

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS**

Разработка демонстрационного веб-сервиса по предоставлению услуг iGaming

Обучающийся / Student Колобов Егор Михайлович

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования

Группа/Group М34021

Направление подготовки/ Subject area 09.03.02 Информационные системы и технологии

Образовательная программа / Educational program Программирование и интернет-технологии 2019

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Бакалавр

Руководитель ВКР/ Thesis supervisor Штумпф Святослав Алексеевич, кандидат физико-математических наук, Университет ИТМО, факультет информационных технологий и программирования, доцент (квалификационная категория "доцент практики")

Консультант не из ИТМО / Third-party consultant Kastiel Michael, Eiger LTD, Lead Developer

Обучающийся/Student

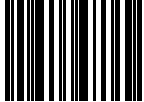
Документ подписан	
Колобов Егор Михайлович	
28.05.2023	

(эл. подпись/ signature)

Колобов Егор
Михайлович

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Штумпф Святослав Алексеевич	
24.05.2023	

(эл. подпись/ signature)

Штумпф
Святослав
Алексеевич

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Колобов Егор Михайлович
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования
Группа/Group M34021
Направление подготовки/ Subject area 09.03.02 Информационные системы и технологии
Образовательная программа / Educational program Программирование и интернет-технологии 2019
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Разработка демонстрационного веб-сервиса по предоставлению услуг iGaming
Руководитель ВКР/ Thesis supervisor Штумпф Святослав Алексеевич, кандидат физико-математических наук, Университет ИТМО, факультет информационных технологий и программирования, доцент (квалификационная категория "доцент практики")
Консультант не из ИТМО / Third-party consultant Kastiel Michael, Eiger LTD, Lead Developer

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Ожидаемой целью ВКР является разработка демонстрационного веб-сервиса для индустрии iGaming. Сервис создаётся для ознакомления пользователя с данной индустрией, помогает понять как устроены подобные сервисы и дает ему возможность потренироваться в таких играх как спортивный покер и шахматы. Данный сервис не предоставляет возможности ставить реальные деньги, поэтому он не подпадает под закон о регулировании азартной деятельности.

Задачи, решаемые в ВКР / Research tasks

Необходимо разработать демонстрационный веб-сервис iGaming, который предоставляет следующие возможности. Пользователям будет доступна возможность создавать учетные записи и входить в них с использованием логина и пароля. В системе будут присутствовать пользователи с разным уровнем доступа, включая администраторов. Администраторы смогут добавлять новые игры на ресурс. Сервис также предоставит возможность играть в различные игры в режиме демонстрации. Пользователи смогут обращаться к обучающим материалам и руководствам по каждой игре, чтобы быстро освоить правила и стратегии игры. В сервисе будут предложены различные виды игр, включая шахматы, покер и другие

игры. Интерфейс сервиса будет простым и интуитивно понятным, чтобы пользователи могли быстро находить нужные игры и функции.

Краткая характеристика полученных результатов / Short summary of results/findings

В результате разработан демонстрационный веб-сервис iGaming с платформой для запуска и тестирования новых игр. Зарегистрированные пользователи могут тренироваться в спортивных играх, таких как спортивный покер и шахматы. Сервис обеспечивает безопасность и конфиденциальность данных с помощью аутентификации на основе djoser. Удобный пользовательский интерфейс обеспечивает простоту использования и навигации. Реализованная архитектура на базе Django Rest Framework и React обеспечивает высокую производительность. В итоге, Завершенный проект полностью соответствует требованиям и целям, предоставляя качественный сервис в области iGaming.

Обучающийся/Student

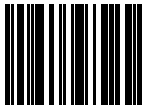
Документ подписан	
Колобов Егор Михайлович	
28.05.2023	

(эл. подпись/ signature)

Колобов Егор
Михайлович

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Штумпф Святослав Алексеевич	
24.05.2023	

(эл. подпись/ signature)

Штумпф
Святослав
Алексеевич

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS**

Обучающийся / Student Колобов Егор Михайлович
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования
Группа/Group М34021
Направление подготовки/ Subject area 09.03.02 Информационные системы и технологии
Образовательная программа / Educational program Программирование и интернет-технологии 2019
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Разработка демонстрационного веб-сервиса по предоставлению услуг iGaming
Руководитель ВКР/ Thesis supervisor Штумпф Святослав Алексеевич, кандидат физико-математических наук, Университет ИТМО, факультет информационных технологий и программирования. доцент (квалификационная категория "доцент практики")

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Требуется разработать демонстрационный веб-сервиса по предоставлению услуг iGaming. Сервис должен позволять пользователям создавать свои учетные записи и входить в них с помощью логина и пароля. Кроме того, необходимо наличие пользователей с разным уровнем доступа к функционалу системы. Пользователь со статусом администратора должен иметь возможность добавить новые игры на ресурс. Помимо этого сервис должен предоставлять возможность играть в различные игры в демонстрационном режиме. Вдобавок, сервис должен предоставлять обучающие материалы и руководства по каждой игре, чтобы пользователи могли быстро понять правила и стратегии игры. Также сервис должен предлагать различные виды игр, включая шахматы, покер и другие настольные игры. Ещё сервис должен иметь простой и интуитивно понятный интерфейс, чтобы пользователи могли быстро находить нужные игры и функции.

Дата выдачи задания / Assignment issued on: 15.03.2023

Срок представления готовой ВКР / Deadline for final edition of the thesis 15.05.2023

Характеристика темы ВКР / Description of thesis subject (topic)

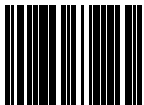
Тема в области фундаментальных исследований / Subject of fundamental research: нет /

not

Тема в области прикладных исследований / Subject of applied research: да / yes

СОГЛАСОВАНО / AGREED:

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Штумпф Святослав Алексеевич	
17.05.2023	

(эл. подпись)

Штумпф
Святослав
Алексеевич

Задание принял к
исполнению/ Objectives
assumed BY

Документ подписан	
Колобов Егор Михайлович	
19.05.2023	

(эл. подпись)

Колобов Егор
Михайлович

Руководитель ОП/ Head
of educational program

Документ подписан	
Маятин Александр Владимирович	
19.05.2023	

(эл. подпись)

Маятин
Александр
Владимирович

Содержание

Содержание	4
Использованные сокращения и обозначения	5
Введение	6
Глава 1. Анализ предметной области	7
1.1. Описание предметной области и возможное применение	7
1.2 Анализ похожих сервисов	8
1.3 Формирование требований	9
Глава 2. Проектирование	11
2.1 Стек технологий	11
2.2 Архитектура системы	13
2.2.1 Архитектура серверной части сервиса.	14
2.2.1.1 Приложение Games	16
2.2.1.1 Приложение Users	19
2.2.1 Архитектура клиентской части сервиса.	21
Глава 3. Программная реализация	24
3.1 Реализация серверной части	24
3.2 Настройка базы данных	33
3.3 Клиентская часть	34
3.4 Ручное тестирование сервиса iGaming	38
Заключение	41
Список использованных источников	43

Использованные сокращения и обозначения

- 1) Пользователь – это зарегистрированный пользователь, который может просматривать информацию об играх и своем профиле. Пользователь реализован на основе модели Profile из стандартного пакета Django, а также связанной с ней моделью профиля User.
- 2) Игра – интерактивное развлечение, которое пользователи могут запускать на платформе. Игра может быть любого типа, включая карточные игры, настольные игры и другие. Все игры в сервисе хранятся в базе данных в виде модели Game (рисунок 3) и могут быть доступны для просмотра, запуска и управления через клиентскую часть приложения.
- 3) Администратор – это пользователь с особыми правами доступа, который может управлять данными в сервисе. Он может создавать, просматривать, изменять и удалять игры и типы игр. Администратор использует модели Game и Type, которые отвечают за хранение данных об играх и типах игр соответственно. Он может выполнять различные CRUD-операции с данными моделей через REST API с помощью DRF.
- 4) iGaming – это термин, сочетающий в себе два понятия: "интернет" (Internet) и "игры" (Gaming). Это область развлечений карточных и настольных игр, которая использует интернет-технологии и онлайн-платформы для предоставления услуг игрокам со всего мира. В этой области игроки могут играть в различные соревновательные и развлекательные игры, такие как спортивный покер, шахматы и многие другие игры.

Введение

Современный рынок iGaming продолжает быстро расти и развиваться, и спрос на качественные услуги в этой области неуклонно растет. В связи с этим возникает потребность в разработке демонстрационного веб-сервиса, который позволит пользователям запускать и тестировать новые игры, а также использовать его в качестве тренировочной площадки для спортивных игр, таких как покер или шахматы.

Демонстрационный веб-сервис по предоставлению услуг iGaming – это платформа, которая позволяет пользователям опробовать новые игры, не рискуя своими средствами, а также позволяет тренироваться в спортивных играх без необходимости вносить настоящие деньги.

Основная цель такого сервиса – предоставить пользователям возможность наслаждаться различными играми, не беспокоясь о возможных потерях. Пользователи могут играть в разные игры, включая блэкджек, покер, шахматы и многое другое.

Кроме того, сервис может быть использован в качестве тренировочной площадки для игроков, которые хотят улучшить свои навыки в спортивных играх, таких как покер и шахматы. Пользователи могут играть против искусственного интеллекта или других игроков на демонстрационных столах, используя виртуальные деньги. Таким образом, они могут попробовать разные стратегии и улучшить свои навыки.

В целом, демонстрационный веб-сервис по предоставлению услуг iGaming является отличной платформой для развлечения, игры и тренировки, которая позволяет пользователям наслаждаться играми без риска потерь и с большим комфортом.

Глава 1. Анализ предметной области

1.1. Описание предметной области и возможное применение

Предметная область iGaming (от английского "Interactive Gaming") – это область развлечений карточных и настольных игр, которая использует интернет-технологии и онлайн-платформы для предоставления услуг игрокам со всего мира. В этой области игроки могут играть в различные игры, такие как покер, шахматы и другие соревновательные игры, а также участвовать в различных турнирах и соревнованиях.

iGaming стал одной из самых быстрорастущих отраслей развлечений в мире, благодаря возможностям, которые предоставляет интернет-технология. Онлайн-платформы позволяют игрокам наслаждаться играми из любой точки мира, не покидая свой дом.

Важным аспектом iGaming является соблюдение правил и законов в различных юрисдикциях. Каждая страна имеет свои правила и законы, которые регулируют деятельность в области азартных игр. Некоторые страны запрещают любые формы азартных игр, в то время как другие разрешают только определенные формы игр. Поэтому компании, занимающиеся iGaming, должны соблюдать местные законы и правила в каждой юрисдикции, где они предоставляют свои услуги.

Кроме того, безопасность и честность игр также являются ключевыми аспектами iGaming. Игроки должны чувствовать себя защищенными от мошенничества и неправомерных действий со стороны операторов. Поэтому компании, занимающиеся iGaming, должны обеспечивать высокий уровень безопасности и честности игр, используя современные технологии и профессиональный подход.

Можно ожидать, что iGaming будет продолжать работать и расширяться в будущем. Разработка новых игр и технологий, а также улучшение безопасности и честности игр будут ключевыми факторами роста в этой области.

Для развития iGaming важно, чтобы компании продолжали разрабатывать новые и увлекательные игры, которые привлекали бы все больше игроков, а также предоставляли услуги высокого качества. Одним из важных аспектов этой области является обучение и тренировка игроков. Многие игры требуют определенных навыков и стратегий, поэтому необходима подготовка и обучение, чтобы достичь успеха в играх. В этом контексте может быть полезной разработка демонстрационного веб-сервиса, который предоставляет возможность запускать и тестировать новые игры, а также может служить в качестве тренировочной площадки для таких спортивных игр, как покер или шахматы.

Такие демонстрационные веб-сервисы могут улучшить опыт игры для существующих игроков. Кроме того, сервисы такого рода могут помочь компаниям, занимающимся iGaming, собирать информацию об использовании игр и оценивать их популярность, что может помочь улучшить и оптимизировать предоставляемые услуги.

1.2 Анализ похожих сервисов

На сегодняшний день существует множество сервисов, предоставляющих услуги iGaming. Однако, не все из них имеют демонстрационный вариант игр, предназначенный для обучения и развлечения.

Один из наиболее популярных сервисов в этой области – PokerStars. Этот сервис предоставляет игры в покер и спортивные пари, а также имеет демонстрационный режим для каждого из этих типов игр. Демонстрационный режим позволяет игрокам играть с виртуальными фишками и учиться игре. Также PokerStars имеет бесплатные турниры и фрироллы.

Еще один интересный сервис – Betfair, который предоставляет услуги спортивных пари и казино. Он также имеет демонстрационный режим для некоторых из своих игр, позволяющий игрокам учиться игре и тестировать

новые стратегии без риска потерять реальные деньги. Betfair также предоставляет различные бонусы и акции для новых и постоянных игроков.

Однако, не все сервисы предоставляют демонстрационный режим игр. Например, William Hill – один из наиболее известных букмекеров, не имеет демонстрационного режима для своих игр. Это может быть недостатком для игроков, которые хотят только учиться игре.

В целом, существует множество сервисов iGaming, предоставляющих различные услуги и игры, и большинство из них имеют демонстрационный режим для некоторых из своих игр. Однако, все еще есть потребность в сервисах, которые бы предоставляли более широкий спектр игр в демонстрационном режиме, например покер или настольные игры.

Также стоит отметить, что некоторые из существующих сервисов iGaming могут быть ограничены законодательством в некоторых странах, что может снизить их доступность для пользователей в этих регионах. Поэтому важно учитывать законодательные ограничения и создавать сервисы, которые будут доступны для широкой аудитории.

В целом, анализ существующих сервисов iGaming показал, что большинство из них имеют демонстрационный режим для некоторых из своих игр, что позволяет пользователям учиться игре и тестировать свои стратегии без риска потерять реальные деньги. Однако, все еще существует потребность в сервисах, предоставляющих более широкий спектр игр в демонстрационном режиме, а также учитывающих законодательные ограничения для своей доступности. Кроме того, создание собственного сервиса iGaming позволяет разработать уникальный продукт, отличающийся от аналогов. Возможность предложить новые игры, инновационные функции и уникальные пользовательские возможности привлечет внимание новых пользователей.

1.3 Формирование требований

Ниже приведен список функциональных требований к разрабатываемому сервису:

- 1) Регистрация и авторизация пользователей. Сервис должен позволять пользователям создавать свои учетные записи и входить в них с помощью логина и пароля.
- 2) Наличие пользователей с разным уровнем доступа (зарегистрированный пользователь и администратор) к функционалу системы.
- 3) Возможность для пользователя со статусом администратора добавить новые игры на ресурс.
- 4) Сервис должен предоставлять возможность играть в различные игры в демонстрационном режиме. Кроме того, сервис должен предоставлять обучающие материалы и руководства по каждой игре, чтобы пользователи могли быстро понять правила и стратегии игры.
- 5) Разнообразие игр. Сервис должен предлагать различные виды игр, включая шахматы, покер и другие настольные игры.
- 6) Удобный интерфейс. Сервис должен иметь простой и интуитивно понятный интерфейс, чтобы пользователи могли быстро находить нужные игры и функции.

Ниже приведены нефункциональные требования к разрабатываемому сервису:

- 1) Быстродействие и отзывчивость, чтобы пользователи могли получать максимальное удовольствие от игры без задержек и прерываний.
- 2) Масштабируемость, чтобы обрабатывать большое количество пользователей без снижения качества обслуживания.
- 3) Совместимость, чтобы пользователи могли получать доступ к сервису с любого устройства.
- 4) Безопасность, чтобы обеспечивать конфиденциальность пользовательских данных.

Эти нефункциональные требования к сервису iGaming важны для обеспечения качественного пользовательского опыта и должны быть учтены в процессе разработки и тестирования сервиса.

Глава 2. Проектирование

2.1 Стек технологий

Backend:

- 1) Django – Python web-фреймворк, который используется для создания backend части приложения.
- 2) Django REST Framework (DRF) – фреймворк, расширяющий Django для создания RESTful API для взаимодействия с клиентом.
- 3) База данных. В данном случае, используется SQLite.

Выбор базы данных SQLite оптимален для этого проекта по нескольким причинам:

- 1) SQLite не требует отдельного сервера баз данных, а хранит все данные в локальном файле, что делает установку и настройку простыми и быстрыми.
- 2) SQLite использует минимальное количество ресурсов, что позволяет ему работать на любом компьютере без каких-либо ограничений.
- 3) SQLite поддерживает работу с большими объемами данных, что позволяет масштабировать проект при необходимости.
- 4) Django имеет встроенную поддержку SQLite, что упрощает работу с базой данных и интеграцию с фреймворком.
- 5) SQLite имеет высокую степень надежности и безопасности, что позволяет использовать его для хранения и обработки конфиденциальной информации.

Frontend:

- 1) React – JavaScript библиотека, которая используется для создания frontend части приложения.
- 2) React Router – библиотека для навигации в приложении.
- 3) Axios – библиотека для отправки запросов на сервер.

Django Rest Framework (DRF) и React – это популярные и широко используемые технологии для разработки веб-приложений. Объединение этих технологий для разработки сервиса iGaming может быть оптимальным выбором по нескольким причинам:

DRF – это надежный и мощный фреймворк для разработки веб-приложений, который предоставляет широкий спектр функций и инструментов для быстрой и удобной разработки RESTful API. С помощью DRF можно легко создавать и настраивать API для обмена данными между клиентской и серверной частями приложения.

React – это одна из самых популярных библиотек для разработки пользовательского интерфейса. React позволяет создавать масштабируемые и высокопроизводительные интерфейсы, которые могут работать на разных устройствах и платформах. Благодаря использованию React можно быстро создавать динамические и интерактивные пользовательские интерфейсы, что важно для iGaming сервисов, где пользователи часто ожидают быстрого и качественного ответа на свои запросы.

React Router – библиотека для роутинга в React приложении. С помощью React Router можно легко создавать маршруты для отображения разных страниц приложения и управлять ими.

Axios – библиотека для отправки HTTP-запросов из React приложения. С помощью Axios можно легко получать данные с сервера и отправлять данные на сервер.

Django и React – это популярные и широко используемые технологии, для которых существует большое количество документации, готовых решений, сторонних библиотек и инструментов. Благодаря этому разработчики могут быстро находить и решать возникающие проблемы и быстро создавать новые функции и компоненты.

Таким образом, использование Django Rest Framework и React для разработки iGaming сервиса может быть оптимальным выбором, который

обеспечит высокое качество, производительность и масштабируемость сервис.

2.2 Архитектура системы

Для данного проекта, системная архитектура построена на основе модульного подхода, где каждый модуль имеет свою собственную функциональность и может быть разработан и отлажен независимо от других модулей.

Системная архитектура (рисунок 1) включает в себя несколько основных компонентов:

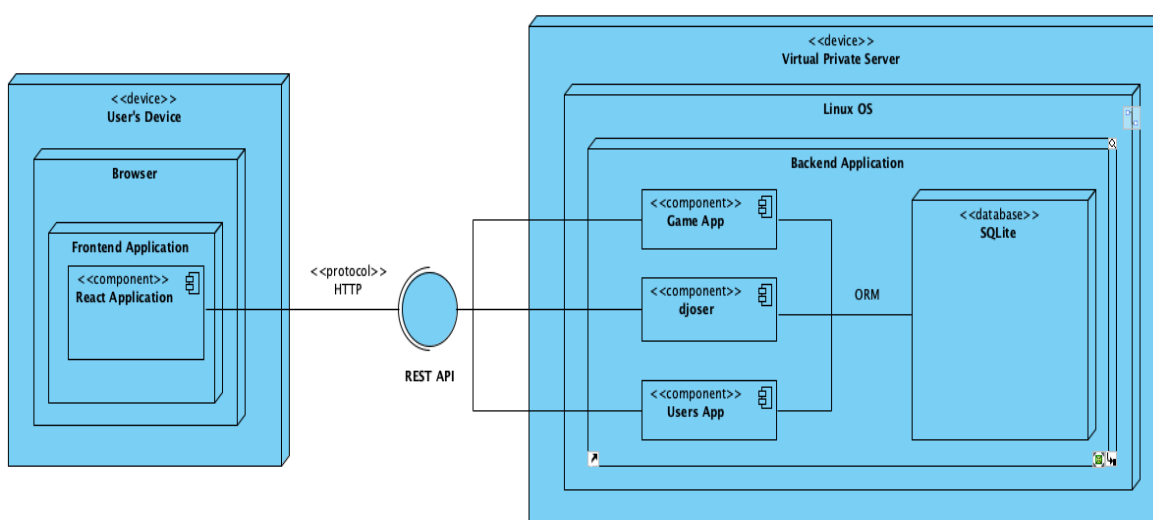


Рисунок 1 – Системная архитектура

Клиентская часть (Frontend) – приложение, которое работает в браузере пользователя. В данном проекте используется React как основной фреймворк для разработки клиентской части.

Серверная часть (Backend) – приложение, которое предоставляет RESTful API для взаимодействия с клиентской частью и базой данных. В качестве серверного фреймворка используется Django Rest Framework (DRF), который предоставляет множество инструментов для разработки RESTful API и хорошо интегрируется с базой данных.

База данных (Database) – система управления базами данных (СУБД), которая хранит информацию о пользователях, играх, и прочих данных, необходимых для функционирования сервиса. В данном проекте используется SQLite, так как она является производительной и надежной СУБД с хорошей поддержкой JSON-поля для хранения данных.

Аутентификация и авторизация (Authentication and Authorization) – система, которая обеспечивает безопасность приложения и контролирует доступ пользователей к различным ресурсам. Для аутентификации и авторизации используется JWT (JSON Web Token), так как это простой и безопасный способ передачи информации между клиентом и сервером.

Инфраструктура (Infrastructure) – сервера, на которых разворачивается приложение, а также средства для автоматизации процесса разработки, тестирования и разворачивания приложения. В данном проекте используется облачную инфраструктуру Virtual Private Servers, которая предоставляет множество инструментов для автоматизации процесса разработки и разворачивания приложения.

Система основывается на RESTful API архитектуре, что обеспечивает более гибкое и масштабируемое взаимодействие между клиентской и серверной частями.

Таким образом, системная архитектура для iGaming сервиса построенная на основе вышеперечисленных компонентов и технологий, обеспечивая высокую производительность, масштабируемость и безопасность.

2.2.1 Архитектура серверной части сервиса.

Архитектура серверной части данного сервиса (рисунок 2), реализованная с помощью Django Rest Framework (DRF), представляет собой модульную структуру, где каждое приложение отвечает за свою часть функционала. В данном случае, есть два основных приложения: *Games* и

Users. iGaming App содержит файл настроек проекта settings.py и файл с эндпоинтами urls.py

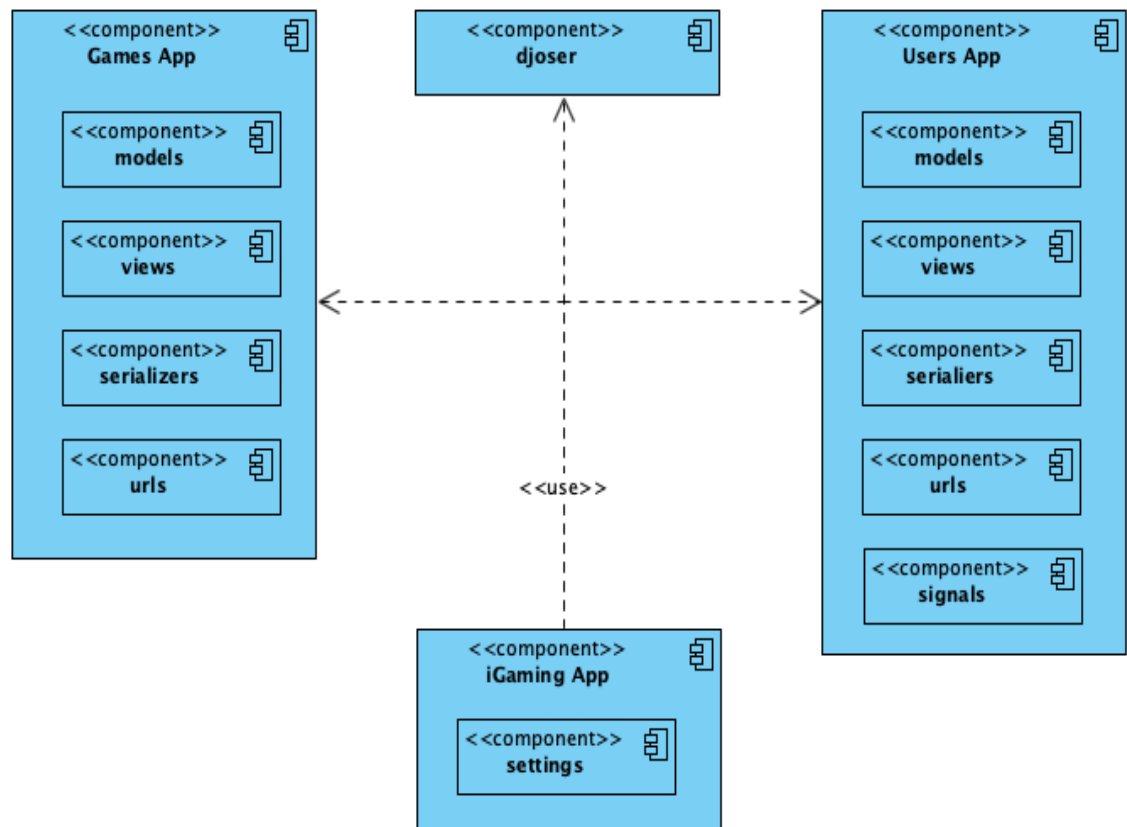


Рисунок 2 – Диаграмма компонентов серверной части

Аутентификация пользователей обрабатывается с помощью пакета *djoser*. Djoser – это пакет, предоставляющий готовое решение для реализации авторизации и аутентификации в Django-приложениях. Он упрощает создание токенов аутентификации и предоставляет необходимые API-методы для работы с ними.

В данном проекте, пакет Djoser используется для обработки аутентификации. Он предоставляет API-методы для регистрации пользователей, авторизации и выхода из системы. Кроме того, он генерирует токен доступа и токен обновления, которые могут использоваться для защиты конфиденциальных данных и выполнения действий в имени пользователя.

Djoser также предоставляет возможность настройки отправки электронной почты для подтверждения адреса электронной почты при регистрации, а также сброса пароля и обновления профиля пользователя.

Использование Djoser в данном проекте является хорошим решением, так как он предоставляет готовое решение для обработки аутентификации, обладает высокой степенью гибкости в настройке и может быть легко интегрирован с другими компонентами проекта. Это позволяет разработчикам сосредоточиться на реализации бизнес-логики и функциональности приложения, не тратя много времени на реализацию системы аутентификации и безопасности.

Таким образом, архитектура серверной части, реализованная с помощью DRF, обеспечивает модульность, гибкость и безопасность при обработке запросов, связанных с играми и пользователями.

2.2.1.1 Приложение Games

Приложение Games (рисунок 3) отвечает за обработку запросов, связанных с играми. Оно содержит модели данных игры (**Game**) и типа игры (**Type**), сериализаторы и представления (views) для каждой игры. Модели данных отображают таблицы базы данных, хранящие информацию о каждой игре. Сериализаторы определяют, как данные будут сериализовываться и десериализовываться между форматами JSON и Python. Представления содержат логику для обработки запросов, например, создание новой игры, получение списка всех игр или обновление данных об игре.

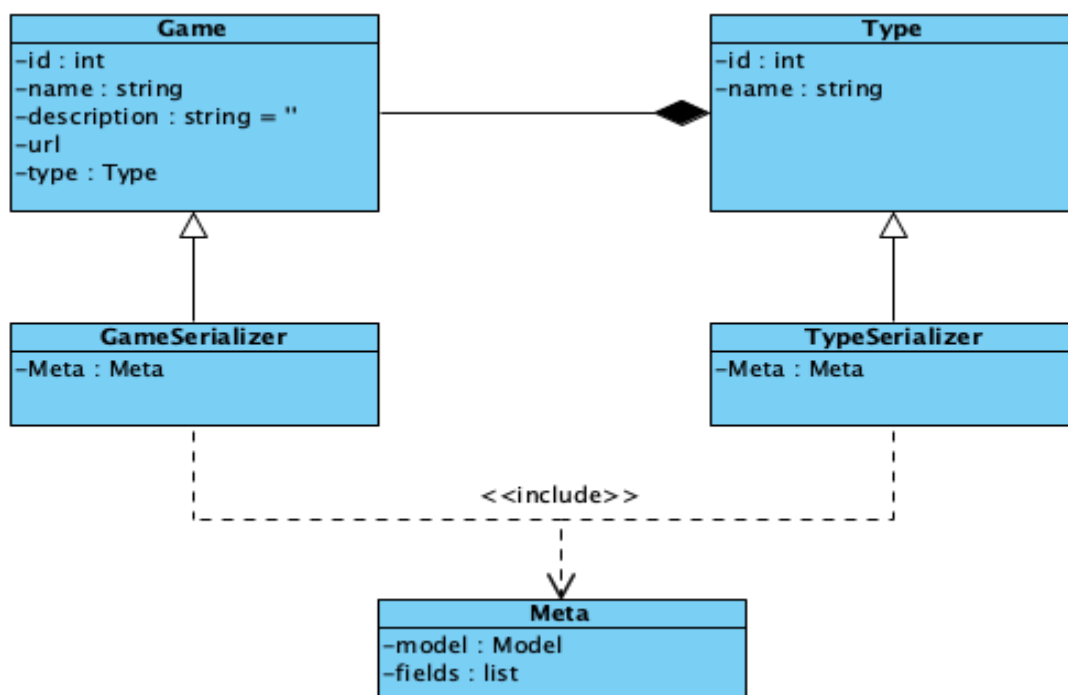


Рисунок 3 – Диаграмма классов приложения Games

Модель **Game** представляет игру, которую пользователи могут запускать и играть на веб-сервисе. Она содержит следующие поля:

- 1) name: название игры, строка длиной не более 255 символов, которое не может быть пустым (null=False) и не может состоять только из пробелов (blank=False).
- 2) description: описание игры, текстовое поле, которое может быть пустым (default="").
- 3) url: ссылка на игру, поле URL, которое не может быть пустым (null=False) и не может состоять только из пробелов (blank=False).
- 4) type: тип игры, ссылка на модель Type, которая определяет категорию игры. Это поле является внешним ключом и связывает игру с ее типом. Если тип игры был удален из базы данных, игры, которые ссылаются на него, будут защищены от удаления (on_delete=models.PROTECT). Поле может быть пустым (null=True), что означает, что игра не относится к определенному типу.

Модель **Type** определяет тип игры. Она содержит следующие поля:

- 1) name: название типа игры, строка длиной не более 255 символов, которое используется в качестве идентификатора типа. Это поле имеет индекс в базе данных (db_index=True), что позволяет быстро искать игры определенного типа. Поле также должно быть уникальным (unique=True), чтобы избежать создания дубликатов типов игр. Не может быть пустым (null=False) и не может состоять только из пробелов (blank=False)

Важно отметить, что каждая модель в Django по умолчанию имеет поле "id" типа IntegerField, которое используется как уникальный идентификатор записи в базе данных. Однако, если явно не указано иное, это поле создается автоматически и не требует объявления в модели.

Сериализаторы в Django REST Framework (DRF) являются классами, которые преобразуют модели Django в JSON или другой формат данных, понятный клиенту, например, браузеру. Например, *GameSerializer* определяет как модель Game должна быть сериализована. Мета-класс Meta определяет, какие поля из модели Game должны быть сериализованы, используя атрибут fields.

Файл views.py в Django – это модуль, содержащий представления (views), которые обрабатывают HTTP-запросы и возвращают HTTP-ответы. Он содержит определения для функций-обработчиков или классов-обработчиков, которые обрабатывают запросы и возвращают соответствующие ответы. В приложении Games, данный файл содержит представления для обработки запросов, связанных с моделями Game и Type, а также запросов, не связанных напрямую ни с одной из этих моделей. В файле определены классы представлений, унаследованные от классов, предоставляемых Django Rest Framework, таких как ListCreateAPIView и RetrieveUpdateDestroyAPIView. Эти классы предоставляют готовые методы для обработки запросов, таких как GET, POST, PUT и DELETE, и связывают представления с соответствующими моделями и сериализаторами. Кроме того, в файле определено кастомное представление GameURLView, которое

обрабатывает GET-запрос на получение URL игры по ее id, а также представление `GameListByType`, которое возвращает список игр определенного типа. Некоторые из классов обработчиков имеют `permission` классы, которые позволяют использовать определённый эндпоинт API только пользователям с конкретным статусом. Например, у `GameURLView` есть следующие параметры:

```
authentication_classes = [SessionAuthentication, BasicAuthentication]
```

```
permission_classes = [IsAuthenticated, IsAdminUser]
```

Таким образом доступ к `GameURLView` будут иметь только пользователи, успешно прошедшие аутентификацию, и администраторы сервиса.

2.2.1.1 Приложение Users

Приложение *Users* отвечает за обработку запросов, связанных с профилями пользователей. Оно также содержит модель данных профиля (*Profile*), сериализатор и представления профиля. Модель данных отображает таблицы базы данных, хранящие информацию о каждом пользователе и его профиле. Сериализаторы определяют, как данные будут сериализоваться и десериализоваться между форматами JSON и Python. Представления содержат логику для обработки запросов, например, обновление данных профиля. Важно отметить, что модель профиля расширяет функциональность встроенной модели *User*, добавляя возможность отслеживать виртуальный баланс профиля. Новый экземпляр модели *Profile* автоматически создаётся, при создании нового экземпляра встроенной модели *User*. Происходит это при помощи функции, которая обрабатывает сигнал о создании *User* и создаёт на его основе новый экземпляр *Profile*. Диаграмма классов для *Profile*, представлена на рисунке 4.

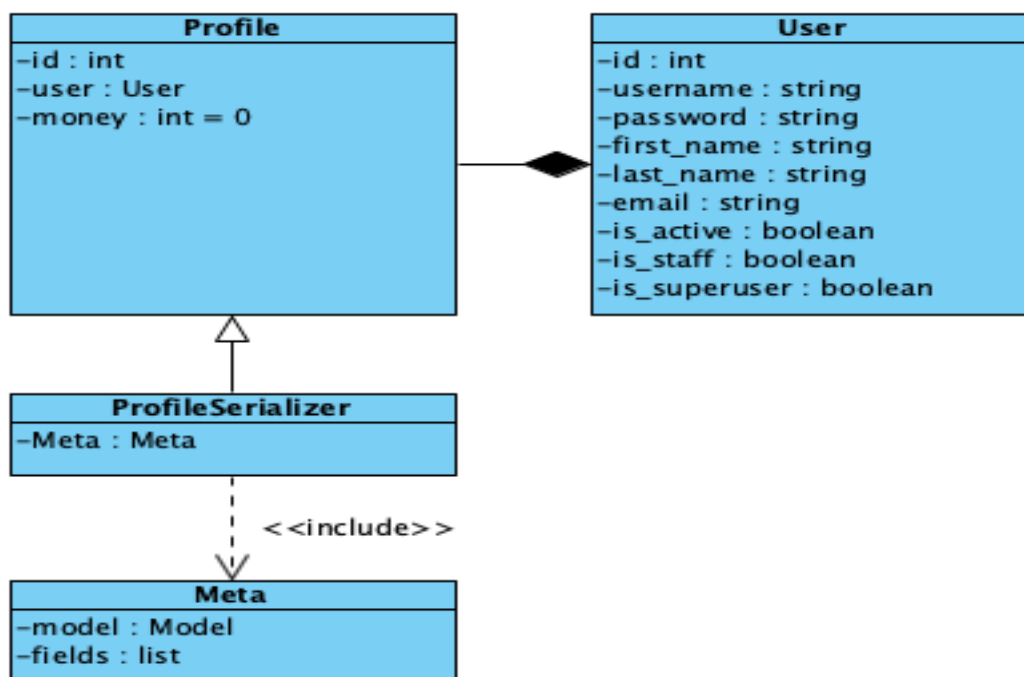


Рисунок 4 – Диаграмма классов приложения Games

Модель **Profile** является моделью профиля пользователя и связана с моделью User с помощью связи OneToOneField. Она содержит следующие поля:

- 1) user – связь с моделью User через связь OneToOneField. Определяет пользователя, которому принадлежит профиль.
- 2) money – поле целочисленного типа, содержащее текущий баланс профиля.

Модель **User** является встроенной моделью Django для аутентификации и авторизации пользователей. Она содержит следующие поля:

- 1) username – имя пользователя. Обязательное поле, должно быть уникальным.
- 2) password – пароль пользователя. Сохраняется в зашифрованном виде.
- 3) first_name – имя пользователя.
- 4) last_name – фамилия пользователя.
- 5) email – адрес электронной почты пользователя.
- 6) is_active – булево значение, указывающее, является ли пользователь активным.

- 7) `is_staff` – булево значение, указывающее, имеет ли пользователь доступ к административной панели Django.
- 8) `is_superuser` – булево значение, указывающее, имеет ли пользователь все права в системе.

Аналогично с приложением Games, в приложении Users присутствует файл `serializers.py`, который содержит ***ProfileSerializer***, определяющий как модель Profile должна быть сериализована. Также присутствует и файл `views.py`, содержащий представления для обработки запросов, связанных с моделью Profile. Особенностью приложения Users можно считать файл ***signals.py***. Он используется для регистрации сигналов (signals) – событий, происходящих в системе Django, и предназначенных для обработки изменений в моделях. В данном конкретном коде мы регистрируем два обработчика сигнала `post_save`, который срабатывает после сохранения модели. Оба обработчика относятся к модели User и предназначены для обновления или создания профиля пользователя при создании или изменении пользователя. Таким образом, эти обработчики позволяют автоматически создавать профили пользователей и поддерживать их актуальность, связывая их с моделью User, без необходимости создавать или обновлять их вручную при каждом создании или изменении пользователей.

2.2.1 Архитектура клиентской части сервиса.

Клиентская часть сервиса (рисунок 5) реализована с использованием React.js и представляет собой SPA (Single Page Application). В системной архитектуре клиентской части можно выделить следующие компоненты:

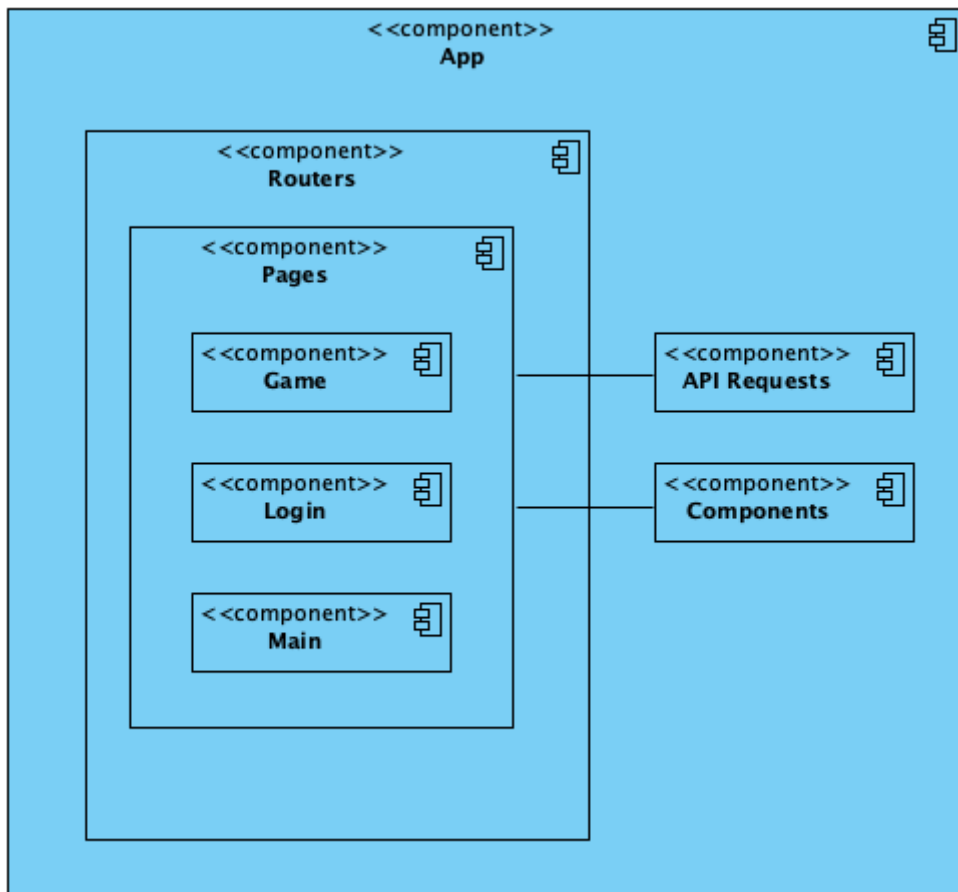


Рисунок 5 – диаграмма компонентов клиентской части

1) Компоненты(Components):

Каждая страница и каждый элемент интерфейса реализуются как компоненты React. Компоненты могут быть классовыми или функциональными. Компоненты могут взаимодействовать друг с другом с помощью передачи пропсов (props).

2) Маршрутизация (Routers):

Для управления маршрутизацией используется React Router. React Router позволяет определять маршруты, которые соответствуют определенным URL-адресам и которые загружают соответствующие компоненты. Маршрутизация позволяет реализовывать SPA-приложение (Single Page Application), в котором все страницы загружаются без перезагрузки браузера.

3) Запросы к серверу (API Requests):

Для отправки запросов к серверу используется библиотека Axios. Axios предоставляет удобный API для отправки запросов на сервер и обработки ответов.

4) Страницы (Pages)

Хранит в себе страницы клиентской веб сервиса для взаимодействия пользователя со всем приложениям. Страницы Game содержат всю логику взаимодействия пользователя с играми, Login отвечает за регистрацию и авторизацию пользователя. Main показывает информацию о сервисе в случае успешной авторизации.

Таким образом, системная архитектура клиентской части сервиса, реализованной с помощью React, Axios и React Router, позволяет создавать интерактивные и масштабируемые приложения с удобным интерфейсом и быстрым откликом.

Глава 3. Программная реализация

Программная реализация – это один из важнейших этапов в разработке любого проекта. В данной секции мы рассмотрим подробно, как был реализован iGaming сервис с использованием Django Rest Framework и React. Мы рассмотрим, как были созданы модели данных, как реализована системная архитектура, какие технологии использовались и как их связать между собой. Также мы рассмотрим примеры кода, описав, как они реализованы, каким образом работают с моделями и как взаимодействуют с базой данных.

3.1 Реализация серверной части

Для создания приложений Games и Users в проекте на Django необходимо выполнить несколько шагов.

1) Создание приложений

Создание приложений выполняется командами (рисунок 6 и рисунок 7) в терминале:

```
(venv) PS C:\Users\USER\Desktop\iGaming> python manage.py startapp users
```

Рисунок 6 – Создание приложения Users

```
(venv) PS C:\Users\USER\Desktop\iGaming> python manage.py startapp games
```

Рисунок 7 – Создание приложения Games

2) Настройка приложений

После создания приложений, необходимо добавить их в список установленных приложений в настройках проекта. Для этого в файле settings.py проекта нужно добавить классы конфигураций приложений (рисунок 8).

```
INSTALLED_APPS = [
    # ...
    'games.apps.GamesConfig',
    'users.apps.UsersConfig',
    # ...
]
```

Рисунок 8 – Добавление созданных приложений

3) Создание моделей

Для создания моделей необходимо определить их в файле `models.py` созданных приложений. В данном проекте для приложения Games были определены модели Game и Type, а для приложения Users – модель Profile.

На рисунке 9 показана реализация моделей Game и Type.

```
from django.db import models

class Game(models.Model):
    name = models.CharField(max_length=255, null=False, blank=False)
    description = models.TextField(default='')
    url = models.URLField(null=False, blank=False)
    type = models.ForeignKey('Type', on_delete=models.PROTECT, null=True)

    def __str__(self):
        return self.name

class Type(models.Model):
    name = models.CharField(max_length=255, db_index=True, unique=True, null=False, blank=False)

    def __str__(self):
        return self.name
```

Рисунок 9 – Модели Game и Type

4) Создание миграций

После создания моделей необходимо создать миграции для их применения к базе данных (рисунок 10).

```
(venv) PS C:\Users\USER\Desktop\iGaming> python manage.py makemigrations
```

Рисунок 10 – Создание миграций

5) Применение миграций

Для применения миграций к базе данных используется команда:

```
(venv) PS C:\Users\USER\Desktop\iGaming> python manage.py migrate
```

Рисунок 11 – Применение миграций

6) Создание сериализаторов

Для работы с данными моделей через API необходимо создать соответствующие сериализаторы. В данном проекте для каждой модели был создан свой сериализатор. На рисунке 11 показана реализация сериализаторов для моделей Game и Type.

```
from rest_framework import serializers
from .models import Game, Type
```

✎ EgorKolobov

```
class GameSerializer(serializers.ModelSerializer):
```

✎ EgorKolobov

```
class Meta:
```

```
    model = Game
```

```
    fields = '__all__'
```

✎ EgorKolobov

```
class TypeSerializer(serializers.ModelSerializer):
```

✎ EgorKolobov

```
class Meta:
```

```
    model = Type
```

```
    fields = '__all__'
```

Рисунок 12 – Сериализаторы для Game и Type

GameSerializer наследуется от serializers.ModelSerializer и определяет, как модель Game должна быть сериализована. Мета-класс Meta определяет, какие поля из модели Game должны быть сериализованы, используя атрибут fields. В данном случае, все поля модели были выбраны для сериализации.

TypeSerializer также наследуется от serializers.ModelSerializer и определяет, как модель Type должна быть сериализована. Он также использует мета-класс Meta для указания всех полей модели, которые должны быть сериализованы.

Оба сериализатора являются очень простыми и используют стандартный подход DRF для сериализации моделей. Эти сериализаторы использованы для создания RESTful API для моделей Game и Type.

7) Создание представлений

Для создания представлений необходимо определить классы-представления в файле views.py созданных приложений. В данном проекте для каждой модели были определены классы ListCreateAPIView и RetrieveUpdateDestroyAPIView, которые предоставляют базовые методы для обработки GET, POST, PUT, DELETE запросов.

Например, на рисунке 13 показано как реализован класс-обработчик API-запросов для получения URL-адреса игры по ее уникальному идентификатору (pk). Он связан с эндпоинтом /api/v1/games/{id}/url

```

EgorKolobov *
class GameURLView(APIView):
    authentication_classes = [SessionAuthentication, BasicAuthentication]
    permission_classes = [IsAuthenticated, IsAdminUser]

EgorKolobov *
@swagger_auto_schema(tags=["Game"])
def get(self, request, pk):
    try:
        game = Game.objects.get(id=pk)
    except Game.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)
    serializer = GameSerializer(game)
    return Response(serializer.data['url'])

```

Рисунок 13 – Обработчик API запроса `/api/v1/games/{id}/url`

Данный код определяет класс представления `GameURLView`, который является наследником класса `APIView`. Класс используется для создания GET запроса по получению URL-адреса игры с заданным идентификатором `pk`.

Класс представления использует два класса аутентификации: `SessionAuthentication` и `BasicAuthentication`. Это означает, что для доступа к данному представлению пользователь должен быть аутентифицирован через сессионную аутентификацию или базовую аутентификацию.

Класс также указывает, какие разрешения (permissions) должны быть у пользователя для выполнения запроса. В данном случае, требуется, чтобы пользователь был аутентифицирован.

Если игра с указанным идентификатором `pk` существует, то метод `get` возвращает URL-адрес игры в формате JSON. Если игра не найдена, то возвращается ошибка `HTTP_404_NOT_FOUND`.

8) Создание URL-адресов

Для того, чтобы представления были доступны через API, необходимо создать URL-адреса для них. В данном проекте для каждой модели были определены URL-адреса для списка объектов и для отдельного объекта. Для

удобства в каждом приложении (Games и Users) были созданы файлы `urls.py`, в которых определены URL адреса, которые позже импортированы в главный файл `urls.py`, находящийся в папке проекта iGaming. Например, на рисунке 14 показано как это реализовано для Games.

```
from django.urls import path
from . import views

urlpatterns = [
    path('types/', views.TypeList.as_view(), name='type-list'),
    path('types/<int:pk>/', views.TypeDetail.as_view(), name='type-detail'),
    path('games/', views.GameList.as_view(), name='game-list'),
    path('games/<int:pk>/', views.GameDetail.as_view(), name='game-detail'),
    path('games/<int:pk>/url/', views.GameURLView.as_view(), name='game-url'),
    path('games/<str:type_name>/', views.GameListByType.as_view(), name='game-type'),
]
```

Рисунок 14 – файл `urls.py` приложения Games

Что позже импортируется в общий файл `urls.py` как на рисунке 15.

```
urlpatterns = [
    path('api/v1/', include('games.urls')),
    # ...
]
```

Рисунок 15 – Добавление URL из Games

9) Настройка аутентификации

Для аутентификации пользователей в данном проекте используется пакет `djoser`. Необходимо добавить его в список установленных приложений, настроить его в `settings.py` и включить в `urls.py` из папки проекта iGaming.

На рисунке 16 показано как это сделать в файле `urls.py`, а на рисунке 17 как это сделать в `settings.py`

```
urlpatterns = [
    path('api/v1/', include('djoser.urls')),
    re_path(r'^auth/', include('djoser.urls.authtoken')),
    # ...
]
```

Рисунок 16 – Добавление URL и обработчиков пакета `djoser`


```

REST_FRAMEWORK = {
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework.renderers.JSONRenderer',
        'rest_framework.renderers.BrowsableAPIRenderer'
    ),
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.TokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication',
    ),
}

```

Рисунок 17 – Настройка djoser в файле settings.py

10) Оформление API

DRF_yasg – это инструмент для автоматической генерации документации OpenAPI/Swagger для приложений Django REST Framework.

Swagger/OpenAPI – это спецификация, которая описывает REST API в машинно-читаемой форме, и позволяет автоматически генерировать документацию и клиентские библиотеки для разных языков программирования.

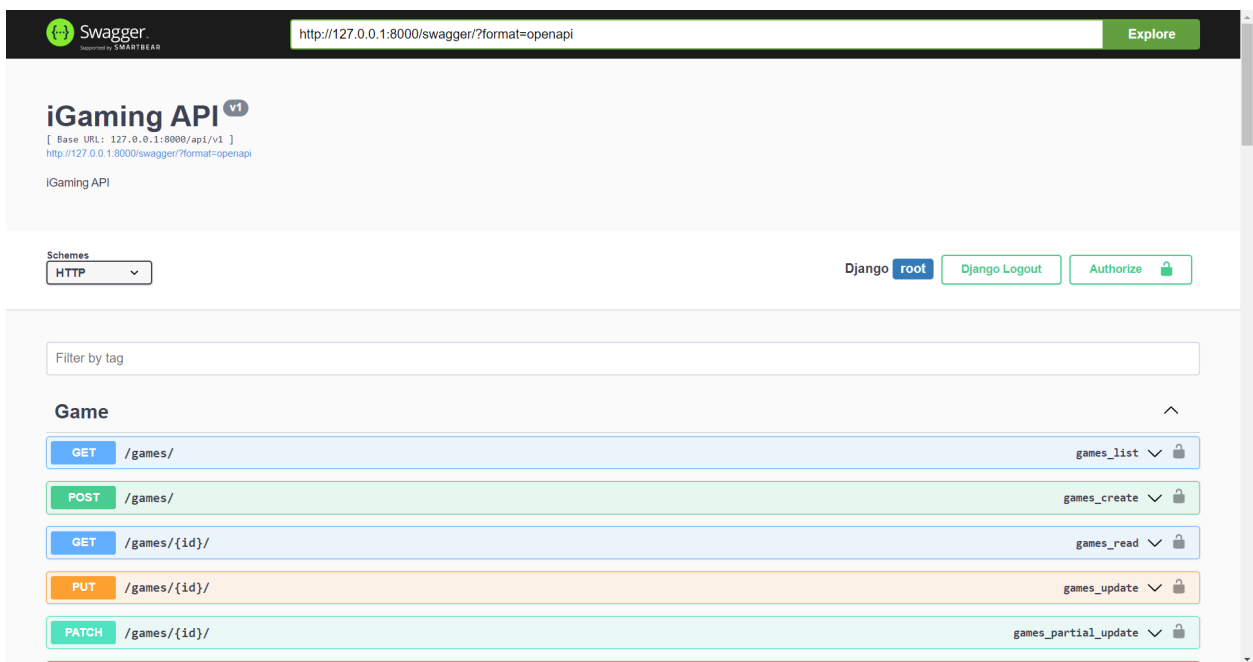


Рисунок 18 – API swagger

DRF_yasg интегрируется в проект Django, и используется для генерации документации Swagger на основе DRF-сериализаторов и представлений (views). Он предоставляет пользовательский интерфейс Swagger UI для документации и интерактивного тестирования API. Чтобы настроить DRF_yasg, нужно установить сам пакет (рисунок 19) добавить его в INSTALLED_APPS в settings.py (рисунок 20)

```
(venv) PS C:\Users\USER\Desktop\iGaming> pip install drf_yasg
```

Рисунок 19 – Установка пакета drf_yasg

```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'games.apps.GamesConfig',
    'users.apps.UsersConfig',
    'rest_framework',
    'drf_yasg',
    'rest_framework.authtoken',
    'djoser',
]

```

Рисунок 20 – Добавление *drf_yasg* в *settings.py*

Далее создаём файл *urls.py* в корневой папке проекта, и заполняем его как на рисунке 21.

```

schema_view = get_schema_view(
    openapi.Info(
        title="iGaming API",
        default_version="v1",
        description="iGaming API",
    ),
    public=True,
    permission_classes=(permissions.AllowAny,),
)

urlpatterns = [
    path('swagger/', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagger-ui'),
    path('admin/', admin.site.urls),
    # ...
]

```

Рисунок 21 – подключение *swagger* в *urls.py*

Это создает маршруты для *swagger-ui* и *redoc*. Swagger UI будет доступен по пути */swagger/*

Для кэширования документации, можно добавить параметр `cache_timeout` (в секундах), который определяет время кэширования документации.

Потом нужно добавить в свои представления (views) дополнительную информацию для генерации документации. Например, для представления для просмотра списка игр (GameList), мы можем добавить теги и описание:

— EgorKolobov *

```
class GameList(generics.ListCreateAPIView):
    queryset = Game.objects.all()
    serializer_class = GameSerializer

    new *
    @swagger_auto_schema(tags=["Game"])
    def get(self, request, *args, **kwargs):
        return super().get(request, *args, **kwargs)

    new *
    @swagger_auto_schema(tags=["Game"])
    def post(self, request, *args, **kwargs):
        return super().get(request, *args, **kwargs)
```

Рисунок 22 – оформление представления для swagger

3.2 Настройка базы данных

База данных для данного проекта должна быть создана на основе моделей, которые были описаны выше. Для создания таблиц и связей между ними можно использовать миграции Django.

Для создания миграций и применения миграций необходимо в консоли перейти в директорию проекта и выполнить следующие команды как на рисунке 10 и рисунке 11.

При выполнении этих команд Django создаст таблицы в базе данных и связи между ними.

При работе с базой данных необходимо учитывать особенности выбранной СУБД. Также следует следить за тем, чтобы в базе данных не было дубликатов записей и нарушений целостности данных.

3.3 Клиентская часть

Для реализации клиентской части нужно первым делом установить все необходимые зависимости и библиотеки для работы с React, Axios и React Router с помощью пакетного менеджера npm. Нужно ввести в терминал следующие команды: *cd client, npm install, npm start*.

Создать файлы компонентов для каждой страницы приложения, которые будут отображать содержимое и взаимодействовать с пользователем. Например, компонент "Main" для отображения главной страницы (рисунок 7), компонент "Game" для отображения информации об играх, компонент "Login" позволяет авторизовываться (рисунок 8) и регистрироваться (рисунок 9) пользователям. Например, часть кода из компонента Login (рисунок 23) проверяет что поля логина и пароля не пустые значения и отправляет запрос на авторизацию на бэкенд. В случае успеха происходит авторизация с последующей записью токена в local storage.

```
if (!isRegistration && email && password) {  
  
  API.auth.login( {username, password}: {username: email, password}) Promise<any>  
    .then((res) => {  
      localStorage.setItem('igaming', res);  
      setIsAuth(true);  
    }) Promise<any>  
    .catch((err) => {  
      toast( options: { title: 'Неверный логин или пароль', status: 'error' })  
      setPassword( value: '');  
    })  
}
```

Рисунок 23 – Часть кода Login.js отвечающая за авторизацию

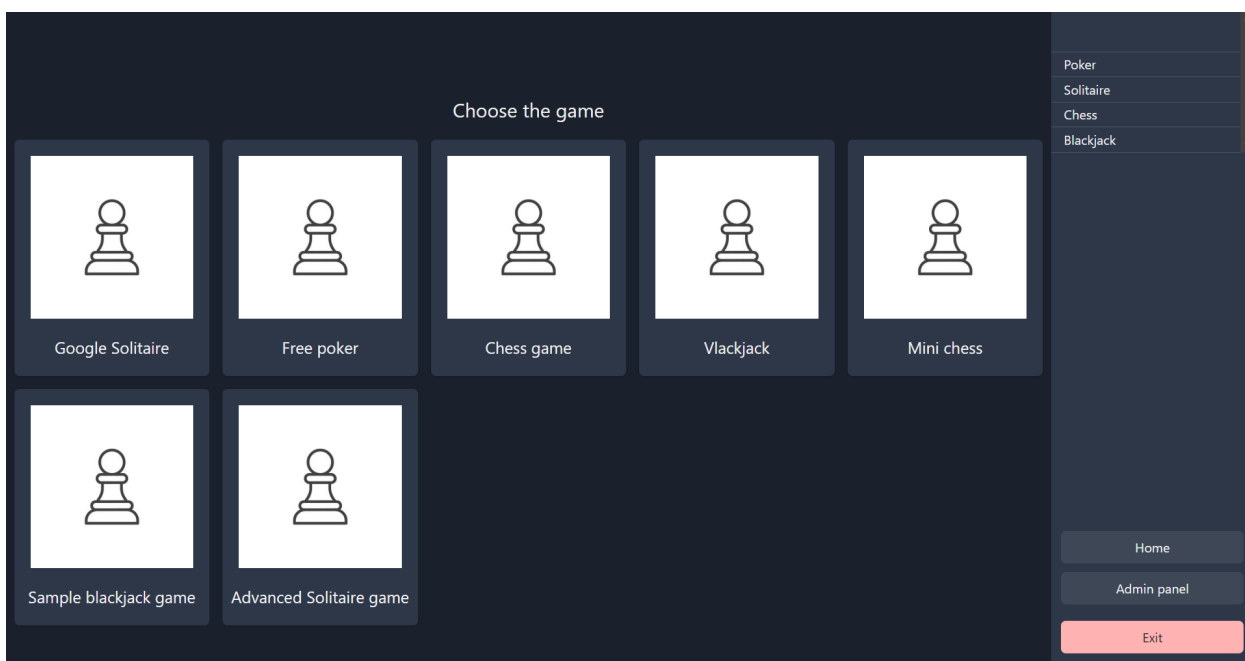


Рисунок 24 – Главная страница

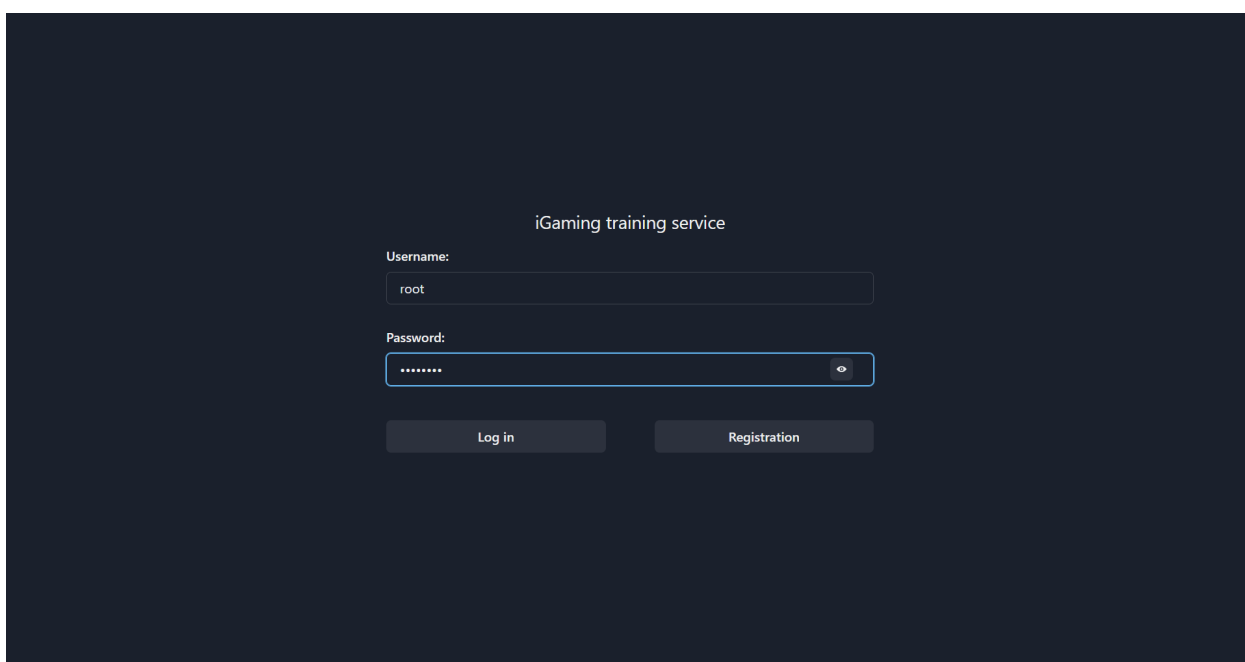
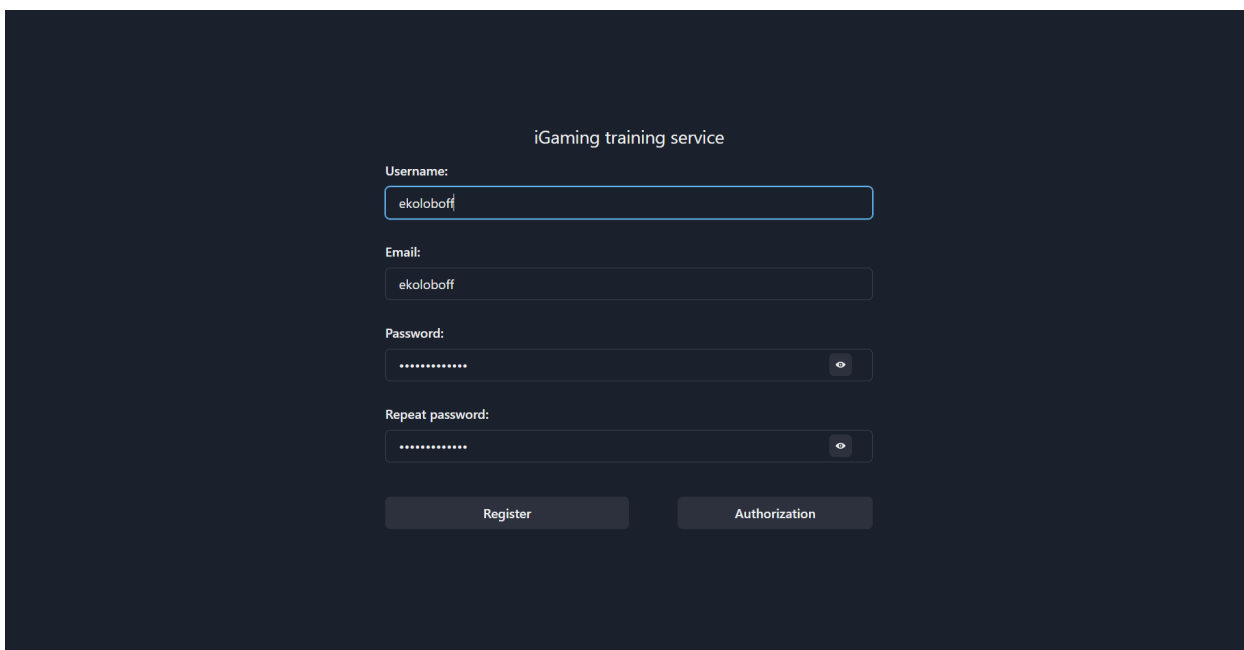


Рисунок 25 – форма для авторизации пользователя



iGaming training service

Username:
ekoloboff

Email:
ekoloboff

Password:

Repeat password:

Register Authorization

Рисунок 26 – форма для регистрации нового пользователя

Дальше необходимо создать файлы для API-запросов с помощью библиотеки Axios. Создаём файл "index.js" в папке api для выполнения запросов к API серверной части, таких как получение списка игр, получения подробной информации об игре и т.д. Каждый запрос должен быть обработан с помощью методов Axios (get, post, put, delete) и возвращать соответствующие данные. Например, часть кода с рисунка 27 показывает реализацию запроса к API для логина пользователя.

```
export const API = {  
  auth: {  
    login: async ({username, password}) => {  
      const sendData = {  
        "username": username,  
        "password": password,  
      };  
      const {data} = await getAPIClient.post( url: '/auth/token/login', sendData);  
      return data;  
    },  
  },  
}
```

Рисунок 27 – Часть кода из api/index.js

Затем нужно создать файлы для маршрутизации страниц с помощью React Router. Например, файл "App.js" для определения маршрутов

приложения, который будет содержать все компоненты и определения маршрутов для каждого из них. Например, маршрут `"/games"` для отображения списка игр, маршрут `"/games/:id"` для отображения подробной информации об игре с определенным идентификатором и т.д. В реализации используется библиотека Chakra UI. Chakra UI является библиотекой компонентов пользовательского интерфейса, которая упрощает создание красивых и функциональных интерфейсов в React (рисунок 10). В данном проекте Chakra UI используется для создания компонентов, которые позволяют создать красивый и доступный интерфейс приложения. В файле `App.js`, который является корневым компонентом приложения, используются компоненты из Chakra UI для создания базового макета приложения. Например, компоненты `Box`, `Flex`, `Stack`, `Container` используются для создания блоков с заданными размерами и расположением. В файле `App.js` используется также компонент `ThemeProvider`, который обеспечивает глобальную настройку стилей приложения с помощью предустановленных тем. Компоненты Chakra UI могут настраиваться с помощью передачи им пропсов, таких как `bgColor`, `color`, `fontWeight`, `borderRadius`, `p`, `m` и других. Благодаря этому можно создавать различные стили компонентов для разных частей приложения. Таким образом, использование Chakra UI позволяет значительно ускорить и упростить создание пользовательского интерфейса в React, обеспечивая при этом хороший уровень доступности и качества.

В итоге, клиентская часть сервиса была реализована с учетом всех требований, предъявляемых к ней, и обеспечивает удобный и интуитивно понятный интерфейс для пользователей (рисунок 28), а также позволяет взаимодействовать с серверной частью сервиса через API.



Рисунок 28 – Пример отображения игры

3.4 Ручное тестирование сервиса iGaming

Ручное тестирование является важной частью процесса тестирования, поскольку позволяет обнаружить проблемы, которые не могут быть выявлены автоматическими тестами. Хотя автоматические тесты могут проверять определенные аспекты функциональности приложения, они не могут заменить взгляд человека на продукт и его способность воспринимать и обрабатывать информацию в контексте реального использования.

Ручное тестирование позволяет тестировщикам оценить, насколько хорошо приложение соответствует требованиям и спецификациям, а также насколько оно удобно для использования. Тестировщики могут проверять функциональность приложения, ручками заполнять формы, переходить по ссылкам и выполнять другие действия, как реальные пользователи. Это позволяет выявлять скрытые проблемы, такие как неочевидные ошибки интерфейса или недостаточно информативные сообщения об ошибках.

В случае данного сервиса iGaming, ручное тестирование поможет обнаружить возможные ошибки в работе интерфейса, проверить

соответствие функциональности требованиям бизнес-логики, а также проверить качество проекта в целом в контексте реального использования.

Например, следующими способами можно, например, протестировать регистрацию и авторизацию пользователя, а также создание и редактирование новой игры:

Тестирование регистрации пользователя:

- 1) Перейти на страницу регистрации
- 2) Ввести валидный email и пароль, подтверждение пароля
- 3) Нажать на кнопку "Зарегистрироваться"
- 4) Проверить, что появляется сообщение об успешной регистрации
- 5) Войти в приложение с использованием зарегистрированных ранее данных
- 6) Проверить, что вход прошел успешно

Тестирование входа пользователя:

- 1) Перейти на страницу входа
- 2) Ввести зарегистрированный email и пароль
- 3) Нажать на кнопку "Войти"
- 4) Проверить, что вход прошел успешно и пользователь перенаправлен на главную страницу

Тестирование создания игры:

- 1) Залогиниться под администратором
- 2) Перейти на страницу создания игры
- 3) Ввести название, описание, URL и тип игры
- 4) Нажать на кнопку "Создать"
- 5) Проверить, что игра добавлена успешно
- 6) Проверить, что игра отображается на главной странице

Тестирование редактирования игры:

- 1) Залогиниться под администратором
- 2) Найти игру на главной странице и нажать на кнопку "Редактировать"
- 3) Изменить какое-либо поле игры

- 4) Нажать на кнопку "Сохранить"
- 5) Проверить, что изменения сохранены успешно
- 6) Проверить, что игра отображается на главной странице с измененными данными

При тестировании был выявлен баг при добавлении игры. Если пользователь пытался добавить игру с неверным URL-адресом, то приложение не обрабатывало эту информацию и выдавало ошибку. До исправления бага не было понятно, что именно вызвало ошибку, и пользователь оказывается запутанным в том, что нужно исправить. Поэтому для его устранения была добавлена проверка на правильность введенного URL-адреса при сохранении данных в базу данных, чтобы избежать возможных ошибок при обработке данных.

Заключение

Данная работа по созданию сервиса iGaming была проведена с целью создания онлайн-платформы для игроков, желающих играть в онлайн-игры. Данный сервис позволяет пользователям зарегистрироваться, создавать профили, выбирать игры из широкого ассортимента и играть в них.

При разработке данного сервиса были использованы технологии Python, Django, React, Chakra UI, Axios и React Router. Для авторизации пользователей использовался пакет djoser. С помощью данной технологической платформы была разработана архитектура, состоящая из двух частей: серверной (backend) и клиентской (frontend).

В рамках данного исследования были описаны основные функциональные возможности, которые были включены в сервис. Была проведена разработка базы данных, созданы модели для игр, пользователей и их профилей. Также были описаны API-методы для получения, создания, изменения и удаления данных в базе данных.

После завершения разработки backend-части было проведено тестирование, включающее тестирование функциональности каждого API-метода. Также была проведена проверка безопасности приложения с помощью тестирования на уязвимости.

В результате разработки клиентской части был создан пользовательский интерфейс, который позволяет пользователям зарегистрироваться, авторизоваться, просматривать и выбирать игры, а также просматривать информацию об их профилях. Были использованы современные технологии, такие как React, Chakra UI, Axios и React Router.

В целом, разработка данного сервиса показала эффективность использования современных технологий и методов программирования при создании онлайн-платформы для игроков. Были достигнуты поставленные цели и задачи, и сервис готов к запуску. Однако, для дальнейшего совершенствования и развития, необходимо провести более широкое

тестирование, в том числе, с учетом различных условий и сценариев использования.

Список использованных источников

Электронные ресурсы:

- 1) Django documentation [Электронный ресурс] URL:
<https://docs.djangoproject.com/>
- 2) Django Rest Framework documentation [Электронный ресурс] URL:
<https://www.django-rest-framework.org/>
- 3) React documentation [Электронный ресурс] URL: <https://reactjs.org/>
- 4) Axios documentation [Электронный ресурс] URL:
<https://axios-http.com/docs/intro>
- 5) React Router documentation [Электронный ресурс] URL:
<https://reactrouter.com/>
- 6) Chakra UI documentation [Электронный ресурс] URL:
<https://chakra-ui.com/docs/getting-started>
- 7) djoser documentation [Электронный ресурс] URL:
<https://djoser.readthedocs.io/en/latest/>
- 8) Docker documentation [Электронный ресурс] URL:
<https://docs.docker.com/>
- 9) PostgreSQL documentation [Электронный ресурс] URL:
<https://www.postgresql.org/docs/>
- 10) NGINX documentation [Электронный ресурс] URL:
<https://docs.nginx.com/>