Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ОТЧЕТ

о преддипломной практике

наименование вида и типа практики

на (в) АО ”ДИАЙПИ”

наименование предприятия, организации, учреждения

Студента 4 курса, группы ПО-01б

курса, группы

Авилова Ивана Александровича

фамилия, имя, отчество

Руководитель практики от предприятия, организации, учреждения

Оценка _____

_____

должность, звание, степень

_____ _____

фамилия и. о. подпись, дата

Руководитель практики от университета

Оценка _____

_____

должность, звание, степень

_____ _____

фамилия и. о. подпись, дата

Члены комиссии

_____ _____

подпись, дата фамилия и. о.

_____ _____

подпись, дата фамилия и. о.

Курск, 2024 г.

СОДЕРЖАНИЕ

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 « » 2024 года № 0000-0 «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.

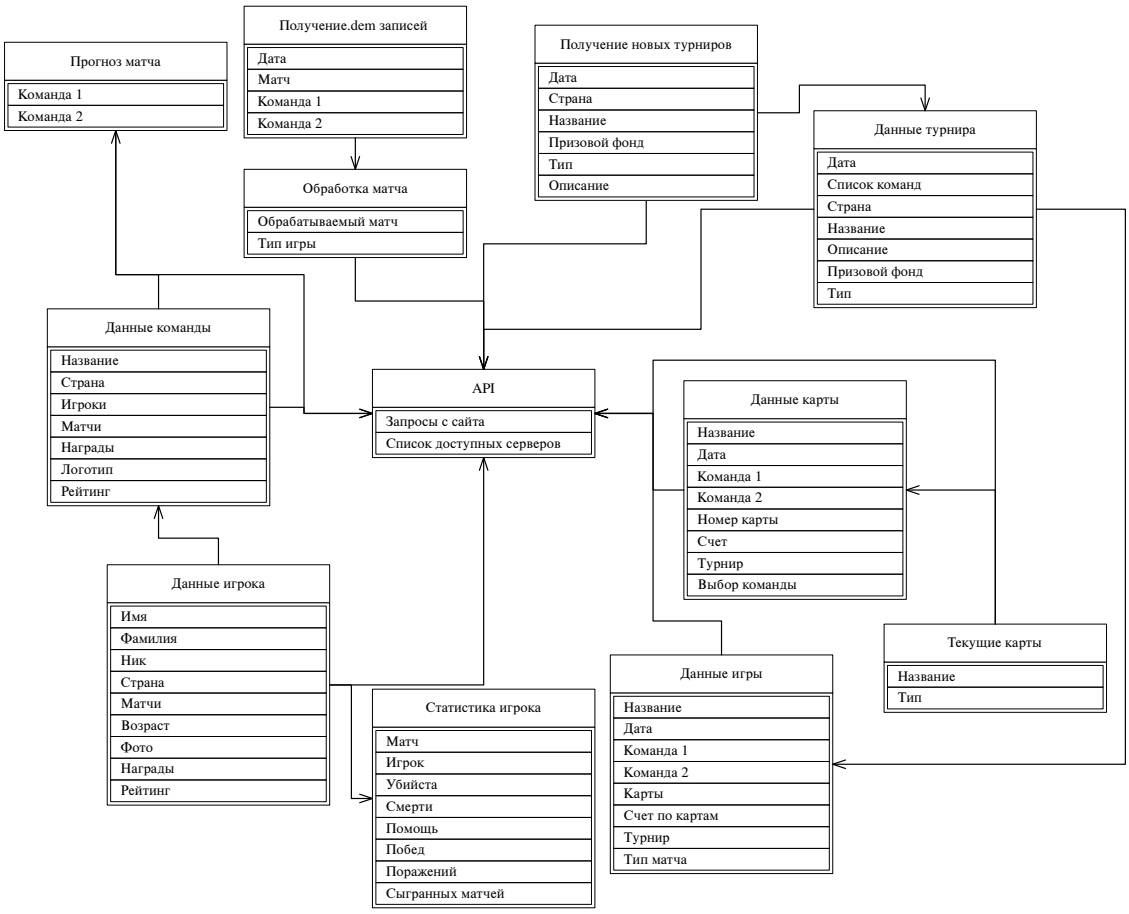

Рисунок 2.1 – Концептуальные классы backend части веб-платформы

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.

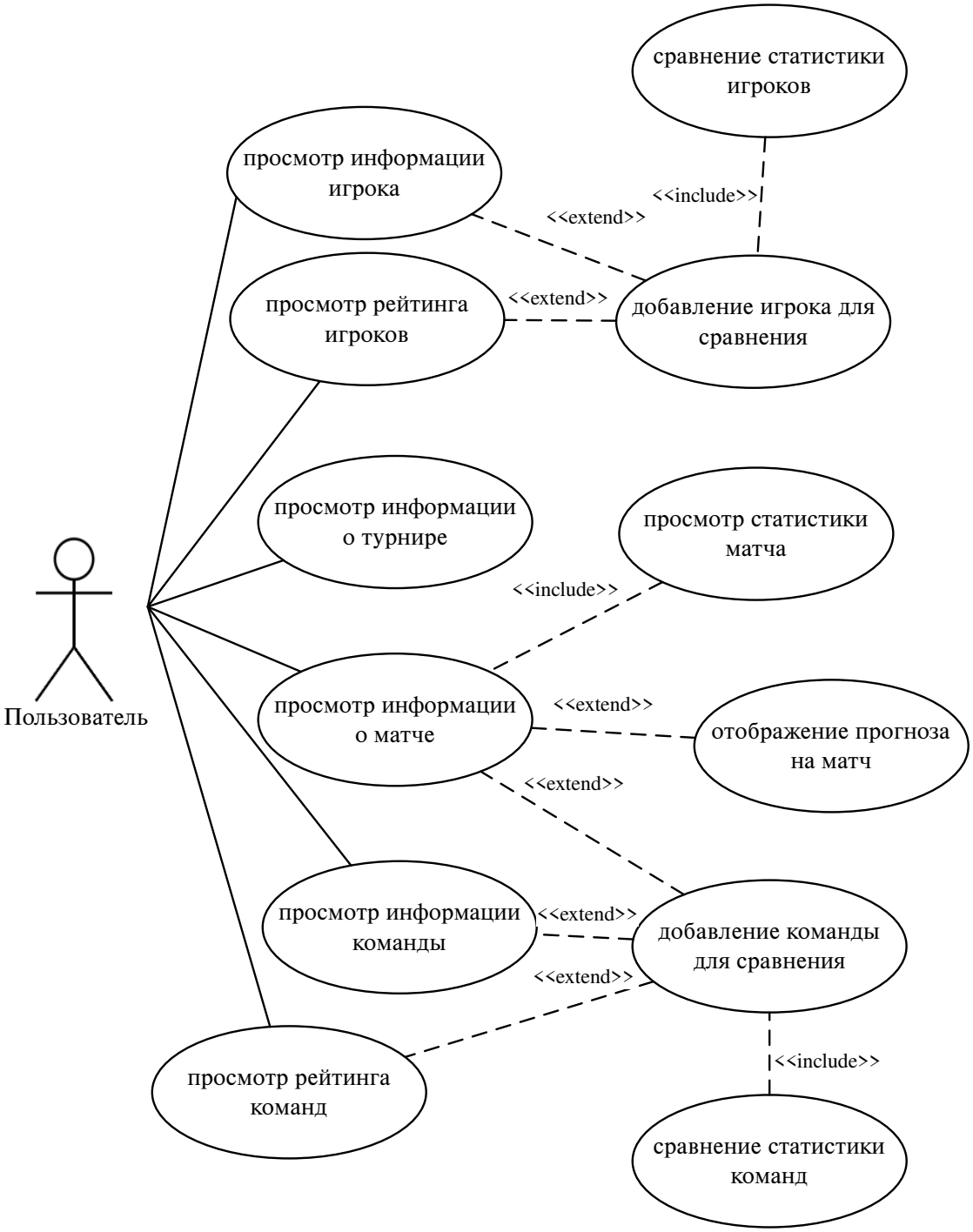

Рисунок 2.2 – Диаграмма прецедентов

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.

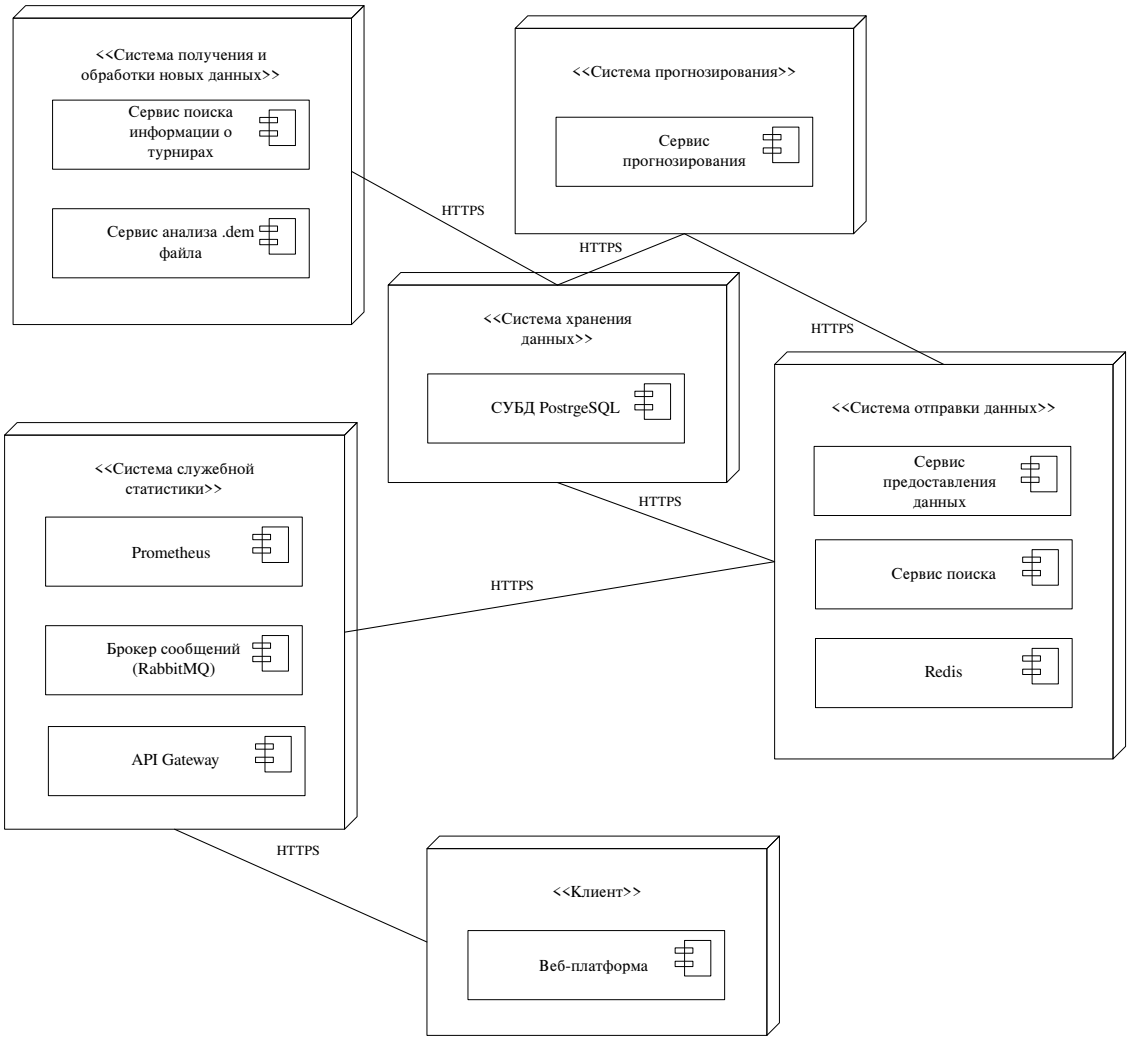

Рисунок 3.1 – Архитектура программной системы

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.

3.3 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 оцен-

ку 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 статистическими 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.

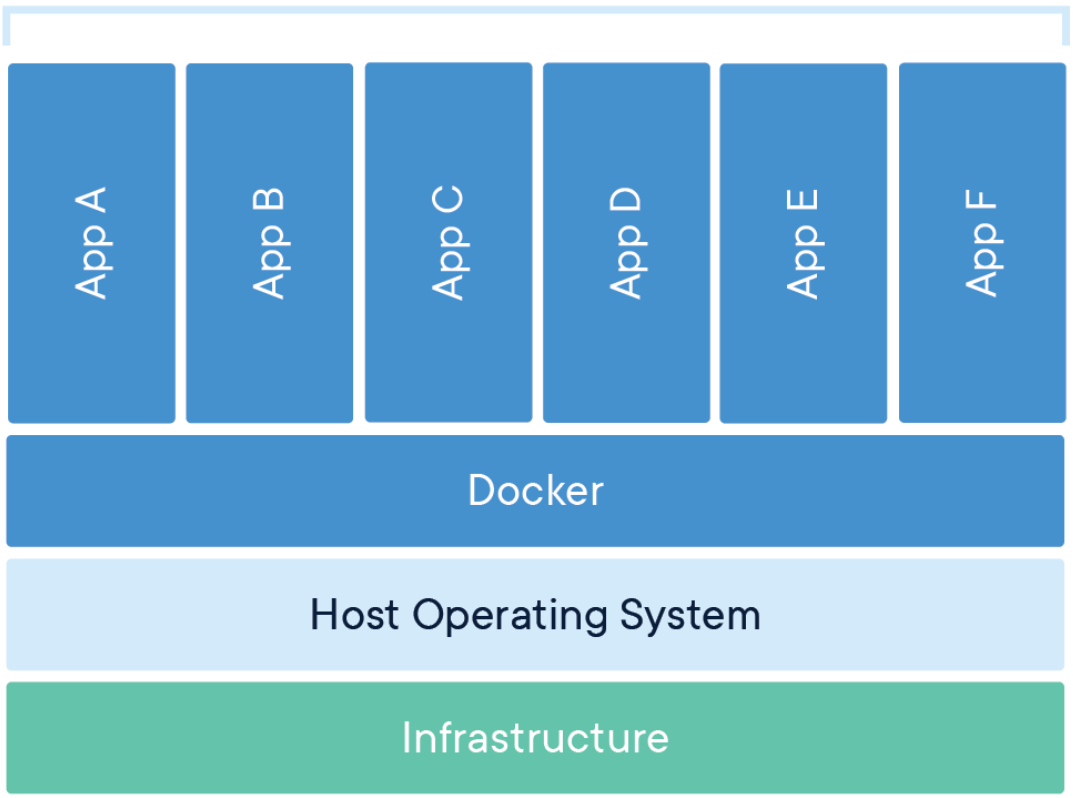

Рисунок 3.2 – Архитектура работы Docker

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.

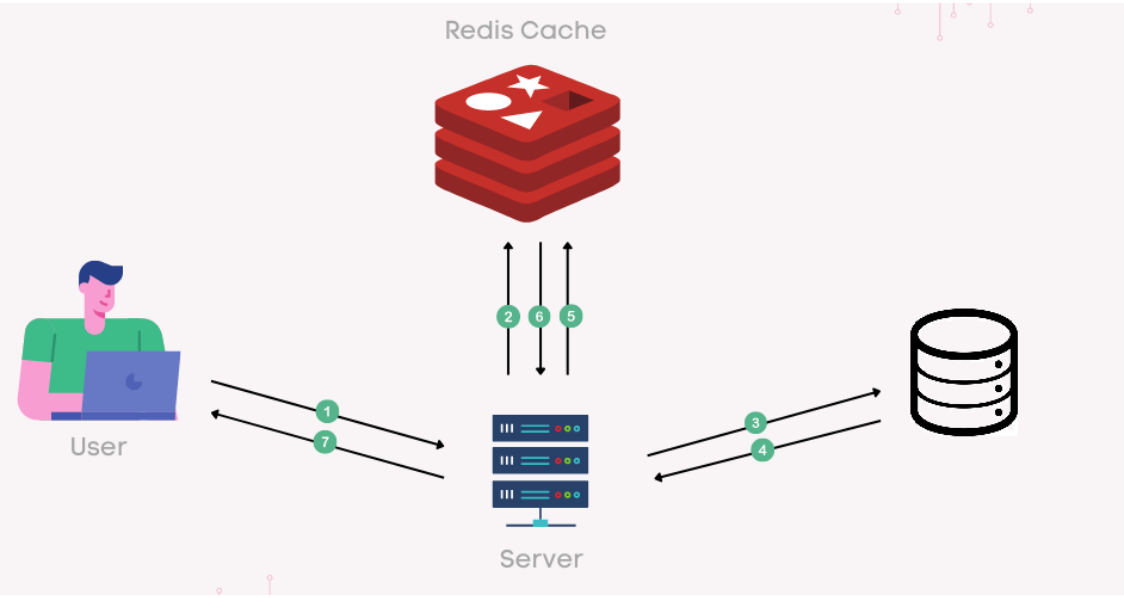

Рисунок 3.3 – Пример работы Redis

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.

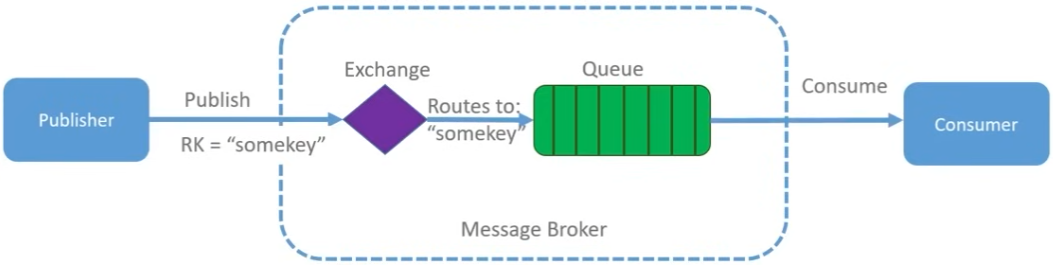

Рисунок 3.4 – Пример работы RabbitMQ

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.

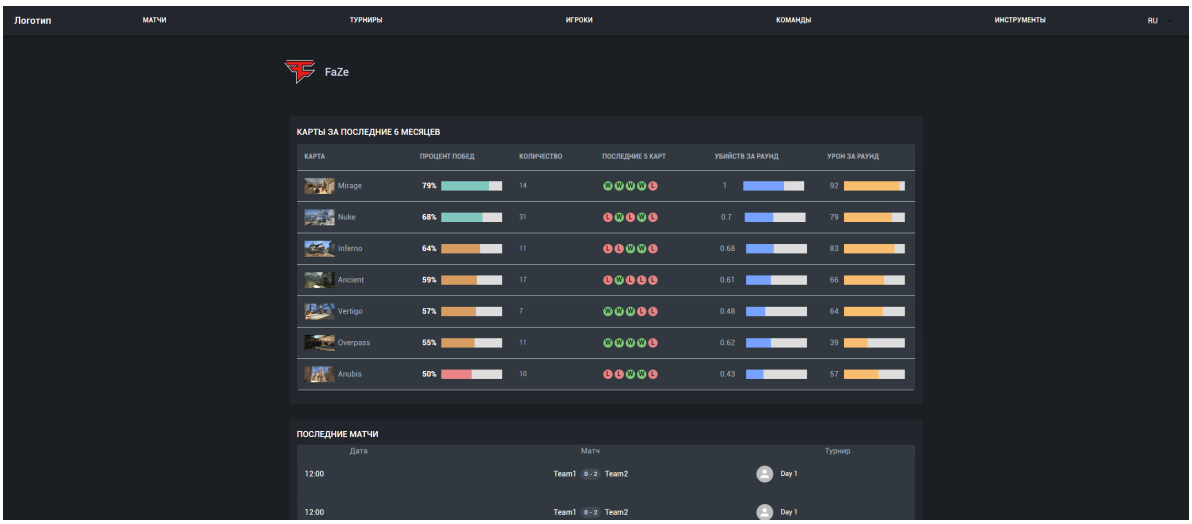

Рисунок 3.5 – Макет страницы команды

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.

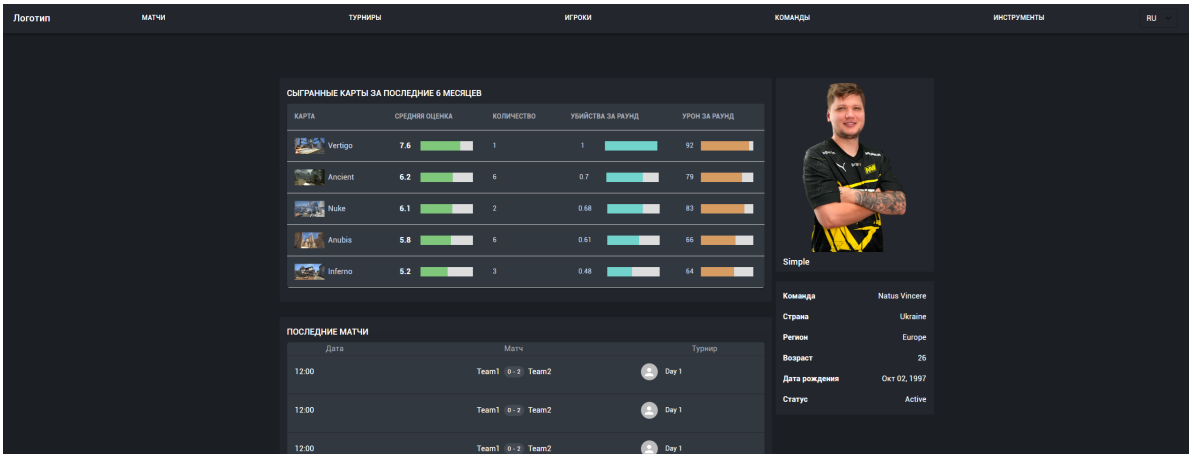

Рисунок 3.6 – Макет страницы игрока

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.

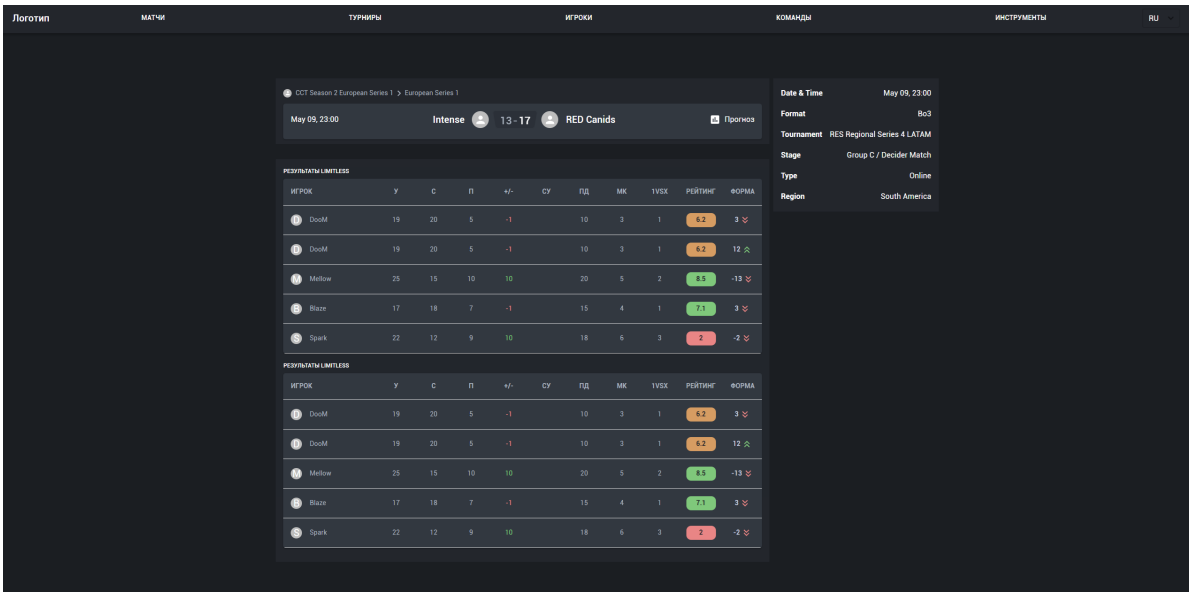

Рисунок 3.7 – Макет страницы матча

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.

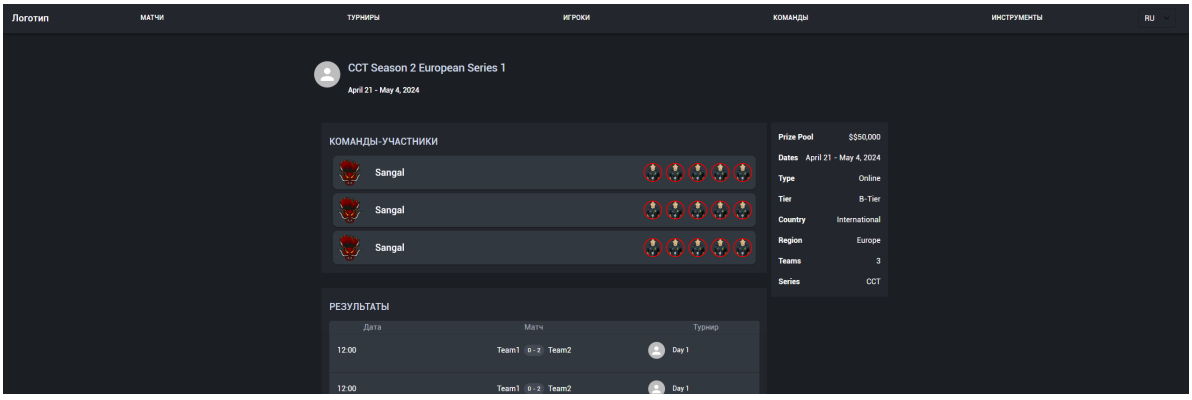

Рисунок 3.8 – Макет страницы турнира

Макет содержит следующие элементы:

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

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.