

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

<i>Факультет</i>	<i>Кафедра</i>
вычислительной техники	МОиПЭВМ
Направление подготовки	09.03.04 «Программная инженерия»
Направленность	Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**«Веб-приложение для организации и
проведения турниров по компьютерным
играм»**

Студент	_____	Смирнов Илья Алексеевич
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой	_____	Козлов А.Ю.
	(подпись)	(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК	_____	Попова Н.А.
	(подпись)	(фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов «26»
декабря 2025 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Смирнов Илья Алексеевич гр. 22ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Веб-приложение для организации и проведения турниров по компьютерным играм
- Тема утверждена приказом ПГУ № _____ от « _____ » _____ 20 _____
3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Такташкин Д.В.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Назначение: веб-приложение для организации и проведения турниров предназначены для оптимизации процесса организации турнира и участия в нем.
Среда выполнения: операционная система, поддерживающая работу Docker
Языковые средства разработки: C#, UML, TypeScript
Инструментальные средства разработки: JetBrains Rider, Docker Compose, DBeaver, Drawio, Swagger UI
Технологии разработки: микросервисная архитектура, ООП, REST API, RabbitMQ, JWT-аутентификация Feature-Sliced Design
Основные функции приложения:
а) регистрация и авторизация пользователей;
б) просмотр списка турниров и информации о турнире;
в) создание и управление турнирами;
г) регистрация игроков на турниры;
д) автоматическое формирование команд;
е) расчет и обновление рейтинга игроков;

<i>ж) просмотр профиля пользователя и списка его турниров;</i>
<i>и) управление пользователями и турнирами администратором;</i>
<i>к) отправка уведомлений участникам турниров.</i>
<i>л) построение и ведение турнирной сетки;</i>

5. Перечень подлежащих разработке вопросов:

<i>1) Анализ предметной области и постановка задачи</i>
<i>2) Проектирование программных средств</i>
<i>3) Реализация программных средств</i>
<i>4) Контроль качества программного обеспечения</i>
<i>5) Оформление пояснительной записки</i>

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 161 лист, 48 рисунков, 53 таблиц, 61 использованный источник, 8 приложений.

ВЕБ-ПРИЛОЖЕНИЕ, ОРГАНИЗАЦИЯ ТУРНИРОВ, КОМПЬЮТЕРНЫЕ ИГРЫ, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, ASP.NET CORE, C#, REACT, TYPESCRIPT, POSTGRESQL, ENTITY FRAMEWORK CORE, RABBITMQ, JWT, DOCKER, NGINX, FEATURE-SLICED DESIGN, UML, ТЕСТИРОВАНИЕ ЧЕРНЫМ ЯЩИКОМ, МОДУЛЬНОЕ ТЕСТИРОВАНИЕ, МЕТРИКИ КОДА

Объектом разработки является веб-приложение для организации и проведения турниров по компьютерным играм.

Цель работы – разработка программного обеспечения, обеспечивающего регистрацию пользователей, создание и управление турнирами, регистрацию участников, автоматическое формирование команд, фиксацию результатов матчей, расчет рейтинга игроков и отправку уведомлений.

Серверная часть реализована на языке C# с использованием платформы ASP.NET Core. Клиентская часть реализована с использованием React и TypeScript. В качестве системы управления базами данных используется PostgreSQL, а взаимодействие с базами данных выполняется с помощью Entity Framework Core. Асинхронный обмен данными между сервисами реализован с использованием брокера сообщений RabbitMQ.

Программное обеспечение разработано на основе микросервисной архитектуры. Приложение предназначено для использования через современные веб-браузеры. Развертывание компонентов системы выполняется с использованием Docker и nginx.

					ПГУ 09.03.04 – 08BP221.20 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата	«Веб-приложение для организации и проведения турниров по компьютерным играм». Пояснительная записка	Лит.	Лист	Листов
Разраб.		Смирнов И.А..					4	161
Пров.		Такташкин Д.В.						
Н. контр.		Такташкин Д.В.				Группа 22ВП1		
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	8
1 Анализ предметной области и требований.....	10
1.1 Анализ предметной области	10
1.2 Анализ программных продуктов.....	11
1.2.1 Сайт «Challonge»	11
1.2.2 Сайт «The Tournament»	12
1.2.3 Сайт «Battlefy»	12
1.2.4 Сравнение аналогов.....	13
1.3 Анализ функциональных требований.....	14
1.4 Анализ нефункциональных требований.....	24
1.5 Планирование разработки и оценка бюджета	25
2 Проектирование веб-приложения для организации и проведения турниров по компьютерным играм	29
2.1 Выбор технологий разработки	29
2.2 Проектирование серверной части веб-приложения	30
2.2.1 Проектирование архитектуры серверной части веб-приложения.....	30
2.3 Проектирование модели данных.....	36
2.3.1 Проектирование концептуальной модели базы данных.....	36
2.3.2 Проектирование логических моделей баз данных.....	39
2.4 Проектирование системы внутреннего рейтинга платформы	45
2.4.1 Проектирование процесса формирования команд.....	45
2.4.2 Проектирование процесса изменения рейтинга игрока	49
2.5 Проектирование клиентской части веб-приложения.....	52
2.5.1 Проектирование структуры клиентской части.....	52
2.5.2 Проектирование пользовательского интерфейса.....	54

3 Реализация веб-приложения для организации и проведения турниров по компьютерным играм	60
3.1 Реализация микросервисов	62
3.1.1 Реализация асинхронного взаимодействия между сервисами	69
3.2 Реализация алгоритмов системы внутреннего рейтинга платформы	74
3.3 Реализация баз данных	79
3.3.1 Реализация базы данных сервиса управления пользователями	80
3.3.2 Реализация базы данных сервиса управления турнирами	81
3.3.3 Реализация базы данных сервиса рейтинга и статистики	82
3.3.4 Реализация базы данных сервиса уведомлений	83
3.4 Реализация клиентской части приложения	83
4 Контроль качества программного продукта	88
4.1 Тестирование черным ящиком	88
4.1.1 Тестирование с помощью юнит-тестов	90
4.2 Оценка кода с помощью метрик	93
4.2.1 Обоснование выбора метрик кода	93
4.2.2 Метрика SLOC	94
4.2.3 Метрика Холстеда	96
Заключение	100
Список использованных источников	102
Приложение А (рекомендуемое) Спецификация логических моделей баз данных	109
Приложение Б (рекомендуемое) Таблицы эндпоинтов микросервисов	116
Приложение В (рекомендуемое) Таблицы разделения классов по слоям в микросервисах	119
Приложение Г (обязательное) Код программы	123

Приложение Д (рекомендуемое) Спецификация физических моделей баз данных	142
Приложение Е (рекомендуемое) Графическое представление классов эквивалентности	149
Приложение Ж (рекомендуемое) Код юнит-тестов	151
Приложение И (рекомендуемое) Справка об апробации.....	160

Введение

В современном мире цифровые технологии оказывают значительное влияние на экономику, образование, развлечения и другие аспекты жизни человека. По информации на конец 2025 года интернетом в России пользовались 136 миллионов человек, что составляет 94,4 процента [1].

Одной из сфер жизни, на которую оказало наибольшее влияние развитие интернета и цифровых технологий, стало развлечение и проведение досуга. Компьютерные игры стали неотъемлемой частью миллионов людей. С 2022 по 2025 годы доля населения России, которая играет в видеоигры ежедневно, выросла с 23 % до 34 %, а общее количество игроков на момент 2026 года составляет до 110 миллионов человек или 81 % населения страны [2].

Более того, видеоигры способны оказывать положительное влияние на когнитивные навыки и моторику, а также снимать стресс и улучшать как психологическое, так и физическое самочувствие [3].

Учитывая темпы развития и масштабы данной отрасли, компьютерные игры уже давно не являются обычным развлечением, а стали полноценными спортивными дисциплинами, которые признаются многими странами на официальном уровне, в том числе и в России. В соответствии с приказом Министерства спорта от 29 апреля 2016 г. № 470 «компьютерный спорт» был признан и включен в реестр видов спорта [4].

Вследствие этого появляется все больше желающих заниматься компьютерными играми на профессиональной основе. А большое внимание к данной отрасли позволяет проводить масштабные мероприятия, посвященные соревнованиям по компьютерным играм, и дает возможность игрокам реализовывать свой потенциал.

В России развитие компьютерного спорта имеет под собой организационную и нормативную основу и осуществляется как при поддержке частных и коммерческих лиц, так и Федерацией компьютерного спорта России (ФКС). По приказу Министерства спорта ФКС была аккредитована в качестве

общероссийской спортивной федерации по виду спорта «Компьютерный спорт» [5].

За 25 лет ФКС было проведено более 10 000 турниров, в которых приняли участие более 2,5 миллионов спортсменов из разных стран с общим призовым фондом более 300 000 000 рублей. С 2024 года ФКС проводит по 7 турниров в день только в онлайн-формате. А в 2025 году только в 9 крупнейших соревнованиях от ФКС приняли участие более 67,5 тыс. игроков из более чем 30 стран мира, суммарная продолжительность событий составила около 200 дней [6].

Текущие данные показывают, что рост внимания к компьютерным играм, в том числе и к соревнованиям по ним, растет из года в год и формирует устойчивый спрос на цифровые средства, способные упростить и ускорить поиск, организацию и проведение игровых событий. Таким образом, разработка веб-приложения для организации и проведения турниров по компьютерным играм является актуальной и практически значимой задачей.

Целью работы является создание программного средства, обеспечивающего удобное взаимодействие игроков, организаторов и администратора на всех этапах проведения соревнований – от поиска турнира и подачи заявки до формирования команд, фиксации матчей и подведения итогов.

Для достижения поставленной цели в работе последовательно решается ряд задач. На первом этапе выполняется анализ предметной области и существующих аналогов, формулируются функциональные и нефункциональные требования к приложению, а также проводится планирование и оценка бюджета разработки. На этапе проектирования разрабатывается архитектура серверной части и модель данных, проектируются система внутреннего рейтинга и клиентская часть приложения. На этапе реализации создаются программные компонентные, соответствующие этапу проектирования. На завершающем этапе проводится тестирование ключевой функциональности приложения и оценка качества программного кода.

1 Анализ предметной области и требований

1.1 Анализ предметной области

В рамках разрабатываемого веб-приложения необходимо обеспечить простой и удобный способ взаимодействия между игроками и организаторами турниров. Основной целью разработки является создание веб-приложения, способного упростить процесс организации и проведения соревнований, а также участия в них.

Разрабатываемая система должна предоставлять пользователям возможность быстро находить информацию о предстоящих соревнованиях и принимать в них участие. Пользователь должен иметь возможность зарегистрироваться в системе, найти интересующий его турнир с учетом выбранной дисциплины и подать заявку на участие. После этого у него должна появиться возможность просмотра информации о проведении соревнования, включая турнирную сетку, расписание матчей, результаты встреч и итоги турнира. По итогам турнира рейтинг участников корректируется на основании результатов проведенных матчей с учетом выбранной игровой дисциплины. Также пользователь должен иметь возможность просматривать свой профиль, содержащий информацию об аккаунте, а также покидать турнир до начала его проведения.

Для организаторов турниров приложение должно предоставлять инструменты для быстрого и удобного создания и настройки турниров. Организатору необходимо иметь возможность задавать формат проведения турнира и количество участников, а также вносить итоги матчей и отменять турниры.

Таким образом, в системе должны быть предусмотрены следующие функции:

- регистрация и авторизация пользователей;
- просмотр списка турниров;
- фильтрация турниров;

- просмотр информации о турнире;
- подача заявки на участие в турнире;
- создание турнира;
- указание результатов матчей;
- просмотр профиля;
- автоматическое формирование команд;
- автоматическое обновление рейтинга;
- подтверждение заявок организаторов администратором;
- отмена турнира;
- выход из турнира.

Также в системе должен быть предусмотрен администратор, способный управлять пользователями и турнирами.

1.2 Анализ программных продуктов

На текущий момент уже существуют сервисы для формирования турнирных сеток и сортировки участников, однако, как правило, их функционал ограничивается конкретной задачей и узконаправлен. Рассмотрим наиболее распространенные программные решения в данной предметной области.

1.2.1 Сайт «Challonge»

Первым рассматриваемым веб-приложением является сайт Challonge [7]. Страница со списком групп турниров сайта «challonge.com» представлена на рисунке 1.

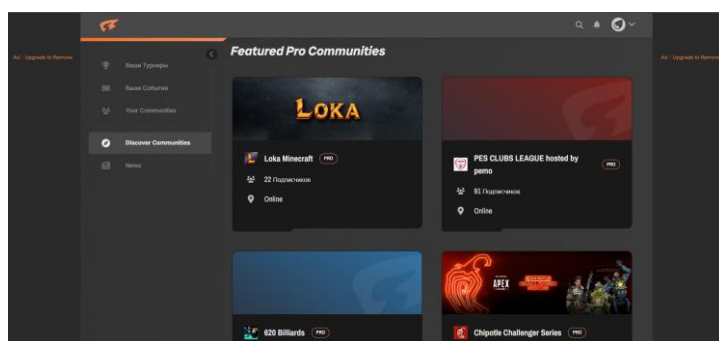


Рисунок 1 – Страница со списком групп турниров сайта «Challonge»

К достоинствам данного сайта можно отнести большое количество форматов турниров и возможность их тонкой настройки, просмотр турниров от других организаторов, а также поддержка различных игровых дисциплин.

К недостаткам данного сайта относятся неполная поддержка русского языка и отсутствие функционала для лиц, являющихся участниками турнира, кроме просмотра его статуса.

1.2.2 Сайт «The Tournament»

Следующим рассматриваемым сайтом является «The Tournament» [8]. При открытии сайта отображается страница с различными вариантами создания турнира.

Главная страница сайта «the-tournament.net» представлена на рисунке 2.

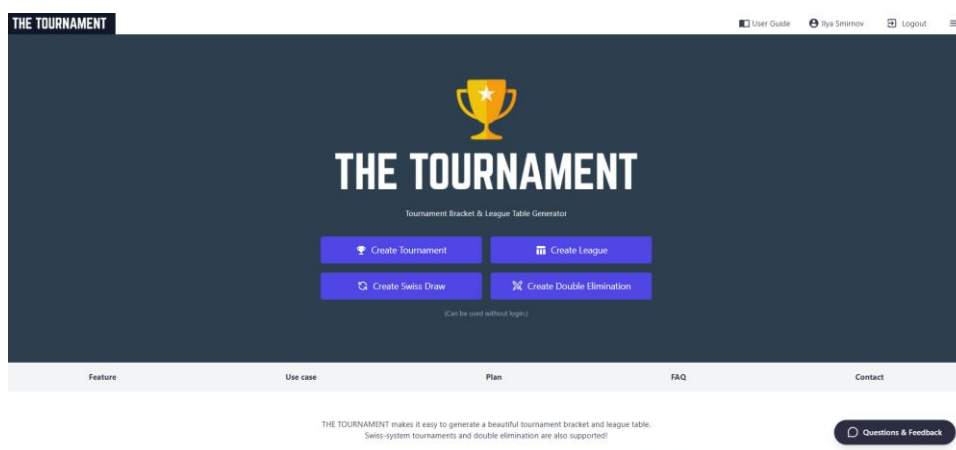


Рисунок 2 – Главная страница сайта «The Tournament»

К достоинствам данного сайта относится возможность создания турниров нескольких типов и их глубокая настройка на этапе создания.

К недостаткам данного сайта можно отнести полное отсутствие русского языка, а в некоторых местах и английского, отсутствие возможности просматривать турниры других организаторов, а также полное отсутствие какого-либо функционала для участников турниров.

1.2.3 Сайт «Battlefy»

Последним рассматриваемым сайтом является «Battlefy» [9].

Главная страница сайта «battlefy.com» представлена на рисунке 3.

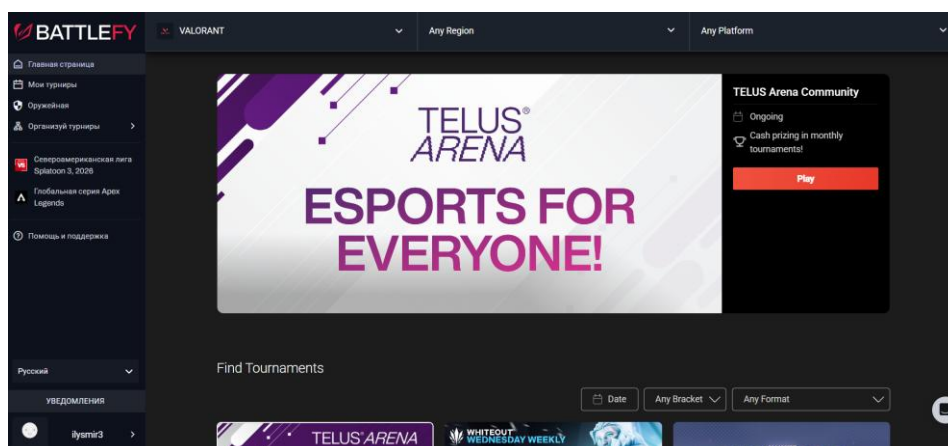


Рисунок 3 – Главная страница сайта «Battlefy»

К достоинствам сайта относится поддержка большого количества языков, включая частично русский, гибкая настройка турниров и возможность принять участие в турнире прямо на сайте.

К недостаткам данного сайта можно отнести отсутствие рейтинговой системы и только три формата проведения турниров.

1.2.4 Сравнение аналогов

На данном этапе был произведен сравнительный анализ рассмотренных ранее сайтов для организации турниров. Результаты анализа представлены в таблице 1.

Таблица 1 – Сравнение аналогов

Сайт	Поддержка русского языка	Поддержка различных форматов турниров	Регистрация на турнир для игроков	Рейтинговая система	Просмотр турниров от других организаторов
Challonge	Неполная	Да	Нет	Нет	Да
The Tournament	Нет	Да	Нет	Нет	Нет
Battlefy	Неполная	Да	Да	Нет	Да
Разрабатываемое веб-приложение	Да	Да	Да	Да	Да

По результатам анализа можно сделать вывод, что существующие сервисы

нацелены в первую очередь на удовлетворение запросов турнирных организаторов, при этом они не предоставляют участникам достаточного набора возможностей для удобного взаимодействия с турниром, что может затруднять вовлечение игроков в организуемое событие. По этой причине было принято решение о разработке собственного сервиса, способного удовлетворить требования всех групп лиц, участвующих в турнире и его организации.

1.3 Анализ функциональных требований

Основными пользователями разрабатываемого веб-приложения выступают игроки, организаторы турнира и администратор. Игрок, используя приложение, должен иметь возможность просмотреть список доступных турниров, отфильтровать их по игровым дисциплинам, ознакомиться с информацией о турнире и подать заявку на участие. После подачи заявки игрок должен иметь возможность отслеживать статус турнира. Помимо этого, игрок должен иметь возможность ознакомиться со своим профилем и со списком турниров, в которых он принимает участие. Также пользователь должен иметь возможность покинуть турнир до начала его проведения.

Организатор турниров, работая с веб-приложением, должен иметь возможность создавать турниры, указывая основные его параметры, включая название, дисциплину, формат проведения и максимально допустимое количество участников. Организатор также должен иметь возможность вносить результаты матчей и отменять турнир. Помимо этого, у него должна быть возможность просматривать список других турниров, список турниров, организованных им, а также свой профиль.

Администратор системы, работая с веб-приложением, должен иметь возможность управлять пользователями и турнирами. Это включает: создание, редактирование, удаление и просмотр профилей пользователей, а также создание, редактирование, удаление и просмотр турниров.

Также все пользователи должны иметь возможность регистрации и авторизации в системе для получения доступа к соответствующему функционалу

приложения.

На основе всего вышесказанного были созданы диаграммы вариантов использования для каждой из возможных ролей пользователя [10]. Наследование функциональных возможностей между ролями пользователя представлено на рисунке 4.

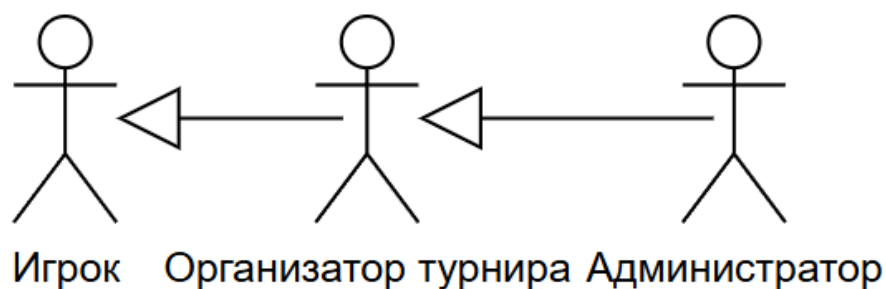


Рисунок 4 – Наследование функциональных возможностей

Рассмотрим подробно диаграммы вариантов использования для каждой из ролей. На рисунке 5 представлена диаграмма вариантов использования для роли «Гость».

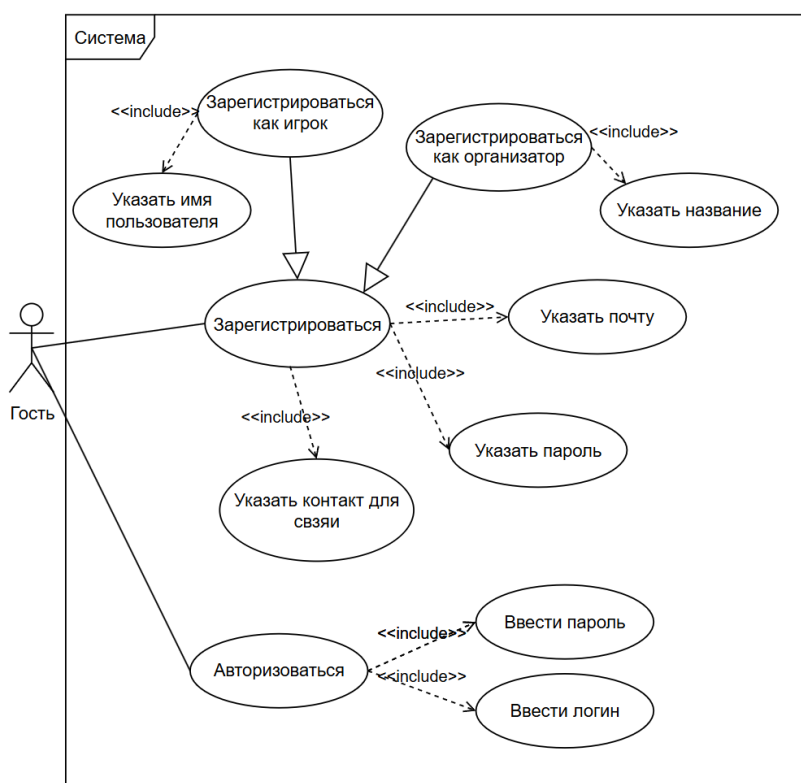


Рисунок 5 – Диаграмма вариантов использования роли «Гость»

Диаграмма вариантов использования для роли «Игрок» представлена на рисунке 6.



Рисунок 6 – Диаграмма вариантов использования роли «Игрок»

Диаграмма вариантов использования для роли «Организатор турнира» представлена на рисунке 7.



Рисунок 7 – Диаграмма вариантов использования роли «Организатор турнира»

Диаграмма вариантов использования для роли «Администратор» представлена на рисунке 8.

Продолжение таблицы 2

Предусловия	Приложение запущено, гость не авторизован в системе
Основной поток	1. Гость открывает веб-приложение. 2. Система отображает форму регистрации. 3. Гость вводит логин, адрес электронной почты и пароль. 4. Пользователь нажимает «Отправить». 5. Система проверяет корректность введенных данных 6. Система проверяет, существует ли пользователь с такими данными. 7. Система сохраняет новую учетную запись. 8. Система выводит сообщение об успешной регистрации. 9. Система перенаправляет пользователя на страницу авторизации.
Постусловия	Учетная запись пользователя создана.
Альтернативные потоки	6а. Если данные введены некорректно, система выводит сообщение об ошибке. 7а. Если пользователь с такими данными уже существует, система выводит сообщение об ошибке.

Таблица 3 – Прецедент «Зарегистрироваться на турнир»

Наименование	Зарегистрироваться на турнир
Идентификатор прецедента	ID: 2
Краткое описание	Пользователь регистрируется на турнир
Главные акторы	Пользователь
Второстепенные акторы	Система
Предусловия	Приложение запущено, пользователь авторизован в системе, турнир существует, регистрация на турнир доступна
Основной поток	1. Пользователь открывает веб-приложение на странице турниров. 2. Система отображает список доступных турниров. 3. Пользователь выбирает нужный турнир. 4. Система отображает информацию о выбранном турнире. 5. Пользователь нажимает кнопку «Зарегистрироваться на турнир». 6. Система проверяет, может ли пользователь быть зарегистрирован на данный турнир. 7. Система обновляет информацию о турнире и пользователе. 8. Система выводит сообщение об успешной регистрации на турнир.
Постусловия	Пользователь зарегистрирован на турнир.

Продолжение таблицы 3

Альтернативные потоки	6а. Если время регистрации на турнир вышло, система выводит соответствующее сообщение. 6б. Если лимит участников превышен, система выводит соответствующее сообщение.
-----------------------	--

Также на основе диаграмм вариантов использования составим спецификации для основных прецедентов, относящихся к организаторам турниров: «Создать турнир» (таблица 4) и «Отменить турнир» (таблица 5).

Таблица 4 – Прецедент «Создать турнир»

Наименование	Создать турнир
Идентификатор прецедента	ID: 3
Краткое описание	Организатор турнира создает новый турнир
Главные акторы	Организатор турнира
Второстепенные акторы	Система
Предусловия	Приложение запущено, организатор турнира авторизован в системе
Основной поток	1. Организатор турнира открывает веб-приложение. 2. Организатор турнира переходит в раздел создания турнира. 3. Система отображает форму создания турнира. 4. Организатор турнира указывает название турнира, игровую дисциплину, формат проведения, дату проведения и количество участников. 5. Организатор турнира нажимает кнопку «Создать турнир». 6. Система проверяет корректность введенных данных. 7. Система проверяет наличие турниров с таким же названием. 8. Система сохраняет информацию о турнире. 9. Система выводит сообщение об успешном создании турнира.
Постусловия	Новый турнир создан и сохранён в системе.
Альтернативные потоки	8а. Если данные были введены некорректно, система выводит соответствующее сообщение. 8б. Если турнир с таким названием уже существует, система выводит соответствующее сообщение.

Таблица 5 – Прецедент «Отменить турнир»

Наименование	Отменить турнир
Идентификатор прецедента	ID: 4
Краткое описание	Организатор турнира отменяет турнир, который был им создан ранее
Главные акторы	Организатор турнира
Второстепенные акторы	Система
Предусловия	Приложение запущено, организатор турнира авторизован в системе, турнир существует и не завершен
Основной поток	1. Организатор турнира открывает веб-приложение. 2. Организатор турнира переходит в список организованных турниров. 3. Система отображает список турниров, созданных данным организатором. 4. Организатор турнира выбирает турнир, который необходимо отменить. 5. Система отображает информацию о выбранном турнире. 6. Организатор турнира нажимает кнопку «Отменить турнир». 7. Система запрашивает подтверждение отмены. 8. Организатор турнира подтверждает отмену турнира. 9. Система отменяет выбранный турнир. 10. Система выводит сообщение об успешной отмене турнира.
Постусловия	Выбранный турнир отменен.
Альтернативные потоки	8а. Если организатор турнира не подтверждает отмену, система возвращает его на страницу турнира без внесения изменений.

Рассмотрим ключевые варианты использования подробнее. Для вариантов использования «Зарегистрироваться на турнир» и «Создать турнир» составлены диаграммы деятельности.

Диаграмма деятельности наглядно отражает последовательность действий и точки принятия решений в рамках варианта использования. Процесс создания турнира начинается с ввода организатором основных параметров турнира, после чего система проверяет их корректность и при успешной проверке создаёт турнир, а при обнаружении ошибок возвращает организатора к их исправлению. Процесс регистрации на турнир включает выбор пользователем доступного турнира, проверку условий участия и добавление пользователя в список участников либо вывод сообщения о невозможности регистрации.

Диаграмма деятельности варианта использования создания турнира организатором представлена на рисунке 9, а диаграмма деятельности варианта использования регистрации пользователя на турнир представлена на рисунке 10.

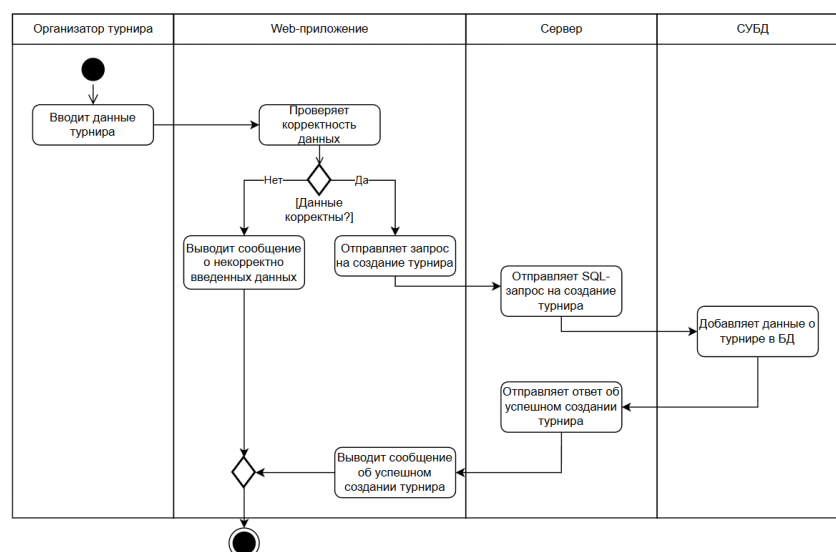


Рисунок 9 – Диаграмма деятельности варианта использования «Создать турнир»

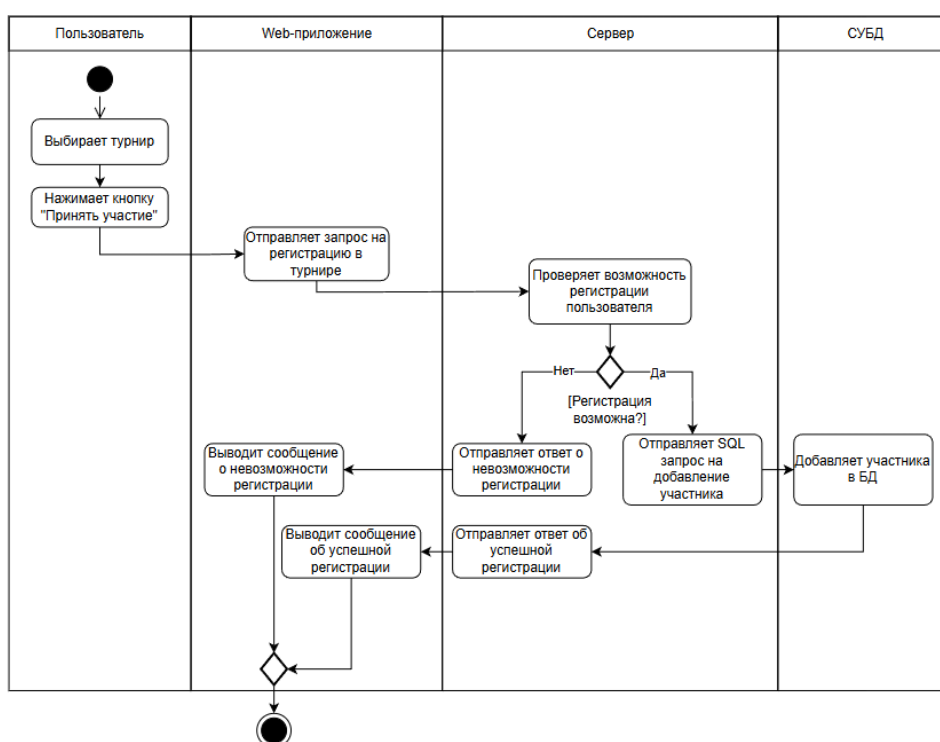


Рисунок 10 – Диаграмма деятельности варианта использования «Зарегистрироваться на турнир»

Также для варианта использования «Отменить турнир» и для варианта

использования «Посмотреть профиль», являющихся одними из основных в разрабатываемом приложении, были составлены диаграммы последовательности. Диаграмма последовательности отмены турнира представлена на рисунке 11, а диаграмма последовательности просмотра профиля представлена на рисунке 12.

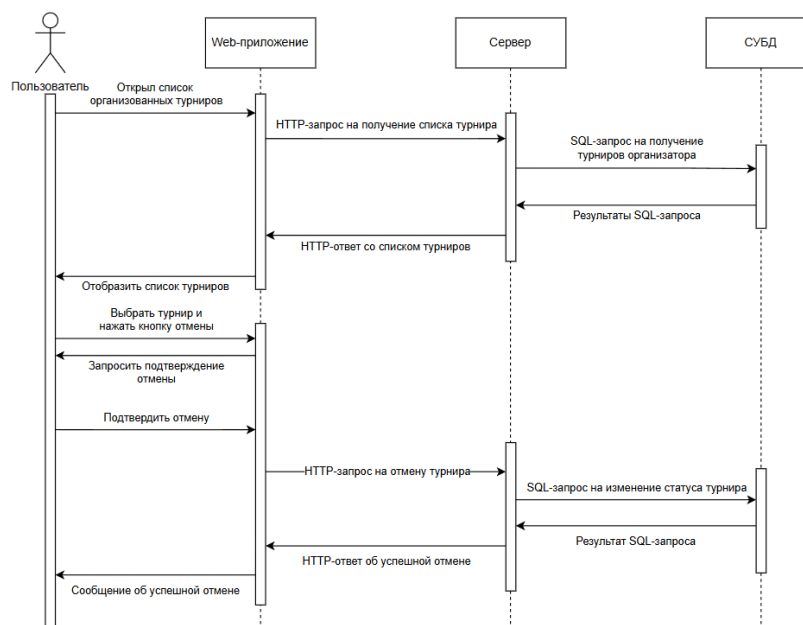


Рисунок 11 – Диаграмма последовательности варианта использования «Отменить турнир»

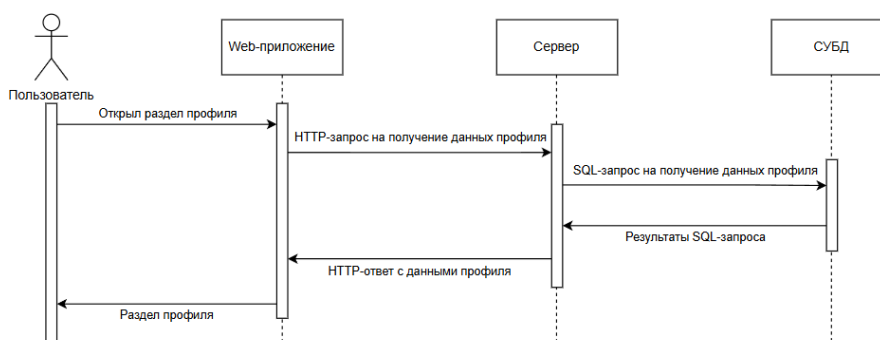


Рисунок 12 – Диаграмма последовательности варианта использования «Посмотреть профиль»

Таким образом, на данном этапе был произведен анализ функциональных требований. Были определены ключевые роли пользователей, построены

диаграммы вариантов использования для каждой из ролей, а также исследованы более подробно одни из ключевых вариантов использования с помощью диаграмм деятельности и последовательности.

1.4 Анализ нефункциональных требований

К разрабатываемому веб-приложению были выдвинуты следующие нефункциональные требования:

- поддержка работы в современных браузерах Google Chrome, Microsoft Edge, Mozilla Firefox и аналогичных;
- адаптивность пользовательского интерфейса для корректного отображения на дисплеях с различным разрешением;
- возможность масштабирования системы при увеличении числа пользователей и турниров;
- хранение паролей в хешированном виде, разграничение доступа по ролям.

Целевые устройства для работы с клиентской частью приложения должны обладать следующими характеристиками:

- наличие современного веб-браузера;
- минимум 2 Гб оперативной памяти;
- наличие интернет-соединения;
- поддержка JavaScript в браузере.

К оборудованию для развёртывания серверной части были выдвинуты следующие нефункциональные требования:

- наличие установленного Docker для развёртывания приложения;
- серверная часть приложения разворачивается в контейнерах, поэтому для её запуска может использоваться любая операционная система, поддерживающая Docker и Docker Compose;
- процессор с тактовой частотой не менее 2 ГГц и количеством ядер не менее 2;

- минимум 4 Гб оперативной памяти;
- минимальный объем свободного дискового пространства – 10 Гб;
- наличие интернет-соединения со скоростью не менее 100 Мбит/с.

1.5 Планирование разработки и оценка бюджета

Для оценки временных и материальных затрат разработки веб-приложения было проведено планирование и бюджетирование проекта с использованием программы Microsoft Office Project 2007. В процессе планирования разработки веб-приложения было выделено пять основных этапов:

- анализ предметной области и постановка задачи;
- проектирование программного обеспечения;
- реализация программного обеспечения;
- тестирование программных средств;
- оформление документации.

Каждый этап включает набор подзадач, необходимых для перехода к следующему этапу. Краткое описание задач каждого этапа приведено на рисунке 13.

Название задачи	Длительность	Начало	Окончание	Предшественники
Начало проекта	0 дней	Вт 01.12.26	Вт 01.12.26	
⇒ Анализ предметной области и постановка задачи	16 дней	Вт 01.12.26	Вт 22.12.26	1
Анализ предметной области	3 дней	Вт 01.12.26	Чт 03.12.26	
Анализ аналогичных решений	2 дней	Пт 04.12.26	Пн 07.12.26	3
Постановка задачи	6 дней	Вт 08.12.26	Вт 15.12.26	4
Анализ функциональных требований	2 дней	Ср 16.12.26	Чт 17.12.26	5
Анализ нефункциональных требований	2 дней	Пт 18.12.26	Пн 21.12.26	6
Планирование разработки и оценка бюджета	1 день	Вт 22.12.26	Вт 22.12.26	7
Завершение анализа	0 дней	Вт 22.12.26	Вт 22.12.26	8
⇒ Проектирование программного обеспечения	13 дней	Ср 23.12.26	Пт 08.01.27	2
Проектирование архитектуры	3 дней	Ср 23.12.26	Пт 25.12.26	9
Проектирование модели данных	3 дней	Пн 28.12.26	Ср 30.12.26	11
Проектирование серверной и алгоритмической части	2 дней	Чт 31.12.26	Пт 01.01.27	12
Проектирование пользовательского интерфейса	3 дней	Пн 04.01.27	Ср 06.01.27	13
Проектирование клиентской части приложения	2 дней	Чт 07.01.27	Пт 08.01.27	14
Завершение проектирования	0 дней	Пт 08.01.27	Пт 08.01.27	15
⇒ Реализация программного обеспечения	24 дней	Пт 08.01.27	Чт 11.02.27	10
Реализация модели данных и слоя доступа к данным	5 дней	Пн 11.01.27	Пт 15.01.27	16
Реализация серверной части приложения	7 дней	Пн 18.01.27	Вт 26.01.27	18
Реализация алгоритмической части приложения	5 дней	Ср 27.01.27	Вт 02.02.27	19
Реализация клиентской части приложения	7 дней	Ср 03.02.27	Чт 11.02.27	20
Завершение реализации	0 дней	Пт 08.01.27	Пт 08.01.27	
⇒ Тестирование программных средств	17 дней	Пн 11.01.27	Вт 02.02.27	
Функциональное тестирование	4 дней	Пн 11.01.27	Чт 14.01.27	22
Оценка метрик покрытия операторов и условий	3 дней	Пт 15.01.27	Вт 19.01.27	24
Оценка метрик качества кода	6 дней	Ср 20.01.27	Ср 27.01.27	25
Модульное тестирование	4 дней	Чт 28.01.27	Вт 02.02.27	26
Завершение тестирования	0 дней	Вт 02.02.27	Вт 02.02.27	27
⇒ Оформление документации	10 дней	Ср 03.02.27	Вт 16.02.27	27
Описание анализа предметной области	2 дней	Ср 03.02.27	Чт 04.02.27	28
Описание проектирования	2 дней	Пт 05.02.27	Пн 08.02.27	30
Описание реализации	2 дней	Вт 09.02.27	Ср 10.02.27	31
Описание тестирования	2 дней	Чт 11.02.27	Пт 12.02.27	32
Оформление итоговой записки	2 дней	Пн 15.02.27	Вт 16.02.27	33
Завершение оформления документации	0 дней	Вт 16.02.27	Вт 16.02.27	34
Конец проекта	0 дней	Вт 16.02.27	Вт 16.02.27	35

Рисунок 13 – Сведения об этапах разработки приложения

Для наглядной визуализации плана разработки была составлена диаграмма Ганта, представленная на рисунке 14.

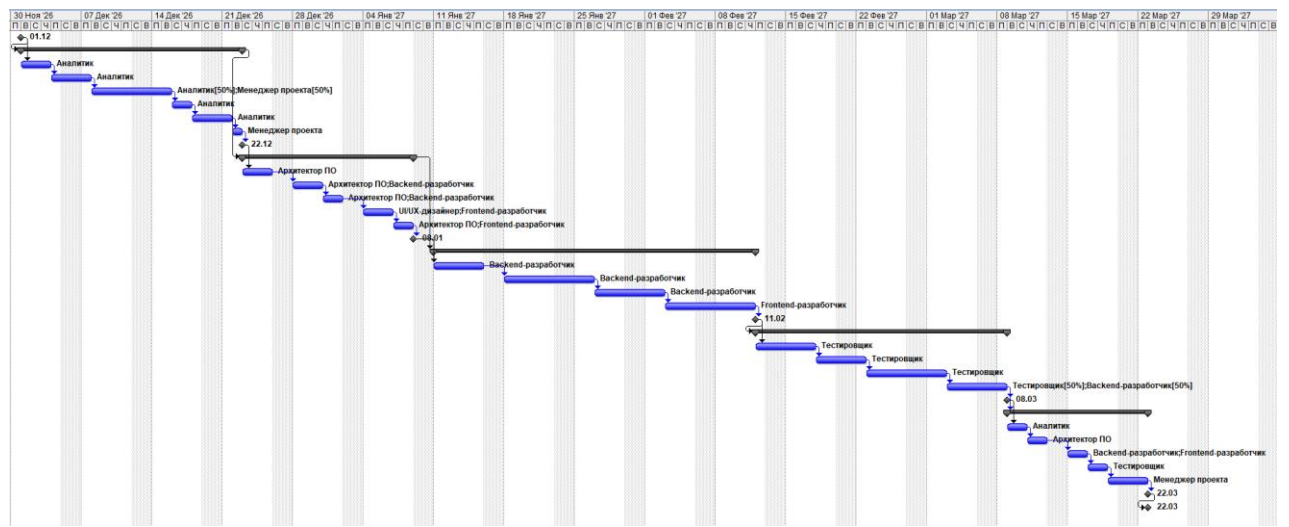


Рисунок 14 – Диаграмма Ганта

Реализация проекта запланирована на период с 1 декабря 2025 года по 3 апреля 2026 года. Исходя из составленного плана можно увидеть, что на реализацию проекта потребуется 80 рабочих дней. Общее время реализации, с учетом выходных составляет 124 календарных дня. Наиболее продолжительным этапом проекта является реализация программного обеспечения (24 дня).

Для оценки бюджета проекта были назначены ресурсы и указаны их почасовые ставки. Стоимость используемых ресурсов представлена на рисунке 15.

Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка
Менеджер проекта	Трудовой		М		100%	368,00 Р/ч
Аналитик	Трудовой		Ан		100%	443,00 Р/ч
Архитектор ПО	Трудовой		Ар		100%	358,00 Р/ч
UI/UX-дизайнер	Трудовой		U		100%	312,00 Р/ч
Backend-разработчик	Трудовой		В		100%	1 200,00 Р/ч
Frontend-разработчик	Трудовой		F		100%	1 000,00 Р/ч
Тестировщик	Трудовой		T		100%	400,00 Р/ч

Рисунок 15 – Список ресурсов

Распределение задач между специалистами продемонстрировано на рисунке 16. Из данного отчета видно, что наибольший уровень трудозатрат в

208 и 136 часов приходится на backend-разработчика и тестировщика соответственно. На третьем месте по количеству затраченных часов находятся аналитик и frontend-разработчик, общее время работы которых составляет по 112 часов.

Ид.	Название ресурса	Трудозатраты				
1	Менеджер проекта	48ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
34	Оформление итоговой записки	100%	16ч	0 дней	Чт 02.04.26	Пн 03.04.26
5	Постановка задачи	50%	24ч	0 дней	Пн 08.12.25	Пн 15.12.25
8	Планирование разработки и оценки	100%	8ч	0 дней	Пн 22.12.25	Пн 22.12.25
2	Аналитик	112ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
3	Анализ предметной области	100%	24ч	0 дней	Пн 01.12.25	Ср 03.12.25
4	Анализ аналогичных решений	100%	16ч	0 дней	Чт 04.12.25	Пт 05.12.25
5	Постановка задачи	50%	24ч	0 дней	Пн 08.12.25	Пн 15.12.25
6	Анализ функциональных требо	100%	16ч	0 дней	Вт 16.12.25	Ср 17.12.25
30	Описание анализа предметной	100%	16ч	0 дней	Пн 23.03.26	Вт 24.03.26
7	Анализ нефункциональных тре	100%	16ч	0 дней	Чт 18.12.25	Пн 19.12.25
3	Архитектор ПО	96ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
11	Проектирование архитектуры	100%	24ч	0 дней	Вт 23.12.25	Чт 25.12.25
12	Проектирование модели данных	100%	24ч	0 дней	Пт 26.12.25	Вт 30.12.25
13	Проектирование серверной и алгоритми	100%	16ч	0 дней	Пн 12.01.26	Вт 13.01.26
15	Проектирование клиентской части прил	100%	16ч	0 дней	Пн 19.01.26	Вт 20.01.26
31	Описание проектирования	100%	16ч	0 дней	Ср 25.03.26	Чт 26.03.26
4	UI/UX-дизайнер	24ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
14	Проектирование пользовательского	100%	24ч	0 дней	Ср 14.01.26	Пт 16.01.26
5	Backend-разработчик	208ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
18	Реализация модели данных и слоя доступ	100%	40ч	0 дней	Ср 21.01.26	Вт 27.01.26
19	Реализация серверной части приложения	100%	56ч	0 дней	Ср 28.01.26	Чт 05.02.26
20	Реализация алгоритмической части прил	100%	40ч	0 дней	Пт 06.02.26	Чт 12.02.26
32	Описание реализации	100%	16ч	0 дней	Пт 27.03.26	Пн 30.03.26
12	Проектирование модели данных	100%	24ч	0 дней	Пт 26.12.25	Вт 30.12.25
13	Проектирование серверной и алгоритми	100%	16ч	0 дней	Пн 12.01.26	Вт 13.01.26
27	Модульное тестирование	50%	16ч	0 дней	Вт 17.03.26	Пт 20.03.26
6	Frontend-разработчик	112ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
21	Реализация клиентской части прилож	100%	56ч	0 дней	Пт 13.02.26	Вт 24.02.26
14	Проектирование пользовательского	100%	24ч	0 дней	Ср 14.01.26	Пт 16.01.26
15	Проектирование клиентской части пр	100%	16ч	0 дней	Пн 19.01.26	Вт 20.01.26
32	Описание реализации	100%	16ч	0 дней	Пт 27.03.26	Пн 30.03.26
7	Тестировщик	136ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
24	Функциональное тестирование	100%	32ч	0 дней	Ср 25.02.26	Пн 02.03.26
25	Оценка метрик покрытия операторо	100%	24ч	0 дней	Вт 03.03.26	Чт 05.03.26
27	Модульное тестирование	50%	16ч	0 дней	Вт 17.03.26	Пт 20.03.26
33	Описание тестирования	100%	16ч	0 дней	Вт 31.03.26	Ср 01.04.26
26	Оценка метрик качества кода	100%	48ч	0 дней	Пт 06.03.26	Пн 16.03.26

Рисунок 16 – Распределение задач по работникам

Также с помощью встроенных средств Microsoft Office Project 2007 была сформирована сводка по проекту, представленная на рисунке 17.

Даты			
Начало:	Пн 01.12.25	Окончание:	Пт 03.04.26
Баз. начало:	Пн 01.12.25	Баз. окончание:	Пт 03.04.26
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
Длительность			
Текущий план:	80 дней	Оставшаяся:	80 дней
Базовый план:	80 дней	Фактическая:	0 дней
Отклонение:	0 дней	Процент завершения:	0%
Трудозатраты			
Текущий план:	736 ч	Оставшаяся:	736 ч
Базовый план:	736 ч	Фактические:	0 ч
Отклонение:	0 ч	Процент завершения:	0%
Затраты			
Текущий план:	523 536,00 Р	Оставшаяся:	523 536,00 Р
Базовый план:	294 000,00 Р	Фактические:	0,00 Р
Отклонение:	229 536,00 Р		
Состояние задачи		Состояние ресурса	
Еще не начатые задачи	36	Трудовые ресурсы:	7
Выполняющиеся задачи	0	Трудовые ресурсы с прев. дост	0
Завершенные задачи:	0	Материальные ресурсы:	0
Итого задач:	36	Итого ресурсов:	7

Рисунок 17 – Сводка по проекту

Для оценки бюджета проекта был сформирован отчет по затратам, представленный на рисунке 18.

Ид.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты
19	Реализация серверной части прило	0,00 Р	Пропорциональное	67 200,00 Р
21	Реализация клиентской части прило	0,00 Р	Пропорциональное	56 000,00 Р
18	Реализация модели данных и слоя л	0,00 Р	Пропорциональное	48 000,00 Р
20	Реализация алгоритмической части	0,00 Р	Пропорциональное	48 000,00 Р
12	Проектирование модели данных	0,00 Р	Пропорциональное	37 392,00 Р
32	Описание реализации	0,00 Р	Пропорциональное	35 200,00 Р
14	Проектирование пользовательского	0,00 Р	Пропорциональное	31 488,00 Р
13	Проектирование серверной и алгори	0,00 Р	Пропорциональное	24 928,00 Р
27	Модульное тестирование	0,00 Р	Пропорциональное	24 000,00 Р
15	Проектирование клиентской части п	0,00 Р	Пропорциональное	21 728,00 Р
5	Постановка задачи	0,00 Р	Пропорциональное	19 464,00 Р
26	Оценка метрик качества кода	0,00 Р	Пропорциональное	19 200,00 Р
24	Функциональное тестирование	0,00 Р	Пропорциональное	12 800,00 Р
3	Анализ предметной области	0,00 Р	Пропорциональное	10 632,00 Р
25	Оценка метрик покрытия операторо	0,00 Р	Пропорциональное	9 600,00 Р
11	Проектирование архитектуры	0,00 Р	Пропорциональное	8 592,00 Р
4	Анализ аналогичных решений	0,00 Р	Пропорциональное	7 088,00 Р
6	Анализ функциональных требовани	0,00 Р	Пропорциональное	7 088,00 Р
7	Анализ нефункциональных требова	0,00 Р	Пропорциональное	7 088,00 Р
30	Описание анализа предметной обле	0,00 Р	Пропорциональное	7 088,00 Р
33	Описание тестирования	0,00 Р	Пропорциональное	6 400,00 Р
34	Оформление итоговой записки	0,00 Р	Пропорциональное	5 888,00 Р
31	Описание проектирования	0,00 Р	Пропорциональное	5 728,00 Р
8	Планирование разработки и оценка	0,00 Р	Пропорциональное	2 944,00 Р
1	Начало проекта	0,00 Р	Пропорциональное	0,00 Р
9	Завершение анализа	0,00 Р	Пропорциональное	0,00 Р
16	Завершение проектирования	0,00 Р	Пропорциональное	0,00 Р
22	Завершение реализации	0,00 Р	Пропорциональное	0,00 Р
28	Завершение тестирования	0,00 Р	Пропорциональное	0,00 Р
35	Завершение оформления документа	0,00 Р	Пропорциональное	0,00 Р
36	Конец проекта	0,00 Р	Пропорциональное	0,00 Р
		0,00 Р		523 536,00 Р

Рисунок 18 – Отчет по затратам

Исходя из данного отчета видно, что наибольшие затраты приходятся на реализацию серверной части приложения, а также на проектирование модели данных и реализацию клиентской части приложения. Данные задачи являются первостепенными при реализации проекта и высокие затраты на них могут считаться оправданными. Общие затраты на весь проект составляют примерно 523 536 рублей, при этом общая трудоемкость проекта составляет 736 человеко-часов.

Таким образом, на данном этапе был произведен анализ нефункциональных требований, а также планирование разработки и оценка бюджета с использованием диаграммы Ганта и программы Microsoft Office Project 2007.

По результату планирования были составлены отчеты, отражающие план проекта и затраты на него.

2 Проектирование веб-приложения для организации и проведения турниров по компьютерным играм

2.1 Выбор технологий разработки

При проектировании веб-приложения для организации и проведения турниров по компьютерным играм был выполнен выбор технологий и инструментальных средств разработки. Они должны обеспечивать реализацию приложения, работу с данными, моделирование проектных решений, обмен сообщениями между компонентами и удобное развертывание.

В качестве технологии разработки был выбран Rational Unified Process (RUP) [11]. Он используется потому, что позволяет связать анализ требований, UML-моделирование, проектирование, реализацию и тестирование в единый итерационный процесс. Это удобно для разрабатываемой системы, так как ее функциональность делится на несколько направлений: авторизацию пользователей, управление турнирами, регистрацию участников, формирование команд, расчет рейтинга и отправку уведомлений.

Для моделирования системы используется язык UML [12]. Он позволяет наглядно описать роли пользователей, варианты использования и последовательность действий в системе. В работе UML применяется для построения диаграмм вариантов использования, диаграмм деятельности и диаграмм последовательности.

Для реализации серверной части выбран язык C# и платформа ASP.NET Core [13]. C# поддерживает объектно-ориентированный подход и строгую типизацию, а ASP.NET Core предоставляет средства для создания веб-API, обработки HTTP-запросов, маршрутизации, внедрения зависимостей и разграничения доступа. Для авторизации пользователей применяется JWT-аутентификация, позволяющая определять права доступа с учетом роли пользователя.

В качестве системы управления базами данных используется PostgreSQL [14]. Она выбрана благодаря поддержке реляционной модели

данных, транзакций, ограничений целостности и индексов, что необходимо для хранения сведений о пользователях, турнирах, матчах, рейтингах и уведомлениях.

Для взаимодействия серверной части с базой данных используется Entity Framework Core [15]. Он позволяет описывать модель данных на уровне классов приложения и изменять структуру базы данных с помощью миграций, что упрощает сопровождение проекта.

Для обмена сообщениями между компонентами приложения используется RabbitMQ [16]. Его преимущество заключается в поддержке асинхронного обмена сообщениями, что позволяет обрабатывать результаты матчей, пересчет рейтинга и формирование уведомлений без жесткой синхронной зависимости между компонентами.

Для развертывания приложения используется Docker [17]. Контейнеризация позволяет изолировать компоненты системы и запускать приложение в заранее описанном окружении. С помощью Docker Compose можно определить состав окружения, включающего серверную часть, базы данных и брокер сообщений.

Выбранные технологии используются далее при проектировании серверной и клиентской частей приложения.

2.2 Проектирование серверной части веб-приложения

2.2.1 Проектирование архитектуры серверной части веб-приложения

Проектирование архитектуры разрабатываемого веб-приложения выполнялось на основе функциональных и нефункциональных требований, сформулированных в первой главе. При проектировании необходимо было обеспечить возможность дальнейшего расширения функциональности, изоляцию ключевых бизнес-подсистем, независимость их сопровождения, а также удобство развития программного продукта.

В качестве основного архитектурного подхода для разрабатываемой

системы была выбрана микросервисная архитектура. Микросервисная архитектура представляет собой разновидность распределённой архитектуры, при которой приложение разделяется на набор независимых сервисов, взаимодействующих между собой посредством сетевых протоколов. Каждый сервис реализует отдельную предметную область или бизнес-процесс и включает все необходимые компоненты для автономной работы, в том числе собственную бизнес-логику и отдельное хранилище данных [18].

Обобщённая топология архитектурного стиля микросервисов представлена на рисунке 19.

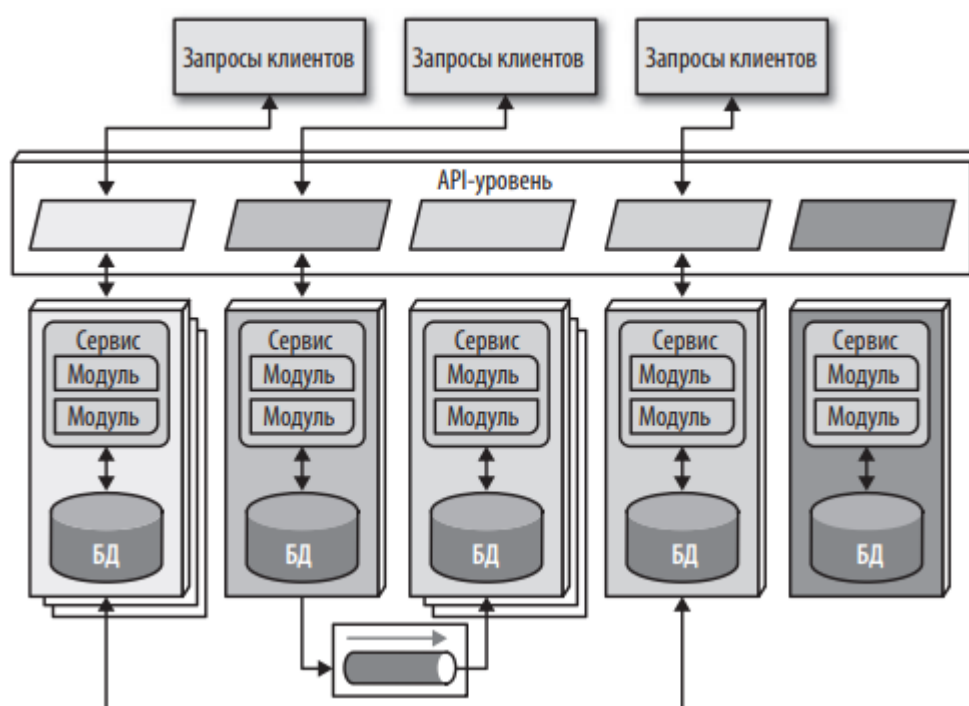


Рисунок 19 – Топология архитектурного стиля микросервисов

Выбор данного архитектурного подхода обусловлен особенностями разрабатываемого веб-приложения. В системе выделяются несколько самостоятельных предметных областей, требующих независимой реализации и сопровождения: управление пользователями, управление турнирами, хранение рейтингов и статистики игроков, а также рассылка уведомлений. Разделение системы на отдельные сервисы позволяет локализовать бизнес-логику каждой из указанных областей в пределах собственного ограниченного контекста.

Ключевыми принципами микросервисной архитектуры являются распределённость, ограниченный контекст, а также изолированность данных. Взаимодействие между сервисами может быть синхронным или асинхронным. Асинхронный обмен через брокер сообщений снижает связанность и повышает устойчивость распределённой системы. Выделенный API-уровень между клиентами и сервисами не является обязательным признаком данной архитектуры и может применяться как дополнительный инфраструктурный компонент, например для маршрутизации запросов.

После выбора общего архитектурного подхода были определены два ключевых аспекта проектирования серверной части: внутренняя структура каждого микросервиса и способ межсервисного взаимодействия.

2.2.1.1 Проектирование внутренней структуры микросервисов

Внутренняя структура микросервисов была спроектирована на основе Clean Architecture. Clean Architecture представляет собой архитектурный подход, при котором бизнес-логика и модель приложения размещаются в центральной части системы, а инфраструктурные и технические детали выносятся во внешние слои [19]. К таким внешним деталям относятся пользовательский интерфейс, база данных, веб-фреймворк и другие механизмы, обеспечивающие выполнение приложения, но не определяющие его предметную логику. Основным правилом данного подхода является правило зависимостей, согласно которому зависимости в исходном коде должны быть направлены к внутренним слоям, а внутренние слои не должны зависеть от внешних механизмов [20].

Использование данного подхода повышает сопровождаемость системы и упрощает изменение отдельных технических компонентов без существенного влияния на бизнес-логику. В контексте микросервисной архитектуры слои являются логическим способом организации кода и используются для управления сложностью программной реализации. В DDD-ориентированном микросервисе доменный слой отвечает за представление понятий предметной области и бизнес-правил, прикладной слой координирует выполнение сценариев

приложения, а инфраструктурный слой обеспечивает сохранение данных и взаимодействие с внешними механизмами [21].

На основе данных принципов для серверных микросервисов разрабатываемого веб-приложения была выбрана единая внутренняя структура, включающая четыре слоя: API, Application, Domain и Infrastructure. Слой API предназначен для обработки внешних HTTP-запросов и содержит контроллеры. Слой Application отвечает за выполнение прикладных сценариев и координацию бизнес-операций. Слой Domain содержит основные сущности предметной области и связанные с ними правила. Слой Infrastructure включает механизмы взаимодействия с базой данных, брокером сообщений RabbitMQ и другими внешними техническими средствами.

Обобщенная структура слоев микросервиса представлена на рисунке 20.

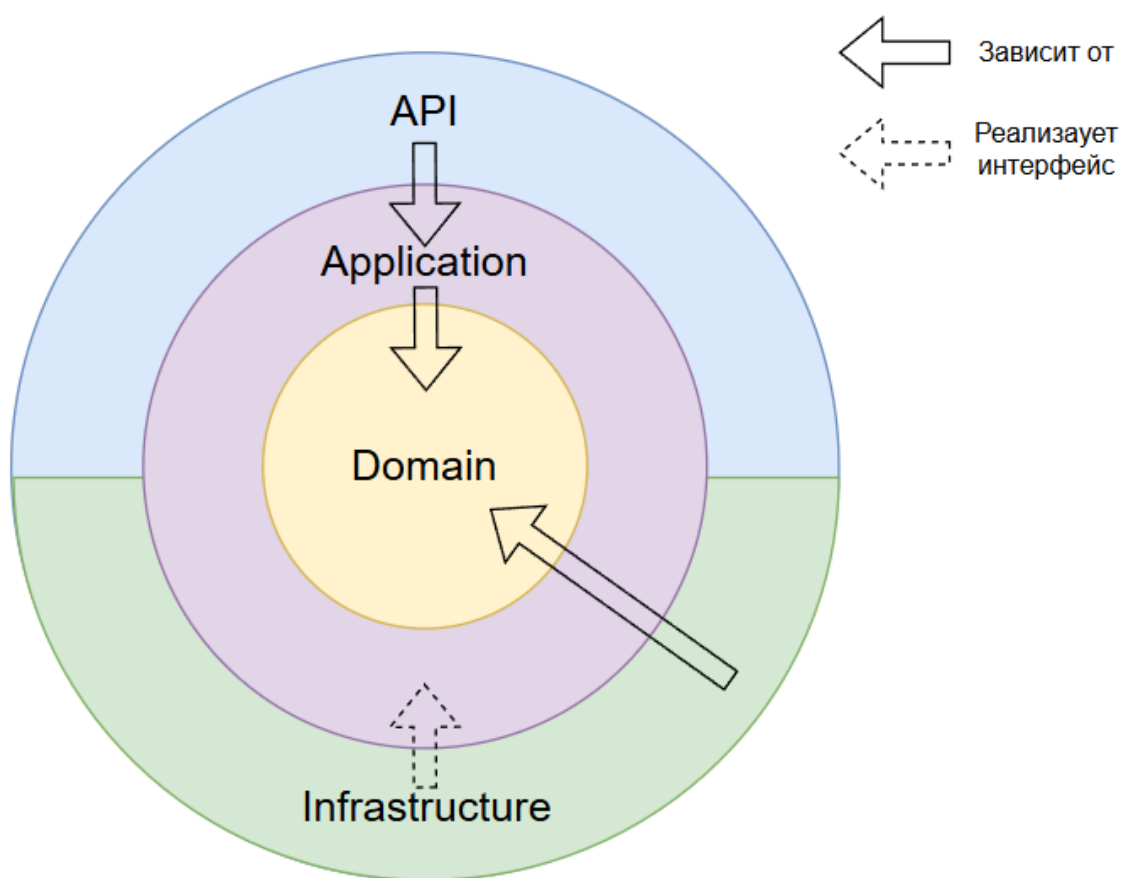


Рисунок 20 – Обобщенная структура слоев микросервиса

Представленная структура используется как общий принцип организации серверных микросервисов приложения. Конкретный состав классов внутри каждого слоя зависит от назначения микросервиса и реализуемых им функций. При этом общее разделение ответственности сохраняется для всех серверных сервисов: обработка запросов выполняется в слое API, прикладные сценарии размещаются в Application, предметные сущности в Domain, а работа с базой данных и внешними механизмами в Infrastructure.

2.2.1.2 Проектирование асинхронного взаимодействия между микросервисами

Поскольку каждый микросервис использует собственную базу данных, при проектировании необходимо было определить способ обмена данными между сервисами без прямого обращения одного сервиса к хранилищу другого. Для этого в архитектуре предусмотрено асинхронное взаимодействие на основе интеграционных событий.

Интеграционное событие представляет собой сообщение о значимом изменении состояния в одном микросервисе, которое может быть использовано другими микросервисами для обновления их локальных данных или выполнения прикладной логики. Например, создание пользователя, завершение матча или изменение рейтинга игрока являются событиями, которые должны быть учтены в разных частях системы.

В качестве брокера сообщений для передачи интеграционных событий был выбран RabbitMQ [16]. В проектируемой архитектуре сервис-источник публикует событие в брокер сообщений, а заинтересованные сервисы-получатели обрабатывают его независимо друг от друга. Такой подход позволяет снизить связанность между микросервисами и сохранить независимость их внутренних моделей данных.

Для повышения надёжности асинхронного обмена в архитектуре предусмотрено использование паттернов Outbox и Inbox. Паттерн Outbox применяется на стороне сервиса-источника события. Он предполагает

сохранение исходящего события в базе данных сервиса до публикации в брокер сообщений [22]. Паттерн Inbox применяется на стороне сервиса-получателя и используется для фиксации уже обработанных входящих событий. Это позволяет избежать повторной обработки одного и того же сообщения при повторной доставке [23].

Обобщённая схема публикации и обработки интеграционного события представлена на рисунке 21.

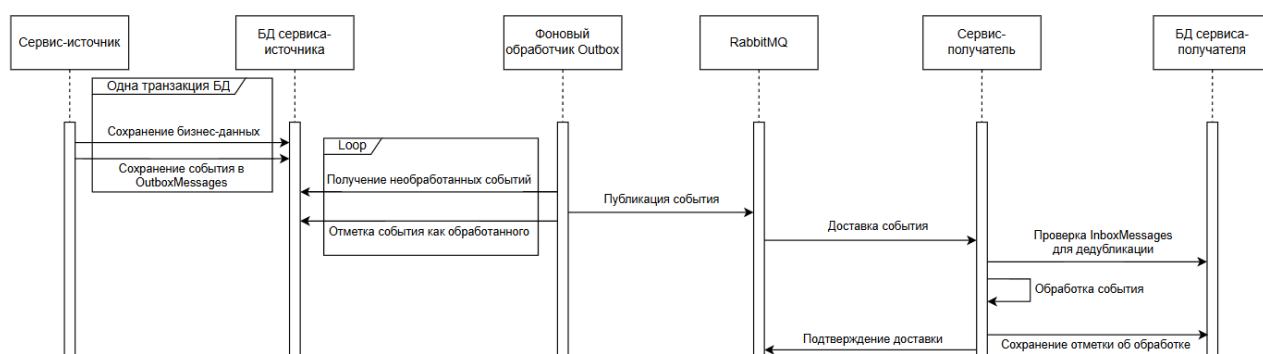


Рисунок 21 – Обобщённая схема публикации и обработки интеграционного события

Как показано на рисунке 21, сервис-источник сохраняет изменение бизнес-данных и исходящее событие в собственной базе данных. После этого фоновый обработчик публикует событие из таблицы Outbox в RabbitMQ. Сервис-получатель принимает событие из брокера сообщений, проверяет наличие отметки о его обработке в Inbox и, если событие ранее не обрабатывалось, выполняет соответствующую прикладную логику. После успешной обработки событие фиксируется как обработанное.

Таким образом, при проектировании серверной части был определён общий механизм межсервисного обмена: сервисы взаимодействуют через интеграционные события, передаваемые брокером RabbitMQ, а надёжность публикации и обработки сообщений обеспечивается использованием Outbox и Inbox.

В концептуальной модели отражены основные сущности предметной области, необходимые для функционирования системы. В таблице 6 приведено краткое описание сущностей, представленных на концептуальной схеме базы данных проекта.

Таблица 6 – Описание сущностей концептуальной схемы базы данных

Наименование	Описание
Пользователи	Хранит учётные записи пользователей (игроков, организаторов, администраторов) и их данные
Дисциплины	Справочник игровых дисциплин
Турниры	Содержит сведения о турнирах и их параметрах
Участники турнира	Регистрации пользователей на турниры
Команды	Команды, сформированные в рамках турнира
Состав команд	Принадлежность игроков к командам
Раунды	Раунды проведения турнира
Матчи	Матчи между командами и их результаты
Положение в швейцарской системе	Текущее положение команд в швейцарской системе
Положение в формате Double Elimination	Положение команд в формате Double Elimination
Рейтинги игроков	Текущий рейтинг игроков по дисциплинам
История рейтинга	История изменений рейтинга игроков
Уведомления	Уведомления, формируемые для пользователей

Ключевыми сущностями концептуальной модели являются «Турнир» и «Матч». Сущность «Турнир» объединяет основные данные соревнования и связывает его с дисциплиной, организатором, участниками, командами и раундами. Структура проведения турнира описывается сущностями «Раунд» и «Матч»: турнир состоит из раундов, а каждый раунд содержит матчи между командами. Сущность «Матч» фиксирует результат встречи двух команд и служит основанием для изменения рейтинга игроков, которое сохраняется в сущностях «Рейтинг игрока» и «История рейтинга». Команды формируются из участников турнира, а принадлежность игроков к командам отражается сущностью «Состав команды».

Выделенные сущности позволяют формализовать основные процессы системы, а связи между ними отражают логику предметной области.

Подробное описание связей между сущностями представлено в таблице 7.

Таблица 7 – Описание связей концептуальной схемы базы данных

Сущность 1	Сущность 2	Тип связи	Описание
Пользователи	Турниры	1 : N	Пользователь-организатор создаёт турниры
Дисциплины	Турниры	1 : N	По одной дисциплине проводится множество турниров
Пользователи	Участники турнира	1 : N	Пользователь участвует в турнирах
Турниры	Участники турнира	1 : N	Турнир включает участников
Турниры	Команды	1 : N	В турнире формируются команды
Пользователи	Команды	1 : N	Пользователь является капитаном команды
Команды	Состав команд	1 : N	Команда состоит из игроков
Пользователи	Состав команд	1 : N	Игрок входит в составы команд
Турниры	Раунды	1 : N	Турнир состоит из раундов
Раунды	Матчи	1 : N	Раунд содержит матчи
Команды	Матчи	1 : N	Команда участвует в матчах (сторона А или В)
Турниры	Положение в швейцарской системе	1 : N	Турнир имеет таблицу положения (швейцарская система)
Команды	Положение в швейцарской системе	1 : N	Команда имеет записи о своём положении в швейцарской системе
Турниры	Положение в формате Double Elimination	1 : N	Турнир имеет таблицу положения (Double Elimination)
Команды	Положение в формате Double Elimination	1 : N	Команда имеет записи о своём положении в формате Double Elimination
Пользователи	Рейтинги игроков	1 : N	Игрок имеет рейтинги по дисциплинам
Дисциплины	Рейтинги игроков	1 : N	В рамках дисциплины ведутся рейтинги игроков
Пользователи	История рейтинга	1 : N	История изменений рейтинга игрока
Дисциплины	История рейтинга	1 : N	В рамках дисциплины фиксируется история изменений рейтинга
Матчи	История рейтинга	1 : 0..N	По результату матча может формироваться запись истории рейтинга
Турниры	История рейтинга	1 : 0..N	В рамках турнира могут формироваться записи истории рейтинга
Пользователи	Уведомления	1 : N	Пользователь получает уведомления

Таким образом, разработанная концептуальная модель базы данных отражает состав основных сущностей предметной области и их взаимосвязи. Построенная модель служит основой для последующего перехода к логическому проектированию баз данных.

2.3.2 Проектирование логических моделей баз данных

После построения концептуальной модели был выполнен переход к логическому проектированию данных. Логическое проектирование базы данных является следующим этапом после построения концептуальной модели. На данном этапе сущности предметной области преобразуются в более детальное представление данных. В отличие от физического проектирования, логическая модель описывает структуру данных без привязки к особенностям конкретной СУБД и способам физического хранения [27]. Логическая диаграмма базы данных позволяет наглядно представить состав таблиц, их поля и связи между ними, что упрощает дальнейший переход к реализации базы данных [28].

Поскольку разрабатываемое веб-приложение построено на основе микросервисной архитектуры, единая централизованная база данных в системе не используется.

При логическом проектировании концептуальные сущности были распределены между базами данных микросервисов. Сущности, связанные с учётными записями и правами доступа, относятся к базе данных сервиса управления пользователями. Сущности, описывающие турниры, участников, команды, раунды и матчи, относятся к базе данных сервиса управления турнирами. Данные о рейтингах игроков выделены в отдельную базу данных сервиса рейтинга, а пользовательские уведомления хранятся в базе данных сервиса уведомлений.

Между таблицами, расположенными в разных базах данных, внешние ключи не используются, поскольку прямые связи между хранилищами разных микросервисов нарушали бы принцип автономности сервисов. Согласование данных между подсистемами выполняется с помощью интеграционных событий. Технические механизмы такого обмена (служебные таблицы исходящих и входящих сообщений) и локальные проекции данных других сервисов относятся к реализации и рассматриваются в разделе «Реализация баз данных». Подробная спецификация атрибутов таблиц приведена в приложении А.

2.3.2.1 Логическая модель базы данных сервиса управления пользователями

Логическая модель базы данных сервиса управления пользователями включает основную таблицу Users, в которой хранятся сведения об учётных записях пользователей: адрес электронной почты, хэш пароля, роль, статус учётной записи, никнейм игрока, название организатора, контактный идентификатор, идентификатор создавшего администратора, а также временные атрибуты создания, одобрения, отклонения и блокировки учётной записи.

При этом учётные записи игроков, организаторов и администраторов представлены в рамках единой сущности пользователя.

Тип пользователя определяется полем Role, а текущее состояние учётной записи – полем Status. Такой подход обусловлен тем, что все роли имеют общее ядро учётных данных.

Отличия между ролями выражаются отдельными необязательными полями, например никнеймом игрока или названием организатора. Поэтому выделение отдельных таблиц для игроков, организаторов и администраторов привело бы к дублированию общей части учётной записи и усложнению модели данных.

В рамках данного подхода заявка организатора также не выделяется в отдельную таблицу, а представляется записью пользователя с ролью организатора и статусом ожидания подтверждения или отклонения.

На основе этого была построена логическая модель базы данных сервиса управления пользователями, отображающая главную сущность этой предметной области и ее атрибуты.

Это позволит построить физическую базу данных пользователя на этапе реализации.

Логическая модель базы данных сервиса управления пользователями представлена на рисунке 23.

Пользователи		
Идентификатор пользователя 🔗	uuid	NN
Электронная почта 📧	varchar(320)	NN
Хэш пароля	varchar(1024)	NN
Роль	varchar(32)	NN
Статус	varchar(32)	NN
Никнейм 🏷️	varchar(64)	
Название организатора	varchar(128)	
Контакт для связи	varchar(64)	
Идентификатор создавшего администратора	uuid	
Дата создания	timestamp	NN
Дата одобрения	timestamp	
Дата отклонения	timestamp	
Дата блокировки	timestamp	

Рисунок 23 – Логическая модель базы данных сервиса управления пользователями

Таким образом, была построена логическая модель базы данных сервиса управления пользователями.

2.3.2.2 Логическая модель базы данных сервиса управления турнирами

Логическая модель базы данных сервиса управления турнирами предусматривает хранение данных о турнирах, участниках, командах, раундах и матчах. Основной таблицей данного сервиса является Tournaments, предназначенная для хранения сведений о турнирах: названия, описания, дисциплины, формата проведения, статуса, организатора, размера команды,

ограничений по количеству участников и временных атрибутов.

Для хранения участников турнира выделяется таблица TournamentParticipants. После формирования команд сведения о командах и их составе сохраняются в таблицах Teams и TeamMembers. Структура проведения турнира представляется таблицами Rounds и Matches, в которых хранятся данные о раундах, матчах, командах-участниках и результатах матчей. Для поддержки отдельных форматов проведения турниров предусматриваются таблицы SwissStandings и DoubleEliminationStandings, содержащие состояние команд в швейцарской системе и формате Double Elimination. Перечень поддерживаемых игровых дисциплин хранится в таблице Disciplines.

Логическая модель базы данных сервиса управления турнирами представлена на рисунке 24.



Рисунок 24 – Логическая модель базы данных сервиса управления турнирами

Таким образом, была построена логическая модель базы данных сервиса управления турнирами.

2.3.2.3 Логическая модель базы данных сервиса рейтинга и статистики

Логическая модель базы данных сервиса рейтинга и статистики

предусматривает хранение текущих рейтинговых показателей игроков и истории изменения рейтинга. Основной таблицей данного сервиса является PlayerRatings, предназначенная для хранения текущего рейтинга игрока в рамках конкретной игровой дисциплины.

В таблице PlayerRatings сохраняются идентификатор игрока, код дисциплины, текущее значение рейтинга Elo, количество сыгранных матчей, побед и поражений, а также временные атрибуты. Для пары PlayerId и DisciplineCode предусматривается ограничение уникальности, что позволяет хранить только одну актуальную запись рейтинга игрока в пределах одной дисциплины.

Для фиксации изменений рейтинга выделяется таблица RatingHistories. Она хранит сведения о предыдущем и новом значении рейтинга, величине изменения, причине изменения, а также идентификаторы матча и турнира, если изменение связано с результатом конкретного матча.

Логическая модель базы данных сервиса рейтинга и статистики представлена на рисунке 25.

Рейтинги игроков			История рейтинга		
Идентификатор записи	uuid	NN	Идентификатор записи	uuid	NN
Идентификатор игрока	uuid	NN	Идентификатор игрока	uuid	NN
Код дисциплины	varchar(64)	NN	Код дисциплины	varchar(64)	NN
Рейтинг	integer	NN	Рейтинг до изменения	integer	NN
Сыграно матчей	integer	NN	Рейтинг после изменения	integer	NN
Победы	integer	NN	Величина изменения	integer	NN
Поражения	integer	NN	Причина изменения	varchar(256)	NN
Дата создания	timestamp	NN	Идентификатор матча	uuid	
Дата обновления	timestamp	NN	Идентификатор турнира	uuid	
			Дата изменения	timestamp	NN

Рисунок 25 – Логическая модель базы данных сервиса рейтинга и статистики

Таким образом, была построена логическая модель базы данных сервиса рейтинга и статистики.

2.3.2.4 Логическая модель базы данных сервиса уведомлений

Логическая модель базы данных сервиса уведомлений предусматривает хранение пользовательских уведомлений. Основной таблицей данного сервиса является Notifications, предназначенная для хранения уведомлений, сформированных для пользователей системы.

В таблице Notifications сохраняются идентификатор получателя, тип уведомления, заголовок, текст уведомления, ссылка для перехода, дополнительные данные в формате JSON, идентификатор исходного события, а также временные атрибуты создания и прочтения уведомления. Для пары SourceEventId и RecipientUserId предусматривается ограничение уникальности, что позволяет исключить повторное создание одного и того же уведомления для одного пользователя при повторной обработке события.

Логическая модель базы данных сервиса уведомлений представлена на рисунке 26.

Уведомления	
Идентификатор уведомления 	uuid NN
Идентификатор получателя	uuid NN
Тип уведомления	varchar(32) NN
Заголовок	varchar(256) NN
Текст	varchar(2048) NN
Ссылка для перехода	varchar(512) NN
Дополнительные данные (JSON)	json NN
Идентификатор исходного события	uuid NN
Дата создания	timestamp NN
Дата прочтения	timestamp

Рисунок 26 – Логическая модель базы данных сервиса уведомлений

Таким образом, была построена логическая модель базы данных сервиса уведомлений.

2.4 Проектирование системы внутреннего рейтинга платформы

Одной из важных задач разрабатываемой платформы является поддержание конкурентной среды игрового процесса. Для обеспечения справедливых условий участия в турнирах необходимо, с одной стороны, формировать команды, максимально близкие по уровню подготовки игроков, а с другой стороны, обеспечивать корректное изменение рейтинга участников по результатам сыгранных матчей. Совместное использование данных механизмов позволяет как снизить начальный дисбаланс между соперниками, так и обеспечить последующую актуализацию уровня игроков на основе их фактических результатов.

Таким образом, проектирование системы внутреннего рейтинга платформы целесообразно разделить на два взаимосвязанных направления:

- проектирование процесса формирования команд;
- проектирование процесса изменения рейтинга игрока.

2.4.1 Проектирование процесса формирования команд

Одной из ключевых задач разрабатываемой системы является автоматическое формирование сбалансированных команд для проведения турниров. От качества распределения игроков напрямую зависит конкурентная среда игрового процесса: чем ближе команды по уровню подготовки, тем выше соревновательная ценность матчей и тем более равные условия создаются для всех участников турнира.

В качестве количественной характеристики уровня подготовки игрока используется рейтинг ELO [29]. Соответственно, задача формирования команд сводится к распределению игроков таким образом, чтобы различие между командами по суммарному уровню подготовки было минимальным.

Полный перебор вариантов разбиения экспоненциален по числу участников, а простые эвристики не гарантируют близости к оптимуму, поскольку не учитывают баланс всех команд одновременно [30]. Поэтому задача

формирования команд сведена к задаче целочисленного линейного программирования с целевой функцией, минимизирующей разрыв между сильнейшей и слабейшей командой [31].

Использование простых эвристических или статических алгоритмов распределения также не гарантирует получения близкого к оптимальному результата, поскольку такие методы, как правило, не учитывают глобальный баланс всех команд одновременно и формируют лишь допустимое, но не обязательно наилучшее или близкое к наилучшему распределение игроков [30].

В связи с этим требуется свести задачу формирования команд к задаче целочисленного линейного программирования. Такой подход позволяет использовать математическую модель, в которой выделяются ограничения и целевая функция для поиска наиболее сбалансированного варианта разбиения игроков на команды за приемлемое время [31]. Построим математическую модель для решения данной задачи.

Количество команд будем определять по формуле (1).

$$m = \frac{n}{q}, \quad (1)$$

где n – количество игроков;

q – количество игроков в одной команде;

m – количество команд.

Для описания принадлежности игрока к команде введем бинарные переменные:

$$x_{ij} = \begin{cases} 1, & \text{если игрок } i \text{ включен в команду } j; \\ 0, & \text{в противном случае.} \end{cases}$$

где x_{ij} – бинарная переменная распределения игрока i в команду j ;

i – номер игрока;

j – номер команды.

Дополнительно введем следующие обозначения:

r_i – рейтинг i -го игрока;

S_j – суммарный рейтинг j -й команды;

S_{max} – максимальный суммарный рейтинг среди всех команд;

S_{min} – минимальный суммарный рейтинг среди всех команд;

D – разность между максимальным и минимальным суммарными рейтингами команд.

Поскольку все команды имеют одинаковое количество игроков, минимизация различия между средними рейтингами команд эквивалентна минимизации различия между их суммарными рейтингами. Таким образом получаем:

$$D = S_{max} - S_{min}, \quad (2)$$

Целевая функция задачи в таком случае будет:

$$F = D \rightarrow \min, \quad (3)$$

Таким образом, требуется минимизировать разницу между наиболее сильной и наиболее слабой командой.

Введем ограничения для данной математической модели.

Каждый игрок должен входить ровно в одну команду:

$$\sum_{j=1}^m x_{ij} = 1, i = 1, 2, \dots, n.$$

Количество игроков в каждой команде должно быть одинаковым и равным q :

$$\sum_{i=1}^n x_{ij} = q, j = 1, 2, \dots, m.$$

Суммарный рейтинг каждой команды определяется как сумма рейтингов игроков, входящих в её состав:

$$S_j = \sum_{i=1}^n r_i x_{ij}, j = 1, 2, \dots, m.$$

Максимальный суммарный рейтинг должен быть не меньше рейтинга любой команды:

$$S_{max} \geq S_j, \quad j = 1, 2, \dots, m.$$

Минимальный суммарный рейтинг должен быть не больше рейтинга

любой команды:

$$S_{\min} \leq S_j, \quad j = 1, 2, \dots, m.$$

Разность между наиболее сильной и наиболее слабой командой определяется по формуле (2).

Переменные распределения являются бинарными:

$$x_{ij} \in \{0, 1\}, \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, m.$$

Дополнительные переменные являются целочисленными:

$$S_j \in Z, \quad j = 1, 2, \dots, m,$$

$$S_{\max}, S_{\min}, D \in Z.$$

Таким образом, заданные ограничения имеют следующий вид:

$$\left\{ \begin{array}{l} \sum_{j=1}^m x_{ij} = 1, \quad i = 1, 2, \dots, n, \\ \sum_{i=1}^n x_{ij} = q, \quad j = 1, 2, \dots, m, \\ S_j = \sum_{i=1}^n r_i x_{ij}, \quad j = 1, 2, \dots, m, \\ S_{\max} \geq S_j, \quad j = 1, 2, \dots, m, \\ S_{\min} \leq S_j, \quad j = 1, 2, \dots, m, \\ D = S_{\max} - S_{\min}, \\ x_{ij} \in \{0, 1\}, \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, m, \\ S_j \in Z, \quad j = 1, 2, \dots, m, \\ S_{\max}, S_{\min}, D \in Z \end{array} \right.$$

При целевой функции, определяемой формулой (3).

Таким образом, задача формирования команд сведена к задаче целочисленного линейного программирования. Построенная математическая модель формализует требования к распределению игроков и может быть использована на последующем этапе разработки алгоритма автоматического формирования сбалансированных команд.

Следует отметить, что в формате матча «один на один» ($q = 1$) каждая команда состоит ровно из одного игрока. В этом случае задача балансировки

сводится к попарному сопоставлению участников и не требует применения модели целочисленного программирования, поскольку суммарный рейтинг каждой «команды» равен рейтингу единственного игрока. Таким образом, построенная математическая модель применяется для форматов с $q \geq 2$.

2.4.2 Проектирование процесса изменения рейтинга игрока

Второй ключевой задачей при проектировании системы внутреннего рейтинга разрабатываемой платформы является поддержание актуального рейтинга игроков. Внутренний рейтинг используется как количественная характеристика уровня подготовки участника и применяется как при последующем формировании сбалансированных команд, так и при отражении текущих результатов игрока в рамках выбранной игровой дисциплины.

В разрабатываемой системе в качестве основы для расчёта внутреннего рейтинга используется система ELO, поскольку она позволяет оценивать ожидаемый результат матча на основании текущих рейтингов соперников и корректировать рейтинг игроков в зависимости от фактического исхода встречи [29].

В рамках текущей логики платформы матч всегда завершается победой одной команды и поражением другой. Ничейный результат не предусматривается. Следовательно, при проектировании алгоритма изменения внутреннего рейтинга рассматриваются только два возможных исхода матча: победа и поражение.

Поскольку в разрабатываемой системе матчи могут проводиться в командном формате, расчёт изменения рейтинга выполняется не для каждого игрока изолированно, а на основе средних рейтингов противоборствующих команд. Средний рейтинг команды определяется как среднее арифметическое рейтингов её игроков:

$$R_A = \frac{1}{q} \sum_{i=1}^q r_i,$$

где R_A – средний рейтинг команды А;
 r_i – рейтинг i -го игрока в команде;
 q – количество игроков в команде.

Аналогично определяется средний рейтинг команды В.

Ожидаемый результат команды А рассчитывается по следующей формуле:

$$E_A = \frac{1}{1 + 10^{\frac{R_B - R_A}{400}}}, [32]$$

где E_A – ожидаемый результат команды А;
 R_A – средний рейтинг команды А;
 R_B – средний рейтинг команды В.

Ожидаемый результат команды В определяется аналогично:

$$E_B = \frac{1}{1 + 10^{\frac{R_A - R_B}{400}}}.$$

При этом выполняется соотношение:

$$E_A + E_B = 1. \quad (4)$$

Фактический результат матча обозначим через S . В данной модели он принимает следующие значения:

$$S = \begin{cases} 1, & \text{если команда победила,} \\ 0, & \text{если команда проиграла.} \end{cases}$$

Изменение рейтинга команды определяется на основе формулы ELO:

$$\Delta R = K \times M \times (S - E),$$

где ΔR – изменение рейтинга команды;
 K – коэффициент чувствительности рейтинговой системы;
 M – коэффициент значимости результата матча;
 S – фактический результат матча;
 E – ожидаемый результат матча.

Поскольку выполняется соотношение (4), изменение рейтинга проигравшей команды В можно выразить через изменение рейтинга победителя:

$$\Delta R_B = K \times M \times (0 - E_B) = -K \times M \times E_B = -K \times M \times (1 - E_A) = -\Delta R_A,$$

Таким образом, изменение рейтинга проигравшей команды равно изменению рейтинга победившей команды с противоположным знаком:

$$\Delta R_B = -\Delta R_A.$$

Для предотвращения чрезмерного снижения рейтинга вводится нижняя граница его значения. Новый рейтинг игрока после матча определяется по формуле:

$$R_i' = \max(R_{\min}, R_i + \Delta R),$$

где R_i' – обновленный рейтинг i -го игрока;

$R_{\min}$ – минимально допустимое значение рейтинга, принятое равным 100;

R_i – текущий рейтинг i -го игрока.

В разрабатываемой системе коэффициент K зависит от размера команды. Для различных форматов матча принимаются следующие значения:

$$K = \begin{cases} 32, & \text{если } q=1; \\ 28, & \text{если } q=2; \\ 24, & \text{если } q=5; \\ 24 & \text{для остальных форматов.} \end{cases}$$

Такой выбор позволяет сделать изменение рейтинга более чувствительным в матчах с меньшим числом участников и менее резким в командных форматах.

Дополнительно в модели используется коэффициент значимости результата матча M , учитывающий разницу в счёте. Он определяется следующим образом:

$$M = 1 + \min(d, 10) \times 0,025,$$

где d – абсолютная разность между счётом победителя и счётом проигравшего.

Если матч завершён техническим поражением, принимается $d=10$, что соответствует максимально возможному значению коэффициента $M = 1,25$. Таким образом, более убедительная победа приводит к большему изменению рейтинга, а техническое поражение интерпретируется как максимальный по значимости проигрыш.

Таким образом, на данном этапе было осуществлено проектирование алгоритма изменения рейтинга игроков.

2.5 Проектирование клиентской части веб-приложения

Клиентская часть веб-приложения проектируется как отдельный компонент системы, обеспечивающий взаимодействие пользователя с серверными микросервисами. Основными задачами клиентской части являются отображение пользовательского интерфейса, организация навигации между экранами, отправка HTTP-запросов к серверной части и обработка состояния текущего пользователя.

При проектировании клиентской части учитывалась ролевая модель системы. В приложении предусмотрены четыре состояния пользователя: гость, игрок, организатор турниров и администратор. Для каждой роли определяется собственный набор доступных разделов и экранов. После успешной авторизации пользователь получает доступ только к тем функциям, которые соответствуют его роли и правам доступа.

Навигация в клиентской части строится по принципу разделения публичных и защищённых маршрутов. Публичные маршруты доступны гостю и включают экраны входа и регистрации. Защищённые маршруты требуют наличия авторизованного пользователя. Дополнительно для отдельных разделов проверяется роль пользователя: например, административные экраны доступны только администратору, а раздел управления турнирами доступен организатору турниров.

С учетом выделенной ролевой модели и набора пользовательских сценариев далее была спроектирована структура клиентской части приложения.

2.5.1 Проектирование структуры клиентской части

Взаимодействие клиентской части с серверной частью проектируется через HTTP-запросы к API соответствующих микросервисов. При выполнении защищённых запросов клиентская часть передаёт токен доступа, полученный после авторизации. Если пользователь заблокирован или его сессия недействительна, клиентская часть должна ограничить доступ к защищённым

экранам и отобразить соответствующее состояние интерфейса.

Наличие нескольких пользовательских ролей, защищённых маршрутов и различных сценариев работы требует выбора подхода к организации структуры клиентского кода. Для этого при проектировании клиентской части был выбран подход Feature-Sliced Design, предназначенный для структурирования frontend-приложений по уровням ответственности.

Feature-Sliced Design предполагает разделение кода на слои, слайсы и сегменты. Слои определяют уровень ответственности модуля в архитектуре приложения, слайсы группируют код по предметной области или пользовательскому сценарию, а сегменты разделяют код внутри слайса по техническому назначению. При этом подход не требует обязательного использования всех возможных слоев: состав слоев может адаптироваться под размер и особенности конкретного проекта.

В рамках Feature-Sliced Design зависимости между модулями направляются от верхних слоев к нижним. Благодаря этому страницы и пользовательские сценарии могут использовать более общие и переиспользуемые элементы, расположенные на нижних уровнях, но нижние уровни не зависят от конкретных страниц и сценариев. Такой принцип позволяет снизить связанность между частями клиентского приложения и упростить сопровождение кода [33]. Обобщенная схема данного подхода представлена на рисунке 27.

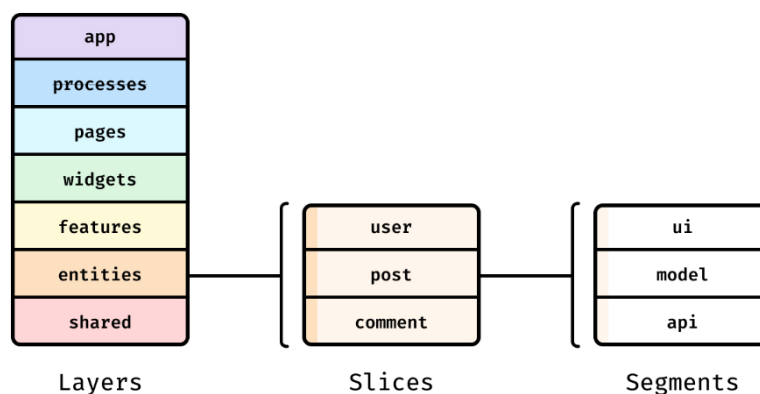


Рисунок 27 – Обобщенная схема подхода Feature-Sliced Design

Для разрабатываемого приложения данный подход является целесообразным, поскольку система включает несколько пользовательских ролей и большое число экранов, связанных с различными сценариями работы. Игрок, организатор турниров и администратор используют разные разделы приложения, однако часть интерфейсных элементов, механизмов авторизации, API-запросов и вспомогательной логики является общей. Feature-Sliced Design позволяет разделить код по сценариям и предметным областям, сохранив при этом общие переиспользуемые элементы на нижних уровнях. За счет этого изменения в одном пользовательском сценарии, например в управлении турнирами или административном разделе, в меньшей степени затрагивают остальные части клиентского приложения.

2.5.2 Проектирование пользовательского интерфейса

Проектирование пользовательского интерфейса разрабатываемого веб-приложения выполнялось на основе функциональных требований и ролевой модели системы. Для каждой роли был определён набор экранов, необходимых для выполнения соответствующих пользовательских сценариев.

На основе выделенных экранов были составлены карты экранов, отражающие возможные переходы между разделами приложения. Их использование позволяет заранее проверить логику переходов между разделами приложения, определить необходимые экраны для выполнения пользовательских сценариев и выявить возможные лишние или отсутствующие элементы интерфейса [34]. Макеты экранов были спроектированы в черно-белом формате, поскольку на данном этапе основной задачей являлось определение структуры интерфейса, состава элементов и логики переходов между экранами [34]. Рассмотрим списки экранов подробнее для каждой роли. Полный комплект макетов экранов представлен в преокте на GitHub [35].

2.5.2.1 Проектирование пользовательского интерфейса гостя

Для роли «Гость» были определены экраны, необходимые для входа в

систему и регистрации нового пользователя. Гость не имеет доступа к основным функциям приложения и может перейти только к сценариям авторизации или создания учетной записи.

На основе анализа функциональных требований гостя у данной роли был выделен следующий список экранов:

- страница входа;
- страница регистрации игрока;
- страница регистрации организатора;
- окно блокировки.

На основе выделенного списка экранов была составлена карта экранов для роли «Гость», представленная на рисунке 28.

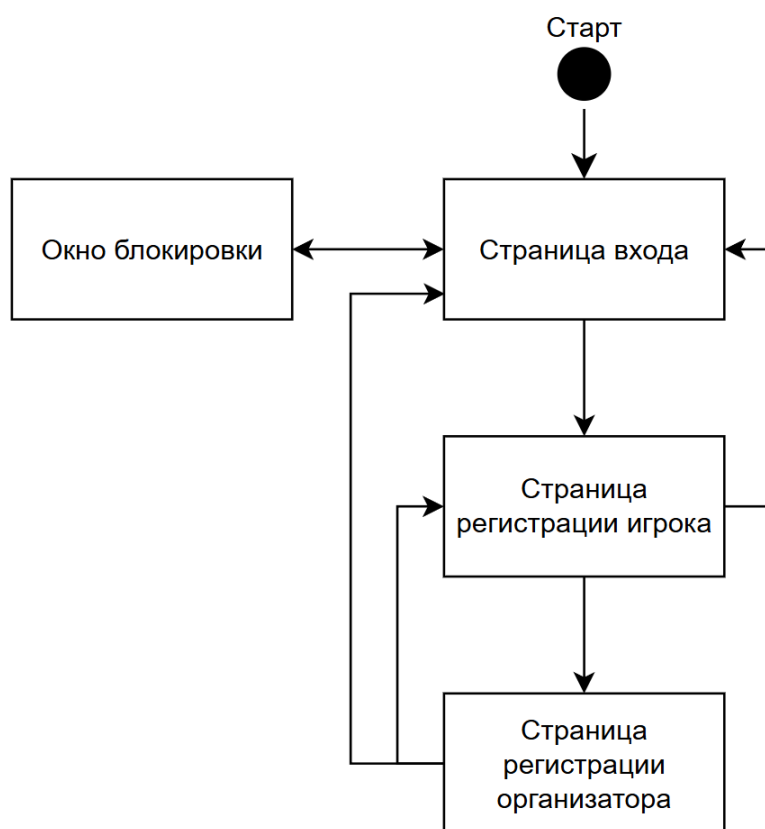


Рисунок 28 – Карта экранов для роли «Гость»

Данная карта отображает возможности пользователя с ролью «Гость» передвигаться между страницами приложения.

2.5.2.2 Проектирование пользовательского интерфейса игрока

На основе анализа функциональных требований игрока у данной роли был выделен следующий список экранов:

- главная страница;
- страница каталога турниров;
- страница турнира;
- страница турниров игрока;
- страница профиля игрока;
- страница уведомлений;
- страница матча.

На основе выделенного списка экранов была составлена карта экранов для роли «Игрок», представленная на рисунке 29.

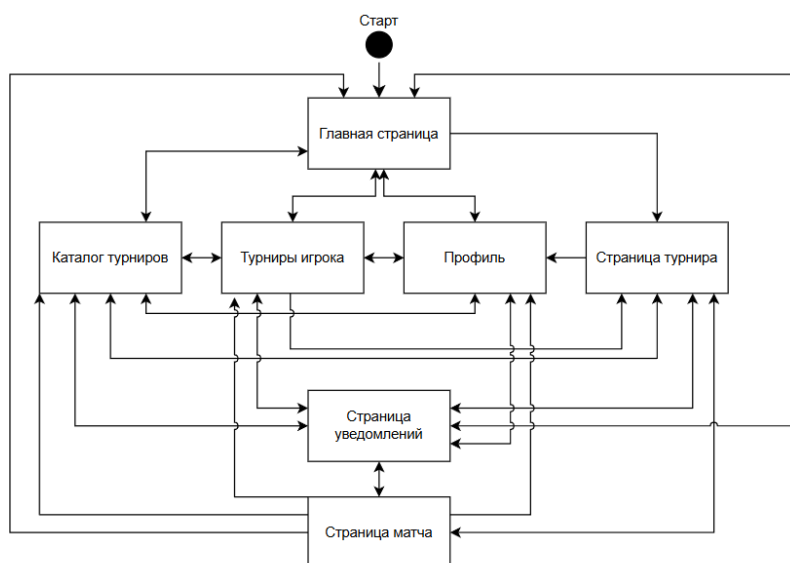


Рисунок 29 – Карта экранов для роли «Игрок»

Страница турнира проектировалась с учетом состояния турнира и формата его проведения. Для открытого турнира предусматривается отображение основной информации и возможности регистрации. Для запущенного турнира состав отображаемых данных зависит от формата проведения. Для швейцарской системы отображаются раунды и результаты, а для форматов с турнирной сеткой отображаются структура сетки и матчи. Также для игрока предусмотрена

отдельная страница просмотра информации о матче.

2.5.2.3 Проектирование пользовательского интерфейса организатора

На основе анализа функциональных требований организатора у данной роли был выделен следующий список экранов:

- страница турниров организатора;
- страница создания турнира;
- окно редактирования турнира;
- окно указания результатов матча;
- страница профиля организатора;
- страница каталога турниров;
- страница турнира;
- страница матча;
- страница ожидания одобрения заявки.

Так как организатор турниров может находиться в двух состояниях: когда его заявка одобрена и когда она ожидает рассмотрения, необходимо предусмотреть две вариации пользовательского интерфейса для данной роли: полную и ограниченную соответственно.

До одобрения заявки организатору доступен ограниченный набор экранов, связанных с просмотром статуса заявки и профиля. Учитывая это, была составлена карта экранов роли «Организатор» в случаях, когда его заявка еще не была одобрена. Данная карта экранов представлена на рисунке 30.

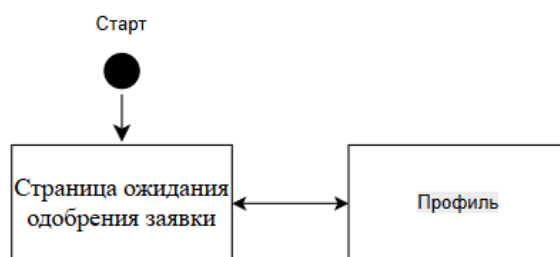


Рисунок 30 – Карта экранов для роли «Организатор» до одобрения заявки

После одобрения заявки администратором, у него появляются широкие функциональные возможности, выделенные на этапе анализа. Для данного случая также была спроектирована карта экранов, представленная на рисунке 31.

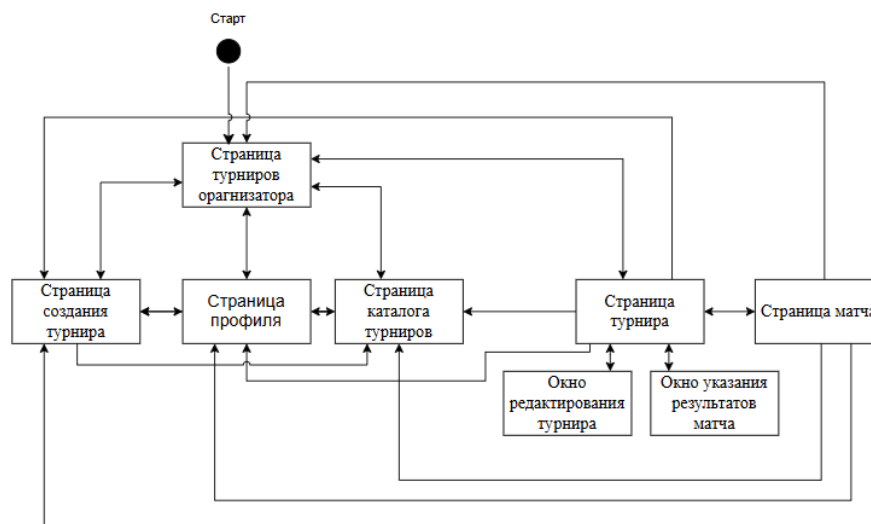


Рисунок 31 – Карта экранов для роли «Организатор» после одобрения заявки

Страница турнира для организатора проектировалась с учётом принадлежности турнира текущему пользователю. Если турнир создан другим организатором, пользователь может просматривать информацию о турнире, но не получает возможности управления им. Если турнир принадлежит текущему организатору, состав доступных действий зависит от состояния турнира: для открытого турнира предусматривается возможность редактирования данных, а для запущенного турнира возможность внесения результатов матчей.

2.5.2.1 Проектирование пользовательского интерфейса администратора

На основе анализа функциональных требований администратора у данной роли был выделен следующий список экранов:

- страница управления пользователями;
- окно создания пользователя;
- страница управления заявками организаторов;
- окно подтверждения одобрения заявки;

- окно подтверждения отклонения заявки;
- страница каталога турниров;
- окно создания турнира;
- страница турнира;
- страница профиля.

На основе выделенного списка экранов была составлена карта экранов для роли «Администратор», представленная на рисунке 32.



Рисунок 32 – Карта экранов для роли «Администратор»

Страницы турнира и матча для администратора проектировались по аналогии с соответствующими экранами организатора, но с учетом административных полномочий.

Таким образом, при проектировании клиентской части веб-приложения были определены принципы ролевой навигации, а также составлен состав экранов для каждой роли пользователя. На основе выделенных экранов были составлены карты экранов и подготовлены макеты пользовательского интерфейса, которые в дальнейшем использовались при реализации клиентской части приложения.

3 Реализация веб-приложения для организации и проведения турниров по компьютерным играм

В соответствии с архитектурными решениями, принятыми на этапе проектирования, веб-приложение было реализовано в виде набора взаимодействующих программных компонентов. Основу серверной части составляют микросервисы, каждый из которых отвечает за отдельную предметную область приложения и имеет собственное хранилище данных. Для наглядного представления состава реализованной системы была построена диаграмма компонентов. Она позволяет отразить основные части приложения, используемые хранилища данных и направления взаимодействия между компонентами [36]. Диаграмма компонентов веб-приложения представлена на рисунке 33.

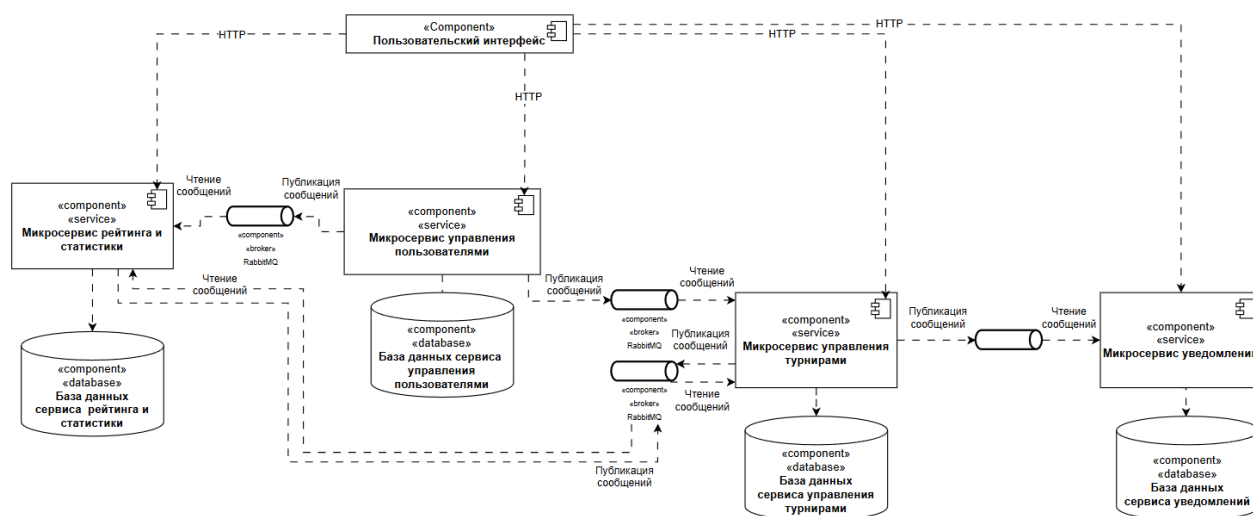


Рисунок 33 – Диаграмма компонентов веб-приложения

Как показано на диаграмме компонентов, серверная часть системы разделена на независимые сервисы, соответствующие различным предметным областям: сервис управления пользователями, сервис управления турнирами, сервис рейтинга и статистики игроков, а также сервис уведомлений. Каждый сервис реализует собственную бизнес-логику и использует отдельную базу данных. Взаимодействие клиентской части с серверными сервисами

осуществляется по протоколу HTTP, а межсервисный обмен данными реализуется асинхронно через брокера сообщений RabbitMQ. На диаграмме несколько элементов RabbitMQ отражают логические каналы обмена сообщениями, а не отдельные физически развернутые экземпляры брокера.

Для пояснения физической организации реализованного веб-приложения была построена диаграмма развертывания, отражающая размещение основных программных компонентов в среде выполнения. В отличие от диаграммы компонентов, показывающей логическую структуру системы, диаграмма развертывания демонстрирует, какие части приложения запускаются как самостоятельные контейнеры, каким образом клиентская часть направляет запросы к серверным микросервисам и через какие инфраструктурные компоненты выполняется межсервисное взаимодействие [37]. Диаграмма развертывания представлена на рисунке 34.

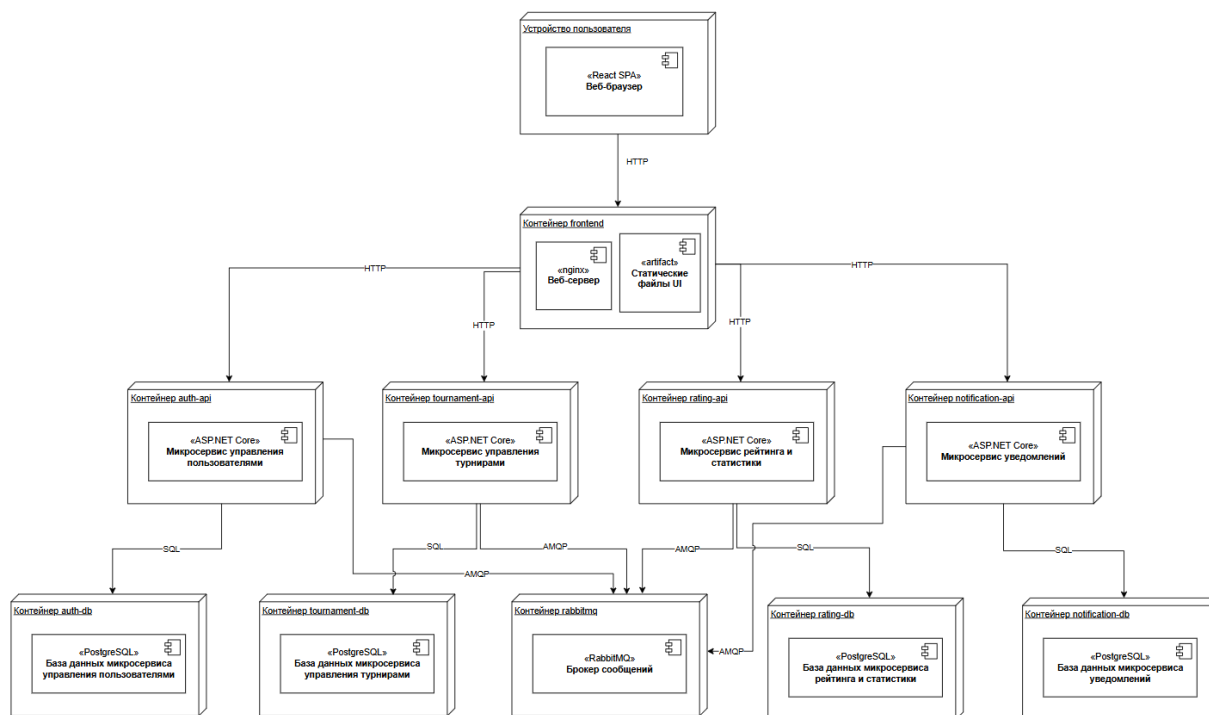


Рисунок 34 – Диаграмма развертывания веб-приложения

Как показано на рисунке 34, пользователь взаимодействует с веб-приложением через браузер. Клиентская часть размещается в отдельном

контейнере, внутри которого используется nginx. Он отдает статические файлы пользовательского интерфейса и выполняет маршрутизацию HTTP-запросов к соответствующим серверным микросервисам. Серверная часть представлена набором самостоятельных ASP.NET Core-сервисов, каждый из которых реализует отдельную предметную область и взаимодействует только с собственной базой данных PostgreSQL. Асинхронный обмен между сервисами выполняется через RabbitMQ по протоколу AMQP, что позволяет передавать интеграционные события без прямого обращения микросервисов к хранилищам друг друга.

Таким образом, реализованная архитектура соответствует архитектуре, спроектированной в разделе 2.2.1. Полный код реализованного приложения расположен в репозитории на GitHub [38].

3.1 Реализация микросервисов

Серверная часть веб-приложения реализована на платформе ASP.NET Core и состоит из набора микросервисов, каждый из которых выполняет отдельный набор функций. Каждый микросервис имеет внутреннее разделение на слои API, Application, Domain и Infrastructure, согласно архитектуре, выделенной на этапе проектирования. Слой API содержит контроллеры для обработки HTTP-запросов, слой Application реализует прикладные сценарии и бизнес-логику, слой Domain содержит сущности предметной области, а слой Infrastructure обеспечивает взаимодействие с базой данных и брокером сообщений, помимо этого он также реализует работу с механизмами аутентификации и безопасности, включая генерацию JWT-токенов и хеширование паролей, а также содержит инфраструктурные компоненты для надежной публикации и обработки интеграционных событий на основе паттернов Inbox и Outbox.

Для демонстрации реализованной архитектуры микросервисов рассмотрим подробно сервис управления пользователями.

Данный сервис предназначен для управления учетными записями пользователей и обеспечения доступа к функциям веб-приложения. На этапе

проектирования для данного микросервиса были определены следующие обязанности: регистрация новых пользователей, выполнение аутентификации, формирование токена доступа, хранение учетных данных и предоставление информации о текущем пользователе.

Для начала рассмотрим реализованные эндпоинты микросервиса, представленные в таблице 8. Эндпоинты остальных микросервисов приведены в приложении Б.

Таблица 8 – Реализованные эндпоинты сервиса управления пользователями

Метод	Эндпоинт	Назначение	Контроллер	Уровень доступа
POST	/api/auth/register/player	Регистрация нового игрока	AuthController	Без авторизации
POST	/api/auth/register/organizer	Регистрация нового организатора	AuthController	Без авторизации
POST	/api/auth/login	Аутентификация и получение токена доступа	AuthController	Без авторизации
GET	/api/auth/me	Получение данных текущего пользователя	AuthController	Авторизованный пользователь
POST	/api/auth/change-password	Изменение пароля	AuthController	Авторизованный пользователь
PUT	/api/auth/contact-handle	Обновление контактного хендла	AuthController	Авторизованный пользователь
GET	/api/admin/users	Получение списка пользователей	AdminUsersController	Администратор
POST	/api/admin/users	Создание пользователя	AdminUsersController	Администратор
POST	/api/admin/users/{id}/block	Блокировка пользователя	AdminUsersController	Администратор
POST	/api/admin/users/{id}/unblock	Разблокировка пользователя	AdminUsersController	Администратор
POST	/api/admin/users/{id}/reset-password	Сброс пароля пользователя	AdminUsersController	Администратор
GET	/api/admin/organizer-applications	Получение списка заявок организаторов	OrganizerApplicationsController	Администратор
GET	/api/admin/organizer-applications/history	История заявок организаторов	OrganizerApplicationsController	Администратор

Продолжение таблицы 8

Метод	Эндпоинт	Назначение	Контроллер	Уровень доступа
GET	/api/admin/organizer-applications/{id}	Получение заявки по ID	OrganizerApplicationsController	Администратор
POST	/api/admin/organizer-applications/{id}/approve	Одобрение заявки организатора	OrganizerApplicationsController	Администратор
POST	/api/admin/organizer-applications/{id}/reject	Отклонение заявки организатора	OrganizerApplicationsController	Администратор

Всего в микросервисе было реализовано 3 контроллера, находящихся в API слое микросервиса. Как видно из таблицы 6, API сервиса можно разделить на три группы. Первая группа эндпоинтов связана с регистрацией и аутентификацией пользователей.

Вторая группа обеспечивает выполнение операций авторизованного пользователя, таких как получение сведений о собственной учетной записи, изменение пароля и обновление контактного хендла. Третья группа предназначена для администратора и включает управление пользователями, а также обработку заявок на получение роли организатора.

Внутри микросервиса выделены проекты Auth.Api, Auth.Application, Auth.Domain и Auth.Infrastructure, каждый из которых отвечает за отдельную часть функциональности. Такое разделение позволяет отделить обработку HTTP-запросов от прикладных сценариев, бизнес-правил и технических механизмов хранения данных.

Разделение элементов микросервиса по слоям представлено в таблице 9.

Разделение классов по слоям остальных микросервисов приведено в приложении В.

Таблица 9 – Разделение по слоям в микросервисе управления пользователями

Слой	Назначение	Содержимое
Auth.Api	Обработка HTTP-запросов	AuthController, AdminUsersController, OrganizerApplicationsController

Продолжение таблицы 9

Слой	Назначение	Содержимое
Auth.Application	Выполнение сценариев регистрации, входа и управления пользователями	AuthService, AdminUsersService, IAuthService, IAdminUsersService, IAuthUserRepository, IJwtTokenGenerator, IPasswordHashingService, IOutboxWriter, LoginRequest, RegisterPlayerRequest, RegisterOrganizerRequest, UpdateContactHandleRequest, ChangePasswordRequest, AuthResponse, CurrentUserResponse, AdminUserResponse, AdminUsersQuery, CreateAdminUserRequest, OrganizerApplicationResponse, OrganizerApplicationsQuery, ResetPasswordRequest, ResetPasswordResponse, JwtToken
Auth.Domain	Описание сущности пользователя и правил изменения состояния	User
Auth.Infrastructure	Работа с БД, JWT, хешированием паролей и outbox	AuthUserRepository, AuthDbContext, JwtTokenGenerator, JwtOptions, PasswordHashingService, OutboxWriter

Таким образом данная таблица отражает распределение файлов микросервиса по его слоям, наглядно показывая придерживание принципу спроектированного подхода.

Для подтверждения правильности взаимодействия системы между слоями была построена диаграмма классов [39] с выделением отдельных слоев микросервиса на ней, представленная на рисунке 35.

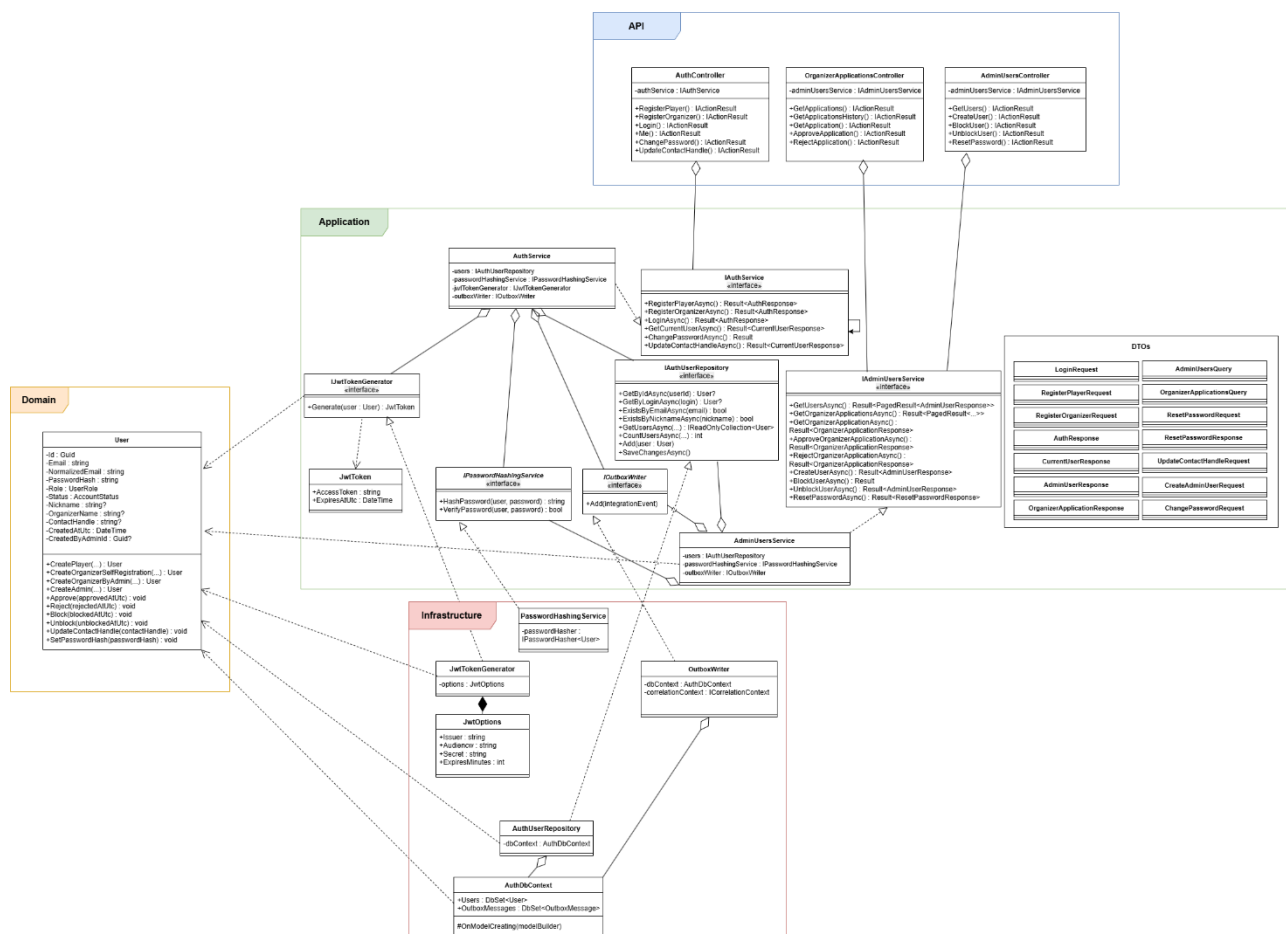


Рисунок 35 – Диаграмма классов микросервиса управления пользователями

На данной диаграмме видно, что зависимость слоев распространяется во внутрь: внешние слои зависят от внутренних, а внутренние от внешних не зависят. Рассмотрим каждый слой подробнее.

Слой API содержит контроллеры, обрабатывающие HTTP-запросы. Контроллеры не обращаются напрямую к базе данных и не реализуют бизнес-правила, а передают выполнение операций сервисам прикладного слоя. Данный слой зависит только от слоя Application.

Слой Application содержит прикладные сценарии работы сервиса и зависит только от слоя Domain. В этом слое отдельно на диаграмме выделены DTO-классы, используемые для передачи данных между клиентской частью и сервисом. Данные классы не содержат бизнес-логики и используются только для описания структуры входных и выходных данных API. По этой причине на диаграмме они представлены в укрупненном виде, без раскрытия полей, методов

и детальных связей с контроллерами и сервисами. Такое решение позволяет сохранить читаемость диаграммы и сосредоточить внимание на ключевых классах, определяющих архитектуру и реализацию сервиса.

Слой Domain представлен сущностью User, которая содержит основные данные пользователя и методы изменения его состояния. Данный слой не имеет зависимостей от других слоев микросервиса, так как является самым внутренним слоем, что соответствует принципам чистой архитектуры.

Слой Infrastructure содержит технические реализации интерфейсов, объявленных в прикладном слое. Таким образом, технические детали хранения данных и генерации токенов вынесены во внешний слой и не влияют на доменную модель, что обеспечивает принцип обратной зависимости.

Таким образом, диаграмма классов подтверждает соблюдение принципов Clean Architecture. Это упрощает сопровождение сервиса, повышает тестируемость прикладной логики и позволяет изменять инфраструктурные компоненты без переработки бизнес-правил.

Остальные серверные микросервисы реализованы по аналогичному принципу. Каждый из них имеет собственный набор контроллеров, прикладных сервисов, доменных сущностей и инфраструктурных компонентов, разделенных по слоям.

Сервис управления турнирами отвечает за создание и проведение турниров. В рамках данного микросервиса реализованы функции создания турнира, просмотра списков турниров по статусам, регистрации и отмены участия игроков, обновления параметров турнира, отмены турнира, формирования команд, генерации турнирной сетки и внесения результатов матчей.

Внутренняя структура сервиса также соответствует Clean Architecture. В слое API размещены контроллеры для обработки запросов игроков, организаторов и администратора. Слой Application содержит прикладные сервисы управления турнирами, жизненным циклом соревнования, результатами матчей, генерацией сетки и балансировкой команд. В слое Domain размещены

основные сущности предметной области: турнир, участник, команда, матч, раунд и связанные с ними состояния. Слой Infrastructure реализует работу с базой данных, репозиториями, транзакциями и обработкой интеграционных событий, поступающих от других микросервисов.

Сервис рейтинга предназначен для хранения и обновления рейтинговых показателей игроков по игровым дисциплинам. Он получает события о завершённых матчах и на их основе выполняет перерасчёт рейтинга игроков. Благодаря выделению рейтинговой логики в отдельный микросервис изменение правил расчёта рейтинга не требует переработки сервиса турниров.

В слое API размещены эндпоинты для получения рейтинговых данных. Слой Application содержит сценарии получения рейтинга игрока, просмотра рейтинговых таблиц и обработки результатов матчей. Слой Domain описывает рейтинговые сущности и правила изменения рейтинга. Слой Infrastructure отвечает за сохранение рейтинговых данных, обработку входящих событий и публикацию изменений рейтинга для других частей системы.

Сервис уведомлений отвечает за формирование и хранение уведомлений пользователей. Он получает интеграционные события от других микросервисов и на их основе формирует уведомления для участников турнира. В текущей реализации обрабатывается событие создания нового раунда (RoundCreatedEvent): сервис формирует для участников уведомления о матчах нового раунда. Обработка других типов уведомлений может быть добавлена по мере расширения системы.

В слое API размещены эндпоинты для получения уведомлений пользователя и изменения их статуса прочтения. Слой Application содержит сценарии работы с уведомлениями, слой Domain описывает сущность уведомления и её состояние, а слой Infrastructure обеспечивает сохранение уведомлений и обработку входящих событий через брокер сообщений.

Подробные перечни эндпоинтов и распределение классов оставшихся микросервисов отображены в приложениях Б и В соответственно. Код основных реализованных контроллеров микросервисов представлен в приложении

Г (листинг Г.3 – Г.6)

Таким образом, все серверные микросервисы реализованы по единому архитектурному принципу. Каждый сервис имеет собственную предметную область, отдельное хранилище данных и внутреннее разделение на слои. Подробное рассмотрение сервиса аутентификации и регистрации пользователей показывает общий подход к организации кода. Остальные микросервисы используют тот же принцип построения, что обеспечивает единообразие реализации, снижает связанность компонентов и упрощает сопровождение серверной части приложения.

3.1.1 Реализация асинхронного взаимодействия между сервисами

В соответствии с архитектурой, определённой на этапе проектирования, межсервисное взаимодействие в серверной части приложения реализовано асинхронно с использованием брокера сообщений RabbitMQ. Обмен данными между микросервисами выполняется посредством интеграционных событий, которые публикуются сервисом-источником и обрабатываются сервисами-получателями [40].

В рамках реализации каждый микросервис сохраняет собственную локальную модель данных и не обращается напрямую к базам данных других сервисов. Такой подход соответствует принципу *database per service*, согласно которому данные сервиса должны быть доступны другим сервисам только через API или события, а не через прямое обращение к его хранилищу [41]. Если изменение состояния одного сервиса должно быть учтено в другом сервисе, оно передаётся в виде интеграционного события.

Рассмотрим подробнее список реализованных событий и их взаимодействие между разработанными микросервисами. Для этого составим таблицу с основными интеграционными событиями приложения, в которой будет наглядно представлено как события распределены по микросервисам и их назначения.

Интеграционные события представлены в таблице 10.

Таблица 10 – Интеграционные события микросервисов

Сервис	Публикуемые события	Обрабатываемые события	Назначение обработки
Сервис аутентификации	UserCreatedEvent, UserBlockedEvent, UserRoleChangedEvent, UserContactHandleChangedEvent	-	Уведомление других сервисов об изменениях учетных записей пользователей
Сервис турниров	MatchCompletedEvent, RoundCreatedEvent	UserCreatedEvent, UserBlockedEvent, UserRoleChangedEvent, UserContactHandleChangedEvent, RatingUpdatedEvent	Актуализация локальных проекций пользователей и рейтингов; публикация событий о ходе турнира
Сервис рейтинга	RatingUpdatedEvent	UserCreatedEvent, UserBlockedEvent, UserRoleChangedEvent, MatchCompletedEvent	Создание и обновление рейтингов игроков, перерасчёт ELO после завершения матча
Сервис уведомлений	-	RoundCreatedEvent	Формирование уведомлений участникам о созданных матчах нового раунда

Общая схема реализованного асинхронного взаимодействия между микросервисами представлена на рисунке 36.

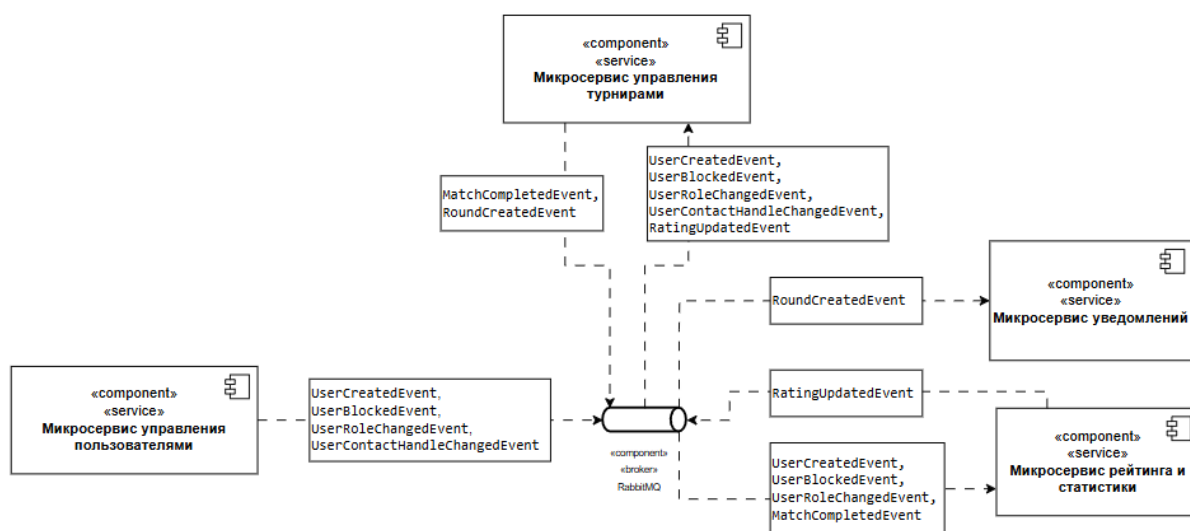


Рисунок 36 – Схема асинхронного взаимодействия между микросервисами

Как показано на рисунке 36, сервис аутентификации публикует события, связанные с созданием пользователей, блокировкой учетных записей, изменением роли пользователя и изменением контактного хендла. Эти события обрабатываются сервисом турниров для актуализации локальных пользовательских проекций. Часть событий также используется сервисом рейтинга для создания, восстановления или изменения рейтинговых записей игроков.

Сервис турниров публикует события, связанные с ходом проведения соревнований. После изменения рейтинга сервис рейтинга публикует событие `RatingUpdatedEvent`, которое обрабатывается сервисом турниров для обновления локальных рейтинговых проекций игроков.

Помимо сервиса турниров и сервиса рейтинга, в обмене событиями участвует сервис уведомлений. Он подписан на событие `RoundCreatedEvent`, публикуемое сервисом турниров при формировании нового тура, и на его основе оповещает участников о ходе соревнований. Событие `MatchCompletedEvent`, также публикуемое сервисом турниров по итогам матча, обрабатывается сервисом рейтинга и инициирует перерасчёт рейтинговых записей игроков; результат этого перерасчёта возвращается в сервис турниров через событие `RatingUpdatedEvent`, замыкая цикл обмена данными между сервисами.

Для повышения надёжности асинхронного обмена, в соответствии с этапом проектирования, в микросервисах реализованы паттерны `Outbox` и `Inbox`. Паттерн `Outbox` используется на стороне сервиса-отправителя.

Для демонстрации реализованного механизма рассмотрим сценарий завершения матча и последующего перерасчёта рейтинга игрока. Данный сценарий затрагивает сервис турниров и сервис рейтинга и показывает работу паттернов `Outbox` и `Inbox` в связке с `RabbitMQ`. Для этого построим диаграмму последовательности публикации и обработки события `MatchCompletedEvent`, отражающую реализацию принципов `Outbox` и `Inbox` паттернов в разработанном веб-приложении. Для этого отобразим на диаграмме узлы, через которые проходит событие и способы взаимодействия между ними.

Последовательность публикации и обработки интеграционного события в реализованной системе на примере завершения матча представлена на рисунке 37.

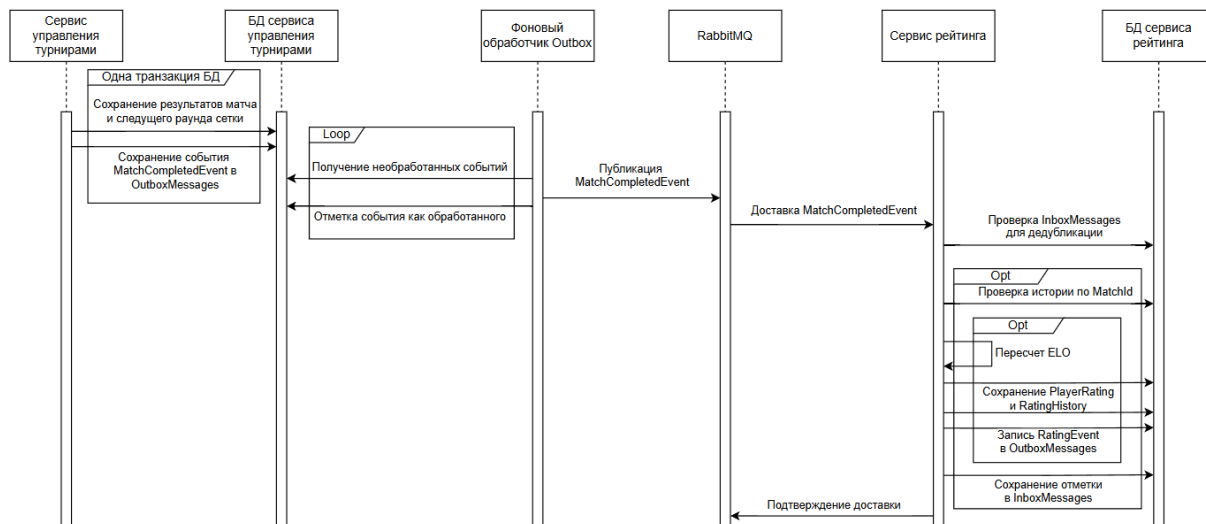


Рисунок 37 – Диаграмма последовательности публикации и обработки события MatchCompletedEvent

Как показано на рисунке 37, после внесения результата матча сервис турниров в рамках одной транзакции, чтобы согласовать изменения в бизнес-логике и запись в таблице OutboxMessages, сохраняет изменения в собственной базе данных и формирует интеграционное событие MatchCompletedEvent. Данное событие записывается в таблицу OutboxMessages той же базы данных. После этого фоновый обработчик Outbox периодически опрашивает эту таблицу и получает необработанные события из нее и публикует их в RabbitMQ.

Со стороны сервиса рейтинга сообщение принимает инфраструктурный обработчик RabbitMQ. Перед передачей события в прикладную логику выполняется проверка таблицы InboxMessages. Если событие с таким идентификатором уже было обработано данным потребителем, повторная обработка не выполняется. Если запись отсутствует, событие передаётся в consumer-класс RatingMatchCompletedConsumer, который вызывает метод HandleMatchCompletedAsync прикладного сервиса RatingService.

Внутри RatingService выполняется дополнительная проверка наличия

истории по идентификатору матча. Она необходима для защиты от повторного применения результата матча в ситуации, когда событие уже привело к изменению рейтинга, но отметка о его обработке в `InboxMessages` ещё не была сохранена. В системах с асинхронной доставкой сообщений обработчик должен учитывать возможность повторной доставки и выполнять обработку идемпотентно, то есть так, чтобы повторное получение того же сообщения не приводило к повторному изменению состояния системы [42]. Такая ситуация возможна, поскольку изменение рейтинговых данных и фиксация факта обработки входящего события выполняются разными инфраструктурными механизмами. Поэтому проверка `HasMatchHistoryAsync` по `MatchId` позволяет сделать обработку события идемпотентной на уровне бизнес-логики.

Если результат матча ранее не применялся, сервис рейтинга загружает рейтинги игроков победившей и проигравшей сторон, рассчитывает изменение ELO, обновляет записи `PlayerRatings` и добавляет записи в `RatingHistories`. После изменения рейтинга формируется исходящее событие `RatingUpdatedEvent`, которое сохраняется в `OutboxMessages` сервиса рейтинга. В дальнейшем это событие публикуется в `RabbitMQ` и используется сервисом турниров для обновления локальных рейтинговых проекций игроков.

После успешного выполнения обработки инфраструктурный механизм фиксирует факт обработки входящего события в таблице `InboxMessages`. Таким образом, при повторной доставке того же `MatchCompletedEvent` сервис рейтинга сможет определить, что событие уже обрабатывалось, и не выполнит повторный перерасчёт рейтинга.

Таким образом, реализованное асинхронное взаимодействие обеспечивает передачу изменений между микросервисами без прямой зависимости от внутренних хранилищ друг друга. Паттерн `Outbox` используется для надёжной публикации исходящих событий, паттерн `Inbox` для защиты от повторной обработки входящих сообщений, а дополнительные проверки в прикладной логике защищают критичные операции от повторного применения эффекта события. На примере обработки `MatchCompletedEvent` видно, что изменение

состояния турнира, перерасчёт рейтинга и последующее обновление рейтинговых проекций выполняются как связанная цепочка асинхронных событий.

3.2 Реализация алгоритмов системы внутреннего рейтинга платформы

На этапе проектирования задача формирования команд была представлена как задача целочисленного программирования. В реализации данная модель была перенесена в программный компонент `CpSatAverageEloTeamBalancer`, который отвечает за автоматическое распределение игроков по командам с учетом их рейтинга ELO. Алгоритм используется в момент запуска турнира: когда количество активных участников становится равным максимальному числу игроков, система закрывает набор, получает рейтинги участников, формирует команды и после этого строит начальную турнирную структуру.

Программная реализация напрямую соответствует переменным x_{ij} , введённым на этапе проектирования (раздел 2.3.1), булевы переменные задаются для каждого возможного включения игрока в команду.

В качестве инструмента для решения оптимизационной задачи была использована библиотека Google OR-Tools. Согласно официальной документации Google, OR-Tools представляет собой открытый программный инструментарий для решения задач комбинаторной оптимизации, включая задачи целочисленного и линейного программирования, программирования в ограничениях, маршрутизации и планирования [43]. Для задач, в которых используются только целочисленные переменные и ограничения, в OR-Tools может применяться решатель CP-SAT [44].

В OR-Tools математическая модель описывается через объект `CpModel`. Булевы переменные создаются методом `NewBoolVar`, а целочисленные переменные методом `NewIntVar`. Ограничения добавляются в модель с помощью методов `Add`, `AddExactlyOne`, `AddMaxEquality` и `AddMinEquality`. Метод `AddExactlyOne` используется для задания условия, согласно которому каждый

игрок должен входить ровно в одну команду. Метод Add применяется для задания линейных равенств и ограничений, включая фиксированный размер команды, расчет суммарного рейтинга команды и определение разности между максимальным и минимальным суммарным рейтингом. Методы AddMaxEquality и AddMinEquality позволяют связать переменные максимального и минимального рейтинга с массивом суммарных рейтингов команд. После задания переменных и ограничений определяется целевая функция с помощью метода Minimize, минимизирующая разницу между наиболее сильной и наиболее слабой командой. Метод формирования команд представлен в приложении Г (листинг Г.1).

В указанном фрагменте сначала выполняется проверка входных данных и формируется резервный вариант распределения команд с помощью вспомогательного жадного алгоритма. Такой подход используется для повышения надежности работы системы: если решатель не сможет найти допустимое решение или найденное решение окажется хуже резервного варианта, система использует заранее построенное распределение.

После подготовки данных создается объект CpModel, в котором последовательно задаются переменные и ограничения. Метод CreateAssignmentVariables создает матрицу булевых переменных, соответствующую переменным x_{ij} из математической модели. Методы AddPlayerConstraints и AddTeamSizeConstraints задают ограничения на включение каждого игрока ровно в одну команду и на фиксированный размер каждой команды. Метод CreateTeamTotalVariables формирует переменные суммарного рейтинга команд S_j и связывает их со значениями ELO игроков через взвешенную сумму.

Затем в модели создаются переменные maxTeamTotal, minTeamTotal и teamTotalGap, соответствующие $S_{\max}$, $S_{\min}$ и D . С помощью методов AddMaxEquality и AddMinEquality переменные максимального и минимального рейтинга связываются с массивом суммарных рейтингов команд. После этого

задается равенство, определяющее разность между ними, и целевая функция минимизации этой разности. Тем самым программная реализация полностью соответствует математической модели, разработанной на этапе проектирования.

После построения модели она передается в решатель CpSolver. Для управления временем работы решателя используются параметры, формируемые методом `BuildSolverParameters`. В частности, задается ограничение максимального времени поиска решения и количество рабочих потоков. Это необходимо, поскольку задача распределения игроков является комбинаторной, и при увеличении числа участников время поиска оптимального решения может возрастать. Ограничение времени позволяет сохранить приемлемую скорость запуска турнира.

Если решатель возвращает статус `Optimal` или `Feasible`, найденное решение считывается из значений булевых переменных. Если переменная принадлежности игрока к команде принимает значение `true`, игрок добавляется в соответствующую команду. После этого выполняется проверка корректности сформированных команд и сравнение качества решения с резервным вариантом. В итоговый результат передаются команды, которые затем сохраняются в доменной модели турнира и используются при генерации турнирной сетки или структуры раундов.

Таким образом, реализация алгоритма формирования турнирной структуры связана с проектной моделью напрямую. На этапе проектирования были определены переменные, ограничения и целевая функция, а в программной реализации эти элементы были выражены средствами OR-Tools. После решения оптимизационной задачи результат преобразуется в прикладные сущности системы: команды, участников команд, капитанов и посев команд в турнире. Это позволяет автоматически формировать сбалансированную турнирную структуру на основе текущих рейтингов игроков.

После проведения матча сформированная турнирная структура используется в алгоритме изменения внутреннего рейтинга игроков. Данный алгоритм также реализован в соответствии с моделью, описанной на этапе

проектирования. Расчёт состоит из трёх шагов: определяется множитель, учитывающий разницу в счёте (метод `CalculateScoreCoefficient`); по средним рейтингам команд вычисляется ожидаемый результат и итоговое изменение рейтинга (метод `CalculateTeamDelta`); полученное изменение применяется к рейтингам игроков. Реализация ключевых методов приведена в листингах 1 и 2.

Листинг 1 – Вычисление множителя за разницу в счёте

```
private static double
CalculateScoreCoefficient(MatchCompletedEvent integrationEvent)
{
    int scoreDifference;
    if (integrationEvent.IsTechnicalDefeat)
    {
        scoreDifference = MaxScoreDifference;
    }
    else if (integrationEvent.WinnerScore is not null &&
integrationEvent.LoserScore is not null)
    {
        scoreDifference =
Math.Abs(integrationEvent.WinnerScore.Value -
integrationEvent.LoserScore.Value);
    }
    else
    {
        scoreDifference = 1;
    }
    return 1 + Math.Min(scoreDifference, MaxScoreDifference) *
0.025;
}
```

Основной метод вычисления изменения рейтинга приведен в листинге 2.

Листинг 2 – Вычисление изменения рейтинга

```
public double CalculateExpectedScore(double rating, double
opponentRating)
{
    return 1 / (1 + Math.Pow(10, (opponentRating - rating) /
400.0));
}
public int CalculateTeamDelta(
    double teamAverageElo,
    double opponentAverageElo,
    int teamSize,
    double actualScore,
    double scoreMultiplier)
{
    var expectedScore = CalculateExpectedScore(teamAverageElo,
opponentAverageElo);
```

```

        var kFactor = GetKFactor(teamSize);
        return (int)Math.Round(kFactor * scoreMultiplier *
(actualScore - expectedScore));
    }
    private static int GetKFactor(int teamSize)
    {
        return teamSize switch
        {
            1 => 32,
            2 => 28,
            5 => 24,
            _ => 24
        };
    }
}

```

Представленные методы напрямую реализуют модель раздела 2.3.2: CalculateExpectedScore – формулу ожидаемого результата, GetKFactor – значения коэффициента K, CalculateScoreCoefficient – коэффициент значимости M; нижняя граница рейтинга обеспечивается константой MinimumElo = 100. Отличие реализации от модели состоит в округлении изменения рейтинга до целого и в значении разницы счёта по умолчанию ($d = 1$), если счёт матча не передан.

Затем по средним рейтингам победившей и проигравшей команд рассчитывается ожидаемый результат expectedScore по классической формуле ELO. Коэффициент kFactor зависит от размера команды и определяется методом GetKFactor: для индивидуального формата он равен 32, для команд из двух игроков 28, для команд из пяти и более 24. Итоговое изменение рейтинга вычисляется как произведение коэффициента kFactor, множителя за разницу в счёте и разности между фактическим и ожидаемым результатом, после чего округляется до целого значения.

Полученное значение изменения рейтинга применяется к игрокам обеих команд: для победившей команды фактический результат равен 1, поэтому изменение прибавляется к текущему рейтингу, а для проигравшей команды используется противоположное по знаку значение и вычитается. При применении изменения используется нижняя граница рейтинга MinimumElo, равная 100, что предотвращает снижение значения ниже минимально допустимого уровня ($\text{newElo} = \text{Math.Max}(\text{MinimumElo}, \text{oldElo} + \text{delta})$).

3.3 Реализация баз данных

Реализация баз данных микросервисов выполнена с использованием PostgreSQL и Entity Framework Core. В проекте применяется подход Code First, при котором структура базы данных формируется на основе доменных классов и конфигураций отображения. Данный подход был выбран потому, что он позволяет поддерживать связь между доменной моделью приложения и структурой базы данных, а также упрощает развитие схемы данных при изменении программной модели [45].

Для каждого микросервиса определён собственный контекст базы данных, наследующийся от DbContext. В Entity Framework Core экземпляр DbContext представляет сеанс взаимодействия с базой данных и используется для выполнения запросов и сохранения экземпляров сущностей [46]. Контекст содержит наборы DbSet, соответствующие таблицам базы данных конкретного сервиса.

Настройка таблиц, типов данных, первичных ключей, индексов и ограничений выполняется в конфигурационных классах Entity Framework Core. На основе описанной модели формируются миграции. Миграции EF Core позволяют постепенно обновлять схему базы данных и поддерживать её в соответствии с моделью данных приложения [47]. В рамках проекта миграции применяются при запуске сервисов и создают физическую структуру баз данных в PostgreSQL.

Физические модели баз данных построены на основе логических моделей, разработанных на этапе проектирования (рисунки 23–26). При переходе к физической модели в структуру баз данных дополнительно включены служебные таблицы, обеспечивающие реализацию микросервисного взаимодействия: таблицы исходящих и входящих интеграционных событий (OutboxMessages и InboxMessages), а в сервисе управления турнирами – также таблицы локальных проекций данных других сервисов (PlayerRatingProjections, user_projections и BlockedUserProjections). Кроме того, в каждой базе данных

3.3.2 Реализация базы данных сервиса управления турнирами

В сервисе управления турнирами работа с базой данных реализована через контекст TournamentDbContext. В данном контексте определены наборы DbSet для таблиц, связанных с созданием и проведением турниров: Tournaments, TournamentParticipants, Teams, TeamMembers, Rounds, Matches, SwissStandings, DoubleEliminationStandings и Disciplines.

Кроме основных таблиц предметной области, в контексте TournamentDbContext определены таблицы локальных проекций PlayerRatingProjections, user_projections и BlockedUserProjections. Они используются для хранения данных, полученных из других микросервисов через интеграционные события, что позволяет сервису работать без прямого обращения к их базам данных. Также в базе данных сервиса предусмотрены таблицы InboxMessages и OutboxMessages для обработки входящих и публикации исходящих интеграционных событий.

Физическая модель базы данных сервиса управления турнирами представлена на рисунке 39.

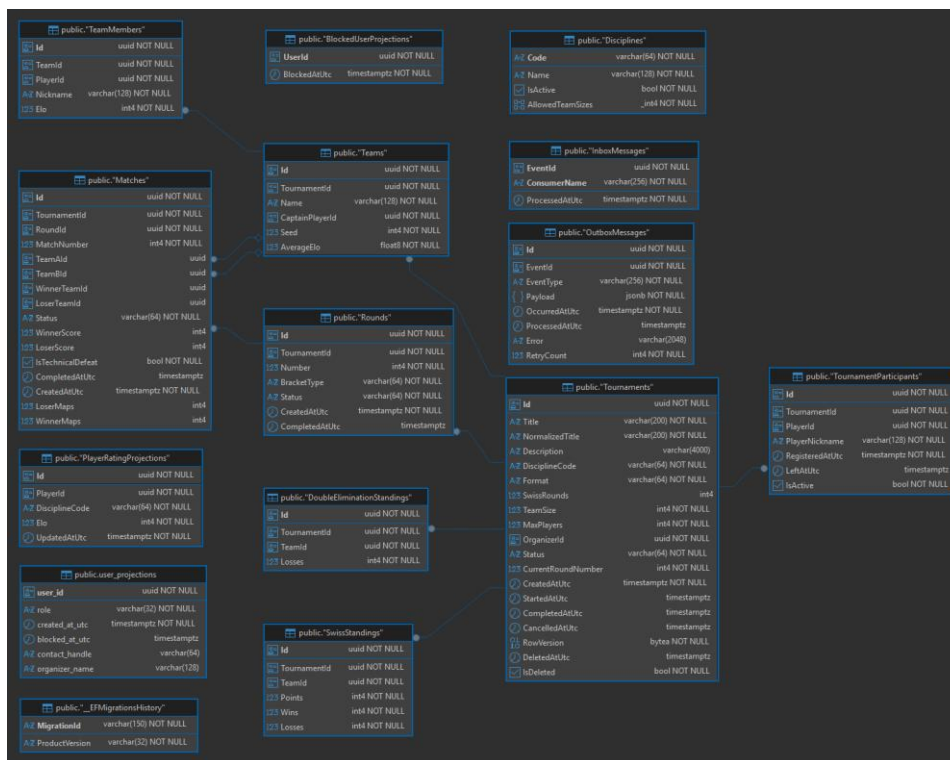


Рисунок 39 – Физическая модель базы данных сервиса управления турнирами

Таким образом, была описана физическая модель базы данных сервиса управления турнирами.

3.3.3 Реализация базы данных сервиса рейтинга и статистики

В сервисе рейтинга и статистики работа с базой данных реализована через контекст RatingDbContext. В данном контексте определены наборы DbSet для таблиц PlayerRatings и RatingHistories. Таблица PlayerRatings предназначена для хранения текущих рейтинговых показателей игроков по игровым дисциплинам, а таблица RatingHistories используется для хранения истории изменений рейтинга.

Также в контексте RatingDbContext определены таблицы InboxMessages и OutboxMessages.

Наличие этих таблиц позволяет сервису получать события о завершённых матчах, выполнять перерасчёт рейтинга и сохранять исходящие события, связанные с обновлением рейтинговых данных.

Физическая модель базы данных сервиса рейтинга и статистики представлена на рисунке 40.

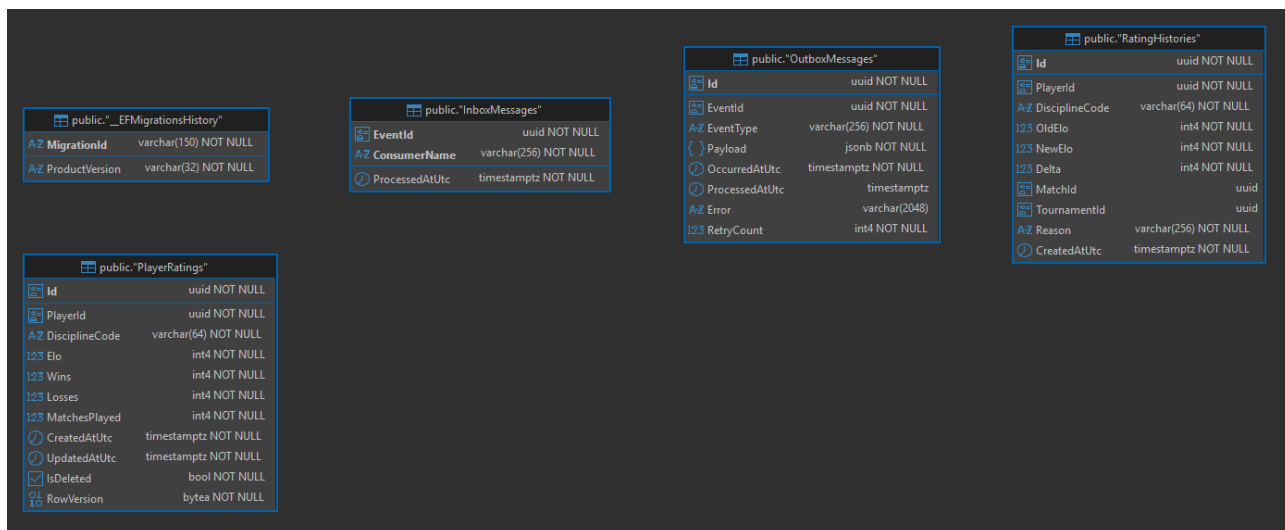


Рисунок 40 – Физическая модель базы данных сервиса рейтинга и статистики

Таким образом, была описана физическая модель базы данных сервиса рейтинга и статистики.

3.3.4 Реализация базы данных сервиса уведомлений

В сервисе уведомлений работа с базой данных реализована через контекст NotificationDbContext. В данном контексте определены наборы DbSet для таблиц Notifications и InboxMessages. Таблица Notifications предназначена для хранения пользовательских уведомлений, сформированных на основе событий, поступающих из других микросервисов.

Таблица InboxMessages используется для фиксации обработанных входящих интеграционных событий и предотвращения их повторной обработки. Таблица OutboxMessages в базе данных сервиса уведомлений не используется, поскольку данный сервис в текущей реализации не публикует исходящие интеграционные события.

Физическая модель базы данных сервиса уведомлений представлена на рисунке 41.

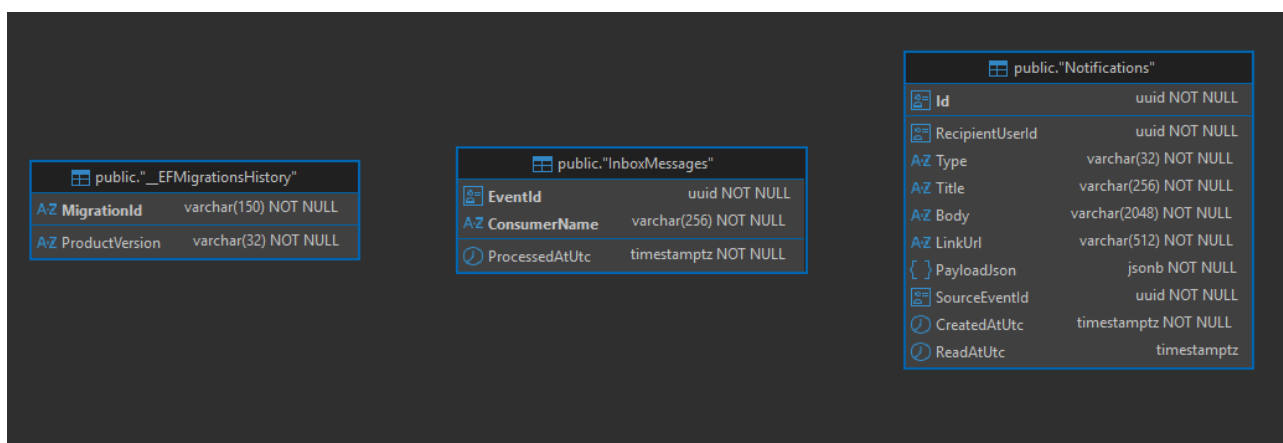


Рисунок 41 – Физическая модель базы данных сервиса уведомлений

Таким образом, была описана физическая модель базы данных сервиса уведомлений.

3.4 Реализация клиентской части приложения

В соответствии с проектными решениями клиентская часть веб-приложения была реализована с использованием подхода Feature-Sliced Design [33]. В структуре проекта выделены слои app, pages, features и shared.

Слой app содержит элементы инициализации приложения и настройки маршрутизации. Слой pages включает экранные компоненты, соответствующие пользовательским маршрутам. Слой features содержит функциональные модули, связанные с отдельными пользовательскими сценариями. Слой shared включает переиспользуемые элементы клиентской части: HTTP-клиент, средства работы с авторизацией, общие компоненты интерфейса, стили и вспомогательные функции [48].

Слои pages и features дополнительно разделены на слайсы. В слое pages слайсы соответствуют группам экранов для разных пользовательских ролей и сценариев: гостя, игрока, организатора, администратора, просмотра матча и уведомлений. В слое features слайсы соответствуют отдельным функциональным областям, таким как авторизация, управление турнирами, работа с рейтингами, административные функции и уведомления. Внутри слайсов выделяются сегменты, определяющие техническое назначение кода, например ui, model и api. Реализация структуры клиентской части на основе Feature-Sliced Design представлена на рисунке 42.

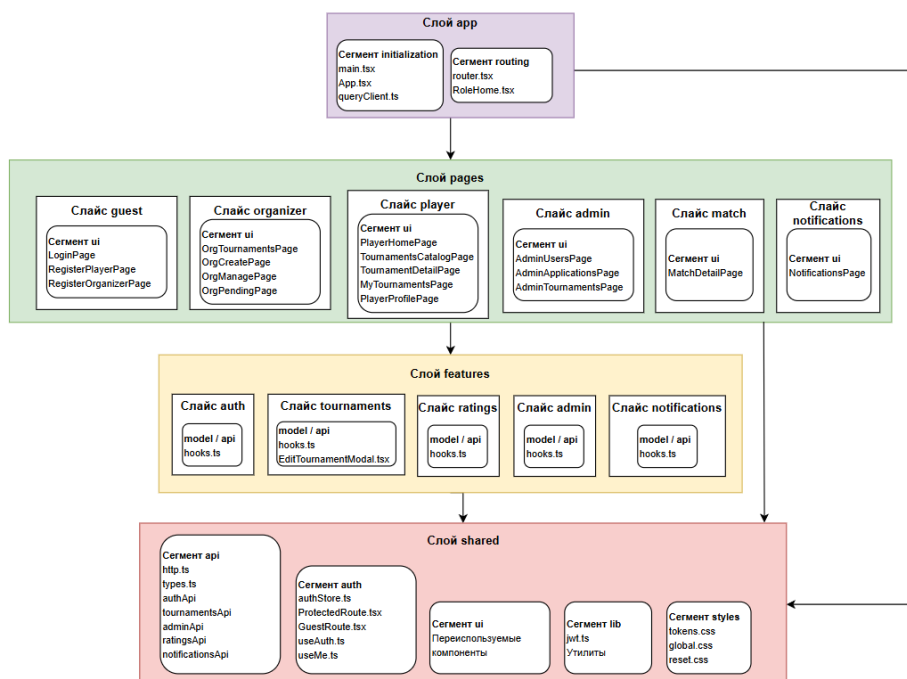


Рисунок 42 – Реализация структуры клиентской части на основе Feature-Sliced Design

Стрелками обозначено направление зависимостей между слоями. В соответствии с выбранным подходом верхние слои используют нижние, но не наоборот.

Клиентская часть веб-приложения реализована с использованием React, TypeScript и сборщика Vite. React используется для построения пользовательского интерфейса на основе компонентов [49], TypeScript обеспечивает статическую типизацию клиентского кода [50], а Vite применяется для формирования итоговой сборки [51]. Приложение выполнено в виде одностраничного веб-приложения, в котором переходы между экранами осуществляются средствами React Router [52].

Для работы с серверным состоянием используется библиотека TanStack React Query, позволяющая выполнять запросы к API, кэшировать полученные данные и обновлять состояние интерфейса после изменения данных [53]. Для отправки HTTP-запросов применяется Axios [54].

В проекте реализован общий HTTP-клиент, который добавляет токен доступа в заголовок Authorization при выполнении защищенных запросов и централизованно обрабатывает ошибки авторизации. Для хранения состояния авторизации используется Zustand. В клиентском хранилище сохраняются токен доступа и сведения о текущем пользователе [55].

Маршрутизация клиентской части построена с учетом ролевой модели системы. В приложении выделены публичные маршруты, доступные неавторизованному пользователю, а также защищенные маршруты для игроков, организаторов и администраторов.

Для ограничения доступа к страницам используется компонент ProtectedRoute, проверяющий наличие авторизации, роль пользователя и, при необходимости, активный статус организатора. Если токен отсутствует, истек или сервер возвращает ошибку авторизации, пользователь перенаправляется на страницу входа.

Взаимодействие клиентской части с серверными микросервисами осуществляется через HTTP API. При этом отдельный API Gateway в системе не

выделяется. Маршрутизация HTTP-запросов от клиентского приложения к соответствующим микросервисам осуществляется средствами nginx [56], входящего в состав контейнера пользовательского интерфейса и выполняющего функции обратного прокси-сервера.

Запросы к маршрутам `/api/auth` и `/api/admin` направляются в сервис управления пользователями, при этом запрос к `/api/admin/tournaments` выделен отдельным правилом и направляется в сервис управления турнирами. Запросы к `/api/tournaments` и `/api/organizer` – в сервис управления турнирами, запросы к `/api/ratings` – в сервис рейтинга, а запросы к `/api/notifications` – в сервис уведомлений.

Схема маршрутизации запросов клиентской части через nginx представлена на рисунке 43.

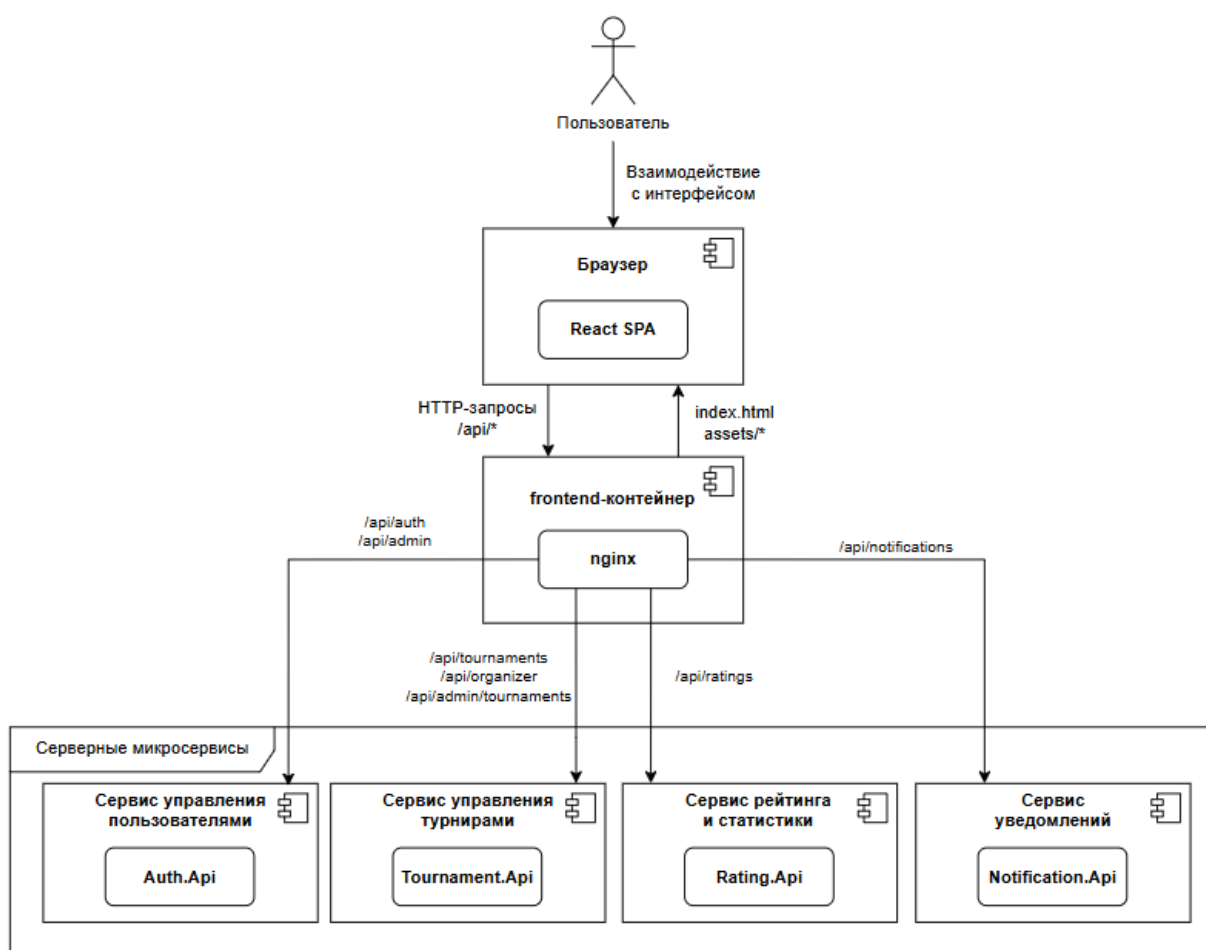


Рисунок 43 – Схема маршрутизации запросов клиентской части через nginx

Как видно из данной схемы, в реализованном веб-приложении обращение клиента происходит не напрямую к разработанным API микросервисам, а к серверу `nginx`, находящимся во `frontend`-контейнере.

Использование `nginx` в составе контейнера пользовательского интерфейса позволяет объединить раздачу статических файлов клиентского приложения и маршрутизацию API-запросов к серверным микросервисам.

При ограниченном числе микросервисов и четком разграничении их ответственности такой подход позволяет не выделять отдельный API Gateway. Вся бизнес-логика остается внутри соответствующих серверных сервисов, а `nginx` выполняет инфраструктурную функцию маршрутизации запросов и раздачи статических файлов. Кроме того, поскольку для браузера все обращения выполняются к единому источнику, такой подход устраняет необходимость в настройке политики совместного использования ресурсов между источниками (CORS) на стороне серверных микросервисов.

Сборка клиентской части выполняется средствами `Vite`. В результате сборки формируются статические файлы, которые размещаются в контейнере пользовательского интерфейса и раздаются `nginx`.

Для маршрутов клиентского приложения настроен `fallback` на `index.html`, что обеспечивает корректную работу одностраничного приложения при прямом переходе по внутренним URL.

Таким образом, клиентская часть веб-приложения была реализована в соответствии с проектными решениями. Использование `Feature-Sliced Design` позволило структурировать клиентский код по слоям, функциональным областям и техническим сегментам.

Применение `React`, `TypeScript`, `React Router`, `TanStack React Query`, `Axios` и `Zustand` обеспечило реализацию компонентного интерфейса, маршрутизации, взаимодействия с API и управления состоянием авторизации.

Использование `nginx` позволило организовать единый вход для клиентской части и маршрутизировать запросы к серверным микросервисам без выделения отдельного API Gateway.

4 Контроль качества программного продукта

Контроль качества программного продукта выполнялся с использованием функционального тестирования методом черного ящика, модульного тестирования и количественной оценки кода с помощью метрик. Такой набор методов выбран потому, что он позволяет проверить программный продукт на разных уровнях.

Функциональное тестирование методом черного ящика применяется для проверки поведения системы с точки зрения пользователя и корректности обработки входных данных.

Модульное тестирование используется для проверки внутренней бизнес-логики отдельных компонентов серверной части.

Метрики кода применяются для количественной оценки выбранного фрагмента реализации, позволяя рассмотреть его объем и сложность с точки зрения дальнейшего сопровождения.

4.1 Тестирование черным ящиком

Тестирование методом черного ящика – это вид тестирования, при котором при проверке не учитывается внутренняя структура приложения. На вход программы подаются наборы входных данных и полученные выходные данные сравниваются с ожидаемыми [57].

Рассмотрим данный вид тестирования на примере вводимого поля названия турнира.

Ниже в таблицах 11-12 определены классы эквивалентности и проведенные тест-кейсы.

Таблица 11 – Классы эквивалентности

	Правильные КЭ	Неправильные КЭ
Длина названия	5–50 (1)	0–4 (2); 51 и более (3)

Продолжение таблицы 11

Структура	Начинается с буквы или цифры (4) Заканчивается буквой или цифрой (5) Не состоит только из пробелов (6) Не состоит только из дефисов (7) Не более 1 пробела подряд (8) Не более 1 дефиса подряд (9)	Начинается не с буквы и не с цифры (10) Заканчивается не буквой и не цифрой (11) Состоит только из пробелов (12) Состоит только из дефисов (13) Более 1 пробела подряд (14) Более 1 дефиса подряд (15)
Содержание	Буквы кириллицы (16) Буквы латиницы (17) Цифры (18) Пробелы (19) Дефис (20)	Прочие символы (21)

Графическое изображение классов эквивалентности длины названия турниров представлено на рисунке 44.



Рисунок 44 – Графическое представление классов эквивалентности длины названия

Полное графическое изображение классов эквивалентности представлено в приложении Е.

На основе выделенных классов эквивалентности и граничных условий были разработаны тест-кейсы. Результаты функционального тестирования представлены в проекте на GitHub (рисунки 1–14) [58].

Таблица 12 – Тест-кейсы

Классы эквивалентности	Входные данные	Выходные данные
2,4,5,6,7,8,9,16	Куб	Ошибка: «Минимум 5 символов» (рисунок 1)
3,4,5,6,7,8,9,16	«Кубок Cyber 2026 Гранд Финал Летнего Игрового Сезона» (51+)	Ошибка: «Максимум 50 символов» (рисунок 2)

Продолжение таблицы 12

1,5,6,7,8,9,16,18,20,10	-Кубок2026	Ошибка: название начинается не с буквы и не с цифры (рисунок 3)
1,5,6,7,8,9,16,17,18,19,20,10	Кубок2026-	Ошибка: название заканчивается не буквой и не цифрой (рисунок 4)
1,4,5,7,8,9,19,12		Ошибка: название состоит только из пробелов (рисунок 5)
1,4,5,6,8,9,20,13	-----	Ошибка: название состоит только из дефисов (рисунок 6)
1,4,5,6,7,9,16,18,19,14	Кубок 2026	Ошибка: более 1 пробела подряд (рисунок 7)
1,4,5,6,7,8,16,18,20,15	Кубок--2026	Ошибка: более 1 дефиса подряд (рисунок 8)
1,4,5,6,7,8,9,21	Кубок@2026	Ошибка: недопустимые символы (рисунок 9)
1,4,5,6,7,8,9,16	Кубок	Успешно (рисунок 10)
1,4,5,6,7,8,9,17	CyberCup	Успешно (рисунок 11)
1,4,5,6,7,8,9,18	123456	Успешно (рисунок 12)
1,4,5,6,7,8,9,16,19	Кубок 2026	Успешно (рисунок 13)
1,4,5,6,7,8,9,16,20	Кубок-2026	Успешно (рисунок 14)
1,4,5,6,7,8,9,16,17,18,19,20	Кубок Cyber-2026	Успешно (рисунок 15)

Таким образом, функциональное тестирование черным ящиком подтвердило корректную работу проверки вводимых данных при создании турнира. Система корректно обрабатывает допустимые и недопустимые значения поля.

4.1.1 Тестирование с помощью юнит-тестов

Модульное тестирование представляет собой процесс проверки отдельных компонентов программы в изолированном виде с целью подтверждения корректности их работы. Основная задача данного вида тестирования – убедиться, что каждый модуль реализует заданную логику независимо от остальной системы [59].

В данной работе модульное тестирование применялось для проверки класса MatchResultService (подсистема Tournament), реализующего сценарий фиксации результата матча. Данный сервис отвечает за обработку завершения матча: он проверяет права пользователя, контролирует состояние турнира и

матча, проверяет допустимость указанного победителя и корректность счёта, переводит матч в завершённое состояние и публикует событие `MatchCompletedEvent`. Выбор данного компонента обусловлен тем, что фиксация результата матча является одним из ключевых бизнес-сценариев платформы, от которого зависит дальнейшее продвижение команд по турнирной сетке.

Для реализации модульных тестов был выбран фреймворк `xUnit 2.5.3`. Каждый проверяемый сценарий оформлен как отдельный тестовый метод с атрибутом `[Fact]`, что обеспечивает независимое выполнение и наглядное отображение результатов.

Зависимости тестируемого сервиса (репозиторий турниров и публикация событий) заменены на упрощённые тестовые реализации, хранящие данные в памяти, что позволяет проверять логику метода `CompleteMatchAsync` в изоляции от базы данных и внешней инфраструктуры.

Набор разработанных тест-кейсов приведён в таблице 13.

Таблица 13 – Модульные тест-кейсы

Название теста	Описание
<code>OrganizerOwner_CanCompleteMatch_AndPublishesMatchCompletedOnce</code>	Проверяет, что организатор, являющийся владельцем турнира, может завершить матч, а событие завершения матча публикуется один раз
<code>OtherOrganizer_GetsAccessDenied</code>	Проверяет, что организатор другого турнира не может завершить матч
<code>Player_GetsAccessDenied</code>	Проверяет, что пользователь с ролью игрока не имеет права завершать матч
<code>Admin_CanCompleteAnyMatch</code>	Проверяет, что администратор может завершить любой матч независимо от принадлежности турнира
<code>InvalidWinner_ReturnsFailure</code>	Проверяет, что система отклоняет результат, если указанный победитель не является участником матча
<code>WinnerScoreMustBeGreaterThanLoserScore</code>	Проверяет, что счет победителя должен быть больше счета проигравшего
<code>WinnerMapsMustBeGreaterThanLoserMaps</code>	Проверяет, что количество выигранных карт у победителя должно быть больше, чем у проигравшего
<code>ValidMapsAndRoundsScore_StoresBoth</code>	Проверяет корректное сохранение счета по раундам и количества выигранных карт
<code>RegularMatchWithoutScore_ReturnsFailure</code>	Проверяет, что обычный матч нельзя завершить без указания счета

Продолжение таблицы 13

TechnicalDefeatWithoutScore_Succeeds_AndStoresNullScore	Проверяет, что матч может быть завершен техническим поражением без указания счета
CompletingAlreadyCompletedMatch_ReturnsConflictAndDoesNotPublishTwice	Проверяет, что уже завершенный матч нельзя завершить повторно, а событие завершения не публикуется второй раз

На рисунке 45 представлен результат успешного выполнения юнит-тестов.

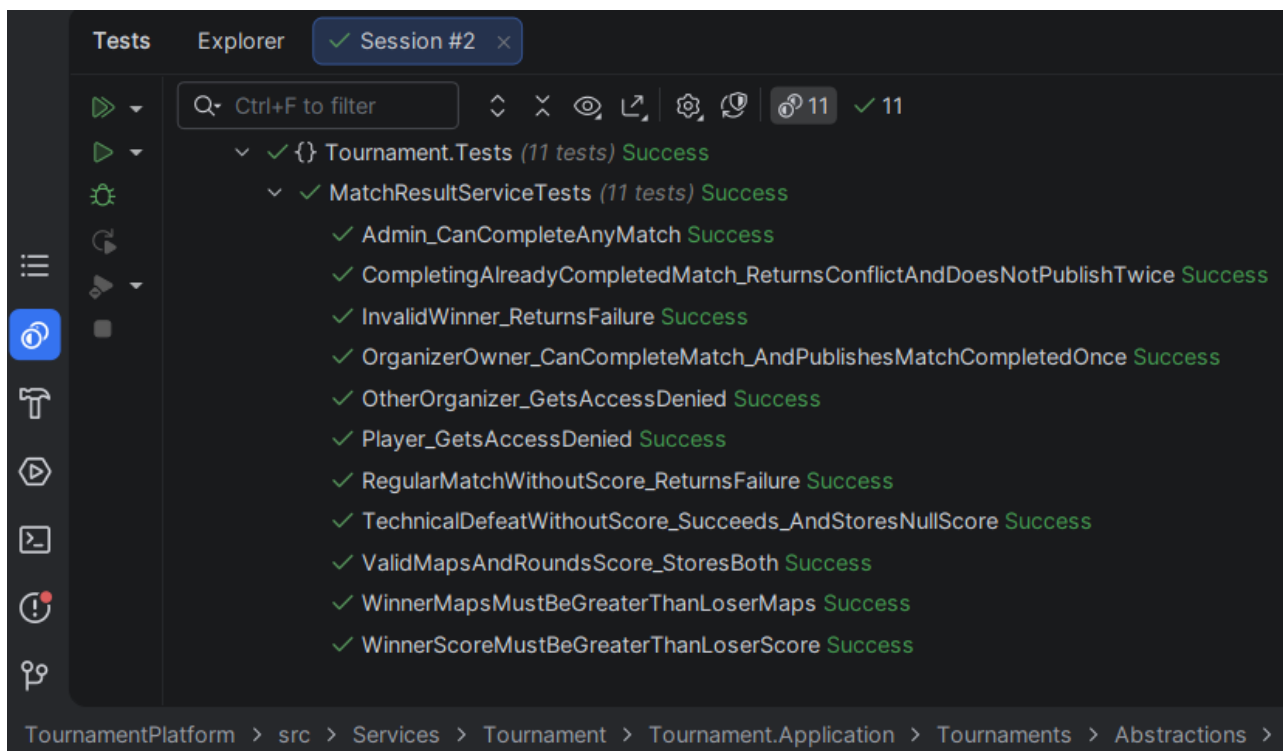


Рисунок 45 – Результат выполнения юнит-тестов

Успешное прохождение данных тестов подтверждает корректность работы сервиса обработки результатов матчей: система учитывает права пользователей, проверяет допустимость победителя, контролирует корректность счета, поддерживает сценарий технического поражения и предотвращает повторную обработку завершенного матча.

Исходный код данных тестов приведен в приложении Ж.

Таким образом, выполненное юнит-тестирование подтвердило корректность одного из ключевых сценариев серверной части системы. Проверенные сценарии показывают, что результат матча сохраняется только при выполнении необходимых условий, а некорректные действия пользователей обрабатываются с возвратом соответствующих ошибок.

4.2 Оценка кода с помощью метрик

Метрики кода применяются для количественной оценки отдельных характеристик программной реализации: сложности, объема, читаемости и удобства сопровождения [60]. В рамках данной работы для анализа был выбран метод `CompleteMatchAsync` из класса `MatchResultService`. Реализация данного метода приведена в приложении Г (листинг Г.2).

Данный метод реализует сценарий фиксации результата матча в турнире. Он выполняет поиск турнира и матча, проверяет права пользователя, контролирует состояние турнира и матча, проверяет корректность победителя и счета, после чего переводит матч в завершенное состояние, публикует событие `MatchCompletedEvent` и передает управление генератору турнирной сетки. Выбор данного метода обусловлен тем, что обработка результата матча является важным бизнес-сценарием турнирной платформы и от него зависит дальнейшее продвижение команд по сетке и изменение состояния турнира.

В следующих разделах данный метод будет оценен с помощью определенных метрик, позволяющих понять сложность и качество реализованного кода.

4.2.1 Обоснование выбора метрик кода

Для количественной оценки реализации выбранного фрагмента серверной части были использованы метрика SLOC и метрика Холстеда. Эти показатели позволяют рассмотреть метод не только с точки зрения его функционального назначения, но и с позиции объема кода, количества операций, числа используемых сущностей и общей трудоемкости восприятия.

Метрика SLOC применяется для определения размера программного фрагмента по количеству строк исходного кода [61].

При расчете обычно учитываются только содержательные строки, то есть строки, в которых расположены операторы, вызовы методов, условия, объявления переменных и другие элементы реализации. Пустые строки и

комментарии в расчет не включаются. За счет этого SLOC позволяет оценить фактический объем метода без учета поясняющего текста и визуального форматирования. Данная метрика удобна при первичном анализе, поскольку дает простое и наглядное представление о размере исследуемого участка программы.

Метрика Холстеда используется для более детальной оценки сложности программного кода [61].

В отличие от SLOC, она учитывает не только количество строк, но и внутренний состав метода: операторы, операнды и частоту их использования. В основе расчета находятся четыре исходных показателя: число уникальных операторов, число уникальных операндов, общее количество операторов и общее количество операндов. На их основе определяются словарь программы, длина программы, объем, сложность и усилие, необходимое для понимания или реализации рассматриваемого фрагмента.

Выбор метрик SLOC и Холстеда обусловлен особенностями данного метода. `CompleteMatchAsync` содержит несколько последовательных проверок, ранние возвраты при ошибках и обращения к нескольким компонентам системы. Поэтому для его анализа важно учитывать как физический объем кода, так и количество логических элементов, участвующих в обработке сценария. Совместное применение SLOC и метрики Холстеда позволяет получить более полное представление о сложности метода и оценить удобство его дальнейшего сопровождения.

4.2.2 Метрика SLOC

В качестве количественного показателя объема кода использовалась метрика SLOC. Она отражает число содержательных строк исходного кода, то есть строк, которые участвуют в реализации логики метода. При подсчете из общего количества строк исключаются пустые строки и комментарии, поскольку они не влияют на выполнение программы, а используются только для оформления и пояснения кода.

Расчет SLOC был выполнен для полного листинга метода CompleteMatchAsync. Полученные исходные значения представлены в таблице 14.

Таблица 14 – Исходные данные для расчета метрики SLOC

Характеристика	Значение
Всего строк метода	98
Количество пустых строк	13
Количество строк комментариев	14

Итоговое значение метрики определяется следующим образом:

$$\text{SLOC} = 98 - 13 - 14 = 71.$$

Следовательно, содержательная часть метода составляет 71 строку исходного кода.

Исключение комментариев и пустых строк из расчета связано с тем, что они не являются исполняемыми элементами программы. Комментарии в данном методе используются для пояснения отдельных этапов обработки результата матча, а пустые строки разделяют код на смысловые блоки и повышают читаемость листинга.

Полученное значение SLOC показывает, что метод CompleteMatchAsync имеет средний объем.

Его размер обусловлен тем, что метод не ограничивается одним действием, а выполняет полный цикл обработки результата матча: проверяет существование турнира, права пользователя, корректность состояния матча, валидность победителя и счета, а затем изменяет состояние доменной модели и сохраняет результат.

Таким образом, по метрике SLOC метод нельзя отнести к чрезмерно крупным, однако он содержит достаточно большое количество проверок.

Это объясняется спецификой предметной области: завершение матча влияет на турнирную сетку и дальнейшее состояние турнира, поэтому перед сохранением результата необходимо последовательно исключить некорректные

сценарии.

Таким образом, рассмотренная метрика SLOC показала, что выбранный метод имеет небольшой размер и хорошо структурированную реализацию бизнес-логики.

4.2.3 Метрика Холстеда

Для анализа сложности выбранного фрагмента серверной части была использована метрика Холстеда. Она позволяет оценить программный код не только по количеству строк, но и по составу используемых языковых конструкций: операторов, операндов и частоты их появления.

В качестве объекта анализа выбран метод `CompleteMatchAsync` класса `MatchResultService`.

Данный метод отвечает за обработку результата матча в турнире. В его логике выполняются проверка существования турнира, контроль прав пользователя, поиск матча, проверка состояния турнира и матча, валидация победителя и счета, фиксация результата, публикация события завершения матча и сохранение изменений.

При подсчете использовались следующие правила:

- комментарии, пустые строки, скобки, запятые, точки с запятой, точки и угловые скобки не учитывались;
- вызовы методов рассматривались как операторы;
- идентификаторы, типы, свойства, значения перечислений и литералы учитывались как операнды.

Для расчета метрики Холстеда используются следующие показатели:

- $n1$ – количество уникальных операторов;
- $n2$ – количество уникальных операндов;
- $N1$ – общее количество операторов;
- $N2$ – общее количество операндов.

Результаты подсчета приведены в таблице 15.

Таблица 15 – Подсчет операторов и операндов метода CompleteMatchAsync

Оператор	Количество	Операнд	Количество
return	11	request	22
if	10	match	13
Failure	10	Result	12
is	8	TournamentDetailsResponse	12
&&	6	null	12
	5	TournamentErrors	10
!=	4	tournament	9
is not	4	cancellationToken	4
=	4	WinnerScore	4
var	3	LoserScore	4
await	3	WinnerMaps	4
!	2	LoserMaps	4
==	2	tournamentId	3
<=	2	WinnerTeamId	3
=>	2	Guid	2
public	1	matchId	2
async	1	currentUser	2
default	1	tournaments	2
GetByIdAsync	1	round	2
CanComplete	1	Status	2
SelectMany	1	TeamAId	2
SingleOrDefault	1	TeamBId	2
Complete	1	IsTechnicalDefeat	2
Add	1	InvalidMatchScore	2
ToMatchCompletedEvent	1	generator	2
GetGenerator	1	Прочие единоразовые операнды	26
HandleMatchCompletedAsync	1	TournamentNotFound	1
SaveChangesAsync	1	AccessDenied	1
Success	1	ParticipantNotFound	1
ToDetailsResponse	1	MatchAlreadyCompleted	1
-	-	TournamentNotInProgress	1
-	-	TournamentRegistrationClosed	1
-	-	InvalidWinnerTeam	1
-	-	MatchScoreRequired	1
-	-	MatchStatus.Completed	1
-	-	TournamentStatus.InProgress	1
-	-	DateTime	1
-	-	UtcNow	1
-	-	outboxWriter	1
-	-	bracketGeneratorFactory	1
-	-	Format	1

Продолжение таблицы 15

Оператор	Количество	Операнд	Количество
-	-	Прочие единоразовые операнды (Task, CompleteMatchRequest, CurrentTournamentUser, MatchStatus, TournamentStatus, ToMatchCompletedEvent, TournamentResponseMapper и др.)	11

Таким образом, после сложения всех уникальных операторов $n1 = 30$, после сложения всех уникальных операндов $n2 = 51$, после сложения всех вхождений операторов $N1 = 91$, после сложения всех вхождений операндов $N2 = 164$.

На основе метрик Холстеда можно вычислить целый ряд характеристик программы. Рассмотрим некоторые из них.

Расчёт словаря программы производится по формуле:

$$n = n1 + n2 = 30 + 51 = 81,$$

где $n1$ – количество операторов в программе;

$n2$ – количество операндов программы.

Расчёт длины программы производится по формуле:

$$N = N1 + N2 = 91 + 164 = 255,$$

где $N1$ – число вхождений всех операторов в программе;

$N2$ – число вхождений всех операндов программы.

Расчёт объёма программы производится по формуле:

$$зV = N * \log_2 n = 255 * \log_2 81 = 1616,7,$$

где N – длина программы;

n – словарь программы.

Расчёт сложности программы производится по формуле:

$$D = (n1/2) * (N2/n2) = (30/2) * (164/51) = 48,24,$$

где $n1$ – количество операторов в программе;

$n2$ – количество операндов программы;

$N2$ – число вхождений всех операндов программы.

Расчёт оценки интеллектуальных усилий производится по формуле:

$$E = D * V = 48,24 * 1616,7 = 77980,16,$$

где D – сложность программы;

V – объем программы.

Общая сводка полученных результатов представлена в таблице 16.

Таблица 16 – Результаты расчета метрики Холстеда

Характеристика	Значение
Словарь программы	81
Длина программы	255
Объем программы	1616,7
Сложность программы	48,24
Усилия разработки	77980,16

Полученные значения показывают, что метод `CompleteMatchAsync` содержит значительное количество операторов и операндов. Это связано с тем, что метод реализует не одно простое действие, а полный сценарий обработки результата матча.

Наибольший вклад в сложность вносят проверки прав доступа, состояния турнира и матча, корректности победителя, обязательности счета и невозможности повторного завершения матча.

Метрика Холстеда показывает, что метод имеет умеренно высокую сложность. При этом его структура остается последовательной: сначала выполняются проверки ошибочных ситуаций, затем фиксируется результат матча, публикуется событие и сохраняются изменения.

Такая организация кода упрощает понимание метода, несмотря на большое количество условий.

Таким образом, на данном этапе было выполнено тестирование разработанного веб-приложение.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано веб-приложение для организации и проведения турниров по компьютерным играм, обеспечивающее взаимодействие игроков, организаторов и администратора на всех этапах проведения соревнований. Поставленная цель достигнута, все сформулированные во введении задачи решены.

На этапе анализа предметной области были изучены существующие сервисы, проведён их сравнительный анализ и выявлены основные недостатки. На основе анализа сформулированы функциональные и нефункциональные требования, построены диаграммы вариантов использования, деятельности и последовательности, а также выполнено планирование разработки и оценка бюджета проекта средствами Microsoft Project.

На этапе проектирования обоснована и разработана архитектура серверной части на основе микросервисного подхода. Внутренняя структура каждого сервиса спроектирована по принципам чистой архитектуры, а межсервисное взаимодействие на основе асинхронных интеграционных событий с применением паттернов Outbox и Inbox.

Разработаны концептуальная и логические модели баз данных. Отдельно спроектирована система внутреннего рейтинга: задача формирования сбалансированных команд была рассмотрена как задача целочисленного линейного программирования, а пересчёт рейтинга игроков основан на системе ELO. Спроектирована клиентская часть с ролевой навигацией, подготовлены макеты пользовательского интерфейса.

На этапе реализации серверная часть выполнена на платформе ASP.NET Core с использованием СУБД PostgreSQL и брокера сообщений RabbitMQ. Реализованы программные компоненты всех четырёх микросервисов. Был реализован алгоритм формирования команд. Реализована поддержка нескольких форматов проведения турниров. Также была разработана клиентская часть приложения.

На этапе контроля качества проведено функциональное и модульное тестирование, а также оценка качества кода. Полученные результаты подтвердили, что метод соответствует умеренной сложности при читаемой структуре кода.

С точки зрения технико-экономических показателей общая трудоёмкость проекта составляет 736 человеко-часов, а суммарные затраты на разработку – примерно 523 000 рублей. Применение разработанного приложения позволяет автоматизировать операции, ранее выполнявшиеся вручную: формирование сбалансированных по уровню подготовки команд и пересчёт рейтинга участников по результатам матчей. В отличие от рассмотренных аналогов, приложение объединяет в едином сервисе поддержку русского языка, нескольких форматов турниров, регистрацию игроков, внутреннюю рейтинговую систему и просмотр турниров других организаторов, что повышает удобство и вовлечённость всех участников соревновательного процесса.

Практическая значимость разработанного решения подтверждена справкой об апробации результатов выпускной квалификационной работы, выданной ООО «АБ СОФТ». В справке отмечено, что результаты работы представляют практический интерес для компании, участвующей в разработке и сопровождении мобильной киберспортивной игры, в рамках которой организуются онлайн- и офлайн-турниры.

Разработанное приложение рекомендовано к дальнейшему использованию при разработке и совершенствовании внутренних инструментов проведения турниров. Справка об апробации результатов работы представлена в приложении И.

Таким образом, в результате работы создано законченное программное решение, отвечающее поставленным требованиям.

Дальнейшее развитие возможно в направлении расширения числа поддерживаемых дисциплин и форматов турниров, добавления новых типов уведомлений и интеграции с внешними сервисами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. DataReportal. Digital 2026: Russian Federation // DataReportal [Электронный ресурс]. URL: <https://datareportal.com/reports/digital-2026-russian-federation> (дата обращения: 01.04.2026).
2. Gaming in Russia // forgamedev.ru [Электронный ресурс]. URL: <https://gaminginrussia.forgamedev.ru/> (дата обращения: 01.04.2026).
3. Новые исследования показывают, что видеоигры могут быть полезны для подростков // Forbes Education [Электронный ресурс]. URL: <https://www.forbes.ru/education/513448-novye-issledovania-pokazyvaut-cto-videoigry-mogut-byt-polezny-dla-podrostkov> (дата обращения: 01.04.2026).
4. Приказ Министерства спорта Российской Федерации от 29 апреля 2016 г. № 470 «О признании и включении во Всероссийский реестр видов спорта спортивных дисциплин, видов спорта и внесении изменений во Всероссийский реестр видов спорта» // КонсультантПлюс [Электронный ресурс]. URL: https://www.consultant.ru/document/cons_doc_LAW_199135/2ff7a8c72de3994f30496a0ccbb1ddafdaddf518/ (дата обращения: 17.04.2026).
5. Федерация компьютерного спорта России. О федерации // РЭСФ [Электронный ресурс]. URL: <https://resf.ru/about/resf/> (дата обращения: 17.04.2026).
6. Сбер стал титульным партнером Федерации компьютерного спорта России // Федерация компьютерного спорта России [Электронный ресурс]. URL: <https://www.resf.ru/news/Sber-become-title-partner-Russian-eSports-Federation/> (дата обращения: 20.04.2026).
7. Challonge // Challonge [Электронный ресурс]. URL: <https://challonge.com> (дата обращения: 21.04.2026).
8. The Tournament // The Tournament [Электронный ресурс]. URL: <https://the-tournament.net/en> (дата обращения: 17.05.2026).
9. Battlefy // Battlefy [Электронный ресурс]. URL: <https://battlefy.com/> (дата обращения: 21.04.2026).

10. Рамбо, Дж. UML 2.0. Объектно-ориентированное моделирование и разработка / Дж. Рамбо, М. Блаха. – 2-е изд. – СПб. : Питер, 2007. – 544 с.
11. Rational Unified Process: Best Practices for Software Development Teams // IBM Rational [Электронный ресурс]. URL: <https://public.dhe.ibm.com/software/rational/web/datasheets/version6/rup.pdf> (дата обращения: 05.04.2026).
12. Unified Modeling Language (UML), version 2.5.1 // Object Management Group [Электронный ресурс]. URL: <https://www.omg.org/spec/UML/2.5.1/About-UML> (дата обращения: 05.04.2026).
13. ASP.NET Core documentation // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/core/> (дата обращения: 10.04.2026).
14. PostgreSQL Documentation // The PostgreSQL Global Development Group [Электронный ресурс]. URL: <https://www.postgresql.org/docs/> (дата обращения: 10.04.2026).
15. Entity Framework Core documentation // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/ef/core/> (дата обращения: 10.04.2026).
16. RabbitMQ Documentation // RabbitMQ [Электронный ресурс]. URL: <https://www.rabbitmq.com/docs> (дата обращения: 20.05.2026).
17. Docker overview // Docker Inc. [Электронный ресурс]. URL: <https://docs.docker.com/get-started/docker-overview/> (дата обращения: 12.04.2026).
18. Ричардс, М. Фундаментальный подход к программной архитектуре / М. Ричардс, Н. Форд. – Б.м.: б.и. [Электронный ресурс]. URL: <https://library.tsilikin.ru/Техника/Программирование/Desing/Фундаментальный%20подход%20к%20программной%20архитектуре.pdf> (дата обращения: 15.04.2026).
19. Common web application architectures // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application->

architectures (дата обращения: 15.04.2026).

20. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. – СПб. : Питер, 2018. – 352 с.

21. Эванс, Э. Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем / Э. Эванс. – М. : Вильямс, 2011. – 448 с.

22. Transactional outbox pattern // microservices.io [Электронный ресурс]. URL: <https://microservices.io/patterns/data/transactional-outbox.html> (дата обращения: 20.04.2026).

23. Outbox, Inbox patterns and delivery guarantees explained // event-driven.io [Электронный ресурс]. URL: https://event-driven.io/en/outbox_inbox_patterns_and_delivery_guarantees_explained/ (дата обращения: 20.04.2026).

24. Chapter 2: Fundamental Concepts of the Entity-Relationship Model // Pressbooks, University of Iowa [Электронный ресурс]. URL: <https://pressbooks.uiowa.edu/fundamentalsofdatabasemanagement/chapter/chapter-2/> (дата обращения: 22.04.2026).

25. Introduction to the Entity-Relationship Model // University of California, Davis [Электронный ресурс]. URL: <https://web.cs.ucdavis.edu/~green/courses/ecs165a-w11/2-er.pdf> (дата обращения: 22.04.2026).

26. Chapter 8: Entity Relationship Model // BCcampus Open Education [Электронный ресурс]. URL: <https://opentextbc.ca/dbdesign01/chapter/chapter-8-entity-relationship-model/> (дата обращения: 22.04.2026).

27. What is the difference between logical and physical data models? // Amazon Web Services [Электронный ресурс]. URL: <https://aws.amazon.com/compare/the-difference-between-logical-and-physical-data-model/> (дата обращения: 25.04.2026).

28. Conceptual, Logical, and Physical Data Models // Redgate Blog [Электронный ресурс]. URL: <https://www.red-gate.com/blog/conceptual-logical->

physical-data-model/ (дата обращения: 25.04.2026).

29. Использование рейтинга Эло в задачах оценки уровня подготовки // Haykaru.ru [Электронный ресурс]. URL: <https://naukaru.ru/ru/nauka/article/72373/view> (дата обращения: 28.04.2026).

30. Еро Аладар. Методы оптимальных решений в задачах целочисленного программирования // Институт проблем управления РАН [Электронный ресурс]. URL: <https://www.ipu.ru/sites/default/files/publications/12896/406-12896.pdf> (дата обращения: 28.04.2026).

31. Зорина, Н. С. Методы оптимальных решений: учебное пособие / Н. С. Зорина. – Пенза: РГРТУ, 2015 [Электронный ресурс]. URL: <https://www.rubinst.ru/sites/default/files/static/vuz/Departaments/EiU/directions/E/Tutorials/Методы%20оптимальных%20решений.%20Зорина%20Н.С.%2С%202015.pdf> (дата обращения: 28.04.2026).

32. Elo, A. E. The Rating of Chessplayers, Past and Present / A. E. Elo. – New York: Arco Publishing, 1978. – 206 p.

33. Overview – Feature-Sliced Design // Feature-Sliced Design [Электронный ресурс]. URL: <https://feature-sliced.design/docs/get-started/overview> (дата обращения: 30.04.2026).

34. Проектирование пользовательского интерфейса: учебное пособие // Пензенский государственный университет [Электронный ресурс]. URL: https://vt.pnzgu.ru/files/vt.pnzgu.ru/int_2_.pdf (дата обращения: 30.04.2026).

35. Спроектированные макеты экранов: веб-приложение для организации и проведения турниров по компьютерным играм // GitHub [Электронный ресурс]. URL: <https://github.com/1sAXC/TournamentPlatform/blob/master/docs/ui-mockups.pdf> (дата обращения: 13.06.2026).

36. Диаграмма компонентов // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Диаграмма_компонентов (дата обращения: 05.05.2026).

37. Диаграммы развертывания // Intuit.ru [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/32/32/lecture/1024> (дата обращения: 05.05.2026).
38. Исходный код веб-приложения для организации и проведения турниров по компьютерным играм // GitHub [Электронный ресурс]. URL: <https://github.com/1sAXC/TournamentPlatform> (дата обращения: 13.06.2026).
39. UML Class Diagram // Ensi Tech [Электронный ресурс]. URL: <https://docs.ensi.tech/analyst-guides/tools/diagrams/uml/class-diagram> (дата обращения: 05.05.2026).
40. RabbitMQ tutorial – «Hello World!» (.NET) // RabbitMQ [Электронный ресурс]. URL: <https://www.rabbitmq.com/tutorials/tutorial-one-dotnet> (дата обращения: 20.05.2026).
41. Database per service pattern // microservices.io [Электронный ресурс]. URL: <https://microservices.io/patterns/data/database-per-service.html> (дата обращения: 10.05.2026).
42. Richardson C. Idempotent Consumer // Microservices.io [Электронный ресурс]. URL: <https://microservices.io/patterns/communication-style/idempotent-consumer.html> (дата обращения: 13.06.2026).
43. Введение в OR-Tools // Google for Developers [Электронный ресурс]. URL: <https://developers.google.com/optimization/introduction?hl=ru> (дата обращения: 13.06.2026).
44. Оптимизация с ограничениями (Constraint Optimization) // Google for Developers [Электронный ресурс]. URL: <https://developers.google.com/optimization/cp?hl=ru> (дата обращения: 13.06.2026).
45. Create a new database using Code First // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/ef/ef6/modeling/code-first/workflows/new-database> (дата обращения: 12.05.2026).
46. DbContext Class // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext> (дата обращения: 12.05.2026).

12.05.2026).

47. Migrations overview – EF Core // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/> (дата обращения: 12.05.2026).

48. Reference layers – Feature-Sliced Design // Feature-Sliced Design [Электронный ресурс]. URL: <https://feature-sliced.design/ru/docs/reference/layers> (дата обращения: 30.04.2026).

49. Quick Start // React [Электронный ресурс]. URL: <https://react.dev/learn> (дата обращения: 15.05.2026).

50. Static type checking // React Legacy Docs [Электронный ресурс]. URL: <https://ru.legacy.reactjs.org/docs/static-type-checking.html> (дата обращения: 15.05.2026).

51. Getting Started // Vite [Электронный ресурс]. URL: <https://vite.dev/guide/> (дата обращения: 15.05.2026).

52. Route Object // React Router [Электронный ресурс]. URL: <https://reactrouter.com/start/data/route-object> (дата обращения: 15.05.2026).

53. TanStack Query – Overview // TanStack [Электронный ресурс]. URL: <https://tanstack.com/query/latest> (дата обращения: 16.05.2026).

54. Axios Documentation // Axios [Электронный ресурс]. URL: <https://axios-js.com/docs/index.html> (дата обращения: 16.05.2026).

55. Getting started – Zustand // Pmnd.rs [Электронный ресурс]. URL: <https://zustand.docs.pmnd.rs/learn/getting-started/introduction> (дата обращения: 16.05.2026).

56. Nginx Beginner's Guide // NGINX [Электронный ресурс]. URL: https://nginx.org/ru/docs/beginners_guide.html (дата обращения: 17.05.2026).

57. Тестирование методом «черного ящика» // tquality.ru [Электронный ресурс]. URL: <https://tquality.ru/blog/testirovanie-metodom-chernogo-yashchika/> (дата обращения: 08.05.2026).

58. Результаты тестирования черным ящиком: веб-приложение для организации и проведения турниров по компьютерным играм // GitHub

[Электронный ресурс]. URL:
<https://github.com/1sAXC/TournamentPlatform/blob/master/docs/blackbox-test-results.pdf> (дата обращения: 13.06.2026).

59. Хориков, В. Принципы юнит-тестирования / В. Хориков. – СПб.: Питер, 2022. – 320 с.

60. Метрики качества ПО и целевые показатели проекта: за чем следить и как ими управлять // Surf [Электронный ресурс]. URL: <https://surf.ru/metrics-of-program-development/> (дата обращения: 10.05.2026).

61. Программный код и его метрики // Habr [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 11.05.2026).

Приложение А

(рекомендуемое)

Спецификация логических моделей баз данных

Таблица А.1 – Спецификация атрибутов отношения «Пользователи»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор пользователя	UUID	Да	Нет	Да	Нет	Уникальный идентификатор пользователя
Электронная почта	VARCHAR(320)	Да	Да	Нет	Нет	Адрес электронной почты (уникален без учёта регистра)
Хэш пароля	VARCHAR(1024)	Да	Нет	Нет	Нет	Хэш пароля
Роль	VARCHAR(32)	Да	Нет	Нет	Нет	Роль: игрок, организатор, администратор
Статус	VARCHAR(32)	Да	Нет	Нет	Нет	Статус учётной записи
Никнейм	VARCHAR(64)	Нет	Да	Нет	Нет	Никнейм игрока (уникален среди заполненных)
Название организатора	VARCHAR(128)	Нет	Нет	Нет	Нет	Название организатора
Контакт для связи	VARCHAR(64)	Нет	Нет	Нет	Нет	Контактный идентификатор
Идентификатор создавшего администратора	UUID	Нет	Нет	Нет	Нет	Администратор, создавший запись
Дата создания	TIMESTAMP	Да	Нет	Нет	Нет	Дата создания
Дата одобрения	TIMESTAMP	Нет	Нет	Нет	Нет	Дата одобрения заявки организатора
Дата отклонения	TIMESTAMP	Нет	Нет	Нет	Нет	Дата отклонения заявки
Дата блокировки	TIMESTAMP	Нет	Нет	Нет	Нет	Дата блокировки

Таблица А.2 – Спецификация атрибутов отношения «Турниры»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор турнира	UUID	Да	Нет	Да	Нет	Уникальный идентификатор турнира
Название	VARCHAR(200)	Да	Да	Нет	Нет	Название турнира (уникально без учёта регистра)
Описание	VARCHAR(4000)	Нет	Нет	Нет	Нет	Описание турнира
Код дисциплины	VARCHAR(64)	Да	Нет	Нет	Нет	Код игровой дисциплины
Формат проведения	VARCHAR(64)	Да	Нет	Нет	Нет	Формат: SingleElimination, DoubleElimination, Swiss
Статус	VARCHAR(64)	Да	Нет	Нет	Нет	Статус турнира
Идентификатор организатора	UUID	Да	Нет	Нет	Нет	Организатор турнира
Размер команды	INTEGER	Да	Нет	Нет	Нет	Число игроков в команде
Максимум участников	INTEGER	Да	Нет	Нет	Нет	Максимальное число участников

Продолжение таблицы А.2

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Число раундов (швейцарская система)	INTEGER	Нет	Нет	Нет	Нет	Количество раундов для швейцарской системы
Номер текущего раунда	INTEGER	Да	Нет	Нет	Нет	Номер текущего раунда
Признак удаления	BOOLEAN	Да	Нет	Нет	Нет	Признак логического удаления
Дата создания	TIMESTAMP	Да	Нет	Нет	Нет	Дата создания
Дата запуска	TIMESTAMP	Нет	Нет	Нет	Нет	Дата запуска турнира
Дата завершения	TIMESTAMP	Нет	Нет	Нет	Нет	Дата завершения
Дата отмены	TIMESTAMP	Нет	Нет	Нет	Нет	Дата отмены
Дата удаления	TIMESTAMP	Нет	Нет	Нет	Нет	Дата удаления

Таблица А.3 – Спецификация атрибутов отношения «Участники турнира»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор участия	UUID	Да	Нет	Да	Нет	Идентификатор записи об участии
Идентификатор турнира	UUID	Да	Нет	Нет	Да	Турнир
Идентификатор игрока	UUID	Да	Нет	Нет	Нет	Игрок
Никнейм игрока	VARCHAR(128)	Да	Нет	Нет	Нет	Никнейм игрока
Дата регистрации	TIMESTAMP	Да	Нет	Нет	Нет	Дата регистрации на турнир
Признак активного участия	BOOLEAN	Да	Нет	Нет	Нет	Активность участия
Дата выхода	TIMESTAMP	Нет	Нет	Нет	Нет	Дата выхода из турнира

Таблица А.4 – Спецификация атрибутов отношения «Команды»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор команды	UUID	Да	Нет	Да	Нет	Идентификатор команды
Идентификатор турнира	UUID	Да	Нет	Нет	Да	Турнир
Название	VARCHAR(128)	Да	Нет	Нет	Нет	Название команды
Идентификатор капитана	UUID	Да	Нет	Нет	Нет	Капитан команды
Средний рейтинг	FLOAT	Да	Нет	Нет	Нет	Средний рейтинг команды (денормализован)
Посев	INTEGER	Да	Нет	Нет	Нет	Посев команды

Таблица А.5 – Спецификация атрибутов отношения «Состав команд»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор записи состава	UUID	Да	Нет	Да	Нет	Идентификатор записи состава
Идентификатор команды	UUID	Да	Нет	Нет	Да	Команда

Продолжение таблицы А.5

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор игрока	UUID	Да	Нет	Нет	Нет	Игрок
Никнейм	VARCHAR(128)	Да	Нет	Нет	Нет	Никнейм игрока
Рейтинг	INTEGER	Да	Нет	Нет	Нет	Рейтинг игрока на момент формирования команды

Таблица А.6 – Спецификация атрибутов отношения «Раунды»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор раунда	UUID	Да	Нет	Да	Нет	Идентификатор раунда
Идентификатор турнира	UUID	Да	Нет	Нет	Да	Турнир
Номер раунда	INTEGER	Да	Нет	Нет	Нет	Номер раунда
Тип сетки	VARCHAR(64)	Да	Нет	Нет	Нет	Тип сетки: Main, Upper, Lower, GrandFinal, ThirdPlace
Статус	VARCHAR(64)	Да	Нет	Нет	Нет	Статус раунда
Дата создания	TIMESTAMP	Да	Нет	Нет	Нет	Дата создания
Дата завершения	TIMESTAMP	Нет	Нет	Нет	Нет	Дата завершения

Таблица А.7 – Спецификация атрибутов отношения «Матчи»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор матча	UUID	Да	Нет	Да	Нет	Идентификатор матча
Идентификатор турнира	UUID	Да	Нет	Нет	Нет	Турнир
Идентификатор раунда	UUID	Да	Нет	Нет	Да	Раунд
Номер матча	INTEGER	Да	Нет	Нет	Нет	Номер матча в раунде
Команда А	UUID	Нет	Нет	Нет	Да	Первая команда матча
Команда В	UUID	Нет	Нет	Нет	Да	Вторая команда матча
Команда-победитель	UUID	Нет	Нет	Нет	Нет	Команда-победитель
Проигравшая команда	UUID	Нет	Нет	Нет	Нет	Проигравшая команда
Счёт победителя	INTEGER	Нет	Нет	Нет	Нет	Счёт победителя по раундам
Счёт проигравшего	INTEGER	Нет	Нет	Нет	Нет	Счёт проигравшего по раундам
Карты победителя	INTEGER	Нет	Нет	Нет	Нет	Выигранные карты победителя
Карты проигравшего	INTEGER	Нет	Нет	Нет	Нет	Выигранные карты проигравшего

Продолжение таблицы А.7

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Техническое поражение	BOOLEAN	Да	Нет	Нет	Нет	Признак технического поражения
Статус	VARCHAR(64)	Да	Нет	Нет	Нет	Статус матча
Дата создания	TIMESTAMP	Да	Нет	Нет	Нет	Дата создания
Дата завершения	TIMESTAMP	Нет	Нет	Нет	Нет	Дата завершения

Таблица А.8 – Спецификация атрибутов отношения «Положение в швейцарской системе»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор записи	UUID	Да	Нет	Да	Нет	Идентификатор записи
Идентификатор турнира	UUID	Да	Нет	Нет	Да	Турнир
Идентификатор команды	UUID	Да	Нет	Нет	Нет	Команда
Победы	INTEGER	Да	Нет	Нет	Нет	Число побед
Поражения	INTEGER	Да	Нет	Нет	Нет	Число поражений
Очки	INTEGER	Да	Нет	Нет	Нет	Набранные очки

Таблица А.9 – Спецификация атрибутов отношения «Положение в формате Double Elimination»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор записи	UUID	Да	Нет	Да	Нет	Идентификатор записи
Идентификатор турнира	UUID	Да	Нет	Нет	Да	Турнир
Идентификатор команды	UUID	Да	Нет	Нет	Нет	Команда
Поражения	INTEGER	Да	Нет	Нет	Нет	Число поражений (для выбывания)

Таблица А.10 – Спецификация атрибутов отношения «Дисциплины»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Код дисциплины	VARCHAR(64)	Да	Нет	Да	Нет	Код дисциплины
Название	VARCHAR(128)	Да	Нет	Нет	Нет	Название дисциплины
Допустимые размеры команд	INTEGER[]	Да	Нет	Нет	Нет	Допустимые размеры команд
Признак активности	BOOLEAN	Да	Нет	Нет	Нет	Признак активности дисциплины

Таблица А.11 – Спецификация атрибутов отношения «Рейтинги игроков»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор записи	UUID	Да	Нет	Да	Нет	Идентификатор записи рейтинга

Продолжение таблицы А.11

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор игрока	UUID	Да	Нет	Нет	Нет	Игрок
Код дисциплины	VARCHAR(64)	Да	Нет	Нет	Нет	Игровая дисциплина
Рейтинг	INTEGER	Да	Нет	Нет	Нет	Текущий рейтинг ELO
Сыграно матчей	INTEGER	Да	Нет	Нет	Нет	Количество сыгранных матчей
Победы	INTEGER	Да	Нет	Нет	Нет	Количество побед
Поражения	INTEGER	Да	Нет	Нет	Нет	Количество поражений
Дата создания	TIMESTAMP	Да	Нет	Нет	Нет	Дата создания записи
Дата обновления	TIMESTAMP	Да	Нет	Нет	Нет	Дата последнего обновления

Таблица А.12 – Спецификация атрибутов отношения «История рейтинга»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор записи	UUID	Да	Нет	Да	Нет	Идентификатор записи истории
Идентификатор игрока	UUID	Да	Нет	Нет	Нет	Игрок
Код дисциплины	VARCHAR(64)	Да	Нет	Нет	Нет	Игровая дисциплина
Рейтинг до изменения	INTEGER	Да	Нет	Нет	Нет	Значение рейтинга до изменения
Рейтинг после изменения	INTEGER	Да	Нет	Нет	Нет	Значение рейтинга после изменения
Величина изменения	INTEGER	Да	Нет	Нет	Нет	Величина изменения рейтинга
Причина изменения	VARCHAR(256)	Да	Нет	Нет	Нет	Причина изменения
Идентификатор матча	UUID	Нет	Нет	Нет	Нет	Матч, вызвавший изменение
Идентификатор турнира	UUID	Нет	Нет	Нет	Нет	Турнир
Дата изменения	TIMESTAMP	Да	Нет	Нет	Нет	Дата изменения

Таблица А.13 – Спецификация атрибутов отношения «Уведомления»

Имя атрибута	Тип данных	Обязательный	Уникальный	PK	FK	Описание
Идентификатор уведомления	UUID	Да	Нет	Да	Нет	Идентификатор уведомления
Идентификатор получателя	UUID	Да	Нет	Нет	Нет	Получатель уведомления
Тип уведомления	VARCHAR(32)	Да	Нет	Нет	Нет	Тип уведомления
Заголовок	VARCHAR(256)	Да	Нет	Нет	Нет	Заголовок
Текст	VARCHAR(2048)	Да	Нет	Нет	Нет	Текст уведомления
Ссылка для перехода	VARCHAR(512)	Да	Нет	Нет	Нет	Ссылка для перехода
Дополнительные данные (JSON)	JSON	Да	Нет	Нет	Нет	Дополнительные данные в формате JSON

Продолжение таблицы А.13

Имя атрибута	Тип данных	Обязательный	Уникальный	РК	FK	Описание
Идентификатор исходного события	UUID	Да	Нет	Нет	Нет	Исходное интеграционное событие
Дата создания	TIMESTAMP	Да	Нет	Нет	Нет	Дата создания
Дата прочтения	TIMESTAMP	Нет	Нет	Нет	Нет	Дата прочтения

Приложение Б

(рекомендуемое)

Таблицы эндпоинтов микросервисов

Таблица Б.1 – Реализованные эндпоинты сервиса управления турнирами

Метод	Эндпоинт	Назначение	Контроллер	Уровень доступа
GET	/api/tournaments	Получение списка всех турниров	TournamentsController	Без авторизации
GET	/api/tournaments/{id}	Получение турнира по ID	TournamentsController	Без авторизации
GET	/api/tournaments/available	Получение турниров, открытых для регистрации	TournamentsController	Без авторизации
GET	/api/tournaments/active	Получение активных турниров	TournamentsController	Без авторизации
GET	/api/tournaments/completed	Получение завершённых турниров	TournamentsController	Без авторизации
GET	/api/tournaments/my	Получение турниров текущего игрока	TournamentsController	Игрок
POST	/api/tournaments/{tournamentId}/registration	Регистрация в турнире	TournamentsController	Игрок
DELETE	/api/tournaments/{tournamentId}/registration	Отмена регистрации в турнире	TournamentsController	Игрок
POST	/api/tournaments	Создание турнира	TournamentsController	Подтвержденный организатор
GET	/api/organizer/tournaments	Получение турниров организатора	OrganizerTournamentsController	Подтвержденный организатор
PUT	/api/tournaments/{id}	Обновление данных турнира	TournamentsController	Организатор / Администратор
POST	/api/tournaments/{id}/cancel	Отмена турнира	TournamentsController	Организатор / Администратор
POST	/api/tournaments/{tournamentId}/swiss/next-round	Создание следующего раунда Swiss	TournamentsController	Организатор / Администратор
POST	/api/tournaments/{tournamentId}/matches/{matchId}/complete	Внесение результата матча	TournamentsController	Организатор / Администратор

Продолжение таблицы Б.1

Метод	Эндпоинт	Назначение	Контроллер	Уровень доступа
GET	/api/tournaments/{tournamentId}/matches/{matchId}	Получение деталей матча	TournamentsController	Любой авторизованный пользователь
POST	/api/admin/tournaments	Создание турнира от имени организатора	AdminTournamentsController	Администратор
DELETE	/api/admin/tournaments/{id}	Удаление турнира	AdminTournamentsController	Администратор

Таблица Б.2 – Реализованные эндпоинты сервиса рейтинга

Метод	Эндпоинт	Назначение	Контроллер	Уровень доступа
GET	/api/ratings/players/{playerId}	Получение текущих рейтингов игрока по дисциплинам	RatingsController	Без авторизации
GET	/api/ratings/players/{playerId}/history	Получение истории изменения рейтинга игрока	RatingsController	Без авторизации

Таблица Б.3 – Реализованные эндпоинты сервиса уведомлений

Метод	Эндпоинт	Назначение	Контроллер	Уровень доступа
GET	/api/notifications	Получение списка уведомлений текущего пользователя с пагинацией	NotificationsController	Авторизованный пользователь
POST	/api/notifications/{id}/read	Отметка уведомления как прочитанного	NotificationsController	Авторизованный пользователь

Приложение В

(рекомендуемое)

Таблицы разделения классов по слоям в микросервисах

Таблица В.1 – Разделение классов по слоям в микросервисе управления турнирами

Слой	Назначение	Содержимое
Tournament.Api	Обработка HTTP-запросов	TournamentsController, OrganizerTournamentsController, AdminTournamentsController
Tournament.Application	Выполнение сценариев управления турнирами, генерации сетки и балансировки команд	TournamentService, TournamentLifecycleService, MatchResultService, UserProjectionService, PlayerRatingProjectionService, ITournamentService, ITournamentLifecycleService, IMatchResultService, IUserProjectionService, IPlayerRatingProjectionService, ITournamentRepository, IUserProjectionRepository, IPlayerRatingProjectionRepository, IOutboxWriter, ITournamentTransaction, IBracketGenerator, IBracketGeneratorFactory, BracketGeneratorFactory, BracketGeneratorBase, SingleEliminationBracketGenerator, DoubleEliminationBracketGenerator, SwissBracketGenerator, ISwissRoundService, SwissRoundService, ITeamBalancer, TeamBalancerBuilder, GreedyTeamBalancer, CpSatAverageEloTeamBalancer, IRandomProvider, RandomProvider, PlayerForBalancing, BalancedTeam, BalancedTeamMember, TeamBalancingOptions, TeamBalancingException, CreateTournamentRequest, AdminCreateTournamentRequest, UpdateTournamentRequest, CompleteMatchRequest, TournamentDetailsResponse, TournamentListItemResponse, MatchDetailsResponse, MatchResponse, RoundResponse, TeamResponse, TeamMemberResponse, TournamentParticipantResponse, TournamentSummaryDto, TournamentResponseMapper

Продолжение таблицы В.1

Слой	Назначение	Содержимое
Tournament.Domain	Описание доменных сущностей турнира и правил изменения их состояния	Tournament, TournamentParticipant, Team, TeamMember, Round, Match, SwissStanding, DoubleEliminationStanding, Discipline, UserProjection, PlayerRatingProjection, BlockedUserProjection
Tournament.Infrastructure	Работа с БД, обработка входящих интеграционных событий и публикация исходящих событий	TournamentRepository, UserProjectionRepository, PlayerRatingProjectionRepository, TournamentDbContext, TournamentDbContextFactory, TournamentEfTransaction, OutboxWriter, TournamentInboxMessageStore, TournamentOutboxMessageStore, TournamentConfiguration, TournamentParticipantConfiguration, TeamConfiguration, TeamMemberConfiguration, RoundConfiguration, MatchConfiguration, DisciplineConfiguration, PlayerRatingProjectionConfiguration, InboxMessageConfiguration, OutboxMessageConfiguration, TournamentUserCreatedConsumer, TournamentRatingUpdatedConsumer, TournamentUserBlockedConsumer, TournamentUserRoleChangedConsumer, TournamentUserContactHandleChangedConsumer

Таблица В.2 – Разделение классов по слоям в микросервисе рейтинга

Слой	Назначение	Содержимое
Rating.Api	Обработка HTTP-запросов и настройка API-сервиса	RatingsController
Rating.Application	Выполнение прикладных сценариев получения рейтингов, обработки событий и расчёта ELO	RatingService, IRatingService, EloCalculator, IEloCalculator, IRatingRepository, IOutboxWriter, PlayerRatingResponse, RatingHistoryResponse
Rating.Domain	Описание доменных сущностей рейтинга и правил изменения состояния	PlayerRating, RatingHistory

Продолжение таблицы В.2

Слой	Назначение	Содержимое
Rating.Infrastructure	Работа с базой данных, EF Core, Inbox/Outbox и обработкой интеграционных событий	RatingInfrastructureServiceCollectionExtensions, RatingDbContext, RatingDbContextFactory, RatingRepository, RatingInboxMessageStore, RatingOutboxMessageStore, OutboxWriter, PlayerRatingConfiguration, RatingHistoryConfiguration, InboxMessageConfiguration, OutboxMessageConfiguration, RatingUserCreatedConsumer, RatingUserBlockedConsumer, RatingUserRoleChangedConsumer, RatingMatchCompletedConsumer

Таблица В.3 – Разделение классов по слоям в микросервисе уведомлений

Слой	Назначение	Классы, интерфейсы и DTO
Notification.Api	Обработка HTTP-запросов и настройка API-сервиса	NotificationsController
Notification.Application	Выполнение сценариев получения уведомлений, отметки прочтения и формирования уведомлений по входящим событиям	NotificationService, INotificationService, RoundCreatedFanout, IRoundCreatedFanout, INotificationRepository, NotificationErrors, NotificationResponse, NotificationListResponse
Notification.Domain	Описание доменной сущности уведомления и типа уведомления	Notification, NotificationType
Notification.Infrastructure	Работа с базой данных, EF Core, Inbox и обработкой интеграционных событий	NotificationInfrastructureServiceCollectionExtensions, NotificationDbContext, NotificationDbContextFactory, NotificationRepository, NotificationInboxMessageStore, NotificationConfiguration, InboxMessageConfiguration, NotificationRoundCreatedConsumer

Приложение Г

(обязательное)

Код программы

Листинг Г.1 – Метод BuildTeams

```

public IReadOnlyList<BalancedTeam> BuildTeams(
    IReadOnlyCollection<PlayerForBalancing> players,
    int teamSize)
{
    TeamBalancerBuilder.Validate(players, teamSize);
    var fallbackTeams = fallbackBalancer.BuildTeams(players,
teamSize);

    if (teamSize == 1)
    {
        return fallbackTeams;
    }

    var playerArray = players
        .OrderByDescending(player => player.Elo)
        .ThenBy(player => player.Nickname,
StringComparer.Ordinal)
        .ThenBy(player => player.PlayerId)
        .ToArray();
    var teamsCount = playerArray.Length / teamSize;
    var totalElo = playerArray.Sum(player =>
(long)player.Elo);
    var minPossibleTeamTotal = playerArray
        .OrderBy(player => player.Elo)
        .Take(teamSize)
        .Sum(player => (long)player.Elo);
    var maxPossibleTeamTotal = playerArray
        .OrderByDescending(player => player.Elo)
        .Take(teamSize)
        .Sum(player => (long)player.Elo);

    try
    {
        var model = new CpModel();
        var assignments = CreateAssignmentVariables(model,
playerArray.Length, teamsCount);

        AddPlayerConstraints(model, assignments,
playerArray.Length, teamsCount);
        AddTeamSizeConstraints(model, assignments,
playerArray.Length, teamsCount, teamSize);

        var teamTotals = CreateTeamTotalVariables(
            model,
            assignments,
            playerArray,
            teamsCount,
            minPossibleTeamTotal,
            maxPossibleTeamTotal);
    }
}

```

```

        var maxTeamTotal =
model.NewIntVar(minPossibleTeamTotal, maxPossibleTeamTotal,
"max_team_total");
        var minTeamTotal =
model.NewIntVar(minPossibleTeamTotal, maxPossibleTeamTotal,
"min_team_total");
        var teamTotalGap = model.NewIntVar(0, totalElo,
"team_total_gap");

        model.AddMaxEquality(maxTeamTotal, teamTotals);
        model.AddMinEquality(minTeamTotal, teamTotals);
        model.Add(teamTotalGap == maxTeamTotal -
minTeamTotal);
        model.Minimize(teamTotalGap);

        var solver = new CpSolver();
        solver.StringParameters =
BuildSolverParameters(options.Value);

        var status = solver.Solve(model);

        if (status is not (CpSolverStatus.Optimal or
CpSolverStatus.Feasible))
        {
            return fallbackTeams;
        }

        var solvedTeams = ReadSolvedTeams(solver, assignments,
playerArray, teamsCount);
        TeamBalancerBuilder.EnsureValidTeams(solvedTeams,
teamSize);
        if (CalculateTeamTotalGap(solvedTeams) >
CalculateTeamTotalGap(fallbackTeams))
        {
            return fallbackTeams;
        }

        return
TeamBalancerBuilder.CreateBalancedTeams(solvedTeams, teamSize,
randomProvider);
    }
    catch (Exception)
    {
        return fallbackTeams;
    }
}

```

Листинг Г.2 – Метод CompleteMatchAsync

```

public async Task<Result<TournamentDetailsResponse>>
CompleteMatchAsync(
    Guid tournamentId,
    Guid matchId,

```

```

    CompleteMatchRequest request,
    CurrentTournamentUser currentUser,
    CancellationToken cancellationToken = default)
{
    // Загружаем турнир и проверяем его наличие в системе.
    var tournament = await tournaments.GetByIdAsync(tournamentId,
cancellationToken);
    if (tournament is null)
    {
        return
Result<TournamentDetailsResponse>.Failure(TournamentErrors.Tournam
entNotFound);
    }

    // Проверяем, имеет ли текущий пользователь право завершать
матч.
    if (!CanComplete(tournament, currentUser))
    {
        return
Result<TournamentDetailsResponse>.Failure(TournamentErrors.AccessD
enied);
    }

    // Находим матч среди всех раундов выбранного турнира.
    var match = tournament.Rounds
        .SelectMany(round => round.Matches)
        .SingleOrDefault(match => match.Id == matchId);
    if (match is null || match.TournamentId != tournamentId)
    {
        return
Result<TournamentDetailsResponse>.Failure(TournamentErrors.Partici
pantNotFound);
    }

    // Проверяем, что матч еще не был завершен ранее.
    if (match.Status == MatchStatus.Completed)
    {
        return
Result<TournamentDetailsResponse>.Failure(TournamentErrors.MatchAl
readyCompleted);
    }

    // Завершение матча допускается только для турнира,
находящегося в процессе проведения.
    if (tournament.Status != TournamentStatus.InProgress)
    {
        return
Result<TournamentDetailsResponse>.Failure(TournamentErrors.Tournam
entNotInProgress);
    }

    // Проверяем, что у матча определены обе команды-участницы.

```

```

        if (match.TeamAId is null || match.TeamBId is null)
        {
            return
Result<TournamentDetailsResponse>.Failure(TournamentErrors.Tournam
entRegistrationClosed);
        }

        // Проверяем, что указанная команда-победитель действительно
участвует в матче.
        if (request.WinnerTeamId != match.TeamAId &&
request.WinnerTeamId != match.TeamBId)
        {
            return
Result<TournamentDetailsResponse>.Failure(TournamentErrors.Invalid
WinnerTeam);
        }

        // Для обычного завершения матча счет по раундам и картам
является обязательным.
        if (!request.IsTechnicalDefeat
            && (request.WinnerScore is null || request.LoserScore is
null
                || request.WinnerMaps is null || request.LoserMaps is
null))
        {
            return
Result<TournamentDetailsResponse>.Failure(TournamentErrors.MatchSc
oreRequired);
        }

        // Проверяем корректность счета по раундам.
        if (request.WinnerScore is not null
            && request.LoserScore is not null
            && request.WinnerScore <= request.LoserScore)
        {
            return
Result<TournamentDetailsResponse>.Failure(TournamentErrors.Invalid
MatchScore);
        }

        // Проверяем корректность счета по выигранным картам.
        if (request.WinnerMaps is not null
            && request.LoserMaps is not null
            && request.WinnerMaps <= request.LoserMaps)
        {
            return
Result<TournamentDetailsResponse>.Failure(TournamentErrors.Invalid
MatchScore);
        }

        // Фиксируем результат матча и переводим его в заверенное
состояние.

```

```

match.Complete(
    request.WinnerTeamId,
    request.WinnerScore,
    request.LoserScore,
    request.WinnerMaps,
    request.LoserMaps,
    request.IsTechnicalDefeat,
    DateTime.UtcNow);

// Публикуем событие завершения матча для дальнейшей обработки
другими компонентами.
outboxWriter.Add(ToMatchCompletedEvent(tournament, match));

// Передаем результат генератору турнирной сетки и сохраняем
изменения.
var generator =
    bracketGeneratorFactory.GetGenerator(tournament.Format);
await generator.HandleMatchCompletedAsync(tournament, match,
    cancellationTokens);
await tournaments.SaveChangesAsync(cancellationTokens);

// Возвращаем обновленную информацию о турнире.
return
    Result<TournamentDetailsResponse>.Success(TournamentResponseMapper
        .ToDetailsResponse(tournament));
}

```

Листинг Г.3 – Основной контроллер микросервиса управления пользователями

```

using System.Security.Claims;
using Auth.Application.Auth;
using Auth.Application.Auth.Dto;
using Auth.Application.Auth.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using TournamentPlatform.Shared.Common;
using TournamentPlatform.Shared.Security;

namespace Auth.Api.Controllers;

[ApiController]
[Route("api/auth")]
public sealed class AuthController(IAuthService authService) :
    ControllerBase
{
    [HttpPost("register/player")]
    [ProducesResponseType(typeof(AuthResponse),
        StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status409Conflict)]
    public async Task<IActionResult> RegisterPlayer(
        RegisterPlayerRequest request,
        CancellationToken cancellationToken)
    {

```

```

    {
        var result = await
authService.RegisterPlayerAsync(request, cancellationTokens);
        return ToActionResult(result);
    }

    [HttpPost("register/organizer")]
    [ProducesResponseType(typeof(AuthResponse),
StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status409Conflict)]
    public async Task<IActionResult> RegisterOrganizer(
        RegisterOrganizerRequest request,
        CancellationTokens cancellationTokens)
    {
        var result = await
authService.RegisterOrganizerAsync(request, cancellationTokens);
        return ToActionResult(result);
    }

    [HttpPost("login")]
    [ProducesResponseType(typeof(AuthResponse),
StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status401Unauthorized)]
    public async Task<IActionResult> Login(
        LoginRequest request,
        CancellationTokens cancellationTokens)
    {
        var result = await authService.LoginAsync(request,
cancellationTokens);
        return ToActionResult(result);
    }

    [Authorize]
    [HttpGet("me")]
    [ProducesResponseType(typeof(CurrentUserResponse),
StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status401Unauthorized)]
    public async Task<IActionResult> Me(CancellationTokens
cancellationTokens)
    {
        if (!User.TryGetCurrentUser(out var currentUser))
        {
            return Unauthorized();
        }

        var result = await
authService.GetCurrentUserAsync(currentUser.UserId,
cancellationTokens);
        return ToActionResult(result);
    }

    [Authorize]

```

```

[HttpPost("change-password")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
public async Task<IActionResult> ChangePassword(
    ChangePasswordRequest request,
    CancellationToken cancellationToken)
{
    if (!User.TryGetCurrentUser(out var currentUser))
    {
        return Unauthorized();
    }

    var result = await
authService.ChangePasswordAsync(currentUser.UserId, request,
cancellationToken);
    return result.IsSuccess ? NoContent() :
ToActionResult(result.Error);
}

[Authorize]
[HttpPut("contact-handle")]
[ProducesResponseType(typeof(CurrentUserResponse),
StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
public async Task<IActionResult> UpdateContactHandle(
    UpdateContactHandleRequest request,
    CancellationToken cancellationToken)
{
    if (!User.TryGetCurrentUser(out var currentUser))
    {
        return Unauthorized();
    }

    var result = await
authService.UpdateContactHandleAsync(currentUser.UserId, request,
cancellationToken);
    return ToActionResult(result);
}

private IActionResult ToActionResult<T>(Result<T> result)
{
    if (result.IsSuccess)
    {
        return Ok(result.Value);
    }

    if (result.Error == AuthErrors.DuplicateEmail ||
result.Error == AuthErrors.DuplicateNickname)
    {
        return Conflict(CreateProblemDetails(result.Error,

```

```

        StatusCodes.Status409Conflict));
    }

    if (result.Error == AuthErrors.InvalidCredentials ||
        result.Error == AuthErrors.AccessDenied)
    {
        return Unauthorized(CreateProblemDetails(result.Error,
            StatusCodes.Status401Unauthorized));
    }

    if (result.Error == AuthErrors.UserNotFound)
    {
        return NotFound(CreateProblemDetails(result.Error,
            StatusCodes.Status404NotFound));
    }

    return BadRequest(CreateProblemDetails(result.Error,
        StatusCodes.Status400BadRequest));
}

private IActionResult ToErrorActionResult(Error error)
{
    if (error == AuthErrors.InvalidCredentials || error ==
        AuthErrors.AccessDenied)
    {
        return Unauthorized(CreateProblemDetails(error,
            StatusCodes.Status401Unauthorized));
    }

    if (error == AuthErrors.UserNotFound)
    {
        return NotFound(CreateProblemDetails(error,
            StatusCodes.Status404NotFound));
    }

    return BadRequest(CreateProblemDetails(error,
        StatusCodes.Status400BadRequest));
}

private static ProblemDetails CreateProblemDetails(Error
error, int statusCode)
{
    return new ProblemDetails
    {
        Status = statusCode,
        Title = error.Message,
        Type = error.Code
    };
}
}

```

Листинг Г.4 – Основной контроллер микросервиса управления турнирами

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Tournament.Application.Brackets;
using Tournament.Application.Matches;
using Tournament.Application.Tournaments;
using Tournament.Application.Tournaments.Dto;
using Tournament.Application.Tournaments.Services;
using TournamentPlatform.Shared.Common;
using TournamentPlatform.Shared.Security;

namespace Tournament.Api.Controllers;

[ApiController]
[Route("api/tournaments")]
public sealed class TournamentsController(
    ITournamentService tournamentService,
    ISwissRoundService swissRoundService,
    IMatchResultService matchResultService) : ControllerBase
{
    [Authorize(Policy =
AuthorizationPolicies.RequireActiveOrganizer)]
    [HttpPost]
    [ProducesResponseType(typeof(TournamentDetailsResponse),
StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status403Forbidden)]
    [ProducesResponseType(StatusCodes.Status409Conflict)]
    public async Task<IActionResult> Create(
        CreateTournamentRequest request,
        CancellationToken cancellationToken)
    {
        if (!TryGetCurrentUser(out var currentUser))
        {
            return Unauthorized();
        }

        var result = await tournamentService.CreateAsync(request,
currentUser, cancellationToken);
        return ToActionResult(result);
    }

    [HttpGet]

    [ProducesResponseType(typeof(IReadOnlyCollection<TournamentListIte
mResponse>), StatusCodes.Status200OK)]
    public async Task<IActionResult> GetAll(CancellationToken
cancellationToken)
    {
        var result = await
tournamentService.GetAllAsync(cancellationToken);
        return Ok(result.Value);
    }
}

```

```

    }

    [HttpGet("{id:guid}")]
    [ProducesResponseType(typeof(TournamentDetailsResponse),
    StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public async Task<IActionResult> GetById(Guid id,
    Cancellation_token cancellationToken)
    {
        var result = await tournamentService.GetByIdAsync(id,
    cancellationToken);
        return ToActionResult(result);
    }

    [HttpGet("available")]

    [ProducesResponseType(typeof(IReadOnlyCollection<TournamentListIte
    mResponse>), StatusCodes.Status200OK)]
    public async Task<IActionResult>
    GetAvailable(Cancellation_token cancellationToken)
    {
        var result = await
    tournamentService.GetAvailableAsync(cancellationToken);
        return Ok(result.Value);
    }

    [HttpGet("active")]

    [ProducesResponseType(typeof(IReadOnlyCollection<TournamentListIte
    mResponse>), StatusCodes.Status200OK)]
    public async Task<IActionResult> GetActive(Cancellation_token
    cancellationToken)
    {
        var result = await
    tournamentService.GetActiveAsync(cancellationToken);
        return Ok(result.Value);
    }

    [HttpGet("completed")]

    [ProducesResponseType(typeof(IReadOnlyCollection<TournamentListIte
    mResponse>), StatusCodes.Status200OK)]
    public async Task<IActionResult>
    GetCompleted(Cancellation_token cancellationToken)
    {
        var result = await
    tournamentService.GetCompletedAsync(cancellationToken);
        return Ok(result.Value);
    }

    [Authorize(Policy = AuthorizationPolicies.RequirePlayer)]
    [HttpGet("my")]

```

```

[ProducesResponseType(typeof(ICollection<TournamentListItemResponse>), StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
public async Task<IActionResult> GetMy(CancellationTok
cancellationToken)
{
    if (!TryGetCurrentUser(out var currentUser))
    {
        return Unauthorized();
    }

    var result = await
tournamentService.GetMyAsync(currentUser, cancellationToken);
    return ToActionResult(result);
}

[Authorize(Policy = AuthorizationPolicies.RequirePlayer)]
[HttpPost("{tournamentId:guid}/registration")]
[ProducesResponseType(typeof(TournamentDetailsResponse),
StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status409Conflict)]
public async Task<IActionResult> Register(Guid tournamentId,
CancellationToken cancellationToken)
{
    if (!TryGetCurrentUser(out var currentUser))
    {
        return Unauthorized();
    }

    var result = await
tournamentService.RegisterPlayerAsync(tournamentId, currentUser,
cancellationToken);
    return ToActionResult(result);
}

[Authorize(Policy = AuthorizationPolicies.RequirePlayer)]
[HttpDelete("{tournamentId:guid}/registration")]
[ProducesResponseType(typeof(TournamentDetailsResponse),
StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status409Conflict)]
public async Task<IActionResult> Leave(Guid tournamentId,
CancellationToken cancellationToken)
{
    if (!TryGetCurrentUser(out var currentUser))
    {

```

```

        return Unauthorized();
    }

    var result = await
tournamentService.LeaveAsync(tournamentId, currentUser,
cancellationTokens);
    return ToActionResult(result);
}

[Authorize(Policy =
AuthorizationPolicies.RequireOrganizerOrAdmin)]
[HttpPut("{id:guid}")]
[ProducesResponseType(typeof(TournamentDetailsResponse),
StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status409Conflict)]
public async Task<IActionResult> Update(
    Guid id,
    UpdateTournamentRequest request,
    CancellationToken cancellationToken)
{
    if (!TryGetCurrentUser(out var currentUser))
    {
        return Unauthorized();
    }

    var result = await tournamentService.UpdateAsync(id,
request, currentUser, cancellationToken);
    return ToActionResult(result);
}

[Authorize(Policy =
AuthorizationPolicies.RequireOrganizerOrAdmin)]
[HttpPost("{id:guid}/cancel")]
[ProducesResponseType(typeof(TournamentDetailsResponse),
StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status409Conflict)]
public async Task<IActionResult> Cancel(Guid id,
CancellationToken cancellationToken)
{
    if (!TryGetCurrentUser(out var currentUser))
    {
        return Unauthorized();
    }

    var result = await tournamentService.CancelAsync(id,
currentUser, cancellationToken);
    return ToActionResult(result);
}

```

```

    }

    [Authorize(Policy =
AuthorizationPolicies.RequireOrganizerOrAdmin)]
    [HttpPost("{tournamentId:guid}/swiss/next-round")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status403Forbidden)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public async Task<IActionResult> CreateNextSwissRound(Guid
tournamentId, CancellationToken cancellationToken)
    {
        if (!TryGetCurrentUser(out var currentUser))
        {
            return Unauthorized();
        }

        var result = await
swissRoundService.CreateNextRoundAsync(tournamentId, currentUser,
cancellationToken);
        if (result.IsSuccess)
        {
            return NoContent();
        }

        if (result.Error == TournamentErrors.TournamentNotFound)
        {
            return NotFound(CreateProblemDetails(result.Error,
StatusCodes.Status404NotFound));
        }

        if (result.Error == TournamentErrors.AccessDenied)
        {
            return Forbid();
        }

        return BadRequest(CreateProblemDetails(result.Error,
StatusCodes.Status400BadRequest));
    }

    [Authorize(Policy =
AuthorizationPolicies.RequireOrganizerOrAdmin)]

    [HttpPost("{tournamentId:guid}/matches/{matchId:guid}/complete")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> CompleteMatch(
        Guid tournamentId,
        Guid matchId,
        CompleteMatchRequest request,
        CancellationToken cancellationToken)
    {
        if (!TryGetCurrentUser(out var currentUser))

```

```

    {
        return Unauthorized();
    }

    var result = await matchResultService.CompleteMatchAsync(
        tournamentId,
        matchId,
        request,
        currentUser,
        cancellationToken);

    return ToActionResult(result);
}

[Authorize]
[HttpGet("{tournamentId:guid}/matches/{matchId:guid}")]
[ProducesResponseType(typeof(MatchDetailsResponse),
StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetMatchDetails(
    Guid tournamentId,
    Guid matchId,
    CancellationToken cancellationToken)
{
    if (!TryGetCurrentUser(out var currentUser))
    {
        return Unauthorized();
    }

    var result = await tournamentService.GetMatchDetailsAsync(
        tournamentId,
        matchId,
        currentUser,
        cancellationToken);

    return ToActionResult(result);
}

private IActionResult ToActionResult<T>(Result<T> result)
{
    if (result.IsSuccess)
    {
        return Ok(result.Value);
    }

    if (result.Error == TournamentErrors.TournamentNotFound
        || result.Error ==
TournamentErrors.ParticipantNotFound)
    {
        return NotFound(CreateProblemDetails(result.Error,
StatusCodes.Status404NotFound));
    }
}

```

```

        if (result.Error == TournamentErrors.AccessDenied
            || result.Error == TournamentErrors.PlayerAccessDenied
            || result.Error == TournamentErrors.AdminAccessDenied)
        {
            return Forbid();
        }

        if (result.Error == TournamentErrors.DuplicateTitle
            || result.Error ==
TournamentErrors.DuplicateRegistration
            || result.Error == TournamentErrors.TournamentFull
            || result.Error ==
TournamentErrors.RegistrationConflict
            || result.Error ==
TournamentErrors.CannotCancelCompleted
            || result.Error ==
TournamentErrors.MatchAlreadyCompleted)
        {
            return Conflict(CreateProblemDetails(result.Error,
StatusCodes.Status409Conflict));
        }

        return BadRequest(CreateProblemDetails(result.Error,
StatusCodes.Status400BadRequest));
    }

    private bool TryGetCurrentUser(out CurrentTournamentUser
currentUser)
    {
        currentUser = default!;
        if (!User.TryGetCurrentUser(out var user))
        {
            return false;
        }

        currentUser = new CurrentTournamentUser(
            user.UserId,
            user.Role,
            user.AccountStatus ?? string.Empty,
            user.Nickname);

        return true;
    }

    private static ProblemDetails CreateProblemDetails(Error
error, int statusCode)
    {
        return new ProblemDetails
        {
            Status = statusCode,
            Title = error.Message,

```

```

        Type = error.Code
    };
}
}

```

Листинг Г.5 – Контроллер микросервиса рейтинга и статистики

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
using Rating.Application.Ratings.Dto;
using Rating.Application.Ratings.Services;

namespace Rating.Api.Controllers;

[ApiController]
[AllowAnonymous]
[Route("api/ratings")]
public sealed class RatingsController(IRatingService
ratingService) : ControllerBase
{
    [HttpGet("players/{playerId:guid}")]

    [ProducesResponseType(typeof(ICollection<PlayerRatingRespo
nse>), StatusCodes.Status200OK)]
    public async Task<IActionResult> GetPlayerRatings(Guid
playerId, CancellationToken cancellationToken)
    {
        var result = await
ratingService.GetPlayerRatingsAsync(playerId, cancellationToken);
        return Ok(result.Value);
    }

    [HttpGet("players/{playerId:guid}/history")]

    [ProducesResponseType(typeof(ICollection<RatingHistoryResp
onse>), StatusCodes.Status200OK)]
    public async Task<IActionResult> GetPlayerHistory(Guid
playerId, CancellationToken cancellationToken)
    {
        var result = await
ratingService.GetPlayerHistoryAsync(playerId, cancellationToken);
        return Ok(result.Value);
    }
}

```

Листинг Г.6 – Контроллер микросервиса уведомления

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Notification.Application.Notifications;
using Notification.Application.Notifications.Dto;
using Notification.Application.Notifications.Services;
using TournamentPlatform.Shared.Common;

```

```

using TournamentPlatform.Shared.Security;

namespace Notification.Api.Controllers;

[ApiController]
[Authorize]
[Route("api/notifications")]
public sealed class NotificationsController(INotificationService
notifications) : ControllerBase
{
    [HttpGet]
    [ProducesResponseType(typeof(NotificationListResponse),
StatusCodes.Status200OK)]
    public async Task<IActionResult> List(
        [FromQuery] int pageNumber = 1,
        [FromQuery] int pageSize = 20,
        CancellationToken cancellationToken = default)
    {
        if (!User.TryGetCurrentUser(out var currentUser))
        {
            return Unauthorized();
        }

        var list = await
notifications.ListAsync(currentUser.UserId, pageNumber, pageSize,
cancellationToken);
        return Ok(list);
    }

    [HttpPost("{id:guid}/read")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    [ProducesResponseType(StatusCodes.Status401Unauthorized)]
    [ProducesResponseType(StatusCodes.Status403Forbidden)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public async Task<IActionResult> MarkRead(Guid id,
CancellationToken cancellationToken)
    {
        if (!User.TryGetCurrentUser(out var currentUser))
        {
            return Unauthorized();
        }

        var result = await notifications.MarkReadAsync(id,
currentUser.UserId, cancellationToken);
        if (result.IsSuccess)
        {
            return NoContent();
        }

        return ToErrorActionResult(result.Error);
    }
}

```

```
private IActionResult ToErrorActionResult(Error error)
{
    if (error == NotificationErrors.NotFound)
    {
        return NotFound(CreateProblemDetails(error,
        StatusCodes.Status404NotFound));
    }

    if (error == NotificationErrors.AccessDenied)
    {
        return StatusCode(StatusCodes.Status403Forbidden,
        CreateProblemDetails(error, StatusCodes.Status403Forbidden));
    }

    return BadRequest(CreateProblemDetails(error,
    StatusCodes.Status400BadRequest));
}

private static ProblemDetails CreateProblemDetails(Error
error, int statusCode)
{
    return new ProblemDetails
    {
        Status = statusCode,
        Title = error.Message,
        Type = error.Code
    };
}
}
```

Приложение Д

(рекомендуемое)

Спецификация физических моделей баз данных

Таблица Д.1 – Атрибуты таблицы Users

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Уникальный идентификатор пользователя
Email	character varying(320)	NOT NULL	Адрес электронной почты
NormalizedEmail	character varying(320)	NOT NULL, UNIQUE	Нормализованный e-mail (уникальность и поиск)
PasswordHash	character varying(1024)	NOT NULL	Хэш пароля
Role	character varying(32)	NOT NULL	Роль: Player, Organizer, Admin
Status	character varying(32)	NOT NULL	Статус: PendingApproval, Active, Rejected, Blocked
Nickname	character varying(64)	NULL	Никнейм игрока
NormalizedNickname	character varying(64)	NULL, UNIQUE	Нормализованный никнейм
OrganizerName	character varying(128)	NULL	Название организатора
NormalizedOrganizerName	character varying(128)	NULL	Нормализованное название организатора
ContactHandle	character varying(64)	NULL	Контактный идентификатор для связи
CreatedByAdminId	uuid	NULL	Идентификатор администратора, создавшего запись
CreatedAtUtc	timestamp with time zone	NOT NULL	Дата и время создания (UTC)
ApprovedAtUtc	timestamp with time zone	NULL	Дата одобрения заявки организатора
RejectedAtUtc	timestamp with time zone	NULL	Дата отклонения заявки
BlockedAtUtc	timestamp with time zone	NULL	Дата блокировки учётной записи
RowVersion	bytea	NOT NULL	Версия строки (оптимистическая блокировка)

Таблица Д.2 – Атрибуты таблицы Tournaments

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор турнира
Title	character varying(200)	NOT NULL	Название турнира
NormalizedTitle	character varying(200)	NOT NULL, UNIQUE	Нормализованное название (уникальность)
Description	character varying(4000)	NULL	Описание турнира
DisciplineCode	character varying(64)	NOT NULL	Код игровой дисциплины
Format	character varying(64)	NOT NULL	Формат: SingleElimination, DoubleElimination, Swiss
Status	character varying(64)	NOT NULL	Статус турнира
OrganizerId	uuid	NOT NULL	Идентификатор организатора
TeamSize	integer	NOT NULL	Размер команды
MaxPlayers	integer	NOT NULL	Максимальное число участников
SwissRounds	integer	NULL	Число раундов для швейцарской системы
CurrentRoundNumber	integer	NOT NULL	Номер текущего раунда
IsDeleted	boolean	NOT NULL	Признак удаления (soft-delete)
CreatedAtUtc	timestamp with time zone	NOT NULL	Дата создания
StartedAtUtc	timestamp with time zone	NULL	Дата запуска турнира

Продолжение таблицы Д.2

Атрибут	Тип данных	Ограничения	Описание
CompletedAtUtc	timestamp with time zone	NULL	Дата завершения
CancelledAtUtc	timestamp with time zone	NULL	Дата отмены
DeletedAtUtc	timestamp with time zone	NULL	Дата удаления
RowVersion	bytea	NOT NULL	Версия строки (оптимистическая блокировка)

Таблица Д.3 – Атрибуты таблицы TournamentParticipants

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор записи об участии
TournamentId	uuid	FK, NOT NULL	Турнир
PlayerId	uuid	NOT NULL	Идентификатор игрока
PlayerNickname	character varying(128)	NOT NULL	Никнейм игрока
IsActive	boolean	NOT NULL	Признак активного участия
RegisteredAtUtc	timestamp with time zone	NOT NULL	Дата регистрации
LeftAtUtc	timestamp with time zone	NULL	Дата выхода из турнира

Таблица Д.4 – Атрибуты таблицы Teams

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор команды
TournamentId	uuid	FK, NOT NULL	Турнир
Name	character varying(128)	NOT NULL	Название команды
CaptainPlayerId	uuid	NOT NULL	Идентификатор капитана
AverageElo	double precision	NOT NULL	Средний рейтинг команды
Seed	integer	NOT NULL	Посев команды

Таблица Д.5 – Атрибуты таблицы TeamMembers

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор записи состава
TeamId	uuid	FK, NOT NULL	Команда
PlayerId	uuid	NOT NULL	Идентификатор игрока
Nickname	character varying(128)	NOT NULL	Никнейм игрока
Elo	integer	NOT NULL	Рейтинг игрока на момент формирования команды

Таблица Д.6 – Атрибуты таблицы Rounds

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор раунда
TournamentId	uuid	FK, NOT NULL	Турнир
Number	integer	NOT NULL	Номер раунда

Продолжение таблицы Д.6

Атрибут	Тип данных	Ограничения	Описание
BracketType	character varying(64)	NOT NULL	Тип сетки: Main, Upper, Lower, GrandFinal, ThirdPlace
Status	character varying(64)	NOT NULL	Статус раунда
CreatedAtUtc	timestamp with time zone	NOT NULL	Дата создания
CompletedAtUtc	timestamp with time zone	NULL	Дата завершения

Таблица Д.7 – Атрибуты таблицы Matches

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор матча
RoundId	uuid	FK, NOT NULL	Раунд
TournamentId	uuid	NOT NULL	Турнир
MatchNumber	integer	NOT NULL	Номер матча в раунде
TeamAId	uuid	FK, NULL	Команда А
TeamBId	uuid	FK, NULL	Команда В
WinnerTeamId	uuid	NULL	Команда-победитель
LoserTeamId	uuid	NULL	Проигравшая команда
WinnerScore	integer	NULL	Счёт победителя (по раундам)
LoserScore	integer	NULL	Счёт проигравшего (по раундам)
WinnerMaps	integer	NULL	Выигранные карты победителя
LoserMaps	integer	NULL	Выигранные карты проигравшего
IsTechnicalDefeat	boolean	NOT NULL	Признак технического поражения
Status	character varying(64)	NOT NULL	Статус матча
CreatedAtUtc	timestamp with time zone	NOT NULL	Дата создания
CompletedAtUtc	timestamp with time zone	NULL	Дата завершения

Таблица Д.8 – Атрибуты таблицы SwissStandings

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор записи
TournamentId	uuid	FK, NOT NULL	Турнир
TeamId	uuid	NOT NULL	Команда
Points	integer	NOT NULL	Набранные очки
Wins	integer	NOT NULL	Число побед
Losses	integer	NOT NULL	Число поражений

Таблица Д.9 – Атрибуты таблицы DoubleEliminationStandings

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор записи
TournamentId	uuid	FK, NOT NULL	Турнир
TeamId	uuid	NOT NULL	Команда
Losses	integer	NOT NULL	Число поражений (для выбывания)

Таблица Д.10 – Атрибуты таблицы Disciplines

Атрибут	Тип данных	Ограничения	Описание
Code	character varying(64)	PK, NOT NULL	Код дисциплины
Name	character varying(128)	NOT NULL	Название дисциплины
AllowedTeamSizes	integer[]	NOT NULL	Допустимые размеры команд
IsActive	boolean	NOT NULL	Признак активности дисциплины

Таблица Д.11 – Атрибуты таблицы PlayerRatingProjections

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор проекции
PlayerId	uuid	NOT NULL	Идентификатор игрока
DisciplineCode	character varying(64)	NOT NULL	Код дисциплины
Elo	integer	NOT NULL	Текущий рейтинг игрока (проекция)
UpdatedAtUtc	timestamp with time zone	NOT NULL	Дата обновления проекции

Таблица Д.12 – Атрибуты таблицы user_projections

Атрибут	Тип данных	Ограничения	Описание
UserId	uuid	PK, NOT NULL	Идентификатор пользователя
Role	character varying(32)	NOT NULL	Роль пользователя
OrganizerName	character varying(128)	NULL	Название организатора
ContactHandle	character varying(64)	NULL	Контактный идентификатор
CreatedAtUtc	timestamp with time zone	NOT NULL	Дата создания проекции
BlockedAtUtc	timestamp with time zone	NULL	Дата блокировки

Таблица Д.13 – Атрибуты таблицы BlockedUserProjections

Атрибут	Тип данных	Ограничения	Описание
UserId	uuid	PK, NOT NULL	Идентификатор заблокированного пользователя
BlockedAtUtc	timestamp with time zone	NOT NULL	Дата блокировки

Таблица Д.14 – Атрибуты таблицы PlayerRatings

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор записи рейтинга
PlayerId	uuid	NOT NULL	Идентификатор игрока
DisciplineCode	character varying(64)	NOT NULL	Код дисциплины
Elo	integer	NOT NULL	Текущий рейтинг ELO
MatchesPlayed	integer	NOT NULL	Количество сыгранных матчей
Wins	integer	NOT NULL	Количество побед
Losses	integer	NOT NULL	Количество поражений
IsDeleted	boolean	NOT NULL	Признак удаления (soft-delete)
CreatedAtUtc	timestamp with time zone	NOT NULL	Дата создания записи
UpdatedAtUtc	timestamp with time zone	NOT NULL	Дата последнего обновления
RowVersion	bytea	NOT NULL	Версия строки (оптимистическая блокировка)

Таблица Д.15 – Атрибуты таблицы RatingHistories

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор записи истории
PlayerId	uuid	NOT NULL	Идентификатор игрока
DisciplineCode	character varying(64)	NOT NULL	Код дисциплины
OldElo	integer	NOT NULL	Рейтинг до изменения
NewElo	integer	NOT NULL	Рейтинг после изменения
Delta	integer	NOT NULL	Величина изменения рейтинга
Reason	character varying(256)	NOT NULL	Причина изменения
MatchId	uuid	NULL	Идентификатор матча (если изменение связано с матчем)
TournamentId	uuid	NULL	Идентификатор турнира
CreatedAtUtc	timestamp with time zone	NOT NULL	Дата изменения

Таблица Д.16 – Атрибуты таблицы Notifications

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор уведомления
RecipientUserId	uuid	NOT NULL	Идентификатор получателя
Type	character varying(32)	NOT NULL	Тип уведомления
Title	character varying(256)	NOT NULL	Заголовок
Body	character varying(2048)	NOT NULL	Текст уведомления
LinkUrl	character varying(512)	NOT NULL	Ссылка для перехода
PayloadJson	jsonb	NOT NULL	Дополнительные данные в формате JSON
SourceEventId	uuid	NOT NULL	Идентификатор исходного интеграционного события
ReadAtUtc	timestamp with time zone	NULL	Дата прочтения
CreatedAtUtc	timestamp with time zone	NOT NULL	Дата создания

Таблица Д.17 – Атрибуты таблицы OutboxMessages

Атрибут	Тип данных	Ограничения	Описание
Id	uuid	PK, NOT NULL	Идентификатор сообщения
EventId	uuid	NOT NULL, UNIQUE	Идентификатор интеграционного события
EventType	character varying(256)	NOT NULL	Тип события
Payload	jsonb	NOT NULL	Сериализованное событие
OccurredAtUtc	timestamp with time zone	NOT NULL	Дата возникновения события
ProcessedAtUtc	timestamp with time zone	NULL	Дата публикации в брокер сообщений
RetryCount	integer	NOT NULL	Количество попыток публикации
Error	character varying(2048)	NULL	Текст последней ошибки публикации

Таблица Д.18 – Атрибуты таблицы InboxMessages

Атрибут	Тип данных	Ограничения	Описание
EventId	uuid	PK, NOT NULL	Идентификатор интеграционного события
ConsumerName	character varying(256)	PK, NOT NULL	Имя потребителя события
ProcessedAtUtc	timestamp with time zone	NOT NULL	Дата обработки события

Приложение Е

(рекомендуемое)

Графическое представление классов эквивалентности

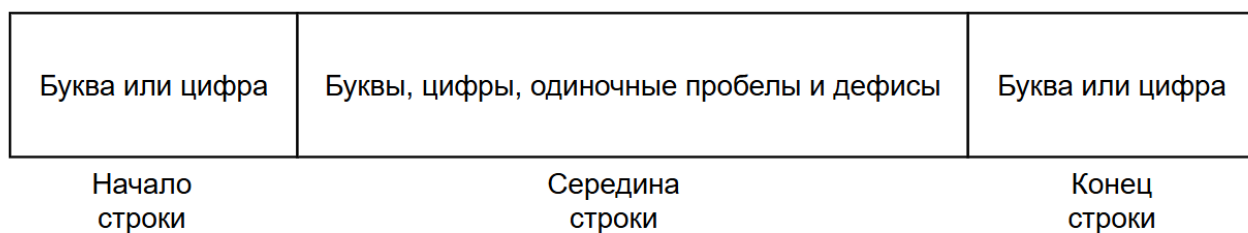


Рисунок Е.1 – Графическое представление классов эквивалентности структуры названия турнира

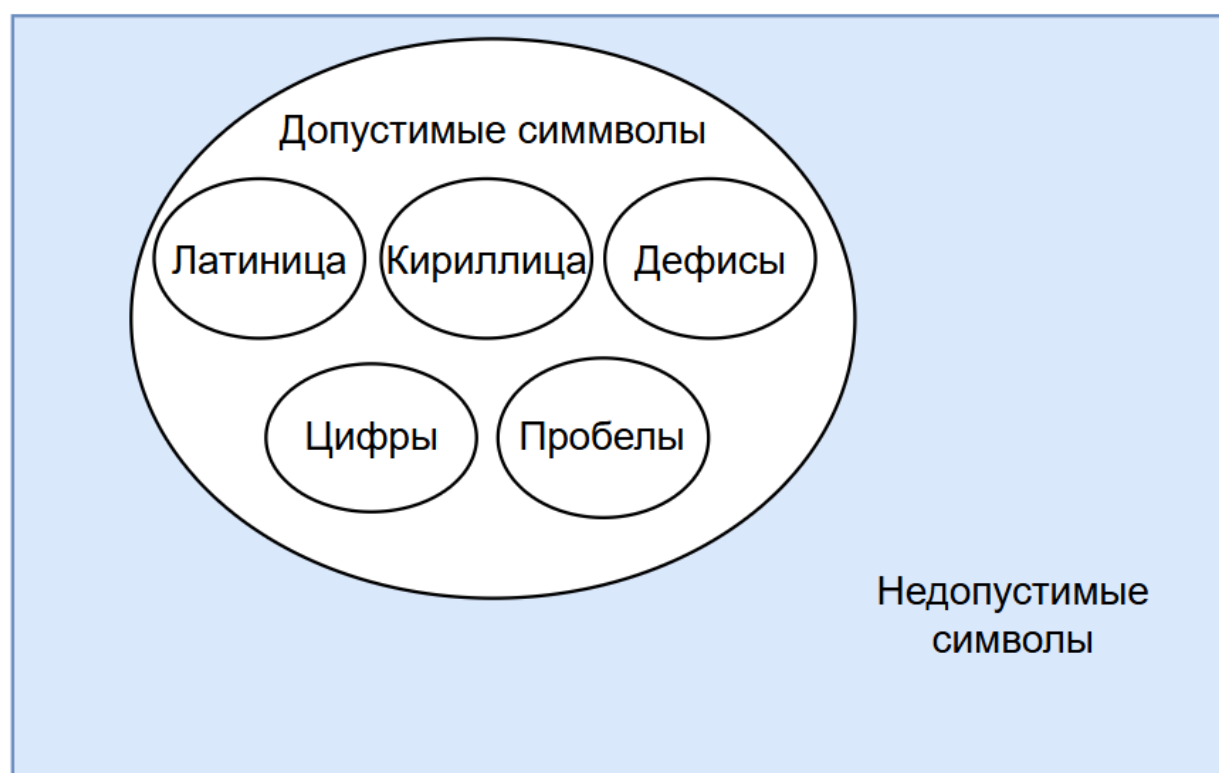


Рисунок Е.2 – Графическое представление классов эквивалентности названия турнира

Приложение Ж

(рекомендуемое)

Код юнит-тестов

```

using Tournament.Application.Brackets;
using Tournament.Application.Matches;
using Tournament.Application.Tournaments;
using Tournament.Application.Tournaments.Abstractions;
using Tournament.Application.Tournaments.Dto;
using Tournament.Domain.Tournaments;
using TournamentPlatform.Contracts.Enums;
using TournamentPlatform.Contracts.Events;

namespace Tournament.Tests;

public sealed class MatchResultServiceTests
{
    [Fact]
    public async Task
    OrganizerOwner_CanCompleteMatch_AndPublishesMatchCompletedOnce()
    {
        var fixture = CreateFixture();

        var result = await fixture.Service.CompleteMatchAsync(
            fixture.Tournament.Id,
            fixture.Match.Id,
            new CompleteMatchRequest(fixture.Match.TeamAId!.Value,
2, 0, 2, 0, false),
            new
CurrentTournamentUser(fixture.Tournament.OrganizerId, "Organizer",
"Active"));

        Assert.True(result.IsSuccess);
        Assert.Equal(MatchStatus.Completed, fixture.Match.Status);

        Assert.Single(fixture.Outbox.Events.OfType<MatchCompletedEvent>())
;
    }

    [Fact]
    public async Task OtherOrganizer_GetsAccessDenied()
    {
        var fixture = CreateFixture();

        var result = await fixture.Service.CompleteMatchAsync(
            fixture.Tournament.Id,
            fixture.Match.Id,
            new CompleteMatchRequest(fixture.Match.TeamAId!.Value,
2, 0, 2, 0, false),
            new CurrentTournamentUser(Guid.NewGuid(), "Organizer",
"Active"));

        Assert.True(result.IsFailure);

        Assert.Equal(Tournament.Application.Tournaments.TournamentErrors.A
ccessDenied, result.Error);
    }
}

```

```

    }

    [Fact]
    public async Task Player_GetsAccessDenied()
    {
        var fixture = CreateFixture();

        var result = await fixture.Service.CompleteMatchAsync(
            fixture.Tournament.Id,
            fixture.Match.Id,
            new CompleteMatchRequest(fixture.Match.TeamAId!.Value,
2, 0, 2, 0, false),
            new CurrentTournamentUser(Guid.NewGuid(), "Player",
"Active"));

        Assert.True(result.IsFailure);

        Assert.Equal(Tournament.Application.Tournaments.TournamentErrors.A
ccessDenied, result.Error);
    }

    [Fact]
    public async Task Admin_CanCompleteAnyMatch()
    {
        var fixture = CreateFixture();

        var result = await fixture.Service.CompleteMatchAsync(
            fixture.Tournament.Id,
            fixture.Match.Id,
            new CompleteMatchRequest(fixture.Match.TeamAId!.Value,
2, 0, 2, 0, false),
            new CurrentTournamentUser(Guid.NewGuid(), "Admin",
"Active"));

        Assert.True(result.IsSuccess);
    }

    [Fact]
    public async Task InvalidWinner_ReturnsFailure()
    {
        var fixture = CreateFixture();

        var result = await fixture.Service.CompleteMatchAsync(
            fixture.Tournament.Id,
            fixture.Match.Id,
            new CompleteMatchRequest(Guid.NewGuid(), 2, 0, 2, 0,
false),
            new CurrentTournamentUser(Guid.NewGuid(), "Admin",
"Active"));

        Assert.True(result.IsFailure);
    }

```

```

Assert.Equal(Tournament.Application.Tournaments.TournamentErrors.InvalidWinnerTeam, result.Error);
    }

    [Fact]
    public async Task WinnerScoreMustBeGreaterThanLoserScore()
    {
        var fixture = CreateFixture();

        var result = await fixture.Service.CompleteMatchAsync(
            fixture.Tournament.Id,
            fixture.Match.Id,
            new CompleteMatchRequest(fixture.Match.TeamAId!.Value,
1, 1, 1, 1, false),
            new CurrentTournamentUser(Guid.NewGuid(), "Admin",
"Active"));

        Assert.True(result.IsFailure);

Assert.Equal(Tournament.Application.Tournaments.TournamentErrors.InvalidMatchScore, result.Error);
    }

    [Fact]
    public async Task WinnerMapsMustBeGreaterThanLoserMaps()
    {
        var fixture = CreateFixture();

        // Rounds are valid (37-29), but maps say it's a draw –
service rejects.
        var result = await fixture.Service.CompleteMatchAsync(
            fixture.Tournament.Id,
            fixture.Match.Id,
            new CompleteMatchRequest(fixture.Match.TeamAId!.Value,
37, 29, 1, 1, false),
            new CurrentTournamentUser(Guid.NewGuid(), "Admin",
"Active"));

        Assert.True(result.IsFailure);

Assert.Equal(Tournament.Application.Tournaments.TournamentErrors.InvalidMatchScore, result.Error);
    }

    [Fact]
    public async Task ValidMapsAndRoundsScore_StoresBoth()
    {
        var fixture = CreateFixture();

        // Bo3 sweep 2-0 by maps, rounds 26-8 total.
        var result = await fixture.Service.CompleteMatchAsync(

```

```

        fixture.Tournament.Id,
        fixture.Match.Id,
        new CompleteMatchRequest(fixture.Match.TeamAId!.Value,
26, 8, 2, 0, false),
        new
CurrentTournamentUser(fixture.Tournament.OrganizerId, "Organizer",
"Active"));

    Assert.True(result.IsSuccess);
    Assert.Equal(26, fixture.Match.WinnerScore);
    Assert.Equal(8, fixture.Match.LoserScore);
    Assert.Equal(2, fixture.Match.WinnerMaps);
    Assert.Equal(0, fixture.Match.LoserMaps);
}

[Fact]
public async Task RegularMatchWithoutScore_ReturnsFailure()
{
    var fixture = CreateFixture();

    var result = await fixture.Service.CompleteMatchAsync(
        fixture.Tournament.Id,
        fixture.Match.Id,
        new CompleteMatchRequest(fixture.Match.TeamAId!.Value,
null, null, null, null, false),
        new CurrentTournamentUser(Guid.NewGuid(), "Admin",
"Active"));

    Assert.True(result.IsFailure);

    Assert.Equal(Tournament.Application.Tournaments.TournamentErrors.M
atchScoreRequired, result.Error);

    Assert.Empty(fixture.Outbox.Events.OfType<MatchCompletedEvent>());
}

[Fact]
public async Task
TechnicalDefeatWithoutScore_Succeeds_AndStoresNullScore()
{
    var fixture = CreateFixture();

    var result = await fixture.Service.CompleteMatchAsync(
        fixture.Tournament.Id,
        fixture.Match.Id,
        new CompleteMatchRequest(fixture.Match.TeamAId!.Value,
null, null, null, null, true),
        new CurrentTournamentUser(Guid.NewGuid(), "Admin",
"Active"));

    Assert.True(result.IsSuccess);
    Assert.Equal(MatchStatus.Completed, fixture.Match.Status);
}

```

```

        Assert.Null(fixture.Match.WinnerScore);
        Assert.Null(fixture.Match.LoserScore);
        Assert.True(fixture.Match.IsTechnicalDefeat);

Assert.Single(fixture.Outbox.Events.OfType<MatchCompletedEvent>())
;
    }

    [Fact]
    public async Task
CompletingAlreadyCompletedMatch_ReturnsConflictAndDoesNotPublishTw
ice()
    {
        var fixture = CreateFixture();
        var admin = new CurrentTournamentUser(Guid.NewGuid(),
"Admin", "Active");

        await fixture.Service.CompleteMatchAsync(
            fixture.Tournament.Id,
            fixture.Match.Id,
            new CompleteMatchRequest(fixture.Match.TeamAId!.Value,
2, 0, 2, 0, false),
            admin);
        var result = await fixture.Service.CompleteMatchAsync(
            fixture.Tournament.Id,
            fixture.Match.Id,
            new CompleteMatchRequest(fixture.Match.TeamAId!.Value,
2, 0, 2, 0, false),
            admin);

        Assert.True(result.IsFailure);

Assert.Equal(Tournament.Application.Tournaments.TournamentErrors.M
atchAlreadyCompleted, result.Error);

Assert.Single(fixture.Outbox.Events.OfType<MatchCompletedEvent>())
;
    }

    private static Fixture CreateFixture()
    {
        var tournament = Domain.Tournaments.Tournament.Create(
            "Match Cup",
            "MATCH CUP",
            null,
            "CS2",
            TournamentFormat.SingleElimination,
            null,
            1,
            2,
            Guid.NewGuid(),
            DateTime.UtcNow);
    }

```

```

        var teamA = Team.Create(tournament.Id, "A",
Guid.NewGuid(), 1, 1100, [TeamMember.Create(Guid.NewGuid(), "A",
1100)]);
        var teamB = Team.Create(tournament.Id, "B",
Guid.NewGuid(), 2, 1000, [TeamMember.Create(Guid.NewGuid(), "B",
1000)]);
        tournament.AddTeams([teamA, teamB]);
        var round = Round.Create(tournament.Id, 1,
BracketType.Main, DateTime.UtcNow);
        round.Start();
        var match = Match.Create(tournament.Id, 1, teamA.Id,
teamB.Id, DateTime.UtcNow);
        round.AddMatch(match);
        tournament.AddRound(round);
        tournament.Start(DateTime.UtcNow);

        var repository = new
InMemoryTournamentRepository(tournament);
        var outbox = new InMemoryOutboxWriter();
        IBracketGenerator[] generators =
        [
            new SingleEliminationBracketGenerator(outbox),
            new DoubleEliminationBracketGenerator(outbox),
            new SwissBracketGenerator(outbox)
        ];
        var service = new MatchResultService(repository, new
BracketGeneratorFactory(generators), outbox);
        return new Fixture(tournament, match, service, outbox);
    }

    private sealed record Fixture(
        Domain.Tournaments.Tournament Tournament,
        Match Match,
        MatchResultService Service,
        InMemoryOutboxWriter Outbox);

    private sealed class InMemoryOutboxWriter : IOutboxWriter
    {
        public List<IntegrationEvent> Events { get; } = [];
        public void Add(IntegrationEvent integrationEvent) =>
Events.Add(integrationEvent);
    }

    private sealed class
InMemoryTournamentRepository(Domain.Tournaments.Tournament
tournament) : ITournamentRepository
    {
        public Task<bool> TitleExistsAsync(string normalizedTitle,
CancellationToken cancellationToken = default) =>
Task.FromResult(false);
    }

```

```

        public Task<Discipline?> GetActiveDisciplineAsync(string
disciplineCode, CancellationToken cancellationToken = default) =>
Task.FromResult<Discipline?>(null);
        public Task<Domain.Tournaments.Tournament?>
GetByIdAsync(Guid id, CancellationToken cancellationToken =
default) => Task.FromResult(id == tournament.Id ? tournament :
null);
        public Task<IReadOnlyCollection<TournamentSummaryDto>>
GetAllAsync(CancellationToken cancellationToken = default) =>
Task.FromResult<IReadOnlyCollection<TournamentSummaryDto>>([ToSumm
ary(tournament)]);
        public Task<IReadOnlyCollection<TournamentSummaryDto>>
GetByStatusAsync(TournamentStatus status, CancellationToken
cancellationToken = default) =>
Task.FromResult<IReadOnlyCollection<TournamentSummaryDto>>([]);
        public Task<IReadOnlyCollection<TournamentSummaryDto>>
GetAvailableAsync(CancellationToken cancellationToken = default)
=> Task.FromResult<IReadOnlyCollection<TournamentSummaryDto>>([]);
        public Task<IReadOnlyCollection<TournamentSummaryDto>>
GetByOrganizerAsync(Guid organizerId, CancellationToken
cancellationToken = default) =>
Task.FromResult<IReadOnlyCollection<TournamentSummaryDto>>([]);
        public Task<IReadOnlyCollection<TournamentSummaryDto>>
GetByPlayerAsync(Guid playerId, CancellationToken
cancellationToken = default) =>
Task.FromResult<IReadOnlyCollection<TournamentSummaryDto>>([]);
        public Task<bool> BlockedUserExistsAsync(Guid userId,
CancellationToken cancellationToken = default) =>
Task.FromResult(false);
        public Task<ITournamentTransaction>
BeginTransactionAsync(CancellationToken cancellationToken =
default) => throw new NotImplementedException();
        public void Add(Domain.Tournaments.Tournament value) { }
        public Task SaveChangesAsync(CancellationToken
cancellationToken = default) => Task.CompletedTask;

        private static TournamentSummaryDto
ToSummary(Domain.Tournaments.Tournament value)
        {
            return new TournamentSummaryDto(
                value.Id,
                value.Title,
                value.Description,
                value.DisciplineCode,
                value.Format,
                value.SwissRounds,
                value.TeamSize,
                value.MaxPlayers,
                value.OrganizerId,
                value.Status,
                value.CurrentRoundNumber,
                value.ActiveParticipantsCount,

```

```
        value.CreatedAtUtc,  
        value.StartedAtUtc,  
        value.CompletedAtUtc,  
        value.CancelledAtUtc);  
    }  
}  
}
```

Приложение И
(рекомендуемое)
Справка об апробации

AB SOFT

АБ СОФТ ООО

ЮР. АДРЕС: 422110 / г. КУЙБИР, ул. ЛЕНИНА, д. 24к2, пом 43 /
MAIL: MANAGER@AXLEBOLT.COM / WEB: ABSOFT.STUDIO

ПОДПИСЬНАЯ ТАБЛИЦА



СПРАВКА

об апробации результатов выпускной квалификационной работы

Настоящей справкой подтверждается, что результаты выпускной квалификационной работы студента Смирнова Ильи Алексеевича на тему: «Веб-приложение для организации и проведения турниров по компьютерным играм» представляют практический интерес для ООО «АБ СОФТ», принимающего участие в разработке и сопровождении мобильной киберспортивной игры, в рамках которой организуются онлайн- и офлайн-турниры.

Рассмотренные в работе решения - микросервисная архитектура турнирной платформы, алгоритм автоматического формирования сбалансированных по уровню подготовки команд (на основе целочисленного линейного программирования) и система внутреннего рейтинга игроков (на основе системы ELO) - соответствуют задачам организации и проведения соревнований в рассматриваемой предметной области.

По итогам апробации указанные результаты получили положительную оценку с точки зрения технической реализации, соответствия задачам организации турниров и учета особенностей предметной области. Результаты работы рекомендованы к дальнейшему использованию при разработке и совершенствовании внутренних инструментов проведения турниров компании.

Директор ООО «АБ СОФТ»



/ Замалиев Р.А. /