

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего
образования

«Пензенский государственный университет»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

ИНФОРМАЦИОННАЯ СИСТЕМА СБОРА И АНАЛИЗА ДАННЫХ О СПОРТИВНЫХ ДОСТИЖЕНИЯХ ОБУЧАЮЩИХСЯ ДЕТСКО- ЮНОШЕСКИХ СПОРТИВНЫХ ШКОЛ

Студент З 03.06.2024 Здерева Сергей Геннадьевич
(подпись, дата) (ФИО полностью)

Руководитель С 03.06.2024 Синев М. П.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр А 03.06.2024 Бычков А. С.
(подпись, дата) (фамилия, инициалы)

Рецензент М 03.06.2024 Мартышкин А.М., зав. кафедрой Программирования Мартышкин
(подпись, дата) (фамилия, инициалы) (должность, место работы)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 № 11)

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

Работа защищена с отметкой Отлично (протокол заседания ГЭК от 06.06.24 № 11)

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

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Пензенский государственный университет»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой

 М. А. Митрохин
«13» 09 2024 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРА

1. Студент Здерев Сергей Геннадьевич гр. 22ВВИм1 факультета вычислительной техники, направления 09.04.01.

2. Тема работы «Информационная система сбора и анализа данных о спортивных достижениях обучающихся детско-юношеских спортивных школ».

Тема утверждена приказом ПГУ № 765/о от "13" сентября 2022 г.

3. Руководитель магистерской диссертации к.т.н., доцент кафедры ВТ ПГУ Синев М. П.

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

Назначение разработки: информационной системы для сбора и анализа достижений спортсменов в детско-юношеском спорте.

Разработать автоматизированное клиент-серверное веб-приложение, позволяющее пользователям получать информацию о спортивных мероприятиях и статистику игроков на основе предсказания нейросетей.

5. Перечень подлежащих разработке вопросов:

1) Анализ системы оценочных критериев, шкалы оценок по каждому критерию, а также сбор данных для обучения.

- 2) Создание базы данных.
- 3) Разработка веб-приложения на основе микросерверной архитектуры.
- 4) Разработка нейронной сети, обучающую модель для дальнейшего использования в веб-приложения.
- 5) Тестирование разработанной информационной системы.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучить теоретические основы	6%	27.09.2022-15.11.2022	
2.	Анализ системы оценочных критериев, шкалы оценок по каждому критерию, а также сбор данных для обучения	6%	16.11.2022-22.03.2023	
3.	Разработать общую структуру базы данных	11%	23.03.2023-14.06.2023	
4.	Разработка серверной части приложения	16%	15.06.2023-10.09.2023	
5.	Разработка клиентской части приложения	15%	11.09.2023-05.01.2024	
6.	Разработка нейронной сети, обучающую модель для дальнейшего использования в веб-приложения	35%	06.01.2024-17.04.2024	
7.	Тестирование разработанной информационной системы.	4%	18.04.2024-9.05.2024	
8.	Оформление выпускной работы	7%	10.05.2024-28.05.2024	

Дата выдачи задания "13" сентября 2022 г.

Руководитель магистерской диссертации  Синева М. П.
(подпись, дата)

Задание к исполнению принял студент  Здерев С. Г.
(подпись, дата)

Работу к защите допустить
Декан факультета  04.06.2024 Фионова Л.П.
(подпись, дата)

РЕФЕРАТ

Работа содержит 109 листа, 17 рисунков, 4 таблиц, 10 источников, 3 приложения

АНАЛИЗ ДАННЫХ, АНАЛИЗ ДАННЫХ, БЕЗОПАСНОСТЬ ДАННЫХ, ИНФОРМАЦИОННАЯ СИСТЕМА, МАСШТАБИРУЕМОСТЬ, МАШИННОЕ ОБУЧЕНИЕ, НЕЙРОННЫЕ СЕТИ, ОБРАБОТКА ДАННЫХ, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, СУБД, СЕРВИС, СБОР ДАННЫХ, СПОРТИВНЫЕ ДОСТИЖЕНИЯ, ФРЕЙМВОРК

Цель ВКР – разработать Web приложение, позволяющее пользователям получать информацию о спортивных мероприятиях и статистику игроков на основе предсказания нейросетей.

Разработанный микросервис предназначен для ведения статистики по спортивным мероприятиям, учитывающим команды по различным видам спорта и персональную статистику игроков.

В процессе выполнения работы производилось:

- изучение особенностей разработки под web-приложения,
- разработка базы данных,
- разработка REST API,
- разработка клиентской части,
- разработка нейронной сети для предсказания статистики.

В результате выполнения работы было создано приложение, удовлетворяющее поставленной задаче.

					ПГУ1.090401.08.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Информационная система сбора и анализа данных о спортивных достижениях обучающихся детско-юношеских спортивных школ	Лит.	Лист	Листов	
Разраб.	59/126	59/126	30.06.20				2	109	
Провер.	59/126	59/126	30.06.20						
Н. Контр	Бычков	Бычков	30.06.20						
Утверд.	Митрофанов	Митрофанов	30.06.20						
						ФВТ, каф. ВТ			
						гр. 22ВВИм1			

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ	6
1 Теоретическая часть	8
1.1 Основные методы и подходы к анализу спортивных данных	8
1.2 Роли нейронных сетей в анализе спортивных данных	9
1.3 Постановка задачи	10
2 Анализ предметной области	12
2.1 Архитектура и инструменты разработки веб-сервиса	12
2.1.1 Описание архитектуры <i>REST API</i>	12
2.1.2 Выбор архитектуры приложения	13
2.1.3 Анализ типов СУБД	16
2.1.4 Выбор реляционной СУБД	17
2.1.5 Выбор языка программирования	18
2.1.6 Анализ фреймворков	19
2.1.7 Архитектурное решение	21
2.2 Выбор и анализ нейронной сети	22
2.2.1 Предобработка данных	25
2.2.2 Нормализация данных	26
2.2.3 Построение модели	27
2.2.4 Обучение модели	27
3 Реализация	29
3.1 Структура БД	29
3.2 Описание REST API	31
3.3 Архитектурная схема	33
3.4 Серверная реализация	33

3.4.1 UserDetailsService	33
3.4.2 SecurityFilterChain	34
3.4.3 AuthenticationProvider	35
3.4.4 PasswordEncoder	35
3.4.5 MyUserDetails	35
3.4.6 MyUserDetailService	36
3.5 Реализация клиента	38
3.5.1 Реализация создания мероприятия через вуб-сокеты	39
3.5.2 UML диаграмма взаимодействия	41
3.6 Программная реализация нейронной сети	43
4 Описание результата	48
4.1 Результаты обучения нейронной сети	52
4.1.1 История обучения	52
4.1.2 Кросс-валидация (MSE)	53
4.1.3 Распределение фактических и предсказанных значений	53
4.2 Выводы	54
ЗАКЛЮЧЕНИЕ	55
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	56
ПРИЛОЖЕНИЕ А Листинг программы	57
ПРИЛОЖЕНИЕ Б UML диаграммы	99
ПРИЛОЖЕНИЕ В Презентация	104

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей магистерской диссертации применяются следующие сокращения и обозначения

БД – база данных

ОС – операционная система

ПО – программное обеспечение

СУБД – система управления базой данных

ЯП – язык программирования

API (Application programming interface) – программный интерфейс приложения

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных

JSON (JavaScript Object Notation) – формат обмена данными, который основан на языке программирования JavaScript

NoSQL (not only SQL) – это семейство нереляционных баз данных

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределённого приложения в сети

SQL (Structured Query Language) – язык структурированных запросов

ВВЕДЕНИЕ

В современном мире информационные технологии играют все более важную роль во всех сферах жизни общества, включая спорт. Спортивная аналитика, основанная на сборе и анализе данных, становится все более важной для тренеров, спортсменов и спортивных организаций.

Целью данной работы является разработка веб-приложения «Информационная система сбора и анализа данных о спортивных достижениях обучающихся детско-юношеских спортивных школ». Задачами исследования являются сбор и обработка информации о спортивных достижениях учащихся, хранение данных о командах и спортсменах, а также обработка этих данных для получения полезной информации о достижениях спортсменов.

Объектом исследования являются спортивные достижения учащихся детско-юношеских спортивных школ. Предметом исследования является процесс обработки и анализа данных о спортивных достижениях с помощью разработанного веб-приложения и нейронной сети.

Основная функциональность заключается в сборе и обработке информации о спортивных достижениях учащихся детско-юношеских спортивных школ. Он позволяет хранить данные о командах и спортсменах, а также обрабатывать эти данные для получения предсказания статистики по результатам мероприятий.

В рамках работы над проектом был разработан код на языке Python, который включает в себя следующие этапы:

- загрузка и предобработка данных,
- разделение данных на обучающую, валидационную и тестовую выборки,
- нормализация данных,
- создание и обучение модели нейронной сети,
- оценка эффективности модели,
- применение модели для предсказания результатов на новых данных.

Так же был разработан веб-сервис на основе языка Java с использованием фреймворка Spring Framework.

Этот код является ключевым элементом системы и обеспечивает ее основную функциональность по хранению и анализу данных о спортивных достижениях.

1 Теоретическая часть

1.1 Основные методы и подходы к анализу спортивных данных

Анализ спортивных данных - это многоуровневый процесс, который включает в себя различные методы и подходы. Он начинается с сбора данных, которые могут включать в себя всё, от статистических данных о производительности до биометрических данных, собранных с помощью датчиков. Эти данные затем подвергаются процессу предварительной обработки, чтобы преобразовать их в формат, который может быть использован для анализа.

Статистический анализ - это один из самых основных методов анализа данных. Он включает в себя различные техники, такие как описательная статистика (среднее значение, медиана, стандартное отклонение и т.д.), корреляционный анализ, регрессионный анализ и т.д. В контексте спортивной аналитики, статистический анализ может быть использован для изучения различных аспектов производительности спортсменов или команд.

Машинное обучение - это подраздел искусственного интеллекта, который использует статистические техники для предсказания результатов на основе исторических данных. В контексте спортивной аналитики, машинное обучение может быть использовано для предсказания будущих результатов на основе прошлых данных. Например, можно использовать машинное обучение для предсказания вероятности победы команды в следующем матче на основе ее прошлых результатов.

Однако, с развитием технологий, особенно в области искусственного интеллекта и машинного обучения, появились новые подходы к анализу спортивных данных. Одним из таких подходов является использование нейронных сетей.

Нейронные сети - это модели машинного обучения, которые вдохновлены структурой и функционированием человеческого мозга. Они состоят из слоев нейронов, которые могут «обучаться» на основе данных и затем использовать

эту информацию для выполнения различных задач, таких как классификация, регрессия или кластеризация.

В моем коде, используется нейронная сеть для предсказания коэффициента статистики на основе различных признаков. Это позволяет не только анализировать текущие данные, но и делать прогнозы о будущих результатах.

Нейронные сети могут быть особенно полезны в спортивной аналитике, поскольку они могут обрабатывать большие объемы данных и улавливать сложные взаимосвязи между различными признаками. Они также могут адаптироваться к изменяющимся условиям, обучаясь на новых данных по мере их поступления.

В целом, нейронные сети представляют собой мощный инструмент для анализа спортивных данных, который может помочь тренерам, спортсменам и спортивным организациям получить ценные инсайты и преимущества.

1.2 Роли нейронных сетей в анализе спортивных данных

Нейронные сети играют ключевую роль в анализе спортивных мероприятий, обеспечивая глубокое понимание и проницательность в спортивные игры и соревнования. Вот некоторые из основных аспектов их роли:

- прогнозирование результатов. Нейронные сети могут анализировать большие объемы исторических данных о прошлых играх и соревнованиях для прогнозирования будущих результатов. Это может включать прогнозирование итогового счета, вероятности победы каждой команды и даже индивидуальной производительности игроков,

- анализ игры. Нейронные сети могут быть использованы для анализа тактики и стратегий, используемых в играх. Они могут распознавать и анализировать шаблоны игры, что может помочь тренерам и игрокам в разработке и применении эффективных стратегий,

- трекинг игроков и объектов. Нейронные сети могут быть использованы для отслеживания движения игроков и объектов (например, мяча) в реальном

времени. Это может помочь в анализе позиционирования игроков, их движений и взаимодействия с другими игроками и объектами,

– анализ здоровья и физической формы. Нейронные сети могут анализировать данные о здоровье и физической форме игроков, чтобы предсказать риск травм, определить оптимальные стратегии тренировок и помочь в восстановлении после травм.

В целом, нейронные сети играют важную роль в современном спортивном анализе, обеспечивая ценные инсайты и помогая улучшить производительность и результаты.

1.3 Постановка задачи

Веб-приложение должно быть максимально понятным и простым для восприятия. Это можно достичь, используя интуитивно понятный интерфейс и обеспечивая четкую навигацию.

В основе приложения должны лежать две основные роли: администратор и пользователь. Пользователи делятся на две категории: зарегистрированные, спортсмены. Зарегистрированные пользователи могут просматривать информацию на сайте, пользователи с ролью спортсмена могут оставлять комментарии и участвовать в обсуждениях, а организации могут публиковать новости о спортивных мероприятиях. Администраторы имеют более широкий спектр доступа и могут создавать события, и имеют полный доступ для администрирования данными

Разработать модель машинного обучения на основе нейронных сетей для анализа и прогнозирования спортивных достижений. Это включает загрузку и предобработку данных, преобразование категориальных признаков в числовые, разделение данных на обучающую, валидационную и тестовую выборки, а также нормализацию данных для улучшения качества обучения модели. Затем создание и обучение модели нейронной сети, используя раннюю остановку и сохраняя лучшую модель. После обучения нужно оценить качество модели на тестовой

выборке, сохранить модель и объекты предварительной обработки для дальнейшего использования, а API.

Приложение должно предоставлять полную и актуальную информацию о спортивных достижениях учащихся, а также статистику игроков. Это может включать информацию о прошедших и предстоящих мероприятиях, а также возможность отметки мероприятий.

2 Анализ предметной области

При разработке программных продуктов важно проанализировать имеющиеся варианты решения проблемы. В этой части обсуждаются общие теоретические вопросы, связанные с техническими элементами, необходимыми для внедрения программного обеспечения предприятия. К ним относятся выбор системы баз данных, архитектура программного обеспечения, язык программирования и архитектура API. В каждом подразделе обосновывается выбор того или иного технического решения проблемы.

2.1 Архитектура и инструменты разработки веб-сервиса

2.1.1 Описание архитектуры REST API

Архитектурный стиль REST предназначен для взаимодействия между компонентами распределенных веб-приложений. При разработке приложений, следующие пять обязательных ограничений играют ключевую роль:

1) модель клиент-сервер. Система разделяется на клиентов и серверы. Это ограничение основано на принципе ограничения запросов. Отделение клиентской и серверной частей упрощает расширение приложения и повышает переносимость кода между разными платформами. Например, микросервис, отвечающий за учет спортивных команд, действует в роли сервера,

2) отсутствие состояния. Взаимодействие между клиентом и сервером не хранит информацию о клиенте между запросами. Каждый клиентский запрос передает всю необходимую информацию об операции серверу до ее выполнения. В то же время клиент сохраняет информацию о состоянии сессии. Это ограничение обеспечивает масштабируемость системы, так как сервер не хранит состояние между запросами, что позволяет обрабатывать множество клиентов одновременно. Например, биллинговый микросервис для спортивных команд не регистрирует клиентские сессии и не отслеживает их статус,

3) кэширование. Клиенты могут кэшировать ответы сервера. Хорошо спроектированный кэш улучшает производительность и масштабируемость распределенной веб-системы, ограничивая связь между клиентом и сервером.

Например, если получены новые данные, ответы сервера кэшируются, а если данные уже были загружены, они не кэшируются,

4) многослойная система. В архитектуре REST система может быть разделена на иерархию уровней. Каждый компонент видит только следующий уровень компонентов. Использование временных агентов улучшает масштабируемость за счет балансировки нагрузки и совместного использования кэша. Это также позволяет применять политики безопасности для обеспечения целостности данных,

5) код по требованию. Это необязательное требование. В архитектуре REST оно позволяет загружать программы (код) на стороне клиента по мере необходимости.

2.1.2 Выбор архитектуры приложения

Существует два основных типа архитектуры для проектирования веб-приложения: монолитная и микросервисная.

Чтобы понять, какую архитектуру выбрать, нужно проанализировать особенности каждой.

Монолитная архитектура приложения.

Монолитная архитектура подразумевает такой подход к разработке приложения, когда приложение проектируется как один большой связанный модуль. Компоненты этого модуля взаимосвязаны между собой, имеют общую память и ресурсы. На рисунке 1 показано графическое представление монолитной архитектуры.



Рисунок 1 - Монолитная архитектура

Преимущества монолитной архитектуры:

- сравнительно простая реализация, развертывание и управление,
- проще поддерживать согласованность кода и обрабатывать ошибки,
- легко добавить типичный функционал к компонентам,
- достаточно высокая производительность для малых и средних приложений.

Но при кажущихся преимуществах монолитной архитектуре присущи достаточно серьезные недостатки:

- возрастающая сложность и запутанность приложения,
- с увеличением приложения все сложнее поддерживать изолированность сервисов, чтобы масштабировать каждый из них независимо или осуществлять поддержку кода,
- внесение изменений сопряжено с проблемами и занимает много времени,
- сбой одного сервиса приводит к сбою всего приложения.

Последний недостаток, при котором от сбоя в одном сервисе начинает сбоить всё приложение, является неприемлемым, если рассматривать его в контексте работы.

Микросервисная архитектура приложения.

Микросервисная архитектура — это подход, который помогает не только ускорить разработку продукта, но и сделать ее гибкой и управляемой: проект из неделимого целого превращается в систему связанных между собой блоков — сервисов. Впервые о микросервисах заговорили ещё в 2000-х, но концепция архитектуры сформировалась только к началу 2010-х. Давайте рассмотрим ключевые особенности микросервисной архитектуры:

- масштабируемость. Микросервисы могут быть масштабированы независимо друг от друга. Это позволяет распределять нагрузку и ресурсы по сервисам, которые нуждаются в поддержке прямо сейчас,

- независимость. Каждый сервис полностью автономен и не затрагивает работу соседей. Это означает, что каждый сервис может быть разработан, развернут и обновлен отдельно, без воздействия на остальные компоненты,
- легковесность. Микросервисы используют легковесные протоколы для взаимодействия между собой, такие как REST или gRPC. Это позволяет сервисам быстро обмениваться данными,
- гибкость: Микросервисы могут быть разработаны с использованием разных технологий и языков программирования. Это дает разработчикам большую гибкость при выборе технологий для конкретных компонентов системы. Это также позволяет эффективнее управлять командой разработки,
- управление ошибками. У каждого сервиса есть свое собственное управление ошибками и восстановлением после сбоев. Если один сервис не отвечает, это не приводит к полной остановке системы. Например, сайт продолжит принимать покупки, если микросервис, отвечающий за логистику, будет какое-то время недоступен,
- легкость развертывания. Микросервисы могут быть легко развернуты на различных серверах или облачных платформах,
- распределенная разработка. При построении микросервисной архитектуры разработка приложения может быть распределена между несколькими командами.

Выбор микросервисной архитектуры позволит вам создать гибкую, масштабируемую и устойчивую систему, которая легко адаптируется к изменениям и требованиям бизнеса.

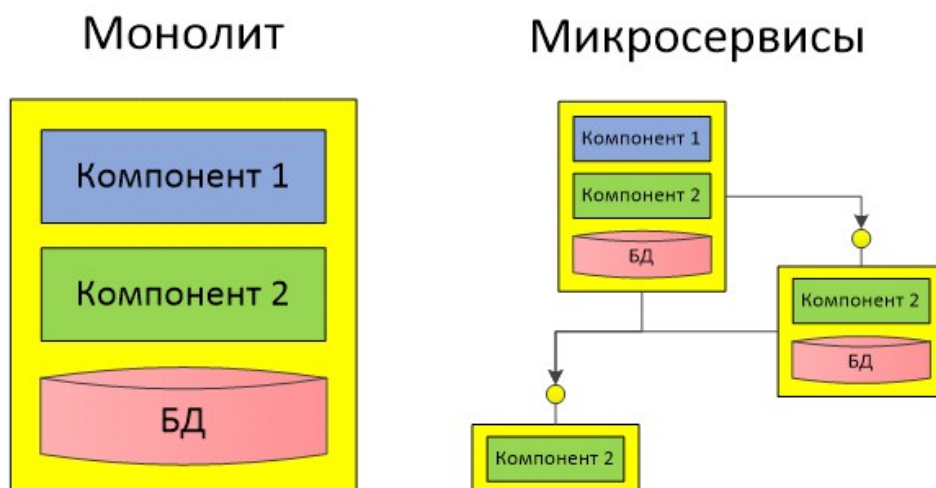


Рисунок 2 - Сравнение монолитной и микросервисной архитектур

Вывод: несмотря на сложность разработки и управления, микросервисная архитектура отлично подходит для реализации функционала микросервиса учёта спортивных команд.

2.1.3 Анализ типов СУБД

Существуют два вида СУБД: реляционные (SQL) и нереляционные (NoSQL). Чтобы выбрать тот или иной вид СУБД, нужно знать характерные отличия их друг от друга.

Существует шесть отличительных признаков, о каждом по подробней.

Разница в структуре и типе хранимых данных. В реляционных СУБД должна быть однозначно определена структура хранения данных, в то время как в NoSQL БД нет таких строгих ограничений, и они допускают хранение любого типа данных. Для микросервиса «Учет спортивных команд» подходит структура данных с однозначно определенными связями между объектами таблиц.

Различия в структуре запросов. В SQL БД данные могут быть получены только посредством языка SQL, так как реляционные СУБД соответствуют стандартам SQL. В нереляционных СУБД нет четкого определения, как должен выглядеть запрос и поэтому для каждой NoSQL БД реализуется свой способ работы с данными. Для микросервиса «Учет спортивных команд» требуется построение множества запросов с выборками по условию, и использование

нереляционной СУБД будет сопряжено с проблемами с созданием запросов, поэтому выбор реляционной СУБД для работы здесь очевиден.

Масштабируемость. Оба вида СУБД могут хорошо масштабироваться по вертикали, однако по горизонтали лучше растягиваются именно нереляционные СУБД. Для микросервиса «Учёт спортивных команд» больше подходит вариант с лучшим горизонтальным масштабированием.

Надежность. SQL базы данных являются более надежными, по сравнению с NoSQL БД. Поэтому реляционные БД более предпочтительны для выбора в качестве решения для микросервиса.

Поддержка. Реляционные СУБД появились гораздо раньше NoSQL, поэтому существуют уже готовые платные и бесплатные решения и поддержка, и найти ответ на свой вопрос гораздо проще, чем когда речь идет о сложном решении в нереляционной СУБД. И снова выбор в пользу SQL.

Хранение и доступ к сложным структурам данных. SQL базы данных выигрывают в производительности по сравнению с нереляционными СУБД, так как изначально предполагалось, что они будут работать именно со сложными структурами данных. Для микросервиса «Учет спортивных команд» снова больше подходит SQL БД.

Подводя итоги анализа возможностей реляционных и нереляционных СУБД, для микросервиса учёта спортивных команд выбор падает в пользу реляционной СУБД.

2.1.4 Выбор реляционной СУБД

PostgreSQL — это бесплатная система управления базами данных (СУБД) с открытым исходным кодом, и она является отличным выбором для вашей задачи. Вот несколько причин, почему PostgreSQL подходит именно для вашего проекта:

Свободный доступ. PostgreSQL можно бесплатно скачать и использовать. Это означает, что любой специалист может начать работу с базами данных без дополнительных затрат.

Если вы размещаете базу данных и СУБД в облачном хранилище, возможно, придется платить за хостинг, но сама PostgreSQL остается бесплатной.

Поддержка разных платформ. PostgreSQL можно установить на разные операционные системы: Linux, macOS, Windows и даже другие ОС.

Это удобно, так как пользователь получает готовую систему без необходимости устанавливать дополнительные инструменты.

Разнообразие форматов данных. PostgreSQL поддерживает множество типов данных, включая сетевые адреса, данные в формате JSON и геометрические данные для координат геопозиций.

Вы также можете создавать собственные пользовательские типы данных для упрощения работы с базой.

Продвинутые возможности. PostgreSQL предоставляет множество функций, таких как транзакции, хранимые процедуры, полнотекстовый поиск и многое другое.

Она активно развивается сообществом, что обеспечивает надежность и поддержку.

В целом, PostgreSQL — это надежная и масштабируемая СУБД, которая позволит эффективно управлять данными в проекте. Она идеально подходит для задач, связанной с хранением данных о мероприятиях, участниках и предсказаниями на основе нейросетей.

2.1.5 Выбор языка программирования

При реализации любого программного продукта одним из самых важных этапов является выбор языка программирования. Этот выбор влияет на продуктивность, чистоту кода и длительность создания минимально жизнеспособного продукта.

В настоящее время наиболее популярными языками программирования для разработки веб-приложений являются Java, JavaScript и Python. Давайте рассмотрим каждый из них:

Java — сильно типизированный объектно-ориентированный язык программирования. Он транслируется в байт-код, который выполняется на виртуальной машине Java (JVM). Это обеспечивает легкую переносимость программы на разные платформы.

Программы на Java обладают хорошей производительностью и широко используются в корпоративных приложениях. У Java большое сообщество разработчиков, хорошая документация и множество популярных фреймворков.

JavaScript — мультипарадигменный язык программирования, который используется в качестве языка сценариев для web-страниц. Он поддерживает императивный, декларативный и объектно-ориентированный стили программирования.

JavaScript часто используется в web-разработке, но его производительность и безопасность зависят от конкретной реализации интерпретатора.

Python — высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемости кода. Он гибкий и обладает широким функционалом.

Недостаток Python — низкое быстродействие, поэтому он часто используется как язык второго плана.

Выбор языка программирования, для реализации проекта, связанного с мероприятиями, участниками и предсказаниями на основе нейросетей, Java — отличный выбор. Программы на Java обладают хорошей производительностью, а также имеют большое сообщество разработчиков и множество популярных фреймворков.

2.1.6 Анализ фреймворков

Фреймворк — это нечто вроде плана, который разработчики используют для создания продукта. Давайте проведем аналогию с домом: если вы решите построить его самостоятельно, то придется купить материалы и своими руками возвести стены, уложить крышу и обустроить дренажную систему. Альтернативно, вы можете скачать готовую модель дома из Интернета. В этом

случае, возможно, придется заниматься отделкой, выбирать обои и мебель. Но, по крайней мере, фундамент, стены и крыша уже на месте. Точно так же фреймворк описывает, как строить приложение.

Spring Framework делает программирование быстрым, простым и эффективным. Он предоставляет готовые фрагменты кода, чтобы вам не пришлось писать все с нуля. Выбор конкретного фреймворка зависит от потребностей вашего проекта.

Универсальность:

- spring подходит практически для любых проектов. Он является фундаментом, на основе которого можно строить приложения разных типов,
- все привнесенные в него решения универсальны и применимы для широкой аудитории.

Легкость:

- важно, чтобы фреймворк Spring минимально влиял на логику проекта и содержал только те функции, которые действительно нужны,
- он облегчает разработку, позволяя сосредоточиться на уникальных функциях программы.

Интегрированность:

- spring содержит в себе другие популярные фреймворки, модули и библиотеки, такие как Struts, Hibernate, Tapestry, EJB и JSF,
- если вы освоите Spring, вы получите доступ к нескольким инструментам, которые уже настроены и связаны в единую систему.

Spring Framework полностью бесплатный, имеет открытый исходный код и подходит для написания кода на Kotlin и Groovy. Он идеально подходит для создания сложных систем, таких как многопользовательские корпоративные веб-приложения.

2.1.7 Архитектурное решение

Моя система представляет собой веб-приложение, построенное на Java с использованием Spring Framework (Spring Boot и Spring Security) для бэкенда и Vue 3 для фронтенда. Для обработки статистики используется отдельное приложение на Python.

Серверная часть.

Spring Boot. Этот фреймворк обеспечивает быстрое создание приложений с минимальной конфигурацией. Он предоставляет инструменты для создания REST API, обработки запросов, аутентификации и авторизации пользователей.

Spring Security: Для обеспечения безопасности приложения. Он управляет аутентификацией, авторизацией и защитой ресурсов.

Клиентская часть.

Vue 3. Этот современный фреймворк обеспечивает динамическую и отзывчивую веб-страницу. Он позволяет создавать компоненты, управлять состоянием и взаимодействовать с бэкендом через API.

Статистика.

Отдельное приложение на Python, которое обрабатывает данные о спортсменах и использует обученную модель “LinearRegression” для предсказания статистики.

Алгоритм приложения:

- создание мероприятия. организация создает мероприятие, на котором будут участвовать спортсмены.
- сбор данных. из мероприятия собираются данные по каждому спортсмену.
- отправка данных в нейронную сеть. собранные данные отправляются в нейронную сеть с обученной моделью “LinearRegression”.

- предсказание статистики. нейронная сеть выдает предсказание по каждому игроку, основанное на его текущей статистике и показателях в текущем мероприятии.

- отображение в профиле игрока. предсказанное число статистики отображается в профиле спортсмена в виде шкалы.

Преимущества архитектурного решения:

- структура и организация. Архитектурное решение помогает организовать компоненты системы и обеспечивает четкую структуру кода.

- Масштабируемость. Система легко масштабируется и адаптируется к изменяющимся требованиям.

- повторное использование и модульность. Архитектурное решение способствует повторному использованию кода и модульности системы.

- тестируемость и отладка. Создаваемая система будет легко тестируемой и отлаживаемой.

2.2 Выбор и анализ нейронной сети

Выбор и анализ нейронной сети представляют собой ключевые этапы в процессе разработки современных интеллектуальных систем. Правильный выбор архитектуры нейронной сети определяется рядом факторов, включая характер и объем данных, целевую задачу моделирования, а также вычислительные ресурсы, доступные для обучения и развертывания модели. Тщательный анализ производительности нейронной сети позволяет выявить её сильные и слабые стороны, а также оптимизировать параметры для достижения максимальной эффективности. Данные аспекты не только влияют на точность предсказаний, но и обеспечивают устойчивость и надежность системы при ее использовании в реальных условиях. Для наглядного понимания структуры нейронной сети полезно обратиться к визуализации. Пример простой нейронной сети можно увидеть на рисунке 3.

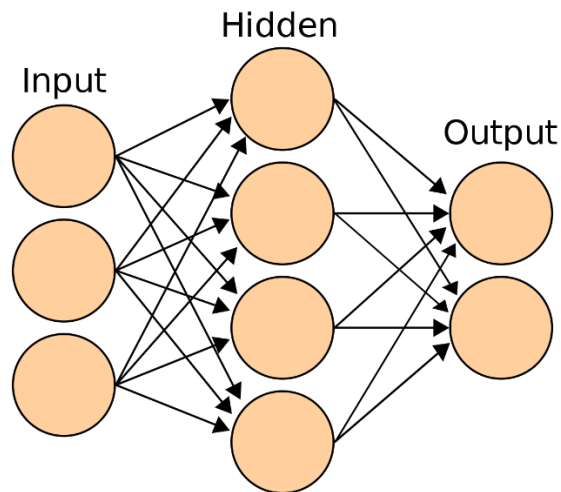


Рисунок 3 — Схема простой нейронной сети.

1. Сверточные нейронные сети (Convolutional Neural Networks, CNN): Эти сети обычно используются для обработки изображений, поскольку они способны улавливать пространственные зависимости в данных благодаря своей уникальной структуре. Однако, в данном случае, мы работаем с табличными данными, где такие зависимости отсутствуют, на рисунке 4 изображена схема сверточной нейронной сети.

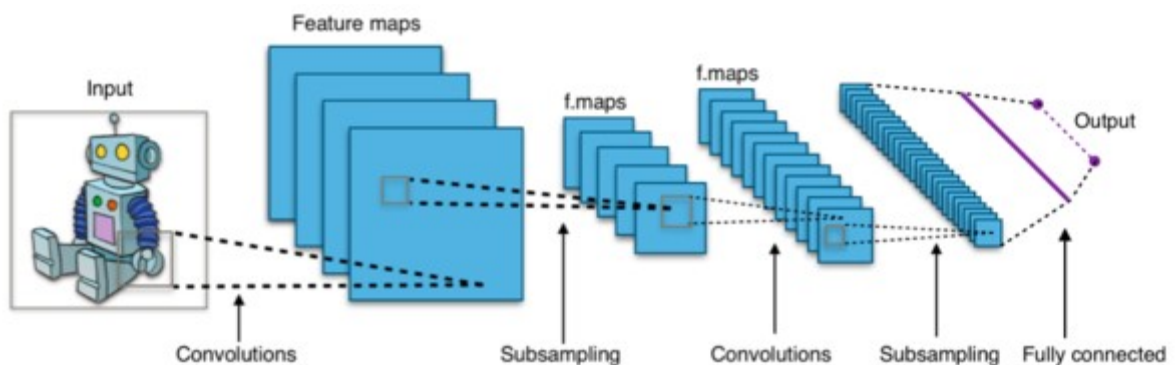


Рисунок 4 — Схема сверточной нейронной сети.

2. Рекуррентные нейронные сети (Recurrent Neural Networks, RNN): RNN хорошо подходят для обработки последовательностей данных, таких как текст или временные ряды. Но в нашем случае данные не имеют временной зависимости, что делает RNN менее подходящими, пример схемы можно увидеть на рисунке 5.

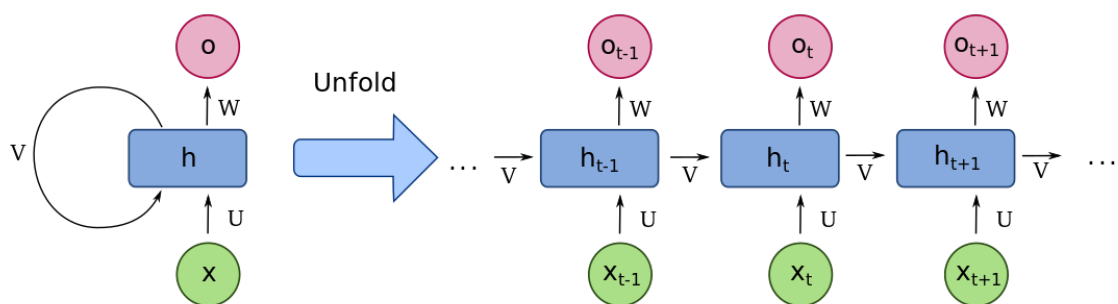


Рисунок 5 — Схема рекуррентные нейронные сети.

3. Сети с долгой краткосрочной памятью (Long Short-Term Memory networks, LSTM), схема изображена на рисунке 6: Это разновидность RNN, которая способна улавливать долгосрочные зависимости в данных. Опять же, поскольку наши данные не представляют собой последовательности, LSTM не являются оптимальным выбором.

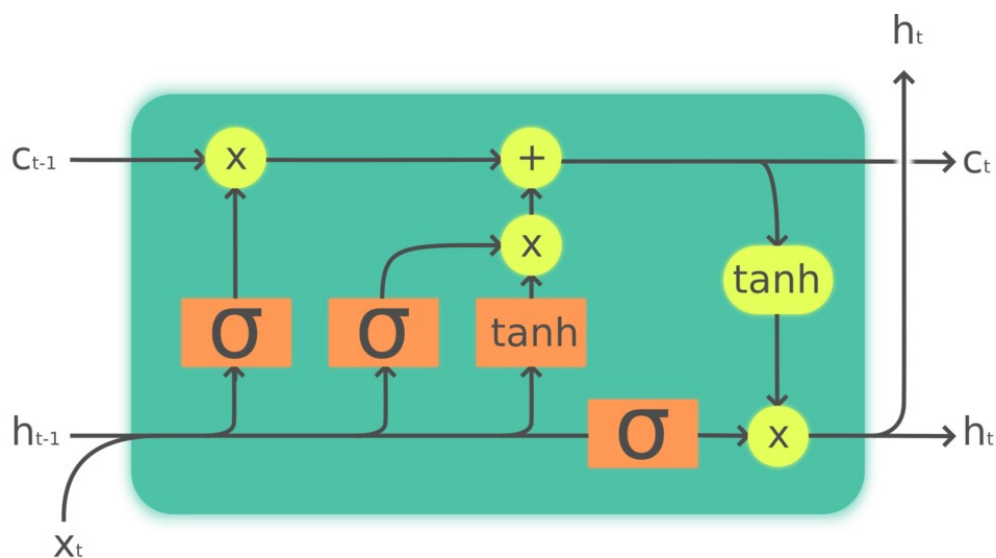


Рисунок 6 — Схема рекуррентные нейронные сети.

4. Сети прямого распространения (Feedforward Neural Networks, FNN): FNN - это самый базовый тип нейронных сетей. Они состоят из входного слоя, одного или нескольких скрытых слоев и выходного слоя. Все узлы связаны друг с другом без циклов или петель, в отличие от RNN, схема на рисунке 1.

В контексте задачи, была выбрана DNN, которая является разновидностью FNN, поскольку она хорошо подходит для работы с табличными данными. DNN способны улавливать сложные нелинейные зависимости в данных, что делает их мощным инструментом для этой задачи. Кроме того, благодаря возможности настройки количества слоев и узлов, DNN обеспечивают большую гибкость, позволяя мне создать модель, которая наилучшим образом соответствует нашим данным.

Моя модель основана на полносвязной нейронной сети (Fully Connected Neural Network), реализованной с помощью библиотеки Keras. Эта модель была выбрана из-за ее способности аппроксимировать любую функцию, что делает ее подходящей для широкого спектра задач. Кроме того, полносвязные сети являются относительно простыми в понимании и реализации, что является важным фактором для научной работы.

Весь процесс, начиная от загрузки и предобработки данных, разделения данных на обучающую, валидационную и тестовую выборки, создания и обучения модели, оценки модели и заканчивая предсказанием на новых данных, был реализован в этом коде. Это позволяет мне контролировать каждый этап процесса и гарантирует воспроизводимость результатов.

В заключение, я считаю, что выбор Python и полносвязной нейронной сети для этой работы был обоснован и целесообразен. Это позволило мне эффективно решить поставленную задачу и получить научно значимые результаты.

2.2.1 Предобработка данных

На начальном этапе исследования выполняется предварительная обработка данных, которая играет критически важную роль, так как качество входных данных напрямую влияет на производительность модели машинного обучения. Данные загружаются из CSV-файла с помощью библиотеки pandas, которая предоставляет структуры данных DataFrame для работы с двумерными таблицами различных типов данных. Функция `read_csv` позволяет быстро и эффективно загружать большие наборы данных. Pandas также включает

множество функций для очистки и предобработки данных, таких как `fillna`, используемая для заполнения пропущенных значений.

Для кодирования категориальных переменных применяется класс `LabelEncoder` из библиотеки `sklearn`. Он преобразует категориальные или текстовые данные в числовые значения, что позволяет моделям машинного обучения корректно работать с этими данными.

Для разделения данных на обучающую, валидационную и тестовую выборки используется метод `train_test_split` из библиотеки `sklearn`. Это помогает обеспечить корректное разделение данных и предотвратить переобучение модели.

2.2.2 Нормализация данных

На втором этапе проводится нормализация данных, что является процессом приведения данных к общему масштабу, позволяя алгоритмам машинного обучения работать более эффективно. Для нормализации данных также используется класс `StandardScaler` из библиотеки `sklearn`. `StandardScaler` преобразует данные так, что их распределение имеет среднее значение 0 и стандартное отклонение 1.

Процесс нормализации включает обучение `StandardScaler` на обучающих данных с использованием метода `fit`, который вычисляет среднее значение и стандартное отклонение каждого признака. Метод `transform` затем используется для преобразования как обучающих, так и тестовых данных с использованием этих параметров, что обеспечивает одинаковое преобразование данных и предотвращает утечку информации о тестовых данных во время обучения.

Все преобразования, примененные к обучающим данным, также должны быть применены к любым новым данным, используемым для предсказаний. Поэтому объект `StandardScaler` сохраняется для последующего использования, что обеспечивает консистентность и воспроизводимость процесса машинного обучения.

2.2.3 Построение модели

На третьем этапе выполняется построение модели. Этот процесс включает выбор архитектуры модели, определение функции потерь и оптимизатора, а также настройку параметров обучения. Для построения модели используется библиотека Keras, предоставляющая простой и удобный интерфейс для создания и обучения моделей глубокого обучения.

В данной работе выбрана полносвязная нейронная сеть (Dense Neural Network, DNN), состоящая из нескольких слоев, каждый из которых полностью соединен с узлами предыдущего слоя. Архитектура модели включает четыре слоя:

- 1) первый слой имеет 128 узлов и функцию активации ReLU,
- 2) второй слой имеет 64 узла и функцию активации ReLU,
- 3) третий слой имеет 32 узла и функцию активации ReLU,
- 4) последний слой имеет один узел с линейной функцией активации, что делает его подходящим для задач регрессии.

Следующим шагом является компиляция модели, где определяется функция потерь и оптимизатор. В качестве функции потерь выбрана среднеквадратичная ошибка (mean squared error), что стандартно для задач регрессии. В качестве оптимизатора используется Adam, эффективный и широко применяемый алгоритм оптимизации.

2.2.4 Обучение модели

Четвертым этапом является обучение модели, включающее подачу обучающих данных в модель, корректировку весов для минимизации функции потерь и проверку производительности на валидационных данных. Обучение модели осуществлено с использованием метода fit из Keras, при этом модель минимизирует функцию потерь, корректируя веса сети на основе обучающих данных и градиентов функции потерь. Обучение проводится в течение 100 эпох с размером пакета 32.

Использование валидационных данных помогает отслеживать, как хорошо модель обобщает данные и предотвращает переобучение.

В процессе обучения применяются обратные вызовы (callbacks) `EarlyStopping` и `ModelCheckpoint`. `EarlyStopping` автоматически прекращает обучение, когда валидационная ошибка перестает уменьшаться в течение 10 эпох, предотвращая переобучение и сокращая время обучения. `ModelCheckpoint` сохраняет модель в конце каждой эпохи, если она показывает лучший результат на валидационных данных, что гарантирует сохранение наилучшей модели.

3 Реализация

Текст первого раздела. Текст первого раздела. Текст первого раздела. Текст первого раздела. Текст первого раздела.

3.1 Структура БД

При разработке сервиса учёт спортивных команд была использована СУБД PostgreSQL. Далее представлено описание структуры разработанной БД. Таблица 1 содержит описание структуры таблицы «Пользователи (app_usr)».

Таблица 1 — Структура таблицы «Пользователи (app_usr)».

Название поля	Описание поля	Тип данных
<i>id</i>	Идентификатор игрока	<i>VarChar(255)</i>
<i>activity</i>	Подтверждение профиля	<i>VarChar(255)</i>
<i>email</i>	Электронная почта	<i>VarChar(255)</i>
<i>first_name</i>	Имя игрока	<i>VarChar(255)</i>
<i>last_name</i>	Фамилия игрока	<i>VarChar(255)</i>
<i>password</i>	Пароль	<i>VarChar(255)</i>
<i>role</i>	Роль	<i>VarChar(255)</i>
<i>age</i>	Возраст	<i>VarChar(255)</i>
<i>current_location</i>	Занятое место	<i>VarChar(255)</i>
<i>discharge</i>	Разряд	<i>VarChar(255)</i>
<i>height</i>	Рост	<i>VarChar(255)</i>
<i>number_of_competitions</i>	Количество соревнований	<i>VarChar(255)</i>
<i>position</i>	Позиция игрока (в футболе)	<i>VarChar(255)</i>
<i>previous_location</i>	Предыдущее место	<i>VarChar(255)</i>
<i>sport_type</i>	Вид спорта	<i>VarChar(255)</i>
<i>statistic</i>	Коэффициент статистики	<i>VarChar(255)</i>
<i>weight</i>	Вес	<i>VarChar(255)</i>

Таблица 2 содержит описание структуры таблицы «Карточки игрока (app_participants)».

Таблица 2 — Структура таблицы «карточка игрока(app_participants)».

Название поля	Описание поля	Тип данных
<i>message_id</i>	Внешний ключ созданного события	<i>VarChar(255)</i>
<i>email</i>	Почтовый ящик	<i>VarChar(255)</i>
<i>goals</i>	Количество голов	<i>VarChar(255)</i>
<i>position</i>	Занятое место учащимся	<i>VarChar(255)</i>
<i>team_place</i>	Занятое место командой	<i>VarChar(255)</i>

Таблица 3 содержит описание структуры таблицы «Мероприятие (app_message)».

Таблица 3 — Структура таблицы «Вид спорта (app_message)».

Название поля	Описание поля	Тип данных
<i>id</i>	Идентификатор вида спорта	<i>VarChar(255)</i>
<i>text</i>	Описание мероприятия	<i>VarChar(255)</i>
<i>creation_date</i>	Дата создания	<i>VarChar(255)</i>
<i>sport</i>	Вид спорта	<i>VarChar(255)</i>

Таблица 4 содержит описание структуры таблицы «Сесстия (app_spring session)».

Таблица 4 — Структура таблицы «История изменений (app_spring)».

Название поля	Описание поля	Тип данных
<i>session_id</i>	Идентификатор игрока	<i>VarChar(255)</i>
<i>creation_time</i>	Дата вступления	<i>VarChar(255)</i>
<i>last_access_time</i>	Дата конца	<i>VarChar(255)</i>
<i>Max_inactive_interval</i>	Наименование игрока	<i>VarChar(255)</i>

<i>sport_type_id</i>	Идентификатор вида спорта	<i>VarChar(255)</i>
<i>team_id</i>	Идентификатор команды	<i>VarChar(255)</i>

3.2 Описание REST API

Создание REST API является критическим этапом разработки микросервиса, поскольку качество и полнота его реализации напрямую влияют на удобство интеграции с другими микросервисами и на общую производительность приложения. В данном тексте подробно рассмотрены функции и спецификация REST API, разработанные для сервиса управления спортивными игроками и командами.

Разрабатываемое API соответствует принципам архитектурного стиля REST (Representational State Transfer), который базируется на стандартах HTTP. REST API обеспечивает возможность гибкого и масштабируемого обмена данными между клиентскими и серверными частями приложения с использованием стандартных HTTP-методов: GET, POST, PUT, DELETE и других.

Функциональные возможности API включают следующие операции.

Учащийся:

- создание новой карточки игрока,
- обновление существующей записи об игроке,
- удаление игрока,
- получение информации об игроке по идентификатору, дате вступления или имени,
- поиск игроков по критериям.

Мероприятие:

- создание нового мероприятия,
- обновление записи о мероприятии,
- удаление мероприятия,

- получение информации о мероприятии по идентификатору.

Для каждой операции определены специфические HTTP-методы и структуры URI. Основные URI запросов строятся по следующему шаблону:

Пример запроса на создание игрока.

Метод создания сущности (например, игрока) обычно реализуется с использованием HTTP-метода POST. Ниже приведен пример спецификации HTTP-запроса для создания игрока.

Метод: POST

```
URI: /player
Тело запроса:
{
  "firstData": "string",
  "id": 0,
  "lastData": "string",
  "name": "string",
  "sportTypeId": 0,
  "teamId": 0
}
```

В данном запросе тело содержит ключевые атрибуты, необходимые для создания новой записи об игроке:

"firstData": строка, представляющая часть данных об игроке;

"id": уникальный идентификатор (в момент создания может быть пустым или нулевым);

"lastData": строка, представляющая другую часть данных об игроке;

"name": имя игрока;

"sportTypeId": идентификатор типа спорта, к которому относится игрок;

"teamId": идентификатор команды, в которой играет игрок.

Ответ:

При успешном выполнении запроса сервер возвращает HTTP-код 200 и объект, представляющий созданного пользователя, включающий в себя все переданные поля, а также новые поля, добавленные после создания (например, актуальный идентификатор). Пример успешного ответа:

```
{
```

```

    "firstData": "string",
    "id": 0,
    "lastData": "string",
    "name": "string",
    "sportTypeId": 0,
    "teamId": 0
  }

```

В случае ошибки (например, при неверно сформированном запросе) возвращается HTTP-код 400 и сообщение об ошибке "Bad Request":

```

{
  "errorMessage": "string"
}

```

Подобная архитектура позволяет создавать легко читаемые и поддерживаемые интерфейсы, обеспечивающие устойчивость и масштабируемость системы. Использование REST API помогает обеспечить эффективное взаимодействие между различными компонентами программного обеспечения, что является основополагающим принципом разработки микросервисной архитектуры.

3.3 Архитектурная схема

Система использует архитектурный стиль клиент-сервер:

- backend реализован на Java Spring,
- frontend разработан на основе Vue.js.

Эти компоненты взаимодействуют через RESTful API, что обеспечивает гибкость и масштабируемость системы, а также упрощает интеграцию с другими сервисами.

3.4 Серверная реализация

3.4.1 UserDetailsService

В проекте используется пользовательская реализация интерфейса UserDetailsService, представленная классом MyUserDetailService. Это сервисный класс отвечает за загрузку данных пользователей при аутентификации. Метод userDetailsService() возвращает экземпляр MyUserDetailService.

```
@Service
```

```

public class MyUserDetailService implements UserDetailsService {
    @Autowired
    private UserRepository userRepository;
    @Override
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
        User user = userRepository.findByEmail(email);
        if (user == null) {
            throw new UsernameNotFoundException("User not
found");
        }
        return new MyUserDetails(user);
    }
    @Bean
    public UserDetailsService userDetailsService() {
        return new MyUserDetailService();
    }
}

```

3.4.2 SecurityFilterChain

Метод `securityFilterChain` конфигурирует основные аспекты безопасности HTTP-запросов. Выключение настроек CSRF осуществляется с помощью метода `AbstractHttpConfigurer::disable`. Аутентификация и авторизация обрабатываются через настройку `authorizeHttpRequests`, где определяется доступ к различным URL-шаблонам. Авторизация входа в систему осуществляется через форму (`formLogin`), а выход из системы организован с использованием `SecurityContextLogoutHandler`.

```

@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity
http) throws Exception {
        http
            .csrf(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(auth -> auth
                .antMatchers("/public/**").permitAll()
                .anyRequest().authenticated()
            )
            .formLogin(form -> form
                .loginPage("/login")
                .permitAll()
            )
            .logout(logout -> logout
                .logoutUrl("/logout")
            )
        }
    }
}

```

```

        .addLogoutHandler(new
SecurityContextLogoutHandler())
        .permitAll()
    );
    return http.build();
}
}

```

3.4.3 AuthenticationProvider

Метод `authenticationProvider` создаёт и настраивает провайдер аутентификации `DaoAuthenticationProvider`, который использует методы `userDetailsService()` для получения данных пользователя и `passwordEncoder()` для шифрования паролей.

```

@Configuration
public class SecurityConfig {
    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new
        DaoAuthenticationProvider();
        authProvider.setUserDetailsService(new
        MyUserDetailService());
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }
}

```

3.4.4 PasswordEncoder

Метод `passwordEncoder` возвращает экземпляр `BCryptPasswordEncoder`, который обеспечивает безопасность хранения паролей посредством их шифрования.

```

@Configuration
public class SecurityConfig {
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}

```

3.4.5 MyUserDetails

Класс `MyUserDetails` реализует интерфейс `UserDetails` и инкапсулирует данные пользователя. Он предоставляет методы для получения учетных данных пользователя и его ролей (авторизаций).

```

public class MyUserDetails implements UserDetails {
    private final User user;
}

```

```

    public MyUserDetails(User user) {
        this.user = user;
    }
    @Override
    public Collection<? extends GrantedAuthority>
getAuthorities() {
        return user.getRoles()
            .stream()
            .map(role -> new
SimpleGrantedAuthority(role.getName()))
            .collect(Collectors.toList());
    }
    @Override
    public String getPassword() {
        return user.getPassword();
    }
    @Override
    public String getUsername() {
        return user.getEmail();
    }
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    @Override
    public boolean isEnabled() {
        return user.isEnabled();
    }
}

```

3.4.6 MyUserDetailService

Класс `MyUserDetailService` реализует интерфейс `UserDetailsService` и переопределяет метод `loadUserByUsername`, который загружает пользователя из базы данных по его электронному адресу. В случае, если пользователь не найден, выбрасывается исключение `UsernameNotFoundException`.

```

@Service
public class MyUserDetailService implements UserDetailsService {
    @Autowired
    private UserRepository userRepository;
    @Override

```

```

    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
        User user = userRepository.findByEmail(email);
        if (user == null) {
            throw new UsernameNotFoundException("User not
found");
        }
        return new MyUserDetails(user);
    }
}

```

Эта интеграция различных компонентов, таких как `UserDetailsService`, `SecurityFilterChain`, `AuthenticationProvider`, и `PasswordEncoder`, обеспечивает высокую степень безопасности в веб-приложении, создавая структурированный и надежный механизм аутентификации и авторизации.

На рисунке 7 изображен полный цикл аутентификации пользователя в системе. Эта реализация Spring Security обеспечивает надёжный механизм аутентификации и авторизации, который использует шифрование паролей и гибкую настройку прав доступа к различным ресурсам веб-приложения.

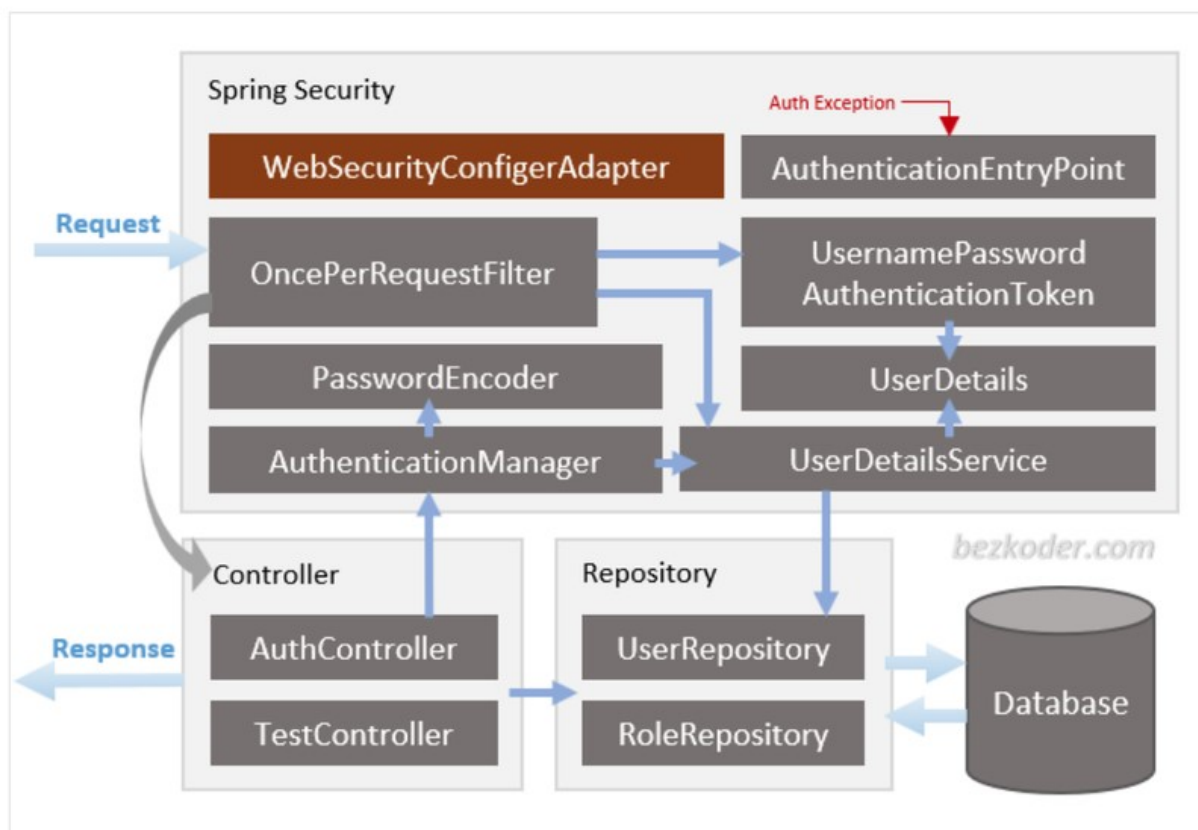


Рисунок 7 — Цикл безопасности безопасности Spring Security.

3.5 Реализация клиента

Для реализации клиентской части проекта были выбраны следующие инструменты.

Vue 3

В основу проекта лег Vue 3 — современный фреймворк для создания пользовательских интерфейсов. Главной его особенностью считается реактивная архитектура, которая значительно упрощает управление состоянием приложения и обновление интерфейса. С использованием Vue 3 можно легко определять компоненты и связывать их, применяя директивы вроде `v-model`, для реактивного управления состоянием:

```
<template>
  <div>
    <input v-model="message" placeholder="Введите сообщение" />
    <p>Вы ввели: {{ message }}</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      message: ''
    };
  }
};
</script>
```

Vuex

Для управления состоянием использован Vuex — централизованная система управления состоянием, которая упрощает хранение всех данных приложения в одном месте. Vuex обеспечивает предсказуемость управления состоянием, что позволяет компонентам быть единственными источниками истины для данных:

```
import { createStore } from 'vuex';
const store = createStore({
  state() {
    return {
      count: 0
    };
  }
});
```

```

    },
    mutations: {
      increment(state) {
        state.count++;
      }
    }
  });
export default store;
Vue Router.

```

Маршрутизация в одностраничном приложении реализована с использованием библиотеки Vue Router. Данная библиотека позволяет пользователям беспрепятственно перемещаться между различными разделами приложения без необходимости перезагрузки страницы. Пример настройки маршрутизации для двух компонентов представлен ниже:

```

import { createRouter, createWebHistory } from 'vue-router';
import Home from './components/Home.vue';
import About from './components/About.vue';

const routes = [
  { path: '/', component: Home },
  { path: '/about', component: About }
];

const router = createRouter({
  history: createWebHistory(),
  routes
});
export default router;

```

3.5.1 Реализация создания мероприятия через вуб-сокеты

Одной из ключевых технологических особенностей проекта стало использование вебсокетов для создания мероприятий. Вебсокеты предоставляют возможность поддержания постоянного двустороннего соединения между клиентом и сервером, что выгодно отличает этот протокол от традиционного HTTP, работающего по схеме запрос-ответ. Такой подход обеспечивает мгновенную передачу данных, существенно улучшая пользовательский опыт.

На серверной стороне был настроен вебсокет-сервер для приема и отправки данных клиенту в реальном времени:

```

const WebSocket = require('ws');
const wss = new WebSocket.Server({ port: 8080 });

```

```

wss.on('connection', (ws) => {
  ws.on('message', (message) => {
    console.log('received: %s', message);
    ws.send('Message received: ' + message);
  });

  ws.send('Welcome to WebSocket server');
});

```

На клиентской стороне использование вебсокетов также оказалось весьма удобным и естественным:

```

<template>
  <div>
    <form @submit.prevent="sendMessage">
      <input v-model="newMessage" placeholder="Введите
сообщение" />
      <button type="submit">Отправить</button>
    </form>
    <ul>
      <li v-for="message in messages" :key="message">{{ message
}}</li>
    </ul>
  </div>
</template>

<script>
const ws = new WebSocket('ws://localhost:8080');

export default {
  data() {
    return {
      newMessage: '',
      messages: []
    };
  },
  created() {
    ws.onmessage = (event) => {
      this.messages.push(event.data);
    };
  },
  methods: {
    sendMessage() {
      ws.send(this.newMessage);
      this.newMessage = '';
    }
  }
};
</script>

```

Этот проект ясно демонстрирует, как современные инструменты создания веб-приложений, такие как Vue 3, Vuex и Vue Router, могут быть использованы для создания структурированной и предсказуемой архитектуры приложения. Одновременно, внедрение вебсокетов обеспечивает высокую реактивность и быстроту обмена данными, что в свою очередь в разы повышает качество и удобство пользовательского опыта.

3.5.2 UML диаграмма взаимодействия

На UML-диаграмме последовательно демонстрируется взаимодействие различных классов в процессе обработки запроса на получение данных игрока по его уникальному идентификатору (ID). Данный процесс включается в несколько ключевых этапов, каждый из которых опирается на четко определенные обязанности и взаимодействия классов в многослойной архитектуре приложения, на рисунке 8 можно увидеть диаграмму взаимодействия.

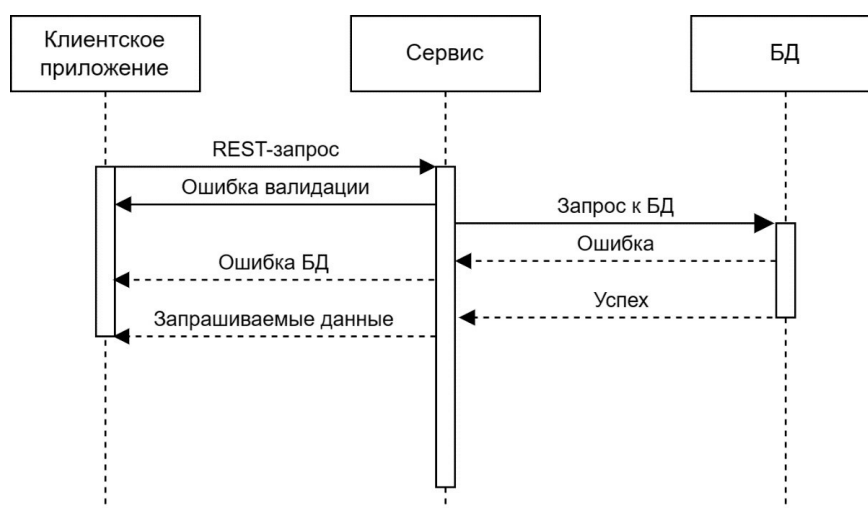


Рисунок 8 — UML диаграмма взаимодействия.

Контроллер

Первым звеном в последовательности является контроллер. Контроллер представляет собой компонент, который непосредственно взаимодействует с внешними запросами, полученными от клиента. При получении запроса на получение данных игрока по ID, он выполняет следующие шаги:

- получение запроса. Контроллер принимает HTTP-запрос, направленный на определенный URL-адрес, который соответствует ресурсу игрока,

- валидация запроса. Прежде чем перенаправить запрос дальше, контроллер валидирует параметры запроса. В данном случае это может быть проверка корректности и существования идентификатора игрока,

- формирование сервисного запроса. После успешной валидации, контроллер формирует запрос к соответствующему сервисному слою и передает ему необходимую информацию (ID игрока).

Сервисный слой

Следующим этапом является сервисный слой, который отвечает за бизнес-логику приложения. Он принимает запрос от контроллера и производит более сложные операции по обработке информации:

- обработка запроса. Сервисный слой осуществляет дополнительные проверки и вероятно подготавливает данные для выполнения запроса к репозиторию,

- взаимодействие с репозиторием. После подготовки данных, сервисный слой инициирует запрос к репозиторию. Здесь же могут быть приняты решения о выполнении каких-либо предварительных операций, таких как кэширование или валидация данных на уровне бизнес-логики.

Репозиторий

Репозиторий представляет собой слой доступа к данным. Он абстрагирует операции с базой данных и обеспечивает интерфейс для выполнения запросов и преобразования данных:

- запрос к базе данных. Получив запрос от сервисного слоя, репозиторий формирует SQL-запрос или использует методы ORM (Object-Relational Mapping) для взаимодействия с базой данных,

- преобразование данных. Данные, полученные из базы данных, преобразуются в объектную модель, предназначенную для использования в бизнес-логике. Это может включать составление сложных объектов из нескольких таблиц или агрегирование данных.

Обратная передача данных

Заключительным этапом является передача данных обратно через все слои к клиенту:

- возврат данных в сервисный слой, репозиторий возвращает данные в сервисный слой, который может выполнить дополнительные операции, такие как фильтрация или агрегация данных,
- формирование ответа. сервисный слой передает обработанные данные контроллеру,
- формирование HTTP-ответа. контроллер, получив данные от сервисного слоя, формирует окончательный HTTP-ответ в формате, требуемом клиентом (например, JSON), и отправляет его обратно клиенту.

Таким образом, последовательность взаимодействия классов на UML-диаграмме представляет комплексный и структурированный подход к обработке запроса на получение данных игрока по ID, обеспечивая четкое разграничение ответственности и улучшая модульность и поддерживаемость кода.

3.6 Программная реализация нейронной сети

Текст подзаголовка. Текст подзаголовка. Текст подзаголовка. Текст подзаголовка. Текст подзаголовка.

В данной работе для прогнозирования ключевых параметров спортсменов была разработана нейронная сеть с использованием библиотек TensorFlow и Scikit-learn. Процесс включает в себя несколько важнейших этапов, каждый из которых подробно освещается ниже.

Загрузка и предобработка данных

Процесс начинается с загрузки данных из файлов формата CSV. Это позволяет легко импортировать большие объемы данных для дальнейшей обработки.

```
warnings.filterwarnings('ignore')
# Загрузка данных
df = pd.read_csv('dataset.csv')
df['Количество голов'].fillna(0, inplace=True)
```

```
df['Количество соревнований'].fillna(0, inplace=True)
df['Вид спорта'].fillna('Unknown', inplace=True)
df['Позиция'].fillna('Unknown', inplace=True)
```

Основные шаги данного этапа включают:

- загрузка данных. Использование методов библиотеки pandas для чтения данных из CSV-файла,
- заполнение пропущенных значений. Применение методов среднего значения или наиболее вероятного значения (моды) для заполнения пропущенных данных, что позволяет избежать искажений в моделировании,
- кодирование категориальных признаков: Преобразование категориальных данных в числовой формат с использованием методов one-hot encoding или label encoding для корректного восприятия модели.

Разделение данных на выборки

Для обеспечения надежности и объективности модели данные разделяются на три основных выборки:

- обучающая выборка. Основная часть данных используется для обучения модели. Это обеспечивает её способность выявлять закономерности и зависимости в данных,
- валидационная выборка. Отдельная часть данных применяется для проверки производительности модели на этапе обучения. Она помогает настраивать гиперпараметры нейронной сети,
- тестовая выборка. Финальная часть данных, которая не используется на этапах обучения и валидации, предназначена для окончательной оценки способности модели делать прогнозы на новых данных.

```
# Разделение данных на обучающую, валидационную и тестовую
выборки
X_train, X_temp, y_train, y_temp = train_test_split(X, y,
test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)
```

Обработка признаков

Обработка признаков включает несколько этапов, направленных на нормализацию и подготовку данных:

- масштабирование числовых признаков. Применение стандартного масштабирования (StandardScaler) для приведения всех числовых данных к общему масштабу с средним значением 0 и стандартным отклонением 1. Это позволяет избежать переименования некоторых признаков на модель,

- преобразование категориальных признаков. Специфические методы обработки категориальных признаков включают их пропуск через слои нейронной сети для последующей интерпретации.

```
# Разделение данных на признаки и целевую переменную
X = df.drop('Коэффициент статистики', axis=1)
y = df['Коэффициент статистики']
```

Построение и обучение модели

Основной этап включает непосредственно разработку и обучение нейронной сети:

- архитектура модели. Использование последовательной модели (Sequential) из TensorFlow, включающей несколько плотных (Dense) слоев. Это позволяет создавать глубоко связную структуру, способную обучаться на сложных данных,

- меры против переобучения. Применение методов ранней остановки (EarlyStopping) и контрольной точки модели (ModelCheckpoint), которые помогают избежать переобучения модели и обеспечить её стабильность.

```
# Создание функции для построения модели
def build_model():
    model = Sequential()
    model.add(Dense(128, input_dim=X_train.shape[1],
activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

class KerasRegressorWrapper(BaseEstimator, RegressorMixin):
    def __init__(self, build_fn, **kwargs):
        self.build_fn = build_fn
        self.kwargs = kwargs
```

```

self.model = None

def fit(self, X, y, **fit_params):
    self.model = self.build_fn(**self.kwargs)
    self.model.fit(X, y, **fit_params)
    return self

def predict(self, X):
    return self.model.predict(X)

def score(self, X, y):
    return -self.model.evaluate(X, y, verbose=0)
keras_regressor = KerasRegressorWrapper(build_fn=build_model,
epochs=100, batch_size=32, verbose=0)
callbacks = [EarlyStopping(patience=10),
ModelCheckpoint('best_model.keras', save_best_only=True)]
model = build_model()
history = model.fit(X_train, y_train, validation_data=(X_val,
y_val), epochs=100, batch_size=32, callbacks=callbacks)

```

Оценка модели

Окончательным этапом является объективная оценка работоспособности и точности модели:

- оценка на валидационном наборе. Производится промежуточная оценка модели на валидационном наборе данных для настройки гиперпараметров и проверки устойчивости модели,
- финальная оценка на тестовом наборе. Итоговая проверка модели на тестовом наборе для определения её точности и возможности использования в реальных условиях.

```

loss = model.evaluate(X_test, y_test)
print(f'Оценка модели на тестовых данных - Потери (Loss): {loss}')

# Кросс-валидация для проверки стабильности модели
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
cross_val_results = cross_val_score(keras_regressor, X_train,
y_train, cv=kfold, scoring='neg_mean_squared_error')
print(f'Средняя ошибка по кросс-валидации (MSE): {-cross_val_results.mean()}')

# Сохранение модели в формате H5 и объектов предварительной
обработки
model.save('final_model.h5')
dump({'preprocessor': preprocessor, 'label_encoder_sport':

```

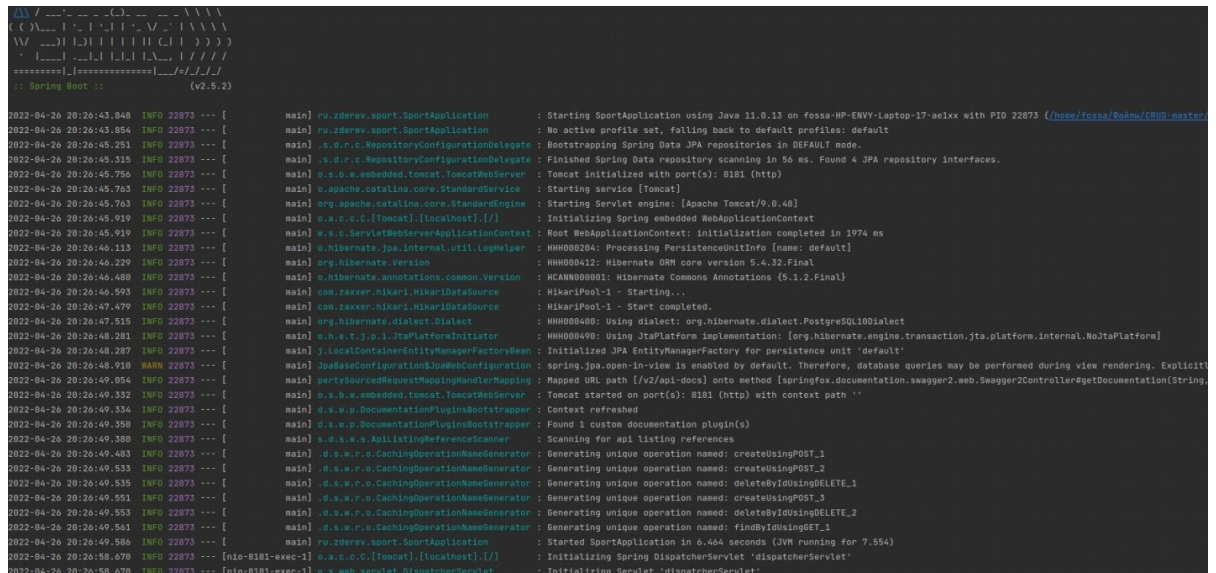
```
label_encoder_sport, 'label_encoder_position':  
label_encoder_position}, 'preprocessing_objects.joblib')
```

Эти шаги обеспечивают создание и оптимизацию нейронной сети для анализа спортивных данных, что позволяет эффективно предсказывать ключевые показатели и принимать обоснованные решения на основе полученных результатов.

4 Описание результата

Текст первого раздела. Текст первого раздела. Текст первого раздела. Текст первого раздела. Текст первого раздела.

В результате компиляции программы был получен исполняемый файл «SEO.jar». Для его запуска можно использовать команду «Java -jar SEO.jar». Результат запуска представлен на рисунке 9.



```
2022-04-26 20:26:43.848 INFO 22873 --- [main] ru.zdev.sport.SportApplication : Starting SportApplication using Java 11.0.13 on fossa-WP-ENVY-Laptop-17-aelix with PID 22873 (C:\home\foossa\Game\CRM-master)
2022-04-26 20:26:43.854 INFO 22873 --- [main] ru.zdev.sport.SportApplication : No active profile set, falling back to default profiles: default
2022-04-26 20:26:45.261 INFO 22873 --- [main] org.springframework.data.jpa.repository.config.JpaRepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-04-26 20:26:45.315 INFO 22873 --- [main] org.springframework.data.jpa.repository.config.JpaRepositoryConfigurationDelegate : Finished Spring Data repository scanning in 56 ms. Found 4 JPA repository interfaces.
2022-04-26 20:26:45.756 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Tomcat initialized with port(s): 8181 (http)
2022-04-26 20:26:45.763 INFO 22873 --- [main] org.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-04-26 20:26:45.763 INFO 22873 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.48]
2022-04-26 20:26:45.919 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Initializing Spring embedded WebApplicationContext
2022-04-26 20:26:45.919 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Root WebApplicationContext: initialization completed in 1974 ms
2022-04-26 20:26:46.113 INFO 22873 --- [main] org.hibernate.jpa.internal.util.LogHelper : HH0000204: Processing PersistenceUnitInfo [name: default]
2022-04-26 20:26:46.229 INFO 22873 --- [main] org.hibernate.Version : HH0000412: Hibernate ORM core version 5.4.32.Final
2022-04-26 20:26:46.480 INFO 22873 --- [main] org.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations (5.1.2.Final)
2022-04-26 20:26:46.593 INFO 22873 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-04-26 20:26:47.499 INFO 22873 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-04-26 20:26:47.515 INFO 22873 --- [main] org.hibernate.dialect.Dialect : HH0000480: Using dialect: org.hibernate.dialect.PostgreSQL10Dialect
2022-04-26 20:26:48.281 INFO 22873 --- [main] org.springframework.orm.jpa.JpaPlatformInitiator : HH0000499: Using JpaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-04-26 20:26:48.287 INFO 22873 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-04-26 20:26:48.910 WARN 22873 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure open-in-view with either 'spring.jpa.open-in-view=false' or 'springfox.documentation.swagger2.web.swagger2.serve=false'
2022-04-26 20:26:49.054 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Mapped URL path [/v2/api-docs] onto method [springfox.documentation.swagger2.web.swagger2.serve]
2022-04-26 20:26:49.332 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Tomcat started on port(s): 8181 (http) with context path ''
2022-04-26 20:26:49.350 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Context refreshed
2022-04-26 20:26:49.350 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Found 1 custom documentation plugin(s)
2022-04-26 20:26:49.380 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Scanning for api listing references
2022-04-26 20:26:49.483 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Generating unique operation named: createUsingPOST_1
2022-04-26 20:26:49.533 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Generating unique operation named: deleteUsingDELETE_1
2022-04-26 20:26:49.535 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Generating unique operation named: createUsingPOST_2
2022-04-26 20:26:49.551 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Generating unique operation named: deleteUsingDELETE_2
2022-04-26 20:26:49.553 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Generating unique operation named: deleteUsingDELETE_3
2022-04-26 20:26:49.561 INFO 22873 --- [main] org.springframework.web.servlet.support.SpringServlet : Generating unique operation named: findUsingGET_1
2022-04-26 20:26:49.586 INFO 22873 --- [main] ru.zdev.sport.SportApplication : Started SportApplication in 6.464 seconds (JVM running for 7.554s)
2022-04-26 20:26:58.678 INFO 22873 --- [nio-8181-exec-1] org.springframework.web.servlet.support.SpringServlet : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-04-26 20:26:58.678 INFO 22873 --- [nio-8181-exec-1] org.springframework.web.servlet.support.SpringServlet : Initializing Servlet 'dispatcherServlet'
```

Рисунок 9 — Результат работы исполняемого файла «sport.jar» сразу после запуска

После запуска приложения нужно собрать фронттовую часть, для сборки используется уарн, так как его преимущество заключается в скорости, так же реализован Hot reload, для быстрого перезапуска при каких-либо изменениях клиентского кода, на рисунке 10 изображен результат сборки.

```
cli> [webpack-dev-server] Project is running at:
cli> [webpack-dev-server] Loopback: http://localhost:8080/
cli> [webpack-dev-server] On Your Network (IPv4): http://192.168.0.155:8080/
cli> [webpack-dev-server] Content not from webpack is served from 'C:\Users\serge\IdeaProjects\SEO\dist' directory
assets by status 1.25 MiB [cached] 1 asset
assets by info 3.44 MiB [immutable]
  asset 2146c3c82b553977abc7.eot 1.25 MiB [emitted] [immutable] [from: node_modules/@mdi/font/fonts/materialdesignicons-webfont.eot] (auxiliary name: main)
  asset 1fcc36272ea3e53d0031.ttf?v=7.4.47 1.25 MiB [emitted] [immutable] [from: node_modules/@mdi/font/fonts/materialdesignicons-webfont.ttf?v=7.4.47] (auxiliary name: main)
  asset e55a20c80650829ec5fd.woff?v=7.4.47 574 KiB [emitted] [immutable] [from: node_modules/@mdi/font/fonts/materialdesignicons-webfont.woff?v=7.4.47] (auxiliary name: main)
  asset ec024da79d2972da002.woff?v=7.4.47 394 KiB [emitted] [immutable] [from: node_modules/@mdi/font/fonts/materialdesignicons-webfont.woff?v=7.4.47] (auxiliary name: main)
asset main.js 6.5 MiB [emitted] (name: main) 1 related asset
orphan modules 2.37 MiB [orphan] 17 modules
runtime modules 27.4 KiB 12 modules
modules by path ./node_modules/ 5.02 MiB (javascript) 4.69 MiB (asset) 972 modules
modules by path ./src/main/resources/static/js/ 110 KiB
  modules by path ./src/main/resources/static/js/pages/*.vue 50.7 KiB
    ./src/main/resources/static/js/pages/App.vue 767 bytes [built] [code generated]
  modules by path ./src/main/resources/static/js/components/messages/*.vue 40.8 KiB
    ./src/main/resources/static/js/components/messages/MessageRow.vue 824 bytes [built] [code generated]
    + 12 modules
    ./src/main/resources/static/js/main.js 10.1 KiB [built] [code generated]
    ./src/main/resources/static/js/util/ws.js 1.13 KiB [built] [code generated]
    ./src/main/resources/static/js/vuetify/vuetify.js 382 bytes [built] [code generated]
  modules by path ./src/main/resources/static/js/components/messages/*.vue 40.8 KiB
    ./src/main/resources/static/js/components/messages/MessageRow.vue 824 bytes [built] [code generated]
    + 12 modules
    ./src/main/resources/static/js/main.js 10.1 KiB [built] [code generated]
    ./src/main/resources/static/js/util/ws.js 1.13 KiB [built] [code generated]
    ./src/main/resources/static/js/vuetify/vuetify.js 382 bytes [built] [code generated]
    ./src/main/resources/static/js/route/router.js 467 bytes [built] [code generated]
data:image/png;base64,iVBORw0KGgoAAAAN... 178 bytes [built] [code generated]
webpack 5.91.0 compiled successfully in 7121 ms
assets by status 11.2 MiB [cached] 6 assets
cached modules 5.13 MiB (javascript) 4.69 MiB (asset) 27.4 KiB (runtime) [cached] 1039 modules
webpack 5.91.0 compiled successfully in 336 ms
```

Рисунок 10 — Результат работы fronted составляющей.

После запуска можно обратиться на локальный хост по порту 8081, после чего вы будете перенаправлены на страничку авторизации. Сама аутентификация реализован с помощью Spring Security, что позволяет создать защищённую среду для пользователей и администраторов.

Если вы аутентифицированы, то по общению к локальному хосту произойдет редирект на страницу авторизации. На рисунке 11 изображено окно авторизации.

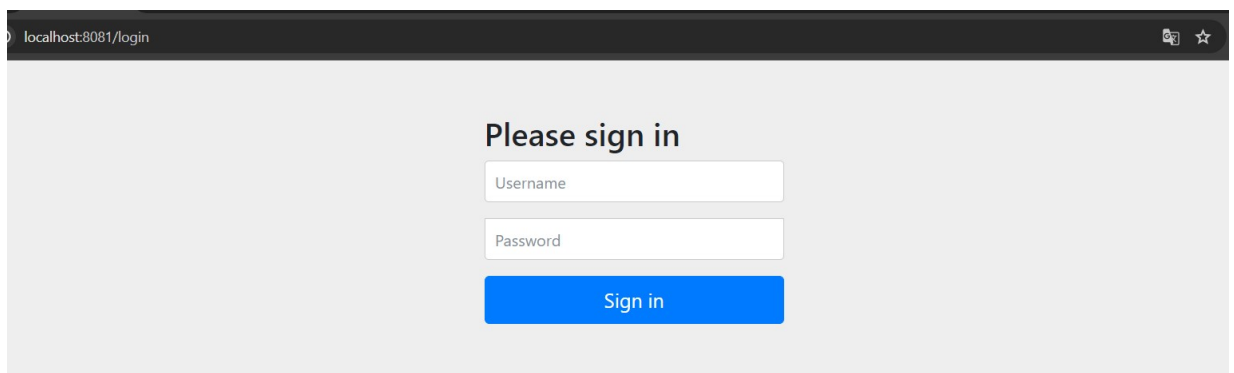


Рисунок 11 — Результат работы fronted составляющей.

После успешной авторизации мы попадаем на главную страницу, на которой изображен главный тулз, в котором указано название информационно системы, название профиля, кнопка выхода из профиля. В мейн части

изображена форма создания мероприятия и мессаджи. На рисунке 12 изображена главная страница.

The screenshot shows a web application interface. At the top, there is a header bar with the text 'ИНФОРМАЦИОННАЯ СИСТЕМА СБОРА И АНАЛИЗА ДАННЫХ' on the left and 'ADMIN ADMIN' with a share icon on the right. Below the header, the main content area has a light gray background. It features a form for creating a new event. The form includes a text input field labeled 'Новое событие', a dropdown menu labeled 'Выберите вид спорта', and a blue button labeled 'СОЗДАТЬ'. Below the form, there is a list of three items, each in a white box with a light gray border. The first item is '(1602) тест' with buttons 'ОТРЕДАКТИРОВАТЬ' and 'УДАЛИТЬ'. The second item is '(1553) мивсмичсмис' with the same buttons. The third item is '(1552) dfaadfsad' and is partially visible.

Рисунок 12 — Графическое отображение главной страницы.

Если выбрать вид спорта то появится возможность заполнить карточки игроков, участвующих в мероприятии, увидеть это можно на рисунке 13. Колличество участников посчитывает на стороне клиента, после заполнения всех данных мы можем создать мероприате. Данные уходят на сервер и с помощью библиотеки собираются и подаются на вход нейронной сети, она в свою очередь делает предсказание коэффециента по каждому участнику и обновляет его в профиле пользователя.

Рисунок 13 — Создание карточки игрока в мероприятии.

Если перейти в профиль пользователя, нажав на имя в дев тулзе, то мы перейдем в профиль пользователя, в личном кабинете можно увидеть персональные данные спортсмена и шкалу его предсказания, интерфейс изображен на рисунке 14.

Так же если пользователь просто зарегистрировался он может подтвердить свои данные в профиле, для этого нужно заполнить данные в личном кабинете и отправить их на сервер, тем самым профиль становится подтвержденным и пользователь может участвовать в мероприятиях. На рисунке 15 можно увидеть страничку с полями ввода для неаутентифицированным пользователем.

Рисунок 14 — Страничка профиля пользователя.

ИНФОРМАЦИОННАЯ СИСТЕМА СБОРА И АНАЛИЗА ДАННЫХ

ADMIN ADMIN

Профиль пользователя

Имя

admin

Фамилия

admin

Email

admin

Роль

ADMIN

Вид спорта

Вес

Рост

Разряд

Рисунок 15 — Страничка неподтвержденного.

4.1 Результаты обучения нейронной сети

В рамках данного проекта была разработана нейронная сеть для прогнозирования ключевых параметров спортсменов. Следующие графики и метрики позволяют наглядно представить результаты обучения и производительности модели.

4.1.1 История обучения

График потерь: На графике визуализированы потери (loss) на обучающей и валидационной выборках по мере прогрессирования эпох, на рисунке 15 изображен график потерь.

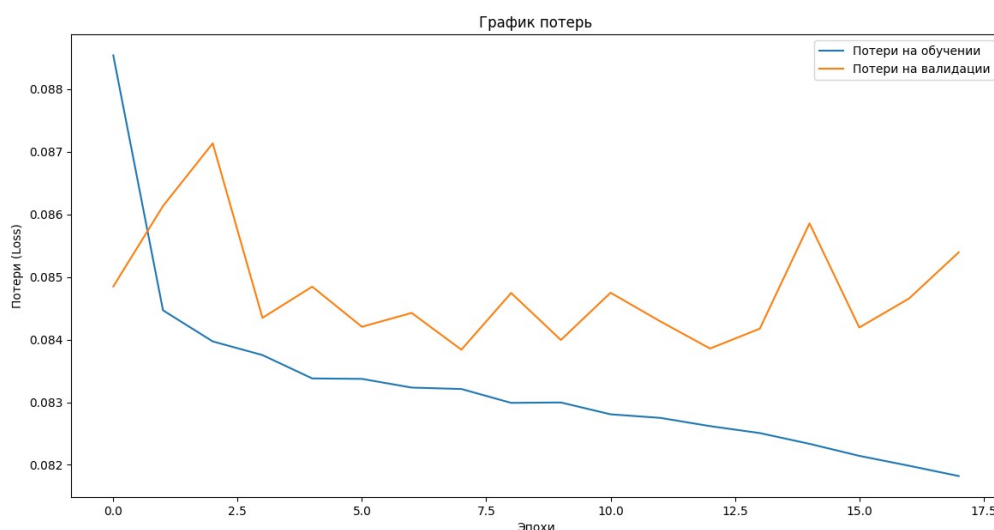


Рисунок 15 — График потерь.

Данный график позволяет отслеживать динамику обучения и помогает выявить возможные проблемы переобучения или недообучения модели.

4.1.2 Кросс-валидация (MSE)

- для оценки стабильности и обобщающей способности модели использовалась кросс-валидация с 5 фолдами,
- среднеквадратичная ошибка (MSE) по кросс-валидационным фолдам представлена в виде боксплота, что наглядно показывает распределение ошибки и наличие выбросов.

В результате обучения среднеквадратичная ошибка была 0.08614708930965884 и результат работы обучения изображен на рисунке 16.

```
600/600 [=====] - 2s 1ms/step - loss: 0.0883
150/150 [=====] - 0s 1ms/step
600/600 [=====] - 2s 1ms/step - loss: 0.0899
150/150 [=====] - 0s 997us/step
600/600 [=====] - 2s 1ms/step - loss: 0.0888
150/150 [=====] - 0s 1ms/step
600/600 [=====] - 2s 1ms/step - loss: 0.0903
150/150 [=====] - 0s 1ms/step
600/600 [=====] - 1s 1ms/step - loss: 0.0911
150/150 [=====] - 0s 1ms/step
Средняя ошибка по кросс-валидации (MSE): 0.08614708930965884
94/94 [=====] - 0s 1ms/step
```

Рисунок 16 — Кросс-валидация (MSE).

4.1.3 Распределение фактических и предсказанных значений

- гистограммы фактических и предсказанных значений коэффициента статистики отражают достоверность модели,
- сглаживание гистограмм позволяет сравнить плотность распределения данных и оценить качество предсказаний нейронной сети.

На рисунке 17 изображен график распределения.

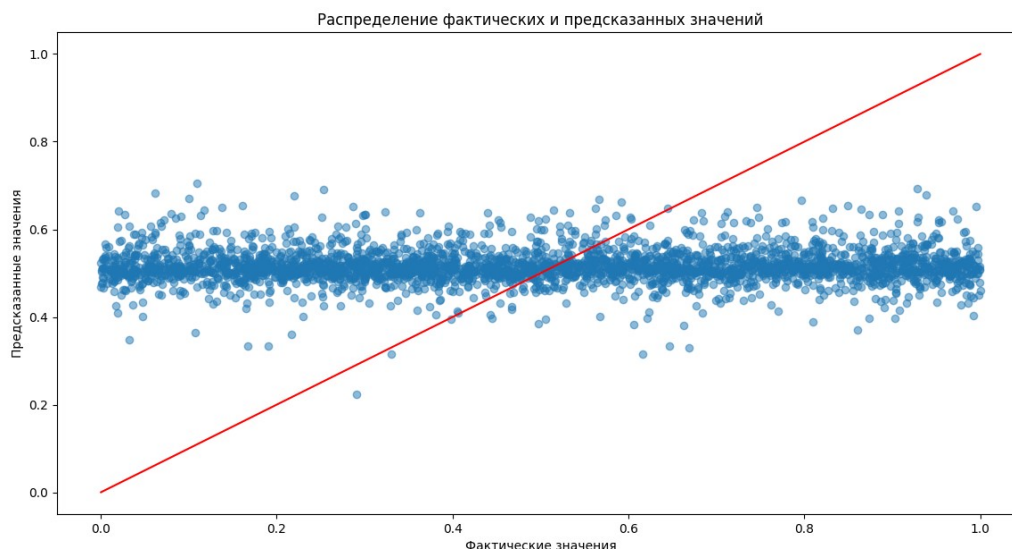


Рисунок 17 — График распределения фактических и предсказанных значений.

4.2 Выводы

- оценка модели на тестовых данных показала потерю (loss) в размере потеря, что говорит о точности модели на независимых данных,
- средняя ошибка по кросс-валидации (MSE) составила средняя ошибка, что подтверждает стабильность и надежность модели на различных подвыборках.

Эти результаты демонстрируют эффективность разработанной модели, её способность предсказывать ключевые параметры спортсменов с высокой точностью и стабильностью.

ЗАКЛЮЧЕНИЕ

В заключение, была разработана веб-платформа, которая обеспечивает максимальную понятность и простоту восприятия для пользователей. Использование интуитивно понятного интерфейса и чёткой навигации позволило достичь высокого уровня удобства в работе с приложением. Реализовано разделение на две основные роли: администратор и пользователь. Зарегистрированные пользователи имеют возможность просматривать информацию на сайте, спортсмены могут оставлять комментарии и участвовать в обсуждениях, а организации — публиковать новости о спортивных мероприятиях. Администраторы обладают расширенными правами доступа, позволяющими им создавать события и администрировать все данные.

Также разработана и внедрена модель машинного обучения на основе нейронных сетей для анализа и прогнозирования спортивных достижений. Этот процесс включал загрузку и предобработку данных, преобразование категориальных признаков в числовые, разделение данных на обучающую, валидационную и тестовую выборки, а также нормализацию данных для улучшения качества обучения модели. Создана и обучена модель нейронной сети с использованием ранней остановки и сохранена лучшая модель. После тщательной оценки на тестовой выборке, модель и объекты предварительной обработки были сохранены для дальнейшего использования через API.

В результате приложение предоставляет полную и актуальную информацию о спортивных достижениях учащихся и статистику игроков, включая данные о прошедших и предстоящих мероприятиях, а также возможность отметки и отслеживания событий.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Крачтен Ф. Введение в Rational Unified Process. 2-е изд. М.: Вильямс, 2002. - 240 с.
2. Лобанов А. А. Большие данные: проблемы обработки //Вестник МГТУ МИРЭА «MSTU MIREA HERALD». – 2014. – №. 3. – с. 50.
3. Бельченко И. В., Дьяченко Р. А., Бельченко В. Е. К вопросу повышения производительности базы данных за счет оптимальной индексации таблиц //«Электронный сетевой политематический журнал" Научные труды КубГТУ". – 2018. – №. 3. – С. 584-586.
4. Elasticsearch B. V. Elasticsearch [software], version. – 2018. – Т. 6. – №. 1.
5. Subramanian S., Inozemtseva L., Holmes R. Live API documentation //Proceedings of the 36th international conference on software engineering. – 2014. – С. 643-652.
6. Cheh C., Chen B. Analyzing openapi specifications for security design issues //2021 IEEE Secure Development Conference (SecDev). – IEEE, 2021. – с. 15-22.
7. Walls C. Spring Boot in action. – Simon and Schuster, 2015.
8. Чернова Е. В. Миграция базы данных //Международный научноисследовательский журнал. – 2015. – №. 5-2 (36). – с. 113-114.
9. Кондуров И. В. Модульное тестирование программного обеспечения на Java с применением библиотек JUnit и Mockito //Программнотехническое обеспечение автоматизированных систем. – 2021. – с. 11-17.
10. Pham K., Reyes F., Rynhart J. Building a Library Search Infrastructure with Elasticsearch //Code4Lib Journal. – 2020. – №. 48.

ПРИЛОЖЕНИЕ А

(обязательное)

Информационная система сбора и анализа данных о спортивных
достижениях обучающихся детско-юношеских спортивных школ

Листинг программы

Файл SeoApplication.java

```
package ru.zderev.seo;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SeoApplication {

    public static void main(String[] args) {
        SpringApplication.run(SeoApplication.class, args);
    }

}
```

Файл MainController.java

```
package ru.zderev.seo.core;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import
org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import ru.zderev.seo.core.messages.web.MessageRepo;
import ru.zderev.seo.core.user.User;
import ru.zderev.seo.core.user.service.UserService;

import java.util.HashMap;

@Controller
@RequestMapping("/")
public class MainController {
    private final MessageRepo repo;
```

```

private final UserService service;

@Value("${spring.profile.active}")
private String profile;

@Autowired
public MainController(MessageRepo messageRepo, UserService
service) {
    this.repo = messageRepo;
    this.service = service;
}

@GetMapping
public String main(Model model) {
    HashMap<Object, Object> data = new HashMap<>();

    // Получите объект Authentication
    Authentication auth =
SecurityContextHolder.getContext().getAuthentication();

    // Получите имя пользователя
    String username = auth.getName();

    // Используйте имя пользователя для получения полной
информации о пользователе
    User user = service.findByEmail(username);

    if (profile != null) {
        // Передайте объект User вместо объекта Principal
        data.put("profile", user);
        data.put("messages", repo.findAll());
    }

    model.addAttribute("frontendData", data);
    model.addAttribute("isDevMode", "dev".equals(profile));
    return "index";
}
}

```

Файл MyConfiguration.java

```

package ru.zderev.seo.core.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.CorsRegistry;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigur
er;

@Configuration

```

```

public class MyConfiguration {

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                    .allowedOrigins("http://localhost:8081")
                    .allowedMethods("GET", "POST", "PUT",
"DELETE", "HEAD", "OPTIONS")
                    .allowedHeaders("*")
                    .allowCredentials(true);
            }
        };
    }
}

```

Файл WebMvcConfig.java

```

package ru.zderev.seo.core.config;

import org.springframework.boot.web.server.ErrorPage;
import
org.springframework.boot.web.server.WebServerFactoryCustomizer;
import
org.springframework.boot.web.servlet.server.ConfigurableServletW
ebServerFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpStatus;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigur
er;

@Configuration
public class WebMvcConfig implements WebMvcConfigurer {
    @Bean
    public
WebServerFactoryCustomizer<ConfigurableServletWebServerFactory>
webServerCustomizer() {
        return container -> {
            container.addErrorPages(new
ErrorPage(HttpStatus.NOT_FOUND, "/"));
        };
    }
}

```

Файл WebSocketConfig.java

```

package ru.zderev.seo.core.config;

import org.springframework.context.annotation.Configuration;

```

```

import
org.springframework.messaging.simp.config.MessageBrokerRegistry;
import
org.springframework.web.socket.config.annotation.EnableWebSocket
MessageBroker;
import
org.springframework.web.socket.config.annotation.StompEndpointRe
gistry;
import
org.springframework.web.socket.config.annotation.WebSocketMessag
eBrokerConfigurer;

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements
WebSocketMessageBrokerConfigurer {
    @Override
    public void registerStompEndpoints(StompEndpointRegistry
registry) {
        registry.addEndpoint("/gs-guide-
websocket").withSockJS();
    }

    @Override
    public void configureMessageBroker(MessageBrokerRegistry
config) {
        config.enableSimpleBroker("/topic");
        config.setApplicationDestinationPrefixes("/app");
    }
}

```

Файл EventType.java

```

package ru.zderev.seo.core.dto;

public enum EventType {
    CREATE, UPDATE, REMOVE
}

```

Файл ObjectType.java

```

package ru.zderev.seo.core.dto;

public enum ObjectType {
    MESSAGE
}

```

Файл WsEventDto.java

```

package ru.zderev.seo.core.dto;

import com.fasterxml.jackson.annotation.JsonRawValue;
import com.fasterxml.jackson.annotation.JsonView;
import ru.zderev.seo.core.view.Views;

```

```

@JsonView(Views.Id.class)
public class WsEventDto {
    private ObjectType objectType;
    private EventType eventType;
    @JsonRawValue
    private String body;

    public WsEventDto(ObjectType objectType, EventType
eventType, String body) {
        System.out.println("WsEventDto: objectType = " +
objectType + ", eventType = " + eventType);
        this.objectType = objectType;
        this.eventType = eventType;
        this.body = body;
    }

    public ObjectType getObjectType() {
        return objectType;
    }

    public void setObjectType(ObjectType objectType) {
        this.objectType = objectType;
    }

    public EventType getEventType() {
        return eventType;
    }

    public void setEventType(EventType eventType) {
        this.eventType = eventType;
    }

    public String getBody() {
        return body;
    }

    public void setBody(String body) {
        this.body = body;
    }
}

```

Файл MessageController.java

```

package ru.zderev.seo.core.messages.web;

import com.fasterxml.jackson.annotation.JsonView;
import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import ru.zderev.seo.core.messages.Message;
import ru.zderev.seo.core.util.WsSender;

```

```

import ru.zderev.seo.core.dto.EventType;
import ru.zderev.seo.core.dto.ObjectType;
import ru.zderev.seo.core.view.Views;

import java.time.LocalDateTime;
import java.util.List;
import java.util.function.BiConsumer;

@RestController
@RequestMapping("seo")
public class MessageController {
    private final MessageRepo messageRepo;
    private final BiConsumer<EventType, Message> wsSender;
    @Autowired
    public MessageController(MessageRepo messageRepo, WsSender
wsSender) {
        this.messageRepo = messageRepo;
        this.wsSender = wsSender.getSender(ObjectType.MESSAGE,
Views.IdName.class);
    }
    @GetMapping
    @JsonView(Views.IdName.class)
    public List<Message> String() {
        return messageRepo.findAll();
    }

    @GetMapping("{id}")
    @JsonView(Views.FullMessage.class)
    public Message getMessage(@PathVariable("id") Message
message) {
        return message;
    }

    @PostMapping
    public Message createUser(@RequestBody Message message) {
        message.setCreationDate(LocalDateTime.now());
        Message updateMessage = messageRepo.save(message);
        System.out.println("createUser: eventType = CREATE,
message = " + updateMessage);
        wsSender.accept(EventType.CREATE, updateMessage);
        return updateMessage;
    }

    @PutMapping("{id}")
    public Message updateUser(@PathVariable("id") Message
messageFromDb, @RequestBody Message message) {
        BeanUtils.copyProperties(message, messageFromDb, "id");
        Message updateMessage = messageRepo.save(messageFromDb);
        System.out.println("updateUser: eventType = UPDATE,
message = " + updateMessage);
        wsSender.accept(EventType.UPDATE, updateMessage);
    }
}

```

```

        return updateMessage;
    }

    @DeleteMapping("{id}")
    public void deleteUser(@PathVariable("id") Message message)
    {
        messageRepo.delete(message);
        System.out.println("deleteUser: eventType = REMOVE,
message = " + message);
        wsSender.accept(EventType.REMOVE, message);
    }
}

```

Файл MessageRepo.java

```

package ru.zderev.seo.core.messages.web;

import org.springframework.data.jpa.repository.JpaRepository;
import ru.zderev.seo.core.messages.Message;

public interface MessageRepo extends JpaRepository<Message,
Long> {
}

```

Файл Message.java

```

package ru.zderev.seo.core.messages;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonView;
import jakarta.persistence.*;
import ru.zderev.seo.core.view.Views;

import java.time.LocalDateTime;
import java.util.List;

@Entity
public class Message {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @JsonView(Views.IdName.class)
    private Long id;

    @JsonView(Views.IdName.class)
    private String text;

    @Column(uptdatable = false)
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern =
"yyyy-MM-dd HH:mm:ss")
    private LocalDateTime creationDate;

    // Поле для вида спорта

```

```

@JsonView(Views.IdName.class)
private String sport;

// Список участников
@ElementCollection
@CollectionTable(name = "participants", joinColumns =
@JoinColumn(name = "message_id"))
private List<Participant> participants;

// Геттеры и сеттеры
public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getText() {
    return text;
}

public void setText(String text) {
    this.text = text;
}

public LocalDateTime getCreationDate() {
    return creationDate;
}

public void setCreationDate(LocalDateTime creationDate) {
    this.creationDate = creationDate;
}

public String getSport() {
    return sport;
}

public void setSport(String sport) {
    this.sport = sport;
}

public List<Participant> getParticipants() {
    return participants;
}

public void setParticipants(List<Participant> participants)
{
    this.participants = participants;
}
}

```

Файл Participant.java

```
package ru.zderev.seo.core.messages;

import com.fasterxml.jackson.annotation.JsonView;
import ru.zderev.seo.core.view.Views;

import jakarta.persistence.Column;
import jakarta.persistence.Embeddable;

@Embeddable
public class Participant {

    @JsonView(Views.IdName.class)
    @Column(name = "email")
    private String email;

    @JsonView(Views.IdName.class)
    @Column(name = "goals")
    private String goals;

    @JsonView(Views.IdName.class)
    @Column(name = "position")
    private String position;

    @JsonView(Views.IdName.class)
    @Column(name = "team_place")
    private String teamPlace;

    // Геттеры и сеттеры
    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getGoals() {
        return goals;
    }

    public void setGoals(String goals) {
        this.goals = goals;
    }

    public String getPosition() {
        return position;
    }

    public void setPosition(String position) {
```

```

        this.position = position;
    }

    public String getTeamPlace() {
        return teamPlace;
    }

    public void setTeamPlace(String teamPlace) {
        this.teamPlace = teamPlace;
    }
}

```

Файл SecurityConfig.java

```

package ru.zderev.seo.core.security.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.authentication.AuthenticationProvid
er;
import
org.springframework.security.authentication.dao.DaoAuthenticatio
nProvider;
import
org.springframework.security.config.annotation.method.configurat
ion.EnableMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.Http
Security;
import
org.springframework.security.config.annotation.web.configuration
.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configurers.A
bstractAuthenticationFilterConfigurer;
import
org.springframework.security.config.annotation.web.configurers.A
bstractHttpConfigurer;
import
org.springframework.security.config.annotation.web.configurers.L
ogoutConfigurer;
import
org.springframework.security.core.userdetails.UserDetailsService
;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import

```

```

org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler;
import
ru.zderev.seo.core.user.service.details.MyUserDetailsService;

@EnableWebSecurity
@Configuration
@EnableMethodSecurity
public class SecurityConfig {

    @Bean
    public UserDetailsService userDetailsService() {
        return new MyUserDetailsService();
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity
http) throws Exception {
        return http.csrf(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/", "/login**",
"/js/**", "/error**", "/user/new-user").permitAll()
                .requestMatchers("/**", "seo/**",
"user/**").authenticated()
                .anyRequest().authenticated())
            .formLogin(AbstractAuthenticationFilterConfigurer::permitAll)
                .logout(logout ->
logout.logoutUrl("/logout").addLogoutHandler(new
SecurityContextLogoutHandler()))
                .build();
    }

    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider provider = new
DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailsService());
        provider.setPasswordEncoder(passwordEncoder());
        return provider;
    }

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

```

Файл MyUserDetails.java

```
package ru.zderev.seo.core.user.service.details;

import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAuthori
ty;
import
org.springframework.security.core.userdetails.UserDetails;
import ru.zderev.seo.core.user.User;

import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;

public class MyUserDetails implements UserDetails {

    private User user;

    public MyUserDetails(User user) {
        this.user = user;
    }

    @Override
    public Collection<? extends GrantedAuthority>
getAuthorities() {
        return Collections.singletonList(new
SimpleGrantedAuthority("ROLE_" + user.getRole().name()));
    }

    @Override
    public String getPassword() {
        return user.getPassword();
    }

    @Override
    public String getUsername() {
        return user.getEmail();
    }

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

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

```

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

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

```

Файл MyUserService.java

```

package ru.zderev.seo.core.user.service.details;

import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService
;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import ru.zderev.seo.core.user.User;
import ru.zderev.seo.core.user.web.UserRepo;

import java.util.Optional;

@Service
public class MyUserService implements UserDetailsService {
    @Autowired
    private UserRepo repo;

    @Override
    public UserDetails loadUserByUsername(String username)
throws UsernameNotFoundException {
        Optional<User> user = repo.findByEmail(username);
        return user.map(MyUserDetails::new)
            .orElseThrow(() -> new
UsernameNotFoundException(username + " not found"));
    }
}

```

Файл UserServiceImpl.java

```

package ru.zderev.seo.core.user.service.impl;

import org.springframework.beans.BeanUtils;
import org.springframework.beans.BeanWrapper;
import org.springframework.beans.BeanWrapperImpl;
import

```

```

org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import ru.zderev.seo.core.messages.Message;
import ru.zderev.seo.core.user.User;
import ru.zderev.seo.core.user.service.UserService;
import ru.zderev.seo.core.user.web.UserRepo;

import java.util.HashSet;
import java.util.Optional;
import java.util.Set;

@Service
public class UserServiceImpl implements UserService {
    private UserRepo repo;
    private PasswordEncoder encoder;

    public UserServiceImpl(UserRepo repo, PasswordEncoder
encoder) {
        this.repo = repo;
        this.encoder = encoder;
    }

    @Override
    public User updateUser(User user, User userFromDb) {
//        BeanUtils.copyProperties(user, userFromDb, "id");
        user.setActivity(true);
        copyNonNullProperties(user, userFromDb);
        User updateUser = repo.save(userFromDb);
        return updateUser;
    }

    @Override
    public void createUser(User user) {
        user.setPassword(encoder.encode(user.getPassword()));
        repo.save(user);
    }

    @Override
    public User findByEmail(String email) {
        Optional<User> userOptional = repo.findByEmail(email);
        if (userOptional.isPresent()) {
            User user = userOptional.get();
            return user;
        } else {
            throw new RuntimeException("Not Found");
        }
    }

    public void copyNonNullProperties(Object src, Object target)
{
        BeanUtils.copyProperties(src, target,

```

```

getNullPropertyNames(src));
    }

    private String[] getNullPropertyNames(Object source) {
        final BeanWrapper src = new BeanWrapperImpl(source);
        java.beans.PropertyDescriptor[] pds =
src.getPropertyDescriptors();

        Set<String> emptyNames = new HashSet<String>();
        for(java.beans.PropertyDescriptor pd : pds) {
            Object srcValue =
src.getPropertyValue(pd.getName());
            if (srcValue == null) emptyNames.add(pd.getName());
        }
        String[] result = new String[emptyNames.size()];
        return emptyNames.toArray(result);
    }
}

```

Файл UserService.java

```

package ru.zderev.seo.core.user.service;

import ru.zderev.seo.core.user.User;

public interface UserService {
    public void createUser(User user);

    public User updateUser(User user, User userFromDb);
    User findByEmail(String email);
}

```

Файл UserController.java

```

package ru.zderev.seo.core.user.web;

import org.springframework.beans.BeanUtils;
import org.springframework.web.bind.annotation.*;
import ru.zderev.seo.core.dto.EventType;
import ru.zderev.seo.core.messages.Message;
import ru.zderev.seo.core.user.User;
import ru.zderev.seo.core.user.service.impl.UserServiceImpl;

@RestController
@RequestMapping("user")
public class UserController {
    private UserServiceImpl service;

    public UserController(UserServiceImpl service) {
        this.service = service;
    }

    @PostMapping("new-user")

```

```

        public String createUser(@RequestBody User user) {
            System.out.println(user);
            service.createUser(user);
            System.out.print(user.toString());
            return "User " + user.getEmail() + " created";
        }

        @PutMapping("{id}")
        public String updateUser(@PathVariable("id") User
userFromDb, @RequestBody User user) {
            service.updateUser(user, userFromDb);
            return "User " + user.getFirstName() + " updated";
        }
    }
}

```

Файл UserRepo.java

```

package ru.zderev.seo.core.user.web;

import org.springframework.data.jpa.repository.JpaRepository;
import ru.zderev.seo.core.user.User;

import java.util.Optional;

public interface UserRepo extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String userEmail);
}

```

Файл Role.java

```

package ru.zderev.seo.core.user;

public enum Role {
    USER,
    ADMIN
}

```

Файл User.java

```

package ru.zderev.seo.core.user;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;
import jakarta.persistence.*;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAuthori
ty;
import
org.springframework.security.core.userdetails.UserDetails;

import java.io.Serializable;
import java.util.Collection;
import java.util.List;

```

```

import java.util.Objects;

@Entity
@Table(name = "usr")
public class User implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String firstName;
    private String lastName;
    @Column(unique = true)
    private String email;
    private String password;
    private Role role;
    private Double statistic;
    private String sportType;
    private Integer weight;
    private Integer height;
    private Integer discharge;
    private Integer age;
    private Integer currentLocation;
    private Integer previousLocation;
    private String position;
    private Double numberOfCompetitions;
    private boolean activity;

    public Double getStatistic() {
        return statistic;
    }

    public void setStatistic(Double statistic) {
        this.statistic = statistic;
    }

    public String getSportType() {
        return sportType;
    }

    public void setSportType(String sportType) {
        this.sportType = sportType;
    }

    public Integer getWeight() {
        return weight;
    }

    public void setWeight(Integer weight) {
        this.weight = weight;
    }

    public Integer getHeight() {

```

```

        return height;
    }

    public void setHeight(Integer height) {
        this.height = height;
    }

    public Integer getDischarge() {
        return discharge;
    }

    public void setDischarge(Integer discharge) {
        this.discharge = discharge;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public Integer getCurrentLocation() {
        return currentLocation;
    }

    public void setCurrentLocation(Integer currentLocation) {
        this.currentLocation = currentLocation;
    }

    public Integer getPreviousLocation() {
        return previousLocation;
    }

    public void setPreviousLocation(Integer previousLocation) {
        this.previousLocation = previousLocation;
    }

    public String getPosition() {
        return position;
    }

    public void setPosition(String position) {
        this.position = position;
    }

    public Double getNumberOfCompetitions() {
        return numberOfCompetitions;
    }
}

```

```

    public void setNumberOfCompetitions(Double
numberOfCompetitions) {
        this.numberOfCompetitions = numberOfCompetitions;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Role getRole() {
        return role;
    }

    public void setRole(Role role) {
        this.role = role;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

```

```

    }

    public boolean isActivity() {
        return activity;
    }

    public void setActivity(boolean activity) {
        this.activity = activity;
    }

    @Override
    public String toString() {
        return "User{" +
            "firstName='" + firstName + '\'' +
            ", lastName='" + lastName + '\'' +
            ", email='" + email + '\'' +
            ", password='" + password + '\'' +
            ", role=" + role +
            ", activity=" + activity +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return
false;
        User user = (User) o;
        return activity == user.activity &&
Objects.equals(firstName, user.firstName) &&
Objects.equals(lastName, user.lastName) && Objects.equals(email,
user.email) && Objects.equals(password, user.password) && role
== user.role;
    }

    @Override
    public int hashCode() {
        return Objects.hash(firstName, lastName, email,
password, role, activity);
    }
}

```

Файл UserDto.java

```

package ru.zderev.seo.core.user;

public class UserDto {
    private String email;
    private Double statistic;
    private String sportType;
    private Integer weight;
    private Integer height;

```

```

private Integer discharge;
private Integer age;
private Integer currentLocation;
private Integer previousLocation;
private Integer gols;
private String position;
private Double numberOfCompetitions;

public UserDto(String email, Double statistic, String
sportType, Integer weight, Integer height, Integer discharge,
Integer age, Integer currentLocation, Integer previousLocation,
Integer gols, String position, Double numberOfCompetitions) {
    this.email = email;
    this.statistic = statistic;
    this.sportType = sportType;
    this.weight = weight;
    this.height = height;
    this.discharge = discharge;
    this.age = age;
    this.currentLocation = currentLocation;
    this.previousLocation = previousLocation;
    this.gols = gols;
    this.position = position;
    this.numberOfCompetitions = numberOfCompetitions;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public Double getStatistic() {
    return statistic;
}

public void setStatistic(Double statistic) {
    this.statistic = statistic;
}

public String getSportType() {
    return sportType;
}

public void setSportType(String sportType) {
    this.sportType = sportType;
}

public Integer getWeight() {

```

```

        return weight;
    }

    public void setWeight(Integer weight) {
        this.weight = weight;
    }

    public Integer getHeight() {
        return height;
    }

    public void setHeight(Integer height) {
        this.height = height;
    }

    public Integer getDischarge() {
        return discharge;
    }

    public void setDischarge(Integer discharge) {
        this.discharge = discharge;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public Integer getCurrentLocation() {
        return currentLocation;
    }

    public void setCurrentLocation(Integer currentLocation) {
        this.currentLocation = currentLocation;
    }

    public Integer getPreviousLocation() {
        return previousLocation;
    }

    public void setPreviousLocation(Integer previousLocation) {
        this.previousLocation = previousLocation;
    }

    public Integer getGols() {
        return gols;
    }

```

```

    public void setGols(Integer gols) {
        this.gols = gols;
    }

    public String getPosition() {
        return position;
    }

    public void setPosition(String position) {
        this.position = position;
    }

    public Double getNumberOfCompetitions() {
        return numberOfCompetitions;
    }

    public void setNumberOfCompetitions(Double
numberOfCompetitions) {
        this.numberOfCompetitions = numberOfCompetitions;
    }
}

```

Файл WsSender.java

```

package ru.zderev.seo.core.util;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.ObjectWriter;
import org.springframework.messaging.simp.SimpMessagingTemplate;
import org.springframework.stereotype.Component;
import ru.zderev.seo.core.dto.WsEventDto;
import ru.zderev.seo.core.dto.EventType;
import ru.zderev.seo.core.dto.ObjectType;

import java.util.function.BiConsumer;

@Component
public class WsSender {
    private final SimpMessagingTemplate template;
    private final ObjectMapper mapper;

    public WsSender(SimpMessagingTemplate template, ObjectMapper
mapper) {
        this.template = template;
        this.mapper = mapper;
    }

    public <T> BiConsumer<EventType, T> getSender(ObjectType
objectType, Class view) {
        ObjectWriter writer = mapper
            .setConfig(mapper.getSerializationConfig())

```

```

        .writerWithView(view);

return (EventType eventType, T payload) -> {
    String value = null;

    try {
        value = writer.writeValueAsString(payload);
    } catch (JsonProcessingException e) {
        throw new RuntimeException(e);
    }

    template.convertAndSend(
        "/topic/activity",
        new WsEventDto(objectType, eventType, value)
    );
};
}
}

```

Файл Views.java

```

package ru.zderev.seo.core.view;

public final class Views {
    public interface Id {}

    public interface IdName extends Id {}

    public interface FullMessage extends Id {}
}

```

ПРИЛОЖЕНИЕ A.25

Файл NotFoundException.java

```

package ru.zderev.seo.exceptions;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.NOT_FOUND)
public class NotFoundException extends RuntimeException{
}

```

Файл MessageForm.vue

```

<template>
  <v-container fluid>
    <v-row
      class="d-flex flex-column align-center"
      style="max-width: 800px; margin: 0 auto;"
    >
      <v-col cols="12">
        <v-text-field
          class="mb-4"

```

```

        label="Новое событие"
        outlined
        v-model="text"
    ></v-text-field>
</v-col>
<v-col cols="12">
    <v-select
        class="mb-4"
        label="Выберите вид спорта"
        :items="['Футбол', 'Легкая атлетика']"
        v-model="sport"
    ></v-select>
</v-col>

<v-expansion-panels v-if="sport" multiple class="w-100">
    <v-expansion-panel v-if="sport === 'Футбол' || sport ===
'Легкая атлетика'">
        <v-expansion-panel-header>
            <span class="header-text">Добавить участников</span>
        </v-expansion-panel-header>
        <v-expansion-panel-content>
            <v-row
                v-for="(participant, index) in participants"
                :key="index"
                class="participant-card mb-4"
            >
                <v-col cols="12" class="card-header">
                    <span class="subtitle-1">Участник {{ index + 1
}}</span>
                    <v-btn icon @click="removeParticipant(index)"
color="error">
                        <v-icon>mdi-close</v-icon>
                    </v-btn>
                </v-col>
                <v-col cols="12">
                    <v-text-field
                        v-model="participant.email"
                        outlined
                        label="Email"
                    ></v-text-field>
                </v-col>
                <v-col cols="12" v-if="sport === 'Футбол'">
                    <v-text-field
                        v-model="participant.teamPlace"
                        outlined
                        label="Место команды"
                    ></v-text-field>
                    <v-text-field
                        v-model="participant.goals"
                        outlined
                        label="Количество голов"

```

```

        ></v-text-field>
      </v-col>
      <v-col cols="12" v-if="sport === 'Легкая
атлетика'">
        <v-text-field
          v-model="participant.position"
          outlined
          label="Занятое место"
        ></v-text-field>
      </v-col>
    </v-row>
    <v-col cols="12" class="d-flex justify-center mb-4">
      <v-btn @click="addParticipant" color="success"
outlined>Добавить участника</v-btn>
    </v-col>
  </v-expansion-panel-content>
</v-expansion-panel>
</v-expansion-panels>

  <v-col cols="12" class="d-flex justify-center mt-4">
    <v-btn @click="save" color="primary"
outlined>Создать</v-btn>
  </v-col>
</v-row>
</v-container>
</template>

<script>
import { mapActions } from "vuex";

export default {
  props: ["messageAttr"],
  data() {
    return {
      text: "",
      id: "",
      sport: "",
      participants: []
    };
  },
  watch: {
    messageAttr(newVal) {
      this.text = newVal.text;
      this.id = newVal.id;
    }
  },
  methods: {
    ...mapActions(["updateMessageAction", "addMessageAction"]),
    addParticipant() {
      const newParticipant = { email: "", teamPlace: "", goals:
"", position: "" };

```

```

        this.participants.push(newParticipant);
    },
    removeParticipant(index) {
        this.participants.splice(index, 1);
    },
    async save() {
        const plainParticipants =
JSON.parse(JSON.stringify(this.participants));

        const message = {
            id: this.id,
            text: this.text,
            sport: this.sport,
            participants: plainParticipants
        };

        try {
            if (this.id) {
                await this.updateMessageAction(message);
            } else {
                await this.addAction(message);
            }

            this.resetForm();
        } catch (error) {
            console.error("Ошибка при сохранении данных:", error);
        }
    },
    resetForm() {
        this.text = "";
        this.id = "";
        this.sport = "";
        this.participants = [];
    }
};
</script>

```

```

<style scoped>
.mb-4 {
    margin-bottom: 16px !important;
}
.mt-4 {
    margin-top: 16px !important;
}
.header-text {
    font-weight: bold;
    font-size: 18px;
}
.subtitle-1 {
    font-weight: 500;
}

```

```

}
.participant-card {
  border: 1px solid #e0e0e0;
  border-radius: 8px;
  padding: 16px;
  background-color: #fafafa;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
.card-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  border-bottom: 1px solid #e0e0e0;
  padding-bottom: 8px;
  margin-bottom: 16px;
}
}
</style>

```

Файл MessagesRow.vue

```

<template>
  <v-card class="my-2">
    <v-card-text primary-title>
      <i>({{ message.id }})</i> {{ message.text }}
    </v-card-text>
    <v-card-actions>
      <v-btn @click="edit" small flat>
        Отредактировать
      </v-btn>
      <v-btn @click="del" small flat>
        Удалить
      </v-btn>
    </v-card-actions>
  </v-card>
</template>

<script>
import { mapActions } from "vuex";

export default {
  props: ['message', 'editMessage'],
  methods: {
    ...mapActions(['removeMessageAction']),
    edit() {
      this.editMessage(this.message);
    },
    del() {
      this.removeMessageAction(this.message)
    }
  }
}
</script>

```

```
<style>
```

```
</style>
```

Файл App.vue

```
<template>
```

```
  <v-layout class="rounded rounded-md">
    <v-app-bar color="teal-lighten-5" dark>
      <v-toolbar-title>
        <router-link to="/">
          <v-btn color="black">
            <span>Информационная система сбора и анализа
данных</span>
          </v-btn>
        </router-link>
      </v-toolbar-title>
      <v-spacer></v-spacer>
      <v-toolbar-items class="hidden-sm-and-down">
        <v-btn>
          <router-link to="/profile">
            <span v-if="profile">{{ profile.firstName + " " +
profile.lastName }}</span>
          </router-link>
        </v-btn>
        <v-btn v-if="profile" icon="mdi-export" href="/logout">
        </v-btn>
      </v-toolbar-items>
    </v-app-bar>

    <v-main class="pa-4 pt-10" style="min-height: 300px;
background: #f5f5f5;">
      <RouterView />
    </v-main>
  </v-layout>
</template>
```

```
<script>
```

```
import { addHandler } from "../util/ws";
import { mapState, mapMutations } from 'vuex'
export default {
  computed: mapState(['profile']),
  methods: mapMutations(['addMessageMutation',
'updateMessageMutation', 'removeMessageMutation']),
  created() {
    addHandler(data => {
      console.log(data.objectType, " ", data.eventType)
      if (data.objectType === 'MESSAGE') {
        switch (data.eventType) {
          case 'CREATE' :
            this.addMessageMutation(data.body)

```

```

        break
      case 'UPDATE' :
        this.updateMessageMutation(data.body)
        break
      case 'REMOVE' :
        this.removeMessageMutation(data.body)
        break
      default:
        console.error(`event type is unknown
"${data.eventType}"`)
    }
  } else console.error(`object type is unknown
"${data.objectType}"`)
  })
}
}
</script>

<style>

</style>

```

Файл Auth.vue

```

<template>
  <v-container>
    Необходимо авторизоваться
    <a href="/login">здесь</a>
  </v-container>
</template>

<script>
  export default {
    name: 'Auth'
  }
</script>

<style>

</style>

```

MessagesList.vue

```

<template>
  <v-container>
    <v-col cols="12">
      <message-form :messageAttr="message" />
      <messages-row v-for="message in sortedMessages"
        :key="message.id"
        :message="message"
        :editMessage="editMessage" />
    </v-col>
  </v-container>

```

```

</template>

<script>
import MessagesRow from
"../components/messages/MessagesRow.vue";
import MessageForm from
"../components/messages/MessageForm.vue";
import { mapGetters } from 'vuex'
export default {
  components: {
    MessagesRow,
    MessageForm
  },
  data() {
    return {
      message: null
    }
  },
  computed: mapGetters(['sortedMessages']),
  methods: {
    editMessage(message) {
      this.message = message;
    }
  }
}
</script>

<style>

</style>

```

Profile.vue

```

<template>
  <v-container>
    <v-card>
      <v-card-title>
        <h1>Профиль пользователя</h1>
      </v-card-title>
      <v-card-text>
        <v-list-item-group>
          <v-list-item>
            <v-list-item-content>
              <v-list-item-title>Имя</v-list-item-title>
              <v-list-item-subtitle>{{ profile.firstName }}</v-
list-item-subtitle>
            </v-list-item-content>
          </v-list-item>
          <v-list-item>
            <v-list-item-content>
              <v-list-item-title>Фамилия</v-list-item-title>
              <v-list-item-subtitle>{{ profile.lastName }}</v-

```

```

list-item-subtitle>
    </v-list-item-content>
</v-list-item>
<v-list-item>
    <v-list-item-content>
        <v-list-item-title>Email</v-list-item-title>
        <v-list-item-subtitle>{{ profile.email }}</v-list-
item-subtitle>
    </v-list-item-content>
</v-list-item>
<v-list-item>
    <v-list-item-content>
        <v-list-item-title>Роль</v-list-item-title>
        <v-list-item-subtitle>{{ profile.role }}</v-list-
item-subtitle>
    </v-list-item-content>
</v-list-item>
<template v-if="profile.activity === false ||
profile.role === 'ADMIN'">
    <v-layout column class="d-flex align-center flex-
wrap">
        <v-container class="col-4">
            <v-select
                label="Вид спорта"
                :items="['Футбол', 'Легкая атлетика']"
                variant="outlined"
                v-model="sport"
                @change="sport = $event"
            ></v-select>
        </v-container>
        <v-container class="col-4" v-if="sport ===
'Футбол'">
            <v-container class="col-4">
                <v-select
                    label="Позиция"
                    :items="['Вратарь', 'Защитник',
'Полузащитник', 'Нападающий']"
                    variant="outlined"
                    v-model="position"
                    @change="position = $event"
                ></v-select>
            </v-container>
        </v-container>
        <v-container class="col-4">
            <v-text-field
                label="Вес"
                variant="outlined"
                v-model="weight"
            ></v-text-field>
        </v-container>
        <v-container class="col-4">

```

```

        <v-text-field
            label="Пост"
            variant="outlined"
            v-model="height"
        ></v-text-field>
    </v-container>
    <v-container class="col-4">
        <v-select
            label="Разряд"
            :items="['1', '2', '3', '4', '5', '6']"
            variant="outlined"
            v-model="discharge"
            @change="discharge = $event"
        ></v-select>
    </v-container>
    <v-container class="col-4">
        <v-text-field
            label="Возраст"
            variant="outlined"
            v-model="age"
        ></v-text-field>
    </v-container>
    <v-container class="col-4">
        <v-text-field
            label="Количество соревнований"
            variant="outlined"
            v-model="numberOfCompetitions"
        ></v-text-field>
    </v-container>
    <v-container class="col-4">
        <v-text-field
            label="Текущее место"
            variant="outlined"
            v-model="currentLocation"
        ></v-text-field>
    </v-container>

    <v-btn class="ma-3" @click="save">
        Сохранить
    </v-btn>
</v-layout>
</template>
<template v-else>
    <v-list-item>
        <v-list-item-content>
            <v-list-item-title>Коэффициент статистики</v-
list-item-title>
            <v-slider
                v-model="profile.statistic"
                :color="getColor(profile.statistic)"
                thumb-label

```

```

        :max="1"
        :step="0.01"
        readonly
    >

    </v-slider>
</v-list-item-content>
</v-list-item>
<v-list-item>
    <v-list-item-content>
        <v-list-item-title>Вид спорта</v-list-item-
title>
        <v-list-item-subtitle>{{ profile.sportType
}}</v-list-item-subtitle>
    </v-list-item-content>
</v-list-item>
<v-list-item>
    <v-list-item-content>
        <v-list-item-title>Вес</v-list-item-title>
        <v-list-item-subtitle>{{ profile.weight }}</v-
list-item-subtitle>
    </v-list-item-content>
</v-list-item>
<v-list-item>
    <v-list-item-content>
        <v-list-item-title>Рост</v-list-item-title>
        <v-list-item-subtitle>{{ profile.height }}</v-
list-item-subtitle>
    </v-list-item-content>
</v-list-item>
<v-list-item>
    <v-list-item-content>
        <v-list-item-title>Разряд</v-list-item-title>
        <v-list-item-subtitle>{{ profile.discharge
}}</v-list-item-subtitle>
    </v-list-item-content>
</v-list-item>
<v-list-item>
    <v-list-item-content>
        <v-list-item-title>Возраст</v-list-item-title>
        <v-list-item-subtitle>{{ profile.age }}</v-list-
item-subtitle>
    </v-list-item-content>
</v-list-item>
<v-list-item>
    <v-list-item-content>
        <v-list-item-title>Текущее место</v-list-item-
title>
        <v-list-item-subtitle>{{ profile.currentLocation
}}</v-list-item-subtitle>
    </v-list-item-content>

```

```

        </v-list-item>
        <v-list-item>
          <v-list-item-content>
            <v-list-item-title>Предыдущее место</v-list-
item-title>
              <v-list-item-subtitle>{{
profile.previousLocation }}</v-list-item-subtitle>
            </v-list-item-content>
          </v-list-item>
          <v-list-item>
            <v-list-item-content>
              <v-list-item-title>Позиция</v-list-item-title>
              <v-list-item-subtitle>{{ profile.position }}</v-
list-item-subtitle>
            </v-list-item-content>
          </v-list-item>
          <v-list-item>
            <v-list-item-content>
              <v-list-item-title>Количество соревнований</v-
list-item-title>
              <v-list-item-subtitle>{{
profile.numberOfCompetitions }}</v-list-item-subtitle>
            </v-list-item-content>
          </v-list-item>
        </template>
      </v-list-item-group>
    </v-card-text>
  </v-card>
</v-container>
</template>

<script>
import { mapState } from 'vuex'

export default {
  name: 'Profile',
  data: () => ({
    firstName: null,
    lastName: null,
    email: null,
    role: null,
    sport: null,
    position: null,
    weight: null,
    height: null,
    discharge: null,
    age: null,
    numberOfCompetitions: null,
    currentLocation: null
  }),
  computed: mapState(['profile']),

```

```

methods: {
  getColor(value) {
    if (value < 0.33) {
      return 'red';
    } else if (value < 0.66) {
      return 'yellow';
    } else {
      return 'green';
    }
  },
  async save() {
    const userData = {
      sportType: this.sport,
      position: this.position,
      weight: this.weight,
      height: this.height,
      discharge: this.discharge,
      age: this.age,
      numberOfCompetitions: this.numberOfCompetitions,
      currentLocation: this.currentLocation,
    };

    console.log(userData)

    const response = await fetch(`/user/${this.profile.id}`, {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(userData),
    });

    if (response.ok) {
      alert('Данные успешно обновлены');
    } else {
      alert('Произошла ошибка при обновлении данных');
    }
  },
}
}
</script>

<style>

</style>

```

Файл router.js

```

import {createRouter, createWebHistory} from 'vue-router'

import MessagesList from "../pages/MessagesList.vue";

```

```

import Auth from "../pages/Auth.vue";
import Profile from "../pages/Profile.vue";

const routes = [
  { path: '/', component: MessagesList },
  { path: '/auth', component: Auth },
  { path: '/profile', component: Profile },
  // { path: '*', component: MessagesList },
]

export default createRouter({
  history: createWebHistory(),
  routes,
})

```

Файл ws.js

```

import { Stomp } from "@stomp/stompjs"
import SockJS from 'sockjs-client'

let stompClient = Stomp.over(() => new
SockJS('http://localhost:8081/gs-guide-websocket'))
const handlers = []

stompClient.debug = function () {}

stompClient.onConnect = frame => {
  console.log('Connected: ' + frame)
  stompClient.subscribe('/topic/activity', message => {
    handlers.forEach(handler =>
handler(JSON.parse(message.body)))
  })
}

export function addHandler(handler) {
  handlers.push(handler)
}

stompClient.onWebSocketError = (error) => {
  console.error('Error with websocket', error)
}

stompClient.onStompError = (frame) => {
  console.error('Broker reported error: ' +
frame.headers['message'])
  console.error('Additional details: ' + frame.body)
}

export function connect() {
  stompClient.activate()
}

```

```

export function disconnect() {
  stompClient.deactivate().then(r =>
    console.log("Disconnected"))
}

export function sendMessage(message) {
  stompClient.publish({
    destination: "/app/changeMessage",
    body: JSON.stringify(message)
  })
}

```

Файл vuetify.js

```

import { createVuetify } from "vuetify";
import * as components from "vuetify/components";
import * as directives from "vuetify/directives";
import "vuetify/styles";
import '@mdi/font/css/materialdesignicons.css'
import 'vuetify/dist/vuetify.min.css'

export const vuetify = createVuetify({
  components,
  directives,
  icons: {
    defaultSet: 'mdi',
  },
});

```

Файл main.js

```

import { createApp } from 'vue';
import App from 'pages/App.vue';
import { connect } from "../util/ws";
import { vuetify } from "../vuetify/vuetify"
import '@babel/polyfill'
import { createStore } from 'vuex'
import router from 'route/router'

const store = createStore({
  state : {
    messages: frontendData.messages,
    profile: frontendData.profile
  },
  getters :{
    sortedMessages: state => state.messages.sort((a, b) => -
(a.id - b.id))
  },
  mutations: {
    addMessageMutation(state, message) {
      state.messages = [

```

```

        ...state.messages,
        message
    ]
},
updateMessageMutation(state, message) {
    const updateIndex = state.messages.findIndex(item =>
item.id === message.id)
    state.messages = [
        ...state.messages.slice(0, updateIndex),
        message,
        ...state.messages.slice(updateIndex + 1)

    ]
},
removeMessageMutation(state, message) {
    const deleteIndex = state.messages.findIndex(item =>
item.id === message.id)
    if (deleteIndex > -1) {
        state.messages = [
            ...state.messages.slice(0, deleteIndex),
            ...state.messages.slice(deleteIndex + 1)

        ]
    }
},
},
actions: {
    async addMessageAction({commit}, message) {
        console.log(message)
        const response = await fetch('/seo', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(message),
        });

        if (response.ok) {
            const data = await response.json();
            commit('addMessageMutation', data);
        } else {
            console.error('Ошибка:', response.status);
        }
    },
    async updateMessageAction({commit}, message) {
        const response = await fetch(`/seo/${message.id}`, {
            method: 'PUT',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(message),
        });
    });
}

```

```

        if (response.ok) {
            const data = await response.json();
            commit('updateMessageMutation', data);
        } else {
            console.error('Ошибка:', response.status);
        }
    },
    async removeMessageAction({commit}, message) {
        const response = await fetch(`/seo/${message.id}`, {
            method: 'DELETE',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(message),
        });

        if (response.ok) {
            commit('removeMessageMutation', message);
        } else {
            console.error('Ошибка:', response.status);
        }
    }
}

}))

if (frontendData.profile) {
    connect();
}

const app = createApp(App);
app.use(router)
app.use(vuetify)
app.use(store)
app.mount('#app');
```

ПРИЛОЖЕНИЕ Б

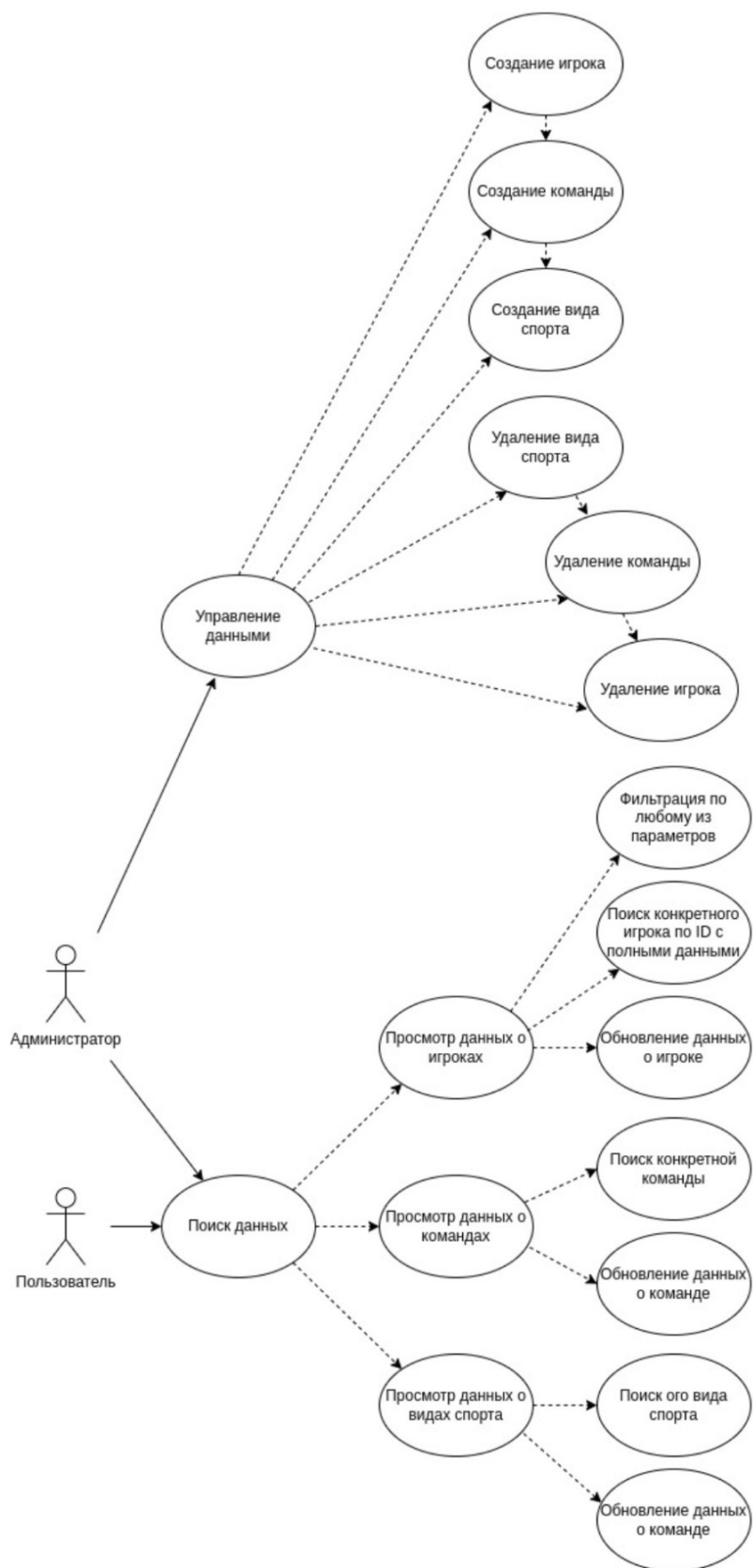
(обязательное)

Информационная система сбора и анализа данных о спортивных
достижениях обучающихся детско-юношеских спортивных школ

UML –диаграммы

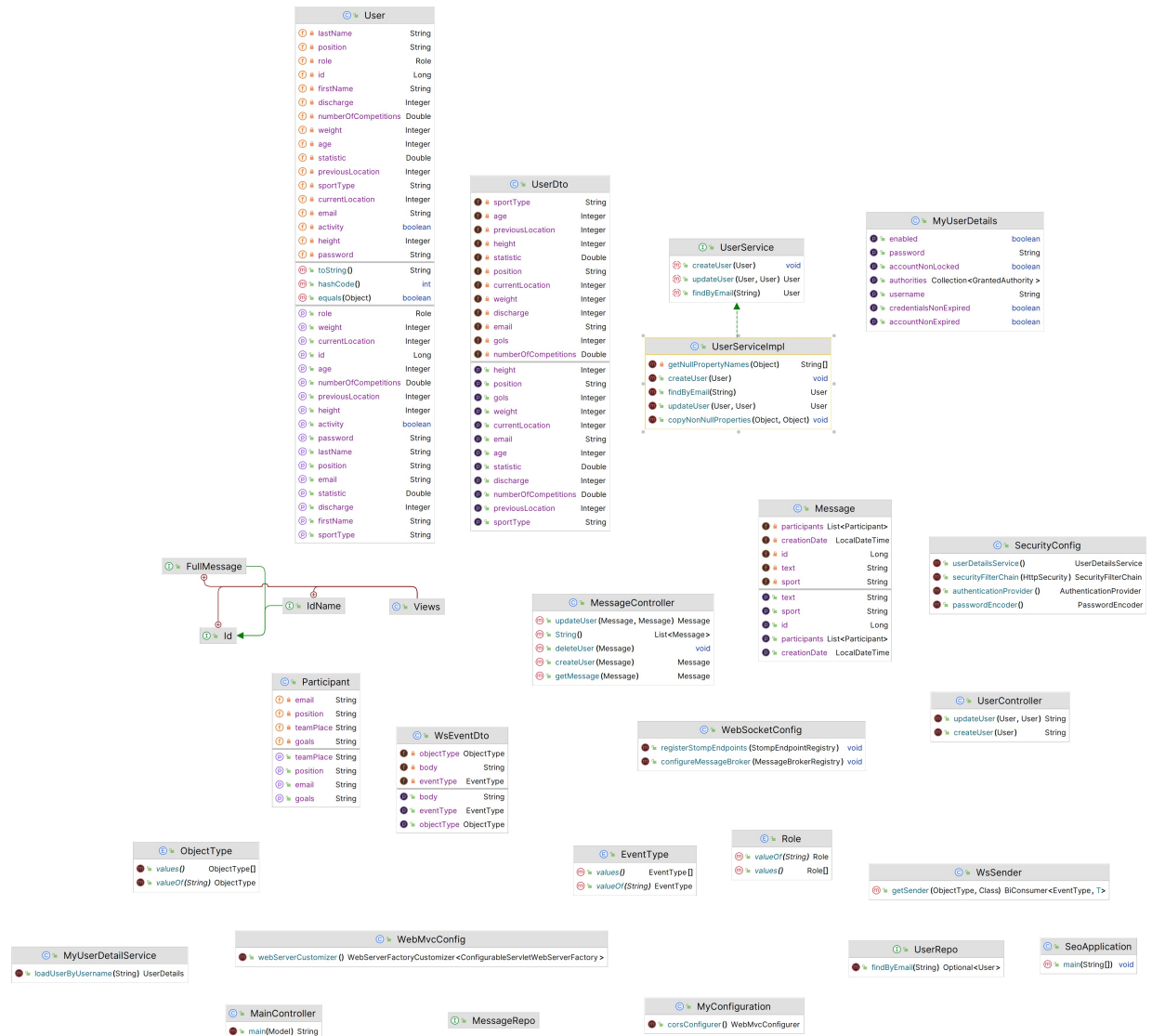
ПРИЛОЖЕНИЕ Б.1

Диаграмма прецедентов



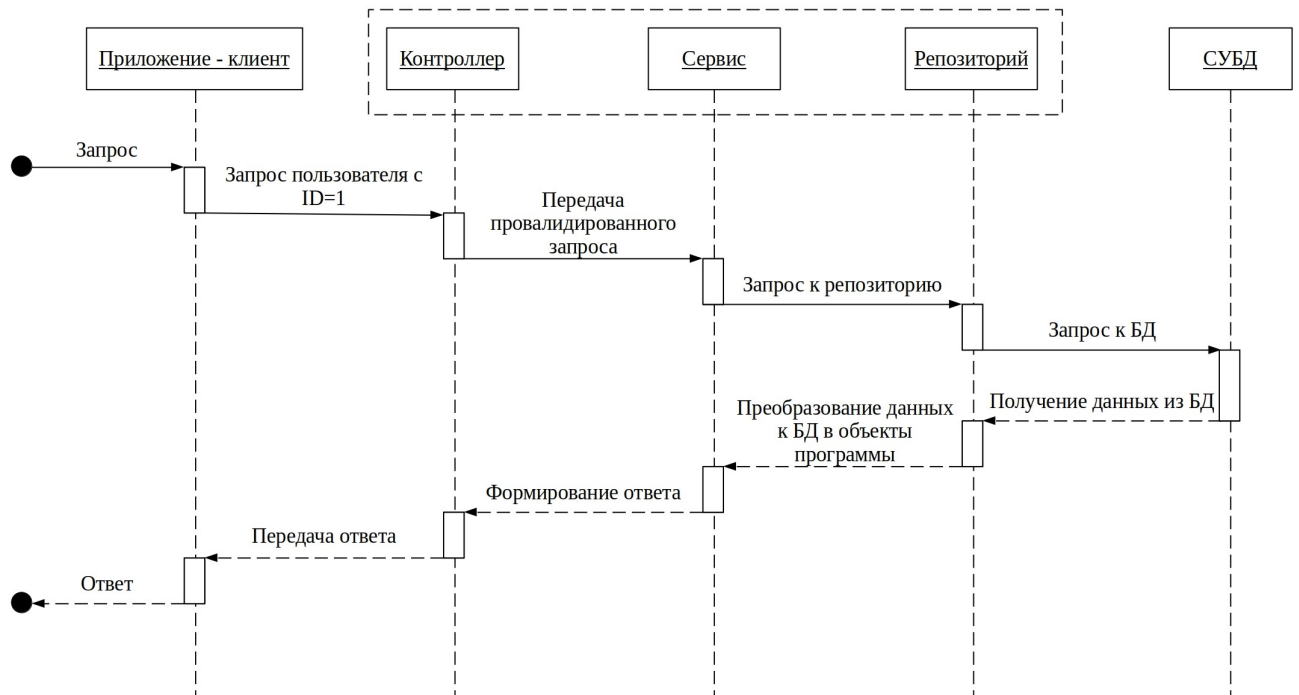
ПРИЛОЖЕНИЕ Б.2

Диаграмма классов



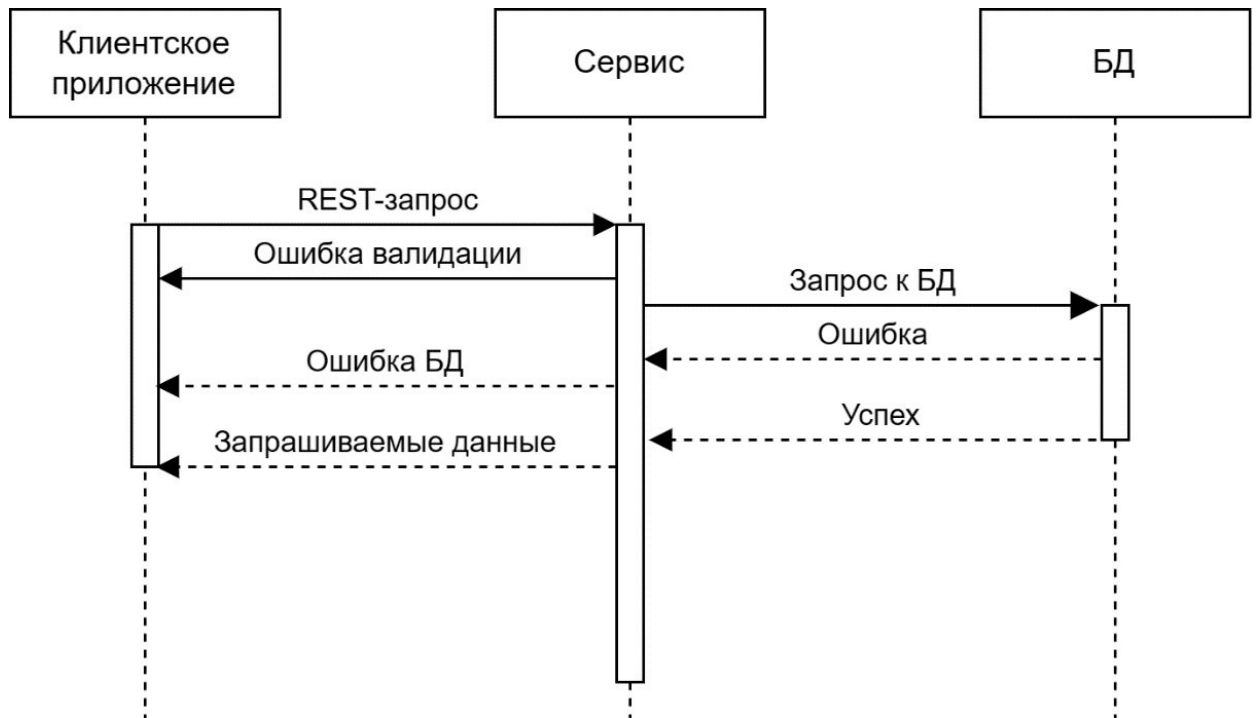
ПРИЛОЖЕНИЕ Б.3

Диаграмма последовательностей



ПРИЛОЖЕНИЕ Б.4

Диаграмма взаимодействия



ПРИЛОЖЕНИЕ В

(обязательное)

Информационная система сбора и анализа данных о спортивных
достижениях обучающихся детско-юношеских спортивных школ

Презентация

ИНФОРМАЦИОННАЯ СИСТЕМА СБОРА И АНАЛИЗА ДАННЫХ О СПОРТИВНЫХ ДОСТИЖЕНИЯХ ОБУЧАЮЩИХСЯ ДЕТСКО- ЮНОШЕСКИХ СПОРТИВНЫХ ШКОЛ

Выполнил студент гр.22ВВИм1

Здерев С.Г.

Научный руководитель:

к.т.н., доцент Синев М.П.

г.Пенза, 2024 г.

- **Предмет исследования:** предметом исследования является разработка информационной системы, предназначенной для эффективного сбора, хранения, обработки и анализа данных о спортивных достижениях обучающихся в детско-юношеских спортивных школах
- **Задача исследования:**
 - сбор и анализ обучающих данных,
 - реализация серверной и клиентской части веб приложения для удобной работы с информационной системой и выбор нейронной сети для обучения,
 - предобработка собранных данных и обучение нейронной сети,
 - тестирование информационной системы

Актуальность и мотивация

- **Устаревшие методы:** В большинстве детско-юношеских спортивных школ данные до сих пор собираются вручную или с помощью устаревших систем, которые не могут обеспечить точность и эффективность анализа.
- **Фрагментированность данных:** Разрозненные данные, которые хранятся в разных форматах и местах, затрудняют их анализ и использование для принятия решений.
- **Недостаток аналитики:** Отсутствие современных инструментов для аналитики затрудняет получение полезных инсайтов из данных и соответственно мешает развитию спортсменов и улучшению тренировочных процессов.

Почему это важно

- **Повышение точности данных:** Автоматизация сбора данных позволит уменьшить количество ошибок и упростить процесс их введения.
- **Улучшение анализа и прогнозирования:** Современные методы анализа данных и машинного обучения позволят лучше понимать прогресс спортсменов и прогнозировать их результаты.
- **Эффективность управления:** Система поможет тренерам и администраторам принимать более обоснованные решения, повысит общую эффективность управления спортивной школой.
- **Спортсмены:** Получат более индивидуализированный подход и улучшат свои результаты.
- **Тренеры:** Станут лучше понимать прогресс и слабые стороны каждого ученика, что позволит эффективнее планировать тренировки.
- **Администраторы:** Смогут оперативно получать необходимые данные для принятия управленческих решений.

Обзор архитектуры

Система состоит из нескольких ключевых компонентов, каждый из которых выполняет определенные функции. Выстроив архитектуру, мы можем более эффективно управлять данными и предоставить пользователям удобный интерфейс для работы.

Бэкэнд:

- **Серверная часть** на базе Java Spring, отвечает за обработку запросов, управление данными и взаимодействие с базой данных.
- **База данных:** для хранения всех данных о спортивных достижениях, результатах тренировок, личных данных спортсменов и т.д. Используется реляционная база данных PostgreSQL



Фронтэнд:

- **Пользовательский интерфейс** на базе Vue3, обеспечивает интерактивные и динамичные веб-страницы для ввода и визуализации данных.
- **Сборка проекта** с использованием Yarn и Webpack для эффективного управления зависимостями и организации кода.



Аналитика данных:

- **Алгоритмы машинного обучения** на Python, отвечают за анализ, предобработку данных и прогнозирование результатов.

Использованные технологии

- **Spring Boot:** Используется для разработки надежной и масштабируемой серверной части системы.
- **Vue3:** Обеспечивает удобный интерфейс пользователя с высокой интерактивностью.
- **Yarn и Webpack:** Используются для управления и сборки фронтэнд части системы, что упрощает разработку и тестирование.
- **Python:** Основной язык для аналитики данных и машинного обучения, использование библиотек позволяет эффективно анализировать данные и строить прогнозы.

Введение в процесс обучения модели

Основные Шаги:

1. Загрузка данных.

- Импортируем необходимые библиотеки и загружаем данные из файла 'dataset.csv'.
- Это позволяет нам начать процесс изучения данных и их подготовки.

2. Предобработка данных.

- Заполняем пропущенные значения, чтобы избежать ошибок в процессе обучения.
- Кодировем категориальные признаки, чтобы их можно было использовать в модели машинного обучения.

```
# Загрузка данных
df = pd.read_csv('dataset.csv')

# Заполнение пропущенных значений и кодирование категориальных признаков
df['Количество голов'].fillna(0, inplace=True)
df['Количество соревнований'].fillna(0, inplace=True)
df['Вид спорта'].fillna('Unknown', inplace=True)
df['Позиция'].fillna('Unknown', inplace=True)

label_encoder_sport = LabelEncoder()
label_encoder_position = LabelEncoder()
df['Вид спорта'] = label_encoder_sport.fit_transform(df['Вид спорта'])
df['Позиция'] = label_encoder_position.fit_transform(df['Позиция'])
```

Разделение и нормализация данных

Основные Шаги:

1. Разделение данных.

- Разделяем данные на признаки (X) и целевую переменную (y).
- Разделяем данные на обучающую, валидационную и тестовую выборки.

2. Нормализация данных.

- Используем StandardScaler для числовых данных.
- Применяем ColumnTransformer для обработки всех признаков с различными типами данных.

```
# Разделение данных на признаки и целевую переменную
X = df.drop('Коэффициент статистики', axis=1)
y = df['Коэффициент статистики']

# Разделение данных на обучающую, валидационную и тестовую выборки
X_train, X_temp, y_train, y_temp = train_test_split(X, y,
                                                    test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
                                                test_size=0.5, random_state=42)

# Нормализация данных
numeric_features = X_train.select_dtypes(include=[np.number]).columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat_sport', 'passthrough', ['Вид спорта']),
        ('cat_position', 'passthrough', ['Позиция'])
    ])
X_train = preprocessor.fit_transform(X_train)
X_val = preprocessor.transform(X_val)
X_test = preprocessor.transform(X_test)
```

Создание и обучение модели

Этот слайд детализирует процесс создания модели машинного обучения на базе нейронных сетей и показывает, как мы её обучаем, используя различные техники для улучшения эффективности и точности.

1. Создание модели.

- Построение модели с несколькими полносвязанными слоями и активацией 'ReLU'.
- Компиляция модели с использованием функции потерь 'mean_squared_error' и оптимизатора 'Adam'.

2. Обучение модели.

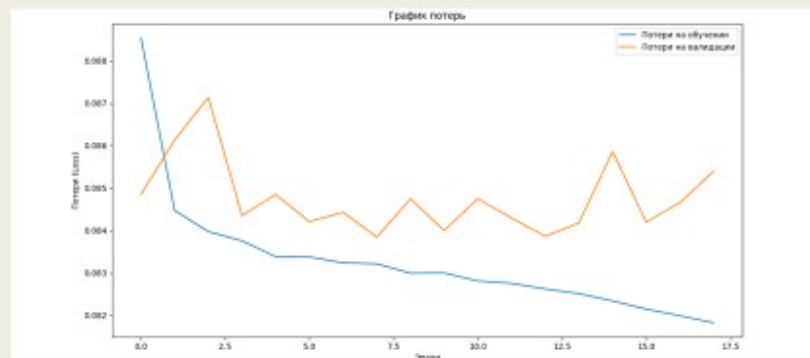
- Использование обратного вызова EarlyStopping для предотвращения переобучения.
- Применение ModelCheckpoint для сохранения наилучшей версии модели.
- Обучение модели с сохранением истории обучения.

```
# Создание функции для построения модели
def build_model():
    model = Sequential()
    model.add(Dense(128, input_dim=X_train.shape[1],
        activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# Обучение модели
callbacks = [EarlyStopping(patience=10),
    ModelCheckpoint('best_model.keras', save_best_only=True)]
model = build_model()
history = model.fit(X_train, y_train, validation_data=(X_val, y_val),
    epochs=100, batch_size=32, callbacks=callbacks)
```

График потерь

- Данный график позволяет отслеживать динамику обучения и помогает выявить возможные проблемы переобучения или недообучения модели.



UML диаграмма взаимодействия

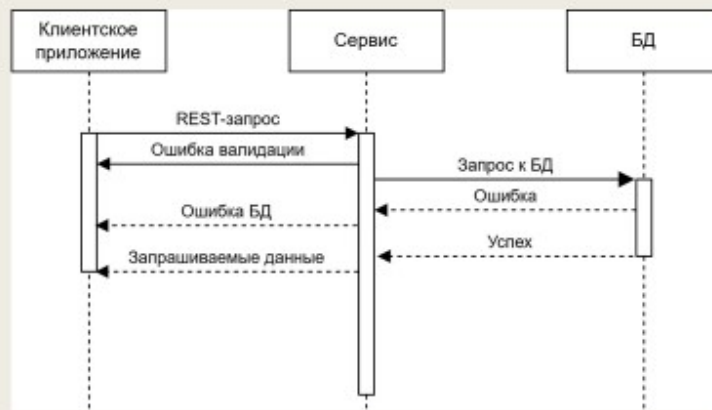
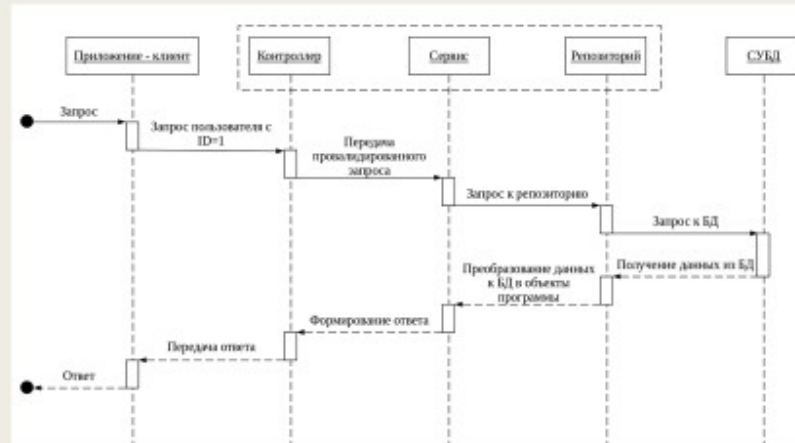


Диаграмма последовательностей



Интерфейс инфомрационной системы

Информационная система сбора и анализа данных

Please sign in

Username

Password

Sign in

Информационная система сбора и анализа данных

Имя игрока

Выбор команды

Создать

Уровень сложности: выбрать

Уровень сложности: выбрать

Отправить: выбрать

Уровень сложности

Информационная система сбора и анализа данных

Добавить участников

Участник 1

Email

Место команды

Количество голов

СОЗДАТЬ УЧАСТНИКА

СОЗДАТЬ

Информационная система сбора и анализа данных

Профиль пользователя

Имя
Александр

Фамилия
Иванов

Email
alex@yandex.ru

Роль
Админ

Вид спорта

Возраст

Рост

Вес

Результат

Профиль пользователя

Имя
Александр

Фамилия
Иванов

Email
alex@yandex.ru

Роль
Админ

Коэффициент статистики

Вид спорта
Футбол

Возраст
20

Рост
170

Вес
70

Результат
100