Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА ПО ПРОГРАММЕ БАКАЛАВРИАТА

09.03.04 Программная инженерия

(код, наименование ОПОП ВО: направление подготовки, направленность (профиль))

«Разработка программно-информационных систем»

Веб-платформа для анализа и визуализации статистических данных киберспортивной игры Counter-Strike 2

(название темы)

Дипломный проект

(вид ВКР: дипломная работа или дипломный проект)

Автор ВКР		И. А. Авилов
	(подпись, дата)	(инициалы, фамилия)
Группа ПО-01б		
Руководитель ВКР		Е. И. Аникина
	(подпись, дата)	(инициалы, фамилия)
Нормоконтроль		А. А. Чаплыгин
	(подпись, дата)	(инициалы, фамилия)
ВКР допущена к защите:		
Заведующий кафедрой		А. В. Малышев
	(подпись, дата)	(инициалы, фамилия)

Курск 2024 г.

Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

УТВЕРЖДАЮ:

Заведующий кафедрой

_____

(подпись, инициалы, фамилия)

«____» _____ 20____ г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ ПО ПРОГРАММЕ БАКАЛАВРИАТА

Студента Авилова И.А., шифр 20-06-0390, группа ПО-01б

1. Тема «Веб-платформа для анализа и визуализации статистических данных киберспортивной игры Counter-Strike 2» утверждена приказом ректора ЮЗГУ от «04» апреля 2024 г. № 1620-с.

2. Срок предоставления работы к защите «11» июня 2024 г.

3. Исходные данные для создания программной системы:

3.1. Перечень решаемых задач:

1) создание базы данных всех киберспортивных матчей;
2) создание нейронной сети для прогнозирования матчей;
3) реализация вывода статистики на веб-платформу;
4) реализация анализа метрик из .dem записей игр.

3.2. Входные данные и требуемые результаты для программы:

1) Входными данными для программной системы являются: записи прошедших матчей в формате .dem; данные с информацией о турнире; метрики игроков из матчей; данные отправленные пользователем через API.

2) Выходными данными для программной системы являются: статистика игроков из матча; прогноз на выбранный матч; данные выбранного турнира; данные выбранного матча.

4. Содержание работы (по разделам):

4.1. Введение

4.2. Анализ предметной области

4.3. Техническое задание: основание для разработки, назначение разработки, требования к веб-платформе, нефункциональные требования к программной системе.

4.4. Технический проект: общие сведения о программной системе, проектирование архитектуры программной системы, обоснование выбора технологий проектирования и программных средств, проектирование пользовательского интерфейса программной системы.

4.5. Рабочий проект: описание сущностей веб-платформы, настройка взаимодействия сервисов, тестирование веб-платформы.

4.6. Заключение

4.7. Список использованных источников

5. Перечень графического материала:

Лист 1. Сведения о ВКРБ

Лист 2. Цель и задачи разработки

Лист 3. Концептуальная модель программной системы

Лист 4. Диаграмма прецедентов

Лист 5. Диаграмма архитектуры программной системы

Лист 6. Заключение

Руководитель ВКР	_____	Е. И. Аникина
	(подпись, дата)	(инициалы, фамилия)
Задание принял к исполнению	_____	И. А. Авилов
	(подпись, дата)	(инициалы, фамилия)

РЕФЕРАТ

Объем работы равен 141 странице. Работа содержит 19 иллюстраций, 93 таблицы, 13 библиографических источников и 6 листов графического материала. Количество приложений – 2. Графический материал представлен в приложении А. Фрагменты исходного кода представлены в приложении Б.

Перечень ключевых слов: киберспорт, Counter-Strike 2, визуализация данных, нейронные сети, .dem файлы, API, веб-платформа, микросервисная архитектура, контейнеризация, Docker, Kubernetes, RabbitMQ, PostgreSQL, C#, React, Entity Framework, Golang, аналитика, статистика, REST API, frontend, backend.

Объектом разработки является веб-платформа для анализа и визуализации статистических данных киберспортивной игры Counter-Strike 2.

Целью выпускной квалификационной работы является создание эффективного инструмента для киберспортивных команд и аналитиков, который позволит улучшить игровые навыки и стратегии путем глубокого анализа игровых данных и прогнозов.

В процессе создания веб-платформы были выделены основные сущности путем анализа игровых данных, использованы классы и методы модулей, обеспечивающие работу с сущностями предметной области, а также корректную работу веб-платформы. Были разработаны следующие сервисы: API Gateway, сервис поиска информации о турнирах, сервис анализа .dem файлов, сервис предоставления данных, сервис поиска, сервис прогнозирования.

При разработке веб-платформы использовались современные технологии, такие как микросервисная архитектура, контейнеризация с Docker и Kubernetes, брокер сообщений RabbitMQ, а также базы данных PostgreSQL и ORM Entity Framework. Фронтенд часть была реализована с использованием React и MaterialUI.

Разработанная веб-платформа успешно прошла интеграционное и нагрузочное тестирование, что подтвердило её стабильную работу и высокую производительность.

ABSTRACT

The volume of work is 141 page. The work contains 19 illustrations, 93 tables, 13 bibliographic sources and 6 sheets of graphic material. The number of applications is 2. The graphic material is presented in annex A. The fragment of the source code is provided in annex B.

List of keywords: esports, Counter-Strike 2, data visualization, neural networks, .dem files, API, web platform, microservice architecture, containerization, Docker, Kubernetes, RabbitMQ, PostgreSQL, C#, React, Entity Framework, Golang, analytics, statistics, REST API, frontend, backend.

The object of the development is a web platform for the analysis and visualization of statistical data of the esports game Counter-Strike 2.

The purpose of the final qualifying work is to create an effective tool for esports teams and analysts, which will improve gaming skills and strategies through deep analysis of game data and predictions.

In the process of creating the web platform, the main entities were identified through the analysis of game data. Classes and methods of modules were used to ensure the work with the entities of the subject area, as well as the correct operation of the web platform. The following services were developed: API Gateway, tournament information search service, .dem file analysis service, data provision service, search service, and prediction service.

During the development of the web platform, modern technologies such as microservice architecture, containerization with Docker and Kubernetes, RabbitMQ message broker, as well as PostgreSQL databases and ORM Entity Framework were used. The frontend part was implemented using React and MaterialUI.

The developed web platform successfully passed integration and load testing, confirming its stable operation and high performance.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 13
1 Анализ предметной области 15
1.1 Описание предметной области 15
1.2 Классификация киберспортивных дисциплин 16
1.3 Компьютерная игра Counter Strike 2 17
1.4 Сбор, хранение и анализ статистических данных игры Counter Strike 2 19
2 Техническое задание 22
2.1 Основание для разработки 22
2.2 Назначение разработки 22
2.3 Требования к веб-платформе 22
2.3.1 Требования к данным веб-платформы 22
2.3.2 Функциональные требования к веб-платформе 24
2.3.2.1 Вариант использования «Просмотр информации игрока» 25
2.3.2.2 Вариант использования «Просмотр рейтинга игроков» 26
2.3.2.3 Вариант использования «Добавление игрока для сравнения» 26
2.3.2.4 Вариант использования «Сравнение статистики игроков» 27
2.3.2.5 Вариант использования «Просмотр информации о турнире» 27
2.3.2.6 Вариант использования «Просмотр информации о матче» 28
2.3.2.7 Вариант использования «Просмотр статистики матча» 28
2.3.2.8 Вариант использования «Отображение прогноза на матч» 28
2.3.2.9 Вариант использования «Просмотр информации команды» 29
2.3.2.10 Вариант использования «Просмотр рейтинга команд» 29
2.3.2.11 Вариант использования «Добавление команды для сравнения» 30
2.3.2.12 Вариант использования «Сравнение статистики команд» 30
2.3.3 Требования пользователя к интерфейсу веб-платформы 30
2.4 Нефункциональные требования к программной системе 32
2.4.1 Требования к архитектуре 32
2.4.2 Требования к надежности 32

2.4.3 Требования к программному обеспечению 32
2.4.4 Требования к аппаратному обеспечению 33
2.4.5 Требования к оформлению документации 33
3 Технический проект 34
3.1 Общие сведения о программной системе 34
3.2 Проектирование архитектуры программной системы 35
3.2.1 Выбор архитектурного стиля и паттернов проектирования 35
3.2.2 Структура базы данных 37
3.2.3 Описание микросервисов 37
3.2.4 Планирование докеризации и оркестрации сервисов 38
3.2.5 Описание REST API микросервисов 39
3.3 Обоснование выбора технологий проектирования и программных средств 43
3.3.1 Выбор используемых технологий и языков программирования 43
3.3.1.1 API Gateway 43
3.3.1.2 Сервис поиска информации о турнирах 43
3.3.1.3 Сервис поиска информации о турнирах 43
3.3.1.4 Сервис предоставления данных 43
3.3.1.5 Сервис поиска 44
3.3.1.6 Сервис прогнозирования 44
3.3.1.7 Брокер сообщений (RabbitMQ) 44
3.3.1.8 Prometheus 44
3.3.1.9 React 45
3.3.2 Выбор программного обеспечения 46
3.3.3 Docker 47
3.3.4 Redis 49
3.3.5 Kubernetes 50
3.3.6 RabbitMQ 50
3.3.7 Prometheus 52

3.4 Проектирование пользовательского интерфейса программной системы 52
3.4.1 Макеты пользовательского интерфейса 52
3.4.2 Компоненты пользовательского интерфейса 55
3.4.2.1 MiddleContent 55
3.4.2.2 MapStatsTable 56
3.4.2.3 PlayerStats 57
3.4.2.4 MapPulStatsTable 58
4 Рабочий проект 60
4.1 Описание сущностей веб-платформы 60
4.1.1 Описание сущностей базы данных 60
4.1.2 Описание внутренних сущностей сервиса анализа .dem файла 73
4.1.3 Описание внутренних сущностей сервиса поиска информации о турнирах 77
4.1.4 Описание внутренних сущностей сервиса поиска 87
4.1.5 Спецификация классов веб-платформы 89
4.1.6 Спецификация классов сервиса анализа .dem файла 89
4.1.7 Спецификация классов сервиса поиска информации о турнирах 105
4.1.8 Спецификация классов сервиса поиска 115
4.1.9 Спецификация классов сервиса прогнозирования 116
4.2 Настройка взаимодействия сервисов 118
4.3 Тестирование веб-платформы 120
4.3.1 Unit тестирование сервисов 120
4.3.2 Описание unit тестов сервиса анализа .dem файла 120
4.3.3 Описание unit тестов сервиса предоставления данных 121
4.3.4 Описание unit тестов сервиса поиска информации о турнирах 122
4.3.5 Интеграционное тестирование веб-платформы 123
ЗАКЛЮЧЕНИЕ 125
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 127
ПРИЛОЖЕНИЕ А Представление графического материала 129

ПРИЛОЖЕНИЕ Б Фрагменты исходного кода программы 136
На отдельных листах (CD-RW в прикрепленном конверте) 141

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

БД – база данных.

ИС – информационная система.

ИТ – информационные технологии.

ПО – программное обеспечение.

РП – рабочий проект.

СУБД – система управления базами данных.

ТЗ – техническое задание.

ТП – технический проект.

REST API – интерфейс программирования приложений, основанный на принципах REST (Representational State Transfer), который используется для взаимодействия между различными системами через HTTP.

CS2 (Counter-Strike 2) – многопользовательская онлайн-игра, разработанная компанией Valve.

HTTP – протокол передачи гипертекста (HyperText Transfer Protocol), используемый для передачи данных в интернете.

DTO (Data Transfer Object) – объект передачи данных, используемый для передачи данных между различными слоями приложения.

MVP (Most Valuable Player) – наиболее ценный игрок, термин, используемый для обозначения лучшего игрока в матче или турнире.

UML (Unified Modelling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения.

Frontend – клиентская часть приложения, которая отвечает за взаимодействие с пользователем и отображение данных.

Backend – серверная часть приложения, которая отвечает за обработку данных, взаимодействие с базой данных и реализацию бизнес-логики.

ORM (Object-Relational Mapping) – технология программирования, которая позволяет преобразовывать данные между несовместимыми типами систем, используя объектно-ориентированные языки программирования. ORM упрощает взаимодействие между объектами в приложении и записями

в реляционной базе данных, обеспечивая разработчикам возможность работать с базой данных, как если бы она была коллекцией объектов.

ВВЕДЕНИЕ

Киберспорт, или профессиональный игровой спорт, за последние годы стал одним из самых быстрорастущих видов спорта в мире. Ежегодно миллионы зрителей следят за международными турнирами по таким играм, как Counter-Strike 2 (CS2), Dota 2, Warcraft III, где сотни команд соревнуются за престижные трофеи и крупные призовые фонды. В этом контексте аналитика и визуализация игровых данных становятся неотъемлемой частью подготовки к матчам и стратегического планирования.

Counter-Strike 2 — это одна из ведущих киберспортивных дисциплин, известная своей глубокой тактической составляющей и требующая от игроков высокого уровня индивидуального мастерства и командной работы. Анализ игровых данных, таких как статистика игроков, результаты матчей и метрики производительности, играет ключевую роль в разработке стратегий и принятии решений. Использование данных позволяет командам и тренерам выявлять слабые и сильные стороны, оптимизировать тактики и прогнозировать результаты будущих матчей.

Цель настоящей работы – разработка веб-платформы для анализа и визуализации статистических данных киберспортивной игры Counter-Strike 2. Данная платформа должна предоставить киберспортивным командам, тренерам и аналитикам мощный инструмент для глубокого анализа игровых метрик и поддержки принятия решений на основе данных. Для достижения поставленной цели необходимо решить *следующие задачи:*

- провести анализ предметной области;
- разработать концептуальную модель веб-платформы;
- создать базу данных всех киберспортивных матчей;
- создать нейронную сеть для прогнозирования матчей;
- реализовать вывода статистики на веб-платформу;
- реализовать анализа метрик из .dem записей игр.

Структура и объем работы. Отчет состоит из введения, 4 разделов основной части, заключения, списка использованных источников, 2 приложений. Текст выпускной квалификационной работы равен 141 странице.

Во введении сформулирована цель работы, поставлены задачи разработки, описана структура работы, приведено краткое содержание каждого из разделов.

В первом разделе на стадии описания предметной области собирается информация о сфере, для которой создается веб-платформа, и о существующих решениях в этой области.

Во втором разделе на стадии технического задания приводятся требования к разрабатываемой веб-платформе.

В третьем разделе на стадии технического проектирования представлены проектные решения для веб-платформы.

В четвертом разделе приводится список классов и их методов, использованных при разработке сайта, производится тестирование разработанного сайта.

В заключении излагаются основные результаты работы, полученные в ходе разработки.

В приложении А представлен графический материал.

В приложении Б представлены фрагменты исходного кода.

1 Анализ предметной области

1.1 Описание предметной области

Киберспорт — это соревновательная деятельность, связанная с компьютерными играми, где участники, индивидуальные игроки или команды, соревнуются друг с другом в специально организованных матчах и турнирах. Он объединяет технологии, спорт и развлечения, предлагая уникальное сочетание физического мастерства и интеллектуальных способностей. Киберспорт, или электронный спорт, в последние десятилетия превратился из ниши видеоигр в полноценную глобальную индустрию, объединяющую миллионы игроков и зрителей по всему миру. Начавшись как любительские соревнования по видеоиграм в 1996 году, киберспорт быстро набирал популярность, превратившись в организованные и профессионально управляемые международные турниры с крупными призовыми фондами.

В России киберспорт получил официальное признание как вид спорта в 2001 году, а в 2016 году была создана Федерация Компьютерного Спорта России. Это признание подчеркивает значимость и влияние киберспортивной индустрии не только как средства развлечения, но и как сектора, способного способствовать развитию технологий, маркетинга и массовой культуры в стране.

Рынок киберспорта демонстрирует впечатляющий рост. По данным различных аналитических агентств, глобальный рынок киберспорта в 2022 году оценивается в 1.4 миллиарда долларов, а количество его зрителей и фанатов за последний год превысило 500 миллионов человек. Это не только свидетельствует о популярности киберспорта, но и подчеркивает его потенциал с точки зрения рекламы и коммерческой выгоды.

Одним из наиболее знаковых аспектов киберспорта являются его масштабные турниры, которые привлекают внимание миллионов зрителей по всему миру. Турниры такие как The International по Dota 2, Чемпионат мира по League of Legends, и Major по Counter-Strike 2 не только собирают лучших

игроков со всего мира, но и предлагают зрелищные шоу, поддерживаемые крупными спонсорами и медиа платформами. Эти события стали важной частью культуры современного интернета, собирая за просмотром миллионы человек одновременно.

Особое внимание заслуживает международный турнир ”Игры будущего который пройдет в Казани в марте 2024 года. Этот масштабный турнир будет включать соревнования по 16 дисциплинам, в том числе CS2, Dota 2, League of Legends и многие другие. Событие обещает стать одним из самых значительных в истории российского киберспорта, собирая лучших игроков со всего мира и предлагая зрителям незабываемое зрелище.

1.2 Классификация киберспортивных дисциплин

Киберспортивные дисциплины могут быть классифицированы по жанрам игр, каждый из которых предлагает свои уникальные стратегии, навыки и форматы соревнований. Основные классы включают:

1. **Стратегии в реальном времени (RTS)**: Примеры игр включают StarCraft II и Warcraft III. Этот жанр требует быстрого стратегического планирования, управления ресурсами и армиями.

2. **Многопользовательские онлайн-боевые арены (MOBA)**: Ключевые игры - Dota 2 и League of Legends. Жанр акцентирует внимание на командных боях, тактическом взаимодействии и индивидуальных навыках игроков.

3. **Шутеры от первого лица (FPS)**: В эту категорию входят такие игры, как Counter-Strike 2 и Call of Duty, где фокусируется внимание на стрельбе, рефлексах, координации команды и стратегическом планировании.

4. **Спортивные симуляторы**: Примеры игр - FIFA и NBA 2K. Эти игры имитируют реальные спортивные состязания и требуют знания спортивных правил реальных игр и тактик.

5. **Боевики и файтинги**: Примеры включают Street Fighter и Super Smash Bros. Жанр сосредоточен на одиночных поединках и требует высокой точности и быстрой реакции.

6. **Карточные игры**: Игры, такие как Hearthstone и Magic: The Gathering Arena, включают элементы стратегического мышления, управления ресурсами и предвидения действий противника.

1.3 Компьютерная игра Counter Strike 2

Counter-Strike 2 (CS2) представляет собой обновление популярной игры Counter-Strike: Global Offensive (CS:GO), разработанное и представленное компанией Valve в 2012 году. Обновление CS2 было выпущено в сентябре 2023 года, принеся значительные улучшения и изменения, включая переход со старого движка Source на новый мощный игровой движок Source 2.Переход на движок Source 2 позволил улучшить графику и производительность игры, а также расширил возможности аналитики и визуализации данных.

CS2 привлекает миллионы игроков по всему миру и является первой 2 среди самых популярных игр в киберспортивной индустрии. По различным оценкам, после обновления игры на новый движок, средний онлайн игры вырос до 800 тысяч активных игроков, а крупные турниры, такие как PGL Major Stockholm 2021, привлекают до 2.748 миллионов зрителей одновременно за просмотром турнира.

Одним из ключевых аспектов выбора CS2 основной дисциплиной для разработки является сохранение записей игр в формате .dem файлов, содержащих подробную информацию о каждом игровом событии, включая движения, стрельбу, покупку и использование предметов игроками, что делает их ценным ресурсом для анализа и улучшения игровых навыков. Аналитики и тренеры используют .dem файлы для разбора игровых моментов, стратегий и тактик, применяемых командами и отдельными игроками. Эти файлы позволяют детально проанализировать игровой процесс, выявить ошибки и моменты для улучшения для каждого отдельного игрока.

Основные правила соревновательной игры CS2 следующие:

1. **Общая структура**: Игра состоит из раундов. В стандартном соревновательном режиме проводится 24 раунда, с командами, играющими 12 раундов за каждую сторону (террористы и контртеррористы).

2. **Цели раунда**: Команда террористов должна либо установить бомбу на одном из мест закладки бомбы и защитить её до взрыва, либо уничтожить всех членов команды контртеррористов. Команда контртеррористов должна либо предотвратить установку бомбы, либо обезвредить её после установки, либо уничтожить всех террористов.

3. **Покупка оружия и снаряжения**: В начале каждого раунда игроки имеют время для покупки оружия и снаряжения. Деньги для покупок зарабатываются в предыдущих раундах посредством убийства противников и победы/проигрыша в раундах.

4. **Победа в матче**: Победа достигается одной из команд, которая первой выигрывает 13 раундов. В случае ничьей (12-12), проводятся дополнительные время.

5. **Дополнительное время**: В дополнительное время команды играют дополнительные раунды, чтобы определить победителя. Это 6 раундов (3 раунда за каждую сторону). Команда, которая выигрывает 4 из этих 6 раундов, становится победителем. В случае ничьи (3-3) проводится еще одно дополнительное время. Дополнительное время добавляется до момента выявления победителя.

6. **Ограничения по времени и бомбе**: Каждый раунд имеет ограничение по времени (1 минута 55 секунд). Если террористы не установили бомбу до истечения времени, побеждают контртеррористы. После установки бомбы, у контртеррористов есть 40 секунд на её обезвреживание.

7. **Тактические паузы**: Команды могут проводить тактические паузы, на которых с командой может общаться их тренер. Во время тактических пауз проводится разбор ошибок и тактик, которые можно применить во время

игры. Каждой команде на 1 матч доступно 4 тактические паузы. Каждая тактическая пауза длится 30 секунд.

Эти метрики играют ключевую роль в анализе игровых стратегий и формировании тактик, помогая командам и игрокам улучшать свои результаты и эффективность на киберспортивной арене.

1.4 Сбор, хранение и анализ статистических данных игры Counter Strike 2

В .dem файлах игры CS2 содержится информация о различных игровых событиях. Эти события предоставляют ценные данные для анализа стратегий и поведения игроков во время матча. В CS2 анализируются различные метрики для оценки эффективности игроков и команд, основными из которых являются:

- **Процент попаданий в голову (Headshot)**: Отражает процент убийств в голову противников от общего числа убийств данным игроком, что отображает точность наведения игрока на цель.

- **Уровень точности стрельбы (Accuracy)**: Отражает процент попаданий по противникам от общего числа сделанных выстрелов.

- **Количество убийств и смертей (K/D Ratio)**: Соотношение между числом убийств игрока и количеством его смертей.

- **Использование утилит (Utility Usage)**: Анализ эффективности использования гранат, дымов, флэш-бангов и других предметов.

- **Позиционирование и перемещение (Positioning and Movement)**: Оценка стратегического расположения на карте и способности к маневрированию в бою.

- **Экономическое управление (Economic Management)**: Отслеживание способности к управлению финансами для покупки оружия и снаряжения.

Существует несколько популярных ресурсов и платформ, предоставляющих статистические данные и аналитику для игры CS2, среди которых основными являются HLTV и Liquipedia:

1. **HLTV**: Ведущий сайт в сфере киберспорта для CS2, предоставляющий результаты матчей, их краткую статистику и аналитические статьи. Данный сайт является официальной платформой хранения .dem записей всех матчей, которые были сыграны за все время.

2. **Liquipedia**: Является обширной вики-платформой по киберспорту, включающая информацию о турнирах, командах и игроках CS2. Она предоставляет статьи о событиях в сфере киберспорта, а также сведения о прошлых и предстоящих проводимых турнирах и матчах.

Дополнительно, веб-сайты, такие как GosuGamers, ESEA или bo3.gg, предлагают статистику и аналитику, связанную с матчами и турнирами CS2. Однако все выше приведенные платформы не предполагают детального изучения и анализа данных игр, а так же прогнозирования будущих исходов матчей.

Развитие технологий анализа данных и их визуализации открывает новые возможности для улучшения тренировочных процессов и стратегического планирования в шутерах от первого лица. Существующие решения, хотя и предоставляют важную информацию, часто ограничены в плане глубины анализа и персонализации данных под конкретные нужды команд или игроков. Разработка специализированной платформы, фокусирующейся на детальном анализе .dem файлов, а так же использовании нейросети для прогнозирования исхода матчей, позволит получить более глубокое понимание игровых процессов, взаимосвязь различных метрик между собой, выявить скрытые паттерны поведения игроков и оптимизировать стратегии команды.

Такая платформа будет включать функции для детального анализа игровых матчей, визуализации статистических данных и создания прогнозов на матчи, что даст командам и игрокам новые инструменты для повышения своего уровня игры. Это не только повысит конкурентоспособность команд

на международной арене, но и способствует развитию киберспорта как спортивной дисциплины.

2 Техническое задание

2.1 Основание для разработки

Основанием для разработки веб-платформа для анализа и визуализации статистических данных киберспортивной игры Counter-Strike 2 является задание на выпускную квалификационную работу приказ ректора ЮЗГУ от «04» апреля 2024 года № 1620-с «Об утверждении тем выпускных квалификационных работ и руководителей выпускных квалификационных работ».

2.2 Назначение разработки

Функциональное назначение разрабатываемой веб-платформы заключается в предоставлении игрокам и киберспортивным тренерам эффективного инструмента отображения статистических данных из матчей с целью повышения эффективности команд и игроков, а так же отображение и предоставление прогнозов на будущие исходы матчей.

Предполагается, что данной веб-платформой будут пользоваться как киберспортивные тренеры и команды, для анализа прошедших матчей и сравнения различных метрик во времени, так и обычные игроки, для получения статистических данных по киберспортивным матчам и предоставления прогнозов на будущие матчи.

Задачами разработки данной веб-платформы являются:

- создание базы данных всех киберспортивных матчей;
- создание нейронной сети для прогнозирования матчей;
- реализация вывода статистики на веб-платформу;
- реализация анализа метрик из .dem записей игр.

2.3 Требования к веб-платформе

2.3.1 Требования к данным веб-платформы

Входными данными для веб-платформы являются:

- записи прошедших матчей в формате .dem;

- данные с информацией о турнире;
- метрики игроков из матчей;
- данные отправленные пользователем через API.

Выходными данными для веб-платформы являются:

- статистика игроков из матча;
- прогноз на выбранный матч;
- данные выбранного турнира;
- данные выбранного матча.

На рисунке 2.1 представлены концептуальные классы backend части веб-платформы.

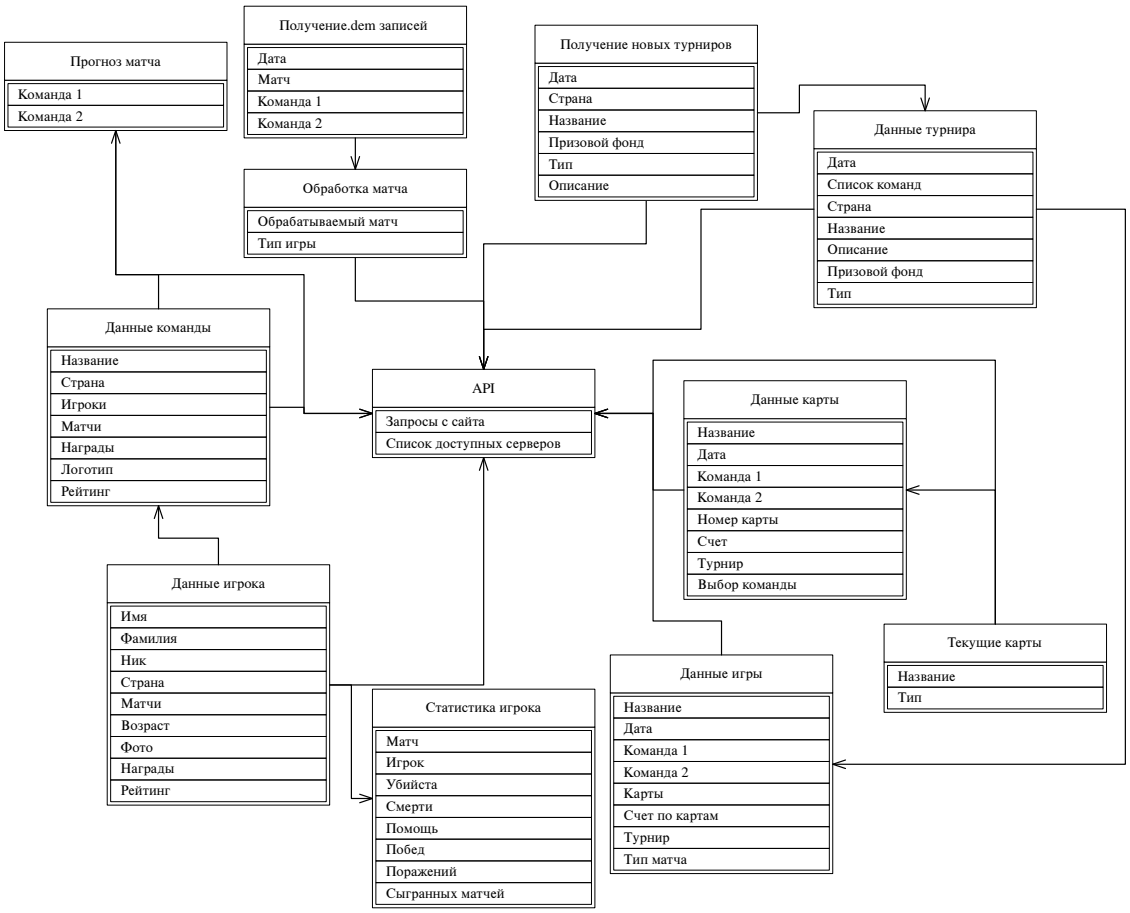

Рисунок 2.1 – Концептуальные классы backend части веб-платформы

2.3.2 Функциональные требования к веб-платформе

На основании анализа предметной области в разрабатываемой веб-платформы анализа и визуализации статистических данных киберспортивной игры CS2 должны быть реализованы следующие функции:

- просмотр информации игрока;
- просмотр рейтинга игрока;
- добавление игрока для сравнения;
- сравнение статистики игроков;
- просмотр информации о турнире;
- просмотр информации о матче;
- просмотр статистики матча;
- отображение прогноза на матч;
- просмотр информации команды;
- просмотр рейтинга команды;
- добавление команды для сравнения;
- сравнение статистики команд.

На рисунке 2.2 представлены функциональные требования к системе в виде диаграммы прецедентов нотации UML.

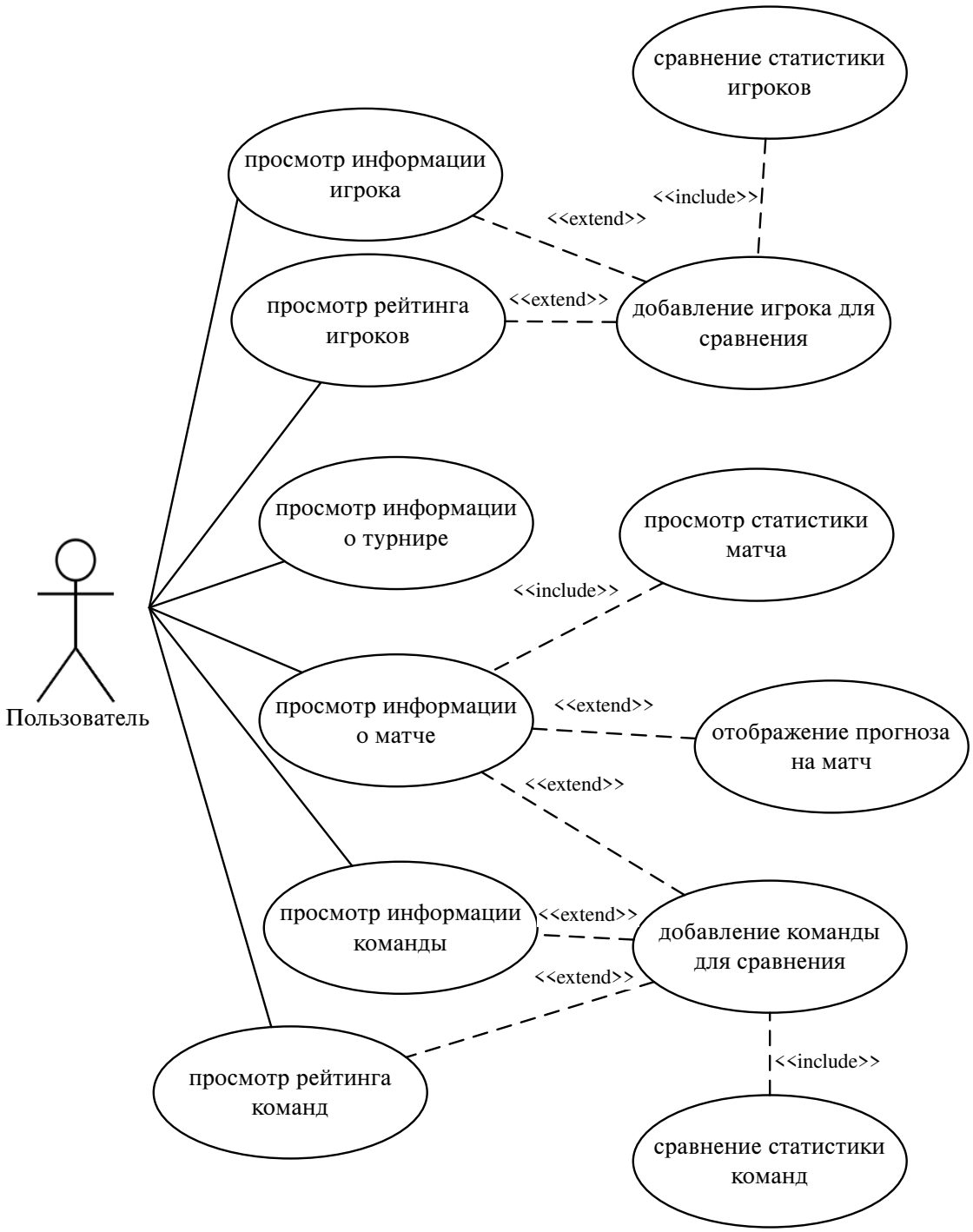

Рисунок 2.2 – Диаграмма прецедентов

2.3.2.1 Вариант использования «Просмотр информации игрока»

Заинтересованные лица и их требования: Пользователи веб-платформы, которые хотят получить детальную информацию о игроке.

Предусловие: Пользователь загружает главную страницу сайта.

Постусловие: Пользователь просматривает необходимую информацию о игроке.

Основной успешный сценарий:

1. Пользователь заходит на главную страницу.
2. Пользователь выбирает раздел с информацией об игроках.
3. Пользователь ищет интересующего его игрока фильтруя игроков по имени, команде или никнейму.
4. Система отображает профиль с подробной информацией об игроке.

2.3.2.2 Вариант использования «Просмотр рейтинга игроков»

Заинтересованные лица и их требования: Пользователи веб-платформы, которые хотят просмотреть рейтинг лучших игроков на данный момент.

Предусловие: Пользователь загружает главную страницу сайта.

Постусловие: Пользователь просматривает рейтинговую таблицу игроков.

Основной успешный сценарий:

1. Пользователь заходит на главную страницу.
2. Пользователь выбирает раздел с рейтинговой таблицей игроков.
3. Пользователь просматривает рейтинговую таблицу игроков.

2.3.2.3 Вариант использования «Добавление игрока для сравнения»

Заинтересованные лица и их требования: Пользователи веб-платформы, которые хотят добавить игрока для сравнения его статистики с другим игроком.

Предусловие: Пользователь находится на странице с рейтингом игроков.

Постусловие: Пользователь добавляет игрока для будущего сравнения.

Основной успешный сценарий:

1. Пользователь наводит курсор на игрока, которого хочет добавить для сравнения.
2. Пользователь нажимает кнопку ”Сравнить”.

3. Игрок добавляется в список для сравнения.

2.3.2.4 Вариант использования «Сравнение статистики игроков»

Заинтересованные лица и их требования: Пользователи веб-платформы, желающие сравнить статистические данные разных игроков для анализа.

Предусловие: Пользователь добавил 2 или более игроков для сравнения.

Постусловие: Пользователь получил сравнительный анализ статистики выбранных игроков.

Основной успешный сценарий:

1. Пользователь переходит в раздел сравнения статистики игроков.
2. Система предоставляет сравнительный отчет статистических данных для выбранных игроков.

2.3.2.5 Вариант использования «Просмотр информации о турнире»

Заинтересованные лица и их требования: Пользователи веб-платформы, желающие просмотреть информацию определенного турнира.

Предусловие: Пользователь загружает главную страницу сайта.

Постусловие: Пользователь просматривает информацию о выбранном турнире.

Основной успешный сценарий:

1. Пользователь заходит на главную страницу.
2. Пользователь выбирает раздел со списком турниров.
3. Пользователь фильтрует список всех турниров.
4. Пользователь выбирает интересующий турнир.
5. Пользователь просматривает информацию о выбранном турнире.

2.3.2.6 Вариант использования «Просмотр информации о матче»

Заинтересованные лица и их требования: Пользователи веб-платформы, желающие просмотреть информацию определенного матча.

Предусловие: Пользователь загружает главную страницу сайта.

Постусловие: Пользователь просматривает информацию о выбранном матче.

Основной успешный сценарий:

1. Пользователь заходит на главную страницу.
2. Пользователь выбирает раздел со всеми матчами.
3. Пользователь выбирает интересующий матч.
4. Пользователь просматривает информацию о выбранном матче.

2.3.2.7 Вариант использования «Просмотр статистики матча»

Заинтересованные лица и их требования: Пользователи веб-платформы, желающие просмотреть и анализировать статистику выбранного прошедшего матча.

Предусловие: Пользователь находится на странице с прошедшим матчем.

Постусловие: Пользователь просмотрел и проанализировал статистику матча.

Основной успешный сценарий:

1. Пользователь на странице матча отображает статистику матча.
2. Система отображает детальную статистику выбранного матча.

2.3.2.8 Вариант использования «Отображение прогноза на матч»

Заинтересованные лица и их требования: Пользователи веб-платформы, желающие просмотреть прогноз нейронной сети на будущий матч.

Предусловие: Пользователь находится на странице с будущим матчем.

Постусловие: Пользователь просмотрел прогноз на матча.

Основной успешный сценарий:

1. Пользователь на странице матча отображает прогноз на данный матч.
2. Система отображает прогноз на выбранный матч.

2.3.2.9 Вариант использования «Просмотр информации команды»

Заинтересованные лица и их требования: Пользователи веб-платформы, которые хотят получить детальную информацию о выбранной команде.

Предусловие: Пользователь загружает главную страницу сайта.

Постусловие: Пользователь просматривает необходимую информацию о команде.

Основной успешный сценарий:

1. Пользователь заходит на главную страницу.
2. Пользователь выбирает раздел с информацией о командах.
3. Пользователь ищет интересующую команду.
4. Система отображает страницу с информацией о команде.

2.3.2.10 Вариант использования «Просмотр рейтинга команд»

Заинтересованные лица и их требования: Пользователи веб-платформы, которые хотят просмотреть рейтинг лучших команд на данный момент.

Предусловие: Пользователь загружает главную страницу сайта.

Постусловие: Пользователь просматривает рейтинговую таблицу команд.

Основной успешный сценарий:

1. Пользователь заходит на главную страницу.
2. Пользователь выбирает раздел с рейтинговой таблицей команд.
3. Пользователь просматривает рейтинговую таблицу команд.

2.3.2.11 Вариант использования «Добавление команды для сравнения»

Заинтересованные лица и их требования: Пользователи веб-платформы, которые хотят добавить команду для сравнения ее статистики среди других команд.

Предусловие: Пользователь находится на странице с рейтингом команд.

Постусловие: Пользователь добавляет команду для будущего сравнения.

Основной успешный сценарий:

1. Пользователь наводит курсор на команду, которую хочет добавить для сравнения.
2. Пользователь нажимает кнопку ”Сравнить”.
3. Команда добавляется в список для сравнения.

2.3.2.12 Вариант использования «Сравнение статистики команд»

Заинтересованные лица и их требования: Пользователи веб-платформы, желающие сравнить статистику команд для анализа.

Предусловие: Пользователь добавил 2 или более команд для сравнения.

Постусловие: Пользователь получил сравнительный анализ статистики выбранных команд.

Основной успешный сценарий:

1. Пользователь переходит в раздел сравнения статистики команд.
2. Система предоставляет сравнительный отчет статистических данных для выбранных команд.

2.3.3 Требования пользователя к интерфейсу веб-платформы

В веб-платформе должны присутствовать следующие графические интерфейсы взаимодействия с пользователем:

1. Главная страница с актуальной информацией о текущих киберспортивных турнирах и меню навигации среди доступных функций.

2. Страница списка турниров с группировкой на прошедшие, текущие и будущие.

3. Страница турнира с отображением информации, участвующих командах и предстоящих матчах.

4. Страница матча с отображением информации, команд, прогнозе на данный матч и сводной статистики для данных команд.

5. Страница с рейтингом лучших команд.

6. Страница с рейтингом лучших игроков.

7. Модуль интерактивных диаграмм и графики для детализации статистики игроков и команд.

8. Страница команды с основной информацией, местом в рейтинге, последними матчами и статистикой команды.

9. Страница игрока с основной информацией, местом в рейтинге, последними матчами и статистикой игрока.

10. Страница сравнения метрик между выбранными игроками или командами.

Все страницы должны иметь адаптивную верстку для взаимодействия с платформой с разных устройств. Каждая страница должны иметь основные принципы UI/UX дизайна. Во время загрузки страницы должен быть реализован каркасный экран загрузки, с анимированным отображением скелета сайта до загрузки всех остальных данных с сервера. Загрузочный экран метрик будет сопровождаться индикатором прогресса обработки данных. Интерфейс результатов анализа должен позволять легко переключаться между различными видами статистики и визуализаций.

2.4 Нефункциональные требования к программной системе

2.4.1 Требования к архитектуре

Веб-платформа должна быть выполнена в микросервисной архитектуре. Между собой микросервисы должны общаться через gRPC. Каждый микросервис будет упакован в свой Docker-контейнер, что облегчит развертывание, масштабирование и обеспечение изоляции зависимостей. Архитектура должна поддерживать автоматическое масштабирование и самовосстановление, а также обеспечивать высокую доступность и отказоустойчивость.

2.4.2 Требования к надежности

При использовании веб-платформы должна быть обеспечена стабильная работа серверов - более 99% от всего времени работы, регулярное создание резервных копий данных, защита от несанкционированного доступа клиентов к неконтролируемой выгрузке статистики из базы данных в обход API.

Платформа должна обрабатывать и выдерживать высокую нагрузку - до 100 запросов в секунду.

2.4.3 Требования к программному обеспечению

Для реализации frontend части веб-платформы должен быть использован язык JavaScript и фреймворк React.

Для реализации backend части веб-платформы должны использоваться следующие языки программирования:

- C# - для разработки API и модуля получения новых турниров.
- Goland - для разработки анализатора .dem файлов.
- Python - для разработки нейронной сети для прогнозов.

2.4.4 Требования к аппаратному обеспечению

Для открытия веб-платформы потребуется устройство с доступом в сеть интернет, а так же браузер Google Chrome, Mozilla Firefox или Microsoft Edge с поддержкой выполнения JavaScript.

Для работы backend части веб-платформы необходим облачный сервер на операционной системе Linux с установленным Docker для развертывания всех сервисов. Так же дисковое пространство не менее 5 Гб, свободная оперативная память в размере не менее 6 Гб, видеокарта с не менее 1024 Мб видеопамяти.

2.4.5 Требования к оформлению документации

Разработка программной документации и программного изделия должна производиться согласно ГОСТ 19.102-77 и ГОСТ 34.601-90. Единая система программной документации.

3 Технический проект

3.1 Общие сведения о программной системе

Необходимо спроектировать и разработать веб-платформу для анализа и визуализации статистических данных киберспортивной игры Counter-Strike 2.

Разрабатываемая программная система предназначена для предоставления комплексного инструмента анализа данных, который позволит игрокам, тренерам и аналитикам изучать подробную статистику матчей, улучшая таким образом свои стратегии и игровые навыки. Платформа предоставит интерактивный пользовательский интерфейс для визуализации данных, позволяя пользователям глубоко погружаться в аналитику матчей.

Основной принцип работы системы заключается в обработке .dem файлов матчей CS2 для извлечения детальных игровых метрик и последующем представлении их через графики, диаграммы и интерактивные отчеты. Пользователи смогут просматривать прогнозы на матч и статистику, такую как точность стрельбы, K/D Ratio, использование гранат и прочее, чтобы оценить силу команды и принять обоснованные решения.

Одним из ключевых компонентов программной системы является база данных для хранения извлеченной статистики матчей и команд. Кроме того, будет реализован функционал машинного обучения для предоставления прогнозов исходов матчей, основанных на исторических данных.

Целью разработки данной программной системы является создание эффективного инструмента анализа данных в киберспорте, который поможет повысить конкурентоспособность игроков и команд на международной арене, способствуя развитию киберспорта как спортивной дисциплины.

3.2 Проектирование архитектуры программной системы

3.2.1 Выбор архитектурного стиля и паттернов проектирования

Для разработки веб-платформы анализа и визуализации статистических данных CS2 был выбран микросервисный подход. Такой стиль архитектуры подразумевает разбиение функционала системы на отдельные сервисы, каждый из которых отвечает за свою узкоспециализированную задачу. Это обеспечивает гибкость в развертывании и масштабировании, упрощает поддержку и обновление компонентов системы без необходимости внесения изменений во все сервисы сразу, а только в конкретный.

В качестве центрального входа в систему будет использоваться **API Gateway Pattern**, что позволит унифицировать обработку входящих запросов, распределение нагрузки, обеспечение безопасности и предоставление единой точки входа для клиентских приложений.

Для взаимодействия с клиентами будет использоваться **REST API**, что гарантирует легкость интеграции и стандартизированное общение между сервисами. REST API идеально подходит для микросервисной архитектуры за счет своей бесшовной и эффективной коммуникации. А так же возможностью предоставления внешнего взаимодействия с сервером.

Все данные, передаваемые между клиентом и сервером, будут защищены с использованием **HTTPS**, обеспечивая конфиденциальность, целостность передачи данных и защиту от муждуузловых атак.

Для запросов к данным со стороны клиента будет применяться протокол **OData**, который позволяет стандартизировать выборку, фильтрацию и пагинацию данных через HTTP запросы, обеспечивая гибкость и удобство в интеграции с различными клиентскими приложениями.

Для повышения производительности и снижения нагрузки на базу данных будет использоваться подход **Cache-Aside** в сочетании с **Redis**. Это позволит хранить часто запрашиваемые данные в быстром кеше, что существенно ускорит время их доставки конечному пользователю.

Для асинхронного обмена сообщениями между различными микросервисами в системе будет применяться **RabbitMQ**. Этот брокер сообщений позволит надежно обрабатывать задачи в фоновом режиме, улучшая производительность и масштабируемость системы, обеспечивая отказоустойчивость и балансировку нагрузки при высоких объемах обмена данными.

Данная архитектура обеспечивает всей системе необходимую гибкость для развития и масштабирования, а также повышает надежность и отказоустойчивость приложения при больших нагрузках.

Архитектура всей системы представлена на рисунке 3.1.

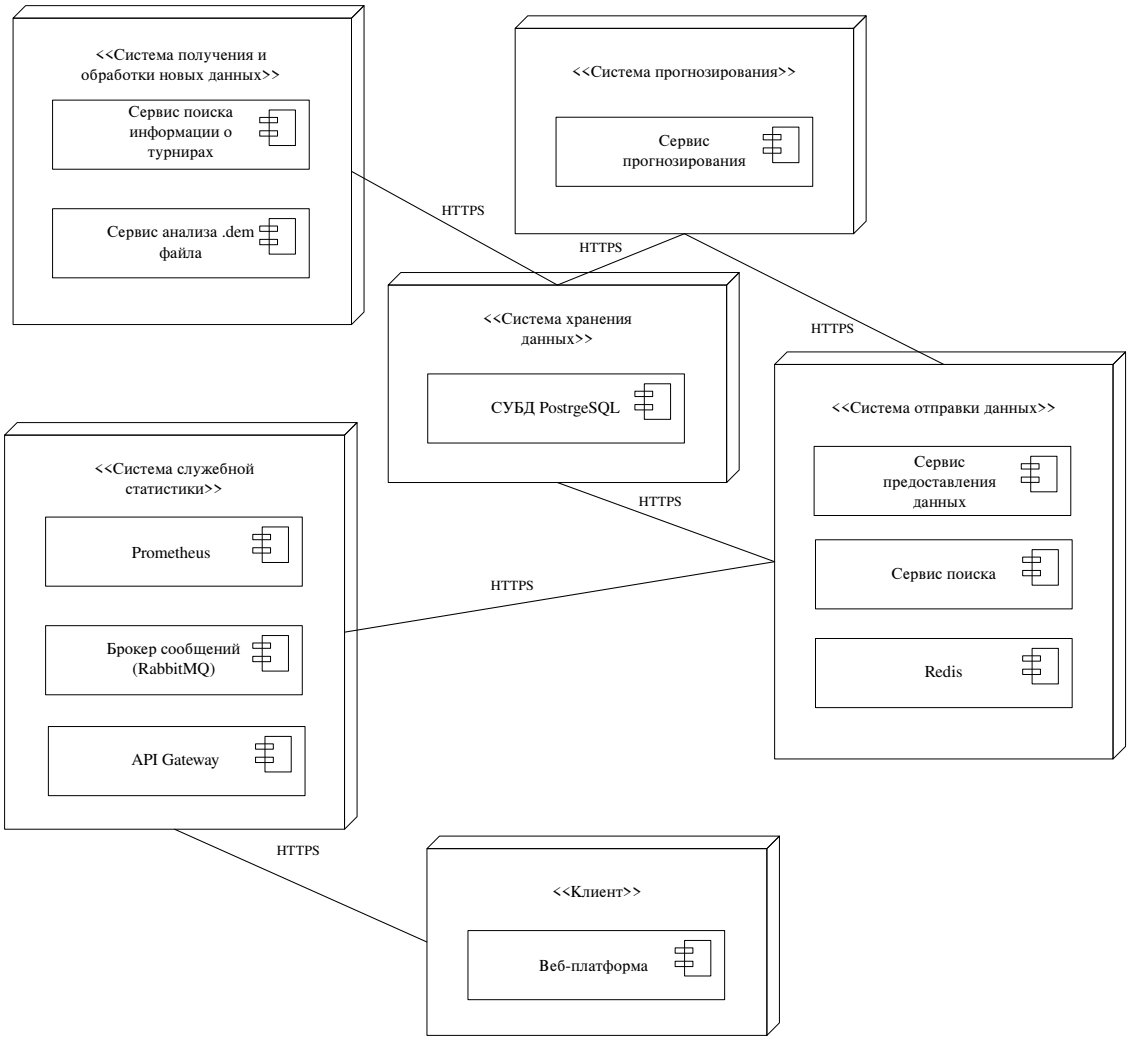

Рисунок 3.1 – Архитектура программной системы

3.2.2 Структура базы данных

Основной базой данных для хранения данных в системе будет выступать **PostgreSQL**. Эта СУБД была выбрана за её надежность, масштабируемость и поддержку сложных запросов, что является критически важным для аналитических запросов, связанных с большими объемами данных. PostgreSQL будет использоваться для хранения всех основных данных системы в SQL формате.

Redis будет использоваться как вспомогательная система для кеширования часто запрашиваемых данных. Это позволит значительно ускорить время отклика системы, уменьшить нагрузку на основную базу данных и предоставить пользователям актуальную информацию с минимальной задержкой. Redis должна использоваться для кеширования данных в формате ключ-значение, где ключом будет выступать хеш объекта, а значением сам данный объект.

3.2.3 Описание микросервисов

Веб-платформа будет включать в себя следующие микросервисы:

- **API Gateway:** Является точкой входа в систему, маршрутизирует запросы к соответствующим сервисам. Связан со всеми сервисами.

- **Сервис поиска информации о турнирах:** Отвечает за предоставление актуальных данных о киберспортивных турнирах. Сервис взаимодействует с базой данных и API сторонних платформ для обновления и поддержания в актуальном состоянии информации о предстоящих и прошедших турнирах.

- **Сервис анализа .dem файла:** Загружает и анализирует загруженные .dem файлы, извлекает из матча основные статистические данные для добавления их в базу данных и обработки другими сервисами.

- **Сервис предоставления данных:** Управляет предоставлением данных для разных разделов. Сервис взаимодействует с базой данных для полу-

чения, обработки и отправки информационных и статистических данных в нужном формате.

- **Сервис поиска:** Реализует функциональность поиска по всем сегментам данных платформы. Интегрирован с другими сервисами и предоставляет пользователю удобный поиск по игрокам, командам и турнирам.

- **Сервис прогнозирования:** Использует машинное обучение и алгоритмы прогнозирования для предсказания исходов матчей. Анализирует исторические данные и текущие тенденции для генерации прогнозов.

- **Брокер сообщений (RabbitMQ):** Обеспечивает надежную передачу сообщений между сервисами, поддерживает распределенные транзакции и асинхронное взаимодействие в микросервисной архитектуре.

- **Prometheus:** Используется для мониторинга работы сервисов, сбора метрик и уведомления о проблемах, позволяя оперативно реагировать на сбои и изменения в работе системы.

3.2.4 Планирование докеризации и оркестрации сервисов

Для достижения высокой отказоустойчивости, удобства в развертывании и эффективного масштабирования, развертывание каждого микросервиса веб-платформы будет выполнено в контейнере Docker. Каждый контейнер в себе будет включать требуемое ядро системы, используемые библиотеки и настройки окружения для запуска данного микросервиса. Это позволит обеспечить изоляцию зависимостей между частями платформы и согласованность окружений независимо от инфраструктуры развертывания.

Оркестрация контейнеров будет производиться с помощью Kubernetes, что гарантирует автоматизацию развертывания, масштабирования и управления приложениями контейнеров. Kubernetes обеспечивает балансировку нагрузки, самовосстановление сервисов при их ошибках, автоматическое распределение ресурсов и управление конфигурацией.

Вся инфраструктура веб-платформы будет развернута в облаке Yandex Cloud, что обеспечит высокую доступность сервисов и возможности простой интеграции с другими облачными сервисами.

3.2.5 Описание REST API микросервисов

Таблица 3.1 – Сервис API Gateway

HTTP-метод	Описание	Входные параметры	Пример JSON ответа
1	2	3	4
POST /api/	Маршрутизация запросов	{”service”: ”string ”endpoint”: ”string ”data”: ”object”}	{”status”: ”success ”data”: ”object”}
GET /api/	Получение данных от сервисов	{”service”: ”string ”endpoint”: ”string ”params”: ”object”}	{”status”: ”success ”data”: ”object”}

Таблица 3.2 – Сервис поиска информации о турнирах

HTTP-метод	Описание	Входные параметры	Пример JSON ответа
1	2	3	4
POST api/BackgroundServiceController/start-team-update	Запуск фонового получения данных команд	Нет	{”result”: ”string”}
POST api/BackgroundServiceController/start-tournament-update	Запуск фонового получения данных турниров и стадий	Нет	{”result”: ”string”}

Продолжение таблицы 3.2

1	2	3	4
POST api/Background ServiceController /start-match-update	Запуск фоно-вого получения данных матчей	Нет	{”result”: ”string”}
POST api/Background ServiceController /start-stats-update	Запуск фонового получения дан-ных статистики	Нет	{”result”: ”string”}
POST api/Background ServiceControlle /stop/:serviceName	Запуск фоно-вого получения данных команд	{”serviceName”: ”string”}	{”result”: ”string”}

Таблица 3.3 – Сервис анализа .dem файла

HTTP-метод	Описание	Входные параметры	Пример JSON ответа
1	2	3	4
POST api/analyze/upload	Загрузка .dem файла	{”file”: ”binary”}	{”status”: ”string”}
POST api/analyze/ analyze	Анализ загру-женного файла	{”tournament_id”: ”int ”stage_id”: ”int ”team1_id”: ”int ”team2_id”: ”int ”map_id”: ”int ”file_name”: ”string”}	{”status”: ”string”}

Таблица 3.4 – Сервис поиска

HTTP-метод	Описание	Входные параметры	Пример JSON ответа
1	2	3	4
GET api/search/	Поиск по запросу	{”query”: ”string”}	{”results”: [{”Players”: [object]}, {”Teams”: [object]}, {”Tournaments”: [object]}] }

Таблица 3.5 – Сервис прогнозирования

HTTP-метод	Описание	Входные параметры	Пример JSON ответа
1	2	3	4
POST api/prediction /predict	Генерация прогноза на матч	{”matchId”: ”int”}	{”prediction”: {”winProbability”: ”float”} }

Таблица 3.6 – Сервис предоставления данных

HTTP-метод	Описание	Входные параметры	Пример JSON ответа
1	2	3	4
GET api/data /tournament /current	Получение будущих турниров	{}	{”tournaments”: [”object”]}
GET api/data /tournament /finished	Получение предыдущих турниров	{}	{”tournaments”: [”object”]}
GET api/data /tournamen t/:slug	Получение определенного турнира	{”slug”: ”string”}	{”tournament”: ”object”}

Продолжение таблицы 3.6

1	2	3	4
GET api/data /match/finished	Получение предыдущих матчей	{}	{”matches”: [”object”]}
GET api/data /match/current	Получение будущих матчей	{}	{”matches”: [”object”]}
GET api/data /match/:slug	Получение конкретного матча	{”slug”: ”string”}	{”match”: ”object”}
GET api/data /match /:slug/stats	Получение статистики конкретного матча	{”slug”: ”string”}	{”statistics”: ”object”}
GET api/data /match/:slug /stats/:mapName	Получение статистики конкретной карты	{”slug”:”string” ”mapName”: ”string”}	{”statistics”: ”object”}
GET api/data /player/	Получение игроков	{}	{”players”: [”object”]}
GET api/data /team/	Получение команд	{}	{”teams”: [”object”]}
GET api/data /player/:slug	Получение отдельной команды	{”slug”: ”string”}	{”team”: ”object”}
GET api/data /team/:slug	Получение отдельного игрока	{”slug”: ”string”}	{”player”: ”object”}
GET api/data /compare /players	Сравнение двух игроков	{”player1Slug”: ”string” ”player2Slug”: ”string”}	{”comparison”: {”player1”: ”object ”player2”: ”object”}}
GET api/data /compare /teams	Сравнение двух команд	{”team1Slug”: ”string” ”team2Slug”: ”string”}	{”comparison”: {”team1”: ”object ”team2”: ”object”}}

3.3 Обоснование выбора технологий проектирования и программных средств

3.3.1 Выбор используемых технологий и языков программирования

3.3.1.1 API Gateway

Для реализации данного микросервиса должен быть выбран язык C# с библиотекой Ocelot для реализации шлюзов и перенаправлений запросов API.

3.3.1.2 Сервис поиска информации о турнирах

Для реализации данного микросервиса должен быть выбран язык C# с использованием библиотек AutoMapper - для реализации мапинга считываемых сущностей из открытых источников и преобразования их в сущности используемые внутри системы, Entity Framework Core - для реализации работы с базами данных, используя объектноориентированный подход, Newtonsoft.Json - для преобразования получаемых JSON файлов в сущности DTO.

3.3.1.3 Сервис поиска информации о турнирах

Для реализации данного микросервиса должен быть выбран язык Go, так как он предлагает высокую производительность и эффективность в обработке и анализе низкоуровневых данных, таких как бинарные файлы .dem записей матчей, благодаря своим встроенным средствам для параллельной обработки. Так же должна использоваться библиотека demoinfocs-golang - библиотека для разбора и анализа .dem файлов игры Counter-Strike 2.

3.3.1.4 Сервис предоставления данных

Для реализации данного микросервиса должен быть выбран язык C# с использованием библиотек Entity Framework Core - для реализации работы

с базами данных, AutoMapper - для реализации мапинга сущностей из базы данных в сущности DTO (Data Transfer Object) отправляемые на frontend часть веб-платформы.

3.3.1.5 Сервис поиска

Для реализации данного микросервиса должен быть выбран язык C# с использованием библиотек Entity Framework Core - для реализации работы с базами данных и библиотеки AutoMapper - для реализации мапинга сущностей из базы данных в сущности DTO (Data Transfer Object) отправляемые на frontend часть веб-платформы.

3.3.1.6 Сервис прогнозирования

Для реализации данного микросервиса должен быть выбран язык Python из-за его превосходной поддержки библиотек машинного обучения и аналитики данных. Для данного модуля должны использоваться библиотека sklearn - для подготовки датасета для обучения нейронной сети, tensorflow - для непосредственно создания нейронной сети и ее обучения.

3.3.1.7 Брокер сообщений (RabbitMQ)

Для реализации взаимодействия между микросервисами через брокер сообщений будет использоваться RabbitMQ. Для работы с RabbitMQ должен быть выбран язык C# с библиотекой RabbitMQ.Client, предоставляющей удобный API для взаимодействия с RabbitMQ.

3.3.1.8 Prometheus

Для мониторинга и алертинга инфраструктуры будет использоваться Prometheus. Он позволит собирать метрики с различных сервисов и узлов, а также устанавливать алерты на основе собранных данных. В микросервисах на C# для экспорта метрик в Prometheus должна использоваться библиотека prometheus-net.

3.3.1.9 React

React - это JavaScript библиотека, разработанная компанией Facebook, предназначенная для создания пользовательских интерфейсов веб-приложений. Она позволяет разрабатывать динамические и интерактивные веб-страницы, обладающие высокой производительностью и масштабируемостью.

Одной из ключевых особенностей React является использование компонентного подхода к созданию интерфейса. Приложение строится из небольших и независимых компонентов, каждый из которых отвечает за определенную часть пользовательского интерфейса. Это позволяет создавать чистый и упорядоченный код, легко поддерживаемый и расширяемый.

Ещё одним важным преимуществом React является использование виртуального DOM (Document Object Model). React создает виртуальное представление DOM в памяти, которое затем сравнивается с реальным DOM и обновляется только та часть, которая изменилась. Это позволяет увеличить производительность приложения и улучшить пользовательский опыт. React также предоставляет множество инструментов и библиотек для управления состоянием приложения, маршрутизации, тестирования и других задач. Экосистема React очень развита, что делает его популярным выбором для разработки веб-приложений.

Во время разработки веб-платформы должны быть реализованы следующие компоненты для создания полноценной структуры и выполнения поставленных требований в пункте 2.3 технического задания:

- **Компонент отображения статистики по всем картам отдельного игрока:** этот компонент предназначен для отображения статистики игры отдельного игрока по различным картам. Он должен включать среднюю оценку игрока на карте, количество сыгранных карт, и средние данные, такие как среднее число убийств и урона за раунд.

- **Таблица игроков команды в матче:** таблица, представляющая собой список всех игроков участвующих в матче с их ключевыми статисти-

ческими данными для данного матча. Это может включать K/D/A (убийства/смерти/ассисты), разницу между убийствами и смертями, рейтинг и форму игрока в данном матче, и другие данные.

- **Компонент вывода статистики команд за последние 6 месяцев на всех картах:** компонент, который показывает собранные данные о производительности команды на различных картах за последние полгода. Информация представлена в виде процента побед на отдельных картах, среднее число убийств и урона за раунд и отображение статуса победы в последних 5 играх на каждой карте.

- **Компонент сравнения статистики команд и игроков по основным метрикам:** данный компонент предоставляет сравнительный анализ между командами или игроками, позволяя пользователям видеть различия в ключевых статистических показателях. Может быть полезен для анализа формы команд перед матчами. В нем выбранные для сравнения команды или игроки представлены в виде таблицы со всеми метриками, которые попарно сравниваются и выделяются больший из них.

- **Блок формирования одинаковой структуры для всех используемых блоков в системе:** данный компонент является основой для стандартизации внешнего вида и структуры различных компонентов на сайте. Это помогает в обеспечении консистентности пользовательского интерфейса и улучшает общий пользовательский опыт.

- **Компонент отображения команды со списком игроков:** компонент для отображения команды и списка игроков в команде, включая их аватары, название команды и прочее. Данный компонент используется для предоставления обзора состава команды в матчах или турнирах.

3.3.2 Выбор программного обеспечения

Разработка веб-платформы для анализа и визуализации статистических данных игры Counter-Strike 2 требует продуманного подхода к выбору аппаратного обеспечения. Основные критерии выбора – масштабируемость,

надежность, и производительность системы. В качестве основы для реализации инфраструктуры выбраны современные технологии контейнеризации, оркестрации, управления очередями сообщений и мониторинга: Docker, Kubernetes, RabbitMQ, и Prometheus.

Важным аспектом является решение о запуске всей инфраструктуры в облачной среде, что обеспечивает высокую гибкость, масштабируемость и доступность ресурсов. Облачные сервера Yandex Cloud, на которых будет развернута система, будут работать под управлением операционной системы Linux, что гарантирует стабильность работы и широкие возможности для настройки системы.

3.3.3 Docker

Docker предоставляет легковесную и удобную платформу для создания, развертывания и управления контейнерами. Контейнеризация упрощает процесс разработки, тестирования и развертывания приложений, позволяя запускать приложения и их зависимости в изолированных средах. В контексте выбора аппаратного обеспечения это означает возможность оптимизации использования ресурсов и увеличение эффективности за счет развертывания на облачных серверах с поддержкой Docker.

На рисунке 3.2 представлена архитектура работы Docker.

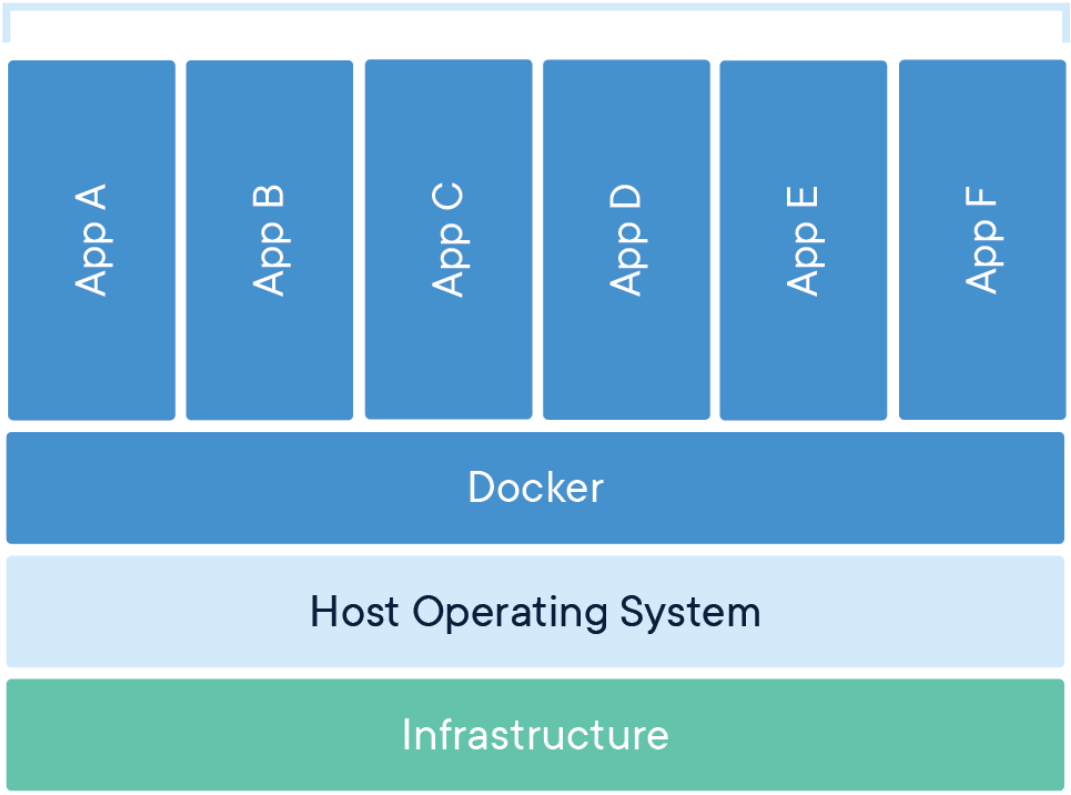

Рисунок 3.2 – Архитектура работы Docker

Данная архитектура состоит из следующих элементов:

1. Infrastructure (Инфраструктура): Это физические серверы или виртуальные машины, на которых развёрнуты контейнеризированные приложения. Инфраструктура предоставляет вычислительные ресурсы, такие как процессор, память, дисковое пространство и сеть.

2. Host Operating System (Хостовая операционная система): Это операционная система, установленная на инфраструктуре. Она управляет аппаратными ресурсами и предоставляет базовые функции для работы контейнеров.

3. Docker: Docker — это платформа для контейнеризации, которая позволяет разрабатывать, отправлять и запускать приложения в изолированных контейнерах. Docker использует технологии виртуализации на уровне операционной системы, такие как cgroups и namespaces, для изоляции контейнеров.

4. Containerized Applications (Контейнеризированные приложения): Это приложения, упакованные в контейнеры Docker. Контейнеры включают все необходимые зависимости, библиотеки и конфигурации для работы приложения, что обеспечивает их портативность и изолированность.

3.3.4 Redis

Redis — это база данных, размещаемая в памяти, которая используется, в основном, в роли кеша, находящегося перед другой, «настоящей» базой данных, вроде MySQL или PostgreSQL. Кеш, основанный на Redis, помогает улучшить производительность приложений. Он эффективно использует скорость работы с данными, характерную для памяти, и смягчает нагрузку центральной базы данных приложения, связанную с обработкой следующих данных:

- Данные, которые редко меняются, к которым часто обращается приложение.

- Данные, не относящиеся к критически важным, которые часто меняются.

Примеры таких данных могут включать в себя сессионные кеши или кеши данных, а так же содержимое панелей управления — вроде списков лидеров и отчётов, включающих в себя данные, агрегированные из разных источников.

Рассмотрим, как работает Redis в качестве кеша на основе архитектуры представленной на рисунке 3.3.

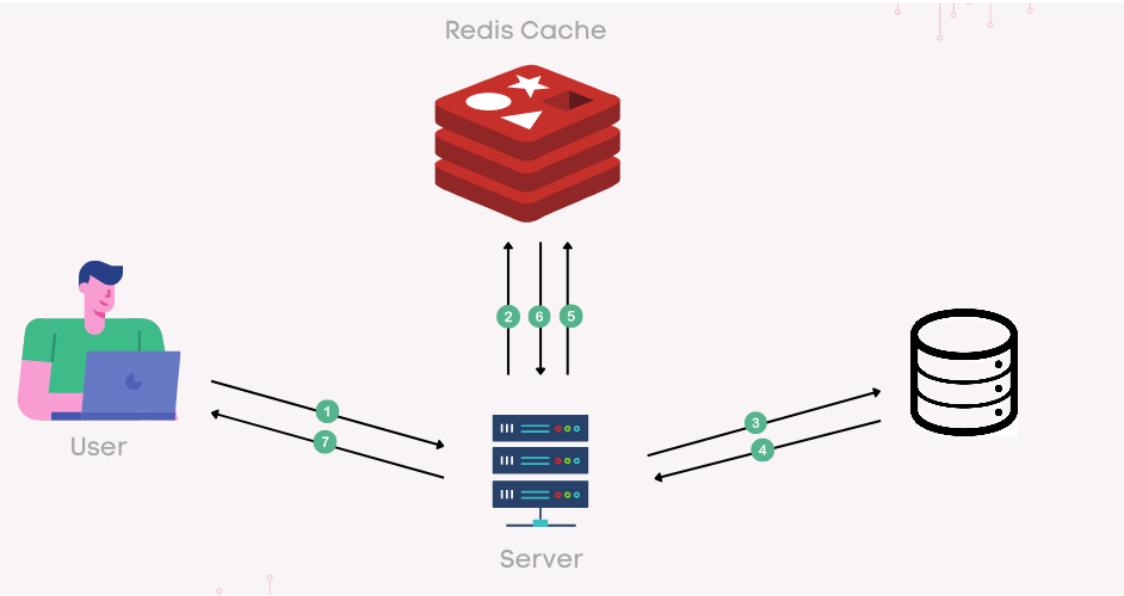

Рисунок 3.3 – Пример работы Redis

Пример работы при первом обращении к данным:

1. Пользователь делает запрос к серверу (Шаг 1).

2. Сервер проверяет наличие необходимых данных в Redis (Шаг 2).

3. Если данных в кэше нет, сервер отправляет запрос к основной базе данных (Шаг 3).

4. Основная база данных обрабатывает запрос и возвращает данные серверу (Шаг 4).

5. Получив данные от основной базы данных, сервер сохраняет их в Redis для последующих запросов (Шаги 5-6).

6. После сохранения данных в кеше они отправляются пользователю (Шаг 7).

Пример работы при обращении к уже сохраненным данным в Redis:

1. Пользователь делает запрос к серверу (Шаг 1).

2. Данные находятся в кеше и возвращаются серверу (Шаг 5).

3. Сервер отправляет данные пользователю (Шаг 7).

Благодаря использованию кеширования удается уменьшить скорость получения данных при их повторном вызове до x10 раз.

3.3.5 Kubernetes

Kubernetes является мощной системой для автоматизации развертывания, масштабирования и управления контейнеризированными приложениями. В архитектуре веб-платформы Kubernetes обеспечивает высокую доступность, автоматическое масштабирование и балансировку нагрузки между контейнерами. Для аппаратного обеспечения это подразумевает необходимость в выделенном или облачном сервере с достаточным количеством процессорного времени и оперативной памяти для поддержания кластера Kubernetes.

3.3.6 RabbitMQ

RabbitMQ, система управления очередями сообщений, играет ключевую роль в обеспечении асинхронной обработки данных и интеграции раз-

личных частей системы. Эффективное использование RabbitMQ позволяет распределить нагрузку, улучшить производительность приложения и обеспечить надежную обработку сообщений. Аппаратное обеспечение должно соответствовать требованиям по пропускной способности и скорости обработки сообщений, что особенно важно при больших объемах данных. Сервер на котором будет находится данный сервис должен быть максимально отказоустойчивым, для исключения лавинного выхода из строя микросервисов.

На рисунке 3.4 представлена работа RabbitMQ и системы управления очередями сообщений (message broker) в процессе передачи сообщений от производителя (publisher) к потребителю (consumer). 3.4.

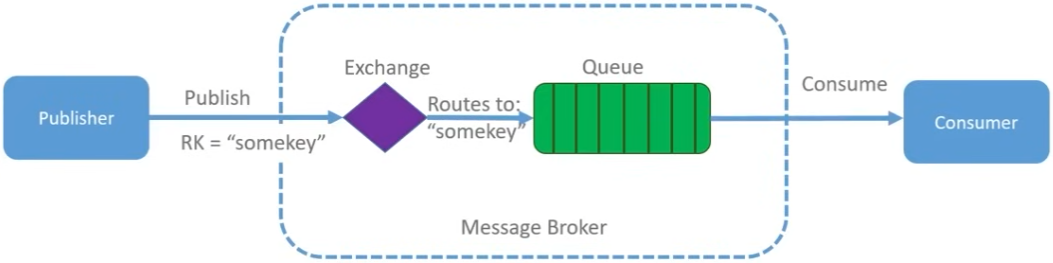

Рисунок 3.4 – Пример работы RabbitMQ

Схема состоит из следующих элементов:

1. Publisher (Производитель): это модуль, который отправляет сообщения. В данном примере производитель публикует сообщение с ключом маршрутизации (routing key), равным ”somekey”.

2. Exchange (Обменник): обменник принимает сообщения от производителей и маршрутизирует их в очереди на основе ключа маршрутизации. Обменники могут быть разных типов (direct, fanout, topic, headers), что определяет правила маршрутизации сообщений. В данном примере обменник получает сообщение с ключом маршрутизации ”somekey”и определяет, в какую очередь его направить.

3. Queue (Очередь): очередь — это место, где сообщения хранятся до тех пор, пока они не будут получены потребителем. Очереди могут быть долговременными (persistent), чтобы сохраняться при перезапуске сервера

RabbitMQ, или временными (transient), которые удаляются при перезапуске. В данном примере сообщение с ключом маршрутизации ”somekey”попадает в определенную очередь.

4. Consumer (Потребитель): потребитель получает сообщения из очереди и обрабатывает их. Потребители могут подтверждать получение сообщений (acknowledgements), чтобы уведомить RabbitMQ, что сообщение было успешно обработано. В данном примере потребитель извлекает сообщение из очереди и выполняет необходимые действия.

3.3.7 Prometheus

Prometheus — система мониторинга и оповещения, которая позволяет собирать и анализировать метрики в реальном времени. Использование Prometheus обеспечивает возможность наблюдения за производительностью системы, оптимизации ресурсов и быстрого реагирования на возникающие проблемы. Аппаратное обеспечение должно обладать достаточным объемом хранилища для сбора и хранения данных метрик, а также процессорной мощностью для их обработки.

3.4 Проектирование пользовательского интерфейса программной системы

3.4.1 Макеты пользовательского интерфейса

На основании требований к пользовательскому интерфейсу, представленных в пункте 2.3 технического задания, был разработан графический интерфейс мобильного приложения, используя React с использованием библиотеки Material UI. Разработанный интерфейс ориентирован на обеспечение легкости в использовании и удобного визуального представления статистических данных.

На рисунке 3.5 представлен макет страницы команды.

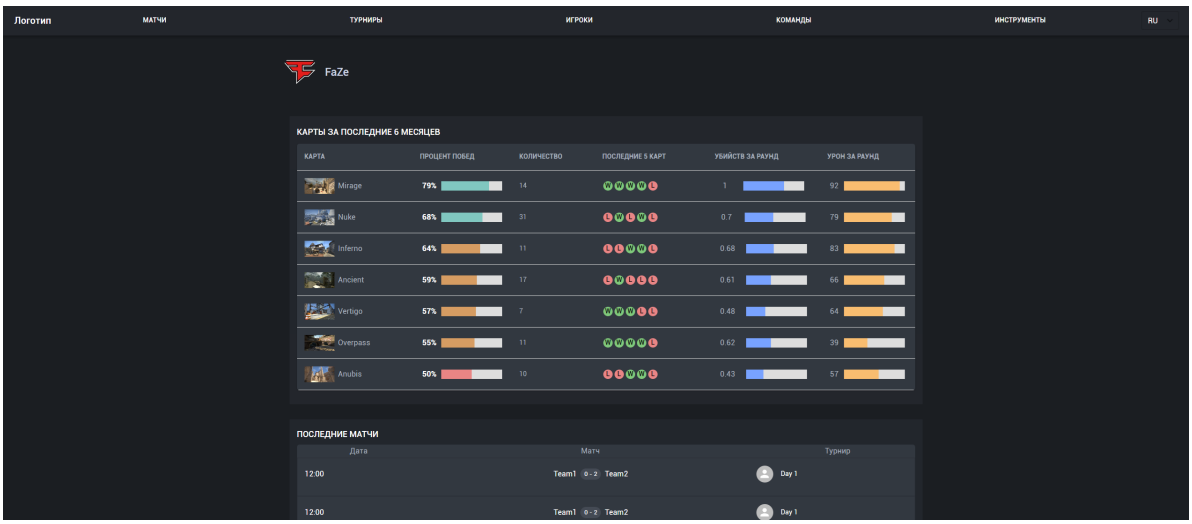

Рисунок 3.5 – Макет страницы команды

Макет содержит следующие элементы:

- Название команды.
- Логотип команды.
- Статистика команды по картам.
- Список матчей команды.
- Состав команды.

На рисунке 3.6 представлен макет страницы игрока.

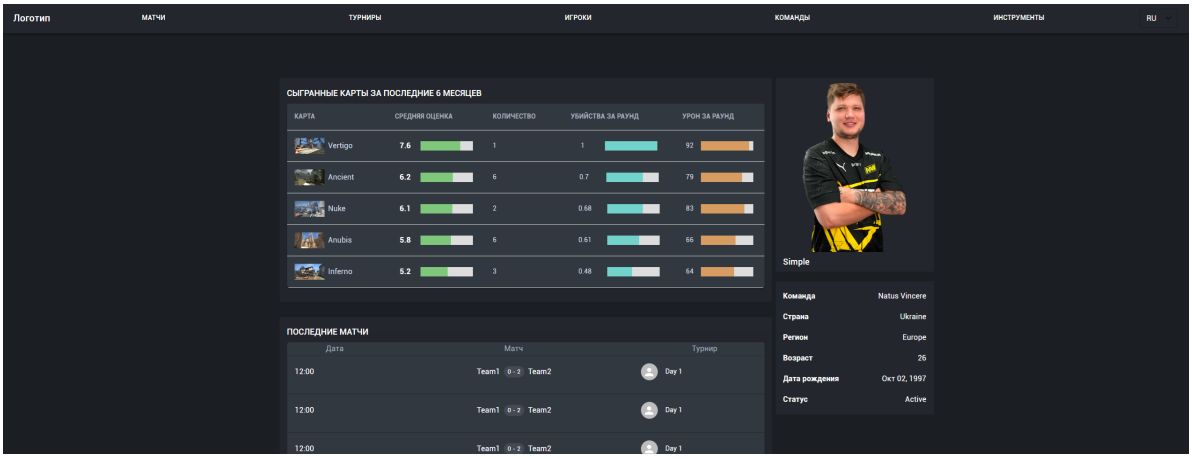

Рисунок 3.6 – Макет страницы игрока

Макет содержит следующие элементы:

- Никнейм игрока.
- Фото игрока.
- Статистика игрока по картам.

- Список матчей данного игрока.
- Основную информацию о игроке.
- Прочую статистику игрока.

На рисунке 3.7 представлен макет страницы матча.

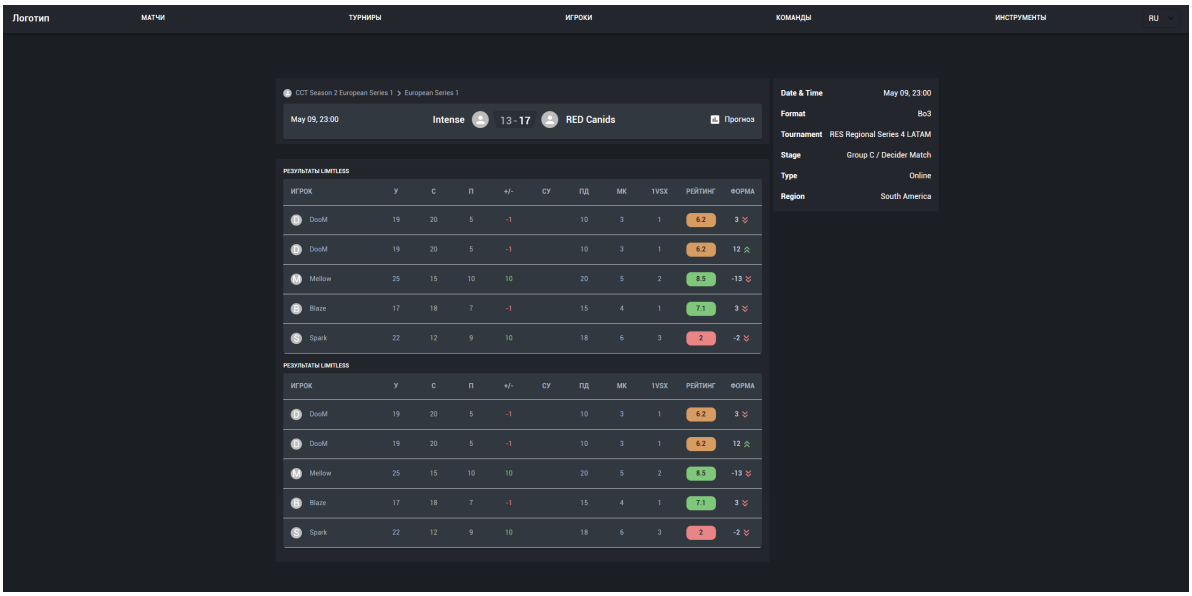

Рисунок 3.7 – Макет страницы матча

Макет содержит следующие элементы:

- Турнир и стадию для данного матча.
- Счет и названия команд матча.
- Основную информацию матча.
- Статистику игроков в данном матче по командам.

На рисунке 3.8 представлен макет страницы турнира.

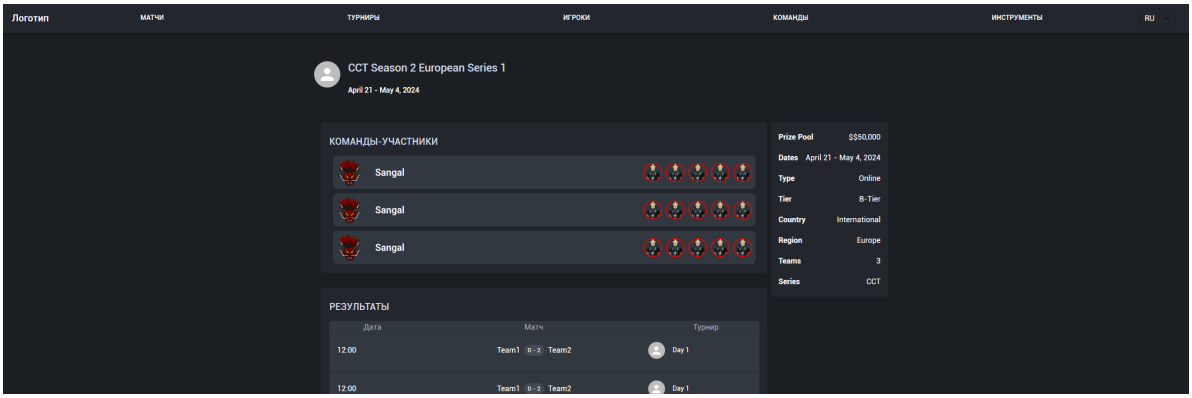

Рисунок 3.8 – Макет страницы турнира

Макет содержит следующие элементы:

- Название турнира, логотип его дата.
- Список участвующих команд.
- Основную информацию турнира.
- Список матчей данного турнира.

3.4.2 Компоненты пользовательского интерфейса

Для создания фронтенд части веб-платформы полноценной структуры созданы следующие компоненты:

3.4.2.1 MiddleContent

Компонент MiddleContent используется для отображения основной информации и содержимого страницы. Он включает в себя изменяемые заголовок, подзаголовок, аватар и дополнительное содержимое с возможностью указания ширины конечного блока.

На рисунке 3.9 представлен пример отображения компонента MiddleContent:

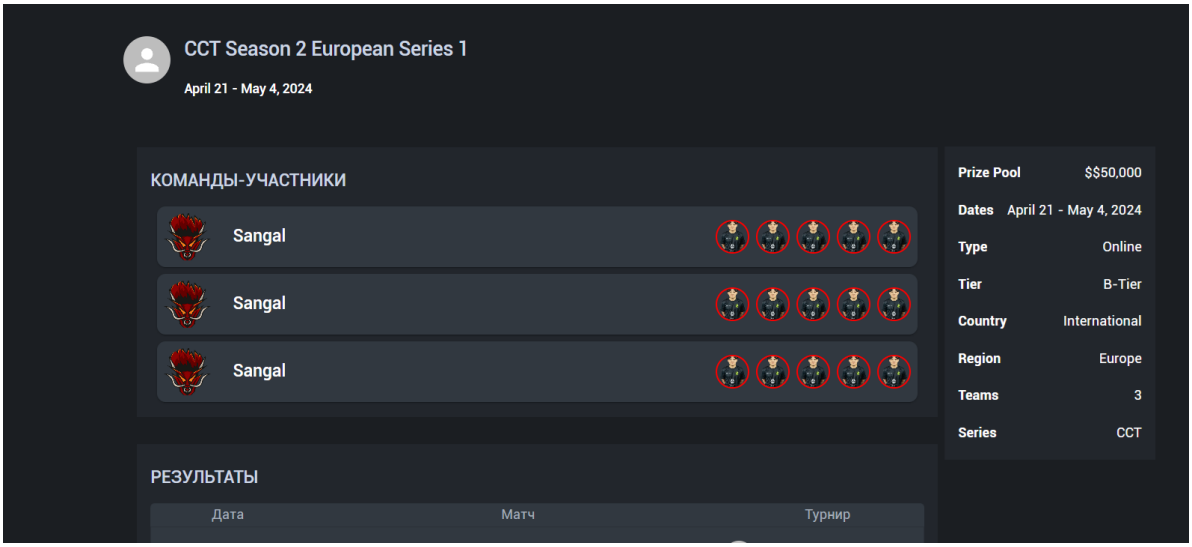

Рисунок 3.9 – Пример отображения компонента MiddleContent

3.4.2.2 MapStatsTable

Компонент MapStatsTable отображает статистику по картам за последние 6 месяцев. Данный компонент представляет данные в виде таблицы, включающей следующие столбцы:

- **Карта**: Название карты и изображение карты.

- **Средняя оценка**: Средний рейтинг карты, выраженный в числовом значении и визуально представлен в виде прогресс-бара.

- **Количество**: Количество сыгранных карт.

- **Убийства за раунд**: Среднее количество убийств за раунд, визуально представленное прогресс-баром.

- **Урон за раунд**: Средний урон за раунд, визуально представлен в виде прогресс-бара.

На рисунке 3.10 представлен пример отображения компонента MapStatsTable:

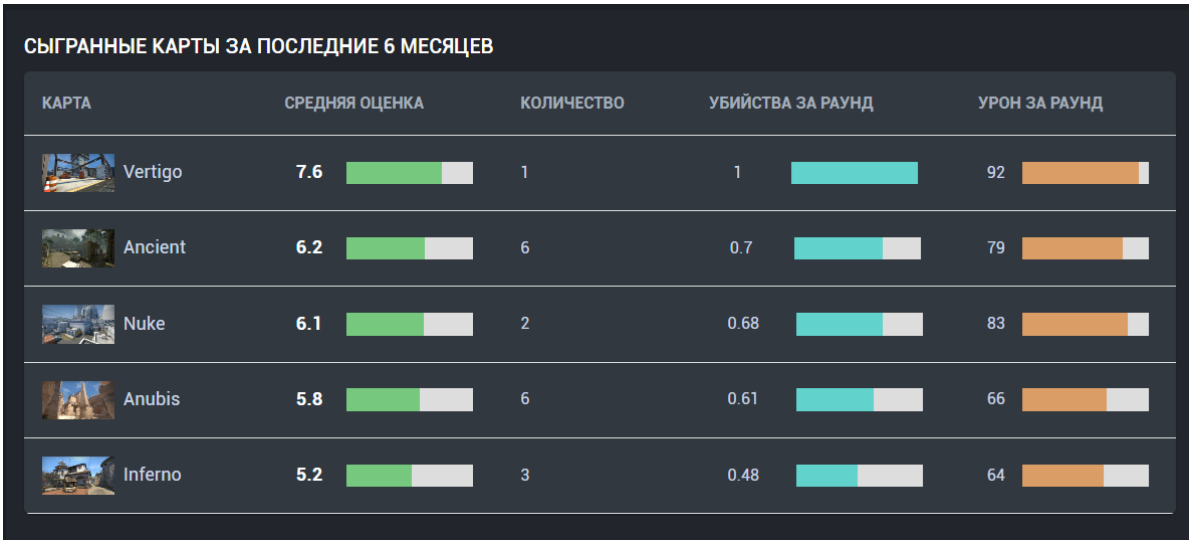

Рисунок 3.10 – Пример отображения компонента MapStatsTable

Компонент включает в себя:

- **Таблица со статистикой**: Таблица отображает информацию по картам, включая их название, изображение, средний рейтинг, количество игр, KPR (киллы за раунд) и DPR (урон за раунд).

- **Прогресс-бары**: Прогресс-бары используются для визуализации средних значений рейтинга, KPR и DPR, обеспечивая наглядное представление данных.

Компонент реализован с использованием библиотеки Material UI и включает стиль, обеспечивающий удобство использования и хорошую визуализацию данных.

3.4.2.3 PlayerStats

Компонент PlayerStats отображает статистику игроков в матче. Данный компонент представляет данные в виде таблицы, включающей следующие столбцы:

- **Игрок**: Имя игрока и его аватар.
- **У**: Количество убийств.
- **С**: Количество смертей.
- **П**: Количество помощи (ассистов).
- **+/-**: Разница между убийствами и смертями.
- **СУ**: Среднее количество убийств за раунд.
- **ПД**: Позиция защиты.
- **МК**: Количество матчей, в которых игрок стал MVP.
- **1VSX**: Количество выигранных ситуаций 1 на X.
- **Рейтинг**: Оценка эффективности игрока.
- **Форма**: Изменение рейтинга формы игрока.

На рисунке 3.11 представлен пример отображения компонента PlayerStats:

РЕЗУЛЬТАТЫ LIMITLESS

ИГРОК	У	С	П	+/-	СУ	ПД	МК	1VSX	РЕЙТИНГ	ФОРМА
DooM	19	20	5	-1		10	3	1	6.2	3
DooM	19	20	5	-1		10	3	1	6.2	12
Mellow	25	15	10	10		20	5	2	8.5	-13
Blaze	17	18	7	-1		15	4	1	7.1	3
Spark	22	12	9	10		18	6	3	2	-2

Рисунок 3.11 – Пример отображения компонента PlayerStats

Компонент включает в себя:

- **Таблица со статистикой**: Таблица отображает информацию по игрокам, включая их имя, аватар, количество убийств, смертей, ассистов, рейтинг и форму.

- **Индикаторы изменений**: Используются индикаторы для отображения изменений в форме игрока.

Компонент реализован с использованием библиотеки Material UI.

3.4.2.4 MapPulStatsTable

Компонент MapPulStatsTable отображает подробную статистику по картам, включая процент побед, количество сыгранных карт и другие метрики. Данный компонент представляет данные в виде таблицы, включающей следующие столбцы:

- **Карта**: Название карты и изображение карты.

- **Процент побед**: Процент побед на данной карте, визуально представлен в виде прогресс-бара.

- **Количество**: Количество сыгранных карт.

- **Последние 5 карт**: Результаты последних пяти игр на данной карте.

- **Убийства за раунд**: Среднее количество убийств за раунд, визуально представленное прогресс-баром.

- **Урон за раунд**: Средний урон за раунд, визуально представлен в виде прогресс-бара.

На рисунке 3.12 представлен пример отображения компонента MapPulStatsTable:

Рисунок 3.12 – Пример отображения компонента MapPulStatsTable

Компонент включает в себя:

- **Таблица со статистикой**: Таблица отображает информацию по картам, включая их название, изображение, процент побед, количество игр, последние пять результатов игр, KPR (киллы за раунд) и DPR (урон за раунд).

- **Прогресс-бары**: Прогресс-бары используются для визуализации значений процента побед, KPR и DPR, обеспечивая наглядное представление данных.

Компонент реализован с использованием библиотеки Material UI и включает стиль, обеспечивающий удобство использования и хорошую визуализацию данных.

4 Рабочий проект

4.1 Описание сущностей веб-платформы

4.1.1 Описание сущностей базы данных

В данном разделе представлены основные сущности базы данных, используемые во всей системе для представления различных данных, связанных с киберспортивными турнирами, матчами, командами и игроками. Эти сущности описываются с указанием их атрибутов, типа данных и ключевых свойств.

Таблица 4.1 – Свойства класса BaseModel

Свойство	Тип	Обязательное	Описание
1	2	3	4
CreatedAt	DateTimeOffset	true	Дата и время создания записи
UpdatedAt	DateTimeOffset	true	Дата и время обновления записи

Таблица 4.2 – Свойства класса Country

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор страны
CountryName	string	false	Название страны
Code	string	true	Код страны

Таблица 4.3 – Свойства класса TeamName

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор альтернативного названия команды
Name	string	true	Альтернативное название команды
TeamId	int	true	Идентификатор команды
Team	Team	false	Связь с основной сущностью команды

Таблица 4.4 – Свойства класса Stage

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор этапа
Status	Tournament StatusEnum	true	Статус турнира
Title	string	true	Название этапа
TournamentId	int	true	Идентификатор турнира
Tournament	Tournament	false	Связь с основной сущностью турнира
Teams	List<Team>	false	Список команд, участвующих в этапе

Продолжение таблицы 4.4

1	2	3	4
StageType	StageTypeEnum	true	Тип этапа
Matches	List<Match>	false	Список матчей на этапе
Description	string	false	Описание этапа

Таблица 4.5 – Свойства класса Player

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор игрока
Nickname	string	true	Никнейм игрока
FirstName	string	false	Имя игрока
LastName	string	false	Фамилия игрока
Slug	string	true	Уникальный слаг для игрока
ImageData	string	false	Изображение игрока в формате Base64
ImageUrl	string	false	URL изображения игрока
TeamId	int	false	Идентификатор команды
Team	Team	false	Связь с основной сущностью команды
CountryId	int	false	Идентификатор страны
Country	Country	false	Связь с основной сущностью страны

Таблица 4.6 – Свойства класса Team

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор команды
Slug	string	true	Уникальный слаг для команды
Name	string	true	Название команды
Acronym	string	false	Аббревиатура команды
AlternativeNames	List<TeamName>	false	Список альтернативных названий команды
CountryId	int	false	Идентификатор страны
Country	Country	false	Связь с основной сущностью страны
IconUrl	string	false	URL иконки команды
Players	List<Player>	false	Список игроков команды
WebsiteUrl	string	false	URL веб-сайта команды
YoutubeUrl	string	false	URL YouTube канала команды
Tournaments	List<Tournament>	false	Список турниров, в которых участвовала команда

Таблица 4.7 – Свойства класса Tournament

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор турнира
Name	string	true	Название турнира
Slug	string	true	Уникальный слаг для турнира
Status	Tournament StatusEnum	true	Статус турнира
Tier	TierEnum	true	Уровень турнира
StartDate	DateTimeOffset	true	Дата начала турнира
EndDate	DateTimeOffset	true	Дата окончания турнира
Prize	int	true	Призовой фонд турнира
EventType	EventTypeEnum	false	Тип события
Description	string	false	Описание турнира
Discipline	DisciplineEnum	false	Дисциплина турнира
ImageUrl	string	false	URL изображения турнира
Country	Country	false	Страна проведения турнира
Teams	List<Team>	false	Список команд, участвующих в турнире
Stages	List<Stage>	false	Список этапов турнира

Таблица 4.8 – Свойства класса Map

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор карты
MatchID	int	true	Идентификатор матча
Match	Match	false	Связь с основной сущностью матча
BeginAt	DateTimeOffset	true	Время начала карты
Status	MapStatusEnum	true	Статус карты
MapName	MapNameEnum	true	Название карты
WinnerScore	int	true	Счет победителя
LoserScore	int	true	Счет проигравшего
WinnerId	int	true	Идентификатор команды-победителя
Winner	Team	false	Связь с командой-победителем
LoserId	int	true	Идентификатор команды-проигравшей
Loser	Team	false	Связь с командой-проигравшей
Number	int	true	Номер карты
RoundCount	int	true	Количество раундов
Discipline	DisciplineEnum	true	Дисциплина карты
PlayerMetrics	List<PlayerMetric>	false	Метрики игроков на карте

Продолжение таблицы 4.8

1	2	3	4
PlayerResult Metrics	List<Player ResultMetric>	false	Результаты игроков на карте
Rounds	List<Round>	false	Раунды карты
DemoName	string	false	Название демозаписи карты

Таблица 4.9 – Свойства класса Match

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор матча
Slug	string	true	Уникальный слаг для матча
Team1Id	int	true	Идентификатор первой команды
Team1	Team	false	Связь с первой командой
Team2Id	int	true	Идентификатор второй команды
Team2	Team	false	Связь со второй командой
WinnerTeamId	int	false	Идентификатор команды-победителя
WinnerTeam	Team	false	Связь с командой-победителем
LoserTeamId	int	false	Идентификатор команды-проигравшей

Продолжение таблицы 4.9

1	2	3	4
LoserTeam	Team	false	Связь с командой-проигравшей
Team1Score	int	true	Счет первой команды
Team2Score	int	true	Счет второй команды
Status	MatchStatusEnum	true	Статус матча
BOType	int	true	Тип формата ”лучший из”(BO1, BO3 и т.д.)
StartDate	DateTimeOffset	true	Дата начала матча
EndDate	DateTimeOffset	true	Дата окончания матча
ParsedStatus	ParsedStatusEnum	true	Статус парсинга матча
Url	string	false	URL матча
DemoUrl	string	false	URL демозаписи матча
Maps	List<Map>	false	Список карт в матче
Discipline	DisciplineEnum	true	Дисциплина матча
TournamentID	int	true	Идентификатор турнира
Tournament	Tournament	false	Связь с основной сущностью турнира

Таблица 4.10 – Свойства класса PlayerMetric

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор метрики игрока
MapId	int	true	Идентификатор карты
PlayerId	int	true	Идентификатор игрока
Kills	int	true	Количество убийств
Deaths	int	true	Количество смертей
Assists	int	true	Количество ассистов
Headshots	int	true	Количество выстрелов в голову
Damage	int	true	Нанесенный урон

Таблица 4.11 – Свойства класса PlayerResultMetric

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентифика-тор результата метрики игрока
MapId	int	true	Идентификатор карты
PlayerId	int	true	Идентификатор игрока

Продолжение таблицы 4.11

1	2	3	4
RoundsPlayed	int	true	Количество сыгранных раундов
RoundsWon	int	true	Количество выигранных раундов
Kills	int	true	Количество убийств
Deaths	int	true	Количество смертей
Assists	int	true	Количество ассистов
Headshots	int	true	Количество выстрелов в голову

Таблица 4.12 – Свойства класса PlayerStats

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор статистики игрока
PlayerId	int	true	Идентификатор игрока
MatchesPlayed	int	true	Количество сыгранных матчей
Kills	int	true	Количество убийств
Deaths	int	true	Количество смертей
Assists	int	true	Количество ассистов

Продолжение таблицы 4.12

1	2	3	4
Headshots	int	true	Количество выстрелов в голову
Damage	int	true	Нанесенный урон
MVPs	int	true	Количество титулов MVP
Rating	float	true	Рейтинг игрока

Таблица 4.13 – Свойства класса Round

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор раунда
MapId	int	true	Идентификатор карты
Number	int	true	Номер раунда
WinningTeamId	int	true	Идентификатор выигравшей команды
WinningTeam	Team	false	Связь с выигравшей командой
LosingTeamId	int	true	Идентификатор проигравшей команды
LosingTeam	Team	false	Связь с проигравшей командой

Таблица 4.14 – Свойства класса RoundPlayerMetric

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор метрики игрока за раунд
RoundId	int	true	Идентификатор раунда
PlayerId	int	true	Идентификатор игрока
Kills	int	true	Количество убийств
Deaths	int	true	Количество смертей
Assists	int	true	Количество ассистов
Headshots	int	true	Количество выстрелов в голову
Damage	int	true	Нанесенный урон

Таблица 4.15 – Свойства класса RoundTeamMetric

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор метрики команды за раунд
RoundId	int	true	Идентификатор раунда
TeamId	int	true	Идентификатор команды

Продолжение таблицы 4.15

1	2	3	4
Kills	int	true	Количество убийств
Deaths	int	true	Количество смертей
Assists	int	true	Количество ассистов
Headshots	int	true	Количество выстрелов в голову
Damage	int	true	Нанесенный урон

Таблица 4.16 – Свойства класса TeamResultMetric

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор результата метрики команды
MapId	int	true	Идентификатор карты
TeamId	int	true	Идентификатор команды
RoundsPlayed	int	true	Количество сыгранных раундов
RoundsWon	int	true	Количество выигранных раундов
Kills	int	true	Количество убийств
Deaths	int	true	Количество смертей

Продолжение таблицы 4.16

1	2	3	4
Assists	int	true	Количество ассистов
Headshots	int	true	Количество выстрелов в голову

4.1.2 Описание внутренних сущностей сервиса анализа .dem файла

Таблица 4.17 – Свойства класса KillEvent

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор события убийства
PlayerId	int	true	Идентификатор игрока
VictimId	int	true	Идентификатор жертвы
Weapon	string	true	Оружие, использованное в убийстве
Headshot	bool	true	Было ли убийство выстрелом в голову

Таблица 4.18 – Свойства класса EntryKillEvent

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор события первого убийства
PlayerId	int	true	Идентификатор игрока
VictimId	int	true	Идентификатор жертвы
Weapon	string	true	Оружие, использованное в убийстве
Time	float	true	Время события

Таблица 4.19 – Свойства класса RoundEndReason

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор причины окончания раунда
Reason	string	true	Описание причины

Таблица 4.20 – Свойства класса RoundType

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор типа раунда

Продолжение таблицы 4.20

1	2	3	4
Type	string	true	Тип раунда (например, пистолетный, эко-раунд и т.д.)

Таблица 4.21 – Свойства класса WeaponFireEvent

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор события выстрела
PlayerId	int	true	Идентификатор игрока
Weapon	string	true	Оружие, использованное в выстреле
Time	float	true	Время события

Таблица 4.22 – Свойства класса Demo

Свойство	Тип	Обязательное	Описание
Parser	dem.Parser	true	Парсер демо файла
Players	[]*Models.Player	true	Список игроков
Rounds	[]Models.Round	true	Список раундов
Team1	*Models.Team	false	Первая команда
Team2	*Models.Team	false	Вторая команда
CurrentRound	Models.Round	true	Текущий раунд
IsFirstKillOccurred	bool	true	Произошло ли первое убийство

Продолжение таблицы 4.22

1	2	3	4
IsFirstShotOccured	bool	true	Произошел ли первый выстрел
PlayerInClutch1	*Models.Player	false	Игрок в ключевой момент
PlayerInClutch2	*Models.Player	false	Второй игрок в ключевой момент
CurrentClutch	*Models .ClutchEvent	false	Текущий ключевой момент
IsMatchStarted	bool	true	Начался ли матч
IsFreezetime	bool	true	Период фризтайма
KillsThisRound	map[*common .Player]int	true	Количество убийств в этом раунде
TimeStartKill	map[*events .PlayerHurt]float64	true	Время начала убийства
Kills	[]Models.KillEvent	true	Список убийств
PlayersHurted	[]*Models .PlayerHurtedEvent	true	Список раненных игроков
WeaponFired	[]Models .WeaponFireEvent	true	Список выстрелов
Date	time.Time	true	Дата игры

Таблица 4.23 – Свойства структуры ClutchEvent

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Идентификатор события
PlayerId	int	true	Идентификатор игрока

Продолжение таблицы 4.23

1	2	3	4
RoundId	int	true	Идентификатор раунда
EventTime	time.Time	true	Время события
Outcome	string	true	Исход события

Таблица 4.24 – Свойства структуры PlayerHurtedEvent

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Идентификатор события
PlayerId	int	true	Идентификатор игрока
RoundId	int	true	Идентификатор раунда
EventTime	time.Time	true	Время события
Damage	int	true	Нанесенный урон
AttackerId	int	true	Идентификатор атакующего игрока

4.1.3 Описание внутренних сущностей сервиса поиска информации о турнирах

Таблица 4.25 – Свойства класса StageDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор этапа
Title	string	true	Название этапа
FormatType	string	true	Тип формата этапа
Status	string	true	Статус этапа
Teams	List<TeamDto>	false	Список команд на этапе

Таблица 4.26 – Свойства класса TeamClanDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
ClanName	string	true	Название клана
TeamId	int	true	Идентификатор команды
Id	int	true	Уникальный идентификатор

Таблица 4.27 – Свойства класса TeamDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор команды
Slug	string	true	Уникальный слаг команды
Name	string	true	Название команды

Продолжение таблицы 4.27

1	2	3	4
IconUrl	string	false	URL иконки команды
YoutubeUrl	string	false	URL YouTube канала команды
WebsiteUrl	string	false	URL веб-сайта команды
DisciplineId	int	true	Идентификатор дисциплины
Acronym	string	false	Аббревиатура команды
Country	CountryDto	false	Страна команды

Таблица 4.28 – Свойства класса CountryDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор страны
Code	string	true	Код страны
Name	string	true	Название страны

Таблица 4.29 – Свойства класса MatchFullDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор матча
WinnerTeamId	int	true	Идентификатор команды-победителя

Продолжение таблицы 4.29

1	2	3	4
LoserTeamId	int	true	Идентификатор команды-проигравшей
TournamentID	int	true	Идентификатор турнира

Таблица 4.30 – Свойства класса MatchGeneralDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Total	TotalDto	true	Общая информация
MatchtMin	List<Match MinDto>	true	Список минимальных данных о матчах

Таблица 4.31 – Свойства класса MatchMinDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Уникальный идентификатор матча
Slug	string	true	Уникальный слаг матча
Team1Id	int	true	Идентификатор первой команды
Team2Id	int	true	Идентификатор второй команды
Team1Score	int	true	Счет первой команды
Team2Score	int	true	Счет второй команды

Продолжение таблицы 4.31

1	2	3	4
Status	string	true	Статус матча
StartDate	DateTimeOffset?	false	Дата начала матча
EndDate	DateTimeOffset?	false	Дата окончания матча
BOType	int	true	Тип формата ”лучший из”(BO1, BO3 и т.д.)
ParsedStatus	string	true	Статус парсинга матча
NestedTournament	Nested TournamentDto	true	Вложенный объект турнира

Таблица 4.32 – Свойства класса PlayerDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Идентификатор игрока
Slug	string	true	Короткое имя игрока
Nickname	string	true	Никнейм игрока
FirstName	string?	false	Имя игрока
LastName	string?	false	Фамилия игрока
ImageUrl	string?	false	URL изображения игрока
TeamId	int?	false	Идентификатор команды
ImageData	string?	false	Данные изображения игрока
Country	CountryDto	true	Страна игрока

Таблица 4.33 – Свойства класса MapDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Идентификатор карты
MatchID	int	true	Идентификатор матча
BeginAt	DateTimeOffset	true	Дата и время начала
Status	string	true	Статус карты
MapName	string	true	Название карты
WinnerScore	int	true	Счет победившей команды
LoserScore	int	true	Счет проигравшей команды
WinnerTeam	NestedTeamDto	true	Победившая команда
LoserTeam	NestedTeamDto	true	Проигравшая команда
Number	int	true	Номер карты
RoundCount	int	true	Количество раундов

Таблица 4.34 – Свойства класса PlayerStatsDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	long	true	Идентификатор статистики
MapId	long	true	Идентификатор карты
PlayerId	NestedPlayerDto	true	Идентификатор игрока

Продолжение таблицы 4.34

1	2	3	4
TeamId	NestedTeamDto	true	Идентификатор команды
Win	int	true	Количество побед
Kills	int	true	Количество убийств
Assists	int	true	Количество ассистов
Death	int	true	Количество смертей
Adr	double	true	Средний урон за раунд
AdditionalValue	int	true	Дополнительное значение
Clutches	int	true	Количество клатчей
CumulativeRound Damages	Dictionary<int, int>	true	Суммарный урон по раундам
Damage	int	true	Урон
FirstKills	int	true	Первые убийства
FirstDeath	int	true	Первые смерти
GotDamage	int	true	Полученный урон
Headshots	int	true	Количество хэд-шотов
Hits	int	true	Количество попаданий
Kast	double	true	Коэффициент участия
MoneySpent	int	true	Потраченные деньги

Продолжение таблицы 4.34

1	2	3	4
MoneySave	int	true	Сохраненные деньги
Multikills	Dictionary<int, int>	true	Мульт-убийства
PistolsValue	int	true	Стоимость пистолетов
PlayerRating	double	true	Рейтинг игрока
Shots	int	true	Количество выстрелов
TotalEquipmentValue	int	true	Общая стоимость экипировки
WeaponsValue	int	true	Стоимость оружия
TradeDeath	int	true	Количество обменов смертями
TradeKills	int	true	Количество обменов убийствами
UtilityValue	int	true	Стоимость всех гранат

Таблица 4.35 – Свойства класса TeamGeneralDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Total	TotalDto	true	Общая информация
TeamRanks	List<TeamRankDto>	true	Список рангов команды

Таблица 4.36 – Свойства класса TeamRankDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Идентификатор ранга
TeamId	int	true	Идентификатор команды
TeamRosterId	int	true	Идентификатор состава команды
Current	bool?	false	Текущий ранг
TopTier	bool?	false	Топовый ранг
SixMonthValue	int?	false	Значение за 6 месяцев
SixMonthTourWins	int?	false	Победы за 6 месяцев
CurrYearValue	int?	false	Значение за текущий год
CurrYearTourWins	int?	false	Победы за текущий год
Rank	int	true	Ранг
Date	DateTime?	false	Дата
CreatedAt	DateTime?	false	Дата создания
UpdatedAt	DateTime?	false	Дата обновления
Team	TeamDto	true	Информация о команде
TeamRoster	TeamRosterDto	true	Состав команды

Таблица 4.37 – Свойства класса TeamRosterDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Players	List<PlayerDto>	true	Список игроков

Таблица 4.38 – Свойства класса TotalDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Count	int	true	Количество
Pages	int	true	Количество страниц
Offset	int	true	Смещение
Limit	int	true	Лимит

Таблица 4.39 – Свойства класса TournamentFullDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Идентификатор турнира
EventType	string	true	Тип события
Description	string	true	Описание турнира
Discipline	string	true	Дисциплина
ImageUrl	string	true	URL изображения
Country	int?	false	Идентификатор страны
Stages	List<StageDto>	true	Список стадий турнира

Таблица 4.40 – Свойства класса TournamentGeneralDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Total	TotalDto	true	Общая информация

Продолжение таблицы 4.40

1	2	3	4
TournamentMin	List<TournamentMinDto>	true	Список минимальных данных о турнирах

Таблица 4.41 – Свойства класса TournamentMinDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Идентификатор турнира
Name	string	true	Название турнира
Slug	string	true	Короткое имя турнира
Status	string	true	Статус турнира
Tier	string	true	Уровень турнира
StartDate	DateTimeOffset	true	Дата начала
EndDate	DateTimeOffset	true	Дата окончания
Prize	int	true	Призовой фонд

4.1.4 Описание внутренних сущностей сервиса поиска

Таблица 4.42 – Свойства класса PlayerSearchDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Идентификатор игрока
Nickname	string	true	Никнейм игрока
FirstName	string?	false	Имя игрока
LastName	string?	false	Фамилия игрока

Продолжение таблицы 4.42

1	2	3	4
ImageUrl	string?	false	URL изображения игрока

Таблица 4.43 – Свойства класса TeamSearchDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Идентификатор команды
Slug	string	true	Короткое имя команды
Name	string	true	Название команды
IconUrl	string?	false	URL иконки команды

Таблица 4.44 – Свойства класса TournamentSearchDto

Свойство	Тип	Обязательное	Описание
1	2	3	4
Id	int	true	Идентификатор турнира
Name	string	true	Название турнира
Slug	string	true	Короткое имя турнира
ImageUrl	string?	false	URL изображения турнира
Prize	int	true	Призовой фонд
StartDate	DateTimeOffset	true	Дата начала турнира

Продолжение таблицы 4.44

1	2	3	4
EndDate	DateTimeOffset	true	Дата окончания турнира

Таблица 4.45 – Свойства класса SearchResult

Свойство	Тип	Обязательное	Описание
1	2	3	4
Players	List<PlayerSearchDto>	true	Список игроков
Teams	List<TeamSearchDto>	true	Список команд
Tournaments	List<TournamentSearchDto>	true	Список турниров

4.1.5 Спецификация классов веб-платформы

4.1.6 Спецификация классов сервиса анализа .dem файла

Класс `Demo` предназначен для анализа демо-файлов киберспортивной игры Counter-Strike 2. Он содержит данные о игроках, раундах, командах и различных событиях, происходящих в матче. Класс также включает переменные для расчета рейтингов игроков на основе различных статистик.

Таблица 4.46 – Спецификация полей класса Demo

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
Parser	package	dem.Parser	Парсер демо-файла
Players	package	[]*Models.Player	Список игроков
Rounds	package	[]Models.Round	Список раундов
Team1	package	*Models.Team	Первая команда

Продолжение таблицы 4.46

1	2	3	4
Team2	package	*Models.Team	Вторая команда
CurrentRound	package	Models.Round	Текущий раунд
IsFirstKill Occurred	package	bool	Произошло первое убийство
IsFirstShot Occured	package	bool	Произошел первый выстрел
PlayerInClutch1	package	*Models.Player	Игрок в первом клатче
PlayerInClutch2	package	*Models.Player	Игрок во втором клатче
CurrentClutch	package	*Models. ClutchEvent	Текущее событие клатча
IsMatchStarted	package	bool	Матч начался
IsFreezetime	package	bool	Время разминки
KillsThisRound	package	map[*common. Player]int	Убийства в этом раунде
TimeStartKill	package	map[*events. PlayerHurt] float64	Время начала убийства
Kills	package	[]Models .KillEvent	События убийств
PlayersHurted	package	[]*Models.Player HurtedEvent	Раненые игроки
WeaponFired	package	[]Models.Weapon FireEvent	События стрельбы из оружия
Date	package	time.Time	Дата

Таблица 4.47 – Спецификация методов класса Demo

Наименование	Метод доступа	Описание
1	2	3
ProcessPlayersRating	public	Обработка рейтингов игроков

Продолжение таблицы 4.47

1	2	3
ComputeHltvOrgRating	public	Вычисление рейтинга HLTV
ComputeHltv2OrgRating	public	Вычисление рейтинга 2 HLTV
ComputeEseaRws	public	Вычисление рейтинга ESEA RWS
findPlayerBySteamID	private	Поиск игрока по Steam ID
ProcessTradeKill	public	Обработка убийств на обмене
ProcessOpenAnd EntryKills	public	Обработка первых убийств
CheckForSpecial ClutchEnd	public	Проверка окончания клэтча
UpdateKillsCount	public	Обновление количества убийств
HandlePlayerHurted	public	Обработка ранений игрока
ProcessClutches	public	Обработка клэтчей
FindAlivePlayerInTeam	public	Поиск живого игрока в команде
HandlePlayerKilled	public	Обработка убийства игрока
HandleRoundEnd	public	Обработка окончания раунда
HandleWeaponFired	public	Обработка стрельбы из оружия
CalcEquipValue	public	Расчет стоимости экипировки
GetSideWithHigher EquipValue	public	Получение стороны с большей стоимостью экипировки
GetTeamBySide	public	Получение команды по стороне

Класс TeamsSheet предназначен для подсчета статистики команд в демо-файлах киберспортивной игры Counter-Strike 2 для их сохранения в базу данных.

Таблица 4.48 – Спецификация полей класса TeamsSheet

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
Demo	package	*Analyzer.Demo	Демо-объект
rowPerTeamName	package	map[string] *Models .TeamSheetRow	Строки статистики для каждой команды
MapID	package	int	Идентификатор карты
TeamId	package	int	Идентификатор команды

Таблица 4.49 – Спецификация методов класса TeamsSheet

Наименование	Метод доступа	Описание
1	2	3
NewTeamsSheet	public	Создание нового объекта TeamsSheet
AddDemo	public	Добавление демо и обновление статистики команды
UpdateTeamStats	public	Обновление статистики команды

Класс PlayersSheet предназначен для структурирования статистики игроков перед их сохранением в базу данных.

Таблица 4.50 – Спецификация полей класса PlayersSheet

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
Demo	package	*Analyzer.Demo	Демо-объект
first	package	bool	Первый запуск анализа

Таблица 4.51 – Спецификация методов класса PlayersSheet

Наименование	Метод доступа	Описание
1	2	3
Generate	public	Генерация статистики игроков

Класс `Team` предназначен для представления данных о командах в киберспортивной игре Counter-Strike 2. Он содержит информацию о названии команды, очках, игроках и различных статистиках, связанных с матчами и раундами.

Таблица 4.52 – Спецификация полей класса Team

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
Name	package	string	Название команды
Score	package	int	Очки команды
ScoreFirstHalf	package	int	Очки команды в первой половине
ScoreSecondHalf	package	int	Очки команды во второй половине
Players	package	[]*Player	Список игроков команды

Продолжение таблицы 4.52

1	2	3	4
WinMatchCount	package	int	Количество выигранных матчей
RoundCount	package	int	Количество раундов
WinRoundCtCount	package	int	Количество выигранных раундов за сторону CT
LostRoundCtCount	package	int	Количество проигранных раундов за сторону CT
WinRoundTCount	package	int	Количество выигранных раундов за сторону T
LostRoundTCount	package	int	Количество проигранных раундов за сторону T
WinPistolRoundCount	package	int	Количество выигранных пистолетных раундов
WinEcoRoundCount	package	int	Количество выигранных эко-раундов
WinSemiEco RoundCount	package	int	Количество выигранных полу-эко раундов
WinForceBuy RoundCount	package	int	Количество выигранных форс-бай раундов
BombPlantedOnACount	package	int	Количество установок бомбы на пленте A
BombPlantedOnBCount	package	int	Количество установок бомбы на пленте B
UtilityBuy	package	int	Количество закупленных гранат
UtilityUse	package	int	Количество использованных гранат

Таблица 4.53 – Спецификация методов класса Team

Наименование	Метод доступа	Описание
1	2	3
WinCount	public	Возвращает количество выигранных матчей
LostCount	public	Возвращает количество проигранных матчей
WinRoundCount	public	Возвращает количество выигранных раундов
LostRoundCount	public	Возвращает количество проигранных раундов
EntryHoldKillCount	public	Возвращает количество фрагов при удержании позиции
EntryKillCount	public	Возвращает количество первых фрагов
EntryHoldKillWonCount	public	Возвращает количество выигранных фрагов при удержании позиции
EntryHoldKillLossCount	public	Возвращает количество проигранных фрагов при удержании позиции
RatioEntryHoldKill	public	Возвращает процент выигранных фрагов при удержании позиции
EntryKillWonCount	public	Возвращает количество выигранных первых фрагов
EntryKillLossCount	public	Возвращает количество проигранных первых фрагов
FlashbangThrownCount	public	Возвращает количество брошенных световых гранат

Продолжение таблицы 4.53

1	2	3
HeGrenadeThrownCount	public	Возвращает количество брошенных осколочных гранат
SmokeThrownCount	public	Возвращает количество брошенных дымовых гранат
MolotovThrownCount	public	Возвращает количество брошенных молотовых
IncendiaryThrownCount	public	Возвращает количество брошенных зажигательных гранат
DecoyThrownCount	public	Возвращает количество брошенных ложных гранат
TradeKillCount	public	Возвращает количество разменов
TradeDeathCount	public	Возвращает количество убийств при размене
RatioEntryKill	public	Возвращает процент успешных разменов
KillCount	public	Возвращает количество убийств
AssistCount	public	Возвращает количество помощи
DeathCount	public	Возвращает количество смертей
BombPlantedCount	public	Возвращает количество установленных бомб
BombDefusedCount	public	Возвращает количество обезвреженных бомб
BombExplodedCount	public	Возвращает количество взрывов бомбы

Продолжение таблицы 4.53

1	2	3
FiveKillCount	public	Возвращает количество пента-убийств (пять убийств в одном раунде)
FourKillCount	public	Возвращает количество квадро-убийств (четыре убийства в одном раунде)
ThreeKillCount	public	Возвращает количество трипл-убийств (три убийства в одном раунде)
TwoKillCount	public	Возвращает количество дабл-убийств (два убийства в одном раунде)
OneKillCount	public	Возвращает количество одиночных убийств
JumpKillCount	public	Возвращает количество убийств в прыжке
CrouchKillCount	public	Возвращает количество убийств при приседании

Класс Player предназначен для представления данных об игроках в киберспортивной игре Counter-Strike 2. Он содержит информацию о статистике игрока, таких как количество убийств, смертей, помощи, а также рейтинги и другие показатели.

Таблица 4.54 – Спецификация полей класса Player

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
OneKillCount	package	int	Количество одиночных убийств
TwoKillCount	package	int	Количество двойных убийств

Продолжение таблицы 4.54

1	2	3	4
ThreeKillCount	package	int	Количество тройных убийств
FourKillCount	package	int	Количество четырех убийств
FiveKillCount	package	int	Количество пяти убийств
HeadshotCount	package	int	Количество убийств в голову
AssistCount	package	int	Количество помощей
DeathCount	package	int	Количество смертей
KillCount	package	int	Количество убийств
CrouchKillCount	package	int	Количество убийств при приседании
JumpKillCount	package	int	Количество убийств в прыжке
TeamKillCount	package	int	Количество убийств своих
KillThroughSmoke	package	int	Количество убийств через дым
RoundMvpCount	package	int	Количество MVP раундов
BombPlantedCount	package	int	Количество установок бомбы
BombDefusedCount	package	int	Количество разминирований бомбы
BombExploded Count	package	int	Количество взрывов бомбы

Продолжение таблицы 4.54

1	2	3	4
Score	package	int	Очки игрока
IsControllingBot	package	bool	Контролирует ли игрок бота
IsAlive	package	bool	Жив ли игрок
IsConnected	package	bool	Подключен ли игрок
Side	package	common.Team	Сторона игрока (T или CT)
TeamName	package	string	Название команды
SteamId	package	uint64	Steam ID игрока
Name	package	string	Имя игрока
HasEntryKill	package	bool	Совершил ли игрок первый фраг
HasEntryHoldKill	package	bool	Совершил ли игрок фраг при удержании позиции
RatingHltv	package	float64	Рейтинг HLTV игрока
RatingHltv2	package	float64	Рейтинг HLTV 2.0 игрока
Kast	package	float64	Показатель KAST
Rage	package	int	Уровень ярости
EseaRwsPointCount	package	float64	Количество очков ESEA RWS
EseaRws	package	float64	Рейтинг ESEA RWS
ShotCount	package	int	Количество выстрелов
HitCount	package	int	Количество попаданий

Продолжение таблицы 4.54

1	2	3	4
HasBomb	package	bool	Есть ли у игрока бомба
IsVacBanned	package	bool	Забанен ли игрок в VAC
IsOverwatchBanned	package	bool	Забанен ли игрок в Overwatch
RankNumberOld	package	int	Старый номер ранга
RankNumberNew	package	int	Новый номер ранга
WinCount	package	int	Количество побед
TradeKillCount	package	int	Количество убийств на обмене
TradeDeathCount	package	int	Количество смертей на обмене
FlashbangThrown Count	package	int	Количество брошенных светошумовых гранат
SmokeThrownCount	package	int	Количество брошенных дымовых гранат
HeGrenadeThrown Count	package	int	Количество брошенных гранат HE
MolotovThrown Count	package	int	Количество брошенных коктейлей Молотова
IncendiaryThrown Count	package	int	Количество брошенных зажигательных гранат
DecoyThrownCount	package	int	Количество брошенных ложных гранат

Продолжение таблицы 4.54

1	2	3	4
RoundPlayedCount	package	int	Количество сыгранных раундов
SuicideCount	package	int	Количество суицидов
FlashDurationTemp	package	float32	Временная продолжительность ослепления
EquipementValue Rounds	package	map[int]int	Стоимость экипировки по раундам
RoundsMoney Earned	package	map[int]int	Заработанные деньги по раундам
TimeDeathRounds	package	map[int]float64	Время смертей по раундам
Kills	package	[]KillEvent	Список событий убийств
Deaths	package	[]KillEvent	Список событий смертей
Assists	package	[]KillEvent	Список событий помощей
EntryKills	package	[]EntryKillEvent	Список событий первых убийств
EntryHoldKills	package	[]EntryHoldKill Event	Список событий убийств при удержании позиции
PlayersHurted	package	[]*PlayerHurted Event	Список событий ранений игроков
Clutches	package	[]ClutchEvent	Список событий клэтчей
TimeKill	package	[]float64	Время убийств

Таблица 4.55 – Спецификация методов класса Player

Наименование	Метод доступа	Описание
1	2	3
KillDeathRatio	public	Возвращает соотношение убийств к смертям
Accuracy	public	Возвращает точность стрельбы игрока
EntryKillWonCount	public	Возвращает количество выигранных первых фрагов
EntryKillLossCount	public	Возвращает количество проигранных первых фрагов
EntryHoldKillWonCount	public	Возвращает количество выигранных фрагов при удержании позиции
EntryHoldKillLossCount	public	Возвращает количество проигранных фрагов при удержании позиции
ClutchCount	public	Возвращает общее количество клатчей
ClutchLostCount	public	Возвращает количество проигранных клатчей
ClutchWonCount	public	Возвращает количество выигранных клатчей
Clutch1V1WonCount	public	Возвращает количество выигранных клатчей 1v1
Clutch1V2WonCount	public	Возвращает количество выигранных клатчей 1v2

Продолжение таблицы 4.55

1	2	3
Clutch1V3WonCount	public	Возвращает количество выигранных клатчей 1v3
Clutch1V4WonCount	public	Возвращает количество выигранных клатчей 1v4
Clutch1V5WonCount	public	Возвращает количество выигранных клатчей 1v5
Clutch1V1LossCount	public	Возвращает количество проигранных клатчей 1v1
Clutch1V2LossCount	public	Возвращает количество проигранных клатчей 1v2
Clutch1V3LossCount	public	Возвращает количество проигранных клатчей 1v3
Clutch1V4LossCount	public	Возвращает количество проигранных клатчей 1v4
Clutch1V5LossCount	public	Возвращает количество проигранных клатчей 1v5
Clutch1V1Count	public	Возвращает общее количество клатчей 1v1
Clutch1V2Count	public	Возвращает общее количество клатчей 1v2
Clutch1V3Count	public	Возвращает общее количество клатчей 1v3
Clutch1V4Count	public	Возвращает общее количество клатчей 1v4
Clutch1V5Count	public	Возвращает общее количество клатчей 1v5

Продолжение таблицы 4.55

1	2	3
ClutchWonPercent	public	Возвращает процент выигранных клатчей
Clutch1V1WonPercent	public	Возвращает процент выигранных клатчей 1v1
Clutch1V2WonPercent	public	Возвращает процент выигранных клатчей 1v2
Clutch1V3WonPercent	public	Возвращает процент выигранных клатчей 1v3
Clutch1V4WonPercent	public	Возвращает процент выигранных клатчей 1v4
Clutch1V5WonPercent	public	Возвращает процент выигранных клатчей 1v5
TotalDamageHealth Count	public	Возвращает общее количество урона по здоровью
TotalDamageArmor Count	public	Возвращает общее количество урона по броне
TotalDamageHealth ReceivedCount	public	Возвращает общее количество полученного урона по здоровью
TotalDamageArmor ReceivedCount	public	Возвращает общее количество полученного урона по броне
AverageHealthDamage	public	Возвращает средний урон по здоровью
HeadshotPercent	public	Возвращает процент убийств в голову

Продолжение таблицы 4.55

1	2	3
RatioEntryKill	public	Возвращает соотношение первых фрагов
RatioEntryHoldKill	public	Возвращает соотношение фрагов при удержании позиции
KillPerRound	public	Возвращает количество убийств за раунд
AssistPerRound	public	Возвращает количество помощей за раунд
DeathPerRound	public	Возвращает количество смертей за раунд
TotalTimeDeath	public	Возвращает общее время до смерти
AverageTimeDeath	public	Возвращает среднее время до смерти
ComputeStats	public	Вычисляет основные статистики игрока
ComputeKast	public	Вычисляет показатель KAST игрока
HasEventInRound	public	Проверяет наличие события в раунде
AverageTimeKill	public	Возвращает среднее время до убийства

4.1.7 Спецификация классов сервиса поиска информации о турнирах

Класс CountryIdsResolver предназначен для разрешения идентификаторов стран в процессе маппинга данных о турнирах.

Таблица 4.56 – Спецификация полей класса CountryIdsResolver

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
_context	private	Application DbContext	Контекст базы данных для разрешения стран

Таблица 4.57 – Спецификация методов класса CountryIdsResolver

Наименование	Метод доступа	Описание
1	2	3
Resolve	public	Разрешает идентификатор страны из DTO турнира

Класс ApplicationDbContext предназначен для управления доступом к данным базы данных о турнирах и командах.

Таблица 4.58 – Спецификация полей класса ApplicationDbContext

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
Maps	public	DbSet<Map>	Таблица карт
Matches	public	DbSet<Match>	Таблица матчей
Players	public	DbSet<Player>	Таблица игроков
PlayerMetrics	public	DbSet<Player Metric>	Таблица метрик игроков
Rounds	public	DbSet<Round>	Таблица раундов
Stages	public	DbSet<Stage>	Таблица этапов
Teams	public	DbSet<Team>	Таблица команд
TeamMetrics	public	DbSet<Team ResultMetric>	Таблица метрик команд

Продолжение таблицы 4.58

1	2	3	4
TeamNames	public	DbSet<Team Name>	Таблица имен команд
Tournaments	public	DbSet <Tournament>	Таблица турниров
Countries	public	DbSet<Country>	Таблица стран
PlayerStats	public	DbSet<Player Stats>	Таблица статистики игроков

Таблица 4.59 – Спецификация методов класса ApplicationDbContext

Наименование	Метод доступа	Описание
1	2	3
OnModelCreating	protected	Настройка моделей базы данных
SaveChangesAsync	public	Асинхронное сохранение изменений
SaveChanges	public	Сохранение изменений

Класс EntityMapper предназначен для маппинга DTO объектов на модели базы данных.

Таблица 4.60 – Спецификация методов класса EntityMapper

Наименование	Метод доступа	Описание
1	2	3
EntityMapper	public	Настройка профилей маппинга

Класс BackgroundServiceController предназначен для управления запуском и остановкой фоновых сервисов обновления данных о командах, турнирах и матчах.

Таблица 4.61 – Спецификация полей класса BackgroundServiceController

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
_taskQueue	private	IBackgroundTaskQueue	Очередь фоновых задач
_serviceScopeFactory	private	IServiceScopeFactory	Фабрика областей служб
_tokens	private	IDictionary<string, CancellationTokenSource>	Токены отмены для сервисов

Таблица 4.62 – Спецификация методов класса BackgroundServiceController

Наименование	Метод доступа	Описание
1	2	3
StartTeamUpdateService	public	Запускает фоновый сервис обновления команд
StartTournamentUpdateService	public	Запускает фоновый сервис обновления турниров
StartMatchUpdateService	public	Запускает фоновый сервис обновления матчей
StopService	public	Останавливает указанный фоновый сервис
StartService	private	Запускает указанный фоновый сервис с предоставленной функцией работы

Класс TierResolver предназначен для разрешения уровня турнира из DTO.

Таблица 4.63 – Спецификация методов класса TierResolver

Наименование	Метод доступа	Описание
1	2	3
Resolve	public	Разрешает уровень турнира из DTO

Класс EventTypeResolver предназначен для разрешения типа события турнира из DTO.

Таблица 4.64 – Спецификация методов класса EventTypeResolver

Наименование	Метод доступа	Описание
1	2	3
Resolve	public	Разрешает тип события турнира из DTO

Класс StageTypeResolver предназначен для разрешения типа этапа турнира из DTO.

Таблица 4.65 – Спецификация методов класса StageTypeResolver

Наименование	Метод доступа	Описание
1	2	3
Resolve	public	Разрешает тип этапа турнира из DTO

Класс ParsedStatusResolver предназначен для разрешения статуса матча из DTO.

Таблица 4.66 – Спецификация методов класса ParsedStatusResolver

Наименование	Метод доступа	Описание
1	2	3
Resolve	public	Разрешает статус матча из DTO

Класс TeamIdsResolver предназначен для разрешения идентификаторов команд в процессе маппинга данных о стадиях турнира.

Таблица 4.67 – Спецификация полей класса TeamIdsResolver

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
_context	private	ApplicationDb Context	Контекст базы данных для разрешения команд

Таблица 4.68 – Спецификация методов класса TeamIdsResolver

Наименование	Метод доступа	Описание
1	2	3
Resolve	public	Разрешает идентификаторы команд из DTO этапа

Класс MatchStatusResolver предназначен для разрешения статуса матча из DTO.

Таблица 4.69 – Спецификация методов класса MatchStatusResolver

Наименование	Метод доступа	Описание
1	2	3
Resolve	public	Разрешает статус матча из DTO

Класс StatusResolver предназначен для разрешения статуса турнира из DTO.

Таблица 4.70 – Спецификация методов класса StatusResolver

Наименование	Метод доступа	Описание
1	2	3
Resolve	public	Разрешает статус турнира из DTO

Класс HttpClientHelper предназначен для конфигурации HTTP-клиента с необходимыми заголовками.

Таблица 4.71 – Спецификация методов класса HttpClientHelper

Наименование	Метод доступа	Описание
1	2	3
ConfigureClient	public	Конфигурирует HTTP-клиент с необходимыми заголовками

Класс MatchUpdateService предназначен для обновления данных о матчах через API.

Таблица 4.72 – Спецификация полей класса MatchUpdateService

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
client	private	static HttpClient	HTTP-клиент для запросов
offset	private	int	Смещение для пагинации
limit	private	int	Лимит для пагинации
_mapper	private	IMapper	Маппер для преобразования DTO
_dbContext	private	IApplicationDb Context	Контекст базы данных

Продолжение таблицы 4.72

1	2	3	4
count	private	int	Количество матчей для обработки
_playerStats	private	PlayerStatsService	Сервис для обработки статистики игроков

Таблица 4.73 – Спецификация методов класса MatchUpdateService

Наименование	Метод доступа	Описание
1	2	3
UpdateMatchAsync	public	Асинхронное обновление данных о матчах
GetMaps	public	Получение карт для матча
UpdateMatchMin FinishedAsync	public	Асинхронное обновление данных о завершенных матчах

Класс PlayerStatsService предназначен для обновления статистики игроков через API.

Таблица 4.74 – Спецификация полей класса PlayerStatsService

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
client	private	static HttpClient	HTTP-клиент для запросов
_mapper	private	IMapper	Маппер для преобразования DTO
_dbContext	private	IApplicationDb Context	Контекст базы данных

Таблица 4.75 – Спецификация методов класса PlayerStatsService

Наименование	Метод доступа	Описание
1	2	3
AddPlayerStatsAsync	public	Асинхронное добавление статистики игрока

Класс TeamUpdateService предназначен для обновления данных о командах через API.

Таблица 4.76 – Спецификация полей класса TeamUpdateService

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
client	private	static HttpClient	HTTP-клиент для запросов
offset	private	int	Смещение для пагинации
limit	private	int	Лимит для пагинации
_mapper	private	IMapper	Маппер для преобразования DTO
_dbContext	private	IApplicationDb Context	Контекст базы данных
count	private	int	Количество команд для обработки

Таблица 4.77 – Спецификация методов класса TeamUpdateService

Наименование	Метод доступа	Описание
1	2	3
UpdateTeamsAsync	public	Асинхронное обновление данных о командах

Продолжение таблицы 4.77

1	2	3
SaveToDatabase	private	Сохранение данных в базу данных
UpdateTeamNameAsync	public	Асинхронное обновление имен команд

Класс TournamentUpdateService предназначен для обновления данных о турнирах через API.

Таблица 4.78 – Спецификация полей класса TournamentUpdateService

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
client	private	static HttpClient	HTTP-клиент для запросов
offset	private	int	Смещение для пагинации
limit	private	int	Лимит для пагинации
_mapper	private	IMapper	Маппер для преобразования DTO
_dbContext	private	IApplicationDb Context	Контекст базы данных
count	private	int	Количество турниров для обработки

Таблица 4.79 – Спецификация методов класса TournamentUpdateService

Наименование	Метод доступа	Описание
1	2	3
UpdateTournamentAsync	public	Асинхронное обновление данных о турнирах
UpdateTournamentMin FinishedAsync	public	Асинхронное обновление данных о завершенных турнирах
UpdateTournamentMin UpcomingAsync	public	Асинхронное обновление данных о предстоящих турнирах

4.1.8 Спецификация классов сервиса поиска

Класс SearchController предназначен для выполнения поиска по игрокам, командам и турнирам в базе данных.

Таблица 4.80 – Спецификация полей класса SearchController

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
_playerRepository	private	IPlayerRepository	Репозиторий для поиска игроков
_teamRepository	private	ITeamRepository	Репозиторий для поиска команд
_tournament Repository	private	ITournament Repository	Репозиторий для поиска турниров

Таблица 4.81 – Спецификация методов класса SearchController

Наименование	Метод доступа	Описание
1	2	3
Search	public	Выполняет поиск по игрокам, командам и турнирам

4.1.9 Спецификация классов сервиса прогнозирования

Класс app предназначен для настройки и запуска веб-приложения сервиса прогнозирования.

Таблица 4.82 – Спецификация полей класса app

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
app	private	Flask	Экземпляр Flask приложения

Таблица 4.83 – Спецификация методов класса app

Наименование	Метод доступа	Описание
1	2	3
run	public	Запускает Flask приложение

Класс PredictionService предназначен для предоставления функциональности прогнозирования на основе обученной модели.

Таблица 4.84 – Спецификация полей класса PredictionService

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
model	private	PredictorModel	Обученная модель для прогнозирования

Таблица 4.85 – Спецификация методов класса PredictionService

Наименование	Метод доступа	Описание
1	2	3
predict	public	Возвращает прогноз на основе входных данных
load_model	public	Загружает модель для прогнозирования

Класс PredictorModel предназначен для представления модели, используемой для прогнозирования.

Таблица 4.86 – Спецификация полей класса PredictorModel

Наименование	Область видимости	Тип данных	Описание
1	2	3	4
model	private	Any	Модель машинного обучения

Таблица 4.87 – Спецификация методов класса PredictorModel

Наименование	Метод доступа	Описание
1	2	3
predict	public	Выполняет прогноз на основе входных данных
load	public	Загружает модель из файла

4.2 Настройка взаимодействия сервисов

Взаимодействие между различными микросервисами веб-платформы для анализа и визуализации статистических данных киберспортивной игры Counter-Strike 2 осуществляется через API Gateway, реализованный с использованием Ocelot. API Gateway обеспечивает маршрутизацию запросов к соответствующим сервисам, обеспечивая централизованный входной узел для всех API-запросов.

Каждый микросервис запущен в отдельном контейнере и прослушивает свой порт:

- API Gateway: 5000
- Сервис предоставления данных: 5001
- Сервис прогнозирования: 5002
- Сервис поиска: 5003
- Сервис анализа .dem файла: 5004
- Сервис поиска информации о турнирах: 5005
- База данных: 5432
- Frontend: 3000
- Redis: 6379
- RabbitMQ: 5672
- Prometheus: 9090

Конфигурация API Gateway находится в файле ocelot.json. В этом файле определены правила маршрутизации (ReRoutes) для различных сервисов.

Каждый маршрут содержит информацию о схеме, хосте, портах и методах HTTP, которые должны использоваться для маршрутизации запросов.

Пример конфигурации ocelot.json для сервиса предоставления данных представлен на рисунке 4.1.

```
1   {
2     "ReRoutes": [
3     {
4       "DownstreamPathTemplate": "/api/data/{everything}",
5       "DownstreamScheme": "https",
6       "DownstreamHostAndPorts": [
7       {
8         "Host": "localhost",
9         "Port": 5001
10      }
11      ],
12      "UpstreamPathTemplate": "/api/data/{everything}",
13      "UpstreamHttpMethod": [ "Get", "Post", "Put", "Delete" ],
14      "FileCacheOptions":  { "TtlSeconds": 30 }
15    },
16    ...
17    ]
18  }
```

Рисунок 4.1 – Пример конфигурации ocelot.json

Из данной конфигурации видно следующее:

- DownstreamPathTemplate: Шаблон пути, по которому API Gateway будет маршрутизировать запросы к внутренним сервисам. В данном случае шаблон пути для downstream сервиса /api/data/everything указывает, что все запросы, соответствующие этому шаблону, будут перенаправляться на внутренний сервис.

- DownstreamScheme: Схема (http или https), которая будет использоваться для соединения с downstream сервисом. В этом случае используется https.

- DownstreamHostAndPorts: Список хостов и портов, на которые будут перенаправляться запросы. В данном случае запросы будут перенаправляться на локальный хост (localhost) с портом 5001.

- UpstreamPathTemplate: Шаблон пути, по которому API Gateway будет принимать запросы от клиентов. В данном случае шаблон пути для upstream сервиса /api/data/everything указывает, что все запросы, соответствующие этому шаблону, будут приняты API Gateway и перенаправлены на downstream сервис.

- UpstreamHttpMethod: Список методов HTTP, которые могут использоваться для запросов. В этом случае это методы GET, POST, PUT и DELETE.

- FileCacheOptions: Опции кэширования для файлов. В данном случае указано, что кэширование будет действовать в течение 30 секунд (”TtlSeconds”: 30).

4.3 Тестирование веб-платформы

4.3.1 Unit тестирование сервисов

Unit тестирование сервисов веб-платформы является важным этапом обеспечения качества и надежности системы. Unit тесты позволяют изолировать и проверить функциональность отдельных компонентов системы, обеспечивая уверенность в их корректной работе. Ниже приведены ключевые аспекты unit тестирования для различных сервисов платформы.

4.3.2 Описание unit тестов сервиса анализа .dem файла

Для тестирования функциональности сервиса анализа .dem файла, были написаны unit тесты, которые проверяют корректность выполнения вычислений рейтингов игроков и команд, считывания и сохранения различных событий и прочее.

На рисунке 4.2 представлены unit тесты сервиса анализа .dem файла.

16/16 tests passed (100%)

- github.com/Fresh-vano/CS2-Parser 5.0s
  - test 5.0s
    - analyze_test.go 960ms
      - TestAnalyzeDemo 200ms
      - TestEndMatch 300ms
      - TestTeamCalculate 460ms
    - ComputeEseaRws_test.go 3.3s
      - TestComputeEseaRws 270ms
      - TestComputeEseaRws1 2.4s
      - TestComputeEseaRws2 640ms
    - ComputeHltvOrgRating_test.go 0.0ms
      - TestComputeHltvOrgRating 0.0ms
        - #00
        - #01
        - #02
    - demoAnalyzer_test.go 780ms
      - TestDemoAnalyzerStub 110ms
      - TestProcessTradeKill 230ms
      - TestProcessOpenAndEntryKills 180ms
      - TestCheckForSpecialClutchEnd 120ms
      - TestUpdateKillsCount 60ms
      - TestHandlePlayerHurted 80ms

Рисунок 4.2 – Unit тесты сервиса анализа .dem файла

4.3.3 Описание unit тестов сервиса предоставления данных

Для тестирования функциональности сервиса предоставления данных, были написаны unit тесты, которые проверяют корректность выполнения запросов к базе данных, возврата списков игроков и команд, а также получения

информации о текущих и завершенных турнирах и матчах. В тестах использовалась библиотека Moq для создания mock-объектов контекста базы данных.

На рисунке 4.3 представлены unit тесты сервиса предоставления данных.

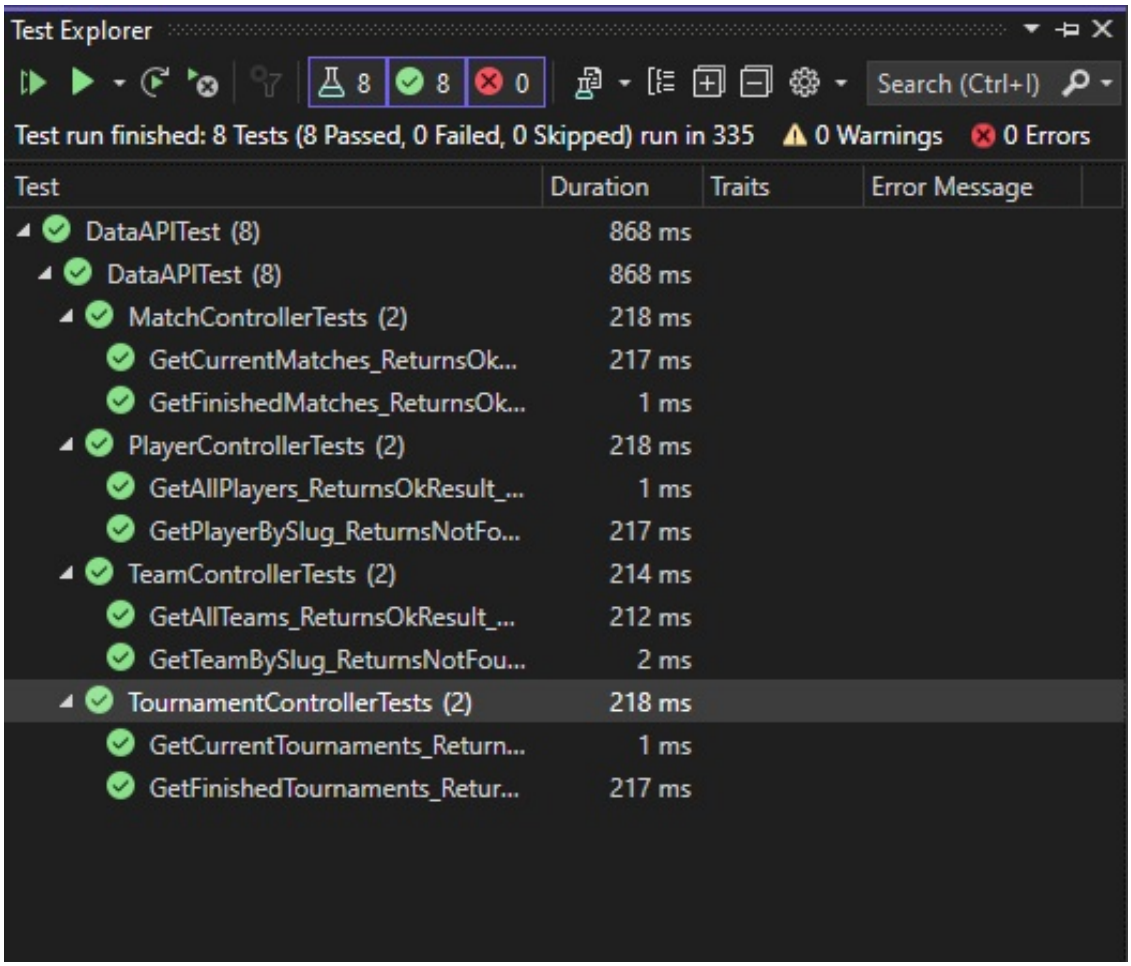

Рисунок 4.3 – Unit тесты сервиса предоставления данных

4.3.4 Описание unit тестов сервиса поиска информации о турнирах

Для тестирования функциональности сервиса поиска информации о турнирах были написаны unit тесты, которые проверяют корректность выполнения запросов к базе данных, обработки данных о текущих и завершённых турнирах, а также обновления информации о турнирах. В тестах исполь-

зовалась библиотека Moq для создания mock-объектов контекста базы данных и маппера объектов.

На рисунке 4.4 представлены unit тесты сервиса поиска информации о турнирах.

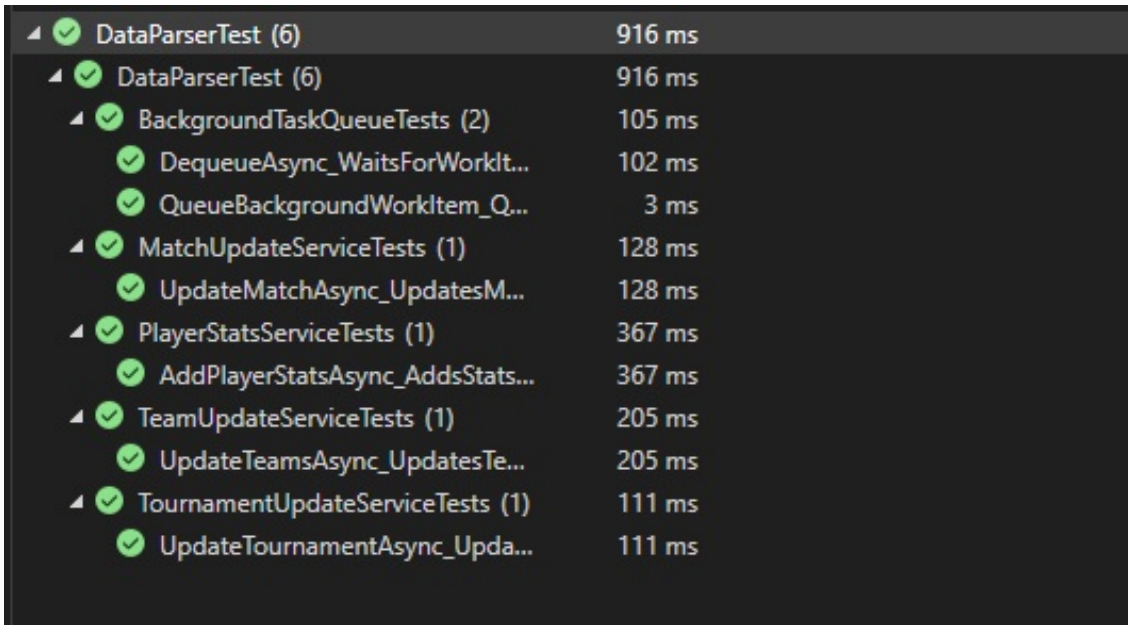

Рисунок 4.4 – Unit тесты сервиса поиска информации о турнирах

4.3.5 Интеграционное тестирование веб-платформы

Интеграционное тестирование веб-платформы направлено на проверку взаимодействия различных компонентов системы и их корректную работу в совокупности. В процессе тестирования проверяются интеграции между фронтендом и бэкендом, взаимодействие с внешними API, а также корректная работа базы данных и всех модулей в целом.

Для проведения интеграционного тестирования были использованы следующие подходы и инструменты:

- Тестирование REST API - с использованием Postman были написаны и автоматизированы тесты для проверки всех конечных точек API, их корректных ответов и обработки ошибок.

- Нагрузочное тестирование - с использованием Postman были созданы сценарии нагрузочного тестирования для проверки производительности си-

стемы под высокой нагрузкой. Это позволило оценить, как система справляется с большим количеством одновременных запросов и определить её пределы производительности.

- Контейнеризация и оркестрация - для обеспечения изоляции и управляемости тестовых окружений использовались Docker и Kubernetes. Это позволило создать реплицируемые и изолированные тестовые окружения, обеспечивая воспроизводимость тестов.

На рисунке 4.5 показан процесс нагрузочного тестирования веб-платформы.

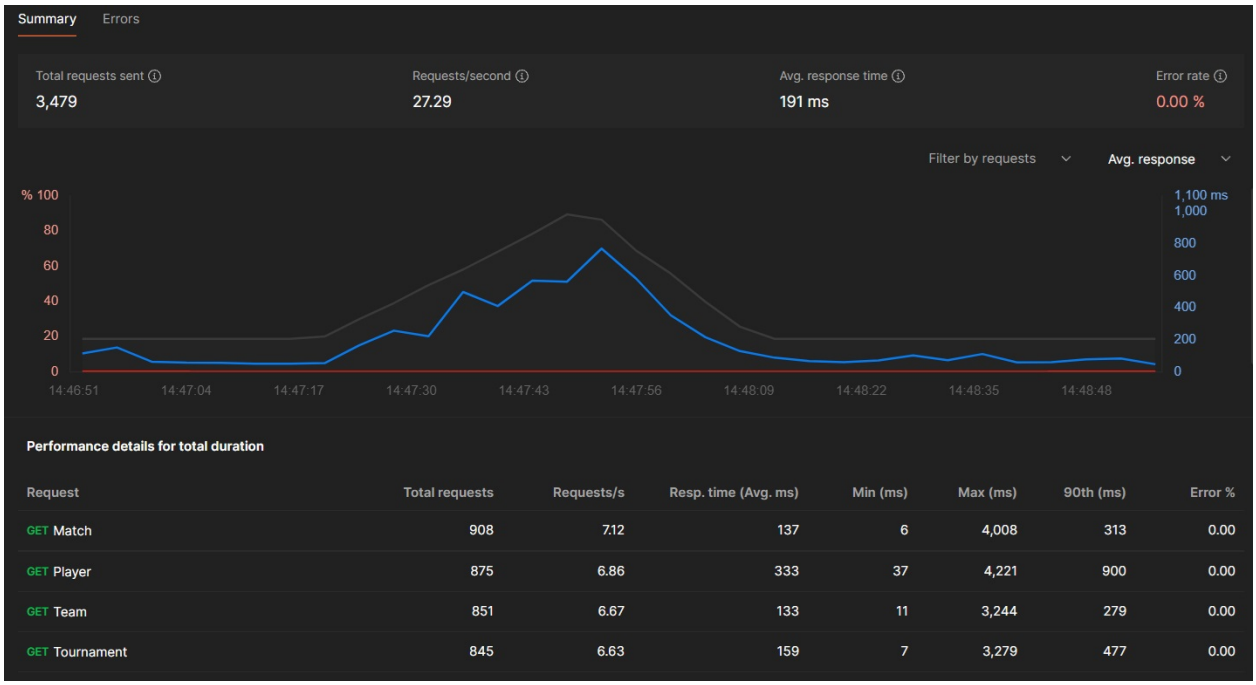

Рисунок 4.5 – Нагрузочное тестирование веб-платформы

ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе был разработан и внедрен комплексный инструмент для анализа и визуализации статистических данных киберспортивной игры Counter-Strike 2 (CS2). Данный проект направлен на улучшение игровых навыков и стратегий киберспортивных команд путем предоставления глубокого анализа игровых данных и прогнозов.

Основные результаты работы:

1. Проведен анализ предметной области киберспорта и выявлены ключевые метрики, необходимые для анализа и прогнозирования игровых результатов.

2. Разработана и реализована база данных для хранения информации о матчах, игроках, командах и турнирной статистике.

3. Создана нейронная сеть для прогнозирования исходов матчей на основе исторических данных и текущих игровых метрик.

4. Реализованы удобные интерфейсы для отображения статистики и прогнозов на веб-платформе, обеспечивая интуитивно понятный доступ к информации.

5. Осуществлен анализ метрик из .dem файлов игр для получения детальных данных о каждом матче.

6. Интегрированы внешние API для автоматического обновления данных о турнирах, матчах и игроках, что обеспечивает актуальность и точность информации.

7. Использованы технологии контейнеризации и оркестрации с Docker и Kubernetes для обеспечения высокой производительности и масштабируемости системы.

8. Внедрен брокер сообщений RabbitMQ для управления потоками данных и коммуникацией между микросервисами.

9. Проведены интеграционные и нагрузочные тестирования, которые подтвердили стабильную работу платформы под высокой нагрузкой.

Все требования, указанные в техническом задании, были полностью реализованы. Поставленные задачи в начале разработки проекта были успешно решены, что позволило создать инновационный инструмент для киберспортивных команд и аналитиков.

Готовая веб-платформа предоставляет возможность детально анализировать игровые метрики, оптимизировать тактики и прогнозировать результаты будущих матчей. Это способствует улучшению игровых навыков команд и их конкурентоспособности на международной арене.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Клеппман, М. Высоконагруженные приложения. Программирование, масштабирование, поддержка / М. Клеппман. – Санкт-Петербург : Питер, 2018. – 640 с. – ISBN 978-5-44-610512-0. – Текст : непосредственный.

2. Мартин, Р. Чистый код. Создание, анализ и рефакторинг / Р. Мартин. – Санкт-Петербург : Питер, 2020. – 464 с. – ISBN 978-5-4461-0960-9. – Текст : непосредственный.

3. Баланов, А. Построение микросервисной архитектуры и разработка высоконагруженных приложений. Учебное пособие / А. Баланов. – Москва : Лань, 2024. – 244 с. – ISBN 978-5-507-48747-9. – Текст : непосредственный.

4. Моуэт, Э. Использование Docker / Э. Моуэт. – Москва : ДМК-Пресс, 2017. – 354 с. – ISBN 978-5-97060-426-7. – Текст : непосредственный.

5. Webber, J. REST in Practice. Hypermedia and Systems Architecture / J. Webber. – Санкт-Петербург : Питер, 2010. – 448 с. – ISBN 978-0-5968-0582-1. – Текст : непосредственный.

6. Террелл, Р. Конкурентность и параллелизм на платформе .NET. Паттерны эффективного проектирования / Р. Террелл. – Санкт-Петербург : Питер, 2019. – 624 с. – ISBN 978-5-4461-1072-8. – Текст : непосредственный.

7. Порселло, Б. React. Современные шаблоны для разработки приложений / Б. Порселло. – Санкт-Петербург : Питер, 2022. – 320 с. – ISBN 978-5-4461-1492-4. – Текст : непосредственный.

8. Скотт, А. Разработка на JavaScript. Построение кроссплатформенных приложений с помощью GraphQL, React / А. Скоттарланд. – Санкт-Петербург : Питер, 2021. – 320 с. – ISBN 978-5-4461-1462-7. – Текст : непосредственный.

9. Бхаргава, А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих / А. Бхаргава. – Санкт-Петербург : Питер, 2022 – 288 с. – ISBN 978-5-4461-0923-4. – Текст : непосредственный.

10. AutoMapper : AutoMapper documentation : сайт. - URL: https://docs.automapper.org/en/stable/ (дата обращения: 14.04.2024). – Текст : электронный.

11. RabbitMQ : RabbitMQ documentation : сайт. - URL: https://www.rabbitmq.com/docs (дата обращения: 17.04.2024). – Текст : электронный.

12. Docker : Docker Docs : сайт. - URL: https://docs.docker.com/ (дата обращения: 27.03.2024). – Текст : электронный.

13. Material UI : Material UI components : сайт. - URL: https://mui.com/material-ui/all-components/ (дата обращения: 20.03.2024). – Текст : электронный.

ПРИЛОЖЕНИЕ А

Представление графического материала

Графический материал, выполненный на отдельных листах, изображен на рисунках А.1–А.6.

ВКРБ 2068443.09.03.04.24.001

1

Сведения о ВКРБ

Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА ПО ПРОГРАММЕ БАКАЛАВРИАТА

«Веб-платформа для анализа и визуализации статистических данных киберспортивной игры Counter-Strike 2»

Руководитель ВКРБ
к.т.н, доцент
Аникина Елена Игоревна

Автор ВКРБ
студент группы ПО-01б
Авилов Иван Александрович

	Фамилия И. О.	Подпись	Дата	ВКРБ 2068443.09.03.04.24.001			
Автор работы	Авилов И.А.			Сведения о ВКРБ	Лит.	Масса	Масштаб
Руководитель	Аникина Е.И.						
Нормоконтроль	Чернышов А.А.				Лист 1	Листов 6	
				Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-01б		

Рисунок А.1 – Сведения о ВКРБ

Цели и задачи разработки

Цель настоящей работы – разработка веб-платформы для анализа и визуализации статистических данных киберспортивной игры Counter-Strike 2.

Данная платформа должна предоставить киберспортивным командам, тренерам и аналитикам мощный инструмент для глубокого анализа игровых метрик и поддержки принятия решений на основе данных. Для достижения поставленной цели необходимо решить следующие задачи:

– провести анализ предметной области;

– разработать концептуальную модель веб-платформы;

– создать базу данных всех киберспортивных матчей;

– создать нейронную сеть для прогнозирования матчей;

– реализовать вывода статистики на веб-платформу;

– реализовать анализа метрик из .dem записей игр.

Рисунок А.2 – Цель и задачи разработки

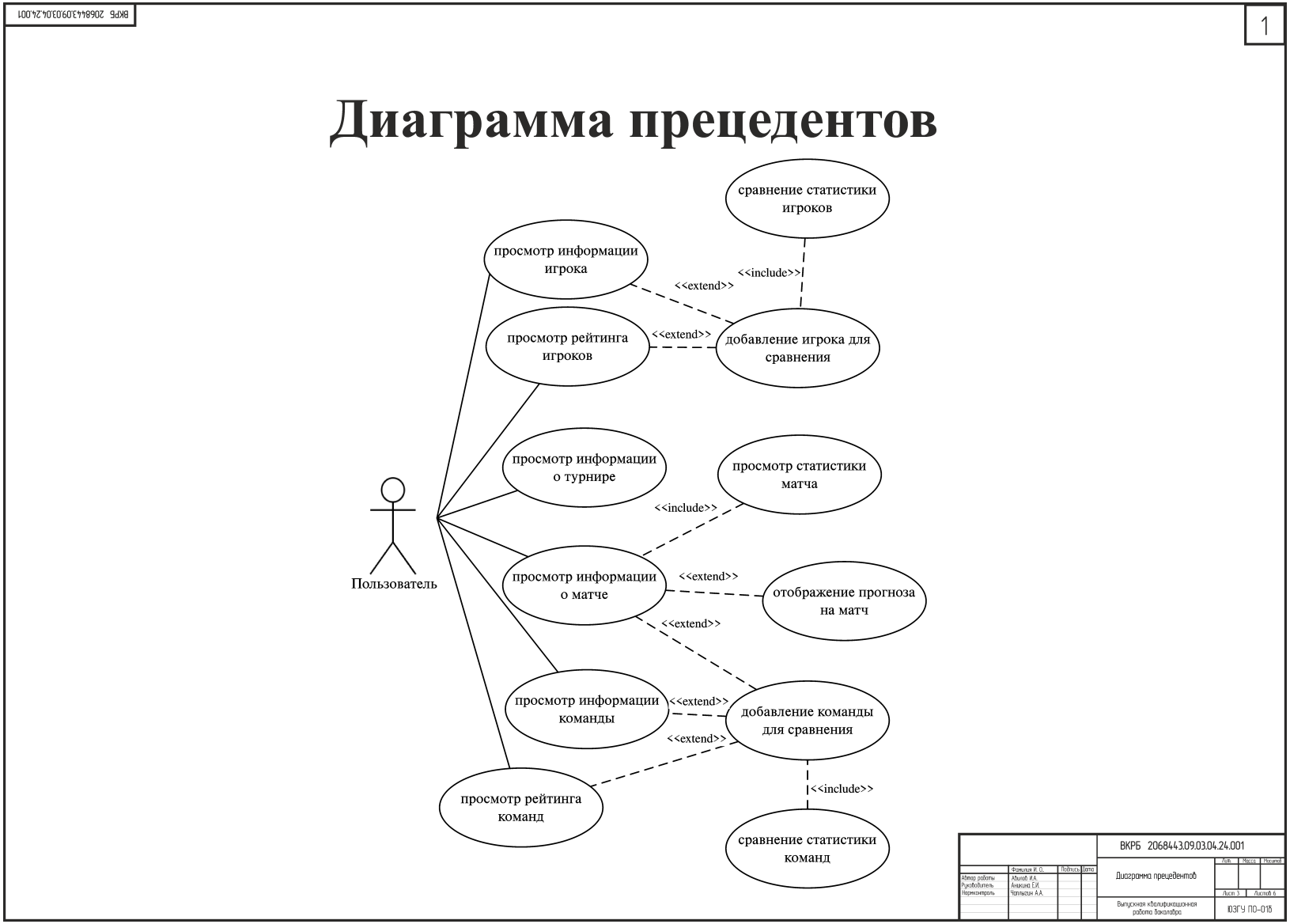

Рисунок А.3 – Диаграмма прецедентов

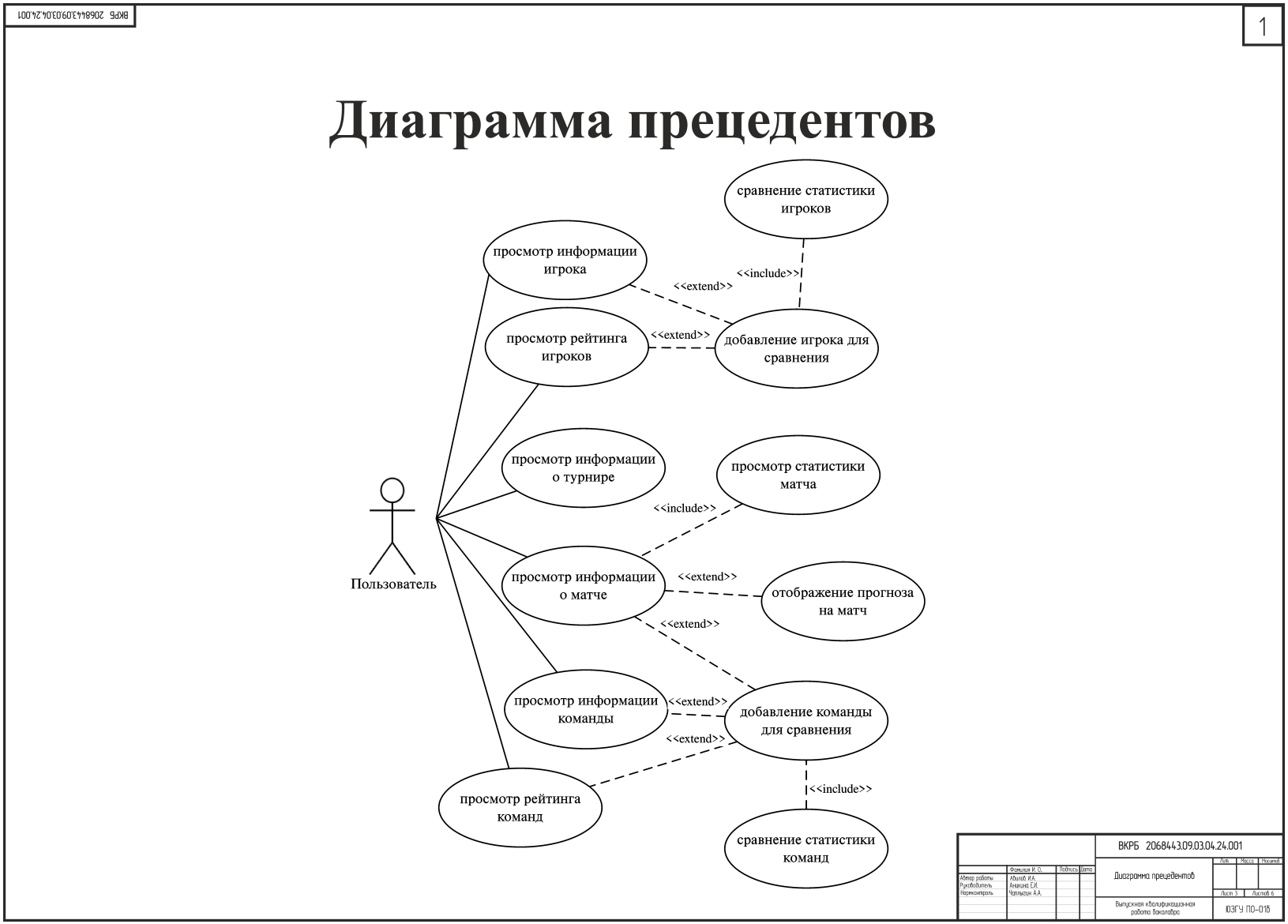

Рисунок А.4 – Концептуальные классы веб-платформы

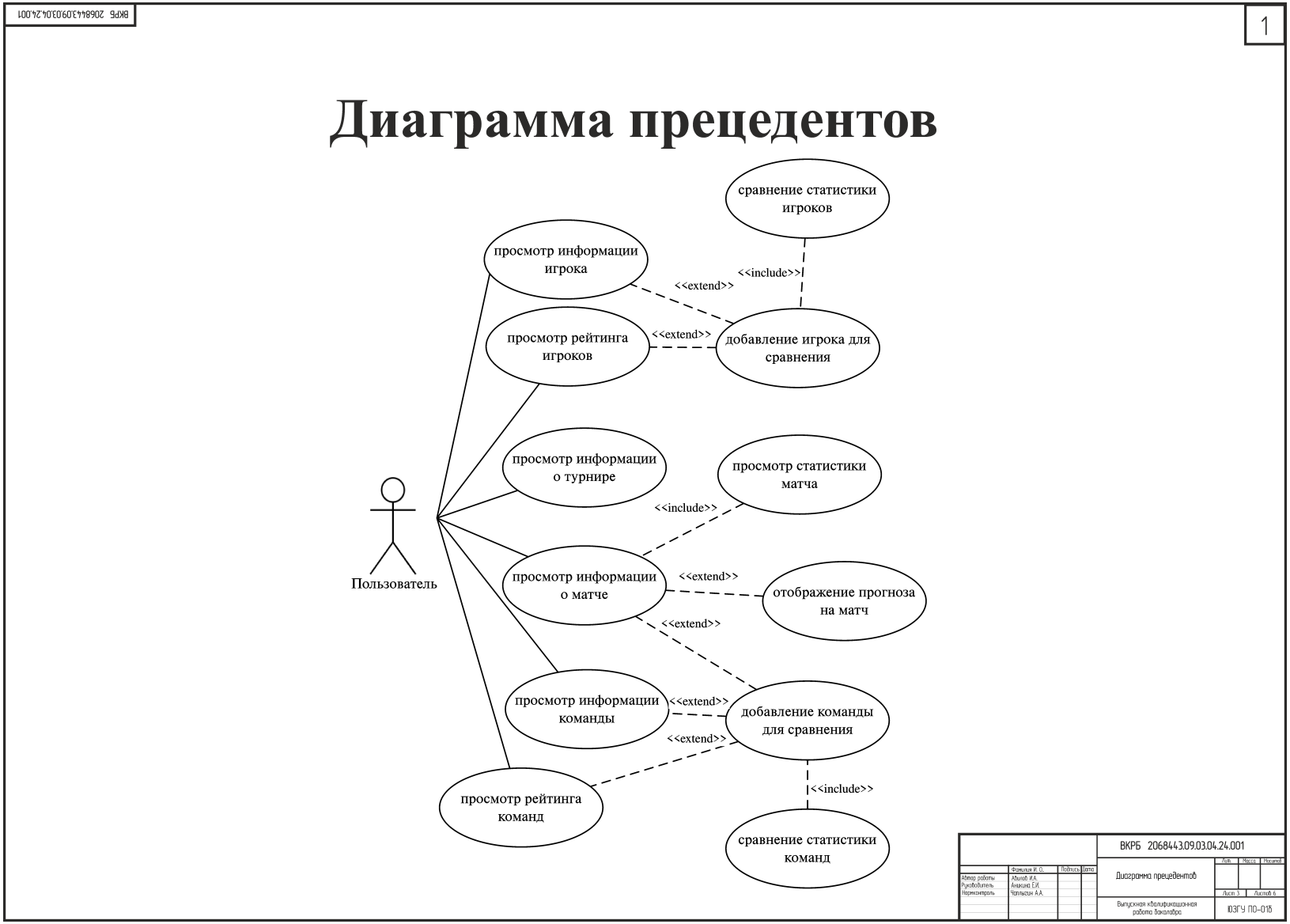

Рисунок А.5 – Архитектура программной системы

Заключение

В данной выпускной квалификационной работе был разработан и внедрен комплексный инструмент для анализа и визуализации статистических данных киберспортивной игры Counter-Strike 2 (CS2). Данный проект направлен на улучшение игровых навыков и стратегий киберспортивных команд путем предоставления глубокого анализа игровых данных и прогнозов.

Основные результаты работы:

1. Проведен анализ предметной области киберспорта и выявлены ключевые метрики, необходимые для анализа и прогнозирования игровых результатов.

2. Разработана и реализована база данных для хранения информации о матчах, игроках, командах и турнирной статистике.

3. Создана нейронная сеть для прогнозирования исходов матчей на основе исторических данных и текущих игровых метрик.

4. Реализованы удобные интерфейсы для отображения статистики и прогнозов на веб-платформе, обеспечивая интуитивно понятный доступ к информации.

5. Осуществлен анализ метрик из .dem файлов игр для получения детальных данных о каждом матче.

6. Интегрированы внешние API для автоматического обновления данных о турнирах, матчах и игроках, что обеспечивает актуальность и точность информации.

7. Использованы технологии контейнеризации и оркестрации с Docker и Kubernetes для обеспечения высокой производительности и масштабируемости системы.

8. Внедрен брокер сообщений RabbitMQ для управления потоками данных и коммуникацией между микросервисами.

9. Проведены интеграционные и нагрузочные тестирования, которые подтвердили стабильную работу платформы под высокой нагрузкой.

Рисунок А.6 – Заключение

ПРИЛОЖЕНИЕ Б

Фрагменты исходного кода программы

app.py

```
1 from flask import Flask, request, jsonify
2 from flask_migrate import Migrate
3 from model.db_models import db
4 from services.prediction_service import PredictionService
5
6 app = Flask(__name__)
7 app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://postgres:11111111
    @localhost:5432/CS2'
8 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
9
10 db.init_app(app)
11 migrate = Migrate(app, db)
12 prediction_service = PredictionService(app)
13
14 @app.route('/predict', methods=['POST'])
15 def predict():
16     data = request.get_json()
17     match_id = data.get('matchId')
18
19     if match_id is None:
20         return jsonify({'error': 'Invalid input, matchId is required'}), 400
21
22     try:
23         prediction = prediction_service.predict(match_id)
24         return jsonify(prediction)
25     except Exception as e:
26         return jsonify({'error': str(e)}), 500
27
28 if __name__ == '__main__':
29     app.run(host='0.0.0.0', port=5000)
```

main.go

```
1 package main
2
3 import (
4   "crypto/rand"
5   "encoding/hex"
6   "fmt"
7   "io"
8   "log"
```

```
9   "net/http"
10  "os"
11  "path/filepath"
12  "strconv"
13
14  "github.com/Fresh-vano/CS2-Parser/Export"
15  Models "github.com/Fresh-vano/CS2-Parser/Models"
16  Analyzer "github.com/Fresh-vano/CS2-Parser/analyzer"
17  "gorm.io/driver/postgres"
18  "gorm.io/gorm"
19 )
20
21 func main() {
22  http.HandleFunc("/upload", uploadFileHandler)
23  http.HandleFunc("/analyze", analyzeFileHandler)
24  log.Fatal(http.ListenAndServe(":8080", nil))
25 }
26
27 func uploadFileHandler(w http.ResponseWriter, r *http.Request) {
28  if r.Method != "POST" {
29    http.Error(w, "Only POST method is allowed", http.StatusMethodNotAllowed)
30    return
31  }
32
33  r.ParseMultipartForm(1024 << 20)
34  file, handler, err := r.FormFile("uploadfile")
35  if err != nil {
36    log.Println("Error Retrieving the File")
37    log.Println(err)
38    http.Error(w, "Error retrieving the file", http.StatusBadRequest)
39    return
40  }
41  defer file.Close()
42
43  randomBytes := make([]byte, 16)
44  rand.Read(randomBytes)
45  uniqueFileName := hex.EncodeToString(randomBytes) + filepath.Ext(handler.
      Filename)
46
47  tempFile, err := os.CreateTemp("uploads", "upload-*"+filepath.Ext(handler.
      Filename))
48  if err != nil {
49    log.Println(err)
50    http.Error(w, "Failed to create temp file", http.
        StatusInternalServerError)
```

```
51      return
52    }
53    defer tempFile.Close()
54
55    fileBytes, err := io.ReadAll(file)
56    if err != nil {
57      log.Println(err)
58      http.Error(w, "Failed to read file data", http.StatusInternalServerError)
59      return
60    }
61    tempFile.Write(fileBytes)
62
63    fmt.Fprintf(w, "File uploaded successfully: %s", uniqueFileName)
64  }
65
66  func analyzeFileHandler(w http.ResponseWriter, r *http.Request) {
67    if r.Method != "POST" {
68      http.Error(w, "Only POST method is allowed", http.StatusMethodNotAllowed)
69      return
70    }
71
72    r.ParseForm()
73    tournamentID := r.FormValue("tournament_id")
74    stageID := r.FormValue("stage_id")
75    team1ID := r.FormValue("team1_id")
76    team2ID := r.FormValue("team2_id")
77    fileName := r.FormValue("file_name")
78    mapId := r.FormValue("map_id")
79
80    dsn := "host=localhost user=postgres password=11111111 dbname=CS2 port=5432
        sslmode=disable"
81    db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
82    if err != nil {
83      log.Fatalf("Failed to connect to database: %v", err)
84    }
85
86    err = analyzeAndSaveToDB(db, tournamentID, stageID, team1ID, team2ID, mapId
        , fileName)
87    if err != nil {
88      http.Error(w, "Failed to analyze and save results", http.
          StatusInternalServerError)
89      return
90    }
91
```

```
92   fmt.Fprintf(w, "Analysis completed for file %s with tournament ID %s, stage
         ID %s, team1 ID %s, team2 ID %s", fileName, tournamentID, stageID,
       team1ID, team2ID)
93 }
94
95 func analyzeAndSaveToDB(db *gorm.DB, tournamentID, stageID, team1ID, team2ID,
       mapId, fileName string) error {
96   fullPath := filepath.Join("uploads", fileName)
97
98   demo := &Analyzer.Demo{
99     Team1: &Models.Team{},
100    Team2: &Models.Team{},
101  }
102
103  err := Analyzer.AnalyzeDemo(fullPath, demo)
104  if err != nil {
105    log.Printf("Error during demo analysis: %v\n", err)
106    return err
107  }
108
109  mapid, _ := strconv.Atoi(mapId)
110  team1id, _ := strconv.Atoi(team1ID)
111
112  playersSheet := Export.PlayersSheet{}
113  playersSheet.Demo = demo
114  playersData := playersSheet.Generate(mapid, team1id)
115
116  for item := range playersData {
117    result := db.Create(item)
118    if result.Error != nil {
119      log.Printf("Error while saving to database: %v\n", err)
120      return err
121    }
122  }
123
124  var temp = Export.TeamsSheet{}
125  teamsSheet := temp.NewTeamsSheet()
126  teamsData := teamsSheet.AddDemo(*demo, mapid, team1id)
127
128  for _, item := range teamsData {
129    result := db.Create(item)
130    if result.Error != nil {
131      log.Printf("Error while saving to database: %v\n", err)
132      return err
133    }
```

```
134   }
135
136   log.Println("Analysis completed and data saved to database successfully.")
137   return nil
138 }
```

ocelot.json

```
1 {
2   "ReRoutes": [
3     {
4       "DownstreamPathTemplate": "/api/data/{everything}",
5       "DownstreamScheme": "https",
6       "DownstreamHostAndPorts": [
7         {
8           "Host": "localhost",
9           "Port": 5001
10        }
11      ],
12      "UpstreamPathTemplate": "/api/data/{everything}",
13      "UpstreamHttpMethod": [ "Get", "Post", "Put", "Delete" ],
14      "FileCacheOptions":  { "TtlSeconds": 30 }
15    },
16    {
17      "DownstreamPathTemplate": "/api/prediction/{everything}",
18      "DownstreamScheme": "https",
19      "DownstreamHostAndPorts": [
20        {
21          "Host": "localhost",
22          "Port": 5002
23        }
24      ],
25      "UpstreamPathTemplate": "/api/prediction/{everything}",
26      "UpstreamHttpMethod": [ "Get", "Post" ],
27      "FileCacheOptions": { "TtlSeconds": 30 }
28    },
29    {
30      "DownstreamPathTemplate": "/api/search/{everything}",
31      "DownstreamScheme": "https",
32      "DownstreamHostAndPorts": [
33        {
34          "Host": "localhost",
35          "Port": 5003
36        }
37      ],
38      "UpstreamPathTemplate": "/api/search/{everything}",
```

```
39        "UpstreamHttpMethod": [ "Get" ],
40        "FileCacheOptions": { "TtlSeconds": 30 }
41      },
42      {
43        "DownstreamPathTemplate": "/api/analyze/{everything}",
44        "DownstreamScheme": "https",
45        "DownstreamHostAndPorts": [
46          {
47            "Host": "localhost",
48            "Port": 5004
49          }
50        ],
51        "UpstreamPathTemplate": "/api/analyze/{everything}",
52        "UpstreamHttpMethod": [ "Get", "Post" ],
53        "FileCacheOptions": { "TtlSeconds": 30 }
54      },
55      {
56        "DownstreamPathTemplate": "/api/background/{everything}",
57        "DownstreamScheme": "https",
58        "DownstreamHostAndPorts": [
59          {
60            "Host": "localhost",
61            "Port": 5005
62          }
63        ],
64        "UpstreamPathTemplate": "/api/background/{everything}",
65        "UpstreamHttpMethod": [ "Get" ],
66        "FileCacheOptions": { "TtlSeconds": 30 }
67      }
68    ],
69    "GlobalConfiguration": {
70      "BaseUrl": "http://localhost:5000"
71    }
72  }
```

Program.cs

```
1 using Microsoft.Extensions.DependencyInjection;
2 using Ocelot.Cache;
3 using Ocelot.Cache.CacheManager;
4 using Ocelot.DependencyInjection;
5 using Ocelot.Middleware;
6 using Prometheus;
7
8 var builder = WebApplication.CreateBuilder(args);
9
```

```
10 builder.Services.AddOcelot().AddCacheManager(x =>
11 {
12   x.WithDictionaryHandle();
13 });
14
15 builder.Services.AddPrometheusHttpClientMetrics();
16 builder.Services.AddPrometheusCounters();
17
18
19 builder.Services.AddControllers();
20 builder.Services.AddEndpointsApiExplorer();
21 builder.Services.AddSwaggerGen();
22
23 var app = builder.Build();
24
25 // Use Prometheus metrics
26 app.UseHttpMetrics();
27
28 // Configure the HTTP request pipeline.
29 if (app.Environment.IsDevelopment())
30 {
31   app.UseSwagger();
32   app.UseSwaggerUI();
33 }
34
35 app.UseHttpsRedirection();
36 app.UseAuthorization();
37 app.MapControllers();
38
39 // Use Ocelot middleware
40 app.UseOcelot().Wait();
41
42 // Map Prometheus metrics endpoint
43 app.UseEndpoints(endpoints =>
44 {
45   endpoints.MapMetrics();
46 });
47
48 app.Run();
```

MatchController.cs

```
1 using DataAPI.Data;
2 using Microsoft.AspNetCore.Mvc;
3
4 namespace DataAPI.Controllers
```

```
5 {
6   [Route("api/data/match")]
7   public class MatchController : ControllerBase
8   {
9     private readonly Cs2Context _context;
10
11    public MatchController(Cs2Context context)
12    {
13      _context = context;
14    }
15
16    [HttpGet("finished")]
17    public IActionResult GetFinishedMatches()
18    {
19      var matches = _context.Matches
20                  .Where(m => m.EndDate < DateTime.UtcNow)
21                  .ToList();
22      return Ok(new { matches });
23    }
24
25    [HttpGet("current")]
26    public IActionResult GetCurrentMatches()
27    {
28      var matches = _context.Matches
29                  .Where(m => m.StartDate <= DateTime.UtcNow && m.EndDate >=
                       DateTime.UtcNow)
30                  .ToList();
31      return Ok(new { matches });
32    }
33
34    [HttpGet("{slug}")]
35    public IActionResult GetMatchBySlug(string slug)
36    {
37      var match = _context.Matches
38                .FirstOrDefault(m => m.Slug == slug);
39      if (match == null) return NotFound();
40      return Ok(new { match });
41    }
42
43    [HttpGet("{slug}/stats")]
44    public IActionResult GetMatchStats(string slug)
45    {
46      // Placeholder: Implement statistics retrieval logic
47      return Ok(new { statistics = "Match statistics data" });
48    }
```

```
49
50      [HttpGet("{slug}/stats/{mapName}")]
51      public IActionResult GetMatchStatsByMap(string slug, string mapName)
52      {
53        // Placeholder: Implement map-specific statistics retrieval logic
54        return Ok(new { statistics = "Map-specific statistics data" });
55      }
56    }
57  }
```

PlayerController.cs

```
1  using DataAPI.Data;
2  using Microsoft.AspNetCore.Mvc;
3
4  namespace DataAPI.Controllers
5  {
6    [Route("api/data/player")]
7    public class PlayerController : ControllerBase
8    {
9      private readonly Cs2Context _context;
10
11     public PlayerController(Cs2Context context)
12     {
13       _context = context;
14     }
15
16     [HttpGet]
17     public IActionResult GetAllPlayers()
18     {
19       var players = _context.Players.ToList();
20       return Ok(new { players });
21     }
22
23     [HttpGet("{slug}")]
24     public IActionResult GetPlayerBySlug(string slug)
25     {
26       var player = _context.Players.FirstOrDefault(p => p.Slug == slug);
27       if (player == null) return NotFound();
28       return Ok(new { player });
29     }
30   }
31 }
```

TeamController.cs

```
1  using DataAPI.Data;
```

```
using Microsoft.AspNetCore.Mvc;

namespace DataAPI.Controllers
{
  [Route("api/data/team")]
  public class TeamController : ControllerBase
  {
    private readonly Cs2Context _context;

    public TeamController(Cs2Context context)
    {
      _context = context;
    }

    [HttpGet]
    public IActionResult GetAllTeams()
    {
      var teams = _context.Teams.ToList();
      return Ok(new { teams });
    }

    [HttpGet("{slug}")]
    public IActionResult GetTeamBySlug(string slug)
    {
      var team = _context.Teams.FirstOrDefault(t => t.Slug == slug);
      if (team == null) return NotFound();
      return Ok(new { team });
    }
  }
}
```

TournamentController.cs

```
using DataAPI.Data;
using Microsoft.AspNetCore.Mvc;

namespace DataAPI.Controllers
{
  [Route("api/data/tournament")]
  public class TournamentController : ControllerBase
  {
    private readonly Cs2Context _context;

    public TournamentController(Cs2Context context)
    {
      _context = context;
```

```
14      }
15
16      [HttpGet("current")]
17      public IActionResult GetCurrentTournaments()
18      {
19        var tournaments = _context.Tournaments.Where(t => t.StartDate >=
            DateTime.UtcNow).ToList();
20        return Ok(new { tournaments });
21      }
22
23      [HttpGet("finished")]
24      public IActionResult GetFinishedTournaments()
25      {
26        var tournaments = _context.Tournaments.Where(t => t.EndDate < DateTime.
            UtcNow).ToList();
27        return Ok(new { tournaments });
28      }
29
30      [HttpGet("{slug}")]
31      public IActionResult GetTournamentBySlug(string slug)
32      {
33        var tournament = _context.Tournaments.FirstOrDefault(t => t.Slug ==
            slug);
34
35        if (tournament == null)
36          return NotFound();
37
38        return Ok(new { tournament });
39      }
40    }
41  }
```

BackgroundTaskQueue.cs

```
1  using System.Collections.Concurrent;
2
3  namespace DataParser.BackgroundService
4  {
5    public class BackgroundTaskQueue : IBackgroundTaskQueue
6    {
7      private ConcurrentQueue<Func<CancellationToken, Task>> _workItems = new
         ConcurrentQueue<Func<CancellationToken, Task>>();
8      private SemaphoreSlim _signal = new SemaphoreSlim(0);
9
10     public void QueueBackgroundWorkItem(Func<CancellationToken, Task>
         workItem)
```

```
11      {
12        if (workItem == null)
13        {
14          throw new ArgumentNullException(nameof(workItem));
15        }
16
17        _workItems.Enqueue(workItem);
18        _signal.Release();
19      }
20
21      public async Task<Func<CancellationToken, Task>> DequeueAsync(
           CancellationToken cancellationToken)
22      {
23        await _signal.WaitAsync(cancellationToken);
24        _workItems.TryDequeue(out var workItem);
25
26        return workItem;
27      }
28    }
29
30  }
```

Worker.cs

```
1  using DataParser.BackgroundService;
2
3  public class Worker : Microsoft.Extensions.Hosting.BackgroundService
4  {
5    private readonly IBackgroundTaskQueue _taskQueue;
6
7    public Worker(IBackgroundTaskQueue taskQueue)
8    {
9      _taskQueue = taskQueue;
10   }
11
12   protected override async Task ExecuteAsync(CancellationToken stoppingToken)
13   {
14     while (!stoppingToken.IsCancellationRequested)
15     {
16       var workItem = await _taskQueue.DequeueAsync(stoppingToken);
17
18       try
19       {
20         await workItem(stoppingToken);
21       }
22       catch (Exception ex)
```

```
23        {
24        }
25      }
26    }
27  }
```

BackgroundServiceController.cs

```
1  using Microsoft.Extensions.DependencyInjection;
2  using Microsoft.AspNetCore.Mvc;
3  using DataParser.BackgroundService;
4  using DataParser.Services;
5  using System;
6  using System.Collections.Generic;
7  using System.Threading;
8  using System.Threading.Tasks;
9
10 [ApiController]
11 [Route("api/[controller]")]
12 public class BackgroundServiceController : ControllerBase
13 {
14   private readonly IBackgroundTaskQueue _taskQueue;
15   private readonly IServiceScopeFactory _serviceScopeFactory;
16   private readonly IDictionary<string, CancellationTokenSource> _tokens;
17
18   public BackgroundServiceController(
19     IBackgroundTaskQueue taskQueue,
20     IServiceScopeFactory serviceScopeFactory,
21     IDictionary<string, CancellationTokenSource> tokens)
22   {
23     _taskQueue = taskQueue;
24     _serviceScopeFactory = serviceScopeFactory;
25     _tokens = tokens;
26   }
27
28   [HttpPost("start-team-update")]
29   public IActionResult StartTeamUpdateService()
30   {
31     return StartService("TeamUpdateService", async (serviceProvider, token)
         =>
32     {
33       var teamUpdateService = serviceProvider.GetRequiredService<
           ITeamUpdateService>();
34       await teamUpdateService.UpdateTeamsAsync();
35     });
36   }
```

```
37
38  [HttpPost("start-tournament-update")]
39  public IActionResult StartTournamentUpdateService()
40  {
41    return StartService("TournamentUpdateService", async (serviceProvider,
        token) =>
42    {
43      var tournamentUpdateService = serviceProvider.GetRequiredService<
          ITournamentUpdateService>();
44      await tournamentUpdateService.UpdateTournamentAsync();
45    });
46  }
47
48  [HttpPost("start-match-update")]
49  public IActionResult StartMatchUpdateService()
50  {
51    return StartService("MatchUpdateService", async (serviceProvider, token)
        =>
52    {
53      var matchUpdateService = serviceProvider.GetRequiredService<
          IMatchUpdateService>();
54      await matchUpdateService.UpdateMatchAsync();
55    });
56  }
57
58  [HttpPost("stop/{serviceName}")]
59  public IActionResult StopService(string serviceName)
60  {
61    if (_tokens.TryGetValue(serviceName, out var tokenSource))
62    {
63      tokenSource.Cancel();
64      _tokens.Remove(serviceName);
65      return Ok($"Background service {serviceName} stopped");
66    }
67
68    return NotFound($"Service {serviceName} not found");
69  }
70
71  private IActionResult StartService(string serviceName, Func<
      IServiceProvider, CancellationToken, Task> work)
72  {
73    var tokenSource = new CancellationTokenSource();
74    if (!_tokens.TryAdd(serviceName, tokenSource))
75    {
76      return BadRequest($"Service {serviceName} is already running.");
```

```
77      }
78
79      _taskQueue.QueueBackgroundWorkItem(async token =>
80      {
81        while (!token.IsCancellationRequested)
82        {
83          using (var scope = _serviceScopeFactory.CreateScope())
84          {
85            try
86            {
87              await work(scope.ServiceProvider, token);
88            }
89            catch (Exception ex)
90            {
91            }
92          }
93
94          try
95          {
96            await Task.Delay(TimeSpan.FromHours(1), token);
97          }
98          catch (TaskCanceledException)
99          {
100           break;
101         }
102       }
103     });
104
105     return Ok($"Background service {serviceName} started");
106   }
107 }
```

TeamUpdateService.cs

```
1 using AutoMapper;
2 using DataParser.Data;
3 using DataParser.DTOs;
4 using DataParser.Models;
5 using Microsoft.EntityFrameworkCore;
6 using Newtonsoft.Json;
7
8 namespace DataParser.Services
9 {
10   public class TeamUpdateService : ITeamUpdateService
11   {
12     static readonly HttpClient client = new HttpClient();
```

```
13      private int offset = 0;
14      private readonly int limit = 30;
15      private readonly IMapper _mapper;
16      private readonly IApplicationDbContext _dbContext;
17      private int count = 30;
18
19      public TeamUpdateService(IMapper mapper, IApplicationDbContext dbContext)
20      {
21        _mapper = mapper;
22        _dbContext = dbContext;
23      }
24
25      public async Task UpdateTeamsAsync()
26      {
27        for (offset = 0; offset < count; offset += limit)
28        {
29          try
30          {
31            await Console.Out.WriteLineAsync($"
                ========================================           {offset}");
32
33            var url = $"https://api.bo3.gg/api/v1/teams/rankings/earn?" +
34                  $"page[offset]={offset}&page[limit]={limit}&" +
35                  $"sort=rank&filter[current][eq]=true&with=team,team_roster";
36
37            HttpClientHelper.ConfigureClient(client, $"https://bo3.gg/teams/
                earnings?page={offset / limit + 1}");
38
39            var response = await client.GetAsync(url);
40            response.EnsureSuccessStatusCode();
41            var responseBody = await response.Content.ReadAsStringAsync();
42
43            var teamDtoData = JsonConvert.DeserializeObject<TeamGeneralDTO>(
                responseBody);
44
45            count = teamDtoData.Total.Count;
46
47            foreach (var teamRankDto in teamDtoData.TeamRanks)
48            {
49              var country = _mapper.Map<Country>(teamRankDto.Team.Country);
50
51              if (country != null)
52                SaveToDatabase(country);
53
54              var team = _mapper.Map<Team>(teamRankDto.Team);
```

```
55            if (team == null)
56              continue;
57            var existingTeam = _dbContext.Teams.FirstOrDefault(t => t.Id ==
                team.Id);
58            if (existingTeam == default)
59            {
60              await Console.Out.WriteLineAsync($"Adding team: {team.Name}");
61
62              team.CreatedAt = DateTimeOffset.UtcNow;
63              team.Country = null;
64              team.Players = new List<Player>();
65
66              _dbContext.Teams.Add(team);
67              _dbContext.SaveChanges();
68            }
69
70
71            foreach (var roster in teamRankDto.TeamRoster.Players)
72            {
73              var player = _mapper.Map<Player>(roster);
74              if (player == null)
75                continue;
76              var existingPlayer = _dbContext.Players.FirstOrDefault(t => t.
                  Id == player.Id);
77
78              if (player.Country != null)
79                SaveToDatabase(player.Country);
80
81              if (existingPlayer == default)
82              {
83                await Console.Out.WriteLineAsync($"Adding player: {player.
                    Nickname}");
84
85                player.CreatedAt = DateTimeOffset.UtcNow;
86                player.Country = null;
87                player.Team = null;
88
89                if (player.TeamId != null)
90                {
91                  if (player.TeamId == team.Id)
92                  {
93                    if (team.Players == null)
94                      team.Players = new List<Player>();
95
96                    team.Players.Add(player);
```

```
97                  }
98                  else
99                  {
100                   var existingPlayerTeam = _dbContext.Teams.FirstOrDefault(
                          t => t.Id == player.TeamId);
101                   if (existingPlayerTeam != default)
102                   {
103                     if (existingPlayerTeam.Players == null)
104                       existingPlayerTeam.Players = new List<Player>();
105
106                     existingPlayerTeam.Players.Add(player);
107                     _dbContext.SaveChanges();
108                   }
109                   else
110                   {
111                     player.TeamId = null;
112                   }
113                 }
114               }
115
116               _dbContext.Players.Add(player);
117             }
118           }
119           _dbContext.SaveChanges();
120         }
121
122         await Task.Delay(TimeSpan.FromSeconds(4));
123       }
124       catch (HttpRequestException e)
125       {
126         await Console.Out.WriteLineAsync("\nException Caught!");
127         await Console.Out.WriteLineAsync($"Message :{e.Message} ");
128       }
129       catch (Exception ex)
130       {
131         await Console.Out.WriteLineAsync("\nException Caught!");
132         await Console.Out.WriteLineAsync($"Message :{ex.Message} ");
133       }
134     }
135
136     await UpdateTeamNameAsync();
137   }
138
139
140   private void SaveToDatabase(Country country)
```

```
141     {
142       var existingCountry = _dbContext.Countries.FirstOrDefault(t => t.Id ==
            country.Id);
143       if (existingCountry == default)
144       {
145         _dbContext.Countries.Add(country);
146       }
147
148       _dbContext.SaveChanges();
149     }
150
151     public async Task UpdateTeamNameAsync()
152     {
153       var teams = _dbContext.Teams.ToList();
154
155       foreach (Team team in teams)
156       {
157         try
158         {
159           await Console.Out.WriteLineAsync($"
              ========================================           TEAM {team.
              Name}");
160
161           var url = $"https://api.bo3.gg/api/v1/teams/{team.Slug}";
162
163           HttpClientHelper.ConfigureClient(client, $"https://bo3.gg/ru/teams
              /{team.Slug}");
164
165           var response = await client.GetAsync(url);
166           response.EnsureSuccessStatusCode();
167           var responseBody = await response.Content.ReadAsStringAsync();
168
169           List<TeamClanDto> teamClans = JsonConvert.DeserializeObject<List<
              TeamClanDto>>(JsonConvert.SerializeObject(JsonConvert.
              DeserializeObject<dynamic>(responseBody).team_clans));
170
171           foreach (var teamClan in teamClans)
172           {
173             TeamName teamName = _mapper.Map<TeamName>(teamClan);
174
175             var existingTeamName = _dbContext.TeamNames.FirstOrDefault(tn =>
                tn.Id == teamName.Id);
176             if (existingTeamName == default)
177             {
178               _dbContext.TeamNames.Add(teamName);
```

```
179             }
180           }
181
182           _dbContext.SaveChanges();
183           await Task.Delay(TimeSpan.FromSeconds(4));
184         }
185         catch (HttpRequestException e)
186         {
187           await Console.Out.WriteLineAsync("\nException Caught!");
188           await Console.Out.WriteLineAsync($"Message :{e.Message} ");
189         }
190         catch (Exception ex)
191         {
192           await Console.Out.WriteLineAsync("\nException Caught!");
193           await Console.Out.WriteLineAsync($"Message :{ex.Message} ");
194         }
195       }
196     }
197   }
198 }
```

SearchController.cs

```
1 using DataSearch.DTOs;
2 using DataSearch.Repository;
3 using Microsoft.AspNetCore.Mvc;
4
5 namespace DataSearch.Controllers
6 {
7   [Route("api/[controller]")]
8   public class SearchController : ControllerBase
9   {
10    private readonly IPlayerRepository _playerRepository;
11    private readonly ITeamRepository _teamRepository;
12    private readonly ITournamentRepository _tournamentRepository;
13
14    public SearchController(IPlayerRepository playerRepository,
          ITeamRepository teamRepository, ITournamentRepository
          tournamentRepository)
15    {
16      _playerRepository = playerRepository;
17      _teamRepository = teamRepository;
18      _tournamentRepository = tournamentRepository;
19    }
20
21    public SearchResult Search(string query)
```

```
22      {
23        var players = _playerRepository.SearchPlayers(query);
24        var teams = _teamRepository.SearchTeams(query);
25        var tournaments = _tournamentRepository.SearchTournaments(query);
26
27        return new SearchResult
28        {
29          Players = players,
30          Teams = teams,
31          Tournaments = tournaments
32        };
33      }
34    }
35  }
```

SearchController.cs

```
1  using DataSearch.DTOs;
2  using DataSearch.Repository;
3  using Microsoft.AspNetCore.Mvc;
4
5  namespace DataSearch.Controllers
6  {
7    [Route("api/[controller]")]
8    public class SearchController : ControllerBase
9    {
10     private readonly IPlayerRepository _playerRepository;
11     private readonly ITeamRepository _teamRepository;
12     private readonly ITournamentRepository _tournamentRepository;
13
14     public SearchController(IPlayerRepository playerRepository,
           ITeamRepository teamRepository, ITournamentRepository
           tournamentRepository)
15     {
16       _playerRepository = playerRepository;
17       _teamRepository = teamRepository;
18       _tournamentRepository = tournamentRepository;
19     }
20
21     public SearchResult Search(string query)
22     {
23       var players = _playerRepository.SearchPlayers(query);
24       var teams = _teamRepository.SearchTeams(query);
25       var tournaments = _tournamentRepository.SearchTournaments(query);
26
27       return new SearchResult
```

```
28        {
29          Players = players,
30          Teams = teams,
31          Tournaments = tournaments
32        };
33      }
34    }
35  }
```

docker-compose.yml

```
1  version: '3.8'
2
3  services:
4    api_gateway:
5      build:
6        context: ./DataParser/APIGateway
7      ports:
8        - "5000:5000"
9      depends_on:
10       - db
11       - redis
12       - rabbitmq
13     environment:
14       - ASPNETCORE_ENVIRONMENT=Development
15       - ConnectionStrings__DefaultConnection=Host=db; Port=5432; Database=CS
           2; Username=postgres; Password=11111111
16
17   data_api:
18     build:
19       context: ./DataParser/DataAPI
20     ports:
21       - "5001:5001"
22     depends_on:
23       - db
24       - redis
25       - rabbitmq
26     environment:
27       - ASPNETCORE_ENVIRONMENT=Development
28       - ConnectionStrings__DefaultConnection=Host=db; Port=5432; Database=CS
           2; Username=postgres; Password=11111111
29
30   predict_service:
31     build:
32       context: ./CS2Predict
33     ports:
```

```
34      - "5002:5002"
35    depends_on:
36      - db
37      - redis
38      - rabbitmq
39    environment:
40      - FLASK_ENV=development
41      - DATABASE_URL=postgresql://postgres:11111111@db:5432/CS2
42
43  data_search:
44    build:
45      context: ./DataParser/DataSearch
46    ports:
47      - "5003:5003"
48    depends_on:
49      - db
50      - redis
51      - rabbitmq
52    environment:
53      - ASPNETCORE_ENVIRONMENT=Development
54      - ConnectionStrings__DefaultConnection=Host=db; Port=5432; Database=CS
          2; Username=postgres; Password=11111111
55
56  dem_parser:
57    build:
58      context: ./CS2Parser
59    ports:
60      - "5004:5004"
61    depends_on:
62      - db
63      - redis
64      - rabbitmq
65    environment:
66      - ASPNETCORE_ENVIRONMENT=Development
67      - ConnectionStrings__DefaultConnection=Host=db; Port=5432; Database=CS
          2; Username=postgres; Password=11111111
68
69  tournament_service:
70    build:
71      context: ./DataParser/DataParser
72    ports:
73      - "5005:5005"
74    depends_on:
75      - db
76      - redis
```

```
77        - rabbitmq
78      environment:
79        - ASPNETCORE_ENVIRONMENT=Development
80        - ConnectionStrings__DefaultConnection=Host=db; Port=5432; Database=CS
            2; Username=postgres; Password=11111111
81
82    frontend:
83      build:
84        context: ./Frontend
85      ports:
86        - "3000:3000"
87      depends_on:
88        - api_gateway
89      environment:
90        - REACT_APP_API_URL=http://localhost:5000
91
92    db:
93      image: postgres:13
94      ports:
95        - "5432:5432"
96      environment:
97        POSTGRES_DB: CS2
98        POSTGRES_USER: postgres
99        POSTGRES_PASSWORD: 11111111
100     volumes:
101       - pgdata:/var/lib/postgresql/data
102
103   redis:
104     image: redis:6
105     ports:
106       - "6379:6379"
107
108   rabbitmq:
109     image: rabbitmq:3-management
110     ports:
111       - "5672:5672"
112       - "15672:15672"
113
114   prometheus:
115     image: prom/prometheus
116     ports:
117       - "9090:9090"
118     volumes:
119       - prometheus_data:/prometheus
120
```

```
volumes:
  pgdata:
  prometheus_data:
```

МЕСТО ДЛЯ ДИСКА