Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**Программное средство «GAME WORLD»**

|  |  |  |
| --- | --- | --- |
| Студент |  | М.Г. Гладкий |
| Руководитель |  | Д.В. Низовцов |

Минск 2021

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc72327202)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc72327203)

[1.1 Обзор аналогов 7](#_Toc72327204)

[1.2 Постановка задачи 11](#_Toc72327205)

[2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 12](#_Toc72327206)

[2.1 Структура приложения 12](#_Toc72327207)

[2.2 Разработка базы данных приложения 12](#_Toc72327208)

[2.3 Сервер программного средства 16](#_Toc72327209)

[2.4 Работа с комментариями и оценками 18](#_Toc72327210)

[2.5 Страница администратора 19](#_Toc72327211)

[2.6 Интерфейс программного средства 22](#_Toc72327212)

[3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 24](#_Toc72327213)

[3.1 Тестирование функционала приложения 24](#_Toc72327214)

[4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 26](#_Toc72327215)

[4.1 Порядок использования 26](#_Toc72327216)

[ЗАКЛЮЧЕНИЕ 30](#_Toc72327217)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 31](#_Toc72327218)

[ПРИЛОЖЕНИЕ А 32](#_Toc72327219)

ВВЕДЕНИЕ

21-й век – век информационных и телекоммуникационных технологий. Всё больше отраслей специализируются в компьютерной сфере, так как это наиболее актуальное и современное направление. Новейшие научные достижения безудержно преображают мир вокруг нас. На сегодняшний день можно с уверенностью сказать, что компьютеры проникли во все сферы деятельности человека, начиная с начального образования и заканчивая изучением новейших технологий, изучения новых видов материи, неизвестных пока человечеству. Благодаря разнообразию программного и аппаратного обеспечения сегодня возможно использование всех потенциальных возможностей компьютерных технологий. Это позволяет хранить огромное количество информации, занимая при этом минимальное место. Также компьютерные технологии позволяют быстро эту информацию обрабатывать и держать ее в защищенном виде. На разработку новых проектов затрачивается гораздо меньше времени, ибо не надо тратить массу времени на вычислительные процессы и можно полностью посвятить время самому процессу. Большую роль компьютерные технологии играют в медицине, создаются различные виртуальные модели развития заболеваний, создаются огромные базы информации, на основании которых изобретаются новые препараты для лечения. Также они не обошли сферу развлечений, в которую добавили нечто новое – видеоигры.

История компьютерных игр начинается ещё в далёких 1950-х с разработки простейших игр и симуляций. Данный вид времяпрепровождения не сразу стал популярным, потребовалось несколько десятков лет, чтобы он вышел в свет и стал частью жизни молодёжи. В дальнейшем компьютерные игры получили большое развитие. Компании по всему миру соревновались в выпуске передовых игр, приставок и консолей. Наблюдались как упадки, так и резкие возрастания популярности видеоигр. Однако индустрия никогда не стояла на месте. Разработчики всё время пытались привнести что-то новое, создать нечто уникальное. По сей день существует конкуренция между ведущими компаниями в данной сфере.

Игроки часто задумываются о том, какую игру им приобрести, насколько она хороша, интересна ли и многое другое. Для этого в интернете существуют специализированные форумы. В них люди со всего мира оценивают игры, оставляют свои отзывы, дискутируют по поводу различных проблем. Всё это помогает пользователю принять решение о покупке какого-либо продукта игровой индустрии. Форумов существует большое множество, как и обсуждений в них. Это позволяет лучше оценить игру по заданным критериям.

Игры могут приобретаться как в обычных магазинах, так и в интернет-магазинах. Самые крупные площадки по продаже компьютерных игр: Steam, Origin, Windows Store, Epic Games Store, Xbox Store, PS Store. В каждом из этих магазинов устанавливаются разные предложения на один и тот же товар, пользователи могут оставить свои впечатления об игре, произвести поиск по жанрам игр, возрастному рейтингу, цене и большому количеству критериев. В таких крупных магазинах часто проводятся различные акции и распродажи, что привлекает большое число потенциальных покупателей.

Целью данного проекта является создание программного средства «Game World».

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ
   1. Обзор аналогов

На данный момент существует множество вариаций приложений для обсуждения игр, выставления им оценок, покупки цифровых изданий популярных компьютерных игр.

Одним из наиболее популярных является площадка Steam – онлайн-сервис цифрового распространения компьютерных игр и программ, разработанный и поддерживаемый компанией Valve. Steam выполняет роль средства технической защиты авторских прав, платформы для многопользовательских игр и потокового вещания, а также социальной сети для игроков. Программный клиент Steam также обеспечивает установку и регулярное обновление игр, облачные сохранения игр, текстовую и голосовую связь между игроками.

На начало 2021 года через Steam предлагалось к продаже свыше 50 тысяч игр, и это число прирастало на 8-10 тысяч каждый год. В 2020 году количество активных пользователей сервиса, использовавших его хотя бы раз в месяц, превышало 120,4 миллиона.

Внешний вид данного программного средства представлен на рисунке 1.1.

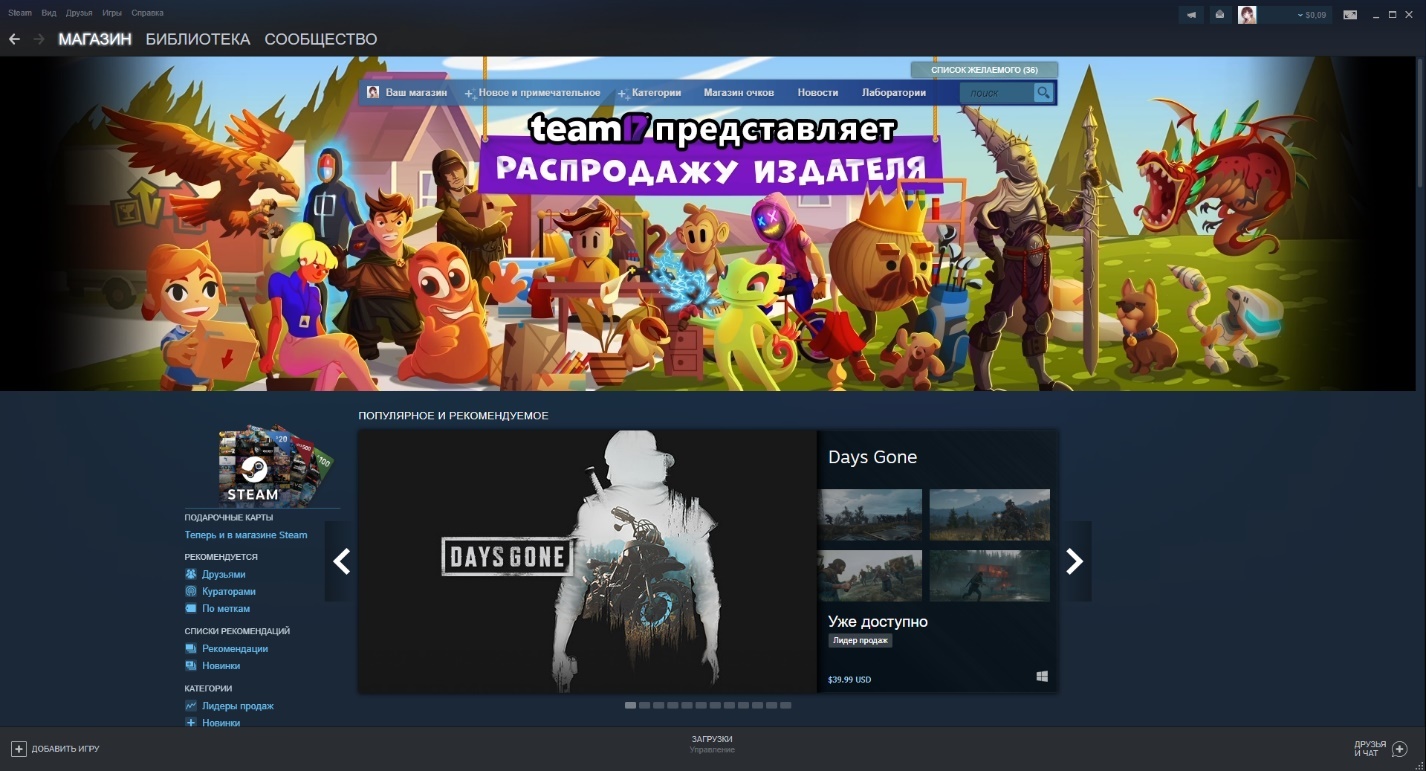


Рисунок 1.1 – Внешний вид программного средства Steam

Преимущества Steam:

– простой и понятный пользовательский интерфейс, не вызывающий трудностей у новых пользователей;

– быстрый процесс поиска и покупки цифровых продуктов;

– присутствует возможность поиска игр по жанрам, по тэгам, например: кооператив, музыка и т.п.;

– полноценная библиотека игр, в которой хранятся все приобретённые продукты;

– предоставление возможности онлайн игры с другими пользователями или друзьями бесплатно;

– поддержка большого количества устройств, включая устройства на операционных системах Windows, Linux, MacOS, iOS, Android;

– сообщество с большим множеством групп и обсуждений по отдельно взятым игровым продуктам.

Одним из менее популярных аналогов является Origin — платформа цифровой дистрибуции компании Electronic Arts, которая даёт возможность пользователям приобретать компьютерные игры через Интернет и загружать их с помощью клиента Origin (ранее EA Download Manager, EA Downloader и EA Link). 3 июня 2011 EA Store был переименован в Origin. В 2011 году компания Electronic Arts заявила, что хотела бы, чтобы Origin соответствовал их главному конкуренту, сервису Steam, путём добавления облачных сохранений, автообновления игр, достижений и кроссплатформенных релизов. В Origin используются социальные функции, такие как управление профилем, общение с друзьями в чате и во время игры с помощью внутригрового приложения, интеграция с Facebook, Xbox Live и PlayStation Network.

На данный момент имеется возможность автообновления игр, синхронизация сохранений игр в облачном хранилище, общение во встроенном чате. К 2013 году, в Origin было зарегистрировано более 50 миллионов пользователей.

Electronic Arts планирует также запустить Origin платформу для мобильных устройств (таких, как iOS/Android), и достижения могут быть синхронизированы на обеих платформах.

Внешний вид главной страницы данного сервиса представлен на рисунке 1.2.

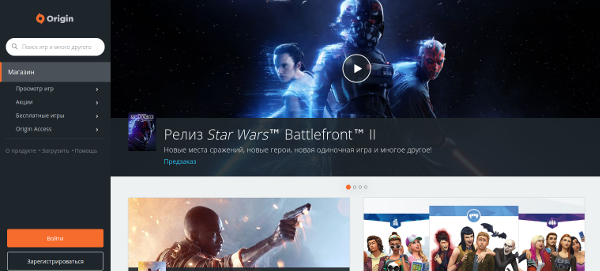


Рисунок 1.2 – Внешний вид главной страницы Origin

Не менее интересным и быстро развивающимся соперником является Epic Games Store. Магазин доступен в качестве веб-сайта и через приложение Epic Games Launcher. В данный момент у него нет таких функций, как достижения или поддержка гарнитуры виртуальной реальности.

Собственные игры Epic Games будут доступны исключительно через данный магазин, и компания планирует финансировать релизы только через их магазин, используя гарантии дохода для разработчиков, которые выбирают данные условия, при этом Epic платит разницу, если игра не оправдывает ожиданий по доходам.

Для популяризации собственного магазина Epic Games устроила акцию, в которой предлагает по одной или несколько бесплатных игр в неделю (изначально, раз в две недели) с 2019 года и по сей день.

В конце концов, витрина магазина будет предлагать отзывы пользователей, но разработчики включат эту функцию, чтобы избежать неправомерного использования таких действий, как ревью-бомбинг. Дополнительные запланированные функции включают в себя сохранение в облаке и достижения; тем не менее, Epic Games не планирует добавлять какие-либо «игровые функции» вроде карточек Steam, предназначенные для стимулирования продаж.

Там, где это возможно, Epic Games планирует расширить свою программу «Поддержка создателя», запущенную в Fortnite Battle Royale, на другие игры, предлагаемые в магазине. Игроки смогут поддержать стримера или создателя контента, выбранного компанией на основе представленных приложений. Поддерживаемые авторы смогут получать доход от микротранзакций, совершенными через Epic Games Store от игроков, которые их поддерживают, стимулируя создателей контента. В Fortnite создатели получили около 5 % от стоимости микротранзакции.

Главное меню программного средства Epic Games Launcher представлено на рисунке 1.3.

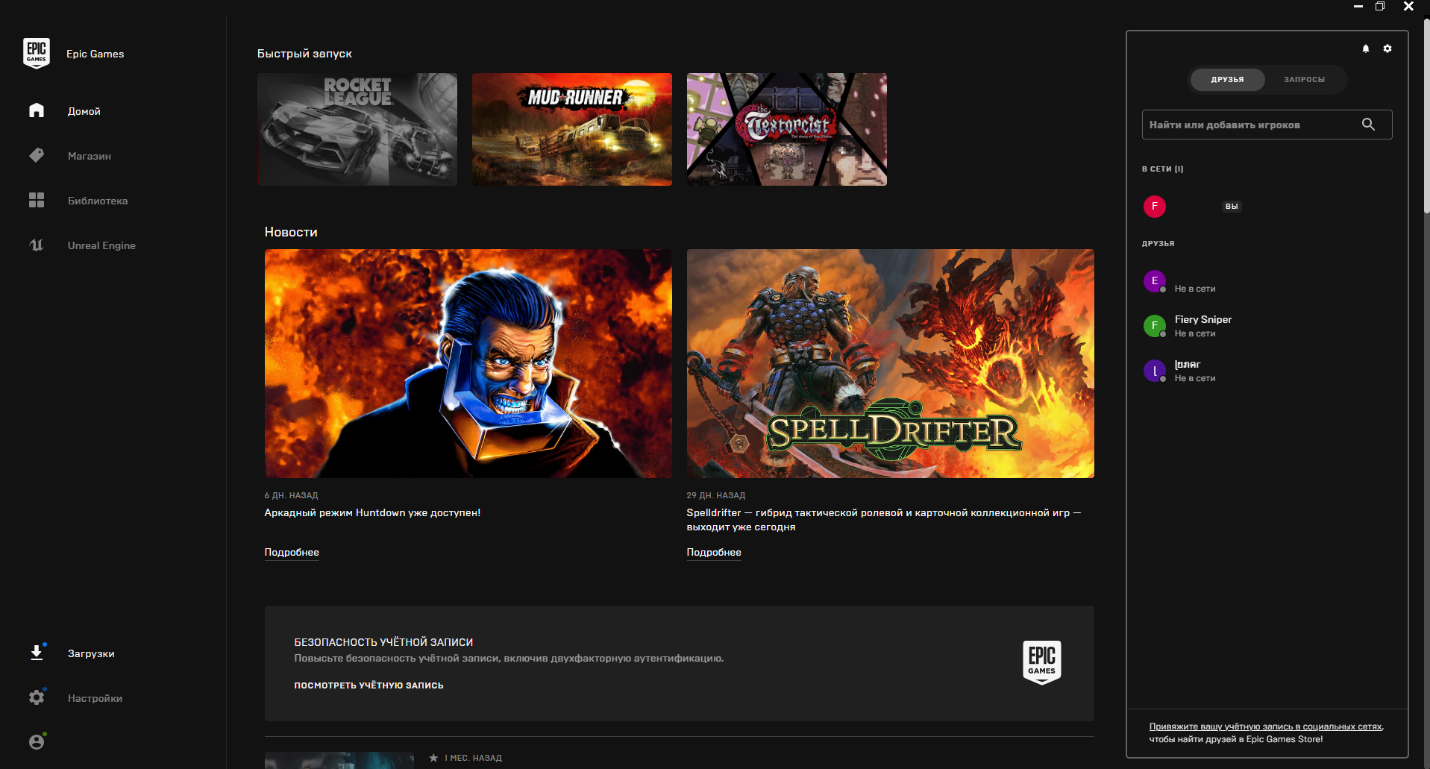


Рисунок 1.3 – Главное меню Epic Games Launcher

Преимущества Epic Games Store:

– удобное расположение элементов в окне;

– сниженная комиссия для разработчиков игр;

– бесплатные раздачи игр каждую неделю;

* 1. Постановка задачи

В рамках данного курсового проекта планируется разработка программного средства «Game World».

Исходя из рассмотренных аналогов можно определить основные требования к проекту.

В клиентской части планируется реализовать следующие функции:

* регистрация;
* вход в приложение;
* выход из аккаунта;
* создание игры;
* возможность выставления рейтинга, комментария к игре;
* поиск игр и других пользователей с применением фильтров;

В административной части планируется реализовать следующие функции:

* регистрация как администратор;
* доступ ко всем возможностям обычного пользователя;
* возможность удаления пользователей;
* возможность удаления комментариев;
* возможность удаления игр;

Логическая составляющая приложения будет включать в себя:

* алгоритмы обработки графических ресурсов;
* передачу данных по сети;
* синхронизацию передачи данных.

Для разработки программного средства будет использоваться язык программирования JavaScript с использованием возможностей среды NodeJS, фреймворка Express, React, база данных mongoDB и среда разработки Visual Studio Code [1].

1. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА
2. 1. Структура приложения

Данное приложение включает в себя серверную и клиентскую части. В клиентской части, представленной приложением для браузера, также реализован доступ к возможностям администратора приложения[2].

Структура серверной части:

– index.js – контроллер, обрабатывающий запросы на добавление, удаление и изменение пользователей, игр, рейтинга, комментариев.

– middlewares – файлы промежуточных обработчиков запросов, позволяющих обработать аутентификацию, авторизацию, состояние пользователей и отлов ошибок;

Структура клиентской и административной части:

– components – файлы, содержащие разметку, стилизацию и логику переиспользуемых компонентов приложения, таких как кнопки, текст, карточки, поля ввода и другие;

– assets – папка, содержащая все изображения, иконки и шрифты приложения;

– pages – файлы, содержащие разметку страниц приложения, применяющие переиспользуемые компоненты, а также логику обработки данных на этих экранах;

* 1. Разработка базы данных приложения

На первом шаге разработки базы данных была проанализирована предметная область приложения и составлена концептуальная модель базы данных в виде таблицы списка.

Таблица 2.1 – Концептуальная модель базы данных

|  |  |  |  |
| --- | --- | --- | --- |
| **Сущность** | **Характеристики** | **Взаимосвязи** | **Уточнения** |
| Пользователь | имя  электронная почта  хэш-пароль  является ли администратором | Связан с сущностью игры, комментарий,  рейтинг | Пароль храниться в виде зашифрованной строки алгоритмом bcrypt |
| Игра | заголовок  картинка  платформа  жанр  возрастной рейтинг  цена  описание  идентификатор пользователя, добавившего игру | Комментарии, рейтинги, пользователь | Картинка представлена ссылкой на файл. |
| Рейтинг | идентификатор игры,  идентификатор пользователя,  значение | Пользователь,  игра | Значение находится в диапазоне от 1 до 5 |
| Комментарий | идентификатор игры, идентификатор пользователя, отображаемое имя пользователя, текст комментария | Пользователь,  игра | Максимальная длина сообщения 1000 символов |
| Заказ | игры в заказе, статус заказа, общая стоимость, идентификатор пользователя | Пользователь, игра |  |
| Скидка | размер, идентификатор игры | Игра | Указывается вещественным числом от 0 до 1 |

Концептуальная модель в виде UML-диаграммы представлена на рисунке 2.1.

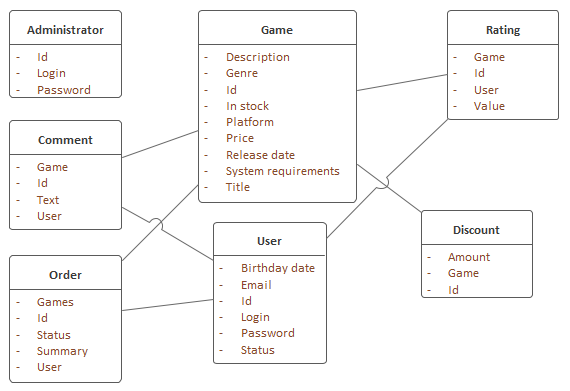


Рисунок 2.1 – Концептуальная модель в виде UML-диаграммы

В процессе разработки приложения было принято решение использовать NoSQL базу данных mongoDB.

MongoDB – документно-ориентированная система управления базами данных, не требующая описания схемы таблиц. Считается одним из классических примеров NoSQL-систем, использует JSON-подобные документы и схему базы данных. Написана на языке C++. Применяется в веб-разработке, в частности, в рамках JavaScript-ориентированного стека MERN.

Выбор обусловлен рядом причин:

– Система поддерживает ad-hoc-запросы: они могут возвращать конкретные поля документов и пользовательские JavaScript-функции.

– Поддерживается поиск по регулярным выражениям. Также можно настроить запрос на возвращение случайного набора результатов.

– Имеется поддержка индексов.

– Система может работать с набором реплик, то есть содержать две или более копии данных на различных узлах. Каждый экземпляр набора реплик может в любой момент выступать в роли основной или вспомогательной реплики. Все операции записи и чтения по умолчанию осуществляются с основной репликой. Вспомогательные реплики поддерживают в актуальном состоянии копии данных. В случае, когда основная реплика дает сбой, набор реплик проводит выбор, которая из реплик должна стать основной. Второстепенные реплики могут дополнительно являться источником для операций чтения.

– Система масштабируется горизонтально, используя технику сегментирования объектов баз данных – распределение их частей по различным узлам кластера. Администратор выбирает ключ сегментирования, который определяет, по какому критерию данные будут разнесены по узлам. Благодаря тому, что каждый узел кластера может принимать запросы, обеспечивается балансировка нагрузки.

– Поддерживается JavaScript в запросах, функциях агрегации [3].

Для проверки результатов обработки запросов к базе данных, правильности введенных данных было использовано приложение MongoDB Compass. Это графический интерфейс для MongoDB. Compass позволяет анализировать и понимать содержимое данных без формального знания синтаксиса запросов. Помимо изучения данных в визуальной среде, также можно использовать Compass для оптимизации производительности запросов, управления индексами и реализации проверки документов. Внешний вид приложения показан на рисунке 2.2:

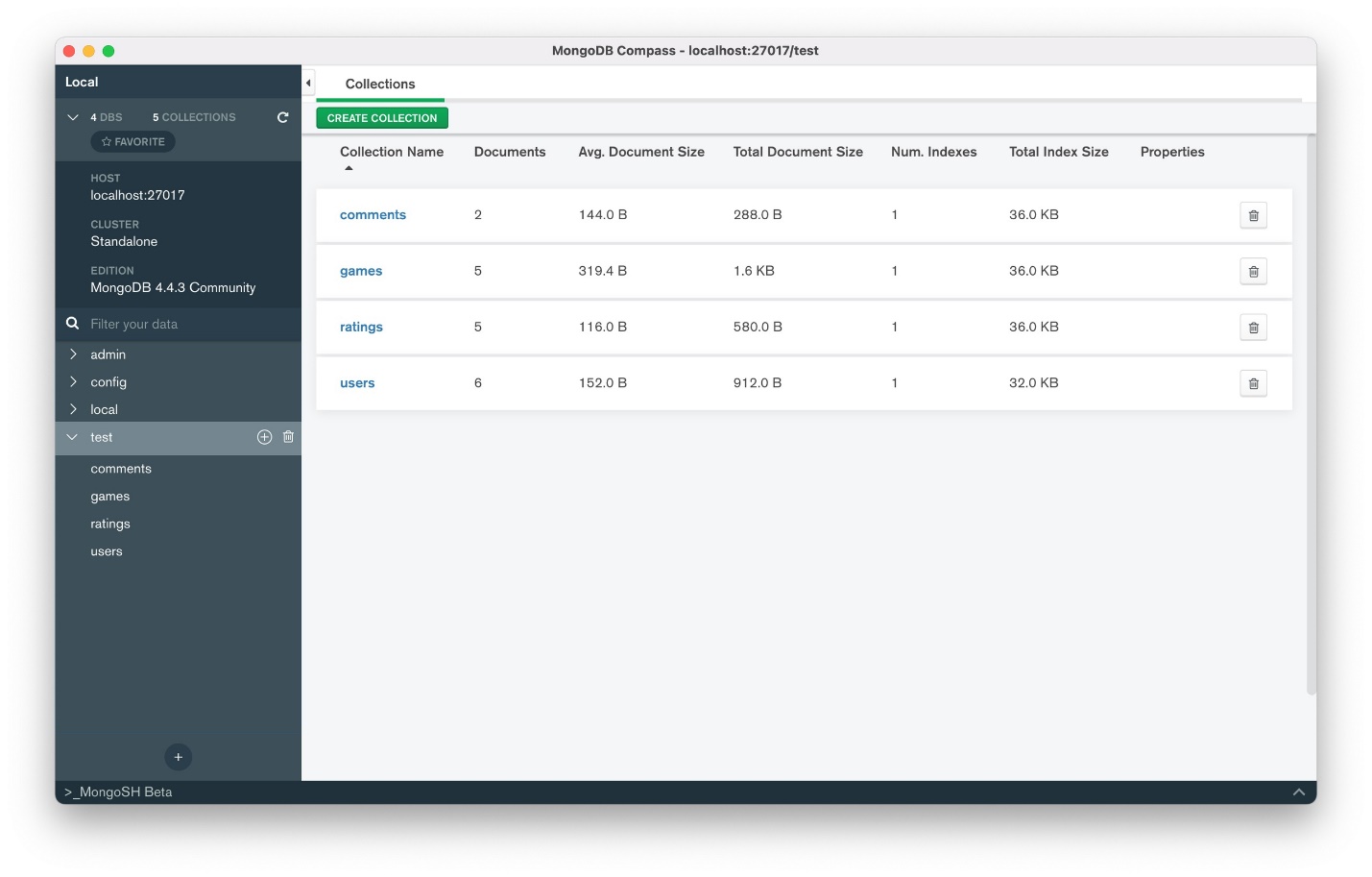


Рисунок 2.2 – Внешний вид MongoDB Compass.

* 1. Сервер программного средства

При разработке сервера приложения была выбрана платформа Node js, а также фреймворк express. Данные технологии позволяют создавать надёжные, быстрые и надёжные серверные приложения.

Клиентское приложение общается с сервером по средством HTTP запросов. При разработке был использован архитектурный паттерн REST.

Representational state transfer – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

На каждый запрос был разработан обработчик, который проверяет входные данные, а именно тело запроса. Если в теле запроса обнаруживается ошибка, то сервер возвращает клиенту ответ со статусом 400 и кодом ошибки в теле ответа[4].

Только авторизованный пользователь может создавать игры, оставлять комментарии и оценки. Для этого пользователю нужно зарегистрироваться. После регистрации или входа в приложение, на сервере генерируется JWT токен и отправляется клиенту посредством HTTP заголовка.

JSON Web Token – это JSON объект, который определен в открытом стандарте RFC 7519. Он считается одним из безопасных способов передачи информации между двумя участниками. Для его создания необходимо определить заголовок с общей информацией по токену, полезные данные, такие как id пользователя, его подписи.

Для генерации и проверки JWT токенов на подлинность был использован пакет jsonwebtoken. Он предоставляет метод sign для генерации JSON объекта подписанным секретным ключом. Затем JWT токен можно проверить с использованием метода verify и секретного ключа.

Не все обработчики запросов должны проверять наличие JWT у пользователя, например, у каждого пользователя должна быть возможность зарегистрироваться, войти в систему, производить поиск по играм и просматривать их. Поэтому был использован паттерн Middleware.

Паттерн Middleware заключается в предварительной обработке запроса. Middleware выступают в качестве функций, которым в качестве аргумента передаётся параметр next, который является функцией для вызова следующего обработчика. Таким образом middleware выступают в качестве фильтра для обработчиков запросов.

На следующих рисунках схематически представлена работа c JWT токеном в клиент-серверной архитектуре, а также схема паттерна Middleware.

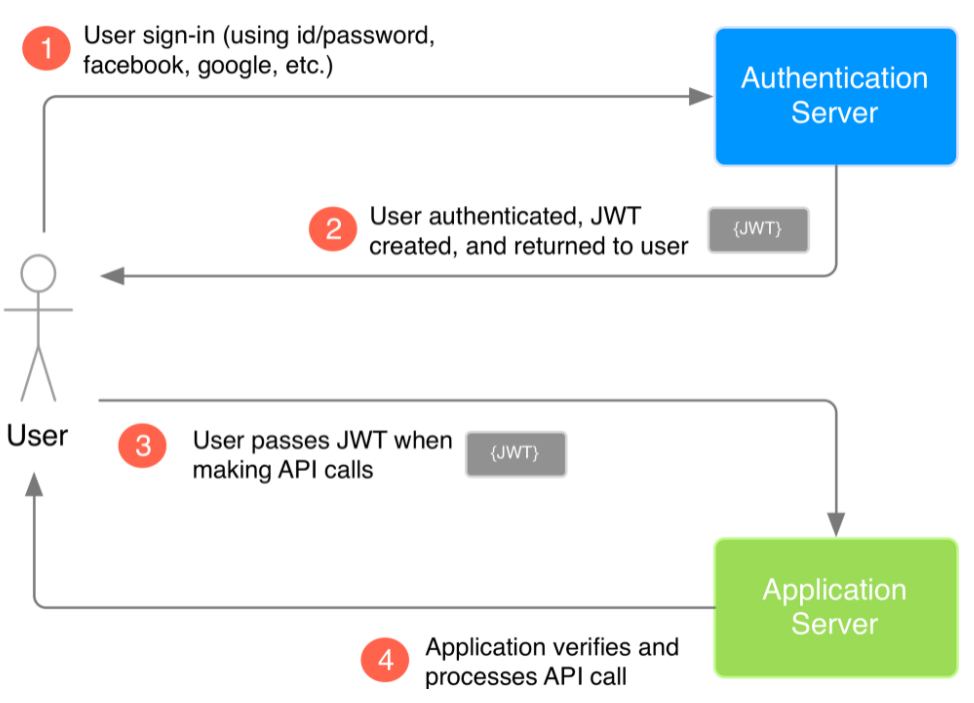


Рисунок 2.3 – Схема работы с JWT токеном

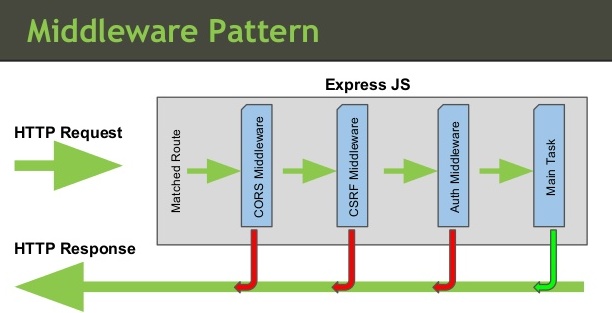


Рисунок 2.4 – Схема паттерна Middleware

* 1. Работа с комментариями и оценками

Обработка комментариев и оценок выполнена с использованием библиотеки socket.io на стороне сервера и socket.io-client на стороне клиента. Socket.IO – это библиотека, которая обеспечивает двустороннюю связь между браузером и сервером в реальном времени и на основе событий.

Клиент попытается установить соединение WebSocket, если это возможно, и в противном случае вернется к длинному опросу HTTP. WebSocket – это протокол связи, который обеспечивает полнодуплексный канал с малой задержкой между сервером и браузером.

После перехода пользователя на страницу с игрой происходит установление соединения с сервером. В случае успеха клиент посылает два сообщения, содержащих в себе идентификатор выбранной игры. В одном из них запрашиваются комментарии, в другом запрашивается средний рейтинг игры.

Сервер производит запрос в базу данных, получая необходимые данные и производя вычисления над ними. После этого отвечает на полученное сообщение новым сообщением, содержащим все необходимые данные для клиента, такие как: список комментариев, значение среднего рейтинга.

После того, как клиент получил ответ от сервера, происходит обновление состояния компонента и отрисовываются полученные комментарии, рейтинг на странице.

Ключевая особенность использования данной библиотеки – возможность обновления данных в реальном времени на глазах у пользователя. В частности, после написания комментария и отправки его на сервер, он сразу же отображается в блоке комментариев последним. То же самое происходит и при удалении комментария, он моментально убирается из списка на странице.

Блок комментариев и рейтинг показаны на рисунке 2.5:

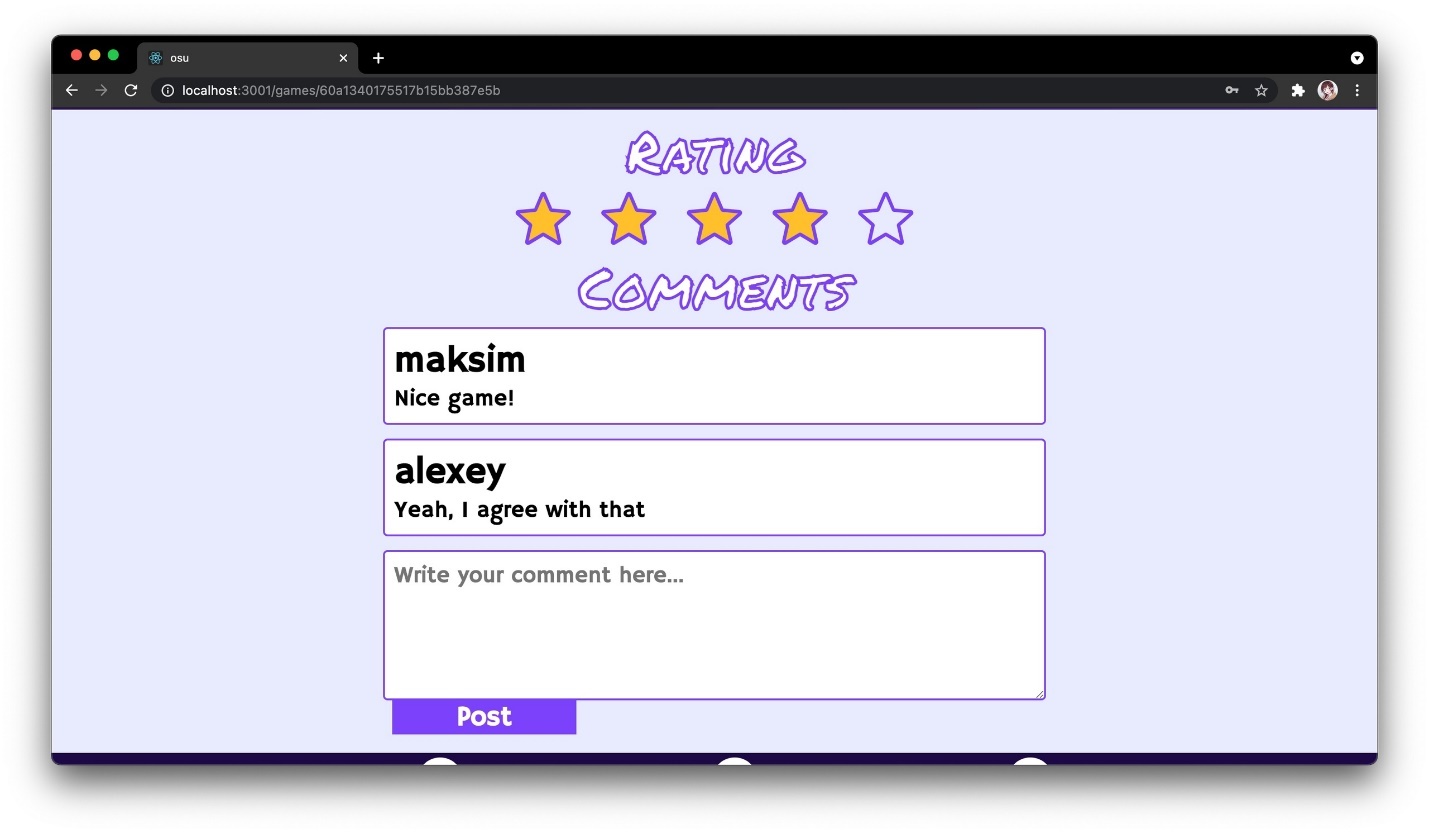


Рисунок 2.5 – Блок комментариев и рейтинг

* 1. Страница администратора

Администратор может попасть на специальную панель по нажатии кнопки сверху слева экрана. На этой панели имеется 2 таблицы: игры и пользователи. Взаимодействие сервера и клиента на данной странице осуществляется благодаря GraphQL. Это язык запросов для API и среда выполнения на стороне сервера для исполнения запросов с использованием системы типов, которая определена для данных. GraphQL не привязан к какой-либо конкретной базе данных или механизму хранения, а вместо этого поддерживается существующим кодом и данными.

Служба GraphQL создается путем определения типов и полей для этих типов, а затем предоставления функций для каждого поля каждого типа.

Таким образом данный сервис является надстройкой над существующим сервером, который отделяет нужные данные от ненужных, позволяет выполнять преобразования в базе данных и скрывает всю логику обработки на стороне сервера, предоставляя простой API для клиента.

Реализован данный паттерн с помощью библиотеки apollo-server-express на стороне сервера и @apollo/client у клиента. Для корректной работы на сервере объявлены типы с их полями, а также обработчики запросов, которые может выполнять клиент.

Участок кода с объявленными типами прилагается ниже:

const typeDefs = gql`

type Query {

games: [Game!]!

users: [User!]!

}

type Mutation {

deleteGame(gameId: ID!): Game!

deleteUser(userId: ID!): User!

}

type Game {

id: ID!

title: String!

platform: String!

genre: String!

maturity: Int!

price: Int!

desc: String

}

type User {

id: ID!

nickname: String!

email: String!

password: String!

isAdmin: Boolean!

}

`;

После того, как администратор попал на панель, производится запрос к серверу на получение списка доступных игр и пользователей. Сервер производит запрос к базе данных и высылает необходимые данные на сторону клиента. Далее происходит обновление состояния компонента страницы и отрисовывается таблица с играми. Переключение на таблицу пользователей происходит по нажатию на соответствующую кнопку Users.

На данной странице администратор имеет право удалить любую существующую игру на данный момент. При этом из базы данных будут удалены все оценки и комментарии, принадлежащие выбранной игре. Также администратор, после переключения на таблицу с пользователями, может удалить рядовых пользователей, однако других администраторов удалять запрещено. Данный случай был предусмотрен для того, чтобы администратор не мог остаться единственным в системе и отобрать права у других пользователей. После нажатия на кнопку удаления происходит запрос к серверу, который убирает выбранного пользователя из базы данных, а также все оставленные им рейтинги и комментарии. Это необходимо для устранения аномалий удаления, когда ненужные данные остаются в базе данных.

Внешний вид панели администратора прилагается на следующих рисунках:

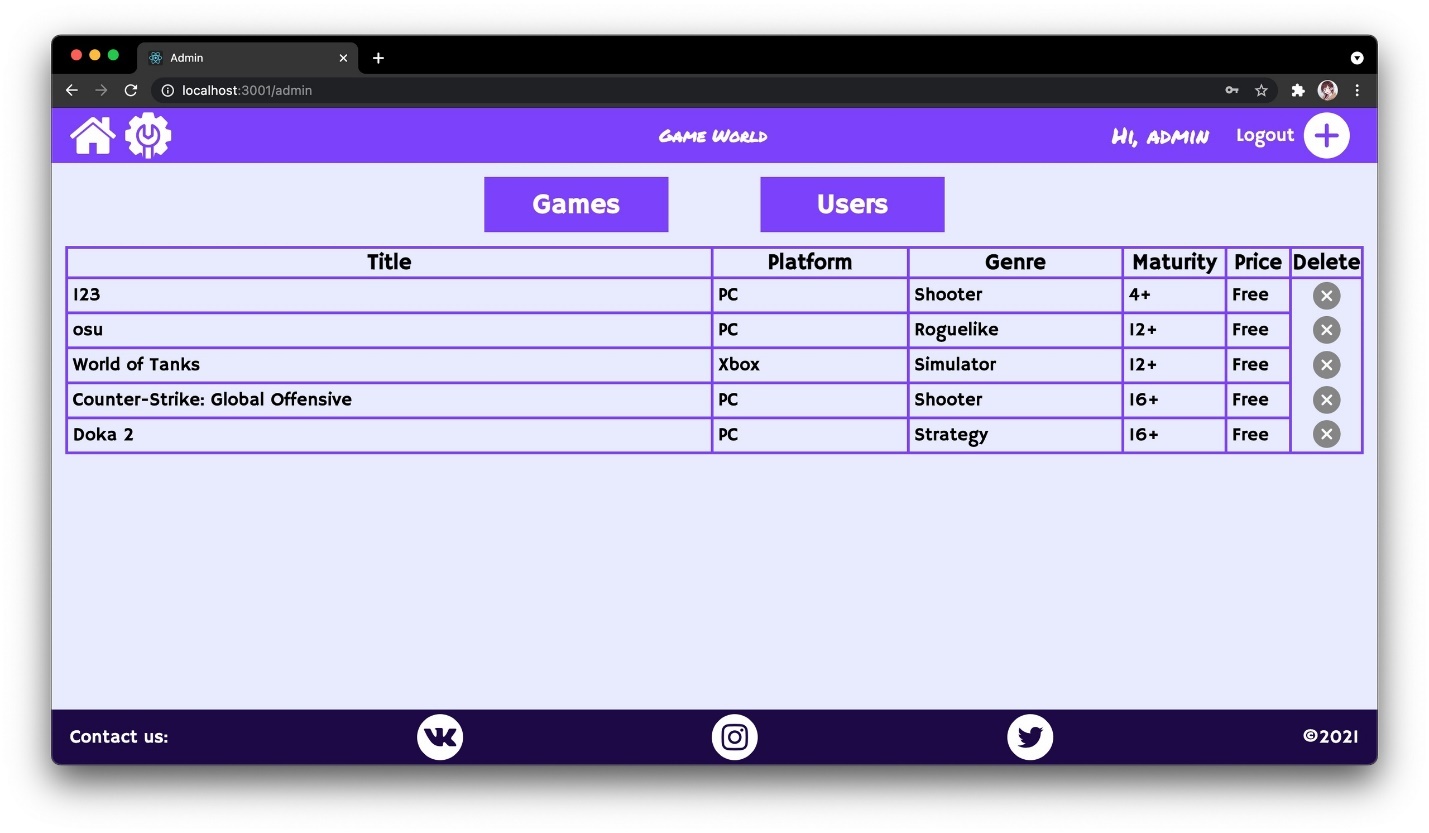


Рисунок 2.6 – Панель администратора с отображенными играми

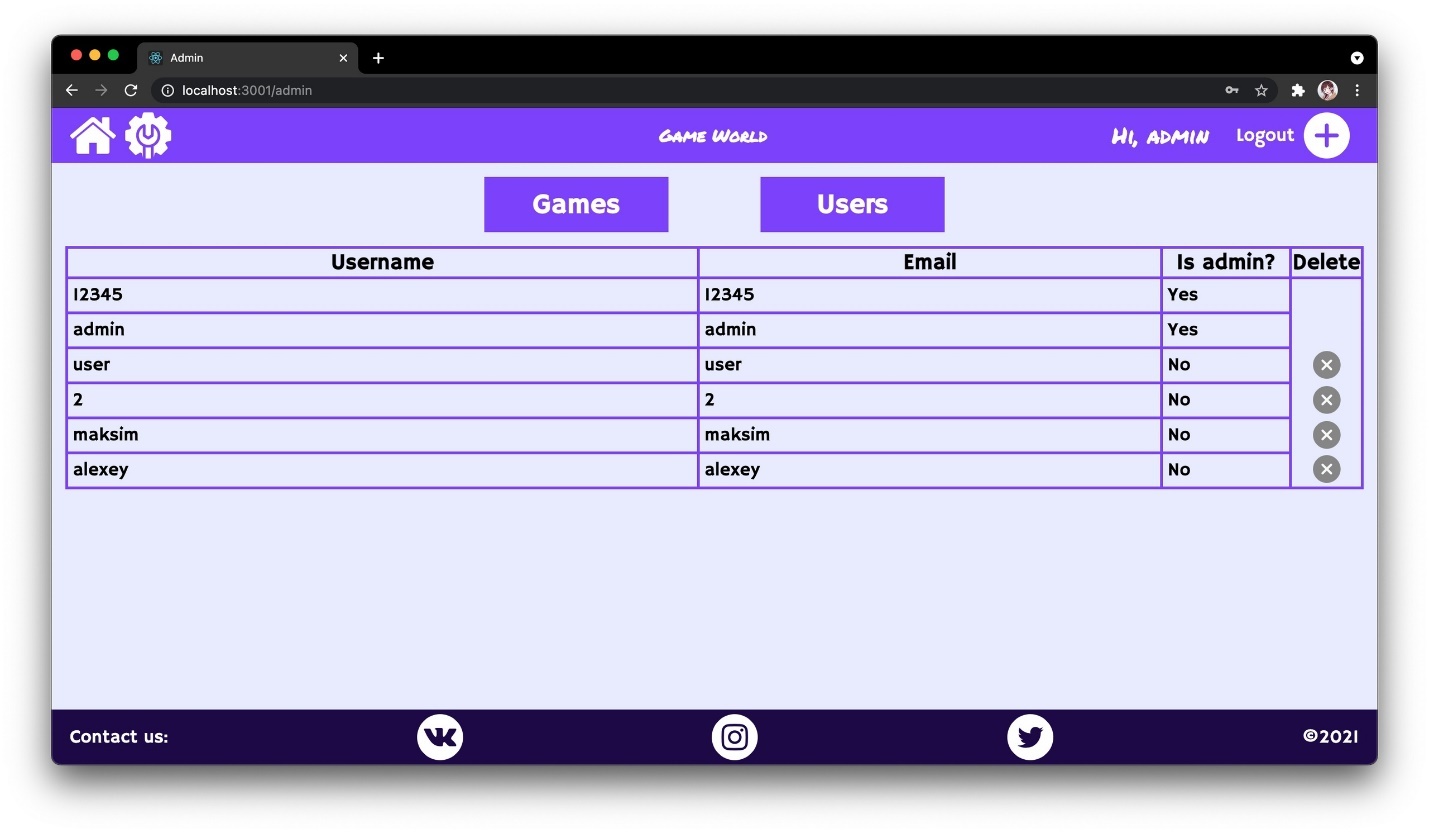


Рисунок 2.7 – Панель администратора с выбранными пользователями

* 1. Интерфейс программного средства

Для разработки программного средства были использованы библиотека React JS и стилизация CSS.

Необходимо было создать приложение, в котором после перехода на другую страницу она не перезагружалась. Это было достигнуто с использованием библиотеки react-router-dom, из которой были задействованы компоненты Router, Switch, Route, которые отслеживают состояние адресной строки браузера и отображают выбранные компоненты[5].

На каждой странице присутствует Header и Footer, которые содержат необходимые элементы взаимодействия с пользователем. В компоненте Footer содержатся ссылки на социальные сети разработчиков приложения, а в Header ссылки на доступные страницы, кнопка выхода из аккаунта и др.

На рисунке 2.8 представлена главная страница приложения.

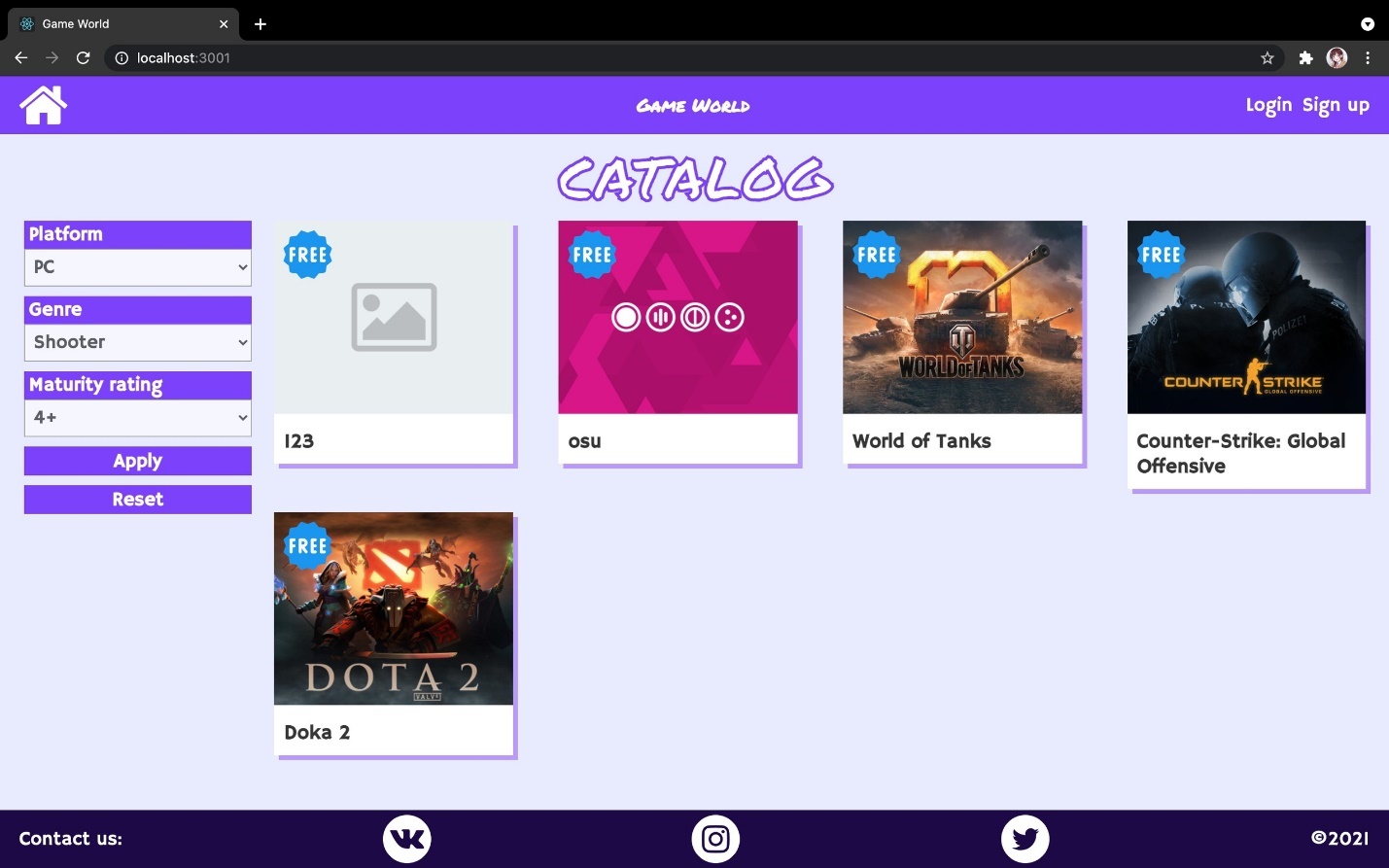


Рисунок 2.8 – Главная страница приложения

Также были созданы компоненты для других страниц, включая такие, как: страница добавления игры, страница с игрой, страницы логина и регистрации, панель администратора.

Все страницы стилизованы с использованием CSS. Это позволяет приложению иметь привлекательный и необычный вид. Для достижения данной цели были созданы отдельные файлы с расширением .css, в которых задаются на специальном языке стили, поддерживаемые браузером и которые можно изменить.

Для удобства написания данного кода было принято использовать BEM-нотацию. Она заключается в том, что класс элементов задается по следующему принципу: первое слово – блок, к которому относится элемент; второе слово – элемент, к которому применяется класс; третье слово – модификатор элемента, например active. Таким образом классовые селекторы выглядят следующим образом: .description\_\_p\_about, .description\_\_p\_title.

CSS позволяет добиться интерактивности с пользователем, что и было использовано следующим образом: при наведении на интерактивный элемент курсор на экране меняется на pointer, элемент подсвечивается или изменяет свой размер. Реализовано это при помощи использования псевдокласса :hover, который получают элементы при наведении на них курсором мыши. Для плавности данных преобразований было использовано css-свойство transition, которое позволяет установить время и тип перехода между состояниями элемента.

В ходе разработки приложения возникла проблема, заключавшаяся в том, что на устройствах с разным разрешением экрана элементы могли наезжать друг на друга и интерфейс был неразличим. В связи с этим было приятно решение использовать медиа-запросы, которые применяют стили в соответствии с текущим разрешением окна. Это позволило сделать сервис адаптивным к любым устройствам. Главная страница приложения при меньшем разрешении экрана представлена на рисунке 2.9:

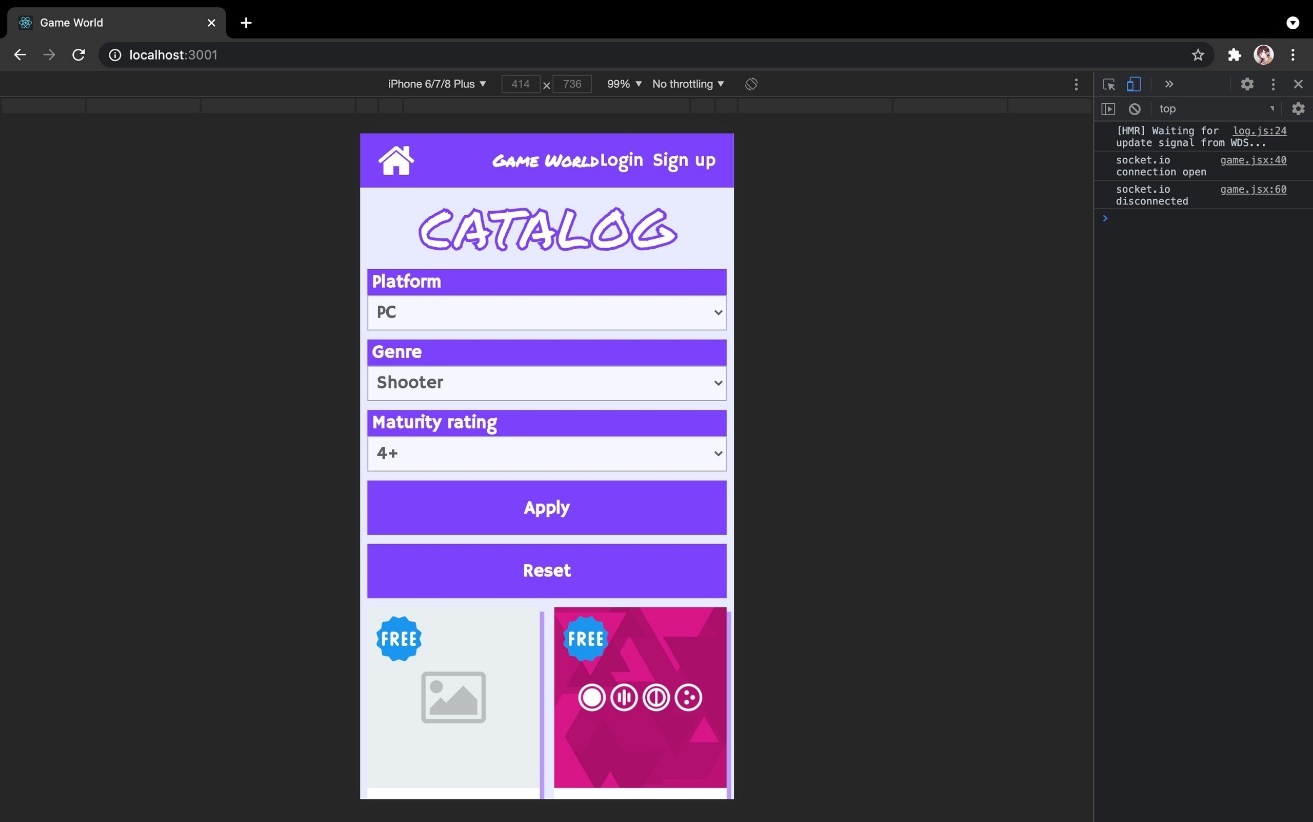


Рисунок 2.9 – Главная страница приложения при меньшем разрешении

1. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Тестирование функционала приложения

В ходе тестирования приложения были выявлены некоторые недостатки интерфейса и некорректная работа основной логики программного средства. Была составлена таблица 3.1, показывающая ожидаемые и реальные результаты, полученные при заданных условиях. Таблица прилагается далее:

Таблица 3.1 – Ожидаемые и реальные результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестовые случаи | Ожидаемый результат | Полученный результат |
| 1. | Нажатие кнопки Login | Переход на страницу авторизации | Переход на страницу авторизации |
| 2. | Нажатие кнопки Sign up | Переход на страницу регистрации | Переход на страницу регистрации |
| 3. | Нажатие кнопки Apply на домашней странице | Отображение игр по заданным фильтрам | Отображение игр по заданным фильтрам |
| 4. | Нажатие кнопки Reset | Отображение всех доступных игр | Отображение всех доступных игр |
| 5. | Нажатие кнопок переключения карты | Изменение превью карты | Изменение превью карты |
| 6. | Нажатие кнопки Sign in на странице авторизации с неправильно введенным паролем | Отображение окна с предупреждением | Ничего не происходит |
| 7. | Нажатие кнопки Sign in на странице авторизации с правильно введенным паролем и эл. почтой | Переход на главную страницу и замена иконок в Header | Переход на главную страницу и замена иконок в Header |
| 8. | Нажатие кнопки Submit на странице регистрации | Вход в аккаунт и переход на главную страницу | Вход в аккаунт и переход на главную страницу |
| 9. | Нажатие кнопки Home на любой странице | Перенаправление на домашнюю страницу | Перенаправление на домашнюю страницу |
| 10. | Нажатие на звездочку на странице с игрой | Выставление оценки игре, полоска показывает поставленный рейтинг | Выставление оценки игре, полоска показывает поставленный рейтинг |
| 11. | Нажатие кнопки Post в блоке комментариев на странице с игрой | Появление нового комментария в конце списка | Появление нового комментария в конце списка |
| 12. | Нажатие на крестик над комментарием | Исчезновение комментария из списка | Исчезновение комментария из списка |
| 13. | Нажатие на крестик над игрой на главной странице | Исчезновение игры из списка | Исчезновение игры из списка |
| 14. | Нажатие на кнопку Admin | Переход на панель администратора | Переход на панель администратора |
| 15. | Нажатие на кнопку Users в панели администратора | Смена таблицы с играми на таблицу с пользователями | Смена таблицы с играми на таблицу с пользователями |
| 16. | Нажатие на кнопку Games в панели администратора | Смена таблицы с пользователями на таблицу с играми | Смена таблицы с пользователями на таблицу с играми |
| 17. | Нажатие кнопки Logout | Возврат на главную страницу и выход из аккаунта | Возврат на главную страницу и выход из аккаунта |

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ
   1. Порядок использования

Главная страница приложения имеет следующие секции: раздел с играми, раздел фильтров, интерактивные кнопки. После нажатия на кнопку Login или Sign up пользователь перемещается на страницу авторизации или регистрации, показанные на следующих рисунках:

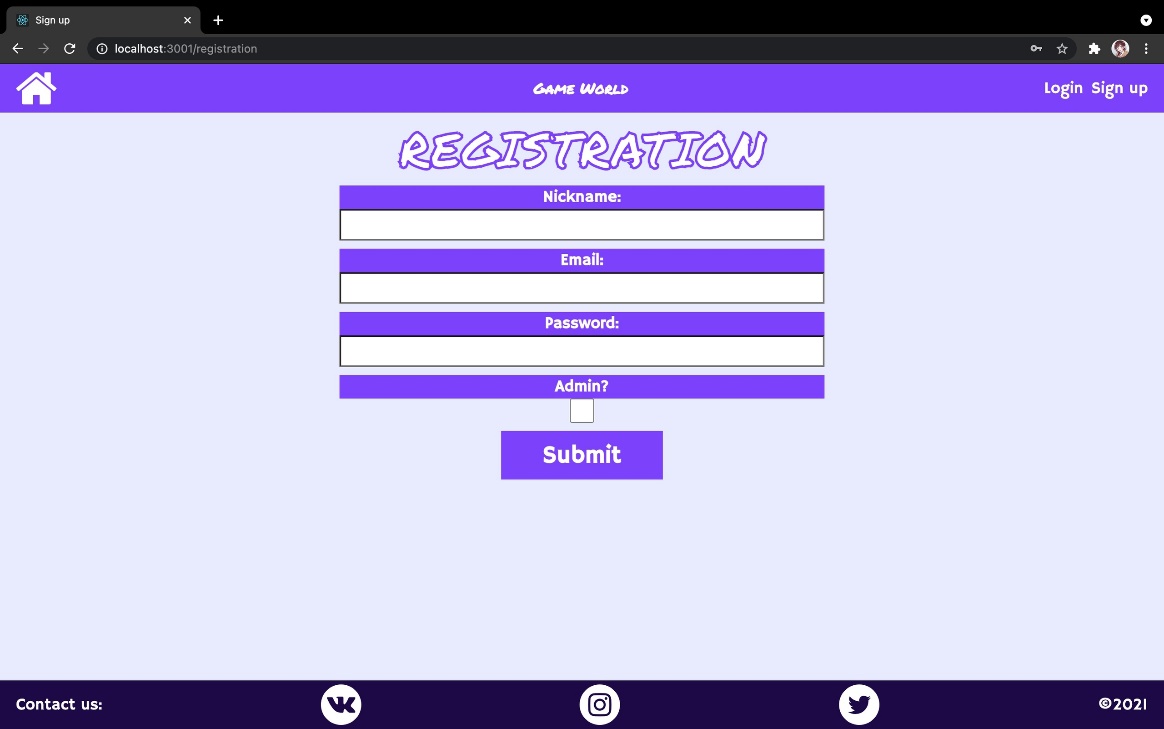


Рисунок 4.1 – Страница регистрации

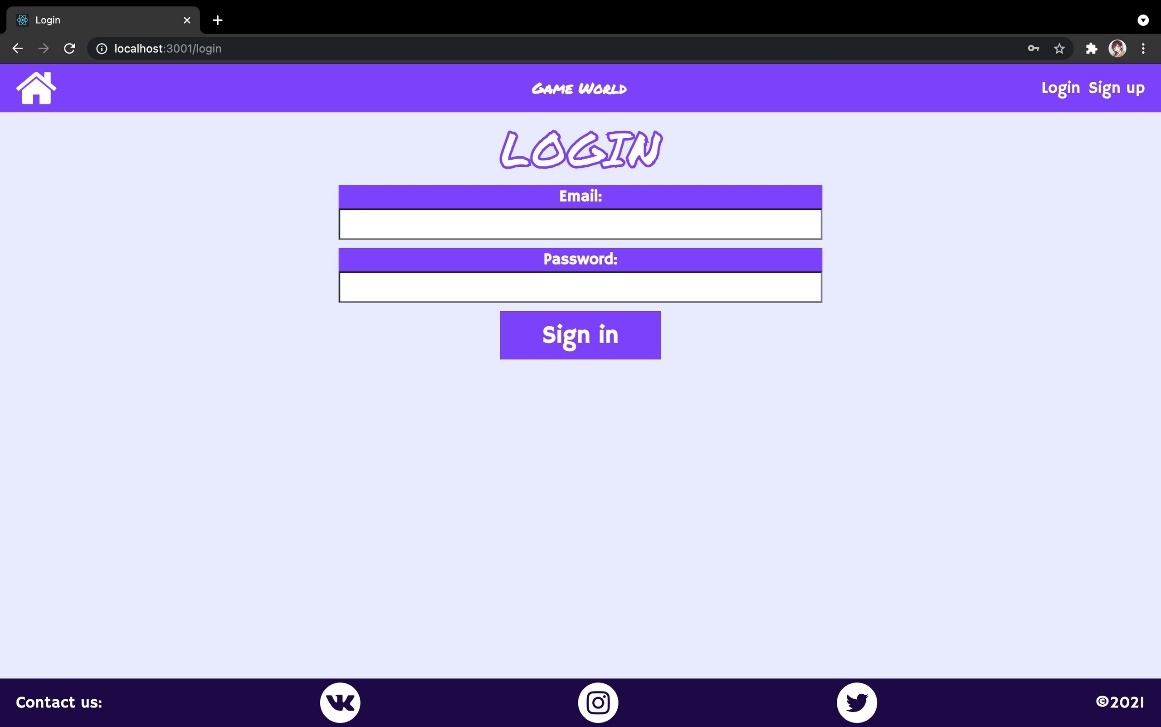


Рисунок 4.2 – Страница авторизации

После успешной регистрации или логина пользователь попадает на главную страницу. У авторизованного пользователя есть возможность посещать следующие страницы:

1) Домашняя страница;

2) Страница игры;

3) Добавление игры;

4) Панель администратора (доступен только администраторам);

На домашней странице пользователь может просмотреть список доступных игр, произвести поиск по фильтрам или перейти на страницу добавления своей игры. На странице игры пользователь видит всю доступную информацию, может выставить рейтинг и оставлять комментарии. На странице добавления игры можно создать игру, заполняя все необходимые поля.

Ниже представлены страницы добавления игры и самой игры:

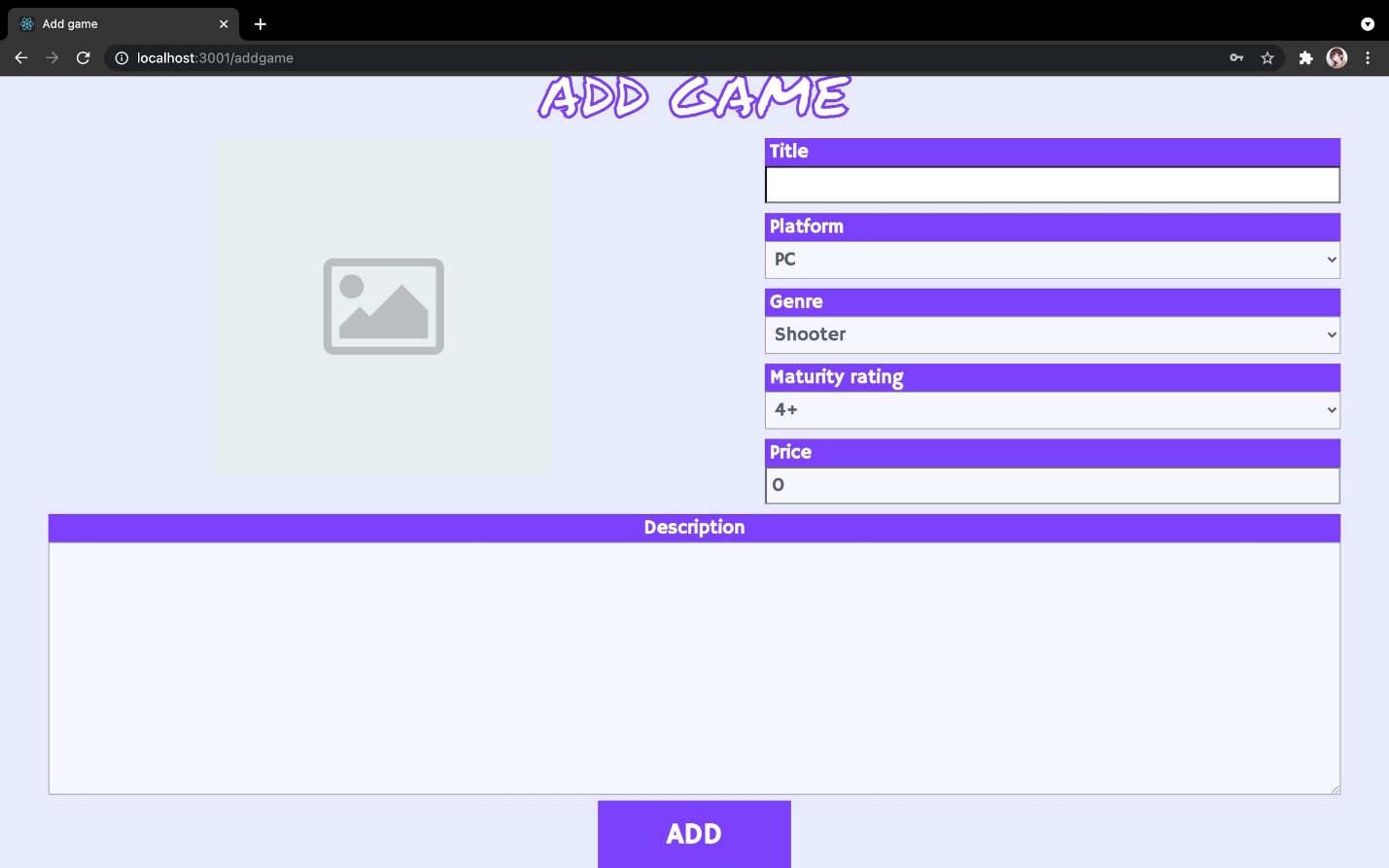


Рисунок 4.3 – Страница добавления игры

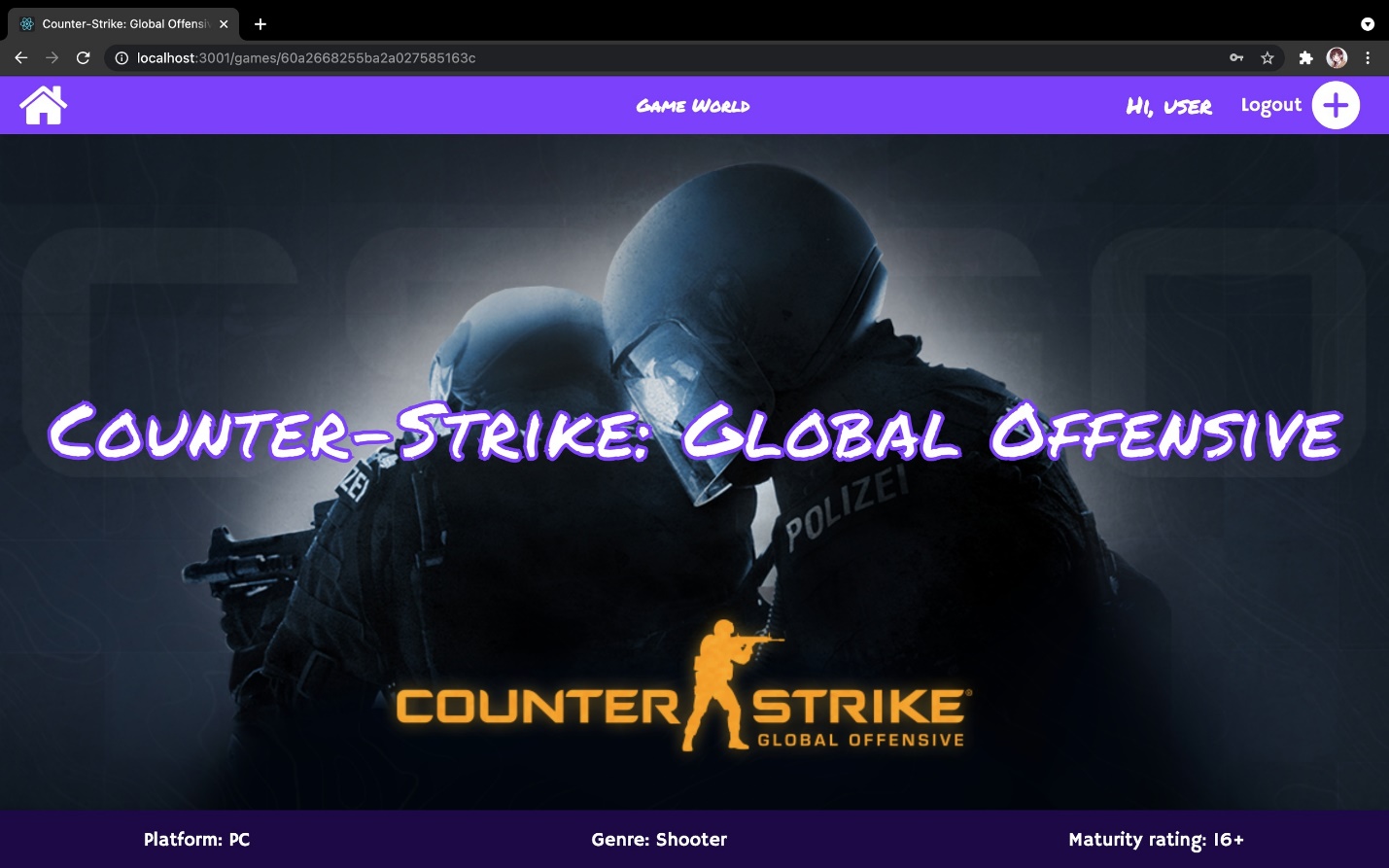


Рисунок 4.4 – Страница игры

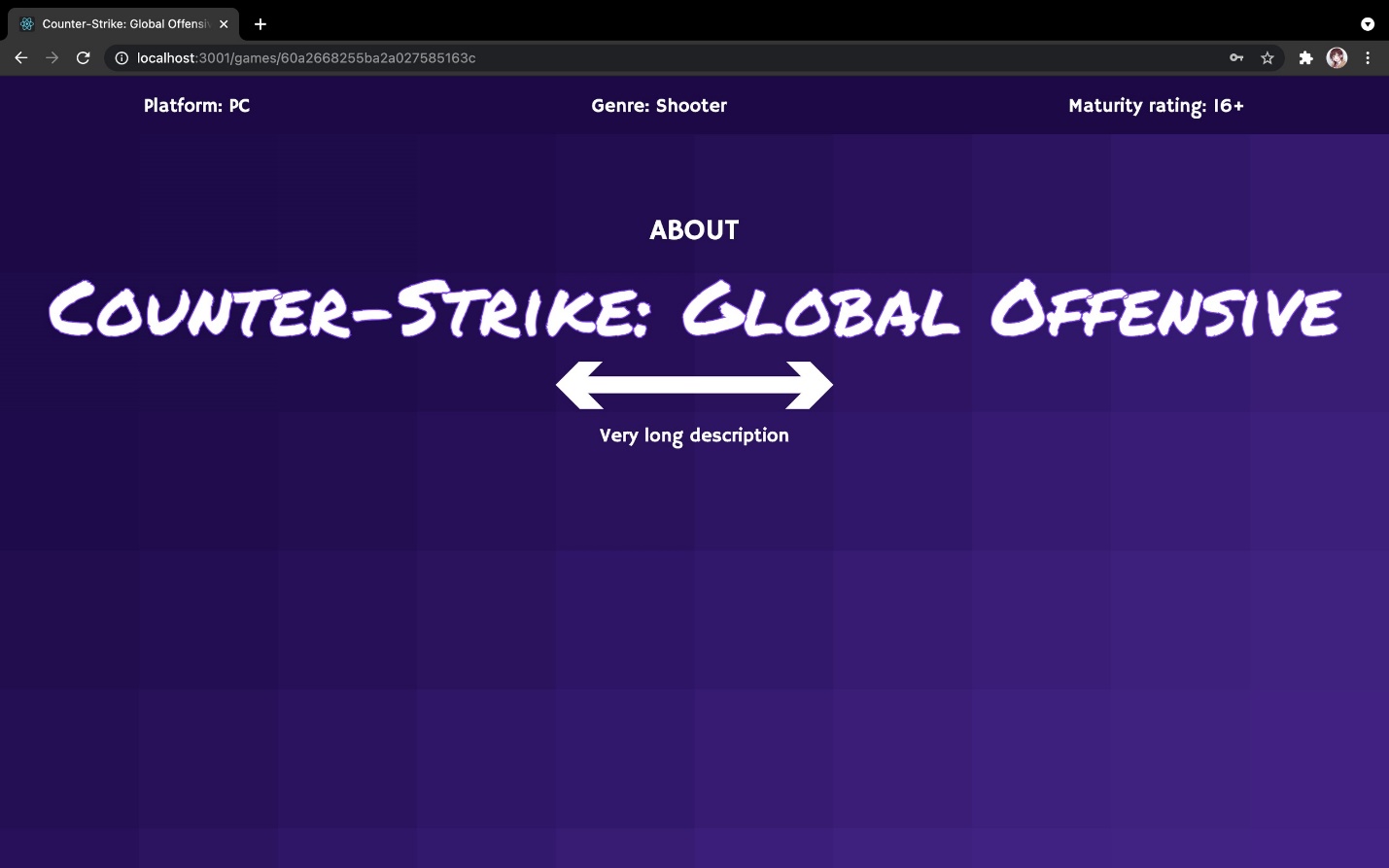


Рисунок 4.5 – Страница игры

На панели администратора присутствует таблица с играми или пользователями, в зависимости от нажатой соответствующей кнопки. В этой таблице содержатся краткие сведения об игре или зарегистрированном пользователе, а также присутствует возможность их удаления. При этом из базы данных будут удалены все связанные сущности.

На рисунке 4.6 представлена панель администратора:

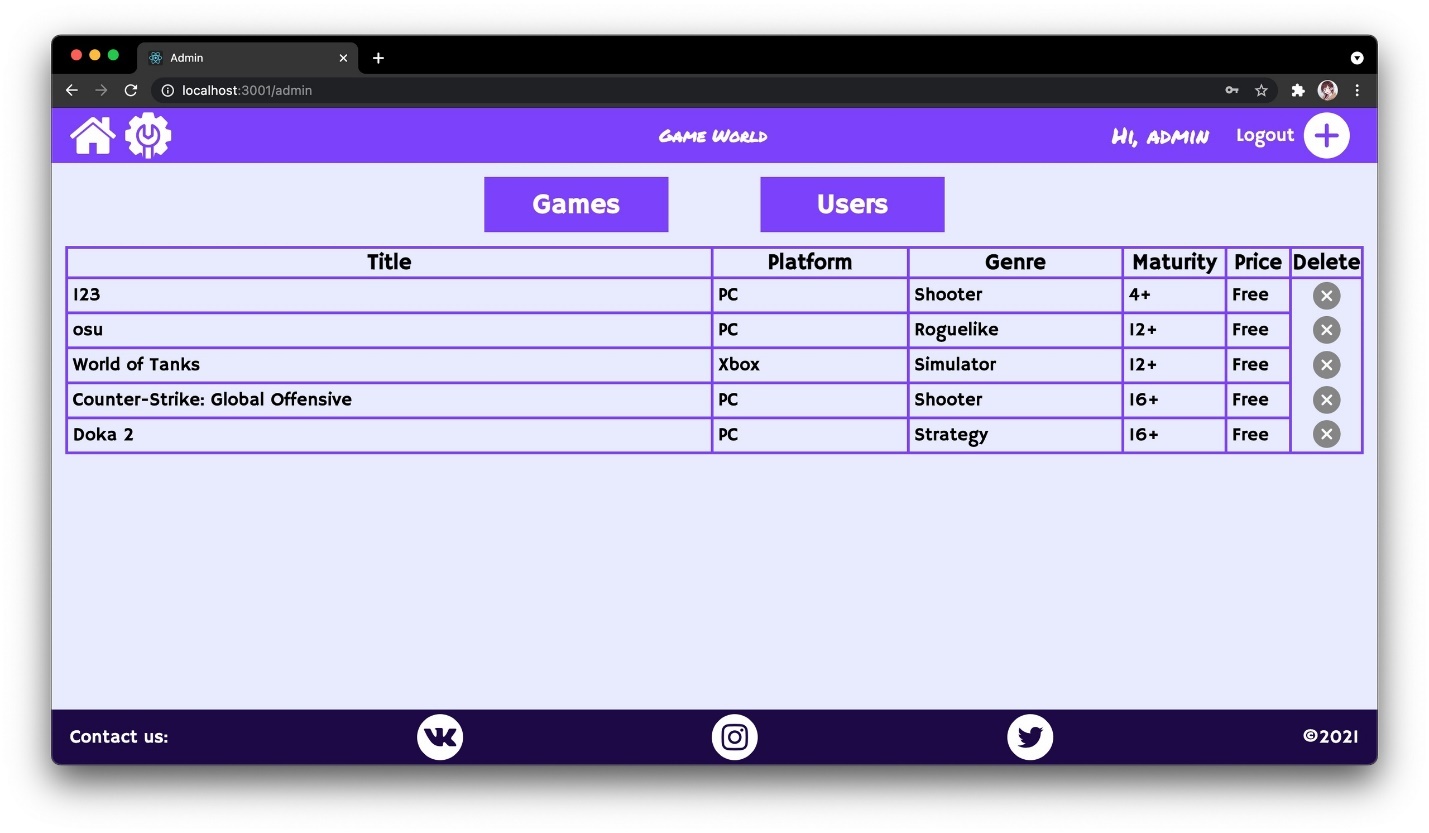


Рисунок 4.6 – Панель администратора

ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта было разработано программное средство «Game World», которое обеспечит возможность создавать, оценивать и комментировать игры. Согласно поставленным задачам, в данном приложении были реализованы следующие функции:

* регистрация;
* авторизация;
* выход из аккаунта;
* создание игры;
* выставление рейтинга, написание комментария к игре;
* поиск созданных игр с применением фильтров;
* возможность удаления пользователей;
* возможность удаления комментариев;
* возможность удаления игр.

Для успешного выполнения всех поставленных целей потребовалось изучить возможности библиотеки React JS, языка Node.js и Express, работу с веб-сокетами посредством библиотеки socket.io, устройство GraphQL. Также потребовалось применить знания возможностей функционального программирования языка JavaScript.

Существует много возможностей для дальнейшего улучшения приложения. Одним из них является добавление личного кабинета пользователя, добавление возможности смены языка, добавление изменяемого цветового оформления приложения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003;
2. Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. – СПб, 2003;
3. MongoDB Documentation – Электронные данные. – Режим доступа: https://docs.mongodb.com/;
4. API Reference – Express 4.x – Электронные данные. – Режим доступа: http://expressjs.com/en/api.html;
5. React Docs– Электронные данные. – режим доступа: https://reactjs.org/;

ПРИЛОЖЕНИЕ А

**Исходный код программы**

const express = require("express");

const mongoose = require("mongoose");

const formData = require("express-form-data");

const fs = require("fs");

const path = require("path");

const cors = require("cors");

const bcrypt = require("bcrypt");

const auth = require("./middlewares/auth");

const jwt = require("jsonwebtoken");

const http = require("http");

const { Server } = require("socket.io");

const { ApolloServer, gql } = require("apollo-server-express");

mongoose

.connect("mongodb://localhost/test", {

useNewUrlParser: true,

useUnifiedTopology: true,

})

.then(() => console.log("Connected to mongodb."))

.catch(() => console.log("Error connecting to mongodb."));

const gameSchema = new mongoose.Schema({

title: String,

imageUrl: String,

platform: String,

genre: String,

maturity: Number,

price: Number,

desc: String,

userId: String,

});

const Game = mongoose.model("Game", gameSchema, "games");

const userSchema = new mongoose.Schema({

nickname: String,

email: String,

password: String,

isAdmin: Boolean,

});

const User = mongoose.model("User", userSchema, "users");

const commentSchema = new mongoose.Schema({

gameId: String,

userId: String,

nickname: String,

text: String,

});

const Comment = mongoose.model("Comment", commentSchema, "comments");

const ratingSchema = new mongoose.Schema({

gameId: String,

userId: String,

value: Number,

});

const Rating = mongoose.model("Rating", ratingSchema, "ratings");

const app = express();

app.use(express.static("public"));

app.use(cors());

app.use(formData.parse());

app.use(formData.format());

app.use(formData.stream());

app.use(formData.union());

const server = http.createServer(app);

const io = new Server(server, {

cors: {

origin: "http://localhost:3001",

methods: ["GET", "POST", "DELETE"],

},

});

function generateToken(user) {

return jwt.sign({ \_id: user.\_id }, "aghkshdjfdgfklyeru42fdg");

}

app.post("/registration", async function (req, res) {

const { nickname, email, password, isAdmin } = req.body;

const salt = await bcrypt.genSalt();

const passwordHash = await bcrypt.hash(password, salt);

const user = new User({

nickname,

email,

password: passwordHash,

isAdmin,

});

user.save().catch(() => res.status(500).send());

const token = generateToken(user);

res

.header("Access-Control-Expose-Headers", "x-auth-token")

.header("x-auth-token", token)

.json(user);

});

app.post("/login", async function (req, res) {

const { email, password } = req.body;

const user = await User.findOne({ email });

if (!user) {

res.status(400).send();

return;

}

const validPassword = await bcrypt.compare(password, user.password);

if (!validPassword) {

res.status(400).send();

return;

}

const token = generateToken(user);

res

.header("Access-Control-Expose-Headers", "x-auth-token")

.header("x-auth-token", token)

.json(user);

});

app.post("/games", auth, function (req, res) {

const { title, platform, genre, maturity, price, desc, image, userId } =

req.body;

let imageName = "";

if (image instanceof fs.ReadStream) {

imageName = mongoose.Types.ObjectId().toHexString() + ".jpg";

const imagePath = "./public/images/games/" + imageName;

fs.writeFileSync(

path.join(process.cwd(), imagePath),

fs.readFileSync(image.path)

);

} else {

imageName = "default-img.png";

}

const game = new Game({

title,

platform,

genre,

maturity,

price,

desc,

imageUrl: "http://localhost:3000/images/games/" + imageName,

userId,

});

game

.save()

.then(() => res.send(game))

.catch(() => res.status(500).send());

});

app.get("/games", function (req, res) {

if (req.query.platform || req.query.genre || req.query.maturity) {

platform = req.query.platform;

genre = req.query.genre;

maturity = req.query.maturity;

Game.find({ platform, genre, maturity })

.lean()

.then((games) => {

res.send(games);

})

.catch(() => res.status(404).send());

} else {

Game.find()

.lean()

.then((games) => {

res.send(games);

})

.catch(() => res.status(503).send());

}

});

app.get("/games/:gameId", function (req, res) {

const gameId = req.params.gameId;

Game.findById(gameId)

.then((game) =>

res.send({

title: game.title,

imageUrl: game.imageUrl,

platform: game.platform,

genre: game.genre,

maturity: game.maturity,

price: game.price,

desc: game.desc,

userId: game.userId,

})

)

.catch(() => res.status(404).send());

});

app.delete("/games/:gameId", auth, function (req, res) {

const gameId = req.params.gameId;

Game.findByIdAndDelete(gameId)

.then((game) => res.send(game))

.catch(() => res.status(404).send());

Rating.find()

.then((ratings) =>

ratings.map((rate) => {

if (rate.gameId === gameId) {

Rating.findByIdAndDelete(rate.\_id).catch(() =>

res.status(404).send()

);

}

})

)

.catch(() => res.status(404).send());

Comment.find()

.then((comments) =>

comments.map((comment) => {

if (comment.gameId === gameId) {

Comment.findByIdAndDelete(comment.\_id).catch(() =>

res.status(404).send()

);

}

})

)

.catch(() => res.status(404).send());

});

io.on("connection", (socket) => {

console.log("user connected");

socket.on("disconnect", () => {

console.log("disconnect");

});

socket.on("message", (msg) => {

switch (msg.type) {

case "comments":

Comment.find({ gameId: msg.gameId }).then((comments) =>

socket.send({ type: "comments", comments })

);

break;

case "add-comment":

const comment = new Comment({

gameId: msg.gameId,

userId: msg.userId,

nickname: msg.nickname,

text: msg.text,

});

comment

.save()

.then(() => {

Comment.find({ gameId: msg.gameId }).then((comments) =>

socket.send({ type: "comments", comments })

);

})

.catch((e) => console.log(e));

break;

case "delete-comment":

Comment.findByIdAndDelete(msg.commentId)

.then(() =>

Comment.find({ gameId: msg.gameId })

.then((comments) => socket.send({ type: "comments", comments }))

.catch((e) => console.log(e))

)

.catch((e) => console.log(e));

break;

case "rating":

Rating.find({ gameId: msg.gameId }).then((ratings) => {

let sum = 0,

i = 0;

for (i; i < ratings.length; i++) {

sum += ratings[i].value;

}

const result = sum / i;

socket.send({ type: "rating", value: result });

});

break;

case "add-rating":

Rating.findOne({ gameId: msg.gameId, userId: msg.userId })

.then((ratingFound) => {

if (!ratingFound) {

const rating = new Rating({

gameId: msg.gameId,

userId: msg.userId,

value: msg.value,

});

rating

.save()

.then(() => {

Rating.find({ gameId: msg.gameId }).then((ratings) => {

let sum = 0,

i = 0;

for (i; i < ratings.length; i++) {

sum += ratings[i].value;

}

const result = sum / i;

socket.send({ type: "rating", value: msg.value });

});

})

.catch((e) => console.log(e));

} else {

Rating.findOneAndUpdate(

{ gameId: msg.gameId, userId: msg.userId },

{ value: msg.value }

)

.then(() => {

Rating.find({ gameId: msg.gameId }).then((ratings) => {

let sum = 0,

i = 0;

for (i; i < ratings.length; i++) {

sum += ratings[i].value;

}

const result = sum / i;

socket.send({ type: "rating", value: msg.value });

});

})

.catch((e) => console.log(e));

}

})

.catch((e) => console.log(e));

break;

default:

break;

}

});

});

const typeDefs = gql`

type Query {

games: [Game!]!

users: [User!]!

}

type Mutation {

deleteGame(gameId: ID!): Game!

deleteUser(userId: ID!): User!

}

type Game {

id: ID!

title: String!

platform: String!

genre: String!

maturity: Int!

price: Int!

desc: String

}

type User {

id: ID!

nickname: String!

email: String!

password: String!

isAdmin: Boolean!

}

`;

const resolvers = {

Query: {

games: () => Game.find(),

users: () => User.find(),

},

Mutation: {

deleteGame: async (root, { gameId }) =>

await Game.findByIdAndDelete(gameId).then((game) => {

Rating.find()

.then((ratings) =>

ratings.map((rate) => {

if (rate.gameId === gameId) {

Rating.findByIdAndDelete(rate.\_id).catch((error) =>

console.log(error)

);

}

})

)

.catch((error) => console.log(error));

Comment.find()

.then((comments) =>

comments.map((comment) => {

if (comment.gameId === gameId) {

Comment.findByIdAndDelete(comment.\_id).catch((error) =>

console.log(error)

);

}

})

)

.catch((error) => console.log(error));

return game;

}),

deleteUser: async (root, { userId }) =>

await User.findByIdAndDelete(userId).then((user) => {

Rating.find()

.then((ratings) =>

ratings.map((rate) => {

if (rate.userId === userId) {

Rating.findByIdAndDelete(rate.\_id).catch((error) =>

console.log(error)

);

}

})

)

.catch((error) => console.log(error));

Comment.find()

.then((comments) =>

comments.map((comment) => {

if (comment.userId === userId) {

Comment.findByIdAndDelete(comment.\_id).catch(() =>

console.log(error)

);

}

})

)

.catch(() => console.log(error));

return user;

}),

},

};

const serverGql = new ApolloServer({ typeDefs, resolvers });

serverGql.applyMiddleware({ app });

server.listen(3000, () => console.log("Server is listening on port 3000.."));