ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

КУРСОВОЙ ПРОЕКТ

Дисциплина: Технологии кроссплатформенного программирования

Тема: Разработка компьютерной игры "морской бой" с использованием WEB технологий

Выполнил: студент группы __202-721_____

_____Паращак Е.В._____
(Фамилия И.О.)

Дата, подпись _____ _____
(Дата) (Подпись)

Проверил: _____
(Фамилия И.О., степень, звание)

Дата, подпись _____ _____
(Дата) (Подпись)

Замечания: _____

_____

_____

Москва

2022

АННОТАЦИЯ

Курсовой проект. Московский политехнический университет. Тема работы: «Разработка компьютерной игры "морской бой" с использованием WEB технологий».

Выполнил — Паращак Е.В.

Руководитель работы — Худайбердиева Г.Б.

Направление подготовки — 09.03.02 «Информационные системы и технологии».

Курсовой проект состоит из пояснительной записки, графической части и разработанной игры «Морской бой». Пояснительная записка состоит из 52 стр. текста, 5 приложений. Включает в себя разделы — введение, описание предметной области и проектирование и реализация игры, заключение, список использованных источников, приложения. В курсовой работе описан процесс проектирования и разработки игры на базе применения кроссплатформенных web - технологий с подключением базы данных. В ходе выполнения курсового проекта были написаны, проверены и оттестированы модули приложения и работа с серверной частью посредством использования технологий Node. Js и подключение к mongodb, позволяющих взаимодействовать с пользователем через регистрацию, авторизацию, создание, сопровождение (внесение изменений и получение различных видов статистической информации) и пользование данными, содержащимися в базе данных приложения. Ключевые слова: node js, html, css, веб-разработка, игра, образование, проверка знаний, mongodb.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Информационные технологии играют важную роль в нашей жизни. С каждым годом они становятся все более сложными и разнообразными. Одной из самых популярных информационных технологий является компьютерная игра “Морской бой”.

Игра “Морской бой” является одной из самых популярных настольных игр в мире. Она была создана в 1931 году и с тех пор завоевала сердца миллионов игроков.

Цель игры - уничтожить корабли противника, стреляя по ним снарядами. Игроки начинают игру с определенным количеством кораблей, которые расположены на поле боя. Корабли имеют разные размеры.

Игровое поле состоит из клеток, на которых расположены корабли. Игрок должен стрелять по кораблю противника, указывая координаты клетки, в которую он хочет попасть. Если снаряд попадает в корабль, он считается уничтоженным.

Игра продолжается до тех пор, пока один из игроков не уничтожит все корабли противника. Победителем становится игрок, который первым уничтожит все корабли соперника.

“Морской бой” - это увлекательная игра, которая развивает стратегическое мышление и умение принимать решения в условиях ограниченного времени. Она также может быть использована для обучения детей основам стратегии и тактики.

1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И НАЗНАЧЕНИЕ ПРОЕКТИРУЕМОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

В данной главе приведено рассмотрение игры с теоретической точки зрения. Описаны её назначение, основные функциональные возможности и структура.

1.1. Функциональные требования

Игровой процесс:

- Игра реализуется на игровом поле, представляющем собой сетку, где каждая ячейка может быть состоянием корабля, пустой или атакованной.
- Корабли размещаются автоматически случайным образом.
- Игрок атакует поле компьютера.

Авторизация и рейтинг:

- Игра предоставляет возможность регистрации и авторизации пользователя.
- Каждый игрок имеет рейтинг, который обновляется в зависимости от результатов игр.

Режимы сложности:

- Реализовать разные уровни сложности для игры: легкий, средний, сложный.
- Сложность влияет на логику игры.

Вывод рейтинга:

- Система рейтинга обновляется после каждой игры и выводится в общий рейтинг.
- Предоставить страницу с топ-игроками.

1.2. Технические требования

Фронтенд:

- Использование HTML, CSS, JavaScript.
- Веб-интерфейс должен быть интуитивно понятным и отзывчивым.

- Использование современных фреймворков и библиотек по желанию.

Бекенд:

- Использование Node.js и Express.js для создания серверной части.
- Интеграция с базой данных (MongoDB) для хранения данных пользователей и рейтинга.
- Реализация системы авторизации и рейтинга.

1.3. Сценарий игры "Морской бой"

1. Начало игры:

- Игрок открывает веб-приложение "Морской бой".
- Если у игрока уже есть аккаунт, он входит в систему, в противном случае проходит регистрацию.

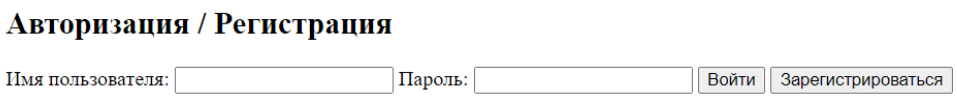


Рисунок 1 - Форма авторизации

- После входа в систему игроку предоставляется возможность выбора: выйти из аккаунта или выбрать сложность и начать игру.

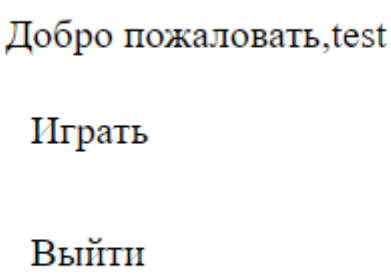


Рисунок 2 - Главное меню игры

- Далее игроку предоставляется возможность выбора уровня сложности (легкий, средний, сложный).

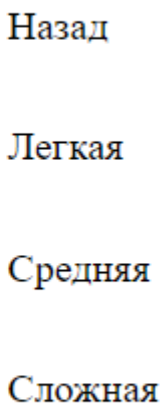


Рисунок 3 - Выбор сложности

2. Размещение кораблей:

- Корабли автоматически расставляются случайным образом на игровом поле перед началом матча.
- Правила размещения кораблей: корабли не могут пересекаться и выходить за границы поля.

Рисунок 4 - Игровая форма с ячейками кораблей

3. Ход игры:

- Игра начинается с того, что игрок делает ход, выбирая ячейку на поле противника для атаки.
- После атаки игровой движок определяет результат хода и отображает его на экране.
- Цель игрока - потопить все корабли противника за минимальное количество ходов.
- Игрок имеет ограниченное количество попыток (зависит от уровня сложности).
- Если игрок попадает по кораблю, количество попыток не уменьшается, но увеличивается счетчик попаданий.
- Если игрок не попадает, количество попыток уменьшается.

4. Завершение игры:

- Когда игрок использует все попытки, игра завершается.
- Рейтинг игрока обновляется в зависимости от количества ходов и сложности.
- Игроку предоставляется возможность начать новую игру или вернуться в главное меню.

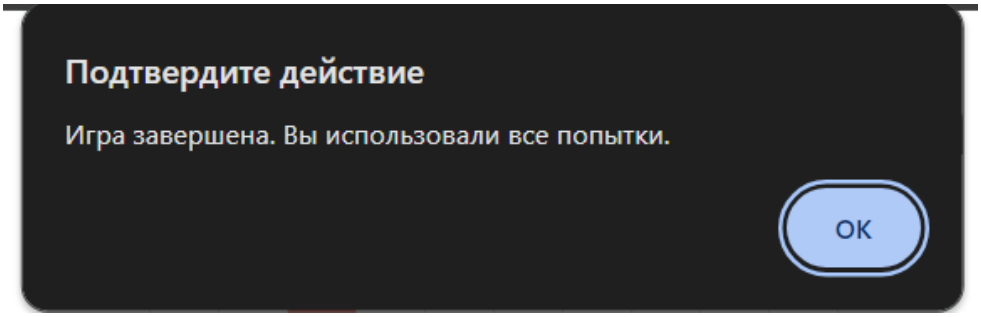


Рисунок 5 - Окно о завершении игры

5. Просмотр рейтинга:

- В главном меню игрок может просмотреть общий рейтинг.
- Рейтинг отображает лучших игроков с указанием их имен и текущих рейтингов.

Рисунок 6 - Рейтинг игроков

6. Выход из аккаунта:

- Игрок может выйти из аккаунта, закрыв веб-приложение или используя соответствующую кнопку в интерфейсе.

1.4. Функциональные модули

Модуль Авторизации и Регистрации:

- Функции: регистрация нового пользователя, аутентификация существующего пользователя.
- Связи: связан с базой данных для хранения информации о пользователях.

Модуль Размещения Кораблей:

- Функции: автоматическое размещение кораблей на поле перед началом игры.
- Связи: взаимодействует с игровым движком.

Модуль Игрового Движка:

- Функции: управление игровым процессом, определение результатов ходов.
- Связи: взаимодействует с модулем размещения кораблей, модулем атаки и базой данных.

Модуль Атаки:

- Функции: обработка атаки игрока, определение попадания и потопления кораблей.
- Связи: взаимодействует с игровым движком.

Модуль Рейтинга:

- Функции: обновление рейтинга игрока после завершения игры, предоставление рейтинговых данных.
- Связи: взаимодействует с базой данных.

Модуль Общего Рейтинга:

- Функции: предоставление информации о лучших игроках.
- Связи: взаимодействует с базой данных.

Модуль Вывода Интерфейса:

- Функции: отображение игрового интерфейса, взаимодействие с пользователем.
- Связи: взаимодействует со всеми остальными модулями для передачи данных и управления игровым процессом.

Модуль Выхода:

- Функции: завершение текущей игры, выход из аккаунта.
- Связи: взаимодействует с интерфейсом и игровым движком.

В Приложении А можно ознакомиться с графом переходов.

1.5. Описание алгоритмов на клиенте

В данной веб-игре предусмотрено много функций для реализации алгоритмов логики игры на клиентской части и небольшая часть обработки данных на сервере. Рассмотрим функционал на клиенте игры, был создан скрипт game.js, в котором и размещена основная логика игры.

Первым делом в ней расположены функции *login()* и *register()*, которые отвечают за авторизацию и регистрацию соответственно, при авторизации отправляются логин и пароль на сервер в виде JSON, а сервер возвращает ответ в виде объекта с полем «success», если оно true то мы скрываем форму авторизации, проставляем имя в функции *updatePlayerName(username)* и переключаем в режим отображения форму главного меню в функции *toggleMenu()*.

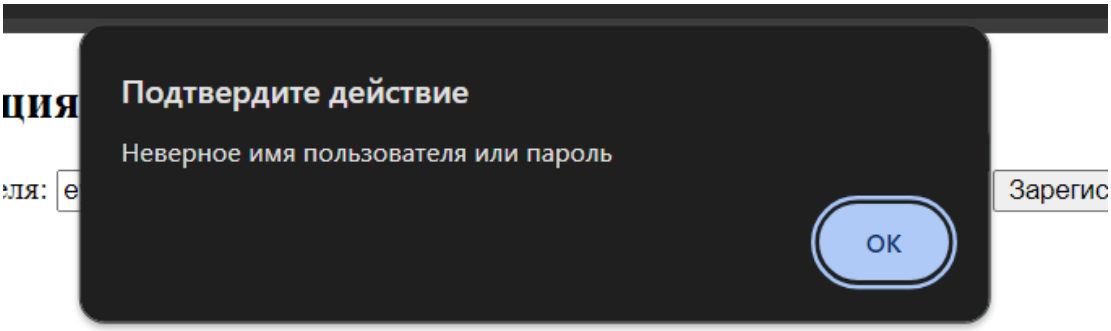


Рисунок 7 - Проверка при авторизации

При регистрации мы отправляем запрос на регистрацию на сервере в файле server.js есть метод register в котором по полученным данным с клиента, мы проверяем проверяем на пустоту поля ввода данных, далее есть ли уже пользователь с таким именем, если есть, то выкидаем на клиент ошибку, иначе

создаем новый объект в БД с название Player и полями username, password, score, где username равен пришедшему с клиента имени, также и с password, а score проставляем значение «0».

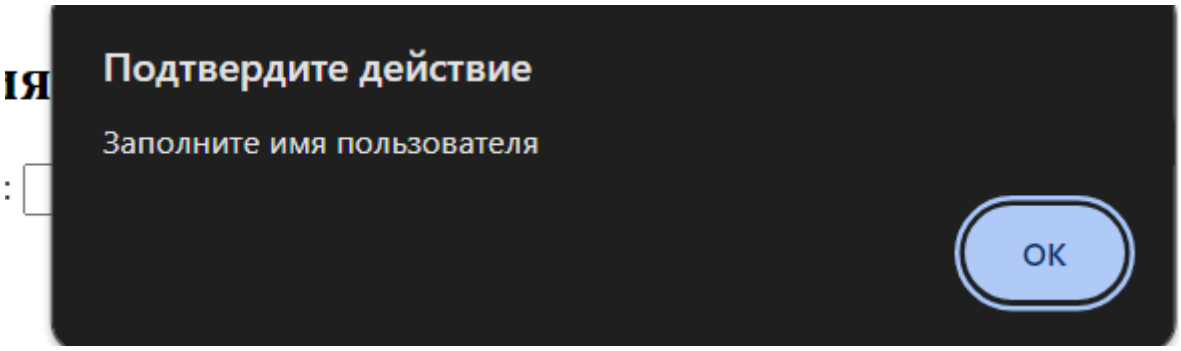


Рисунок 8 - Проверка на пустое имя

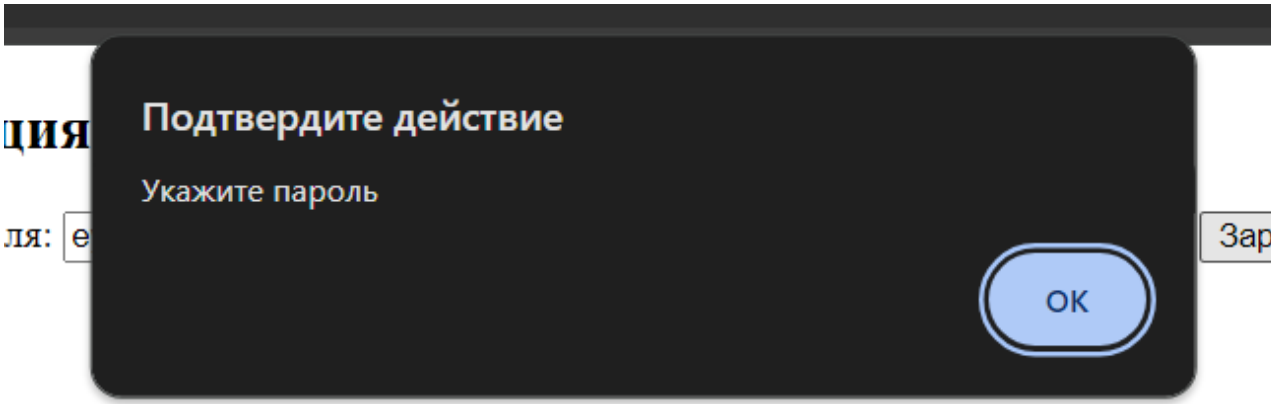


Рисунок 9 - Проверка на пустой пароль

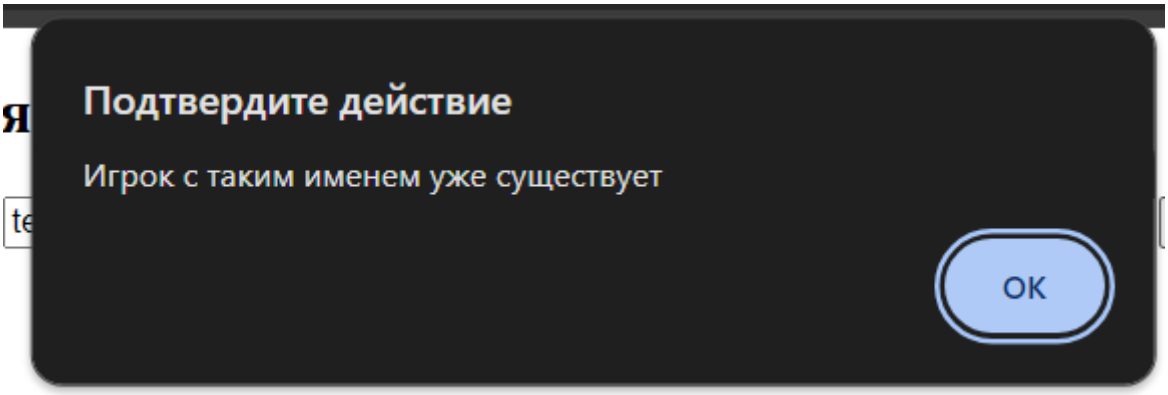


Рисунок 10 - Проверка на существование пользователя в БД

Далее начинается основная логика клиентской части, к текущему корневому документу DOM дерева сайта добавляем событие, чтобы при загрузке страницы выполнялся наш код, в нем мы задаем начальные значение некоторых полей по-умолчанию, таких как:

- Размер игрового поля, «boardSize» по умолчанию ставим 10

- Максимальное кол-во попыток maxAttempts, ставим в 60

Находим элемент списка рейтинга 'player-rating-list' и присваиваем его в переменную ratingList

Далее обновляем ratingList данными с сервера при помощи функции *updateRatingList()* в нем мы отправляем запрос на сервер в метод *rating* в нем мы ищем в БД пользователей и возвращаем их. На клиенте получаем список игроков в виде объектов «Palyer» и обрабатываем их, сначала сортируем по убыванию по полю score. И для каждого объекта создаем HTML элемент ‘li’ с таким видом: «индекс». «имя»: «счет».

Находим элемент 'game-content' который является основным HTML контейнером игры, в котором находятся игровые элементы: игровое поле и т.д.

Создаем список кораблей в виде массива объектов с полями «size» и массивом «positions», где size – размер корабля, а «positions» - координаты корабля на поле.

Инициализируем элементы главного меню в *initMenu()* и привязываем события «click» чтобы при нажатии на «Играть» выполнилась функция *showDifficultyMenu(),* а при выходе - *exitGame()*

В *showDifficultyMenu(),* скрываем главное меню, удаляем элементы старого игрового поля и счета оставшихся попыток и вызываем метод *createDifficultyMenu()*

В *createDifficultyMenu()* мы создаем элементы списка сложностей и при клике вызываем методы *setDifficulty(сложность)* и *startGame()*

В *setDifficulty(сложность)* мы проставляем значение для поля maxAttempts в зависимости от сложности

В *startGame()* происходит инициализация всех основных элементов: игрового поля, текста об оставшихся попытках и счёт, также там происходит расстановка кораблей на поле.

Поле инициализируется в методе *initializeBoard()* где в зависимости от переменной boardSize создается матрица board с индексами строчек и столбцов.

В *renderBoard()* именно создается HTML элемент игрового поля, т.е. элемента table, тоже размером в boardSize

Далее происходит размещение кораблей в *placeShips()* где для каждого объекта ships задается значение в поле positions случайным образом, где сначала задается случайным образом начальные координаты и случайное направление, потом в зависимости от направления и размера корабля, в positions добавляются координаты оставшихся точек корабля.

Ход самой игры происходит по клику на ячейку таблицы в *handleCellClick()* где происходи проверка на превышение попыток, проверка на попадание и при этом тогда проверяется победил ли игрок в методе *checkVictory()*, где проверяется попадание по всем позициям каждого корабля и если мы попали по всем, то счет умножается на модификатор сложности.

Когда заканчиваются попытки, то мы отображаем оставшиеся корабли в методе *markRemainingShipParts()*, где ячейке таблицы с координатам корабля нет css класса 'hit' проставляется css класс 'remaining-part' для наглядности.

При окончании игры всегда вызывается метод *updateScore(username, newScore)*где происходи запрос на сервер в метод *update-score*, где происходи проверка на есть ли такой игрок в БД, и если новый счет больше старого, то обновляем поле score в БД у этого пользователя.

Весь листинг скрипта клиентской части игры будет в Приложении Б, HTML разметки в Приложении В а весь код серверной части в Приложении Г. Также алгоритм работы игры в Приложении Д.

2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ИГРЫ

Создание игры начинается с проектирования, что является первым и самым важным шагом. Начинать разработку с этого этапа является правильным подходом, который поможет сэкономить много денег и времени.

В процессе проектирования определяется суть игры, её увлекательность, удобство, возможность выиграть и внешний вид. То есть в самом начале мы тщательно продумываем всю игру. Затем, следуя предварительному сценарию, можно приступить к разработке дизайна, программной части и заполнению игры контентом и прочими элементами. Таким образом, проектирование игры является основой для последующей разработки.

2.1. Выбор технологий

Для проектирования игры "Морской бой" был использован язык программирования JavaScript вместе с HTML и CSS для создания пользовательского интерфейса. Проект также использует библиотеку Express.js для создания серверной части приложения на языке JavaScript с использованием Node.js. Для хранения данных об игроках и их рейтинге используется MongoDB в связке с библиотекой Mongoose для работы с базой данных.

Вот основные технологии и инструменты, использованные в проектировании игры:

Frontend (Клиентская часть):

- HTML, CSS для разметки и стилизации интерфейса.
- JavaScript для логики игры и взаимодействия с сервером.
- Fetch API для отправки запросов на сервер.

Backend (Серверная часть):

- Node.js: Среда выполнения JavaScript на сервере.
- Express.js: Фреймворк для создания веб-приложений на Node.js.
- MongoDB: Для хранения данных об игроках.
- Mongoose: ODM (Object Data Modeling) для работы с MongoDB из JavaScript.

Взаимодействие с сервером:

- HTTP протокол для обмена данными между клиентом и сервером.
- RESTful API для определения методов взаимодействия с данными на сервере.

Проектирование интерфейса:

- DOM (Document Object Model) для динамического обновления элементов интерфейса.
- Fetch API для асинхронных запросов к серверу без перезагрузки страницы.
- Обработка событий для реагирования на действия пользователя.

Теперь расскажем про некоторые из них поподробнее

Node.js - это среда выполнения JavaScript, построенная на движке V8, разработанном Google для браузера Chrome. Однако Node.js предоставляет среду выполнения для JavaScript на сервере, что позволяет разработчикам создавать серверные приложения на JavaScript, а не только фронтенд-код для веб-страниц.

В контексте игры "Морской бой", Node.js используется для создания серверной части приложения. Вот несколько ключевых моментов, как Node.js используется в данном проекте:

Express.js:

Express.js - это фреймворк для Node.js, который упрощает создание веб-приложений и API. В данной игре используется Express.js для обработки маршрутов, обработки HTTP-запросов и управления логикой приложения.

MongoDB и Mongoose:

Node.js взаимодействует с базой данных MongoDB при помощи библиотеки Mongoose. Mongoose обеспечивает объектно-документное отображение (ODM), что упрощает работу с MongoDB через Node.js. В игре используется MongoDB для хранения информации о пользователях, их счетах и других данных.

CORS (Cross-Origin Resource Sharing):

В игре установлен промежуточный обработчик CORS, предоставляемый библиотекой cors. Это позволяет обрабатывать запросы от фронтенда, выполняющегося на другом домене, что является распространенным сценарием при создании веб-приложений.

Работа с HTTP-запросами:

Node.js используется для обработки HTTP-запросов, отправленных клиентом (веб-браузером). В коде сервера обработчики маршрутов определены для регистрации новых игроков, аутентификации, обновления счетов и других действий.

Пример использования Node.js в коде сервера игры "Морской бой" вы можете видеть в предоставленном в Приложении В коде сервера. В этом коде Node.js работает вместе с Express.js для создания веб-сервера, обработки запросов, взаимодействия с MongoDB и обновления данных игры.

В игре "Морской бой", выше упомянутые MongoDB и Mongoose используются для хранения информации о пользователях, их счетах и других связанных данных в базе данных. Давайте рассмотрим их использование более подробно:

MongoDB

MongoDB - это документоориентированная NoSQL база данных. В отличие от традиционных реляционных баз данных, MongoDB хранит данные в формате BSON (бинарный JSON) в виде документов.

MongoDB является схема-менее базой данных, что означает, что каждый документ в коллекции может иметь различную структуру. Это гибкость особенно полезна в контексте, где структура данных может изменяться или где разные пользователи могут иметь разные поля в своих профилях.

Данные в MongoDB организованы в коллекции, которые могут сравниваться с таблицами в реляционных базах данных.

Mongoose

Mongoose - это библиотека ODM (Object Data Modeling) для MongoDB в среде Node.js. Она предоставляет возможность описывать структуру данных в

виде моделей, что упрощает работу с MongoDB в объектно-ориентированном стиле.

В игре создается модель Player, которая представляет собой схему данных для игроков. Она содержит поля, такие как username, password, и score.

Mongoose позволяет добавлять методы экземпляра и статические методы к моделям. Например, в коде сервера игры используются методы для регистрации новых игроков, поиска игроков по имени пользователя и обновления счета. Ниже представлен пример создания схемы игрока - «Player».

```
// Определение схемы и модели для игрока
const playerSchema = new mongoose.Schema({
  username: String,
  password: String,
  score: Number,
});

const Player = mongoose.model('Player', playerSchema);

app.use(express.json());
```

Рисунок 11 - Создание схемы игрока в БД

2.2. Выбор инструментальных средств

Sublime Text — это мощный текстовый редактор, широко используемый среди разработчиков программного обеспечения. В контексте написания кода для игры "Морской бой", Sublime Text предоставляет несколько преимуществ, сделав процесс более эффективным и удобным.

Sublime Text отличается минималистичным интерфейсом и высокой производительностью. Редактор предоставляет быстрый доступ к основным функциям, что делает процесс написания кода более эффективным.

Sublime Text обладает мощными инструментами для подсветки синтаксиса, что облегчает визуальное восприятие кода. Для игры "Морской бой", где используются различные языки программирования (HTML, JavaScript, CSS), это является ключевым аспектом.

Sublime Text поддерживает обширное сообщество разработчиков, которые создают различные плагины и дополнения для расширения функциональности редактора. Это может включать в себя плагины для

форматирования кода, автодополнения, проверки ошибок и другие полезные инструменты.

Редактор обладает удобной системой вкладок и возможностью работать с несколькими файлами одновременно. Это полезно при создании игр, где обычно используется несколько файлов для различных частей проекта.

Рисунок 12 - Окно Sublime Text

Sublime Text, будучи легким, быстрым и гибким инструментом, отлично подходит для написания кода игры "Морской бой". Его функциональность и настраиваемость делают его предпочтительным выбором для многих разработчиков, особенно тех, кто ценит производительность и комфорт при написании кода.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была достигнута цель работы – разработана игра на базе применения кроссплатформенных web-технологий с подключенной базой данных. Результатом выполнения курсовой работы стало веб-приложение - игра «Морской бой».

Для достижения цели был решен ряд задач:

1. Описана предметная область;
2. Спроектирована схема работы игры;
3. Разработан дизайн и сама игра.

Проект "Морской бой" представляет собой интересное и увлекательное веб-приложение, реализованное с использованием современных технологий и подходов. В ходе разработки игры были применены различные технологии, включая веб-технологии на стороне клиента, Node.js для серверной части, MongoDB для хранения данных и Mongoose для удобного взаимодействия с базой данных.

Основными компонентами проекта стали веб-интерфейс для игрока, позволяющий регистрироваться, авторизовываться и играть в "Морской бой", а также серверная часть, ответственная за обработку запросов, взаимодействие с базой данных и поддержание игровой логики.

Одним из ключевых элементов проекта стало использование технологии Fetch API для взаимодействия между клиентом и сервером. Fetch API обеспечил асинхронную передачу данных, что позволило реализовать регистрацию, авторизацию, а также обновление счета игрока без перезагрузки страницы.

Технологии Node.js, MongoDB и Mongoose обеспечили эффективное взаимодействие с базой данных, предоставляя гибкость и удобство при работе с данными. MongoDB, как NoSQL база данных, демонстрирует преимущества гибкости в структуре данных, что важно в контексте проекта с изменяющейся структурой игровых данных.

Опыт разработки этого проекта позволил ознакомиться с различными аспектами создания веб-приложений, включая фронтенд и бэкенд аспекты, взаимодействие с базой данных, асинхронное программирование и обработку HTTP-запросов.

Проект "Морской бой" не только предоставляет пользователю интересное времяпрепровождение, но и служит примером использования современных технологий для разработки веб-приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальная документация по MongoDB [Электронный ресурс] // сайт. – Режим доступа: https://www.mongodb.com/

2. Руководства и статьи по использованию MongoDB [Электронный ресурс] // сайт. – Режим доступа: https://www.codecademy.com/learn/learn-mongodb

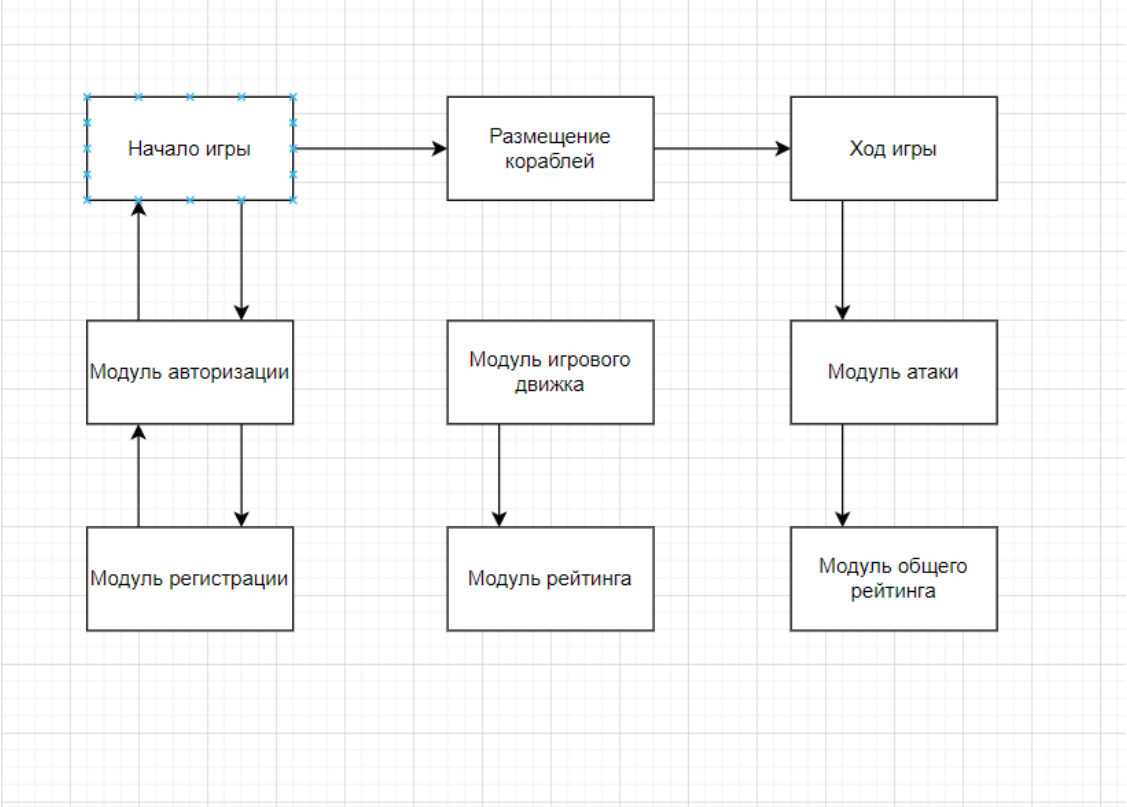
3. Руководства и статьи по HTML и не только [Электронный ресурс] // сайт. – Режим доступа: https://htmldog.com/

4. Кайл Симпсон «Вы пока еще не знаете JS» с англ. - СПб.:, 2022. – 192 стр.

5. Кантелон, Янг, Мек «Node.js в действии» с англ. - СПб.:, 2018. – 432 стр.

6. Требования к выполнению курсовой работы.

ПРИЛОЖЕНИЕ А



ПРИЛОЖЕНИЕ Б

```
// Функция для отправки запроса на сервер при авторизации
    function login() {
      const username = document.getElementById('username').value;
      const password = document.getElementById('password').value;

      // Отправка данных на сервер
      fetch('http://localhost:3000/login', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify({ username, password }),
      })
          .then(response => response.json())
          .then(data => {
            console.log(data);

            if (data.success) {
                // Если авторизация успешна, скрываем форму
                const authContainer = document.getElementById('auth-
container');
                if (authContainer) {
                  authContainer.style.display = 'none';
                }
                updatePlayerName(username);
                toggleMenu();
            }

        if (data.error)
```

```
            alert(data.error);
                    })
                    .catch(error => console.error('Ошибка:', error));
            }

            // Переключение показа меню
        function toggleMenu() {
          const menu = document.getElementById('menu');
          if (menu.style.display === 'none' || menu.style.display === '') {
            menu.style.display = 'flex';
          } else {
            menu.style.display = 'none';
          }
        }

            function updatePlayerName(username) {
                    const playerNameElement = document.getElementById('menu-player-name');
                    if (playerNameElement) {
                            playerNameElement.textContent = 'Добро пожаловать,' + username;
                    }
            }
        // Функция для отправки запроса на сервер при регистрации
        function register() {
          const username = document.getElementById('username').value;
          const password = document.getElementById('password').value;

          // Отправка данных на сервер
          fetch('http://localhost:3000/register', {
            method: 'POST',
```

```
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({ username, password }),
  })
    .then(response => response.json())
    .then(data => {
            if (data.error)
                alert(data.error);
            else
                alert(data.message);
      console.log(data);
    })
    .catch(error => console.error('Ошибка:', error));
}

document.addEventListener('DOMContentLoaded', () => {
        const boardSize = 10;
  let maxAttempts = 60;
        const ratingList = document.getElementById('player-rating-list');
        updateRatingList();
        const gameContainer = document.getElementById('game-content');
        var username = document.getElementById('username').value;
  const board = [];
  const ships = [
    { size: 5, positions: [] },
    { size: 4, positions: [] },
    { size: 3, positions: [] },
    { size: 2, positions: [] },
    { size: 1, positions: [] }
  ];
```

```
let attempts = 0;
let score = 0;

let infoElement;
      let difficulty = 'easy';

      function setDifficulty(selectedDifficulty) {
         difficulty = selectedDifficulty;
         switch (difficulty) {
           case 'easy':
             maxAttempts = 60;
             break;
           case 'medium':
             maxAttempts = 40;
             break;
           case 'hard':
             maxAttempts = 25;
             break;
           default:
             maxAttempts = 60;
         }
       }

function getRandomDirection() {
  return Math.random() < 0.5 ? 'horizontal' : 'vertical';
}

// Инициализация поля
function initializeBoard() {
  for (let i = 0; i < boardSize; i++) {
```

```
    const row = [];
    for (let j = 0; j < boardSize; j++) {
      row.push(0);
    }
    board.push(row);
  }
}

// Размещаем корабли
function placeShips() {
  for (const ship of ships) {
    let validPlacement = false;
    while (!validPlacement) {
      const row = Math.floor(Math.random() * boardSize);
      const col = Math.floor(Math.random() * boardSize);
      const direction = getRandomDirection();
      validPlacement = checkValidPlacement(row, col, direction, ship);
      if (validPlacement) {
      clearShipPositions(ship);
        placeShipOnBoard(row, col, direction, ship);
      }
    }
  }
}

      //Очищаем позиции кораблей для старта новой игры
      function clearShipPositions(ship) {
        ship.positions = [];
      }

// Проверяем можем ли поместить корабль на данное место
```

```
function checkValidPlacement(row, col, direction, ship) {
 if (
  row < 0 || row >= boardSize ||
  col < 0 || col >= boardSize
 ) {
  return false;
 }

 // Проверяем, что рядом нет других кораблей
 for (let i = -1; i <= ship.size; i++) {
  for (let j = -1; j <= 1; j++) {
   const newRow = row + i;
   const newCol = col + j;

   if (
    newRow >= 0 && newRow < boardSize &&
    newCol >= 0 && newCol < boardSize &&
    board[newRow][newCol] !== 0
   ) {
    return false;
   }
  }
 }

 if (direction === 'horizontal') {
  if (col + ship.size > boardSize) {
   return false;
  }

  for (let i = 0; i < ship.size; i++) {
   if (board[row][col + i] !== 0) {
```

```
        return false;
      }
    }
  } else {
    if (row + ship.size > boardSize) {
      return false;
    }

    for (let i = 0; i < ship.size; i++) {
      if (board[row + i][col] !== 0) {
        return false;
      }
    }
  }

  return true;
}

// Размещаем корабль
function placeShipOnBoard(row, col, direction, ship) {
  if (direction === 'horizontal') {
    for (let i = 0; i < ship.size; i++) {
      board[row][col + i] = 1;
      ship.positions.push({ row, col: col + i });
    }
  } else {
    for (let i = 0; i < ship.size; i++) {
      board[row + i][col] = 1;
      ship.positions.push({ row: row + i, col });
    }
  }
```

```
}

//Отображение таблицы или её рендер
function renderBoard() {
  const table = document.createElement('table');
  for (let i = 0; i < boardSize; i++) {
    const tr = document.createElement('tr');
    for (let j = 0; j < boardSize; j++) {
      const td = document.createElement('td');
      td.dataset.row = i;
      td.dataset.col = j;
      tr.appendChild(td);
    }
    table.appendChild(tr);
  }
  gameContainer.appendChild(table);
}

//Обработка клика по ячейке таблице
        function handleCellClick(event) {
          if (attempts >= maxAttempts) {
            alert('Игра завершена. Вы использовали все попытки.');
            showScore();
                  updateScore(username, score);
            toggleMenu();
            return;
          }

          const row = parseInt(event.target.dataset.row);
          const col = parseInt(event.target.dataset.col);
```

```
        if (board[row][col] === 1) {
          event.target.classList.add('hit');
  board[row][col] = 2;
          score++;
          checkVictory();
          showScore(score);
        } else {
          attempts++;
        }

        updateInfo();

        if (attempts === maxAttempts) {
          alert('Игра завершена. Вы использовали все попытки.');
          markRemainingShipParts();
          showScore();
                updateScore(username, score);
          toggleMenu();
        }
      }

// Проверяем попадание по всем кораблям
      function checkVictory() {
        let allShipsHit = true;
        for (const ship of ships) {
          for (const position of ship.positions) {
            if (board[position.row][position.col] !== 2) {
              allShipsHit = false;
              break;
            }
          }
```

```
        }

        if (allShipsHit) {
          const difficultyMultiplier = getDifficultyMultiplier();
          const finalScore = Math.round(score * difficultyMultiplier);
          alert(`Поздравляем! Вы победили!\nВаш итоговый счет: ${finalScore}`);
                updateScore(username, finalScore);
          showScore(finalScore);

          attempts = 0;
          score = 0;

          toggleMenu();
          removeGameTable();
        }
      }

      function removeGameTable() {
        const existingTable = document.querySelector('table');
        if (existingTable) {
          gameContainer.removeChild(existingTable);
        }

        // Удаляем элемент отображения оставшихся попыток
        const infoElement = document.getElementById('info');
        if (infoElement) {
          gameContainer.removeChild(infoElement);
        }

        // Удаляем элемент отображения счёта
```

```
            const scoreElement = document.getElementById('score');
            if (scoreElement) {
             gameContainer.removeChild(scoreElement);
            }
           }
        // Отмечаем ячейки кораблей которые не были выбраны
        function markRemainingShipParts() {
          for (const ship of ships) {
            for (const position of ship.positions) {
              const cell = document.querySelector(`td[data-row="${position.row}"][data-col="${position.col}"]`);
              if (cell && !cell.classList.contains('hit')) {
                cell.classList.add('remaining-part');
              }
            }
          }
        }

        // Обновляем оставшиеся попытки
        function updateInfo() {
          infoElement.textContent = `Оставшиеся попытки: ${maxAttempts - attempts}`;
        }

            // Показываем счёт
            function showScore(finalScore) {
              const scoreElement = document.getElementById('score');
              if (scoreElement) {
                if (finalScore !== undefined) {
                  scoreElement.textContent = `Счёт: ${finalScore}`;
                } else {
```

```
            scoreElement.textContent = `Счёт: ${score}`;
          }
        }
      }

//При победе умножаем итоговый счет
function getDifficultyMultiplier() {
  switch (maxAttempts) {
    case 60:
      return 1;
    case 40:
      return 1.5;
    case 25:
      return 2;
    default:
      return 1;
  }
}

function showDifficultyMenu() {
        const menu = document.getElementById('menu');
        menu.style.display = 'none';

        // Проверяем наличие элемента таблицы перед его удалением
        const existingTable = document.querySelector('table');
        if (existingTable) {
         gameContainer.removeChild(existingTable);
        }

        // Проверяем наличие элемента отображения оставшихся попыток
```
перед его удалением

```
            const infoElement = document.getElementById('info');
            if (infoElement) {
             gameContainer.removeChild(infoElement);
            }

            createDifficultyMenu();
}

    function createDifficultyMenu() {
      const difficultyMenu = document.createElement('div');
      difficultyMenu.id = 'difficulty-menu';

      const backButton = document.createElement('div');
      backButton.textContent = 'Назад';
      backButton.className = 'menu-button';
      backButton.addEventListener('click', () => {
        difficultyMenu.style.display = 'none';
        toggleMenu();
      });
      difficultyMenu.appendChild(backButton);

      const easyButton = document.createElement('div');
      easyButton.textContent = 'Легкая';
      easyButton.className = 'menu-button';
      easyButton.addEventListener('click', () => {
        setDifficulty('easy');
        difficultyMenu.style.display = 'none';
        startGame();
      });
      difficultyMenu.appendChild(easyButton);
```

```
  const mediumButton = document.createElement('div');
  mediumButton.textContent = 'Средняя';
  mediumButton.className = 'menu-button';
  mediumButton.addEventListener('click', () => {
    setDifficulty('medium');
    difficultyMenu.style.display = 'none';
    startGame();
  });
  difficultyMenu.appendChild(mediumButton);

  const hardButton = document.createElement('div');
  hardButton.textContent = 'Сложная';
  hardButton.className = 'menu-button';
  hardButton.addEventListener('click', () => {
    setDifficulty('hard');
    difficultyMenu.style.display = 'none';
    startGame();
  });
  difficultyMenu.appendChild(hardButton);

  gameContainer.appendChild(difficultyMenu);
  difficultyMenu.style.display = 'flex';
}

    function startGame() {
            updateRatingList();
            removeGameTable();
            username = document.getElementById('username').value;
        const difficultyMenu = document.getElementById('difficulty-menu');
        if (difficultyMenu) {
          difficultyMenu.style.display = 'none';
```

```
        }

            attempts = 0;
            score = 0;
            finalScore = 0;

        setDifficulty(difficulty);

        const existingTable = document.querySelector('table');
        if (existingTable) {
         gameContainer.removeChild(existingTable);
        }

        board.length = 0;
        initializeBoard();

        renderBoard();
        placeShips();
        initInfo();
        showScore(0);

        document.querySelectorAll('td').forEach(cell => {
          cell.addEventListener('click', handleCellClick);
        });
        const scoreElement = document.getElementById('score');

            if (scoreElement) {
                scoreElement.style.display = 'block';
            }
}
```

```
    function exitGame() {
        showAuthForm();
        removeGameTable();
        toggleMenu();
    }

function initInfo() {
  infoElement = document.createElement('div');
  infoElement.id = 'info';
  gameContainer.appendChild(infoElement);

  const scoreElement = document.createElement('div');
      scoreElement.id = 'score';
      scoreElement.textContent = 'Счёт: 0';
      gameContainer.appendChild(scoreElement);

      gameContainer.appendChild(infoElement);
}

initMenu();

function initMenu() {
  const menu = document.createElement('div');
  menu.id = 'menu';

      const playerName = document.createElement('div');
      playerName.id = 'menu-player-name';
      playerName.textContent = 'Добро пожаловать';
      menu.appendChild(playerName);

  const playButton = document.createElement('div');
```

```
    playButton.className = 'menu-button';
    playButton.textContent = 'Играть';
    playButton.addEventListener('click', showDifficultyMenu);
    menu.appendChild(playButton);

        const exitButton = document.createElement('div');
        exitButton.className = 'menu-button';
        exitButton.textContent = 'Выйти';
        exitButton.addEventListener('click', exitGame);
        menu.appendChild(exitButton);

    gameContainer.appendChild(menu);
  }

 // Функция для отображения формы авторизации и регистрации
function showAuthForm() {
  const authContainer = document.getElementById('auth-container');
  if (authContainer) {
    authContainer.style.display = 'block';
  }
}
        // Обновление счета игрока
        function updateScore(username, newScore) {
                // Отправляем запрос на сервер для обновления счета
                fetch('http://localhost:3000/update-score', {
                        method: 'PUT',
                        headers: {
                                'Content-Type': 'application/json',
                        },
                        body: JSON.stringify({ username: username, score:
newScore }),
```

```
                })
                .then(response => response.json())
                .then(data => {
                        console.log('Счет успешно обновлен:', data);
                })
                .catch(error => console.error('Ошибка при обновлении счета:',
error));
        }

        // Функция для обновления списка рейтинга
    function updateRatingList() {
        // Очищаем текущий список
        ratingList.innerHTML = '';

        // Отправляем запрос на сервер для получения рейтинга
        fetch('http://localhost:3000/rating?username=' + username)
            .then(response => response.json())
            .then(playerRatingData => {
                // Сортируем игроков по убыванию счета
                playerRatingData.sort((a, b) => b.score - a.score);

                // Создаем новый список
                playerRatingData.forEach((player, index) => {
                    const listItem = document.createElement('li');
                    listItem.textContent = `${index + 1}. ${player.username}:
${player.score}`;
                    ratingList.appendChild(listItem);
                });
            })
            .catch(error => console.error('Ошибка при получении рейтинга:',
error));
```

```
    }

  // Вызываем функцию отображения формы при загрузке страницы
  showAuthForm();
});
```

ПРИЛОЖЕНИЕ В

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
    body {
      display: flex;
      flex-direction: column;
      align-items: center;
      height: 100vh;
      margin: 0;
      position: relative;
    }

    table {
      border-collapse: collapse;
      margin-bottom: 10px;
    }

    td {
      width: 30px;
      height: 30px;
      border: 1px solid #ccc;
      text-align: center;
      cursor: pointer;
    }

    .ship {
```

```
  background-color: #3498db;
}

.hit {
  background-color: #e74c3c;
}

.remaining-part {
  background-color: #ccc;
}

#info {
  font-size: 18px;
  margin-bottom: 10px;
        text-align: center;
}

#score {
  position: absolute;
  top: 10px;
  left: 10px;
  font-size: 16px;
}

#menu {
  display: none;
  flex-direction: column;
  gap: 10px;
  position: absolute;
  top: 50%;
  left: 50%;
```

```
  transform: translate(-50%, -50%);
}

.menu-button {
  cursor: pointer;
  padding: 10px;
  font-size: 16px;
}

#difficulty-menu {
  display: none;
  flex-direction: column;
  gap: 10px;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
    #game-content {
        flex-grow: 1;
    }

    #rating-container {
        width: 200px;
        padding: 20px;
        background-color: #f2f2f2;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }

    #player-rating-list {
        list-style-type: none;
```

```
            padding: 0;
        }

        #player-rating-list li {
            margin-bottom: 10px;
        }

    </style>
    <title>Морской бой</title>
</head>
<body>
<script src="game.js"></script>
    <div id="game-content">
    <div id="auth-container" style="display: none;">
                <h2>Авторизация / Регистрация</h2>
                <label for="username">Имя пользователя:</label>
                <input type="text" id="username" required>
                <label for="password">Пароль:</label>
                <input type="password" id="password" required>
                <button onclick="login()">Войти</button>
                <button
onclick="register()">Зарегистрироваться</button>
            </div>
    </div>
    <div id="rating-container">
    <h2>Рейтинг игроков</h2>
    <ul id="player-rating-list"></ul>
    </div>

</body>
</html>
```

ПРИЛОЖЕНИЕ Г

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');

const app = express();
const port = 3000;

app.use(cors());

mongoose.connect('mongodb://127.0.0.1:27017/battleship', {
useNewUrlParser: true, useUnifiedTopology: true });
const db = mongoose.connection;

db.on('error', console.error.bind(console, 'MongoDB connection error:'));
db.once('open', () => {
  console.log('Connected to MongoDB');
});

// Определение схемы и модели для игрока
const playerSchema = new mongoose.Schema({
  username: String,
  password: String,
  score: Number,
});

const Player = mongoose.model('Player', playerSchema);

app.use(express.json());
```

```
// Регистрация нового игрока
app.post('/register', async (req, res) => {
  try {
    const { username, password } = req.body;
    if (username == ""){
      return res.status(400).json({ error: 'Заполните имя пользователя' });
    }
    if (password == "") {
      return res.status(400).json({ error: 'Укажите пароль' });
    }
    const existingPlayer = await Player.findOne({ username });

    if (existingPlayer) {
      return res.status(400).json({ error: 'Игрок с таким именем уже существует' });
    }

    const player = new Player({ username, password, score: 0 });
    await player.save();

    res.status(201).json({ message: 'Игрок зарегистрирован успешно' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
  }
});

// Авторизация игрока
app.post('/login', async (req, res) => {
  try {
    const { username, password } = req.body;
```

```
    const player = await Player.findOne({ username, password });

    if (!player) {
      return res.status(401).json({ error: 'Неверное имя пользователя или пароль' });
    }

    res.json({ username: player.username, score: player.score, success: true });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
  }
});

// Обновление счета игрока
app.put('/update-score', async (req, res) => {
  try {
    const { username, score } = req.body;
    const player = await Player.findOne({ username });

        if (!username || !score) {
      return res.status(400).json({ error: 'Отсутствует имя пользователя или счет' });
    }

    if (!player) {
      return res.status(404).json({ error: 'Игрок не найден' });
    }

    if (score > player.score) {
      player.score = score;
```

```
      await player.save();
      res.json({ message: 'Счет обновлен успешно' });
    }
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
  }
});

// Получение рейтинга
app.get('/rating', async (req, res) => {
  try {
    const playerRatingData = await Player.find({}, 'username score -_id');
    res.json(playerRatingData);
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
  }
});

app.listen(port, () => {
  console.log(`Сервер запущен на порту ${port}`);
});
```

ПРИЛОЖЕНИЕ Д