OK)
    @GetMapping("/singup")
    public String registrationPage(Model model){
        model.addAttribute("userCreateReq", new UserCreateReq("", "", ""));
        return "singup";
    }

    @PostMapping("/singup")
    public String registrationUser(@Valid @ModelAttribute @NotNull
UserCreateReq req,
                                   BindingResult bindingResult,
                                   Model model) {
        if (bindingResult.hasErrors()) {
            return "singup";
        }

        UserView user;

        try {
            user = handler.handlerCreateUser(req);
        } catch (ConflictResourceException e) {
            model.addAttribute("error", e.getMessage());
            return "singup";
        }

        if(user == null){
            model.addAttribute("error", "Пользователь с email: " +
req.email() + " уже существует");
            return "singup";
        }

        return "redirect:/auth/login";
    }

    @ResponseStatus(HttpStatus.OK)
    @GetMapping("/login")
    public String loginPage(@RequestParam(value = "error", required = false)
boolean error, Model model) {
        if (error) {
            model.addAttribute("errorMsg", "Неправильный логин или пароль");
        }
        return "login";
    }
}
```

Файл ClubWebController.java

```
@Controller
@RequestMapping("/club")
public class ClubWebController {
    private final PrincipalUtil principalUtil;
    private final TeamHandler teamHandler;
    private final StaffHandler staffHandler;
    private final PlayerHandler playerHandler;

    public ClubWebController(PrincipalUtil principalUtil,
                             TeamHandler teamHandler,
                             StaffHandler staffHandler,
                             PlayerHandler playerHandler){
        this.principalUtil = principalUtil;
        this.teamHandler = teamHandler;
```

```
        this.staffHandler = staffHandler;
        this.playerHandler = playerHandler;
    }

    @ResponseStatus(HttpStatus.OK)
    @GetMapping
    public String getClubTeam(
            Principal principal,
            Model model) {
        TeamView team = teamHandler.handlerGetClubTeam();
        StaffView headCoach = staffHandler.handlerGetHeadCoachByTeamId();
        List<PlayerView> playerViews =
playerHandler.handlerGetPlayerByTeamId(team.id());

        model.addAttribute("players", playerViews);
        model.addAttribute("user", principalUtil.getCurrentUser(principal));
        model.addAttribute("team", team);
        model.addAttribute("headCoach", headCoach);
        return "club";
    }
}
```

Файл HomeWebController.java

```
@Controller
@RequestMapping("/")
public class HomeWebController {
    private final NewsHandler handler;
    private final PrincipalUtil principalUtil;

    public HomeWebController(NewsHandler handler,
                             PrincipalUtil principalUtil){
        this.handler = handler;
        this.principalUtil = principalUtil;
    }

    @ResponseStatus(HttpStatus.OK)
    @GetMapping
    public String home(
            Principal principal,
            @RequestParam(name = "size", required = false, defaultValue =
"4") Integer size,
            @RequestParam(name = "page", required = false, defaultValue =
"0") Integer page,
            Model model) {
        Pageable pageable = PageRequest.of(page, size);
        Page<NewsView> newsPage = handler.handlerGetAllNews(pageable);

        model.addAttribute("user", principalUtil.getCurrentUser(principal));
        model.addAttribute("allNews", newsPage.getContent());
        model.addAttribute("currentPage", page);
        model.addAttribute("totalPages", newsPage.getTotalPages());
        return "home";
    }
}
```

Файл MatchWebController.java

```
@Controller
@RequestMapping("/match")
public class MatchWebController {
```

```
    private final MatchHandler matchHandler;
    private final TournamentHandler tournamentHandler;
    private final PrincipalUtil principalUtil;

    public MatchWebController(MatchHandler matchHandler,
                              TournamentHandler tournamentHandler,
                              PrincipalUtil principalUtil) {
        this.matchHandler = matchHandler;
        this.tournamentHandler = tournamentHandler;
        this.principalUtil = principalUtil;
    }

    @ResponseStatus(HttpStatus.OK)
    @GetMapping
    public String match(
            Principal principal,
            @RequestParam(required = false) Integer tournamentId,
            @RequestParam(name = "size", required = false, defaultValue =
"10") Integer size,
            @RequestParam(name = "page", required = false, defaultValue =
"0") Integer page,
            Model model) {
        List<TournamentView> allTournaments =
tournamentHandler.handlerGetAllTournament();

        if (tournamentId != null) {
            Pageable pageable = PageRequest.of(page, size);
            Page<MatchView> matchViews =
matchHandler.handlerGetMatchByTournament(pageable, tournamentId);

            model.addAttribute("allMatch", matchViews.getContent());
            model.addAttribute("currentPage", page);
            model.addAttribute("totalPages", matchViews.getTotalPages());
            model.addAttribute("selectedTournamentId", tournamentId);
        }

        model.addAttribute("allTournaments", allTournaments);
        model.addAttribute("user", principalUtil.getCurrentUser(principal));
        return "match";
    }
}
```

Файл NewsWebController.java

```
@Controller
@RequestMapping("/news")
public class NewsWebController {
    private final PrincipalUtil principalUtil;
    private final NewsHandler newsHandler;
    private final CommentHandler commentHandler;
    public NewsWebController(NewsHandler newsHandler,
                             PrincipalUtil principalUtil,
                             CommentHandler commentHandler) {
        this.newsHandler = newsHandler;
        this.principalUtil = principalUtil;
        this.commentHandler = commentHandler;
    }

    @GetMapping("/{id}")
    public String newsById(@PathVariable @NotNull Integer id,
                           Principal principal,
                           Model model) {
        NewsViewWithComments newsView = newsHandler.handlerGetNewsById(id);
```

```
        model.addAttribute("news", newsView);
        model.addAttribute("user", principalUtil.getCurrentUser(principal));
        return "news";
    }

    @PostMapping("/comment/create")
    public String newsComment(@Valid @NotNull CommentReq req,
                              RedirectAttributes redirectAttributes){
        try {
            commentHandler.handlerCreateComment(req);
        } catch (IllegalArgumentException e) {
            redirectAttributes.addFlashAttribute("error", e.getMessage());
            return "redirect:/news/"  + req.newsId();
        }
        return "redirect:/news/"  + req.newsId();
    }
}
```

Файл ResultTableWebController.java

```
@Controller
@RequestMapping("/result-table")
public class ResultTableWebController {

    private final PrincipalUtil principalUtil;
    private final ResultHandler resultHandler;
    private final TournamentHandler tournamentHandler;

    public ResultTableWebController(PrincipalUtil principalUtil,
                                    ResultHandler resultHandler,
                                    TournamentHandler tournamentHandler) {
        this.principalUtil = principalUtil;
        this.resultHandler = resultHandler;
        this.tournamentHandler = tournamentHandler;
    }

    @ResponseStatus(HttpStatus.OK)
    @GetMapping
    public String result(Principal principal,
                         @RequestParam(required = false) Integer
tournamentId,
                         Model model) {
        // Получаем все турниры
        List<TournamentView> allTournaments =
tournamentHandler.handlerGetAllTournament();

        if (tournamentId != null) {
            List<ResultView> resultsByTournament =
resultHandler.handlerGetResultByTournament(tournamentId);
            model.addAttribute("resultsByTournament", resultsByTournament);
            model.addAttribute("selectedTournamentId", tournamentId);
        }

        model.addAttribute("allTournaments", allTournaments);
        model.addAttribute("user", principalUtil.getCurrentUser(principal));

        return "result";
    }
}
```

Файл SquadWebController.java

```
@Controller
@RequestMapping("/squad")
```

```
public class SquadWebController {
    private final PrincipalUtil principalUtil;
    private final TeamHandler teamHandler;
    private final StatisticsHandler statisticsHandler;
    private final PlayerHandler playerHandler;

    public SquadWebController(PrincipalUtil principalUtil,
                              TeamHandler teamHandler,
                              StatisticsHandler statisticsHandler,
                              PlayerHandler playerHandler) {
        this.principalUtil = principalUtil;
        this.teamHandler = teamHandler;
        this.statisticsHandler = statisticsHandler;
        this.playerHandler = playerHandler;
    }

    @ResponseStatus(HttpStatus.OK)
    @GetMapping
    public String squad(Principal principal,
                        Model model) {
        TeamView team = teamHandler.handlerGetClubTeam();
        List<PlayerView> playerViews =
playerHandler.handlerGetPlayerByTeamId(team.id());

        Map<String, List<PlayerView>> playersByPosition =
playerViews.stream()
                .collect(Collectors.groupingBy(PlayerView::position));

        List<String> positionOrder = Arrays.asList(
                "Вратарь",
                "Центральный защитник",
                "Правый защитник",
                "Левый защитник",
                "Полузащитник",
                "Нападающий");
        Map<String, List<PlayerView>> sortedPlayers = new LinkedHashMap<>();

        positionOrder.forEach(pos -> {
            if (playersByPosition.containsKey(pos)) {
                sortedPlayers.put(pos, playersByPosition.get(pos));
            }
        });

        playersByPosition.keySet().stream()
                .filter(pos -> !positionOrder.contains(pos))
                .forEach(pos -> sortedPlayers.put(pos,
playersByPosition.get(pos)));

        model.addAttribute("playersByPosition", sortedPlayers);
        model.addAttribute("team", team);
        model.addAttribute("user", principalUtil.getCurrentUser(principal));
        return "squad";
    }

    @GetMapping("/{id}")
    public String profileById(@PathVariable @NotNull Integer id,
                              Principal principal,
                              Model model) {
        PlayerView playerView = playerHandler.handlerGetPlayerById(id);
        List<StatisticsView> statisticsViews =
statisticsHandler.handlerGetStatisticsByPlayerId(id);

        model.addAttribute("player", playerView);
```

```
        model.addAttribute("statList", statisticsViews);
        model.addAttribute("user", principalUtil.getCurrentUser(principal));
        return "player-profile";
    }
```

Файл OpenApiConfig

```
@Configuration
public class OpenApiConfig {
    @Bean
    public OpenAPI customOpenAPI() {
        return new OpenAPI()
                .info(new Info()
                        .title("Football Club API")
                        .version("1.0")
                        .description("API для управления футбольным
клубом"));
    }
}
```

Файл WebConfig.java

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/static/**")
                .addResourceLocations("classpath:/static/");
    }

    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/login").setViewName("login");
    }
}
```

Файл WebSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {
    private final CustomUserDetailService userDetailService;

    public WebSecurityConfig(CustomUserDetailService userDetailService){
        this.userDetailService = userDetailService;
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
                .csrf(csrf -> csrf.disable())
                .authorizeHttpRequests((requests) -> requests
                        .requestMatchers("/admin/**").hasRole("ADMIN")
                        .requestMatchers("/news/comment/**").authenticated()
                        .anyRequest().permitAll()
                )
                .formLogin((form) -> form
                        .defaultSuccessUrl("/")
                        .loginPage("/auth/login")
                        .usernameParameter("email")
                        .passwordParameter("password")
                        .failureUrl("/auth/login?error=true")
                        .permitAll()
```

```
                )
                .logout((logout) -> logout
                        .logoutUrl("/logout")
                        .logoutSuccessUrl("/")
                        .invalidateHttpSession(true)
                        .deleteCookies("JSESSIONID")
                        .permitAll());
        return http.build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(8);
    }

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration authConfig) throws
Exception {
        return authConfig.getAuthenticationManager();
    }
}
```

Приложение В

(Обязательное)

Веб-приложение «Сайт футбольной команды ПГУ»

Презентация

Веб-приложение «Сайт футбольной команды ПГУ»

АВТОР РАБОТЫ:
СТУДЕНТ ГР. 21ВВП1
ХАБИБУЛИН АЙРАТ МАРАТОВИЧ
НАУЧНЫЙ РУКОВОДИТЕЛЬ:
К.Т.Н., ДОЦЕНТ КАФЕДРЫ ВТ ПГУ
ЮРОВА О.В.

Пенза 2025

АКТУАЛЬНОСТЬ ПРОБЛЕМЫ

- Цифровизация
- Укрепление имиджа университета
- Доступность
- Практическая значимость для студентов
- Развитие студенческого спорта и его популяризация
- Отсутствие единой информационной платформы для команды

ЦЕЛИ И ЗАДАЧИ

- Цель работы – разработать веб-приложение, позволяющее получать всю необходимую информацию о команде: расписание игр, новости, состав, турнирная таблица, результаты матчей, статистика игроков.
- Задачи:
  - Изучить предметную область;
  - Определить функционал;
  - Выбрать технологии и средства разработки приложения;
  - Разработать приложение.

ФУНКЦИОНАЛ

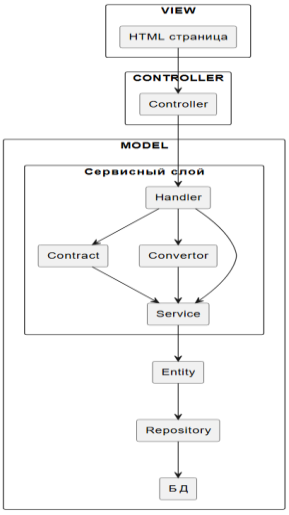
Приложение включает в себя:

- Регистрацию и авторизацию пользователей с разными правами доступа.
- Просмотр информации о команде: состав, расписание матчей, результаты, новости, статистика игроков.
- Возможность оставлять комментарии под новостями.
- Административную панель для управления контентом (добавление, редактирование, удаление данных).
- Личный кабинет пользователя.

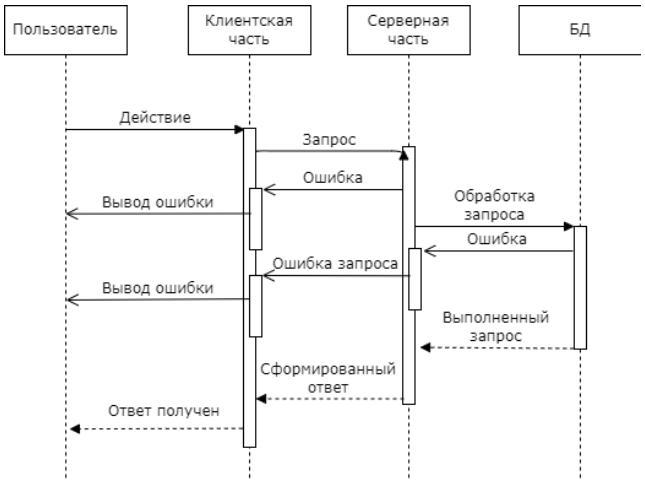
ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

АРХИТЕКТУРА ПРИЛОЖЕНИЯ

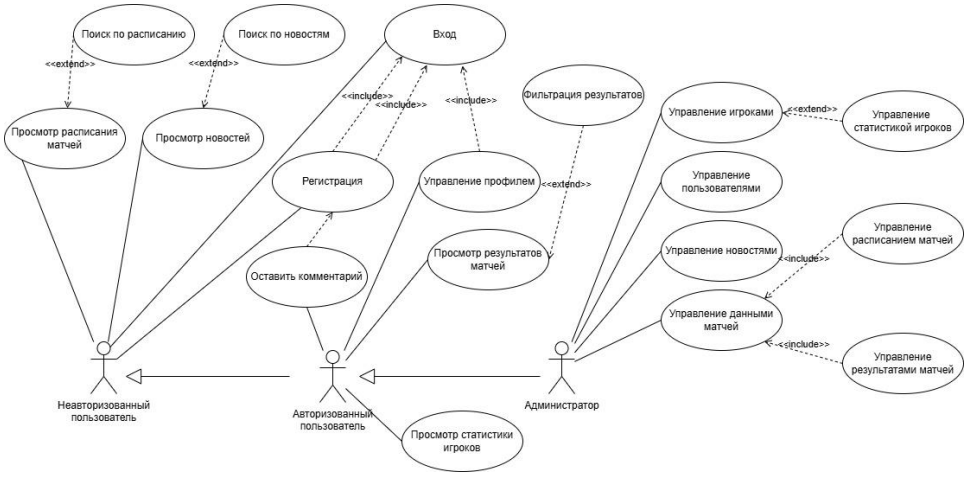
MVC - это способ разработки приложения, при котором само приложение делится на три компонента: модель (*Model*), представление (*View*), контроллер (*Controller*).



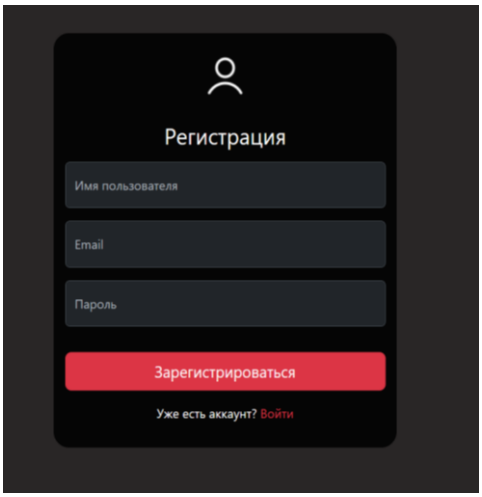
ПОСЛЕДОВАТЕЛЬНОСТЬ РАБОТЫ ПРИЛОЖЕНИЯ

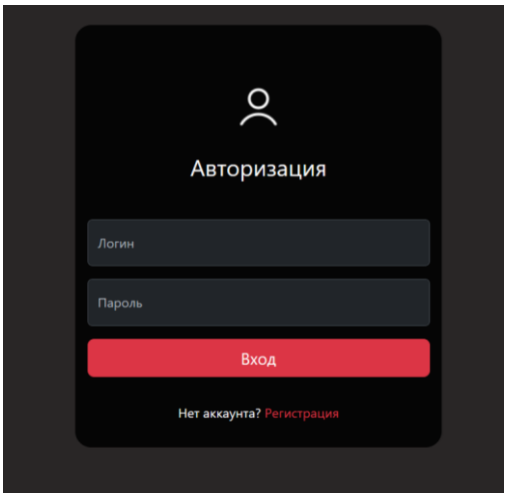


UML ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

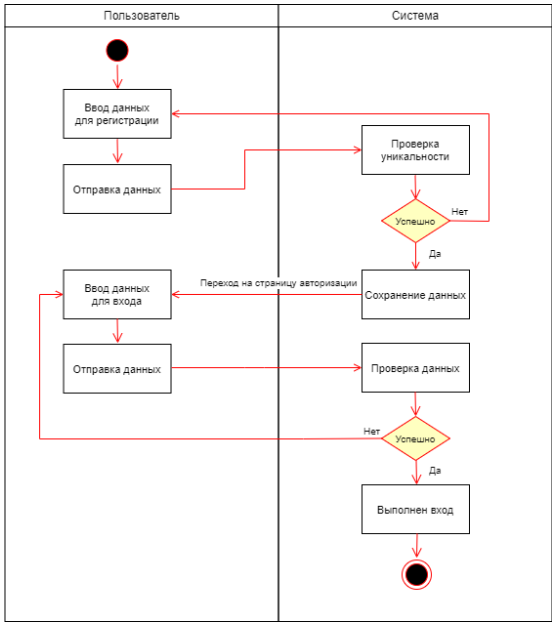


РЕГИСТРАЦИЯ И АВТОРИЗАЦИЯ

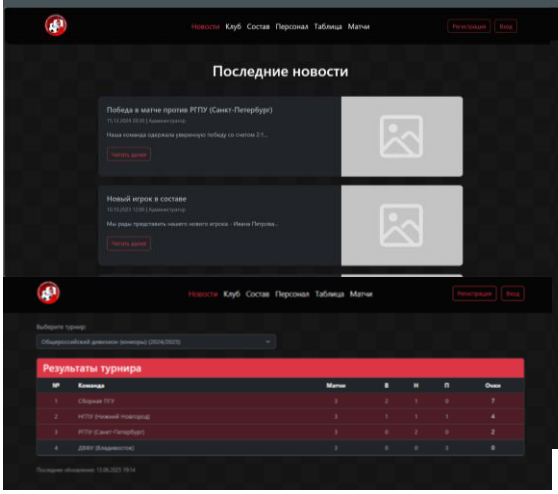


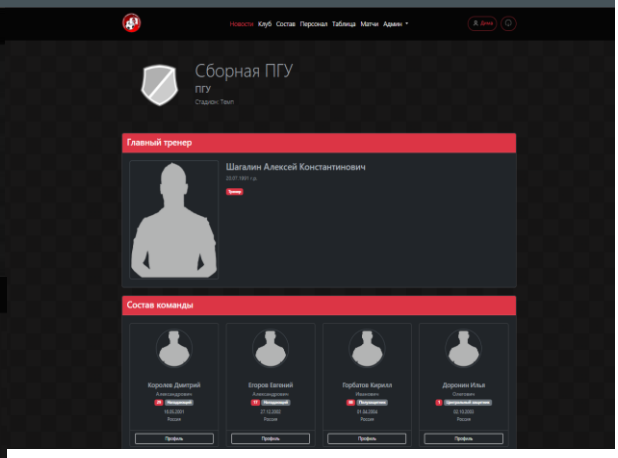


UML ДИАГРАММА ДЕЯТЕЛЬНОСТИ РЕГИСТРАЦИИ И АВТОРИЗАЦИИ

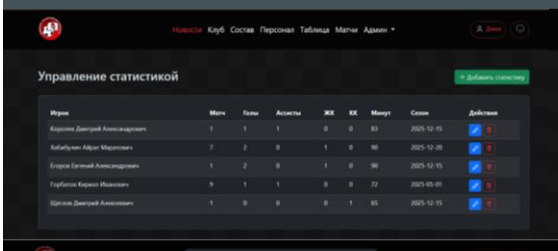


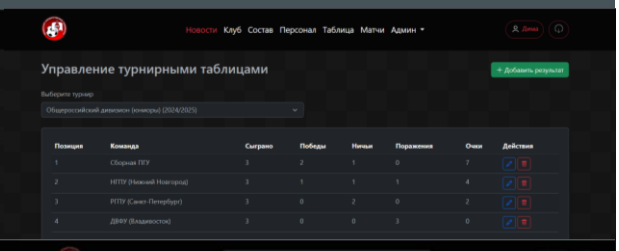
ИНТЕРФЕЙС ПРИЛОЖЕНИЯ

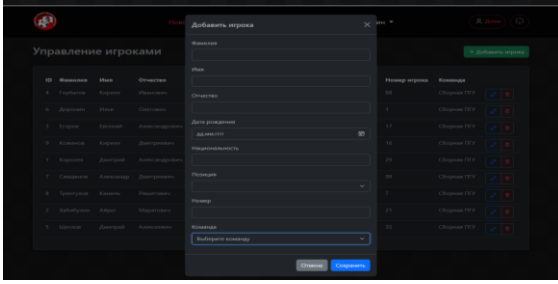


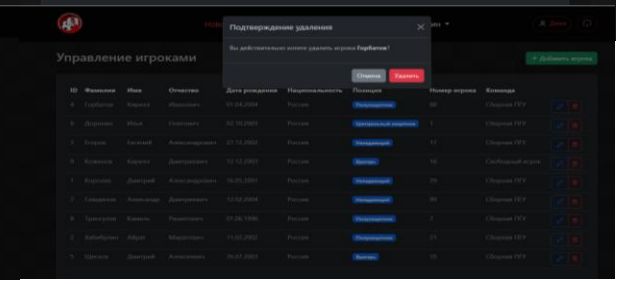


УПРАВЛЕНИЕ АДМИНИСТРАТОРОМ

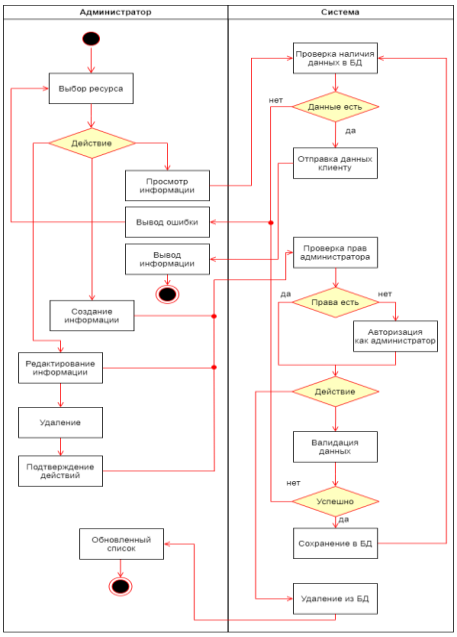








UML ДИАГРАММА ДЕЯТЕЛЬНОСТИ АДМИНИСТРАТОРА



ЗАКЛЮЧЕНИЕ

В рамках написания данной выпускной квалификационной работы были выполнены следующие задания: было проведено сравнение аналогов, анализ и выбор среды разработки, представлена архитектура приложения, выбрана база данных, разработано и протестировано веб – приложение «Сайт футбольной команды ПГУ».

Спасибо за внимание!