МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационные системы и технологии
Специальность 1–40 01 01 Программное обеспечение информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ НА ТЕМУ:

Web-приложение «Социальная сеть с возможностью создания и проведения интеллектуальных игр и викторин»

Выполнил студент Пузиков А. А.
(Ф.И.О.)

Руководитель работы к.т.н., доцент Смелов В.В.
(учен. степень, звание, должность, Ф.И.О., подпись)

И.о. зав. кафедрой к.т.н., доцент Смелов В.В.
(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой _____

Минск 2024

Содержание

Введение

В условиях стремительного развития цифровых технологий и популярности онлайн-развлечений, интеллектуальные игры и викторины становятся важным элементом развития логического мышления и социального взаимодействия. Современные социальные сети предоставляют платформу для обмена знаниями и создания игровых сообществ.

Целью проекта является разработка функционального решения для обеспечения возможности проведения интеллектуальных викторин, сочетающие в себе функции социальной сети. Приложение будет поддерживать различные роли пользователей (гость, пользователь, ведущий) с соответствующими правами доступа.

Для достижения указанной цели поставлены следующие задачи:

1. Сформулировать постановку задачи и рассмотреть аналогичные решения (рассмотрено в разделе 1 настоящей записки).
2. Спроектировать архитектуру приложения, обеспечивающую поддержку всех заявленных функций (рассмотрено в разделе 2 настоящей записки).
3. Разработать приложение по спроектированной архитектуре (рассмотрено в разделе 3 настоящей записки).
4. Организовать тестирование и отладку приложения для обеспечения его стабильной работы (освещено в разделе 4 настоящей записки).
5. Сформировать руководство пользователя и описать инструкции по использованию *web*-приложения (освещено в разделе 5 настоящей записки).

Приложение ориентировано на широкую аудиторию, включая любителей интеллектуальных игр, активных пользователей социальных сетей, а также тех, кто заинтересован в саморазвитии и участии в командных или индивидуальных викторинах.

Разрабатываемое приложение представляет собой *web*-приложение, доступное через интернет-браузеры на компьютерах, планшетах и смартфонах. Оно обеспечит взаимодействие пользователей с сервером и сохранение данных в базе данных.

1 Постановка задачи и обзор аналогичных решений

1.1 Постановка задачи

Разработка *web*-приложения «Социальная сеть с возможностью создания и проведения интеллектуальных игр и викторин» предусматривает поддержку ролей «гость», «пользователь» и «ведущий» с четким разграничением их возможностей.

Гости смогут зарегистрироваться, просматривать информацию о викторинах, играх, рейтингах игроков и профилях пользователей. Пользователи смогут авторизоваться, просматривать викторины, игры и профили, создавать игровые комнаты, редактировать профиль, участвовать в играх и викторинах, просматривать результаты, оставлять комментарии. Ведущие смогут авторизоваться, создавать вопросы, запускать и завершать игры, управлять вопросами и комментариями.

Для реализации проекта требуется анализ аналогичных решений, проработка данных и создание удобного интерфейса.

1.2 Аналитический обзор аналогов

1.2.1 Интернет-ресурс «Quiz»

«Quiz» – это онлайн-платформа, предназначенная для создания и прохождения интерактивных викторин [1]. Пользователи могут выбирать из широкого ассортимента готовых викторин или создавать собственные, используя разнообразные настройки для персонализации. Платформа ориентирована на развлечение и обучение, что делает её популярной как среди учащихся, так и в корпоративной среде.

Интерфейс интернет-ресурса «*Quiz*» представлен на рисунке 1.1.

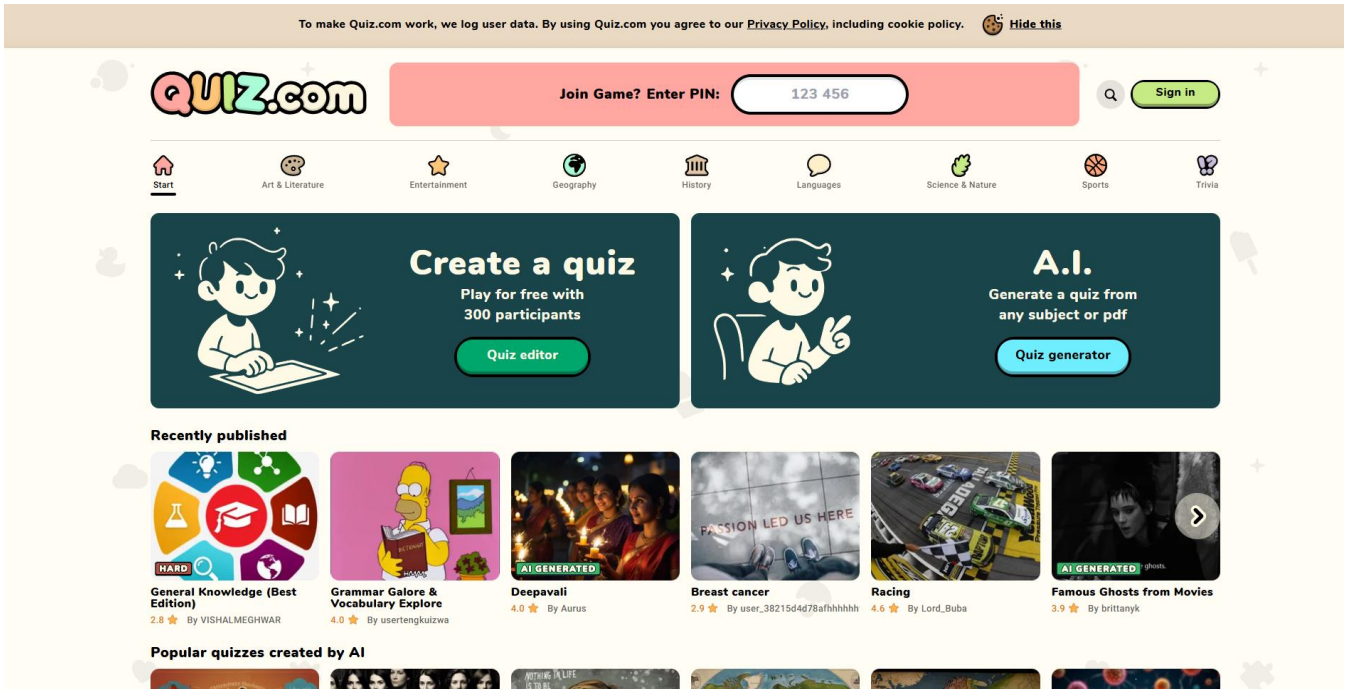

Рисунок 1.1 – Интерфейс интернет-ресурса «*Quiz*»

Достоинства:

– Широкий выбор тем: «*Quiz*» предлагает викторины на самые разные темы, от науки и искусства до поп-культуры и спорта, что позволяет привлечь интересы широкого круга пользователей.

– Возможность создания собственных викторин: пользователи могут легко создавать уникальные тесты, что делает платформу удобной для преподавателей и организаторов мероприятий.

– Интерактивность и мгновенные результаты: платформа предоставляет мгновенные результаты, что делает процесс прохождения викторин более увлекательным и познавательным.

Недостатки:

– Ограниченные социальные функции: платформа имеет ограниченные возможности для взаимодействия пользователей вне процесса прохождения викторин.

– Зависимость от интернета: «*Quiz*» требует стабильного интернет-соединения, что может быть неудобным в условиях ограниченного доступа к сети.

1.2.2 Интернет-ресурс «Kahoot!»

«Kahoot!» – образовательная платформа для создания викторин, которая часто используется в школах и университетах для проведения интерактивных занятий [2]. Она ориентирована на группы участников, что делает её идеальной для обучения и совместных мероприятий.

Интерфейс интернет-ресурса «*Kahoot!*» представлен на рисунке 1.2.

Рисунок 1.2 – Интерфейс интернет-ресурса «*Kahoot!*»

Достоинства:

– Поддержка живых игр: «*Kahoot!*» позволяет проводить викторины в режиме реального времени, что делает процесс участия более динамичным и интерактивным.

– Ориентированность на образование: платформа широко используется в образовательных учреждениях для тестирования знаний и вовлечения учеников.

– Простота создания игр: пользователи могут легко создавать свои викторины и адаптировать их под нужды учебного процесса.

Недостатки:

– Ограниченные возможности персонализации: несмотря на возможность создания собственных викторин, пользователи не могут изменять многие аспекты интерфейса или игровых сценариев.

– Фокус на группе: платформа больше подходит для групповых мероприятий, чем для индивидуального использования.

1.2.3 Интернет-ресурс «Jackbox Games»

«Jackbox Games» – это набор многопользовательских игр, которые можно играть на различных устройствах с участием большого количества людей [3]. Игры отличаются оригинальными сценариями и креативными задачами, ориентированными на взаимодействие с друзьями.

Интерфейс интернет-ресурса «*Jackbox Games*» представлен на рисунке 1.3.

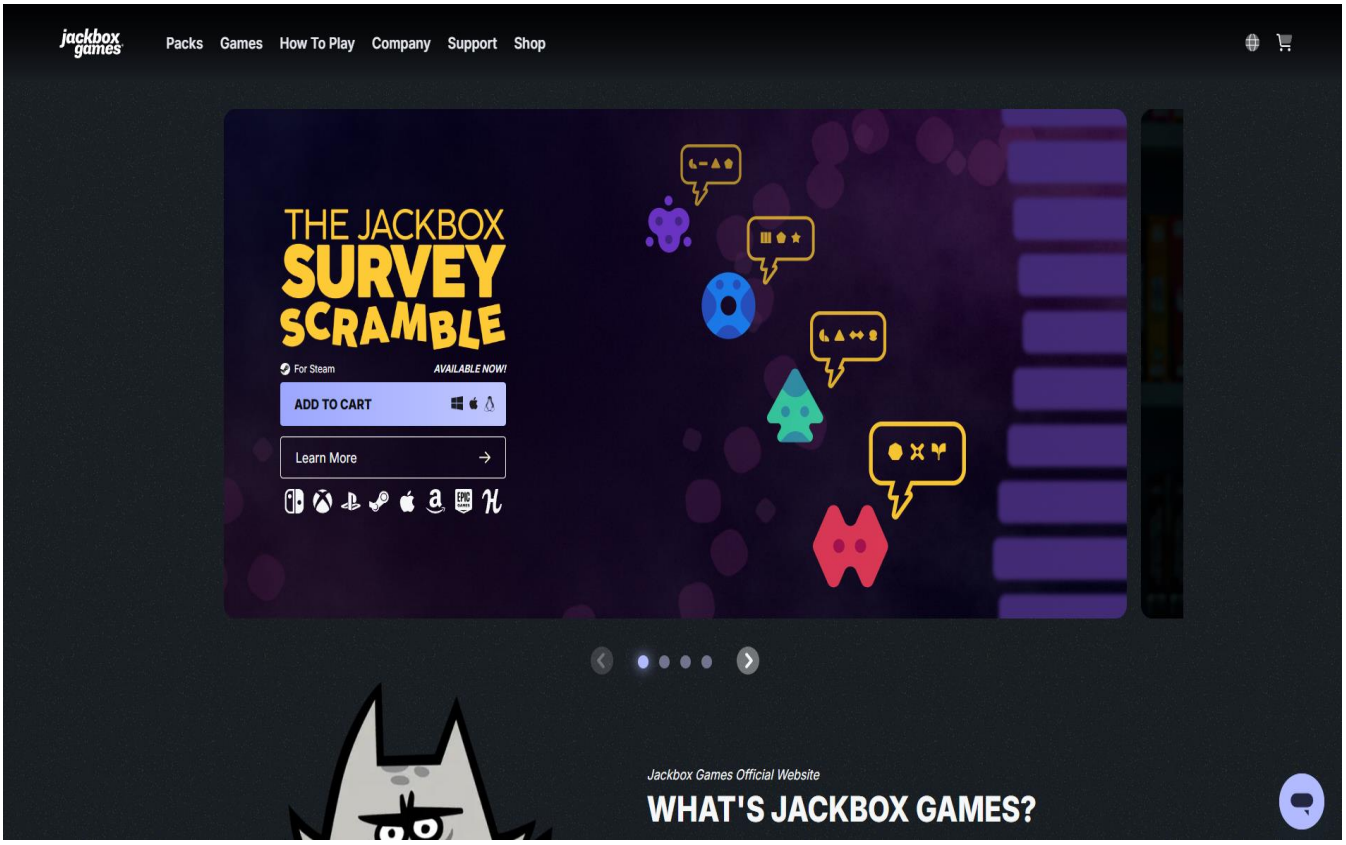

Рисунок 1.3 – Интерфейс интернет-ресурса «*Jackbox Games*»

Достоинства:

– Веселые и креативные игры: «*Jackbox Games»* предлагает оригинальные игровые сценарии, которые делают каждый игровой сеанс уникальным.

– Поддержка большого числа участников: играть в «*Jackbox Games»* можно в больших компаниях, что делает его идеальным для вечеринок и социальных мероприятий.

– Мульти-платформенность: игры можно запускать на любом устройстве, а участники подключаются через свои смартфоны.

Недостатки:

– Отсутствие социальных функций: платформа не предоставляет возможностей для взаимодействия пользователей вне игрового процесса.

– Ограниченное количество игр: несмотря на разнообразие игр в каждом наборе, их количество ограничено, и игроки могут быстро потерять интерес.

1.3 Выводы по разделу

1. Основной задачей разработки *web*-приложения является создание платформы с поддержкой ролей «гость», «пользователь» и «ведущий», а также реализация функционала для взаимодействия с викторинами и игровыми комнатами, управления профилями и данными пользователей.

2. В ходе анализа существующих решений, таких как «*Quiz*», «*Kahoot*!» и «*Jackbox Games*», были выявлены их сильные и слабые стороны. «*Quiz*» предоставляет простую платформу для создания викторин. «*Kahoot!*» ориентирован на интерактивность и образовательное использование. «*Jackbox Games*» фокусируется на развлекательных многопользовательских играх.

3. Разрабатываемое приложение объединяет сильные стороны аналогов, расширяя функционал за счет интеграции элементов взаимодействия между пользователями и ведения интеллектуальных игр.

2 Проектирование web-приложения

2.1 Функциональность web-приложения

В системе предусмотрены следующие роли: Гость, Пользователь и Ведущий. Каждая из этих ролей имеет определённый набор прав и ограничений, в зависимости от своей функциональности. Для лучшего понимания взаимодействия ролей и функций, была построена диаграмма вариантов использования, которая отражает основные действия пользователей в приложении. Она отображает ключевые сценарии использования, охватывая все важные процессы и действия. Диаграмма вариантов использования представлена на рисунке 2.1.

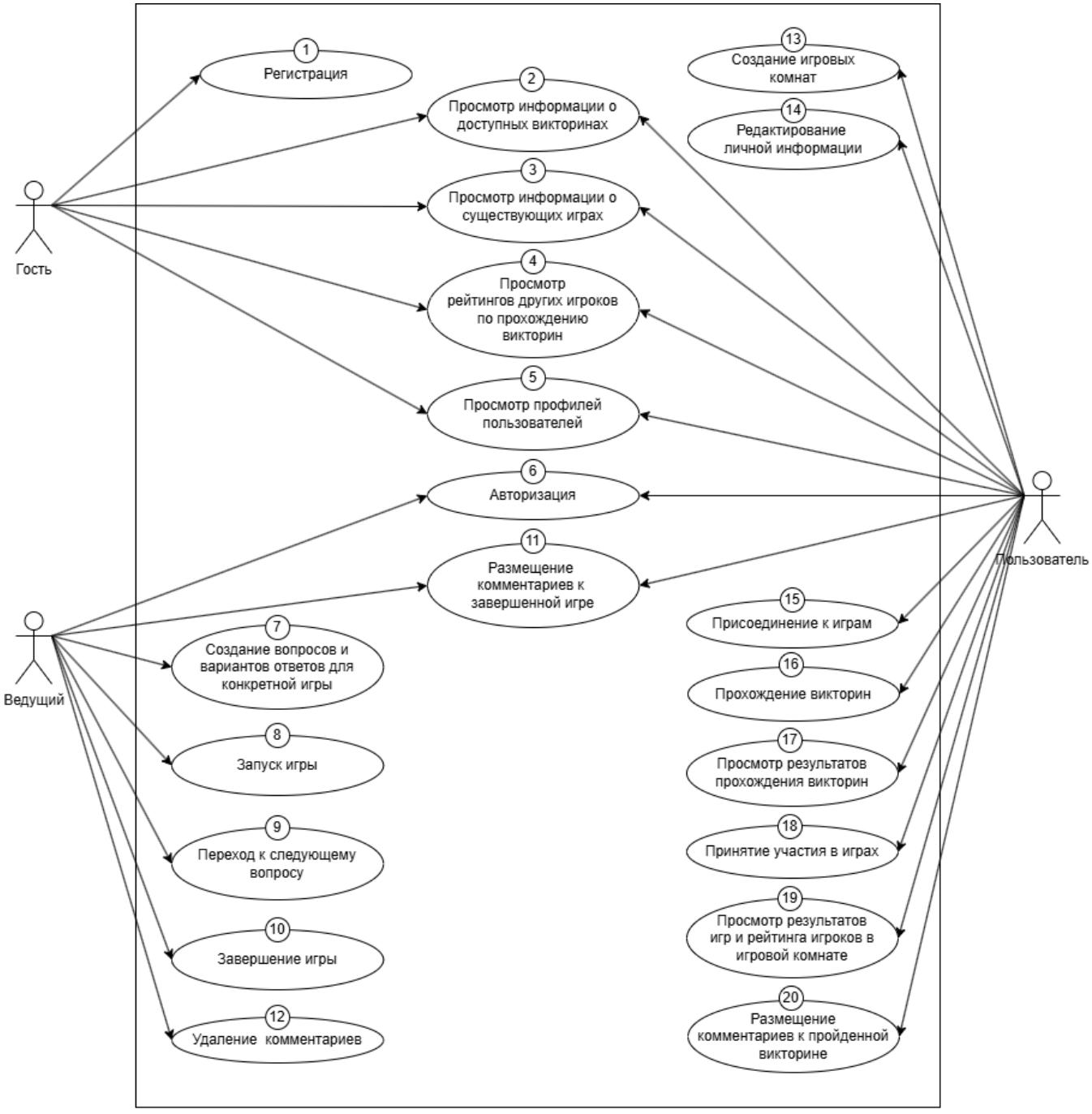

Рисунок 2.1 – Диаграмма вариантов использования

В таблице 2.1 представлено описание ролей.

Таблица 2.1 – Описание ролей

Роль	Описание роли
Гость	Пользователь, не авторизованный в системе, имеющий доступ к ограниченному функционалу. Может просматривать информацию о викторинах, играх, рейтингах и профилях пользователей. Не может создавать игровые комнаты или участвовать в играх
Пользователь	Авторизованный пользователь, имеющий расширенные права. Может просматривать информацию о викторинах, играх, профилях пользователей, создавать игровые комнаты, редактировать личную информацию, участвовать в играх, проходить викторины, оставлять комментарии и просматривать результаты игровых сессий
Ведущий	Пользователь с правами администратора для проведения игр. Может создавать вопросы и варианты ответов, запускать и завершать игру, переходить между вопросами, а также удалять комментариями к завершенным играм. Ведущий несет ответственность за управление игровым процессом

В таблице 2.2 представлены все функции, их номера, наименования и описания, а также соответствующие роли пользователей.

Таблица 2.2 – Описание функций

№	Роль	Наименование функции	Описание функций
1	Гость	Регистрация в системе	Процесс создания учетной записи пользователя для доступа к системе
2	Гость, Пользователь	Просмотр информации о доступных викторинах	Отображение списка доступных викторин с кратким описанием
3	Гость, Пользователь	Просмотр информации о существующих играх	Получение сведений о текущих играх, включая их статус
4	Гость, Пользователь	Просмотр рейтингов других игроков по прохождению викторин	Ознакомление с таблицей лидеров, содержащей данные о лучших результатах игроков
5	Гость, Пользователь	Просмотр профилей пользователей	Получение информации о других пользователях, включая их достижения и статистику
6	Пользователь, Ведущий	Авторизация в системе	Вход в систему с использованием учетных данных пользователя
7	Ведущий	Создание вопросов и вариантов ответов для конкретной игры	Процесс добавления вопросов с вариантами ответов для проведения игры

Продолжение таблицы 2.2

№	Роль	Наименование функции	Описание функций
8	Ведущий	Запуск игры	Инициация начала игры для участников
9	Ведущий	Переход к следующему вопросу	Переход к следующему вопросу игры
10	Ведущий	Завершение игры	Фиксация результата игры и завершение игрового процесса
11	Пользователь, Ведущий	Размещение комментариев к завершенной игре	Добавление отзывов или замечаний к завершенной игре
12	Ведущий	Удаление комментариев	Удаление ранее опубликованных комментариев к играм
13	Пользователь	Создание игровых комнат	Создание игровых комнат для проведения игр
14	Пользователь	Редактирование личной информации	Изменение данных профиля, включая имя, фамилию и картинку пользователя
15	Пользователь	Присоединение к играм	Возможность присоединения к игровой комнате
16	Пользователь	Прохождение викторин	Участие в викторинах, включая ответ на вопросы и завершение викторин
17	Пользователь	Просмотр результатов прохождения викторин	Получение данных о своих ответах, правильности и общем результате викторины
18	Пользователь	Принятие участия в играх	Возможность пользователя быть участником игры
19	Пользователь	Просмотр результатов игр и рейтинга игроков в игровой комнате	Анализ итогов завершенных игр и сравнение с результатами других участников игровой комнаты
20	Пользователь	Размещение комментариев к пройденной викторине	Возможность оставить отзыв о конкретной викторине, в которой пользователь принимал участие

Проектирование функциональности *web*-приложения позволяет создать удобный и интуитивно понятный интерфейс, удовлетворяющий потребности различных групп пользователей. Диаграмма вариантов использования и таблицы ролей помогают визуализировать и структурировать ключевые процессы взаимодействия с системой. Такое структурирование упрощает разработку и тестирование приложения, а также улучшает его масштабируемость и возможность добавления новых функций в будущем.

2.2 Логическая схема базы данных

Структура базы данных состоит из четырнадцати таблиц, каждая из которых отвечает за хранение и управление определенными данными. Логическая схема базы данных представлена на рисунке 2.2.

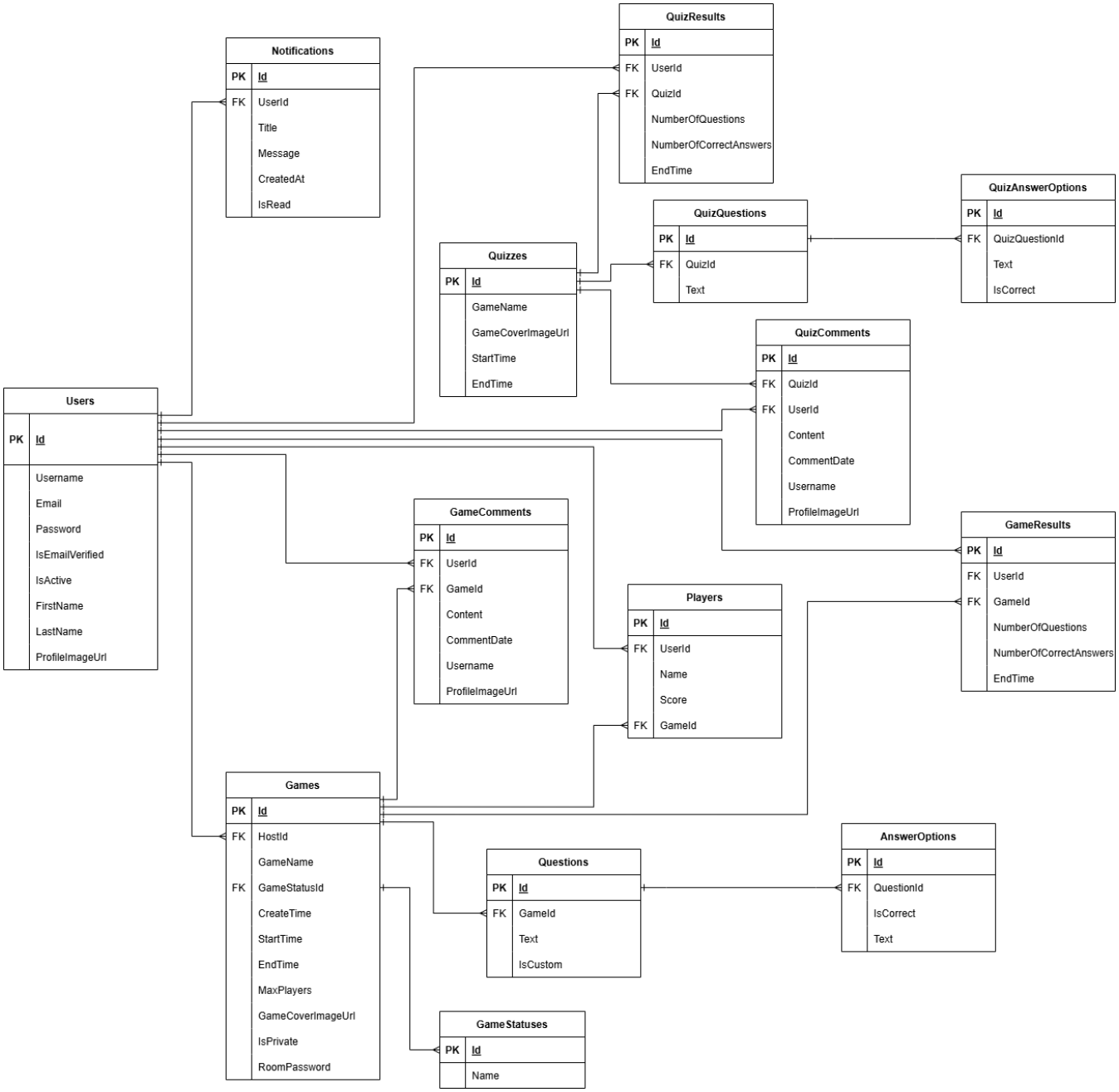

Рисунок 2.2 – Логическая схема базы данных

В таблице 2.3 предоставлена краткая информация о каждой таблице, включая наименования и описание все сущностей базы данных. Эти таблицы взаимодействуют между собой с помощью внешних ключей, обеспечивая целостность данных и поддержку ключевых функциональных возможностей системы, таких как управление пользователями, хранение викторин, вопросов и ответов, а также отслеживание прогресса и результатов.

Таблица 2.3 – Таблицы базы данных

Название таблицы	Описание использования
AnswerOptions	Хранит варианты ответов для вопросов, включая текст ответа, его корректность и связь с вопросами
GameComments	Содержит комментарии пользователей к играм, включая текст комментария, имя пользователя, аватар и дату
GameResults	Хранит результаты игр для пользователей, включая количество правильных ответов и общее количество вопросов
Games	Содержит информацию об играх, таких как название, ведущий, время начала/окончания, статус, приватность и пароль
GameStatuses	Хранит возможные статусы игр
Notifications	Содержит уведомления для пользователей, включая заголовок, текст сообщения, дату создания и статус прочтения
Players	Хранит данные о игроках игр, включая имя, счёт, связь с игрой и пользователем
Questions	Содержит текст вопросов для игр, а также информацию о кастомных вопросах
QuizAnswerOptions	Хранит варианты ответов для викторин, включая текст ответа и информацию о корректности
QuizComments	Содержит комментарии пользователей к викторинам, включая текст комментария, имя пользователя и дату
QuizQuestions	Хранит вопросы викторин с текстом и привязкой к конкретной викторине
QuizResults	Содержит результаты прохождения викторин, включая количество правильных ответов, время завершения и пользователя
Quizzes	Хранит информацию о викторинах, включая название, обложку, время начала/окончания
Users	Хранит информацию о пользователях

AnswerOptions – хранит варианты ответов для вопросов, включая текст ответа, его корректность и связь с вопросами. Её структура представлена в таблице 2.4.

Таблица 2.4 – Структура таблицы *AnswerOptions*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор варианта ответа
Text	*NVARCHAR(200)*	Текст варианта ответа
IsCorrect	*BIT*	Признак корректности ответа
QuestionId	*INT*	Идентификатор связанного вопроса, к которому относится данный ответ

GameComments – содержит комментарии пользователей к играм, включая текст комментария, имя пользователя, аватар и дату. Структура представлена в таблице 2.5.

Таблица 2.5 – Структура таблицы *GameComments*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор комментария
GameId	*INT*	Идентификатор игры, к которой относится комментарий
Username	*NVARCHAR(MAX)*	Имя пользователя, оставившего комментарий
UserId	*INT*	Идентификатор пользователя
Content	*NVARCHAR(500)*	Текст комментария
CommentDate	*DATETIME2(7)*	Дата и время создания комментария
ProfileImageUrl	*NVARCHAR(MAX)*	Ссылка на аватар пользователя

GameResults – хранит результаты игр для пользователей, включая количество правильных ответов и общее количество вопросов. Структура представлена в таблице 2.6.

Таблица 2.6 – Структура таблицы *GameResults*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор результата игры
NumberOfCorrectAnswers	*INT*	Количество правильных ответов
NumberOfQuestions	*INT*	Общее количество вопросов
UserId	*INT*	Идентификатор пользователя, к которому относится результат
GameId	*INT*	Идентификатор игры, связанной с результатом

Games – содержит информацию об играх, таких как название, ведущий, время начала/окончания, статус, приватность и пароль. Её структура представлена в таблице 2.7.

Таблица 2.7 – Структура таблицы *Games*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор игры.
GameName	*NVARCHAR(100)*	Название игры
HostId	*INT*	Идентификатор ведущего игры
CreateTime	*DATETIME2(7)*	Время создания игры
StartTime	*DATETIME2(7)*	Время начала игры
EndTime	*DATETIME2(7)*	Время окончания игры
MaxPlayers	*INT*	Максимальное количество участников

Продолжение таблицы 2.7

Название	Тип данных	Назначение
GameCoverImageUrl	*NVARCHAR(MAX)*	Ссылка на изображение обложки игры
IsPrivate	*BIT*	Признак приватности игры
RoomPassword	*NVARCHAR(50)*	Пароль для доступа к приватной игре
GameStatusId	*INT*	Идентификатор текущего статуса игры

GameStatuses – содержит возможные статусы игр. Структура представлена в таблице 2.8.

Таблица 2.8 – Структура таблицы *GameStatuses*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор статуса
Name	*NVARCHAR(100)*	Название статуса игры

Notifications – содержит уведомления для пользователей, включая заголовок, текст сообщения, дату создания и статус прочтения. Структура представлена в таблице 2.9.

Таблица 2.9 – Структура таблицы *Notifications*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор уведомления
UserId	*INT*	Идентификатор пользователя, получившего уведомление
Title	*NVARCHAR(100)*	Заголовок уведомления
Message	*NVARCHAR(1000)*	Текст сообщения уведомления
CreatedAt	*DATETIME2(7)*	Дата и время создания уведомления
IsRead	*BIT*	Признак прочтения уведомления

Players – Хранит данные о игроках игр, включая имя, счёт, связь с игрой и пользователем. Структура таблицы представлена в таблице 2.10.

Таблица 2.10 – Структура таблицы *Players*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор игрока
GameId	*INT*	Идентификатор игры, к которой относится игрок
Name	*NVARCHAR(100)*	Имя игрока
Score	*INT*	Текущий счёт игрока
UserId	*INT*	Идентификатор пользователя, связанного с игроком

Questions – содержит информацию о вопросах, связанных с играми. Структура представлена в таблице 2.11.

Таблица 2.11 – Структура таблицы *Questions*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор вопроса
Text	*NVARCHAR(500)*	Текст вопроса
GameId	*INT*	Идентификатор игры, к которой относится вопрос
IsCustom	*BIT*	Признак кастомного вопроса

QuizAnswerOptions – хранит варианты ответов для викторин, включая текст ответа и информацию о корректности. Структура представлена в таблице 2.12.

Таблица 2.12 – Структура таблицы *QuizAnswerOptions*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор варианта ответа для викторины
Text	*NVARCHAR(200)*	Текст варианта ответа
IsCorrect	*BIT*	Признак корректности ответа
QuizQuestionId	*INT*	Идентификатор вопроса викторины, связанного с ответом

QuizComments – Содержит комментарии пользователей к викторинам, включая текст комментария, имя пользователя и дату. Структура представлена в таблице 2.13.

Таблица 2.13 – Структура таблицы *QuizComments*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор комментария к викторине
QuizId	*INT*	Идентификатор викторины, к которой относится комментарий
UserId	*INT*	Идентификатор пользователя, оставившего комментарий
Content	*NVARCHAR (500)*	Текст комментария
Username	*NVARCHAR (max)*	Имя пользователя, оставившего комментарий
ProfileImageUrl	*NVARCHAR (max)*	Ссылка на аватар пользователя
CommentDate	*DATETIME2(7)*	Дата и время создания комментария

QuizQuestions – содержит вопросы викторины. Структура представлена в таблице 2.14.

Таблица 2.14 – Структура таблицы *QuizQuestions*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор вопроса викторины
Text	*NVARCHAR(500)*	Текст вопроса викторины
QuizId	*INT*	Идентификатор викторины, к которой относится вопрос

QuizResults – таблица для хранения результатов викторины. Структура представлена в таблице 2.15.

Таблица 2.15 – Структура таблицы *QuizResults*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор результата викторины
NumberOfCorrectAnswers	*INT*	Количество правильных ответов
NumberOfQuestions	*INT*	Общее количество вопросов в викторине
EndTime	*DATETIME2(7)*	Дата и время завершения викторины
UserId	*INT*	Идентификатор пользователя, связанного с результатом
QuizId	*INT*	Идентификатор викторины, связанной с результатом

Quizzes – хранит информацию о викторинах, связанных с играми. Структура представлена в таблице 2.16.

Таблица 2.16 – Структура таблицы *Quizzes*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор викторины
GameName	*NVARCHAR(200)*	Название викторины
GameCoverImageUrl	*NVARCHAR(500)*	Ссылка на изображение обложки викторины
StartTime	*DATETIME2(7)*	Дата и время начала викторины
EndTime	*DATETIME2(7)*	Дата и время окончания викторины

Users – хранит информацию о пользователях, зарегистрированных в системе. Структура представлена в таблице 2.17.

Таблица 2.17– Структура таблицы *Users*

Название	Тип данных	Назначение
Id	*INT*	Уникальный идентификатор пользователя
Email	*NVARCHAR(MAX)*	Адрес электронной почты пользователя
Username	*NVARCHAR(MAX)*	Имя пользователя
Password	*NVARCHAR(MAX)*	Пароль пользователя
IsEmailVerified	*BIT*	Признак подтверждения электронной почты
IsActive	*BIT*	Признак активности пользователя
FirstName	*NVARCHAR(MAX)*	Имя пользователя
LastName	*NVARCHAR(MAX)*	Фамилия пользователя
ProfileImageUrl	*NVARCHAR(MAX)*	Ссылка на аватар пользователя

Помимо самих таблиц базы данных, были определены связи этих таблиц. Так, каждая таблица, которая имеет столбец, значения в котором, зависят от значений в другом столбце, этой или другой таблицы, образует связь. В таблице 2.18 представлены связи между таблицами базы данных.

Таблица 2.18 – Связи между таблицами базы данных

Таблица источник	Связанная таблица	Тип связи
Users	*GameComments*	Один ко многим
Users	*GameResults*	Один ко многим
Users	*Players*	Один ко многим
Users	*Notifications*	Один ко многим
Users	*QuizComments*	Один ко многим
Users	*QuizResults*	Один ко многим
Users	*Games*	Один ко многим
Games	*GameComments*	Один ко многим
Games	*GameResults*	Один ко многим
Games	*Questions*	Один ко многим
Games	*Players*	Один ко многим
GameStatuses	*Games*	Один ко многим
Questions	*AnswerOptions*	Один ко многим
Quizzes	*QuizQuestions*	Один ко многим
Quizzes	*QuizComments*	Один ко многим
Quizzes	*QuizResults*	Один ко многим
QuizQuestions	*QuizAnswerOptions*	Один ко многим

Читать данную таблицу следует как: связанная таблица ссылается на таблицу источник таким образом, что одной записи в таблице источнике соответствует одна или множество записей в связанной таблице, в зависимости от типа связи.

Таким образом, представленные таблицы и связи формируют основу базы данных, обеспечивая возможность гибкого хранения и обработки всех данных, необходимых для функционирования системы.

2.3 Выбор программной платформы и технологий

Для развертывания *web*-приложения выбрана серверная операционная система *ОС Windows Server 2022 Datacenter Evaluation 21H2*, которая обеспечивает надежность, открытость и активную поддержку со стороны сообщества [4]. *Windows Server* широко применяется благодаря простоте настройки, богатому набору доступного программного обеспечения и поддержке контейнеризации, что делает её оптимальным выбором для размещения *web*-приложений.

2.3.1 Выбор стека для создания пользовательского интерфейса

Для разработки пользовательского интерфейса *web*-приложения было принято решение использовать библиотеку *React 18.3.1* [5].

Основные причины данного выбора:

– компонентный подход: позволяет создавать *UI* на основе переиспользуемых компонентов;

– высокая производительность: использует виртуальный *DOM* для эффективного обновления интерфейса, минимизируя количество изменений в реальном *DOM* и повышая производительность приложения;

– гибкость и кроссплатформенность;

– большая экосистема и сообщество: имеет широкую поддержку со стороны разработчиков, множество готовых библиотек и инструментов для работы с *UI*, что позволяет ускорить процесс разработки.

2.3.2 Выбор стека для создания API

Для разработки серверного приложения (*API*) было решено использовать *ASP.NET Core Web API 8.0.400* в сочетании с *Entity Framework 8.0.8* и *MS SQL Server 2019 (RTM) 15.0.2000.5* для работы с базой данных [6, 7, 8].

Основные причины выбора *ASP.NET Core Web API*:

– высокая производительность и масштабируемость, обеспечиваемые оптимизированным серверным фреймворком, который подходит для разработки *RESTful API*;

– полная поддержка языка *C#*, который позволяет реализовать надежную и типобезопасную серверную логику [9];

– кроссплатформенность, что позволяет развертывать приложения на *Linux* (включая *Ubuntu Server*) и *Windows*;

– богатая экосистема и встроенная поддержка безопасности, включая интеграцию с *JWT* для авторизации и аутентификации.

Основные причины выбора *Entity Framework*:

– предоставляет удобный *ORM*-инструмент для работы с базами данных;

– поддержка миграций, что облегчает управление схемой базы данных на протяжении жизненного цикла приложения;

– гибкость в настройке и возможность прямого выполнения *SQL*-запросов при необходимости.

Основные причины выбора *MS SQL Server*:

– надежное решение для управления реляционными базами данных, с широким набором инструментов для мониторинга и оптимизации производительности;

– отличная интеграция с *Entity Framework*, минимизирует настройку и упрощает процесс разработки;

– высокая совместимость с современными корпоративными системами и облачными решениями.

2.3.3 REST-архитектура

В данном проекте для разработки приложения используется стек *ASP.NET Core* и *React*. Архитектура сервера построена в соответствии с принципами *REST*

(Representational State Transfer), что обеспечивает удобное взаимодействие между клиентом и сервером через *HTTP*-протокол версии 1.1 [10].

Основные компоненты:

– контроллеры обрабатывают *HTTP*-запросы от клиента и управляют потоком данных между сервисами и внешним миром. В *REST*-архитектуре контроллеры обеспечивают реализацию *CRUD*-операций (*Create, Read, Update, Delete*);

– сервисы представляют собой бизнес-логику приложения и отвечают за обработку данных. В данном проекте сервисы работают с моделью данных и выполняют все основные операции;

2.4 Архитектура web-приложения

Система, изображённая на схеме, представляет собой распределённую архитектуру с несколькими компонентами, работающими на различных операционных системах, использующих различные протоколы связи и версии ПО.

Структурная схема приложения представлена на рисунке 2.3.

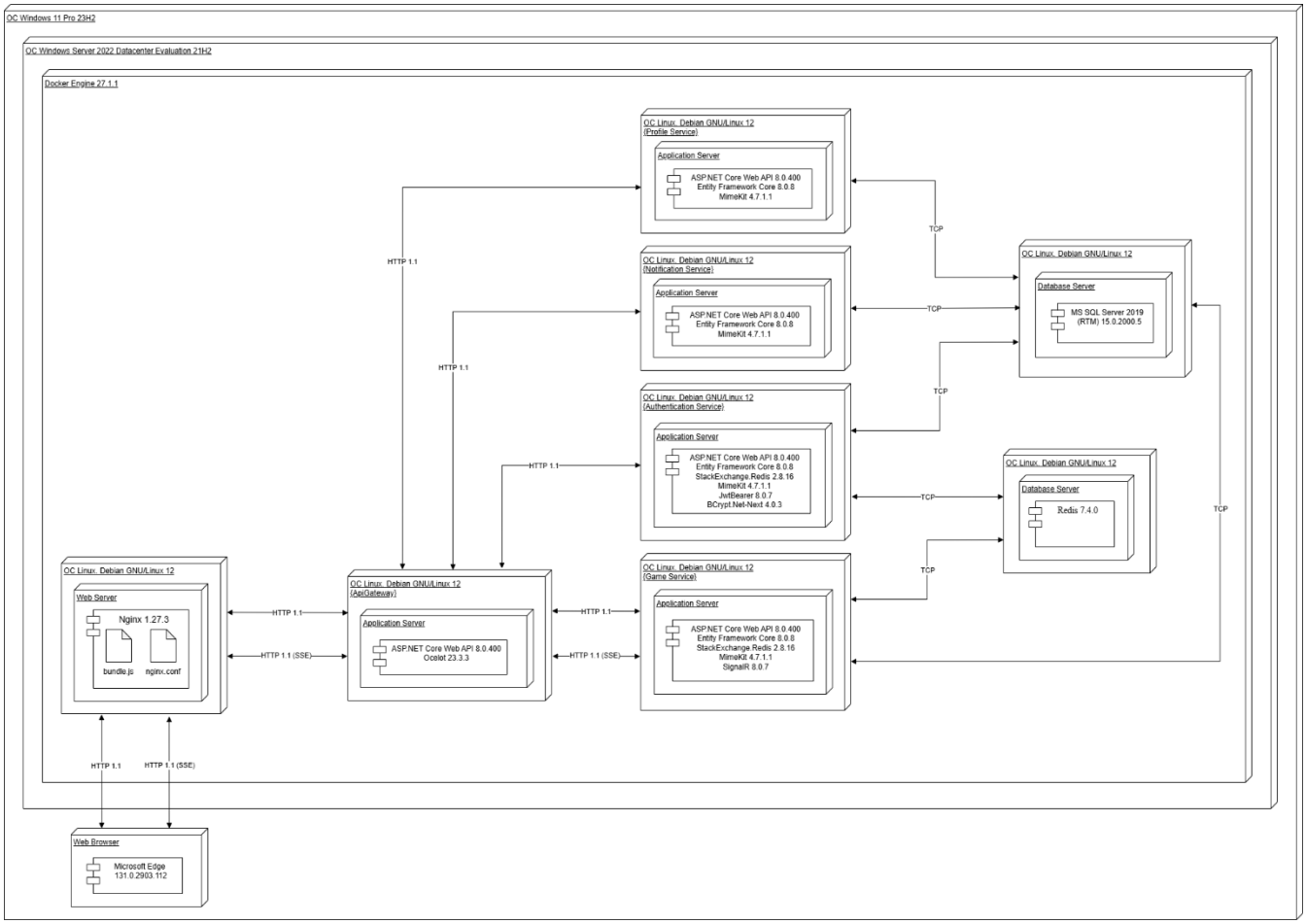

Рисунок 2.3 – Структурная схема приложения

ОС Linux, Debian GNU/Linux 12 – используется для работы большинства сервисов в системе, включая серверную часть и базы данных [11].

MS SQL Server (система управления базами данных) также развернута на этой операционной системе, обеспечивая хранение и управление данными.

ОС Windows Server 2022 Datacenter Evaluation 21H2 – используется для развертывания *Docker* контейнеров на движке версии 27.1.1 [12].

Клиенты работают на операционных системах, включая: *Windows 11*, где используется *Microsoft Edge 131.0.2903.112* [13].

Система представляет собой распределённую архитектуру, состоящую из серверных, клиентских компонентов, а также баз данных и систем кэширования. Компоненты развернуты на операционной системе *Debian GNU/Linux 12* и взаимодействуют через стандартные сетевые протоколы: *HTTP* 1.1, *SSE* и *TCP* [14, 15].

Серверная часть построена на *ASP.NET Core* 8.0 и включает несколько сервисов:

– *Profile Service* отвечает за логику взаимодействия с пользователями.

– *Game Service* обрабатывает игровую функциональность.

– *Notification Service* передаёт уведомления пользователям.

– *Authentication Service* управляет аутентификацией и авторизацией.

Эти сервисы взаимодействуют между собой и с клиентскими компонентами по *HTTP* 1.1. Для отправки событий в реальном времени используется протокол *Server-Sent Events* (*SSE*).

Хранение данных обеспечивается *MS SQL Server*, развернутым на той же операционной системе. Для передачи данных с сервисами используется *TCP*-соединение, что гарантирует надёжный и упорядоченный обмен информацией. Кэширование реализовано с помощью *Redis 7.4.0*, который ускоряет доступ к данным и уменьшает нагрузку на основную базу [16].

Клиентская часть разработана на *ReactJS 18.3.1* с использованием *Vite 5.4.0* для сборки и *Node.js 20.17.0* для обработки сборочного процесса [17, 18]. Динамическое обновление данных и взаимодействие с сервером происходит через *HTTP* 1.1 и *SSE*.

Маршрутизация запросов осуществляется с помощью *Nginx 1.27.3* [19]. Он обслуживает статические ресурсы фронтенда: файлы index.html, bundle.js и другие. Файл *bundle.js* будет находиться внутри контейнера *Nginx* в директории */usr/share/nginx/html*.

Клиенты системы – это *web*-приложения, запускаемые в браузерах (например, *Microsoft Edge*). Данные передаются с использованием *HTTP* 1.1 для *REST*-запросов и *SSE* для односторонней связи. Двусторонняя коммуникация реализована с помощью *SignalR 8.0.7*, где *SSE* используется для отправки событий от сервера клиенту [20].

Протоколы взаимодействия:

– *HTTP* 1.1 используется для *REST API* и обмена данными между клиентами и сервисами.

– *SSE* применяется для передачи событий в реальном времени, что особенно важно для интерфейсов с живыми обновлениями.

– *TCP* обеспечивает надёжный обмен данными между сервисами, базой данных и *Redis*.

Таким образом, система использует надёжную распределённую архитектуру, включающую серверные сервисы на *ASP.NET Core* 8.0, *React Vite* для фронтенда,

MS SQL Server для хранения данных и *Redis* для кэширования. *Nginx* и использование протоколов *HTTP* 1.1, *SSE* и *TCP* обеспечивают эффективное взаимодействие компонентов и передачу данных в реальном времени.

2.5 Выводы по разделу

1. *Web*-приложение «Социальная сеть с возможностью создания и проведения интеллектуальных игр и викторин» поддерживает три роли пользователей: гость, пользователь и ведущий. Гость может зарегистрироваться, просматривать информацию о доступных викторинах, играх, рейтингах и профилях пользователей. Пользователь получает доступ к основным функциям, таким как создание игровых комнат, участие в играх, прохождение викторин, просмотр результатов и размещение комментариев. Ведущий, кроме функций пользователя, может создавать вопросы, управлять игровым процессом и редактировать комментарии.
2. Приложение реализует ключевые функции: регистрация, авторизация и управление учетными записями, создание и управление викторинами, игровыми комнатами, а также взаимодействие с пользователями через комментарии, просмотр рейтингов, результатов игр и викторин для повышения интерактивности и вовлечения пользователей. Полный перечень функций представлен в таблице 2.2.
3. Логическая структура базы данных включает таблицы для хранения информации о пользователях, викторинах, играх, результатах и комментариях. Эта структура обеспечивает надежное хранение данных и позволяет быстро обрабатывать запросы.
4. Серверная часть разработана с использованием *ASP.NET Core Web API, MS SQL Server* и *Entity Framework*, что гарантирует стабильность и высокую производительность приложения. Клиентская часть использует *React* для создания удобного пользовательского интерфейса.

3 Реализация web-приложения

3.1 Реализация серверной части приложения

Проект реализован на основе трехуровневой архитектуры, что обеспечивает разделение функциональности на независимые слои: слой доступа к данным, слой бизнес-логики и слой представления. Данное архитектурное решение позволяет увеличить гибкость и масштабируемость системы, упростить поддержку кода, а также повысить отказоустойчивость.

Для реализации приложения выбрана платформа *.Net* с использованием языка *C#*. Это обеспечит стабильную работу с асинхронным пользовательским интерфейсом, а также позволит эффективно управлять серверной частью и базой данных.

3.1.1 Слой доступа к данным

Слой доступа к данным в проекте играет ключевую роль в управлении и хранении данных, обеспечивая безопасное и эффективное взаимодействие с базой данных.

Этот слой спроектирован как библиотека классов на языке *C#*, а в качестве *ORM* используется *Entity Framework* с подходом *Code First*. Это упрощает поддержку структуры данных и позволяет гибко адаптировать её по мере изменения бизнес-требований. В таблице 3.1 представлено соответствие моделей таблицам базы данных.

Таблица 3.1 – Соответствие моделей таблицам базы данных

Класс модель	Таблица базы данных	Описание таблицы
User	*Users*	Хранит информацию о пользователях
Game	*Games*	Содержит информацию об играх, таких как название, ведущий, время начала/окончания, статус, приватность и пароль
Player	*Players*	Хранит данные о игроках игр, включая имя, счёт, связь с игрой и пользователем
GameStatus	*GameStatuses*	Хранит возможные статусы игр
Question	*Questions*	Содержит текст вопросов для игр, а также информацию о кастомных вопросах
AnswerOption	*AnswerOptions*	Хранит варианты ответов для вопросов, включая текст ответа, его корректность и связь с вопросами
GameComment	*GameComments*	Содержит комментарии пользователей к играм, включая текст комментария, имя пользователя, аватар и дату

Продолжение таблицы 3.1

Класс модель	Таблица базы данных	Описание таблицы
QuizComment	*QuizComments*	Содержит комментарии пользователей к викторинам, включая текст комментария, имя пользователя и дату
Notification	*Notifications*	Содержит уведомления для пользователей, включая заголовок, текст сообщения, дату создания и статус прочтения
Quiz	*Quizzes*	Хранит информацию о викторинах, включая название, обложку, время начала/окончания
QuizQuestion	*QuizQuestions*	Хранит вопросы викторин с текстом и привязкой к конкретной викторине
QuizAnswerOption	*QuizAnswerOptions*	Хранит варианты ответов для викторин, включая текст ответа и информацию о корректности
QuizResults	*QuizResults*	Содержит результаты прохождения викторин, включая количество правильных ответов, время завершения и пользователя
GameResults	*GameResults*	Хранит результаты игр для пользователей, включая количество правильных ответов и общее количество вопросов

Классы моделей представляют объекты базы данных, описывающие основные сущности и их взаимосвязи. Они используют атрибуты и методы конфигурации *Entity Framework* для указания типа данных, ограничений, зависимостей и других характеристик. Это позволяет *Entity Framework* автоматически создавать соответствующие таблицы и связи в базе данных. Дабы отличать классы моделей от остальных классов, они были унаследованы от абстрактного класса *BaseModel*. Для хранения данных используется СУБД *MS SQL Server*. Скрипт создания всех таблиц представлен в приложении А.

Класс контекста данных представлен в листинге 3.1.

```
public class AppDbContext : DbContext {
  public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options) { }
  protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder) =>
optionsBuilder.UseSqlServer("Server=sqlserver;Database=BELLINI;User
Id=sa;Password=StrongPassword123!;TrustServerCertificate=true;");

  public DbSet<User> Users { get; set; }
  public DbSet<Game> Games { get; set; }
  public DbSet<Player> Players { get; set; }
```

```
  public DbSet<GameStatus> GameStatuses { get; set; }
  public DbSet<Question> Questions { get; set; }
  public DbSet<AnswerOption> AnswerOptions { get; set; }
  public DbSet<GameComment> GameComments { get; set; }
  public DbSet<QuizComment> QuizComments { get; set; }
  public DbSet<Notification> Notifications { get; set; }
  public DbSet<Quiz> Quizzes { get; set; } = null!;
  public DbSet<QuizQuestion> QuizQuestions { get; set; }
  public DbSet<QuizAnswerOption> QuizAnswerOptions {get; set;}
  public DbSet<QuizResults> QuizResults { get; set; }
  public DbSet<GameResults> GameResults { get; set; }

  protected override void OnModelCreating(ModelBuilder builder) {
    builder.ApplyConfiguration(new UserConfiguration());
    builder.ApplyConfiguration(new GameConfiguration());
    builder.ApplyConfiguration(new PlayerConfiguration());
    builder.ApplyConfiguration(new GameStatusConfiguration());
    builder.ApplyConfiguration(new QuestionConfiguration());
    builder.ApplyConfiguration(new AnswerOptionConfiguration());
    builder.ApplyConfiguration(new GameCommentConfiguration());
    builder.ApplyConfiguration(new QuizCommentConfiguration());
    builder.ApplyConfiguration(new NotificationConfiguration());
    builder.ApplyConfiguration(new QuizConfiguration());
    builder.ApplyConfiguration(new QuizAnswerOptionConfiguration());
    builder.ApplyConfiguration(new QuizQuestionConfiguration());
    builder.ApplyConfiguration(new QuizResultsConfiguration());

    QuizSeedData.Seed(modelBuilder);
    base.OnModelCreating(modelBuilder);
    }
}
```

Листинг 3.1 – Класс контекста данных

Для реализации паттерна «Репозиторий» в слое доступа к данным выделены интерфейсы и соответствующие классы репозиториев [21]. Репозитории предоставляют стандартный набор методов *CRUD* (создание, чтение, обновление, удаление), а также могут включать специфичные для приложения методы, упрощающие взаимодействие с данными.

Взаимодействие данных между слоями приложения происходит через репозитории, обеспечивающие контролируемый доступ к данным, что позволяет скрыть детали взаимодействия с базой данных.

Настройки для моделей данных и их отображения в базе данных определены с использованием интерфейса *IEntityTypeConfiguration<T>*. Классы конфигурации управляют характеристиками, такими как индексы, внешние ключи, уникальные ограничения, а также типы данных, используемые в базе данных. Этот подход позволяет отделить настройки базы данных от логики модели, сохраняя чистоту кода и увеличивая гибкость.

Entity Framework Code First позволяет управлять миграциями для обновления структуры базы данных при изменениях в модели. Благодаря миграциям процесс поддержания базы данных в актуальном состоянии становится предсказуемым и

управляемым, а история изменений фиксируется и доступна для отката при необходимости. Для создания и применения миграций необходимо в консоли диспетчера пакетов выполнить команды, представленные в листинге 3.2.

```
Add-Migration Init
Update-Database
```

Листинг 3.2 – Команды для создания и применения миграций

После выполнения команды в базе данных появятся необходимые таблицы.

3.1.2 Слой бизнес-логики

Слой бизнес-логики в проекте отвечает за реализацию функциональности и бизнес-процессов, обеспечивая взаимодействие между слоем данных и представлением. Этот слой также создан в виде библиотеки классов на языке *C#*, что обеспечивает его модульность и независимость. Все функции и процессы, необходимые для работы приложения, сосредоточены в сервисах (классах), которые обеспечивают основную бизнес-логику, а также позволяют централизованно управлять правилами и обработкой данных.

В рамках слоя бизнес-логики реализованы интерфейсы и классы сервисов, которые инкапсулируют основные операции над данными и взаимодействуют с репозиториями для выполнения *CRUD*-операций. Интерфейсы определяют набор обязательных методов и контрактов для работы с бизнес-логикой, что делает код легко расширяемым и позволяет создавать новые сервисы без изменения существующих. Каждый сервис представлен в виде отдельного класса, который реализует интерфейс и содержит конкретную бизнес-логику, необходимую для обработки данных. Пример интерфейса сервиса авторизации представлен в листинге 3.3.

```
public interface ILoginService {
    Task<TokenDto> AuthenticateAsync(LoginDto loginDto,
CancellationToken token = default);
    Task<TokenDto> RefreshTokenAsync(string refreshToken,
CancellationToken token = default);

    public string GenerateRefreshToken(User user);
    public string GenerateAccessToken(User user);
}
```

Листинг 3.3 – Код интерфейса сервиса авторизации

В слое бизнес-логики также используются *DTO* (*Data Transfer Object*) объекты, которые служат для передачи данных между слоями и обеспечивают защиту внутренней структуры данных от внешнего вмешательства. *DTO* объекты описывают только те свойства, которые необходимы для конкретных операций, и помогают избегать избыточной передачи информации. Является реализацией паттерна *DTO* (*Data Transfer Object*) [22].

Чтобы обеспечить корректность данных, передаваемых в приложении, *DTO* объекты проходят проверку с помощью валидаторов, реализованных на основе библиотеки *FluentValidation 11.9.2* [23]. Это позволяет автоматически проверять свойства, такие как длина, наличие обязательных полей и формат данных, что упрощает управление ошибками и повышает надёжность. Пример реализации валидатора для *DTO* объекта входных данных от пользователя при авторизации представлен в листинге 3.4.

```
public class LoginDtoValidator : AbstractValidator<LoginDto> {
    public LoginDtoValidator() {
        RuleFor(x => x.Email).NotEmpty().
            WithMessage("Email cannot be empty.");
        RuleFor(x => x.Email).
            EmailAddress(EmailValidationMode.AspNetCoreCompatible).
            WithMessage("Email is not valid");
        RuleFor(x => x.Password).NotEmpty().
            WithMessage("Password cannot be empty.");
        RuleFor(x => x.Password).Length(8, 60).
            WithMessage("Password length must be > 8 and < 60");
    }
}
```

Листинг 3.4 – Код интерфейса сервиса авторизации

Для обработки ошибок в слое бизнес-логики созданы собственные исключения, которые помогают управлять возникающими проблемами и обрабатывать их в рамках бизнес-логики. Собственные исключения позволяют точно указать на причину ошибки и передать её в представление, а также облегчают диагностику, благодаря централизованному управлению обработкой ошибок.

В рамках слоя бизнес-логики реализован набор ключевых сервисов, которые поддерживают основные бизнес-операции приложения: регистрацию и авторизацию пользователей, управление профилями, комментариями и кеширование данных.

Регистрация и авторизация реализованы через контроллеры регистрации и авторизации. Сервисы регистрации и авторизации выполняют валидацию данных, проверяют существующие учетные записи и обеспечивают создание новых профилей. В процессе авторизации используется безопасная система обработки и хранения данных пользователей, а также внедрены механизмы шифрования и защиты информации, что гарантирует надежную аутентификацию. Кроме того, пользовательские данные, такие как профиль и статус, кешируются в *Redis*, что позволяет значительно ускорить доступ к информации, снижая нагрузку на базу данных.

Присоединение к игре и её проведение в реальном времени обеспечивается с помощью технологии *SignalR*, позволяющей организовать двустороннюю передачу данных в реальном времени. На уровне бизнес-логики используется специализированный сервис для организации и управления игровыми сессиями, что позволяет пользователям моментально подключаться к играм и участвовать в игровом процессе. Хабы *SignalR*, используемые на уровне представления,

обрабатывают запросы в реальном времени и взаимодействуют с сервисом игр на уровне бизнес-логики, обеспечивая плавный и интерактивный игровой процесс для пользователей.

3.1.3 Слой представления (API)

Слой представления реализует интерфейс взаимодействия пользователя с функциональностью приложения и разработан на основе *ASP.NET Core Web API*. Этот слой является общей точкой доступа к сервисам бизнес-логики и предоставляет *API*-интерфейсы для управления различными компонентами приложения, такими как аутентификация, профили пользователей, уведомления и игровые функции.

Каждое *Web API* приложение в проекте разработано для выполнения своей части бизнес-логики, что способствует модульности и структурированности архитектуры. Например, *Web API* для аутентификации и регистрации пользователей предоставляет методы для создания и управления учетными записями, *Web API* для работы с профилями пользователей управляет персональной информацией, а *Web API* для управления игрой включает контроллеры вопросов, игр и викторин. Такой подход позволяет делегировать обработку различных задач отдельным контроллерам, что облегчает их поддержку и тестирование, а также делает приложение более гибким и масштабируемым. В таблице 3.2 представлены название сервисов и их описание.

Таблица 3.2 – Описание сервисов

Название сервиса	Описание
Authentication Service	Отвечает за аутентификацию пользователей, включая вход, регистрацию и управление токенами
Notification Service	Управляет отправкой уведомлений пользователям, включая получение и отображение уведомлений
Profile Service	Обрабатывает данные профиля пользователя, такие как информация о пользователе и изображения
Game Service	Управляет игровыми процессами, комментариями, вопросами, викторинами и статистикой игр. Также поддерживает *SignalR* для взаимодействия в реальном времени

Контроллеры ориентированы на высокую эффективность взаимодействия с бизнес-логикой. Каждый контроллер выполняет только необходимые действия, делегируя основную обработку данных сервисам слоя бизнес-логики. Это достигается благодаря использованию механизма внедрения зависимостей (*Dependency Injection, DI*) [24], что упрощает доступ к нужным сервисам. Таким образом, контроллеры взаимодействуют с бизнес-логикой без непосредственной работы с данными, обеспечивая строгое разделение ответственности. Пример реализации контроллера на примере сервиса авторизации представлен в листинге 3.5.

```
[ApiController] [Route("api/auth/[controller]")]
public class LoginController : ControllerBase {
    private readonly ILoginService _loginService;
    public LoginController(ILoginService authService)
        =>_loginService = authService;
    [HttpPost]
    public async Task<ActionResult<TokenDto>> Login([FromBody]
LoginDto, CancellationToken = default)
        => Ok( await _loginService.AuthenticateAsync(loginDto,
cancellationToken));
}
```

Листинг 3.5 – Реализация контроллера авторизации

Подобная тонкость в реализации контроллеров повышает производительность и делает интерфейс *API* более простым и понятным для клиента.

В таблице 3.3 представлены соответствие маршрутов, методов, контроллеров и функций. В колонке «Контроллер» слово «Controller» в конце названия класса опущено.

Таблица 3.3 – Соответствие маршрутов, методов, контроллеров и функций

Маршрут	Метод	Контроллер	№	Наименование функции
api/auth/login	*POST*	*Login*	6	Авторизация
api/auth/login/refresh	*POST*	*Login*		
api/auth/password/change	*POST*	*Password*		
api/auth/password/reset	*POST*	*Password*		
api/auth/password/forgot	*POST*	*Password*		
api/auth/password/verify	*POST*	*Password*		
api/auth/register/check-email	*POST*	*Register*		
api/auth/register/verify-code	*POST*	*Register*		
api/auth/register	*POST*	*Register*	1	Регистрация
api/comments/game/{id}	*GET*	*Comments*		
api/comments/quiz/{id}	*GET*	*Comments*		
api/comments/game/{id}	*POST*	*Comments*	11	Размещение комментариев к завершенной игре
api/comments/quiz/{id}	*POST*	*Comments*	20	Размещение комментариев к пройденной викторине
api/comments/{id}	*DELETE*	*Comments*	12	Удаление комментариев
api/questions	*POST*	*Questions*	7	Создание вопросов и вариантов ответов для конкретной игры
api/questions/{id }	*GET*	*Questions*		

Продолжение таблицы 3.3

Маршрут	Метод	Контроллер	№	Наименование функции
api/questions	*GET*	*Questions*		
api/questions/game/{id }	*GET*	*Questions*		
api/questions/{id }	*DELETE*	*Questions*		
api/quizzes	*GET*	*Quizzes*	2	Просмотр информации о доступных викторинах
api/quizzes/{id }	*GET*	*Quizzes*	16	Прохождение викторин
api/quizzes/rating	*GET*	*Quizzes*	4	Просмотр рейтингов других игроков по прохождению викторин
api/quizzes/{id }/start	*POST*	*Quizzes*	16	Прохождение викторин
api/quizzes/{id}/end	*POST*	*Quizzes*	17	Просмотр результатов прохождения викторин
api/game/create	*POST*	*Game*	13	Создание игровых комнат
api/game/{id }	*GET*	*Game*	15	Присоединение к играм
api/game/all	*GET*	*Game*	3	Просмотр информации о существующих играх
api/game/{id}/statistics	*GET*	*Game*	19	Просмотр результатов игр и рейтинга игроков в игровой комнате
api/game/{availability}	*GET*	*Game*		
api/game/{id }/start	*POST*	*Game*	8	Запуск игры
api/game/{id }/end	*POST*	*Game*	10	Завершение игры
api/notifications/{id}	*GET*	*Notifications*		
api/profile/{id}	*GET*	*Profile*	5	Просмотр профилей пользователей
api/profile/{id}/info	*GET*	*Profile*		
api/profile	*GET*	*Profile*	5	Просмотр профилей пользователей
api/profile/{id}	*PUT*	*Profile*	14	Редактирование личной информации
api/profile/{id}	*DELETE*	*Profile*		

Таким образом, слой представления играет ключевую роль в обеспечении взаимодействия конечных пользователей с приложением и предоставляет доступ к функциональности, скрывая при этом детали внутренней реализации и управляя

данными через контроллеры, которые обращаются к бизнес-логике через строго типизированные сервисы.

3.1.4 API Gateway

API Gateway – это *web*-приложение, реализованное на основе *Ocelot 23.3.3*, которое служит единой точкой доступа к микросервисам [25]. Его основное назначение заключается в упрощении взаимодействия клиентов с несколькими сервисами через один конечный пункт, что улучшает производительность и снижает задержки.

API Gateway также является паттерном проектирования, который позволяет централизовать обработку входящих запросов и распределять их между различными микросервисами [26].

Настройка *API Gateway* с *Ocelot* включает создание файла конфигурации *ocelot.json*, где определяются маршруты, по которым запросы перенаправляются на соответствующие микросервисы. В этом файле указываются параметры маршрутизации, такие как *DownstreamPathTemplate* и *UpstreamPathTemplate*. Для корректной работы необходимо зарегистрировать *Ocelot* в *Program.cs* и вызвать *UseOcelot* в методе у объекта *app*.

Управление маршрутизацией запросов позволяет гибко распределять трафик между микросервисами, основываясь на *HTTP* методах и правилах обработки. Важно также обеспечить безопасность, используя механизмы аутентификации, такие как *JWT*, что защищает доступ к сервисам.

Таким образом, *API Gateway* на основе *Ocelot* упрощает архитектуру взаимодействия с микросервисами и обеспечивает высокий уровень безопасности и управления трафиком.

3.1.5 Используемые библиотеки

Также, в рамках приложения, были реализованы две дополнительные библиотеки, которые облегчают разработку и повышают качество приложения: *EmailSenderLibrary* и *GlobalExceptionHandlerLibrary*.

GlobalExceptionHandlerLibrary реализует глобальный обработчик исключений, который используется во всех микросервиса. Эта библиотека представлена в виде расширения *Middleware*, что позволяет легко интегрировать её в существующие приложения. Глобальный обработчик обеспечивает централизованное управление ошибками, позволяя перехватывать и обрабатывать исключения на уровне приложения, а также возвращать понятные сообщения об ошибках клиентам. Это улучшает отладку и поддерживаемость приложения, а также обеспечивает более стабильную работу микросервисов.

EmailSenderLibrary представляет собой библиотеку для отправки электронных сообщений с использованием пакета *MimeKit* 4.7.1.1 [27]. Эта библиотека предоставляет удобный клиент, который упрощает процесс настройки и отправки *email*-уведомлений. Дополнительно для аутентификации используется пакет *JwtBearer 8.0.7*, который применяется в функциях генерации *access*- и *refresh*-токенов [28]. Для взаимодействия с *Redis* используется пакет *StackExchange.Redis*

2.8.16, обеспечивающий высокопроизводительное кэширование и хранение временных данных, таких как временные коды валидации и данные для аутентификации пользователей [29]. Для работы с паролями используется пакет *BCrypt.Net-Next 4.0.3*, который обеспечивает безопасное хэширование и проверку паролей, что повышает уровень безопасности приложения [30]. Для реализации *Api Gateway* используется пакет *Ocelot 23.3.3*.

3.2 Разработка клиентской части приложения

Клиентская часть приложения предоставляет интерфейс для взаимодействия с серверным приложением, используя современные технологии, которые делают взаимодействие более интуитивным и гибким. В качестве основного фронтенд-фреймворка мы используем *React* с *TypeScript*, который обеспечивает строгую типизацию и высокую производительность, что особенно важно для крупного и интерактивного приложения [31].

Shadcn UI 2.1.8 – это библиотека компонентов для *React*, ориентированная на гибкость и кастомизацию [32]. В отличие от традиционных *UI*-библиотек, *Shadcn UI* поставляется не в виде готовых компонентов, а в виде шаблонов, которые можно адаптировать под конкретные нужды проекта. Каждый компонент представляет собой настроенную функциональную и визуальную единицу, которая легко интегрируется и адаптируется к общему стилю приложения. Этот подход позволяет нам избежать чрезмерной зависимости от заранее заданных стилей, обеспечивая высокую степень контроля над интерфейсом и возможность внедрения уникального стиля.

Для стилизации мы используем *Tailwind CSS 3.4.9* – утилитарный *CSS*-фреймворк, который предоставляет набор готовых классов для оформления пользовательских интерфейсов [33]. *Tailwind* отличается от классических *CSS*-фреймворков тем, что использует утилитарные классы, позволяя описывать стиль компонента прямо в его коде. Этот подход делает код более структурированным и снижает количество дублирующегося *CSS*. *Tailwind* позволяет легко адаптировать дизайн, делая его отзывчивым и управляемым, а также улучшает производительность за счёт минимизации и оптимизации *CSS*.

React Router DOM 6.26.0 используется для управления маршрутизацией [34]. Он позволяет создавать одностраничные приложения, где навигация происходит плавно, без перезагрузки страницы. В приложении *React Router DOM* обеспечивает интуитивный переход между основными страницами и функциональными модулями, что значительно улучшает пользовательский опыт. Настройки маршрутизации указываются в файле *App.tsx*. Содержимое файла *App.tsx* представлено в приложении Б.

Для создания анимаций в интерфейсе мы используем библиотеку *AOS 2.3.4* (*Animate On Scroll*) [35]. *AOS* добавляет анимационные эффекты, которые активируются, когда элементы становятся видимыми на экране при прокрутке страницы.

Для создания пользовательского интерфейса было создано некоторое количество компонентов. В проекте *React* созданные компоненты делятся на два типа: компоненты для отображения страниц (название компонента заканчивается на

Page, хранятся в директории */pages/*), повторноиспользуемые компоненты (хранятся в директории */partials/*).

В таблице 3.4 представлено название и описания для всех компонентов, используемых для отображения страниц.

Таблица 3.4 – Компоненты клиентской части для отображения страниц

Название	Описание
AboutPage	Страница «О нас»
ForgotPasswordPage	Страница с формой восстановления пароля
GameFinishedPage	Страница с результатами для завершенной игры
GameListPage	Страница для отображения списка игр
GameRoomPage	Страница игровой комнаты
GameStartedPage	Страница для начавшейся игры
InternalServerErrorPage	Страница для ошибки со статусом 500
LoginPage	Страница с формой авторизации
MainPage	Главная страница
NotFoundPage	Страница для ошибки со статусом 404, отображается при переходе по несуществующему пути
NotifiactionsPage	Страница с уведомлениями пользователя
ProfilePage	Страница профиля пользователя
QuizFinishedPage	Страница для завершенной викторины
QuizRoomPage	Страница не пройденной викторины
QuizStartedPage	Страница для начавшейся викторины
QuizzesListPage	Страница со списком викторин
RegisterPage	Страница с формой регистрация
SettingsPage	Страница настроек аккаунта пользователя

В таблице 3.5 представлено название и описания для всех компонентов, используемых повторно.

Таблица 3.5 – Компоненты клиентской части для повторного использования

Название	Описание
Breadcrumbs	Хлебные крошки
Footer	Футер (подвал) – нижняя часть сайта
GameListItem	Представляет собой один элемент списка игр на странице списка игр
GamesListTabContent	Представляет собой одну вкладку со списком игр на странице списка игр
GameQuestionItem	Представляет собой один вопрос в игровой комнате в списке вопросов
QuizzesListTabContent	Представляет собой одну вкладку со списком викторин на странице списка викторин
Header	Хедер (шапка) – верхняя часть сайта
DialogCreateGame	Представляет собой модальное окно с формой создания игры

Продолжение таблицы 3.5

Название	Описание
DialogCreateQuestion	Представляет собой модальное окно с формой создания вопроса и вариантов ответа
DialogEditProfile	Представляет собой модальное окно с формой для редактирования профиля
DialogGamePassword	Представляет собой модальное окно с формой для ввода пароля при попытке входа в игровую комнату
DialogShareButton	Представляет собой модальное окно с ссылкой на текущую страницу

Совокупное использование этих технологий, библиотек и компонентов помогает нам создавать продуманный, современный интерфейс, который не только эстетически привлекателен, но и удобен в использовании, а также оптимизирован для различных устройств и экранов.

3.3 Конфигурация Nginx

В данном разделе рассмотрим конфигурацию *Nginx*, используемую для развертывания *React*-приложения и настройки проксирования запросов к серверу *API*, разработанному на платформе *ASP.NET Core Web API*. *Nginx* выполняет роль обратного прокси-сервера и обеспечивает безопасный доступ к приложению, а также обработку статических файлов.

Nginx выступает мощным инструментом для развертывания *web*-приложений, благодаря высокой производительности и гибкости в настройке. В текущей конфигурации *Nginx* выполняет следующие задачи:

– обслуживание статических файлов *React*-приложения, таких как *index.html* и другие ресурсы, скомпилированные при сборке;

– проксирование запросов, направленных на *API Gateway*, для маршрутизации запросов к соответствующим обработчикам на сервере *ASP.NET Core*;

– перенаправление *HTTP*-запросов на *HTTPS* для обеспечения безопасности соединения.

В листинге 3.6 представлено содержимое файла *nginx.conf*.

```
server {
    listen 80;
    server_name localhost;

    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name reactapp;
    ssl_certificate /etc/nginx/certs/LAB.crt;
    ssl_certificate_key /etc/nginx/certs/LAB.key;
    root /usr/share/nginx/html;
    index index.html;
```

```
    location / {
        try_files $uri /index.html;
    }
    location /apigateway/ {
        proxy_pass https://apigateway:443/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
    location /signalr/ {
        proxy_pass https://apigateway:443/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
    error_page 404 /index.html;
}
```

Листинг 3.6 – Файл конфигурации *nginx.conf*

Выполнив команды, представленные в листинге 3.7, получаем ключи и самоподписанные сертификаты, используемые как на клиенте *React* (при запуске локально), так и при развертывании при помощи *Nginx*.

```
openssl req -new -newkey rsa:2048 -keyout
certificates/reactapp/key.pem -out
certificates/reactapp/certificate.csr -subj "/CN=reactapp" -addext
"subjectAltName=DNS:reactapp"
openssl x509 -req -in certificates/reactapp/certificate.csr -CA
certificates/ca/ca.crt -CAkey certificates/ca/ca.key -CAcreateserial
-out certificates/reactapp/certificate_signed.pem -days 365 -sha256
openssl pkcs12 -export -out certificates/reactapp/certificate.pfx -
inkey certificates/reactapp/key.pem -in
certificates/reactapp/certificate_signed.pem -passout pass:password
```

Листинг 3.7 – Скрипт генерации ключей безопасности

3.4 Реализация функций

3.4.1 Регистрация

Функция «Регистрация», представленная номером 1, реализована в методе *RegisterUserAsync* сервиса регистрации *RegisterService*. Сначала происходит валидация данных, переданных в *RegisterDto*, с использованием *_registerValidator*.

В случае ошибок валидации выбрасывается исключение *ValidationException*. Содержимое класса *RegisterDto* представлено в листинге 3.8

```
public class RegisterDto{
   public string Username { get; set; } = null!;
   public string Email { get; set; } = null!;
   public string Password { get; set; } = null!;
   public string RegistrationCode { get; set; } = null!;
}
```

Листинг 3.8 – Содержимое класса *RegisterDto*

Далее создаётся новый пользователь. Его пароль хэшируется с использованием *BCrypt.Net.BCrypt*, а остальные поля, включая статус подтверждения *email* и активности, заполняются согласно переданным данным. Созданный пользователь сохраняется в базе данных через *_userRepository*. Код функции *RegisterUserAsync* представлен в листинге 3.9.

```
public async Task RegisterUserAsync(RegisterDto registerDto,
CancellationToken = default) {
     var validationResult = await
_registerValidator.ValidateAsync(registerDto, cancellationToken);
     if (!validationResult.IsValid)
         throw new ValidationException(validationResult.Errors);
     var user = new User(registerDto.Email, registerDto.Username,
BCrypt.Net.BCrypt.HashPassword(registerDto.Password), true, true);
     await _userRepository.CreateAsync(user, cancellationToken);
     await _notificationService.CreateNotificationForUserAsync(new
CreateNotificationDto("Welcome to Bellini", "Registration
notification",
_userRepository.GetElementsAsync().Result.FirstOrDefault(x =>
x.Email == user.Email).Id));
 }
```

Листинг 3.9 – Код функции *RegisterUserAsync*

После успешной регистрации пользователю отправляется уведомление через *_notificationService*, содержащее приветственное сообщение и информацию о завершении регистрации. Этот процесс обеспечивает безопасное создание пользователей.

3.4.2 Просмотр информации о доступных викторинах

Функция «Просмотр информации о доступных викторинах», представленная номером 2, реализована в методе *GetAllQuizzesAsync*, который отвечает за предоставление списка доступных викторин с поддержкой пагинации. Сначала происходит получение всех викторин из репозитория *_quizRepository*. После этого определяется общее количество викторин для возвращения в ответе.

Для реализации пагинации используется смещение (*offset*) и ограничение на количество записей (*limit*). Из полного списка викторин отбираются только те

записи, которые соответствуют указанным параметрам, с помощью методов *Skip* и *Take*.

Каждая викторина преобразуется в объект *QuizDto*, включающий основные данные, такие как идентификатор, название, время начала и окончания, *URL* изображения, количество вопросов, а также статус прохождения викторины пользователем. Статус прохождения определяется путём проверки наличия результатов пользователя (*QuizResults*) для конкретной викторины.

Код функции *GetAllQuizzesAsync* представлен в листинге 3.10.

```
public async Task<(IEnumerable<QuizDto> Quizzes, int TotalCount)>
GetAllQuizzesAsync(int limit, int offset, int userId,
CancellationToken = default) {
    var allQuizzes = await
_quizRepository.GetElementsAsync(cancellationToken);
    var totalCount = allQuizzes.Count();

    var paginatedQuizzes =
allQuizzes.Skip(offset).Take(limit).Select(q => new QuizDto {
            Id = q.Id,
            GameName = q.GameName,
            StartTime = q.StartTime,
            EndTime = q.EndTime,
            GameCoverImageUrl = q.GameCoverImageUrl,
            NumberOfQuestions = q.Questions.Count,
            HasUserCompleted = userId != 0 && q.QuizResults.Any(qr
=> qr.UserId == userId)
        }).ToList();
    return (paginatedQuizzes, totalCount);
}
```

Листинг 3.10 – Код функции *GetAllQuizzesAsync*

Метод возвращает список отфильтрованных и преобразованных викторин вместе с их общим количеством. Такой подход обеспечивает удобный и эффективный доступ к викторинам с учётом их состояния для конкретного пользователя.

3.4.3 Просмотр информации о существующих играх

Функция «Просмотр информации о существующих играх», представленная номером 3, реализована в методе *GetAllActiveGamesAsync*. Сначала из репозитория *_gameRepository* извлекаются все записи, после чего происходит фильтрация по статусу игры. Выбираются только те игры, у которых статус равен «*Not started*».

Для реализации пагинации используется параметризация с помощью *limit* и *offset*. Из отфильтрованного списка игр выбирается подмножество записей, соответствующее указанным ограничениям, с помощью методов *Skip* и *Take*.

Каждая игра преобразуется в объект *GameDto*, который содержит ключевую информацию: идентификатор, название, идентификатор хоста, время начала,

максимальное количество игроков, статус игры, признак приватности, пароль комнаты, и URL изображения игры.

Код функции *GetAllActiveGamesAsync* представлен в листинге 3.11.

```
public async Task<(IEnumerable<GameDto> Games, int TotalCount)>
GetAllActiveGamesAsync(int limit, int offset, CancellationToken =
default) {
    var allGames = await
_gameRepository.GetElementsAsync(cancellationToken);
    var filteredGames = allGames.Where(g =>
g.Status.Name.Equals("Not started",
StringComparison.OrdinalIgnoreCase));
    var totalCount = filteredGames.Count();
    var paginatedGames =
filteredGames.Skip(offset).Take(limit).Select(g => new GameDto {
            Id = g.Id, GameName = g.GameName,
            HostId = g.HostId,
            StartTime = g.StartTime,
            MaxPlayers = g.MaxPlayers,
            GameStatus = g.Status, IsPrivate = g.IsPrivate,
            RoomPassword = g.RoomPassword,
            GameCoverImageUrl = g.GameCoverImageUrl
        }).ToList();
    return (paginatedGames, totalCount);
}
```

Листинг 3.11 – Код функции *GetAllActiveGamesAsync*

Метод возвращает список игр, которые ещё не начались, вместе с общим их количеством, что позволяет эффективно отображать доступные игры пользователям приложения.

3.4.4 Просмотр рейтингов других игроков по прохождению викторин

Функция «Просмотр рейтингов других игроков по прохождению викторин», представленная номером 4, реализована в методе *GetQuizRatingAsync*. Данные о результатах игр извлекаются из репозитория *_quizResultsRepository*.

Сначала результаты группируются по пользователям, где для каждой группы рассчитываются: общее количество правильных ответов, общее количество вопросов, и время завершения последней викторины. Затем данные сортируются по количеству правильных ответов в убывающем порядке, а при равенстве результатов – по времени завершения. Для отображения берутся только 10 лучших игроков.

Каждый игрок преобразуется в объект *QuizRatingDto*, содержащий позицию в рейтинге (ранг), имя пользователя, *email*, количество правильных ответов, общее количество вопросов, точность (в процентах), а также время завершения последней викторины.

Код функции *GetQuizRatingAsync* представлен в листинге 3.12.

```
public async Task<List<QuizRatingDto>>
GetQuizRatingAsync(CancellationToken cancellationToken = default) {
```

```
    var quizResults = await
_quizResultsRepository.GetElementsAsync(cancellationToken);
    var topPlayers = quizResults.GroupBy(qr => qr.User).Select(group
=> new {
            User = group.Key,
            TotalCorrectAnswers = group.Sum(qr =>
qr.NumberOfCorrectAnswers),
            TotalQuestions = group.Sum(qr => qr.NumberOfQuestions),
            LastEndTime = group.Max(qr => qr.EndTime)
        })
        .OrderByDescending(x => x.TotalCorrectAnswers).ThenBy(x =>
x.LastEndTime).Take(10).ToList();

    var result = topPlayers.Select((player, index) => new
QuizRatingDto(index + 1, player.User.Username, player.User.Email,
player.TotalCorrectAnswers, player.TotalQuestions,
player.TotalQuestions > 0 ?
Math.Round((double)player.TotalCorrectAnswers /
player.TotalQuestions * 100, 2) : 0, player.LastEndTime)).ToList();

    return result;
}
```

Листинг 3.12 – Код функции *GetQuizRatingAsync*

Результатом работы метода является список из топ-10 игроков, отсортированный по их достижениям, что позволяет пользователям сравнить свои успехи с другими.

3.4.5 Просмотр профилей пользователей

Функция «Просмотр профилей пользователей», представленная номером 5, реализована в методе *GetUserByIdAsync*, который обращается к репозиторию пользователей, извлекает данные о пользователе и преобразует их в объект типа *ProfileDto*.

Код функции *GetUserByIdAsync* представлен в листинге 3.13.

```
public async Task<List<QuizRatingDto>>
GetQuizRatingAsync(CancellationToken cancellationToken = default) {
    var quizResults = await
_quizResultsRepository.GetElementsAsync(cancellationToken);
    var topPlayers = quizResults.GroupBy(qr => qr.User).Select(group
=> new {User = group.Key, TotalCorrectAnswers = group.Sum(qr =>
qr.NumberOfCorrectAnswers), TotalQuestions = group.Sum(qr =>
qr.NumberOfQuestions), LastEndTime = group.Max(qr => qr.EndTime)})
        .OrderByDescending(x => x.TotalCorrectAnswers).ThenBy(x =>
x.LastEndTime).Take(10).ToList();
    var result = topPlayers.Select((player, index) => new
QuizRatingDto(index + 1, player.User.Username, player.User.Email,
player.TotalCorrectAnswers, player.TotalQuestions,
player.TotalQuestions > 0 ?
```

```
Math.Round((double)player.TotalCorrectAnswers /
player.TotalQuestions * 100, 2) : 0, player.LastEndTime)).ToList();
    return result;
}
```

Листинг 3.13 – Код функции *GetUserByIdAsync*

Метод начинается с вызова функции *GetItemAsync* из репозитория, которая принимает идентификатор профиля и токен отмены, что позволяет корректно обрабатывать отмену запроса в случае необходимости. Если профиль с заданным идентификатором не найден, выбрасывается исключение *NotFoundException*, что информирует систему о невозможности найти пользователя с указанным *ID*.

Если пользователь найден, его данные преобразуются в объект типа *ProfileDto* с помощью маппера. Это позволяет отделить слой данных от представления, обеспечив гибкость при изменении структуры хранения данных. Преобразованный объект *ProfileDto* возвращается в качестве результата работы метода, что делает возможным дальнейшее использование данных профиля в приложении.

Данная реализация поддерживает асинхронность, что позволяет не блокировать выполнение приложения при запросах, улучшая производительность и пользовательский опыт.

3.4.6 Авторизация

Функция «Авторизация», представленная под номером 6, реализована в методе *AuthenticateAsync* сервиса аутентификации *LoginService*. Валидация входных данных выполняется с помощью валидатора *_loginValidator*, который проверяет корректность переданных *email* и *password*. В случае ошибок выбрасывается *ValidationException*. Содержимое класса *LoginDto* представлено в листинге 3.14.

```
public class LoginDto {
    public string Email { get; set; } = null!;
    public string Password { get; set; } = null!;
}
```

Листинг 3.14 – Содержимое класса *RegisterDto*

Для оптимизации используется кэш: сначала приложение пытается получить данные пользователя из кэша по ключу *User_email*}. Если данные отсутствуют, выполняется запрос к базе данных через *_userRepository*, после чего пользовательские данные кэшируются на 30 минут.

Далее происходит проверка существования пользователя и верификация пароля с использованием *BCrypt.Net.BCrypt.Verify*. При несовпадении данных выбрасывается исключение о неверных учётных данных.

Код функции *AuthenticateAsync* представлен в листинге 3.15.

```
public async Task<TokenDto> AuthenticateAsync(LoginDto loginDto,
CancellationToken = default) {
    var validationResult = await
_loginValidator.ValidateAsync(loginDto, cancellationToken);
```

```
    if (!validationResult.IsValid) throw new
ValidationException(validationResult.Errors);
    var cachedUser = await
_cache.GetStringAsync($"User_{loginDto.Email}", cancellationToken);
    User? user;
    if (string.IsNullOrEmpty(cachedUser)) {
        user = (await
_userRepository.GetElementsAsync(cancellationToken)).FirstOrDefault(
u => u.Email == loginDto.Email);
        if (user is not null) {
            await _cache.SetStringAsync(cacheKey,
JsonConvert.SerializeObject(user), new DistributedCacheEntryOptions
{
                AbsoluteExpirationRelativeToNow =
TimeSpan.FromMinutes(30)
            }, cancellationToken);
        }
    }
    else
     user = JsonConvert.DeserializeObject<User>(cachedUser);
    if (user is null || !BCrypt.Net.BCrypt.Verify(loginDto.Password,
user.Password))
        throw new ValidationException("Invalid username or
password.");
    await _notificationService.CreateNotificationForUserAsync(new
CreateNotificationDto("Account", "Your account has been successfully
logged in.", user.Id));
    return new TokenDto(    GenerateAccessToken(user),
GenerateRefreshToken(user), user.Username, user.Id);
}
```

Листинг 3.15 – Код функции *AuthenticateAsync*

При успешной аутентификации создаётся уведомление для пользователя через *_notificationService*. Затем генерируются *AccessToken* и *RefreshToken*, а также возвращаются *Username* и *UserId*. Такой подход обеспечивает безопасность, производительность и удобство работы приложения.

3.4.7 Создание вопросов и вариантов ответов для конкретной игры

Функция «Создание вопросов и вариантов ответов для конкретной игры», представленная под номером 7, реализована в методе *CreateQuestionAsync*, который добавляет новый вопрос с вариантами ответов в указанную игру.

Сначала выполняется проверка количества правильных ответов в переданном DTO. Если правильных ответов не ровно один, выбрасывается исключение *IncorrectNumberOfAnswersException*. Затем проверяется существование игры по переданному идентификатору. Если игра не найдена, выбрасывается исключение *NotFoundException*. Содержимое класса *CreateQuestionDto* представлено в листинге 3.16

```
public class CreateQuestionDto {
```

```
    public string Text { get; set; } = null!;
    public int GameId { get; set; }
    public IEnumerable<CreateAnswerDto> Answers { get;set; } = new();
}
```

Листинг 3.16 – Содержимое класса *CreateQuestionDto*

Создание вопроса осуществляется с использованием маппера, который преобразует данные из *DTO* в модель *Question*. Поле *IsCustom* устанавливается в *true*, чтобы отметить, что вопрос создан вручную. После сохранения вопроса в репозитории добавляются варианты ответов, переданные в *DTO*. Каждый вариант ответа маппируется в модель *AnswerOption*, связывается с идентификатором вопроса и сохраняется в соответствующем репозитории.

Код функции *CreateQuestionAsync* представлен в листинге 3.17.

```
 public async Task<int> CreateQuestionAsync(CreateQuestionDto
createQuestionDto, CancellationToken cancellationToken = default) {
    var correctAnswers = createQuestionDto.Answers.Count(a =>
a.IsCorrect);
    if (correctAnswers != 1) throw new
IncorrectNumberOfAnswersException("Each question must have exactly
one correct answer.");

    var game = await
_gameRepository.GetItemAsync(createQuestionDto.GameId,
cancellationToken);
    if (game is null) throw new NotFoundException("Game not
found.");

    var question = _mapper.Map<Question>(createQuestionDto);
    question.IsCustom = true;
    await _questionRepository.CreateAsync(question,
cancellationToken);

    foreach (var answerDto in createQuestionDto.Answers) {
        var answer = _mapper.Map<AnswerOption>(answerDto);
        answer.QuestionId = question.Id;
        await _answerRepository.CreateAsync(answer,
cancellationToken);
    }

    game.Questions.Add(question);
    await _gameRepository.UpdateAsync(game.Id, game,
cancellationToken);
    return question.Id;
}
```

Листинг 3.17 – Код функции *CreateQuestionAsync*

В завершение, вопрос добавляется в коллекцию вопросов игры, и обновленные данные игры сохраняются. Возвращается идентификатор созданного вопроса, подтверждающий успешное выполнение операции.

3.4.8 Запуск игры

Функция «Запуск игры», представленная под номером 8, реализована в методе *StartGame*, который подготавливает игру к началу, изменяет ее статус, уведомляет участников и возвращает данные о запущенной игре в виде объекта *StartedGameDto*. Содержимое класса *StartedGameDto* представлено в листинге 3.18.

```
public class StartedGameDto {
   public int Id { get; set; }
   public string GameName { get; set; } = null!;
   public int HostId { get; set; }
   public DateTime CreateTime { get; set; } = DateTime.Now;
   public DateTime StartTime { get; set; } = DateTime.MinValue;
   public int MaxPlayers { get; set; }
   public string GameCoverImageUrl { get; set; } = null!;
   public List<Player> Players { get; set; } = null!;
   public List<Question> Questions { get; set; } = null!;
}
```

Листинг 3.18 – Содержимое класса *RegisterDto*

На начальном этапе метод проверяет наличие игры в базе данных через репозиторий *_gameRepository*. Если игра не найдена, выбрасывается исключение *NotFoundException*. Также проводится проверка наличия вопросов в игре. Если вопросы отсутствуют или их количество меньше трех, выбрасывается исключение *NotFoundGameQuestionsException*. Далее происходит извлечение списка игроков, которые зарегистрировались на игру.

Код функции *StartGame* представлен в листинге 3.19.

```
 public async Task<StartedGameDto> StartGame(int gameId,
StartGameDto startGameDto, CancellationToken cancellationToken =
default) {
     var game = await _gameRepository.GetItemAsync(gameId,
cancellationToken);
     if (game is null) throw new NotFoundException($"Game with ID
{gameId} not found.");
     if (game.Questions is null || !game.Questions.Any()) throw new
NotFoundGameQuestionsException("Cannot start a game without any
questions.");
     if (game.Questions.Count < 3) throw new
NotFoundGameQuestionsException("There must be at least 3 questions
to start the game.");

     var gameStatus = await
_gameStatusRepository.GetElementsAsync(cancellationToken);
     var inProcessingStatus = gameStatus.FirstOrDefault(s =>
s.Name.Equals("In process", StringComparison.OrdinalIgnoreCase));

     if (inProcessingStatus is null) throw new
InvalidOperationException("No status found for StatusName 'Not
started'");
```

```
     var db = _redis.GetDatabase();
     var playersInGame = await
db.ListRangeAsync($"game:{gameId}:players");
     var playerList = playersInGame.Select(p =>
JsonSerializer.Deserialize<PlayerDto>(p)).ToList();

     game.Players = playerList.Select(p => new
Player(int.Parse(p.UserId), p.Username, p.GameId,)).ToList();
     game.GameStatusId = inProcessingStatus.Id;
     game.StartTime = DateTime.Now;

     await _gameRepository.UpdateAsync(gameId, game,
cancellationToken);
     var result = new StartedGameDto(game);
     await _gameHub.Clients.All.SendAsync("GameStarted", result,
cancellationToken);

     await _notificationService.CreateNotificationForUserAsync(new
CreateNotificationDto($"Your game {game.GameName} has been
successfully launched. The number of players at the start is
{startGameDto.Players.Count}.", "Game started",
startGameDto.HostId));

     return result;
 }
```

Листинг 3.19 – Код функции StartGame

Метод отправляет уведомления всем участникам игры через хаб *_gameHub*, используя событие *GameStarted*. Кроме того, создается уведомление для организатора игры с информацией о запуске и количестве участников.

Метод возвращает объект *StartedGameDto*, содержащий сведения о текущем состоянии игры. Таким образом, реализуется полный цикл подготовки и запуска игры.

3.4.9 Переход к следующему вопросу

Функция «Переход к следующему вопросу», представленная под номером 9, реализована в методе *NextQuestion*, который обеспечивает передачу информации о следующем вопросе всем подключенным клиентам в игровой комнате.

Метод принимает два параметра:

– *gameId* – уникальный идентификатор игры, используемый для группировки клиентов в хабе;

– *nextQuestionIndex* – индекс следующего вопроса, который будет передан участникам игры.

Реализация метода состоит из одного вызова метода *SendAsync* у объекта *Clients.Group(gameId)*. Этот вызов отправляет сообщение с событием «*NextQuestion*» и передает индекс следующего вопроса всем клиентам, присоединенным к группе, связанной с указанной игрой.

Код функции *NextQuestion* представлен в листинге 3.20.

```
public async Task NextQuestion(string gameId,int nextQuestionIndex){
    await Clients.Group(gameId).SendAsync("NextQuestion",
nextQuestionIndex);
}
```

Листинг 3.20 – Код функции *NextQuestion*

Таким образом, данная функция выполняет задачу передачи информации о переходе к следующему вопросу, обеспечивая синхронизацию между сервером и участниками игры в режиме реального времени.

3.4.10 Завершение игры

Функция «Завершение игры», представленная под номером 10, реализована с помощью асинхронного метода *CompleteGameAsync*, который принимает идентификатор игры и отменяет выполнение операции при необходимости с использованием *CancellationToken*.

Процесс завершения игры начинается с извлечения информации об игре из репозитория *_gameRepository*. Если игра не найдена, выбрасывается исключение *NotFoundException*. Далее, проверяется статус игры – игра должна находиться в процессе, чтобы ее можно было завершить. Если статус игры не соответствует ожидаемому, генерируется исключение *InvalidOperationException*.

Затем из репозитория *_gameStatusRepository* извлекаются все возможные статусы игры, и находится статус «Завершена». Если такой статус не найден, выбрасывается исключение.

Для расчета результатов игры, извлекаются ответы игроков из *Redis*. Используется ключевой паттерн для поиска всех записей с ответами игроков, соответствующих текущей игре. Каждому ключу соответствует ответ игрока, который десериализуется в список объектов *AnswerSubmitedDto*. Далее выполняется проверка правильности каждого ответа, и подсчитывается количество правильных ответов.

После подсчета результатов для каждого игрока создаются объекты GameResults, которые сохраняются в репозитории через *_gameResultsRepository*.

Когда все результаты игроков сохранены, игра помечается как завершенная. Для этого в объекте игры обновляется поле *GameStatusId*, и указывается время завершения игры. После этого игра обновляется в базе данных.

Для уведомления клиентов, участвующих в игре, используется *Hub SignalR*, и всем пользователям, присоединившимся к игре, отправляется сообщение о завершении игры.

Код метода *CompleteGameAsync* представлен на листинге 3.21.

```
public async Task CompleteGameAsync(int gameId, CancellationToken
cancellationToken = default) {
     var db = _redis.GetDatabase();
     var game = await _gameRepository.GetItemAsync(gameId,
cancellationToken);
     if (game is null) throw new NotFoundException($"Game with ID
{gameId} not found.");
```

```
     var gameStatuses = await
_gameStatusRepository.GetElementsAsync(cancellationToken);
     var completedStatus = gameStatuses.FirstOrDefault(s =>
s.Name.Equals("Completed", StringComparison.OrdinalIgnoreCase));
     if (completedStatus is null) throw new
InvalidOperationException("Status 'Completed' not found in
database.");
     string answersPattern = $"running:game:{gameId}:answers:*";
     var server = _redis.GetServer(_redis.GetEndPoints().First());
     var keys = server.Keys(pattern: answersPattern);
     var gameResultsList = new List<GameResults>();
     foreach (var key in keys) {
         var userAnswersJson = await db.StringGetAsync(key);
         if (userAnswersJson.HasValue) {
             var userAnswers =
JsonSerializer.Deserialize<List<AnswerSubmitedDto>>(userAnswersJson)
           var userId = int.Parse(key.ToString().Split(':').Last());
           int correctAnswersCount = 0;
             foreach (var answer in userAnswers) {
                 if (answer is not null) {
                     var isCorrect = game.Questions.FirstOrDefault(q
=> q.Id == answer.QuestionId)?.AnswerOptions.FirstOrDefault(opt =>
opt.Id == answer.AnswerId)?.IsCorrect ?? false;
                     if (isCorrect) correctAnswersCount++;
                 }
             }
             var gameResults = new GameResults(gameId, userId,
game.Questions.Count, correctAnswersCount);
             gameResultsList.Add(gameResults);
         }
     }
     foreach (var gameResult in gameResultsList)
         await _gameResultsRepository.CreateAsync(gameResult,
cancellationToken);
     game.GameStatusId = completedStatus.Id;
     game.EndTime = DateTime.Now;
     await _gameRepository.UpdateAsync(gameId, game,
cancellationToken);
     await
_gameHub.Clients.Group(gameId.ToString()).SendAsync("GameCompleted",
gameId, game);
     await _notificationService.CreateNotificationForUserAsync(new
CreateNotificationDto($"Your game {game.GameName} has been
successfully finished.", "Game finished", game.HostId));
 }
```

Листинг 3.21 – Код функции *CompleteGameAsync*

Кроме того, организатор игры получает уведомление о завершении игры с помощью сервиса уведомлений, который создает сообщение для пользователя-организатора.

Таким образом, этот метод завершает игру, сохраняет результаты, уведомляет участников и организатора, а также обновляет статус игры в базе данных.

3.4.11 Размещение комментариев к завершенной игре

Функция «Размещение комментариев к завершенной игре», представленная под номером 11, реализована в асинхронном методе *CreateGameCommentAsync*. Этот метод позволяет пользователям оставлять комментарии к играм, которые уже завершены, предоставляя информацию о пользователе и содержимом комментария.

Метод начинается с запроса игры по идентификатору через репозиторий игр. Если игра с заданным *gameId* не найдена, выбрасывается исключение *NotFoundException*, что сигнализирует о проблеме с указанием некорректного идентификатора игры.

После того как игра найдена, создается новый объект комментария *GameComment*, который включает в себя данные о соответствующей игре, пользователе, а также текст комментария и *URL* изображения профиля пользователя. Эти данные передаются через объект *CreateGameCommentDto*, который содержит информацию о пользователе, который оставляет комментарий.

Содержимое класса *CreateGameCommentDto* представлено в листинге 3.22

```
public class CreateGameCommentDto {
   public int GameId { get; set; }
   public int UserId { get; set; }
   public string Content { get; set; } = null!;
   public string Username { get; set; } = null!;
   public string ProfileImageUrl { get; set; } = null!;
   public DateTime CommentDate { get; set; }
}
```

Листинг 3.22 – Содержимое класса *CreateGameCommentDto*

Новый комментарий сохраняется в базе данных с помощью репозитория комментариев. По завершении операции метод возвращает идентификатор созданного комментария, который может быть использован для дальнейшей работы с комментариями в системе.

Код метода *CreateGameCommentAsync* представлен на листинге 3.23.

```
public async Task<int> CreateGameCommentAsync(int gameId,
CreateGameCommentDto createCommentDto, CancellationToken
cancellationToken = default) {
    var game = await _gameRepository.GetItemAsync(gameId,
cancellationToken);
    if (game is null) throw new NotFoundException($"Game with ID
{gameId} not found.");

    var comment = new GameComment
    (game.Id, createCommentDto.UserId, createCommentDto.Content,
createCommentDto.Username, createCommentDto.ProfileImageUrl);
    await _gameCommentRepository.CreateAsync(comment,
cancellationToken);
    return comment.Id;
}
```

Листинг 3.23 – Код функции *CreateGameCommentAsync*

Таким образом, данная реализация позволяет пользователям взаимодействовать с играми, оставляя свои комментарии и отзывы после завершения игрового процесса.

3.4.12 Удаление комментариев

Функция «Удаление комментариев», представленная под номером 12, реализована с использованием асинхронного метода *DeleteCommentAsync*. Этот метод принимает идентификатор комментария и объект *CancellationToken,* позволяющий отменить выполнение операции.

В рамках выполнения метода вызывается репозиторий *_gameCommentRepository*, который отвечает за операции с комментариями. Для удаления комментария используется метод *DeleteAsync*, которому передается идентификатор удаляемого комментария. После успешного выполнения операции метод возвращает идентификатор удаленного комментария.

Код метода *DeleteCommentAsync* представлен на листинге 3.24.

```
public async Task<int> DeleteCommentAsync(int commentId,
CancellationToken cancellationToken = default) {
    await _gameCommentRepository.DeleteAsync(commentId,
cancellationToken);
    return commentId;
}
```

Листинг 3.24 – Код функции *DeleteCommentAsync*

Таким образом, метод обеспечивает простое и эффективное удаление комментариев из базы данных с использованием стандартных подходов работы с репозиториями.

3.4.13 Создание игровых комнат

Функция «Создание игровых комнат», представленная под номером 13, реализована в методе *CreateGameRoomAsync*, который позволяет пользователям создавать новые игровые комнаты с заданными параметрами. Метод принимает на вход объект *CreateGameRoomDto*, содержащий необходимые данные для создания комнаты, такие как имя игры, идентификатор хозяина, максимальное количество игроков, пароль для доступа, а также флаг приватности комнаты.

Содержимое класса *CreateGameRoomDto* представлен на листинге 3.25.

```
public class CreateGameRoomDto {
   public string GameName { get; set; } = null!;
   public int HostId { get; set; }
   public DateTime CreateTime { get; set; } = DateTime.Now;
   public DateTime? StartTime { get; set; }
   public int MaxPlayers { get; set; }
   public string DifficultyLevel { get; set; } = null!;
```

```
    public bool IsPrivate { get; set; } = false;
    public string RoomPassword { get; set; } = "";
}
```

Листинг 3.25 – Содержимое класса *CreateGameRoomDto*

Код функции *CreateGameRoomAsync* представлен в листинге 3.26.

```
public async Task<int> CreateGameRoomAsync(CreateGameRoomDto
createGameRoomDto, CancellationToken cancellationToken = default) {
     string randomCoverUrl = $"https://localhost:7292/covers/{new
Random().Next(1, 11)}.jpg";
     var gameStatus = await
_gameStatusRepository.GetElementsAsync(cancellationToken);
     var notStartedStatus = gameStatus.FirstOrDefault(s =>
s.Name.Equals("Not started", StringComparison.OrdinalIgnoreCase));

     if (notStartedStatus is null) throw new
InvalidOperationException("Статус 'Not started' не найден в базе
данных.");

     var game = new Game(createGameRoomDto.GameName,
createGameRoomDto.HostId, createGameRoomDto.MaxPlayers,
randomCoverUrl, notStartedStatus.Id, DateTime.Now,
createGameRoomDto.RoomPassword, createGameRoomDto.IsPrivate);
     await _gameRepository.CreateAsync(game, cancellationToken);
     await _gameHub.Clients.All.SendAsync("GameCreated",
game.GameName, cancellationToken);
     return game.Id;
 }
```

Листинг 3.26 – Код функции *CreateGameRoomAsync*

После этого создается новый объект игры *Game*, в котором устанавливаются все необходимые параметры, включая имя игры, идентификатор хозяина, максимальное количество игроков, *URL* обложки, а также пароль и статус приватности. Новый объект игры сохраняется в базе данных с помощью репозитория игр.

Для оповещения всех подключенных клиентов о создании новой игровой комнаты, вызывается метод *SendAsync* на объекте хаба, который отправляет сообщение всем клиентам с уведомлением о создании комнаты.

В завершение работы метода возвращается идентификатор созданной игры, что позволяет использовать его для дальнейших операций с комнатой.

3.4.14 Редактирование личной информации

Функция «Редактирование личной информации», представленная под номером 14, реализована в методе *UpdateProfileAsync* сервиса профилей *ProfileService*. Сначала переданные данные в *UpdateProfileDto* проверяются валидатором *_updateProfileValidator*. Если данные не проходят проверку,

выбрасывается исключение *ValidationException*. Содержимое класса *UpdateProfileDto* представлено в листинге 3.27.

```
public class UpdateProfileDto {
    public string? FirstName { get; set; }
    public string? LastName { get; set; }
    public DateTime? DateOfBirth { get; set; }
    public string? ProfileImageUrl { get; set; }
}
```

Листинг 3.27 – Содержимое класса *UpdateProfileDto*

Далее приложение обращается к репозиторию *_userRepository* для получения информации о пользователе с указанным *profileId*. Если пользователь не найден, генерируется исключение *NotFoundException*.

После успешного получения данных они обновляются с помощью маппинга: новый объект пользователя создаётся на основе переданных данных и существующего профиля. Обновлённая информация сохраняется в базе данных через вызов метода *UpdateAsync*.

После обновления приложение повторно запрашивает обновлённые данные из репозитория для проверки их корректности. Если профиль всё ещё не найден, выбрасывается исключение.

Код функции *UpdateProfileAsync* представлен в листинге 3.28.

```
public async Task<ProfileDto> UpdateProfileAsync(int profileId,
UpdateProfileDto updateProfileDto, CancellationToken
cancellationToken = default) {
    var validationResult = await
_updateProfileValidator.ValidateAsync(updateProfileDto,
cancellationToken);
    if (!validationResult.IsValid) throw new
ValidationException(validationResult.Errors);

    var existingUser = await _userRepository.GetItemAsync(profileId,
cancellationToken);
    if (existingUser is null) throw new NotFoundException($"Profile
with ID {profileId} not found.");
    var user = _mapper.Map(updateProfileDto, existingUser);
    await _userRepository.UpdateAsync(profileId, user,
cancellationToken);

    var updatedUser = await _userRepository.GetItemAsync(profileId,
cancellationToken);

    if (updatedUser is null) throw new NotFoundException($"Profile
with ID {profileId} not found.");
    await _notificationService.CreateNotificationForUserAsync(new
CreateNotificationDto("Your profile has been successfully updated",
"Profile update", profileId));

    return _mapper.Map<ProfileDto>(updatedUser);
}
```

Листинг 3.28 – Код функции *UpdateProfileAsync*

В завершение пользователю отправляется уведомление через *_notificationService* с информацией о том, что его профиль успешно обновлён. Возвращаемое значение преобразуется в формат *ProfileDto*, предоставляя клиенту актуальную информацию. Такой подход обеспечивает безопасное и корректное редактирование профиля пользователя.

3.4.15 Присоединение к играм

Функция «Присоединение к играм», представленная под номером 15, реализована в методе *JoinGame*, который отвечает за добавление новых игроков в игровую комнату, управление их данными и синхронизацию с другими участниками. В начале выполняется подключение к *Redis* для работы с данными игроков, хранящимися под ключом, который формируется на основе идентификатора игры. Список текущих игроков извлекается и десериализуется из *Redis*. Если пользователь с указанным *UserId* уже находится в игре, метод завершает выполнение. Функция принимает объект *JoinGameDto*. Содержимое класса *JoinGameDto* представлено в листинге 3.29.

```
public class JoinGameDto {
   public string GameId { get; set; } = null!;
   public string UserId { get; set; } = null!;
   public string Username { get; set; } = null!;
   public string Email { get; set; } = null!;
   public string ProfileImageUrl { get; set; } = null!;
}
```

Листинг 3. 29 – Содержимое класса *JoinGameDto*

Если игрок отсутствует, создается объект *PlayerDto* с информацией о пользователе, который затем сериализуется и добавляется в список игроков в *Redis*. После обновления списка все участники уведомляются о новом подключении. Для этого используется механизм *SignalR*, где соединение пользователя добавляется в группу, связанную с игрой, а через события «*GetPlayers*» и «*PlayerJoined*» информация передается всем клиентам и участникам игровой комнаты.

Код функции *JoinGame* представлен в листинге 3.30.

```
 public async Task JoinGame(JoinGameDto joinGameDto) {
    var db = _redis.GetDatabase();
    var playersInGame = await
db.ListRangeAsync($"game:{joinGameDto.GameId}:players");
    var existingPlayers = playersInGame.Select(p =>
JsonSerializer.Deserialize<PlayerDto>(p)).ToList();
    if (existingPlayers.Any(p => p.UserId == joinGameDto.UserId))
return;

    var playerInfo = new PlayerDto(joinGameDto.UserId,
joinGameDto.Username, joinGameDto.Email,
joinGameDto.ProfileImageUrl);
```

```
    var serializedPlayer = JsonSerializer.Serialize(playerInfo);
    await db.ListRightPushAsync(gameKey, serializedPlayer);

    playersInGame = await db.ListRangeAsync(gameKey);
    var playerList = playersInGame.Select(p =>
JsonSerializer.Deserialize<PlayerDto>(p)).ToList();

    await Groups.AddToGroupAsync(Context.ConnectionId,
joinGameDto.GameId);
    await Clients.All.SendAsync("GetPlayers", joinGameDto.GameId,
playerList);
    await
Clients.Group(joinGameDto.GameId).SendAsync("PlayerJoined",
playerList, joinGameDto);
}
```

Листинг 3.30 – Код функции *JoinGame*

Такой подход позволяет динамически обновлять состав игроков в игре и поддерживать синхронизацию в реальном времени.

3.4.16 Прохождение викторин

Функция «Прохождение викторин», представленная под номером 16, реализована в методах *StartQuizAsync* для старта викторины, и *EndQuizAsync* для завершения викторины. Логика переключения вопросов реализована на клиенте в компоненте *QuizStartedPage*. Содержимое компонента представлено в приложении В.

Метод *StartQuizAsync* используется для начала викторины. Он принимает идентификатор викторины и идентификатор пользователя, инициирующего викторину. Метод выполняет поиск викторины в репозитории по переданному идентификатору. Если викторина не найдена, выбрасывается исключение *ArgumentException*. Если же викторина существует, она возвращается вызывающему коду, предоставляя начальные данные для ее прохождения.

Код метода *StartQuizAsync* представлен на листинге 3.31.

```
public async Task<Quiz> StartQuizAsync(int quizId, int userId,
CancellationToken token = default) {
    var quiz = await _quizRepository.GetItemAsync(quizId, token);
    if (quiz == null)
        throw new ArgumentException("Quiz not found");
    return quiz;
}
```

Листинг 3.31 – Код функции StartQuizAsync

Метод *EndQuizAsync* предназначен для завершения викторины и сохранения ее результатов. Он принимает идентификатор викторины и *DTO* с данными, предоставленными пользователем после завершения викторины. Метод находит

соответствующую викторину в репозитории и проверяет правильность ответов, предоставленных пользователем.

Содержимое класса *QuizFinishedDto* представлено в листинге 3.32.

```
public class QuizFinishedDto {
    public int UserId { get; set; }
    public List<QuizUserAnswerDto> UserAnswers { get; set; }
}
```

Листинг 3.32 – Содержимое класса *QuizFinishedDto*

Для каждого ответа из списка проверяется, совпадает ли идентификатор выбранного варианта с правильным вариантом ответа в вопросе. Количество правильных ответов подсчитывается и сохраняется как часть результата викторины в репозитории результатов. После этого метод возвращает обновленные данные о викторине.

Код метода *EndQuizAsync* представлен на листинге 3.33.

```
public async Task<Quiz> EndQuizAsync(int quizId, QuizFinishedDto
quizFinishedDto, CancellationToken token = default) {
    var quiz = await _quizRepository.GetItemAsync(quizId, token);
    if (quiz == null) throw new Exception("Quiz not found.");
    int correctAnswersCount = 0;
    foreach (var userAnswer in quizFinishedDto.UserAnswers) {
        var question = quiz.Questions.FirstOrDefault(q => q.Id ==
userAnswer.QuestionId);
        if (question != null && question.AnswerOptions.Any(a => a.Id
== userAnswer.AnswerId && a.IsCorrect)) {
                    correctAnswersCount++;
            }
        }
        var quizResult = new QuizResults(quizId, DateTime.UtcNow,
quizFinishedDto.UserId, correctAnswersCount, quiz.Questions.Count);

        await _quizResultsRepository.CreateAsync(quizResult, token);
        return await _quizRepository.GetItemAsync(quizId, token);
        }
```

Листинг 3.33 – Код функции *EndQuizAsync*

Эти методы обеспечивают серверную обработку начала и завершения викторин, в то время как управление вопросами, таймерами и отображением ответов выполняется на стороне клиента.

3.4.17 Просмотр результатов прохождения викторин

Функция «Просмотр результатов прохождения викторин», представленная под номером 17, реализована в виде асинхронного метода *GetQuizRatingAsync*, который возвращает список объектов *QuizRatingDto*. Метод предназначен для вычисления и отображения рейтинга пользователей по итогам участия в викторинах.

Метод начинается с извлечения всех результатов викторин из репозитория *_quizResultsRepository*. Полученные данные группируются по пользователю, чтобы агрегировать их достижения. Для каждой группы рассчитываются общее количество правильных ответов, общее количество вопросов, а также время завершения последней викторины.

После группировки данные сортируются по убыванию количества правильных ответов и по времени завершения. Выбираются первые 10 записей, чтобы отобразить рейтинг из топ-10 игроков.

Код метода *GetQuizRatingAsync* представлен на листинге 3.34.

```
public async Task<List<QuizRatingDto>>
GetQuizRatingAsync(CancellationToken cancellationToken = default) {
    var quizResults = await
_quizResultsRepository.GetElementsAsync(cancellationToken);
    var topPlayers = quizResults.GroupBy(qr => qr.User).Select(group
=> new {
            User = group.Key,
            TotalCorrectAnswers = group.Sum(qr =>
qr.NumberOfCorrectAnswers),
            TotalQuestions = group.Sum(qr => qr.NumberOfQuestions),
            LastEndTime = group.Max(qr => qr.EndTime)
        }).OrderByDescending(x => x.TotalCorrectAnswers).ThenBy(x =>
x.LastEndTime).Take(10).ToList();
    var result = topPlayers.Select((player, index) => new
QuizRatingDto(index + 1, player.User.Username, player.User.Email,
player.TotalCorrectAnswers, player.TotalQuestions,
player.TotalQuestions > 0 ?
Math.Round((double)player.TotalCorrectAnswers /
player.TotalQuestions * 100, 2) : 0, player.LastEndTime)).ToList();
    return result;
}
```

Листинг 3.34 – Код функции *GetQuizRatingAsync*

Метод возвращает список преобразованных данных в виде итогового рейтинга, позволяя удобно отображать их в пользовательском интерфейсе.

3.4.18 Принятие участия в играх

Функция «Принятие участия в играх», представлена под номером 18, реализована методе *SubmitAnswers* для сохранения ответа на текущий вопрос и на клиенте в компоненте *GameStartedPage*.

На клиентской стороне компонент *GameStartedPage* отвечает за отправку ответов, вызов метода *SubmitAnswers*, а также за управление интерфейсом, отображающим текущие вопросы, варианты ответов и статус игры. За переключение вопросов, запуск игры и её завершение отвечает ведущий игры. Содержимое компонента представлено в приложении Г.

Код метода *SubmitAnswers* представлен на листинге 3.35.

```
public async Task SubmitAnswers(string gameId, string userId,
List<AnswerSubmitedDto> answers) {
  try {
    var db = _redis.GetDatabase();
    var serializedAnswers = JsonSerializer.Serialize(answers);
  await db.StringSetAsync($"running:game:{gameId}:answers:{userId}",
serializedAnswers);
     }
     catch (Exception ex) {
         Console.WriteLine($"Error in SubmitAnswers: {ex.Message}");
         throw;
     }
}
```

Листинг 3.35 – Код функции *SubmitAnswers*

Метод *SubmitAnswers* отвечает за обработку и сохранение ответов пользователя. Он принимает три параметра: идентификатор игры (*gameId*), идентификатор пользователя (*userId*) и список предоставленных ответов (*answers*), представленных в объекте *AnswerSubmitedDto*.

Содержимое класса *AnswerSubmitedDto* представлено в листинге 3.36.

```
public class AnswerSubmitedDto {
    public int QuestionId { get; set; }
    public int AnswerId { get; set; }
}
```

Листинг 3.36 – Содержимое класса *AnswerSubmitedDto*

После получения данных метод выполняет их сериализацию в строку формата *JSON* и сохраняет результат в базе данных *Redis* с использованием уникального ключа, сформированного на основе идентификаторов игры и пользователя. Это позволяет надежно сохранить ответы каждого участника игры отдельно.

3.4.19 Просмотр результатов игр и рейтинга игроков в игровой комнате

Функция «Результаты игр и рейтинг игроков в игровой комнате», представлена под номером 19, реализована методом *GetGameStatisticsAsync*, который возвращает список объектов *GameRatingDto*. Метод предоставляет статистику игроков по определенной игре, включая рейтинг и точность ответов.

Метод начинает работу с проверки существования игры в базе данных через репозиторий *_gameRepository*. Если игра не найдена, выбрасывается исключение *KeyNotFoundException*. Затем из репозитория *_gameResultsRepository* извлекаются результаты игры, фильтруя их по идентификатору игры. Если результаты отсутствуют, генерируется исключение *InvalidOperationException*.

Общее количество вопросов в игре извлекается из объекта *game*, чтобы использовать его для расчета точности (*Accuracy*) игроков.

Код метода *GetGameStatisticsAsync* представлен на листинге 3.37.

```
public async Task<IEnumerable<GameRatingDto>>
GetGameStatisticsAsync(int gameId, CancellationToken
cancellationToken = default) {
    var game = await _gameRepository.GetItemAsync(gameId,
cancellationToken);
    if (game == null) throw new KeyNotFoundException($"Game with ID
{gameId} not found.");
    var gameResults = (await
_gameResultsRepository.GetElementsAsync(cancellationToken)).Where(gr
=> gr.GameId == gameId).ToList();
    if (!gameResults.Any()) throw new InvalidOperationException($"No
results found for Game ID {gameId}.");
    var TotalQuestions = game.Questions.Count();
    var rankedResults = gameResults.Select(gr => new {
            gr.UserId, gr.User.Username,
            gr.User.ProfileImageUrl,
            gr.NumberOfCorrectAnswers,
            TotalQuestions = TotalQuestions,
            Accuracy = TotalQuestions > 0 ?
(double)gr.NumberOfCorrectAnswers / TotalQuestions * 100 : 0
        }).OrderByDescending(r => r.Accuracy).ToList();
    var gameRatingDtos = rankedResults.Select((result, index) => new
GameRatingDto(index + 1, result.Username ?? "Unknown",
result.ProfileImageUrl, result.NumberOfCorrectAnswers,
result.Accuracy)).ToList();
    return gameRatingDtos;
}
```

Листинг 3.37 – Код функции *GetGameStatisticsAsync*

Каждый результат преобразуется в объект, включающий идентификатор пользователя, имя пользователя, *URL* изображения профиля, количество правильных ответов и точность, которая вычисляется как процентное соотношение правильных ответов к общему количеству вопросов. Затем результаты сортируются по убыванию точности.

После сортировки данные преобразуются в список объектов *GameRatingDto*, где каждому игроку присваивается ранг на основе его позиции в рейтинге. В случае отсутствия имени пользователя, используется значение «*Unknown*».

Метод возвращает список *DTO*, представляющий рейтинг игроков в рамках указанной игры, что позволяет отобразить информацию о производительности участников в игровой комнате.

3.4.20 Размещение комментариев к пройденной викторине

Функция «Размещение комментариев к пройденной викторине», представлена под номером 20, реализована методом *CreateQuizCommentAsync*, который обрабатывает добавление комментария к конкретной викторине. Сначала выполняется проверка существования викторины в репозитории по переданному идентификатору. Если викторина с указанным *quizId* не найдена, выбрасывается исключение *NotFoundException*.

Если викторина существует, создается объект *QuizComment*, содержащий данные из *DTO*: идентификатор пользователя, текст комментария, имя пользователя и *URL* изображения профиля. Этот объект связывается с идентификатором викторины. Содержимое класса *CreateQuizCommentDto* представлено в листинге 3.38.

```
public class CreateQuizCommentDto {
   public int QuizId { get; set; }
   public int UserId { get; set; }
   public string Content { get; set; } = null!;
   public string Username { get; set; } = null!;
   public string ProfileImageUrl { get; set; } = null!;
   public DateTime CommentDate { get; set; }
}
```

Листинг 3.38 – Содержимое класса *CreateQuizCommentDto*

Код метода *CreateQuizCommentAsync* представлен на листинге 3.39.

```
 public async Task<int> CreateQuizCommentAsync(int quizId,
CreateQuizCommentDto createCommentDto, CancellationToken
cancellationToken = default) {
    var quiz = await _quizRepository.GetItemAsync(quizId,
cancellationToken);
    if (quiz is null) throw new NotFoundException($"Quiz with ID
{quizId} not found.");
    var comment = new QuizComment(quiz.Id, createCommentDto.UserId,
createCommentDto.Content, createCommentDto.Username,
createCommentDto.ProfileImageUrl);
    await _quizCommentRepository.CreateAsync(comment,
cancellationToken);
    return comment.Id;
}
```

Листинг 3.39 – Код функции *CreateQuizCommentAsync*

Созданный комментарий сохраняется в репозитории через вызов метода *CreateAsync*. В завершение возвращается идентификатор созданного комментария. Такой подход обеспечивает обработку данных комментария и их сохранение в базе, а также позволяет идентифицировать каждый добавленный комментарий.

3.5 Вывод к разделу

1. Для серверной части приложения реализованы слои доступа к данным, бизнес-логики и представления (*API*). Применена архитектура с использованием *API Gateway*, обеспечивающего маршрутизацию запросов. Использованы дополнительные библиотеки для повышения производительности и обеспечения безопасности.

2. В клиентской части приложения разработаны пользовательские интерфейсы с учетом удобства использования и адаптации к различным

устройствам. Интеграция с серверной частью выполнена через *API*, что обеспечивает полноценное взаимодействие между компонентами системы.

3. Настроен *Nginx*, выполняющий функции обратного прокси-сервера и маршрутизации запросов. Обеспечена поддержка SSL-соединений и статических файлов.

4. Реализовано 20 функций, включая регистрацию, авторизацию, просмотр и редактирование профилей пользователей, создание и прохождение викторин, взаимодействие с игровыми комнатами, управление комментариями и рейтинговыми данными.

5. В проекте применены современные технологии, обеспечивающие масштабируемость, безопасность и удобство использования системы. Для серверной части был использован паттерн *DI* совместно с *DTO*.

6. В ходе реализации функций также было внедрено 37 маршрутов, которые направляют запросы на контроллеры. Схема маршрутов описана в таблице 3.16.

4 Тестирование web-приложения

4.1 Функциональное тестирование

В данном разделе будет рассмотрен процесс тестирования готового, целостного приложения. Тестирование будет проводиться с использованием клиентской части. В таблице 4.1 представлено описание тест-кейсов.

Таблица 4.1 – Описание тест-кейсов

№	Функция	Описание тест-кейса	Ожидаемый результат	Фактический результат
1	Регистрация в системе	1. Ввести некорректную почту и отправить форму	Появляется сообщение об ошибке	Тест-кейс пройден, ошибок не обнаружено
		2. Ввести почту, на которую зарегистрирован другой пользователь и отправить форму	Появляется сообщение об ошибке	Тест-кейс пройден, ошибок не обнаружено
		3. Ввести корректную почту и отправить форму	Отправка системой кода подтверждения на указанную почту. Перенаправление на форму ввода кода подтверждения	Тест-кейс пройден, ошибок не обнаружено
		4. Ввести некорректный код подтверждения и отправить форму	Появляется сообщение об ошибке	Тест-кейс пройден, ошибок не обнаружено
		5. Ввести корректный код, отправленный системой на указанную почту и отправить форму	Перенаправление на форму ввода имени пользователя и пароля	Тест-кейс пройден, ошибок не обнаружено
		6. Ввести некорректные имя пользователя или пароль	Появляется сообщение об ошибке	Тест-кейс пройден, ошибок не обнаружено
		7. Ввести корректные имя пользователя и пароль и отправить форму	Учетная запись создана, пользователь получает доступ к системе	Тест-кейс пройден, ошибок не обнаружено

Продолжение таблицы 4.1

№	Функция	Описание тест-кейса	Ожидаемый результат	Фактический результат
2	Просмотр информации о доступных викторинах	1. Переход на страницу викторин	Отображение списка доступных викторин в левой части страницы	Тест-кейс пройден, ошибок не обнаружено
3	Просмотр информации о существующих играх	1. Переход на страницу игр	Отображается список существующих игр	Тест-кейс пройден, ошибок не обнаружено
4	Просмотр рейтингов других игроков по прохождению викторин	1. Переход на страницу викторин	Отображение рейтинга игроков по прохождению викторин в правой части страницы	Тест-кейс пройден, ошибок не обнаружено
5	Просмотр профилей пользователей	1. Переход на страницу конкретного пользователя	Отображение профиля пользователя	Тест-кейс пройден, ошибок не обнаружено
6	Авторизация в системе	1. Ввести корректную почту и пароль. Отправить форму	Пользователь успешно авторизуется и перенаправляется на главную страницу.	Тест-кейс пройден, ошибок не обнаружено
		2. Ввести некорректные почту или пароль. Отправить форму	Появляется сообщение об ошибке	Тест-кейс пройден, ошибок не обнаружено
7	Создание вопросов и вариантов ответов для конкретной игры	1. Нажать кнопку «Create question» и в появившемся диалоговом окне ввести корректное название вопроса и текст для четырех вариантов ответа. Выбрать один правильный вариант ответа и отправить форму	Отображение созданного вопроса в списке вопросов в левой части страницы	Тест-кейс пройден, ошибок не обнаружено
		2. Нажать кнопку «Create question» и в появившемся диалоговом окне ввести некорректные данные и отправить форму	Появляется сообщение об ошибке	Тест-кейс пройден, ошибок не обнаружено

Продолжение таблицы 4.1

№	Функция	Описание тест-кейса	Ожидаемый результат	Фактический результат
		3. Нажать кнопку «Create question» и в появившемся диалоговом окне ввести корректные данные, но не выбрать ни одного правильного ответа и отправить форму	Появляется сообщение об ошибке	Тест-кейс пройден, ошибок не обнаружено
8	Запуск игры	1. Нажатие кнопки «Start»	Появление обратного отсчёта и запуск игры	Тест-кейс пройден, ошибок не обнаружено
9	Переход к следующему вопросу	1. Нажатие кнопки «Next»	Отображение следующего вопроса и вариантов ответа	Тест-кейс пройден, ошибок не обнаружено
10	Завершение игры	1. Нажатие кнопки «End game»	Перенаправление на страницу завершенной игры. Завершение игры. Отображение результатов	Тест-кейс пройден, ошибок не обнаружено
11	Размещение комментариев к завершенной игре	1. Ввести текст комментария и отправить форму	Комментарий успешно сохранен и отображён в списке комментариев	Тест-кейс пройден, ошибок не обнаружено
		2. Оставить поле для ввода комментария пустым и отправить форму	Отображение сообщения об ошибке	Тест-кейс пройден, ошибок не обнаружено
12	Удаление комментариев	1. Выбрать комментарий и нажать на кнопку «Delete»,	Удаление выбранного комментария. Отображение обновленного списка комментариев в нижней части страницы	Тест-кейс пройден, ошибок не обнаружено

Продолжение таблицы 4.1

№	Функция	Описание тест-кейса	Ожидаемый результат	Фактический результат
13	Создание игровых комнат	1. Перейти на страницу списка игр и нажать кнопку «Create game». В диалоговом окне ввести название игры и максимальное количество игроков. Отправить форму	Игра успешно создана и отображена в списке игр	Тест-кейс пройден, ошибок не обнаружено
		2. Перейти на страницу списка игр и нажать кнопку «Create game». В диалоговом окне ввести некорректные название игры или максимальное количество игроков и отправить форму	Появляется сообщение об ошибке	Тест-кейс пройден, ошибок не обнаружено
14	Редактирование личной информации	1. Перейти на страницу собственного профиля. Нажать кнопку «Update profile» и в диалоговом окне ввести корректные обновляемые данные и отправить форму	Обновление личной информации пользователя в его профиле	Тест-кейс пройден, ошибок не обнаружено
15	Присоединение к играм	1. На странице конкретной игры нажать кнопку «Connect»	Отображение себя как пользователя в списке подключенных к игровой комнате игроков	Тест-кейс пройден, ошибок не обнаружено
16	Прохождение викторин	1. На странице викторины нажатие кнопки «Start» для запуска викторины. Выбрать один из 4 вариантов ответа на вопрос, до тех пор, пока вопросы не закончатся. По достижению последнего вопроса нажать кнопку «End»	Запуск викторины. Ответы на вопросы. Перенаправление на страницу итогов викторины	Тест-кейс пройден, ошибок не обнаружено

Продолжение таблицы 4.1

№	Функция	Описание тест-кейса	Ожидаемый результат	Фактический результат
17	Просмотр результатов прохождения викторин	1. Переход на страницу завершенной викторины	Отображение диаграммы корректности данных пользователем ответов на вопросы. Отображение количества правильных ответов на вопросы	Тест-кейс пройден, ошибок не обнаружено
18	Принятие участия в играх	1. На странице игры после подключения к игровой комнате необходимо дождаться запуска игры от ведущего. Выбрать один из 4 вариантов ответа на вопрос, до тех пор, пока вопросы не закончатся	Запуск игры. Ответы на вопросы. Перенаправление на страницу итогов игры	Тест-кейс пройден, ошибок не обнаружено
19	Просмотр результатов игр и рейтинга игроков в игровой комнате	1. Переход на страницу завершенной игры	Отображение диаграммы корректности данных пользователями ответов на вопросы. Отображение рейтинга игроков в таблице	Тест-кейс пройден, ошибок не обнаружено
20	Размещение комментариев к пройденной викторине	1. Ввести текст комментария и отправить форму	Комментарий успешно сохранен и отображён в списке комментариев	Тест-кейс пройден, ошибок не обнаружено
		2. Оставить поле для ввода комментария пустым и отправить форму	Отображение сообщения об ошибке	Тест-кейс пройден, ошибок не обнаружено

Каждая функция была протестирована вручную по заранее подготовленным тест-кейсам, результаты которых зафиксированы в таблице. Все выявленные ошибки были исправлены, а функции повторно протестированы.

Функциональное тестирование подтвердило корректность реализации заявленного функционала системы.

4.2 Выводы по разделу

1. Тестирование проводилось ручным методом. Проведено тестирование всех 20 функций приложения. Все операции работают корректно.
2. Было разработано 20 тест-кейсов, все тесты выполнены успешно, а пользовательский интерфейс обновляется в ответ на изменения состояния данных.
3. Использование ручного тестирования позволило выявить и устранить мелкие недочеты в работе интерфейса и функционала.
4. Покрытия тестами *web*-приложения составляет 100%.

5 Руководство пользователя

5.1 Авторизация в системе

Для того чтобы авторизироваться в системе пользователю необходимо нажать на кнопку «*Login*» в верхнем правом углу экрана. На странице входа пользователь может произвести аутентификацию, заполнив все необходимые поля формы. Произойдет проверка существования данного пользователя и проверка правильности ввода пароля. Страница входа представлена на рисунке 5.1.

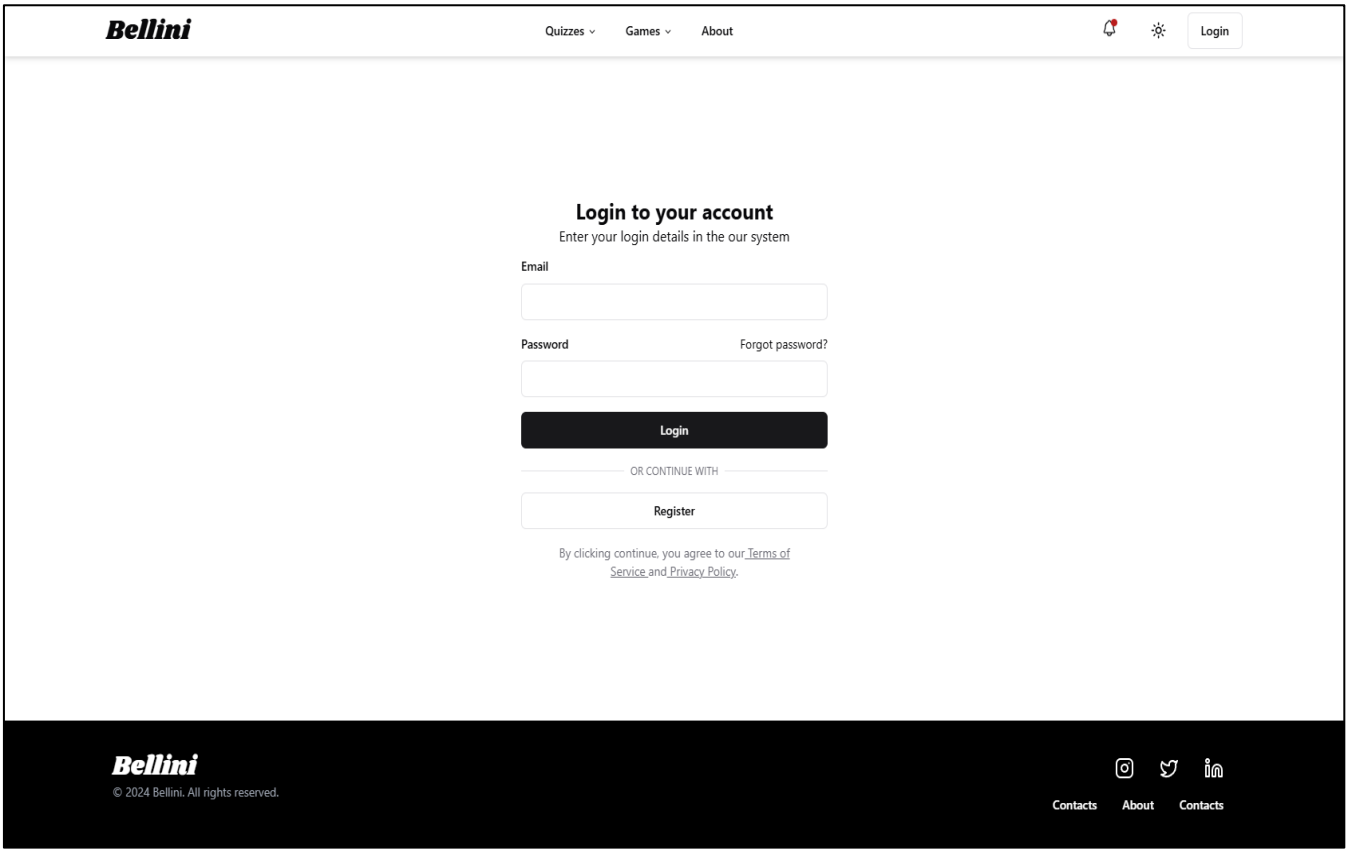

Рисунок 5.1 – Страница входа

В случае успешной аутентификации, пользователь будет перенаправлен на главную страницу.

5.2 Регистрация в системе

Для того чтобы зарегистрироваться в системе пользователю необходимо со страницы входа «*Login*»перейти на страницу регистрации, нажав на кнопку «*Register*».

На странице регистрации первоначально перед пользователем появляется форма для ввода электронной почты. Форма для ввода электронной почты представлена на рисунке 5.2.

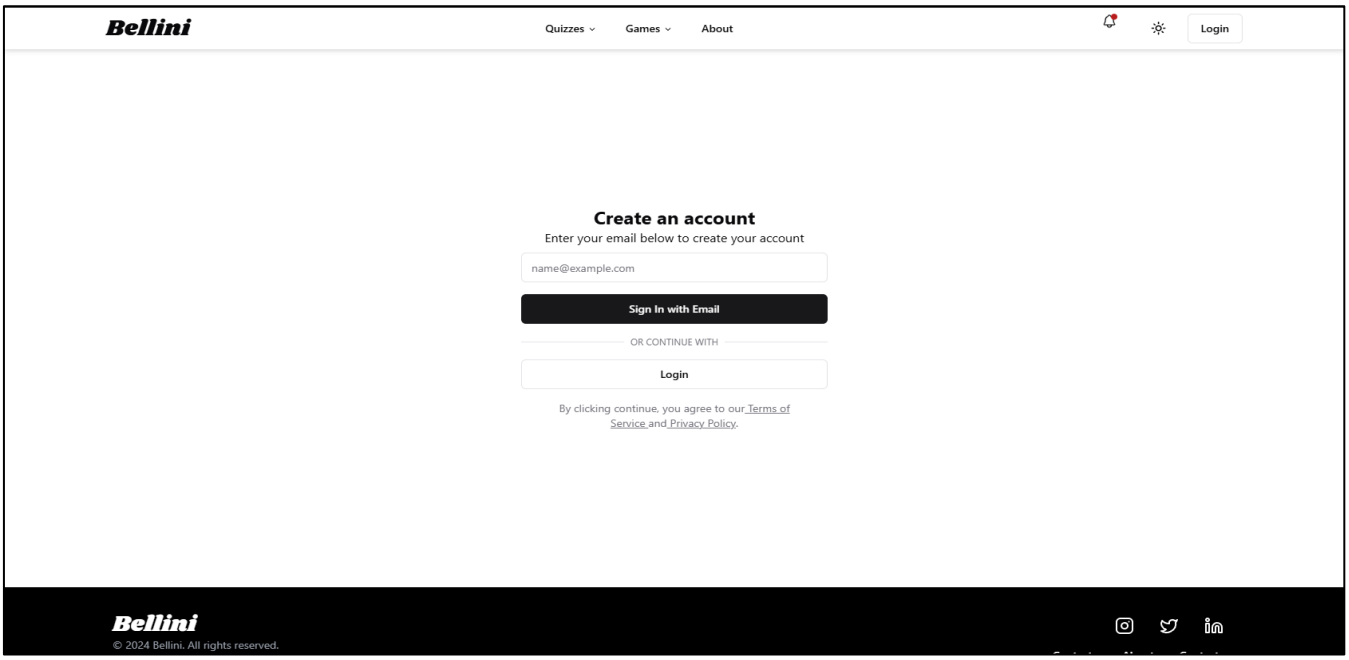

Рисунок 5.2 – Форма ввода электронной почты на странице регистрации

После ввода электронной почты запрос отправляется на сервер. Система проверяет, существует ли пользователь, зарегистрированный на данный почтовый ящик. Если совпадений не найдено, то происходит отправка письма с кодом на почту для подтверждения принадлежности электронной почты тому или иному пользователю. В этот момент пользователю отображается форма для ввода кода. Форма для ввода кода представлена на рисунке 5.3.

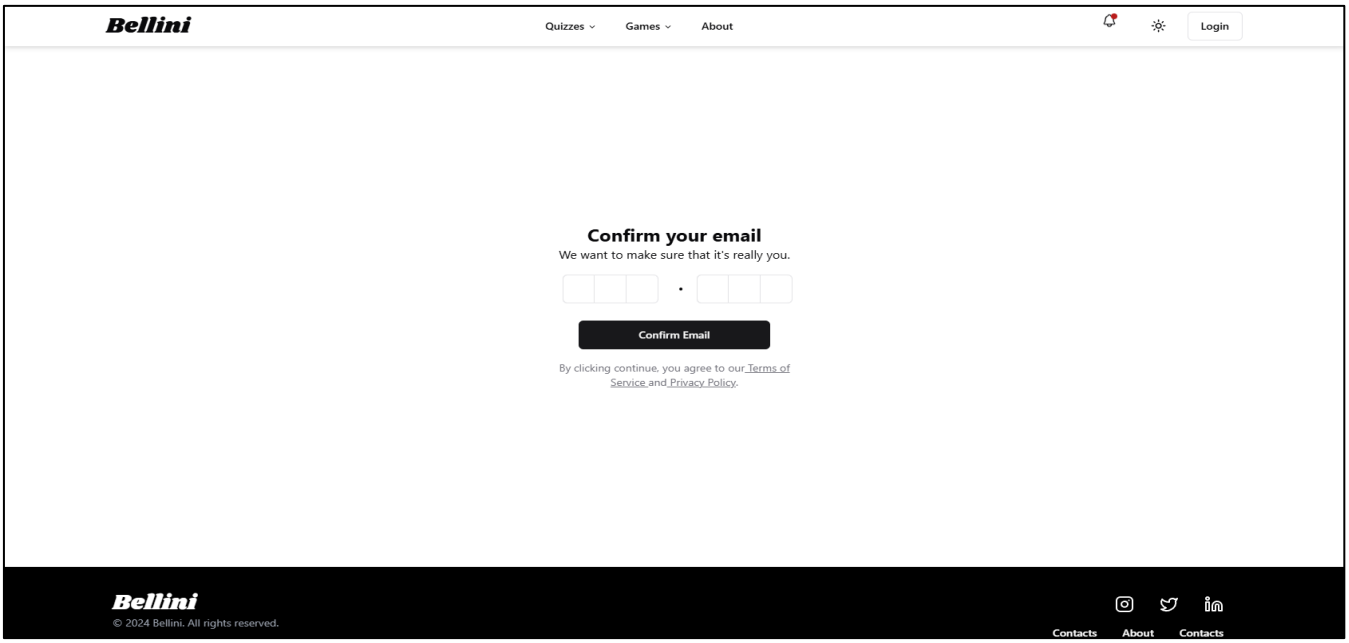

Рисунок 5.3 – Форма ввода кода на странице регистрации

После ввода кода из письма запрос с кодом отправляется на сервер и если все хорошо и данные корректны, то перед пользователем отображается форма для ввода

имени пользователя и пароля. Форма для ввода имени пользователя и пароля представлена на рисунке 5.4.

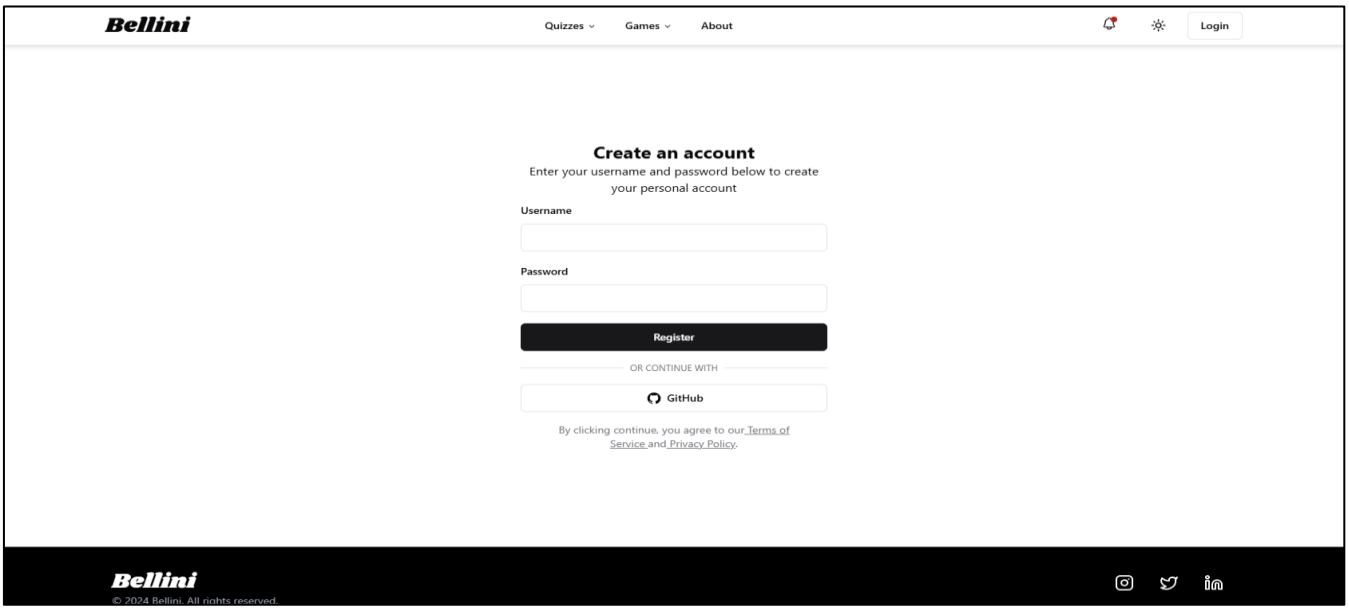

Рисунок 5.4 – Форма ввода имени пользователя и пароля на странице регистрации

После нажатия на кнопку «*Register*» пользователя перенаправит на главную страницу, что будет означать, что регистрация прошла успешно.

5.3 Просмотр информации о доступных викторинах

Для просмотра доступных викторин необходимо навести курсор на «*Quizzes*» в верхнем навигационном меню по центру экрана и во всплывающем окне выбрать интересующие викторины. Для просмотра всех викторин необходимо выбрать «*All quizzes*». После чего произойдет перенаправление на страницу викторин. Страница викторин представлена на рисунке 5.5.

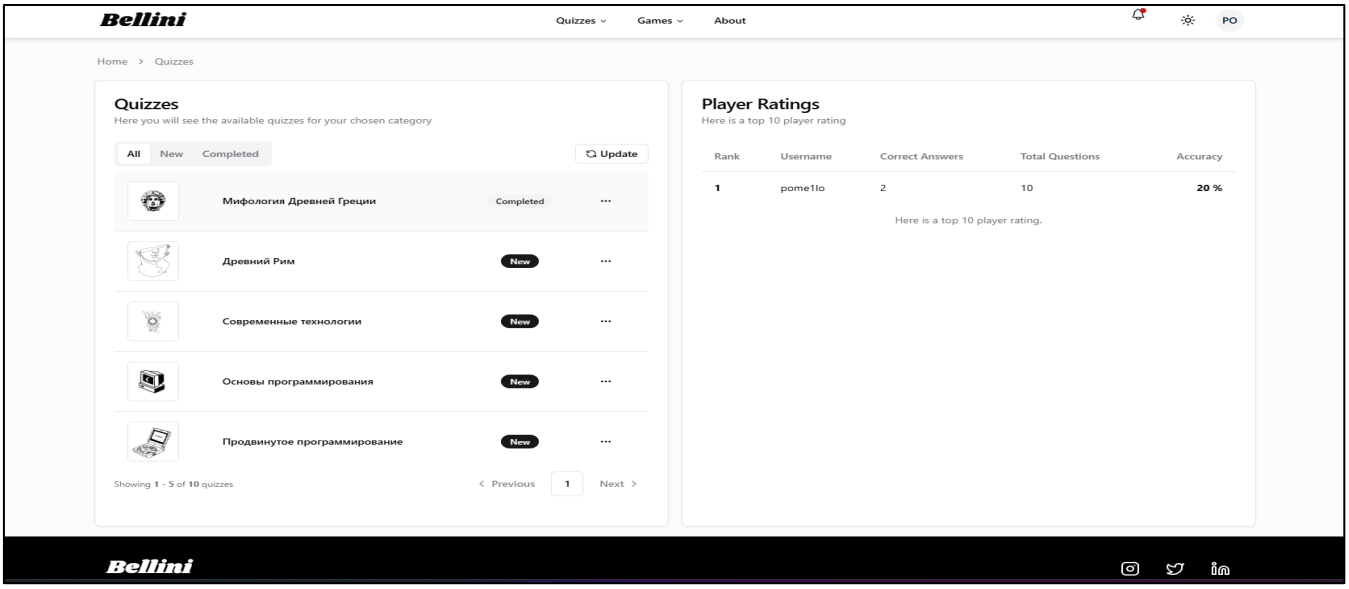

Рисунок 5.5 – Страница викторин

Слева отображен доступные викторины. Справа отображена таблица с рейтингом игроков.

5.4 Просмотр информации о существующих играх

Для просмотра существующих игр необходимо навести курсор на «*Games*» в верхнем навигационном меню по центру экрана и во всплывающем окне выбрать интересующие игры. Для просмотра всех игр необходимо выбрать «*All games*». После чего произойдет перенаправление на страницу игр. Здесь будет отображен список всех доступных игр для выбранной категории.

Страница игр представлена на рисунке 5.6.

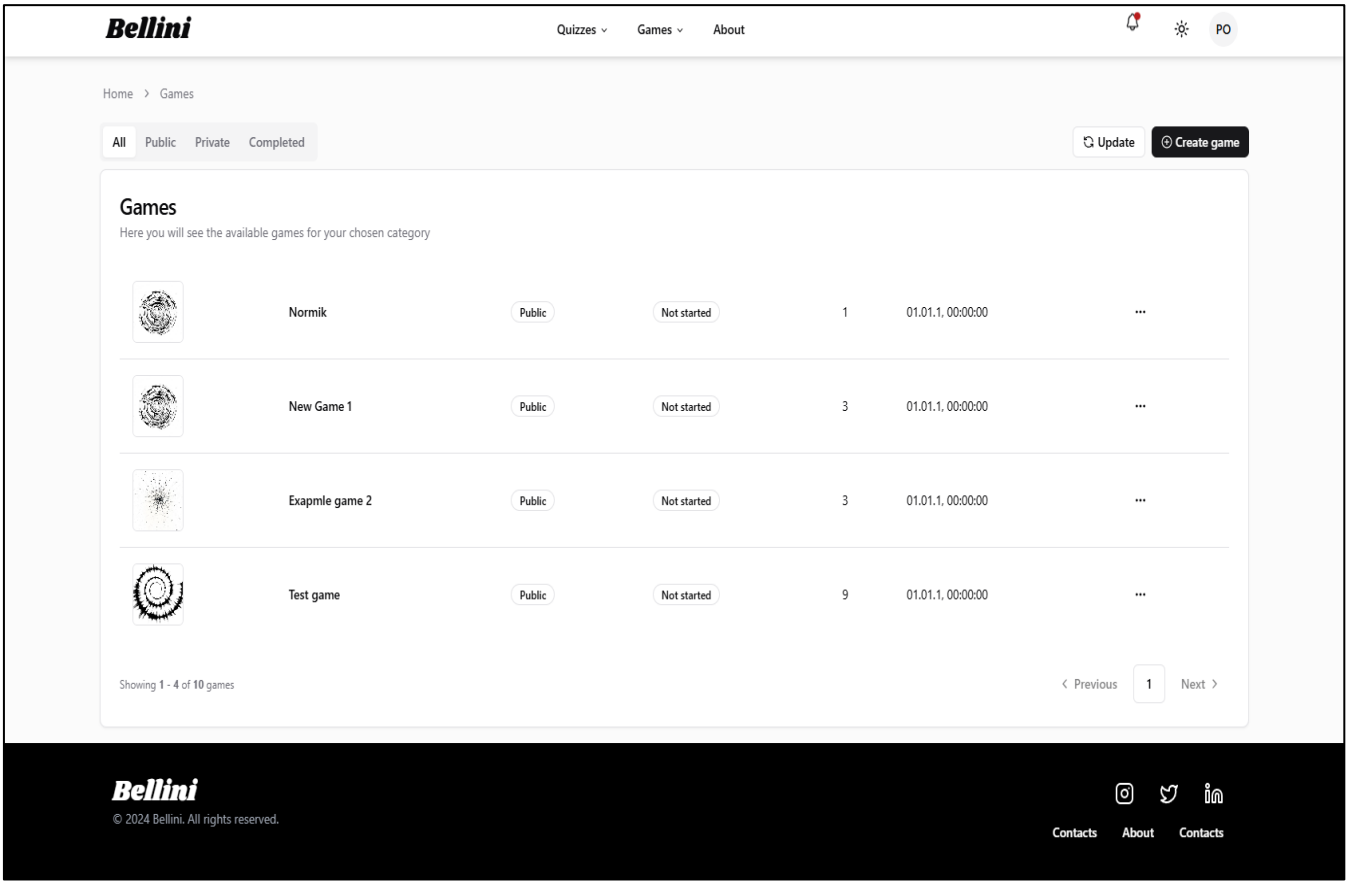

Рисунок 5.6 – Страница игр

На странице игр представлен список игр, доступных для участия всем пользователям.

5.5 Просмотр рейтингов других игроков по прохождению викторин

Для просмотра рейтингов игроков необходимо навести курсор на «*Quizzes*» в верхнем навигационном меню по центру экрана и во всплывающем окне выбрать викторины и перейти на страницу викторин, Страница викторин представлена на рисунке 5.7.

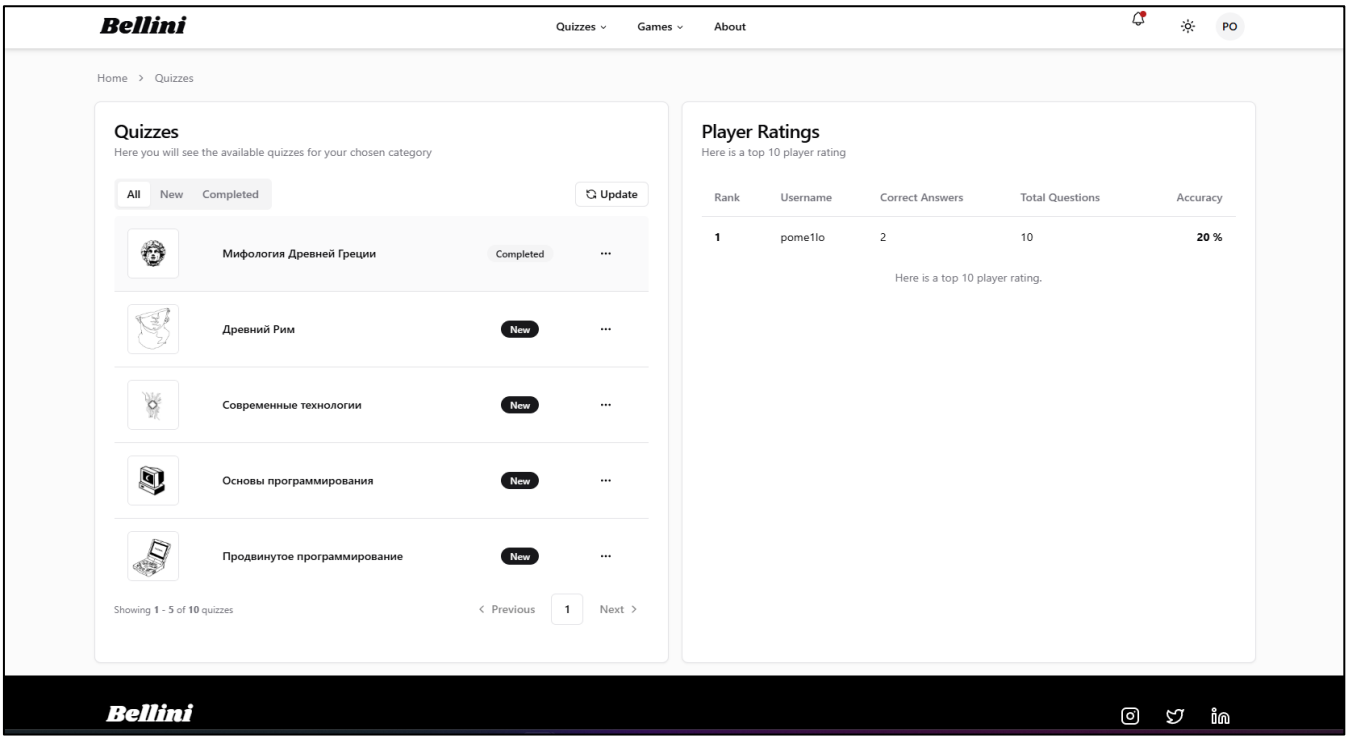

Рисунок 5.7 – Страница викторин

Слева отображен доступные викторины. Справа отображена таблица с рейтингом игроков.

5.6 Просмотр профилей пользователей

Для просмотра профиля того или иного пользователя необходимо перейти на страницу профиля пользователя, имея на неё ссылку. Ссылку на страницу можно получить если пользователь поделиться своей страницей, либо перейти на самостоятельно кликнув, например имя пользователя на комментарии.

Страница пользователя представлена на рисунке 5.8.

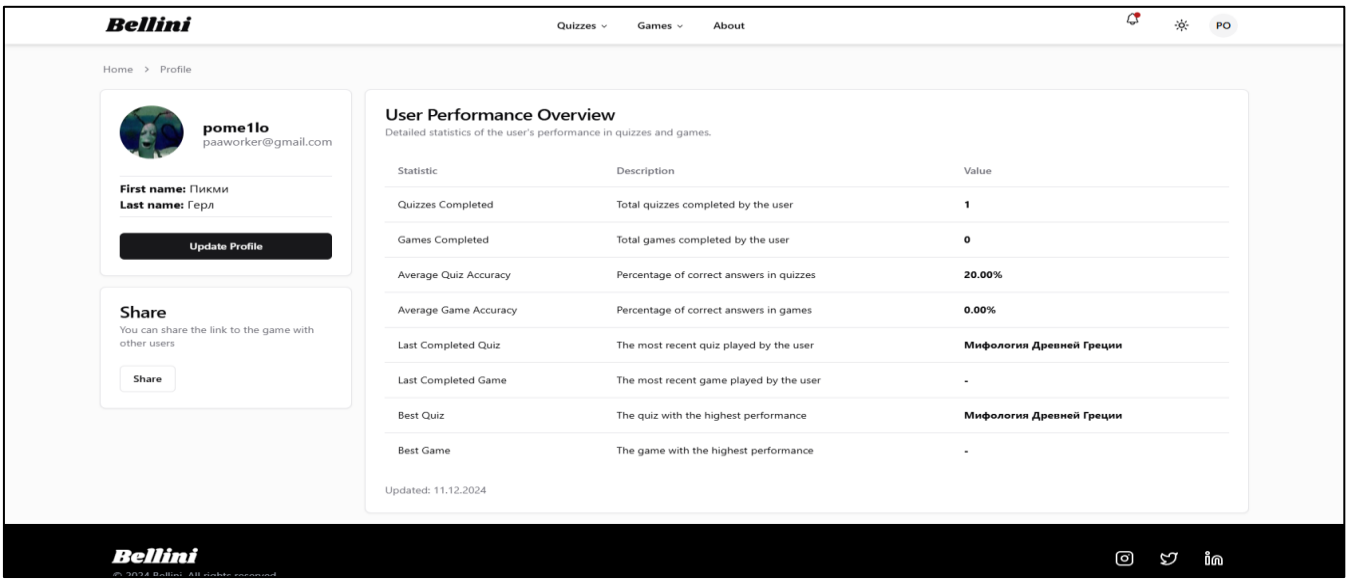

Рисунок 5.8 – Страница пользователя

На странице пользователя отображена его личная информация, а также статистические данные прохождения игр и викторин.

5.7 Создание игровых комнат

Для того чтобы создать игровую комнату пользователю необходимо авторизироваться в системе и перейти на страницу игр. В правом верхнем углу обнаружить кнопку «*Create game*» и нажать на неё. После чего перед пользователем отобразится диалоговое окно, в котором необходимо указать все необходимые параметры для создания игровой комнаты. Диалоговое окно представлено на рисунке 5.9.

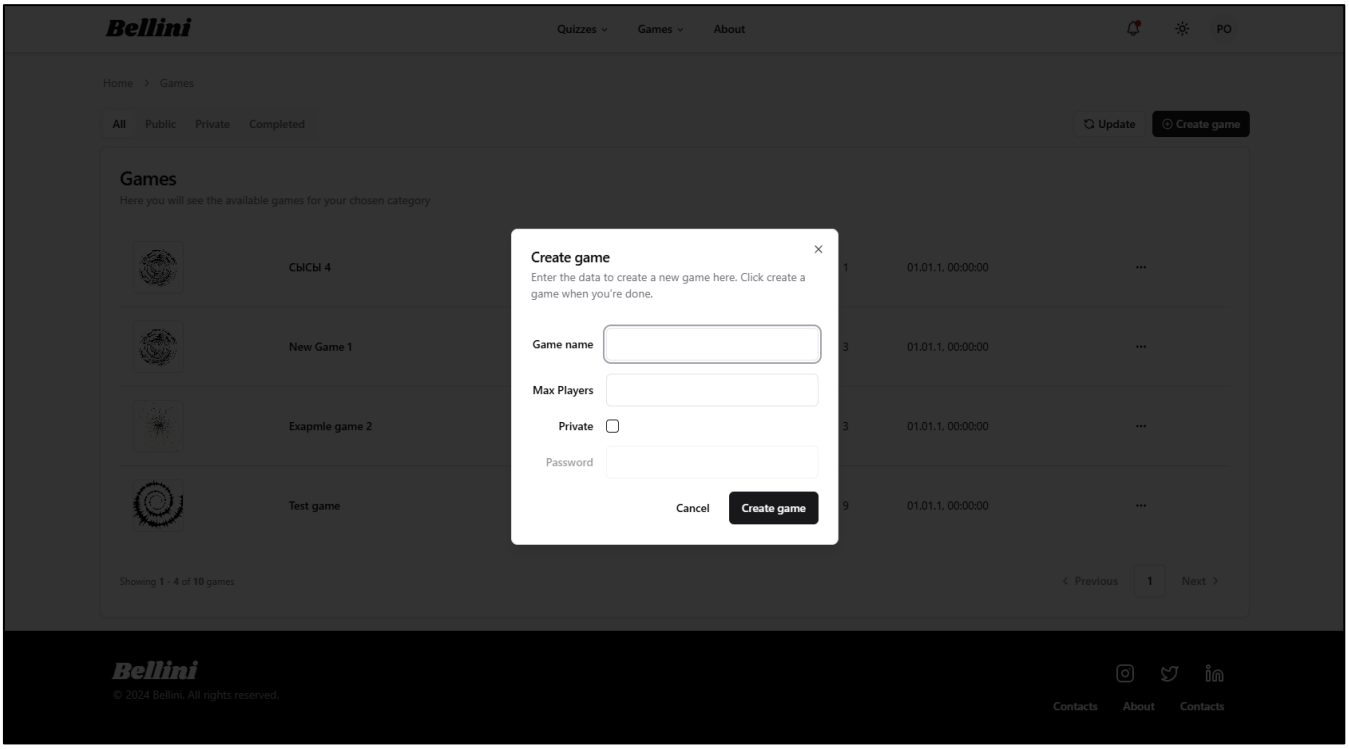

Рисунок 5.9 – Диалоговое окно для создания игровой комнаты

После ввода всех необходимых параметров игра будет успешно создана. Новая игра отобразиться в списке среди других игр.

5.8 Редактирование личной информации

Для того чтобы отредактировать личную информацию пользователю необходимо авторизироваться и нажать в верхнем правом углу на кружок со своими инициалами. В выпадающем меню выбрать «*Profile*». После чего произойдет перенаправление на страницу профиля.

На странице профиля пользователю необходимо нажать на кнопку «*Update profile*». После чего перед ним появится диалоговое окно для редактирования личной информации. Диалоговое окно представлено на рисунке 5.10.

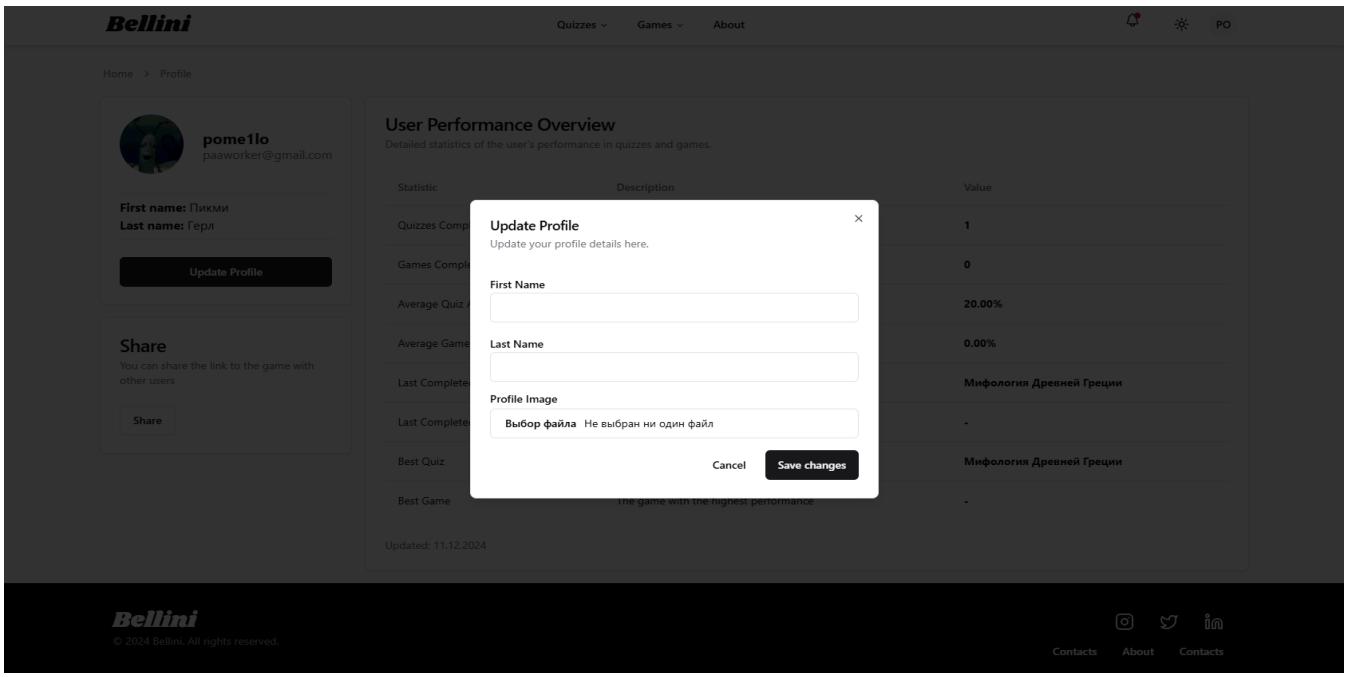

Рисунок 5.10 – Диалоговое окно для редактирования личной информации

В появившемся окне пользователь может ввести необходимые данные и после нажатия на кнопку «*Save changes*» увидеть внесенные изменения на странице профиля.

5.9 Присоединение к играм

Для присоединения к игре пользователю необходимо пройти процесс авторизации и перейти на страницу игр. После чего выбрать интересующею его игровую комнату и кликнуть на неё. После чего произойдет перенаправление в саму игровую комнату. Страница игровой комнаты представлена на рисунке 5.11.

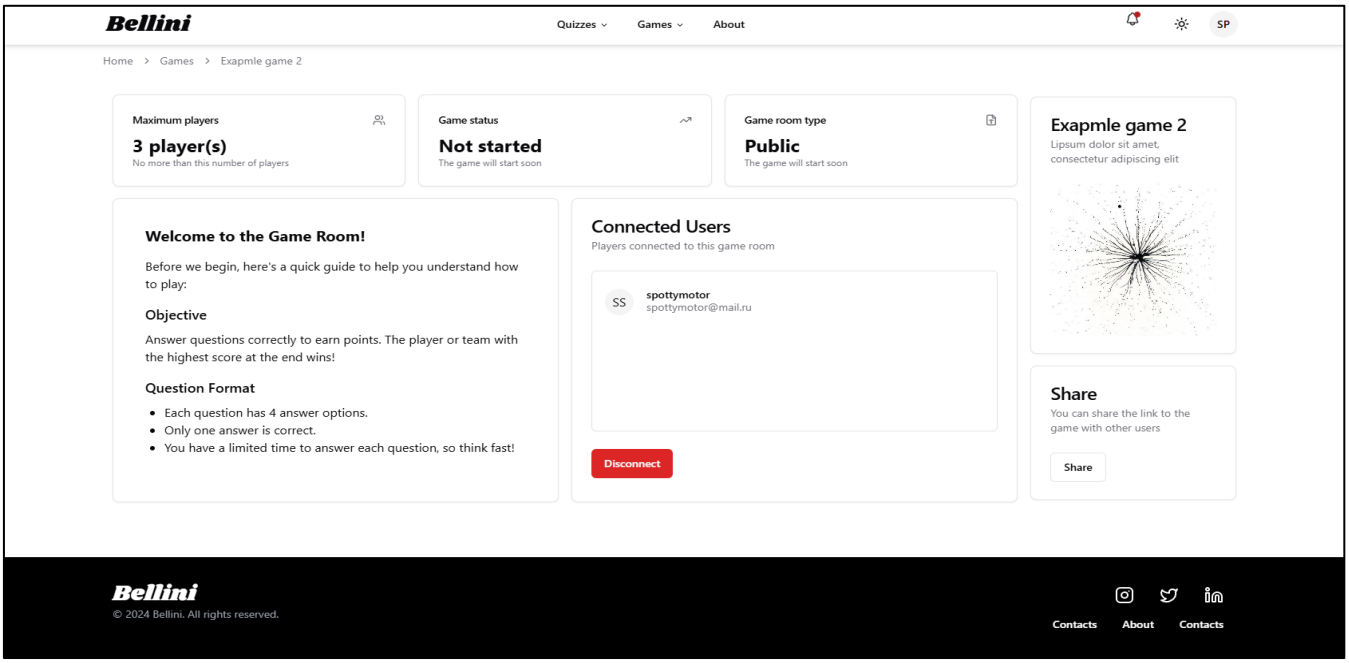

Рисунок 5.11 – Страница игровой комнаты

В блоке «*Connected Users*» отображается список пользователей, подключенных к игре. При нажатии на кнопку «*Connect*» пользователь успешно подключиться к игре.

5.10 Прохождение викторин

Для прохождения той или иной викторины пользователю необходимо пройти процесс авторизации и перейти на страницу викторин. После чего в списке викторин выбрать понравившеюся и кликнуть на неё. После чего произойдет перенаправление на страницу викторины. Страница викторины представлена на рисунке 5.12.

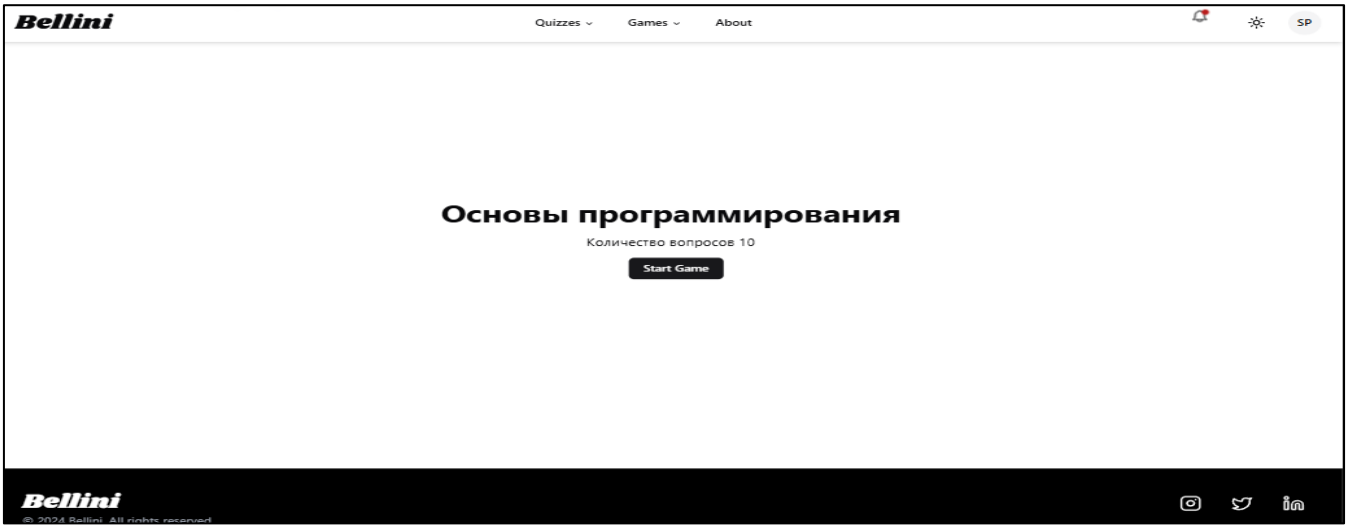

Рисунок 5.12 – Страница викторины

При нажатии на кнопку «*Start game*» викторина начнется и перед пользователем начнется обратный отсчет. После чего появится вопрос и 4 варианта ответа, один из которых правильный. Пример ответа на вопрос представлен на рисунке 5.13.

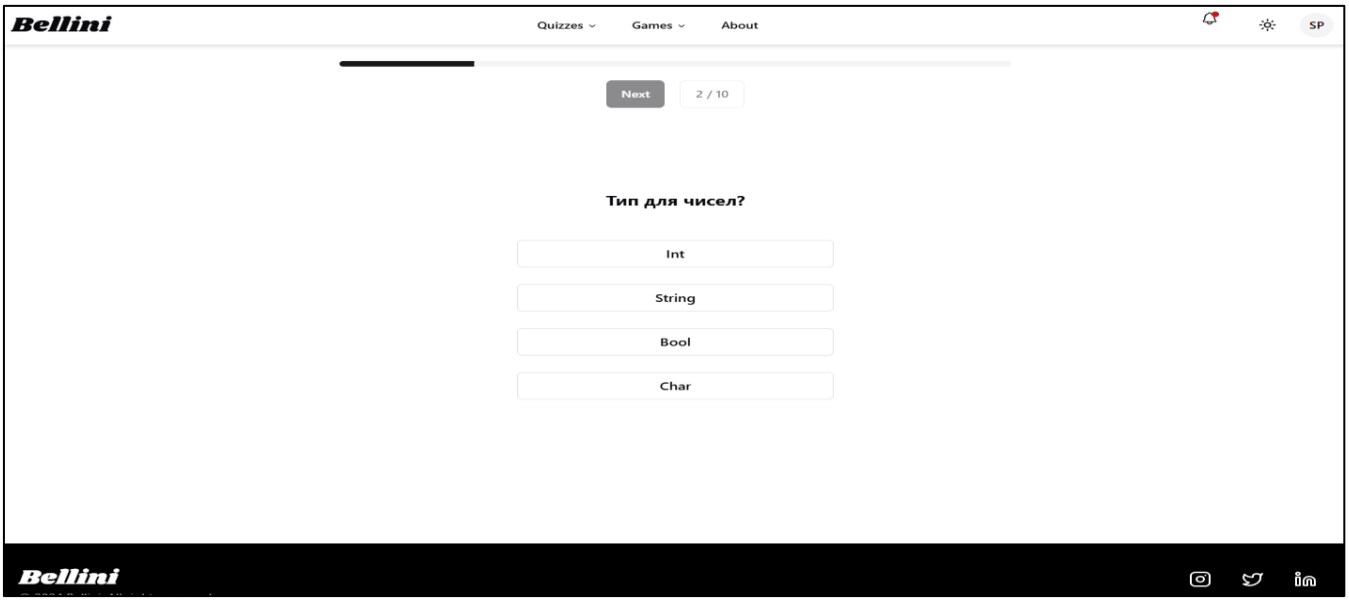

Рисунок 5.13 – Пример ответа на вопрос

После выбора варианта ответа необходимо нажать на кнопку «*Next*» и перейти к следующему вопросу. Этот процесс необходимо повторять до тех пор, пока не закончатся вопросы. По достижению последнего вопроса вместо кнопки «*Next*» появится кнопка «Finish», при нажатии на которую викторина будет завершена.

5.11 Просмотр результатов прохождения викторин

Для просмотра результатов прохождения викторины необходимо пройти процесс авторизации, после чего перейти на страницу викторин, выбрать и пройти одну из викторин. По итогу прохождения и нажатии кнопки «*Finish*» пользователю отобразится страница с результатами. Страница с результатами представлена на рисунке 5.14.

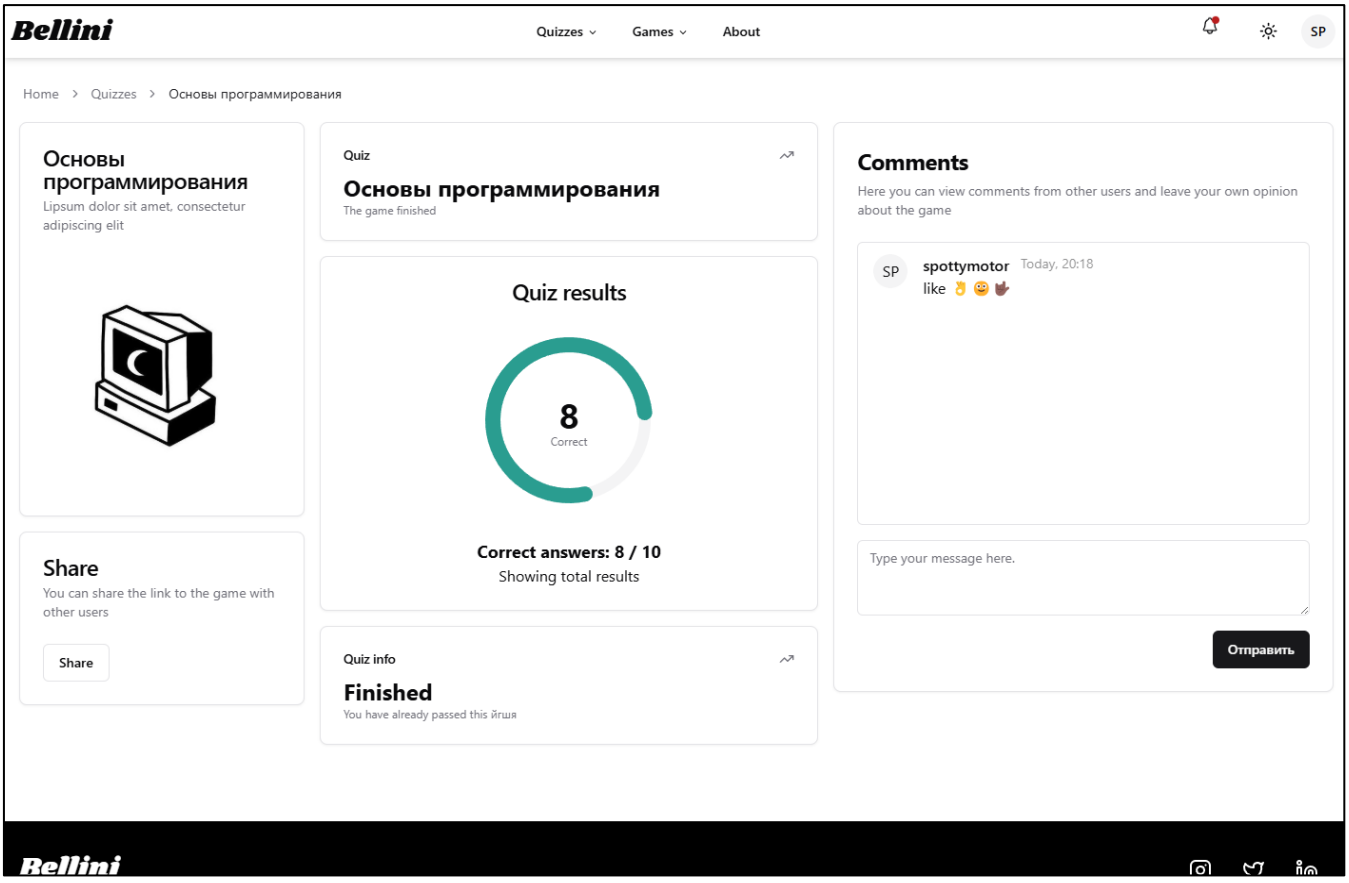

Рисунок 5.14 – Страница результатов викторины

Здесь отображено количество вопросов, количество правильных ответов, статус викторины и список комментариев.

5.12 Принятие участия в играх

Для принятия участия в играх пользователю необходимо пройти процесс авторизации и перейти на страницу игр. После чего выбрать понравившуюся игру и кликнуть на неё. После перенаправления в игровую комнату нажать на кнопку «*Connect*» и дожидаться старта игры от ведущего. После старта перед всеми

пользователями появится обратный отсчет. После чего отобразится вопрос и 4 варианта ответа. Пример ответа на вопрос в игре представлен на рисунке 5.15.

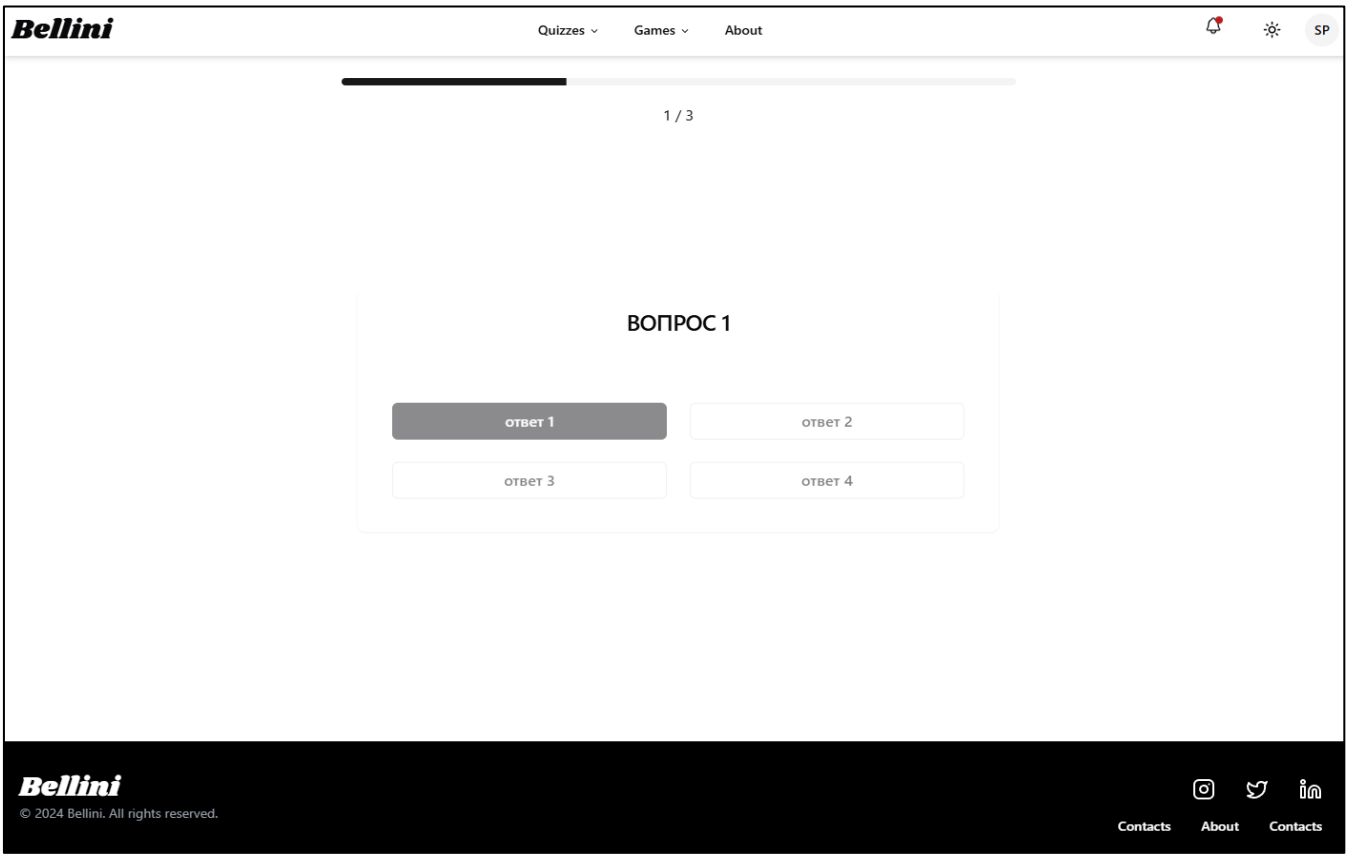

Рисунок 5.15 – Страница ответа на вопрос в игре

После выбора варианта ответа пользователю необходимо дождаться, пока ведущий не переключит вопрос. Данная процедура повторяется до тех пор, пока вопросы не закончатся. После чего всем участникам игры будут отображены результаты игры.

5.13 Просмотр результатов игр и рейтинга игроков в игровой комнате

Для просмотра результатов игры пользователю необходимо пройти процесс авторизации и перейти на страницу игр. После чего выбрать понравившуюся игру и кликнуть на неё. После перенаправления в игровую комнату нажать на кнопку «*Connect*» и дожидаться старта игры от ведущего. После старта перед всеми пользователями появится обратный отсчет. После чего отобразится вопрос и 4 варианта ответа. После выбора варианта ответа пользователю необходимо дождаться, пока ведущий не переключит вопрос. Данная процедура повторяется до тех пор, пока вопросы не закончатся. После чего всем участникам игры будут отображены результаты игры.

Страница результатов игры представлена на рисунке 5.16.

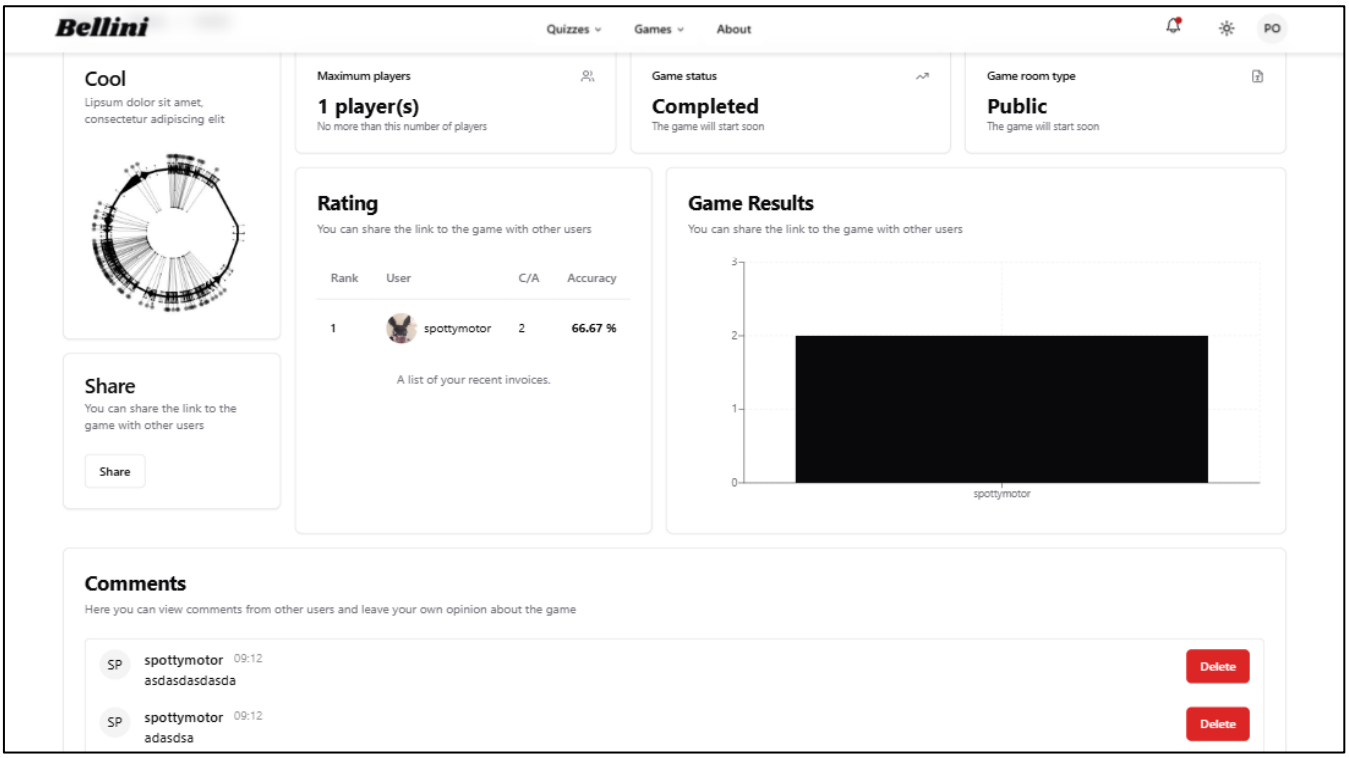

Рисунок 5.16 – Страница результатов игры

5.14 Размещение комментариев к завершенной игре

Для размещения комментариев к игре пользователю необходимо пройти процесс авторизации и перейти на страницу игр. После чего выбрать понравившуюся игру и кликнуть на неё. После перенаправления в игровую комнату нажать на кнопку «*Connect*» и дожидаться старта игры от ведущего. После старта перед всеми пользователями появится обратный отсчет. После чего отобразится вопрос и 4 варианта ответа. После выбора варианта ответа пользователю необходимо дождаться, пока ведущий не переключит вопрос. Данная процедура повторяется до тех пор, пока вопросы не закончатся. После чего всем участникам игры будут отображены результаты игры. Снизу будет отображаться список комментариев. Страница результатов игры со списком комментариев представлена на рисунке 5.16. В списке комментариев отображены комментарии, оставленные участниками игры.

5.15 Размещение комментариев к пройденной викторине

Для размещения комментариев к викторине необходимо пройти процесс авторизации, после чего перейти на страницу викторин, выбрать и пройти одну из викторин. По итогу прохождения и нажатии кнопки «*Finish*» пользователю отобразится страница с результатами. Справа будет отображен список комментариев.

Страница со списком комментариев представлена на рисунке 5.17.

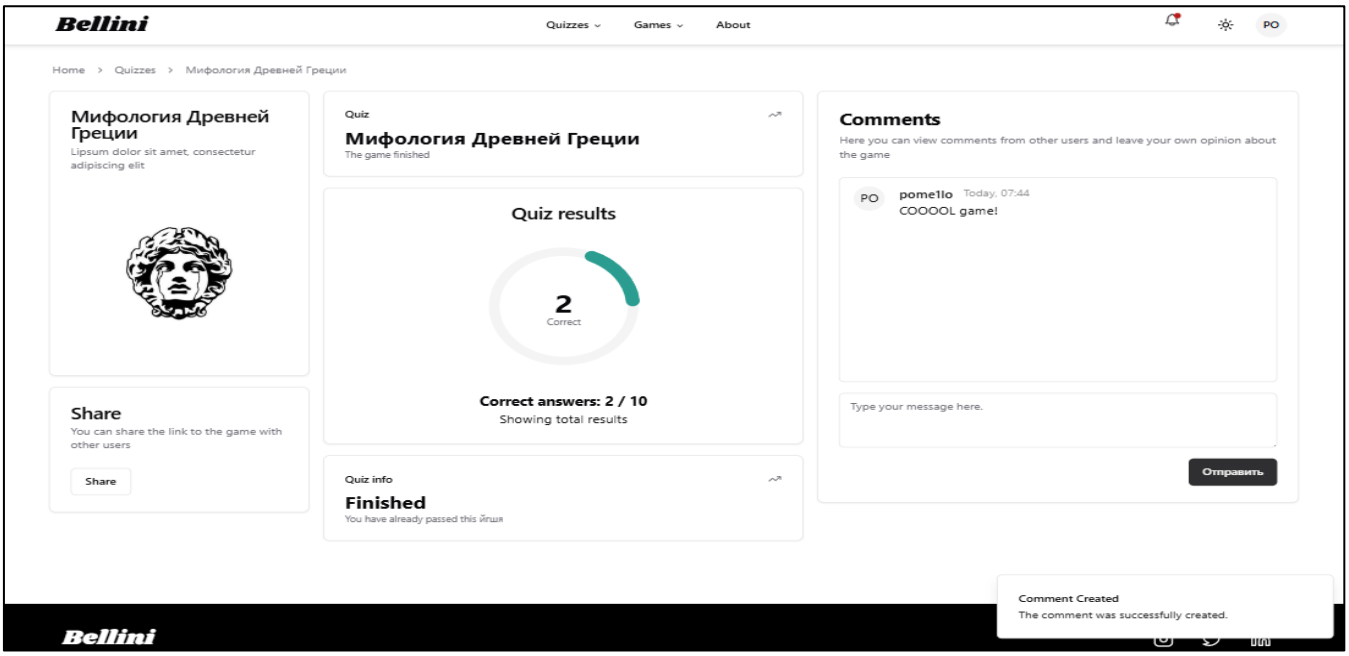

Рисунок 5.17 – Страница результатов викторины со списком комментариев

В списке комментариев отображены комментарии, оставленные всеми пользователями, прошедшими викторину.

5.16 Создание вопросов и вариантов ответов для конкретной игры

Для того чтобы создать вопросы и варианты ответов в игровой комнате пользователю необходимо авторизироваться в системе и перейти на страницу игр. В правом верхнем углу обнаружить кнопку «*Create game*» и нажать на неё. После чего перед пользователем отобразится диалоговое окно, в котором необходимо указать все необходимые параметры для создания игровой комнаты. После чего необходимо найти созданную игру в списке игр. Кликнув на неё перейти в игровую комнату. Для создателя игры (ведущего), слева будет доступен блок, в котором отображен список комментариев. По нажатию кнопки «*Create question*» открывается диалоговое окно для создания вопроса и 4 вариантов ответов.

Диалоговое окно для создания вопроса представлено на рисунке 5.18.

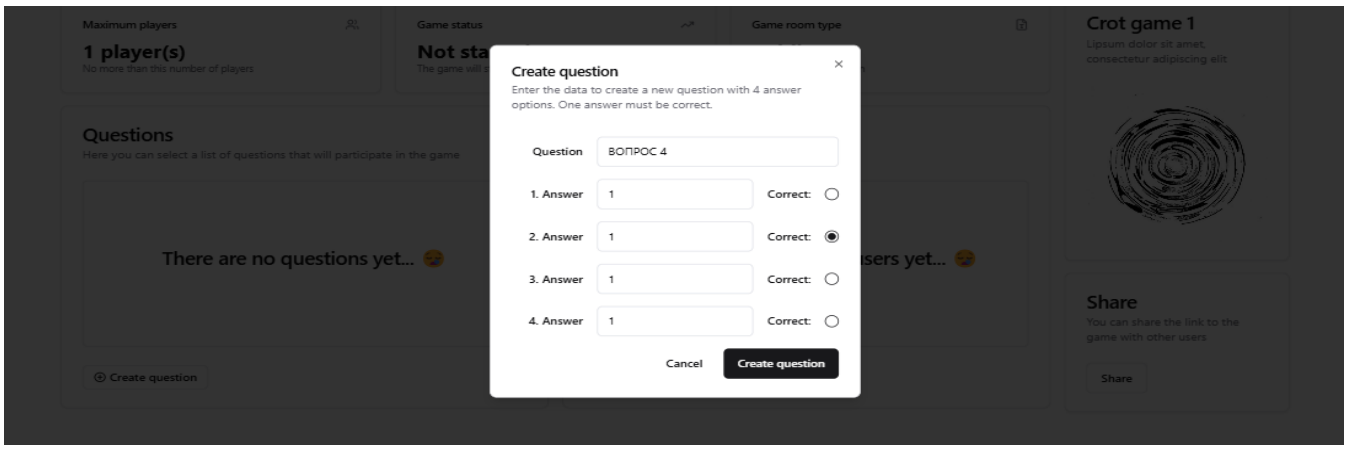

Рисунок 5.18 – Диалоговое окно для создания вопроса

Здесь пользователю необходимо ввести текст вопроса и текст для четырех вариантов ответа. А также, выбрать отметить один вариант ответа как верный.

5.17 Запуск игры

Для создания игровой комнаты пользователю необходимо пройти процесс авторизации и перейти на страницу игр. В правом верхнем углу обнаружить кнопку «*Create game*» и нажать на неё. После чего перед пользователем отобразится диалоговое окно, в котором необходимо указать все необходимые параметры для создания игровой комнаты. После ввода всех необходимых параметров игра будет успешно создана. Новая игра отобразиться в списке среди других игр. После чего необходимо кликнуть на неё и перейти в игровую комнату. Для создателя игры (ведущего) в правом верхнем углу экрана будет доступна кнопка «*Start game*».

Вид игровой комнаты для ведущего вместе с кнопкой «Start game» представлен на рисунке 5.19.

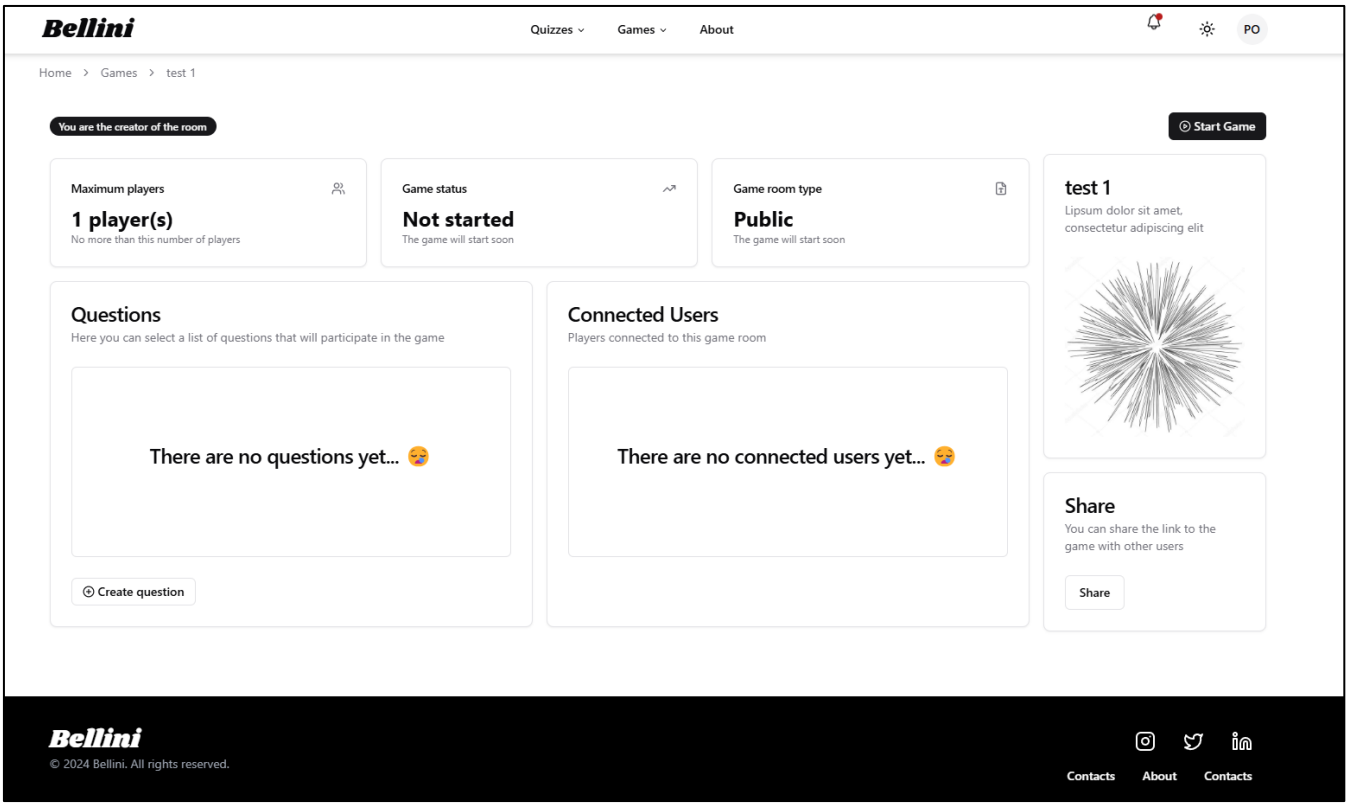

Рисунок 5.19 – Вид игровой комнаты от имени ведущего с кнопкой «Start game»

5.18 Переход к следующему вопросу

Для создания игровой комнаты пользователю необходимо пройти процесс авторизации и перейти на страницу игр. В правом верхнем углу обнаружить кнопку «*Create game*» и нажать на неё. После чего перед пользователем отобразится диалоговое окно, в котором необходимо указать все необходимые параметры для создания игровой комнаты. После ввода всех необходимых параметров игра будет успешно создана. Новая игра отобразиться в списке среди других игр. После чего необходимо кликнуть на неё и перейти в игровую комнату. Для создателя игры

(ведущего) в правом верхнем углу экрана будет доступна кнопка «*Start game*». По нажатии на неё начнется обратный отсчет, по истечению которого появится вопрос и 4 варианта ответа. Сверху будет доступна кнопка «*Next*», по нажатию на которую произойдет переход к следующему вопросу. Кнопка «*Next*» для перехода к следующему вопросу отображена на рисунке 5.13.

5.19 Завершение игры

Для завершения игры пользователю необходимо обладать ролью «Ведущий» (быть создателем игровой комнаты) и находиться в уже запущенной игре. Сверху, на ровне с кнопкой «*Next*», по нажатию на которую произойдет переход к следующему вопросу, будет доступна красная кнопка «*End* Game». Отображение кнопки «End game» представлено на рисунке 5.20.

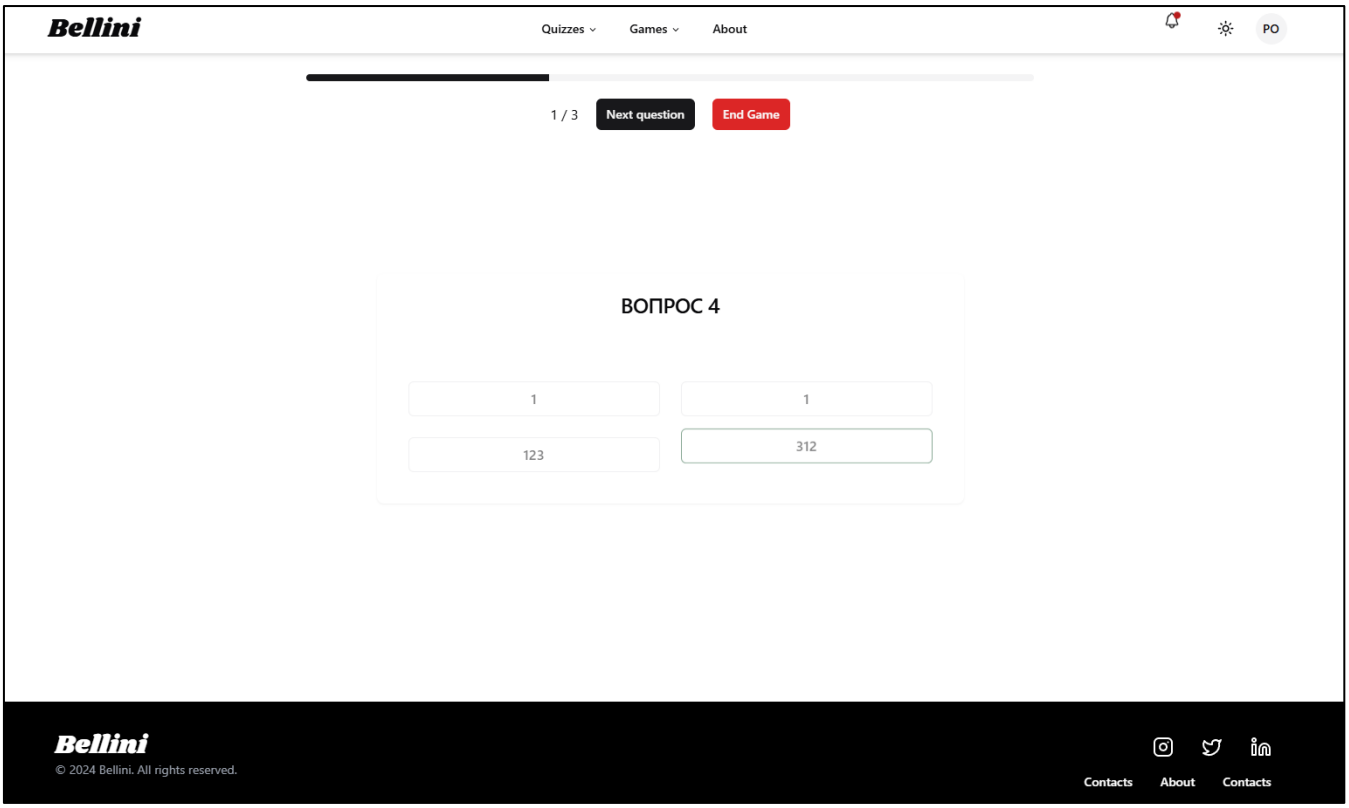

Рисунок 5.20 – Отображение кнопки «End Game»

По нажатию красной кнопки «End Game» игра завершается.

5.20 Удаление комментариев

Для удаления комментариев у завершенной игры пользователю необходимо обладать ролью «Ведущий» (быть создателем игровой комнаты) и находиться в уже завершенной игре. Здесь отображены результаты игры и список комментариев. Для создателя игры будут доступны кнопки «*Delete*» на против каждого комментария, по нажатию на которые происходит удаление комментария.

Отображение кнопки «*Delete*» для удаления комментариев в завершенной игре представлено на рисунке 3.21.

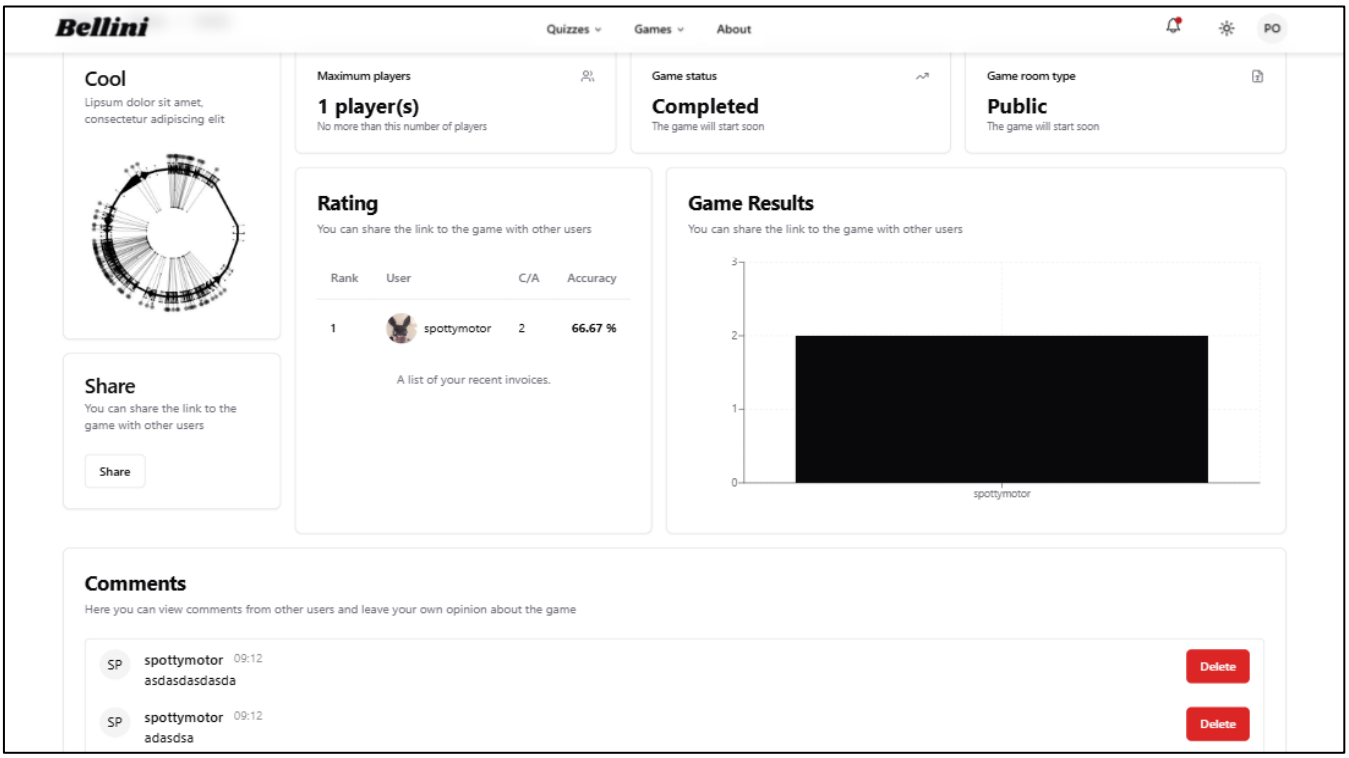

Рисунок 3.21 – Отображение кнопки «Delete» для удаления комментариев

5.21 Выводы по разделу

1. Руководство пользователя охватывает 20 функций приложения. Включены основные операции, такие как регистрация, авторизация, просмотр профилей, рейтингов, игр и викторин, а также создание игровых комнат, вопросов и ответов.

2. В разделе представлены подробные инструкции для каждой функции системы, позволяющие пользователю с любым уровнем подготовки эффективно взаимодействовать с приложением.

3. Руководство учитывает обработку ошибок, предоставляя рекомендации по устранению наиболее часто встречающихся проблем.

Заключение

В ходе данного курсового проекта было разработано многофункциональное *web*-приложение «Социальная сеть с возможностью создания и проведения интеллектуальных игр и викторин».

1. *Web*-приложение поддерживает 3 роли: гость, пользователь, ведущий.
2. В ходе рассмотрения аналогов разрабатываемого приложения были выявлены основные функции и элементы.
3. Реализовано 20 функций.
4. База данных включает 14 таблиц.
5. Использована *REST*-архитектура, обеспечивающая удобное взаимодействие клиента и сервера через *HTTP*. Основные компоненты: Модели данных: описывают структуру и связи таблиц базы данных. Контроллеры: реализуют CRUD-операции. Сервисы: выполняют бизнес-логику приложения. Поддержка двусторонней связи через *SignalR* для обновлений в реальном времени.
6. Объем кода: объем пользовательского кода составил более 10000 строк кода.
7. Было разработано 20 тестов, которые покрыли 100% функционала *web*-приложения.

В результате разработки были выполнены все поставленные задачи, в том числе:

– поддерживать роли гость, пользователь и ведущий;

– гость: регистрация, просмотр информации о доступных викторинах, просмотр информации о существующих играх, просмотр рейтингов других игроков по прохождению викторин, просмотр профилей пользователей;

– пользователь: авторизация, просмотр информации о доступных викторинах, просмотр информации о существующих играх, просмотр профилей пользователей, просмотр рейтингов других игроков по прохождению викторин, создание игровых комнат, редактирование личной информации, присоединение к играм, прохождение викторин, просмотр результатов прохождения викторин, принятие участия в играх, просмотр результатов игр и рейтинга игроков в игровой комнате, размещение комментариев к завершенной игре, размещение комментариев к пройденной викторине;

– ведущий: авторизация, создание вопросов и вариантов ответов для конкретной игры, запуск игры, переход к следующему вопросу, завершение игры, размещение комментариев к завершенной игре, удаление комментариев.

Список используемых источников

1. Quiz [Электронный ресурс] / Режим доступа: https://quiz.com – Дата доступа: 07.10.2024
2. Kahoot! [Электронный ресурс] / Режим доступа: https://kahoot.com – Дата доступа: 08.10.2024
3. Jackbox Games [Электронный ресурс] / Режим доступа: https://www.jackboxgames.com/ – Дата доступа: 09.10.2024
4. Windows Server 2022 [Электронный ресурс]. – Режим доступа: https://www.microsoft.com/en-us/evalcenter/download-windows-server-2022 – Дата доступа: 10.10.2024
5. React [Электронный ресурс] / Режим доступа: https://react.dev – Дата доступа: 10.10.2024
6. ASP.NET Core Web API [Электронный ресурс] / Режим доступа: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-9.0 – Дата доступа: 10.10.2024
7. Entity Framework [Электронный ресурс] / Режим доступа: https://learn.microsoft.com/en-us/ef/ – Дата доступа: 13.12.2024
8. MS SQL Server [Электронный ресурс] / Режим доступа: https://www.microsoft.com/en-us/sql-server – Дата доступа: 14.10.2024
9. C# [Электронный ресурс] / Режим доступа: https://learn.microsoft.com/en-us/dotnet/csharp/ – Дата доступа: 14.10.2024
10. HTTP 1.1 [Электронный ресурс] / Режим доступа: https://www.w3.org/Protocols/ – Дата доступа: 15.10.2024
11. Debian [Электронный ресурс]. – Режим доступа: https://www.debian.org/releases/bookworm/ – Дата доступа: 15.10.2024
12. Docker [Электронный ресурс]. – Режим доступа: https://docs.docker.com – Дата доступа: 15.10.2024
13. Microsoft Edge [Электронный ресурс] / Режим доступа: www.microsoft.com/en-us/edge – Дата доступа: 16.10.2024
14. Server-Sent Events (SSE) [Электронный ресурс] / Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events – Дата доступа: 16.10.2024
15. TCP [Электронный ресурс] / Режим доступа: https://www.rfc-editor.org/rfc/rfc793 – Дата доступа: 16.10.2024
16. Redis [Электронный ресурс] / Режим доступа: https://redis.io – Дата доступа: 16.10.2024
17. Vite [Электронный ресурс] / Режим доступа: https://vite.dev – Дата доступа: 17.10.2024
18. Node.js [Электронный ресурс] / Режим доступа: https://nodejs.org/en – Дата доступа: 17.10.2024
19. Nginx [Электронный ресурс] / Режим доступа: https://nginx.org – Дата доступа: 17.10.2024
20. SignalR [Электронный ресурс] / Режим доступа: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-9.0 – Дата доступа: 17.10.2024

21. Репозиторий паттерн [Электронный ресурс] / Режим доступа: https://learn.microsoft.com/ru-ru/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application – Дата доступа: 22.10.2024

22. Data Transfer Objects(DTOs) [Электронный ресурс] / Режим доступа: https://learn.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5 – Дата доступа: 23.10.2024

23. FluentValidation [Электронный ресурс] / Режим доступа: https://docs.fluentvalidation.net/en/latest/ – Дата доступа: 23.10.2024

24. Dependency injection [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/dependency-injectiondi-design-pattern/– Дата доступа: 24.10.2024

25. Ocelot [Электронный ресурс] / Режим доступа: https://ocelot.readthedocs.io/en/latest/ – Дата доступа: 25.10.2024

26. ApiGateway паттерн [Электронный ресурс] / Режим доступа: https://dotnetfullstackdev.medium.com/api-gateway-in-net-microservice-architecture-411cdf52c22d – Дата доступа: 29.10.2024

27. MimeKit [Электронный ресурс] / Режим доступа: https://github.com/jstedfast/MimeKit – Дата доступа: 02.11.2024

28. JwtBearer [Электронный ресурс] / Режим доступа: https://learn.microsoft.com/dotnet/api/microsoft.aspnetcore.authentication.jwtbearer?view=aspnetcore-9.0 – Дата доступа: 03.11.2024

29. StackExchange.Redis [Электронный ресурс] / Режим доступа: https://stackexchange.github.io/StackExchange.Redis/ – Дата доступа: 04.11.2024

30. BCrypt.Net-Next [Электронный ресурс] / Режим доступа: https://github.com/BcryptNet/bcrypt.net – Дата доступа: 04.11.2024

31. TypeScript [Электронный ресурс] / Режим доступа: https://www.typescriptlang.org – Дата доступа: 05.11.2024

32. Shadcn UI [Электронный ресурс] / Режим доступа: https://ui.shadcn.com – Дата доступа: 05.11.2024

33. Tailwind CSS [Электронный ресурс] / Режим доступа: https://tailwindcss.com – Дата доступа: 05.11.2024

34. React Router DOM [Электронный ресурс] / Режим доступа: https://reactrouter.com/home – Дата доступа: 05.11.2024

35. AOS (Animate On Scroll) [Электронный ресурс] / Режим доступа: https://michalsnik.github.io/aos/ – Дата доступа: 05.11.2024

ПРИЛОЖЕНИЕ А Скрипт создания базы данных

```
CREATE DATABSE BELLINI;
CREATE TABLE [dbo].[AnswerOptions](
     [Id] [int] IDENTITY(1,1) NOT NULL,
     t] [bit] NOT NULL,
     [QuestionId] [int] NOT NULL,
 CONSTRAINT [PK_AnswerOptions] PRIMARY KEY CLUSTERED ([Id] ASC)
);

ALTER TABLE [dbo].[AnswerOptions] ADD CONSTRAINT
[FK_AnswerOptions_Questions_QuestionId]
FOREIGN KEY([QuestionId]) REFERENCES [dbo].[Questions] ([Id]) ON
DELETE CASCADE;

CREATE TABLE [dbo].[GameComments](
     [Id] [int] IDENTITY(1,1) NOT NULL,
     [GameId] [int] NOT NULL,
     [UserId] [int] NOT NULL,
       NOT NULL,
     [UsernamLL,
     [ProfileImageUrl] [nvarchar](max) NOT NULL,
       NOT NULL,
 CONSTRAINT [PK_GameComments] PRI)
);

ALTER TABLE [dbo].[GameComments] ADD CONSTRAINT
[FK_GameComments_Games_GameId]
FOREIGN KEY([GameId]) REFERENCES [dbo].[Games] ([Id]) ON DELETE
CASCADE;

CREATE TABLE [dbo].[GameResults](
     [Id] [int] IDENTITY(1,1) NOT NULL,
     [NumberOfCorrectAnswers] [int] NOT NULL,
     [NumberOfQuestions] [int] NOT NULL,
     [UserId] [int] NOT NULL,
     [GameId] [int] NOT NULL,
 CONSTRAINT [PK_GameResults] PRIMARY KEY CLUSTERED ([Id] ASC)
);

ALTER TABLE [dbo].[GameResults] ADD CONSTRAINT
[FK_GameResults_Games_GameId]
FOREIGN KEY([GameId]) REFERENCES [dbo].[Games] ([Id]) ON DELETE
CASCADE;

ALTER TABLE [dbo].[GameResults] ADD CONSTRAINT
[FK_GameResults_Users_UserId]
FOREIGN KEY([UserId]) REFERENCES [dbo].[Users] ([Id]) ON DELETE
CASCADE;

CREATE TABLE [dbo].[Games](
     [Id] [int] IDENTITY(1,1) NOT NULL,
       NOT NULL,
     [HostId] [int] NOT NULL,
```

```
       NOT NULL,
       NOT NULL,
       NOT NOT NULL,
     [GameCoverImageUrl] [nvarchate] [bit] NOeStatusId] [NT
[PK_Games] PRIMARY KEY CLUSTERED ([Id] ASC)
);

ALTER TABLE [dbo].[Games] ADD CONSTRAINT
[FK_Games_GameStatuses_GamtatusId]) REFERENCES [dbo].[GameStatuses]
([Id]);

CREATE TABLE [dbo].[GameStatuses](
     [Id] [int] NOT NULL,
       NOT NULL,
 CONSTRAINT [PK_GameStatuses] PRIMARY KEY CLUSTERED ([Id] ASC)
);

CREATE TABLE [dbo].[Notifications](
     [Id] [int] IDENTITY(1,1) NOT NULL,
     [UserId] [int] NOT NULL,
     [Title] [nvarchar](100) NOT NULL,
       NOT NULL,
       NOT NULL,
     [IsRead] [bit] NOT NULL,
 CONSTRAINT [PK_Notifications] PRIMARY KEY CLUSTERED ([Id] ASC)
);

ALTER TABLE [dbo].[Notifications] ADD CONSTRAINT
[FK_Notifications_Users_UserId]
FOREIGN KEY([UserId]) REFERENCES [dbo].[Users] ([Id]) ON DELETE
CASCADE;

CREATE TABLE [dbo].[Players](
     [Id] [int] IDENTITY(1,1) NOT NULL,
     [GameId] [int] NOT NULL,
     [Name] [nvarchar](100) NOT NULL,
     [Score] [int] NOT NULL,
     [UserId] [int] NOT NULL,
 CONSTRAINT [PK_Players] PRIMARY KEY CLUSTERED ([Id] ASC)
);

ALTER TABLE [dbo].[Players] ADD CONSTRAINT [FK_Players_Games_GameId]
FOREIGN KEY([GameId]) REFERENCES [dbo].[Games] ([Id]) ON DELETE
CASCADE;

ALTER TABLE [dbo].[Players] ADD CONSTRAINT [FK_Players_Users_UserId]
FOREIGN KEY([UserId]) REFERENCES [dbo].[Users] ([Id]);

CREATE TABLE [dbo].[Questions](
     [Id] [int] IDENTITY(1,1) NOT NULL,
     [Text] [nvarchar](500) NOT NULL,
     [GameId] [int] NOT NULL,
     [IsCustom] [bit] NOT NULL,
 CONSTRAINT [PK_Questions] PRIMARY KEY CLUSTERED ([Id] ASC)
```

```
);

ALTER TABLE [dbo].[Questions] ADD CONSTRAINT
[FK_Questions_Games_GameId]
FOREIGN KEY([GameId]) REFERENCES [dbo].[Games] ([Id]) ON DELETE
CASCADE;

CREATE TABLE [dbo].[QuizAnswerOptions](
     [Id] [int] IDENTITY(1,1) NOT NULL,
     [Text] [nvarchar](200) NOT NULL,
     [IsCorrect] [bit] NOT NULL,
     [QuizQuestionId] [int] NOT NULL,
 CONSTRAINT [PK_QuizAnswerOptions] PRIMARY KEY CLUSTERED ([Id] ASC)
);

ALTER TABLE [dbo].[QuizAnswerOptions] ADD CONSTRAINT
[FK_QuizAnswerOptions_QuizQuestions_QuizQuestionId]
FOREIGN KEY([QuizQuestionId]) REFERENCES [dbo].[QuizQuestions]
([Id]) ON DELETE CASCADE;

CREATE TABLE [dbo].[QuizComments](
     [Id] [int] IDENTITY(1,1) NOT NULL,
     [QuizId] [int] NOT NULL,
     [UserId] [int] NOT NULL,
     [Content] [nvarchar](500) NOT NULL,
     [Username] [nvarchar](max) NOT NULL,
     [ProfileImageUrl] [nvarchar](max) NOT NULL,
     [CommentDate] [datetime2](7) NOT NULL,
 CONSTRAINT [PK_QuizComments] PRIMARY KEY CLUSTERED ([Id] ASC)
);

ALTER TABLE [dbo].[QuizComments] ADD CONSTRAINT
[FK_QuizComments_Quizzes_QuizId]
FOREIGN KEY([QuizId]) REFERENCES [dbo].[Quizzes] ([Id]) ON DELETE
CASCADE;


CREATE TABLE [dbo].[QuizQuestions](
  [Id] [int] IDENTITY(1,1) NOT NULL,
  [Text] [nvarchar](500) NOT NULL,
  [QuizId] [int] NOT NULL,
  CONSTRAINT [PK_QuizQuestions] PRIMARY KEY CLUSTERED
(
  [Id] ASC
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY];

ALTER TABLE [dbo].[QuizQuestions]  WITH CHECK ADD  CONSTRAINT
[FK_QuizQuestions_Quizzes_QuizId] FOREIGN KEY([QuizId])
REFERENCES [dbo].[Quizzes] ([Id])
ON DELETE CASCADE;
```

```
ALTER TABLE [dbo].[QuizQuestions] CHECK CONSTRAINT
[FK_QuizQuestions_Quizzes_QuizId];


CREATE TABLE [dbo].[QuizResults](
  [Id] [int] IDENTITY(1,1) NOT NULL,
  [NumberOfCorrectAnswers] [int] NOT NULL,
  [NumberOfQuestions] [int] NOT NULL,
  [UserId] [int] NOT NULL,
  [EndTime] [datetime2](7) NOT NULL,
  [QuizId] [int] NOT NULL,
  CONSTRAINT [PK_QuizResults] PRIMARY KEY CLUSTERED
(
  [Id] ASC
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY];

ALTER TABLE [dbo].[QuizResults]  WITH CHECK ADD  CONSTRAINT
[FK_QuizResults_Quizzes_QuizId] FOREIGN KEY([QuizId])
REFERENCES [dbo].[Quizzes] ([Id])
ON DELETE CASCADE;

ALTER TABLE [dbo].[QuizResults] CHECK CONSTRAINT
[FK_QuizResults_Quizzes_QuizId];

ALTER TABLE [dbo].[QuizResults]  WITH CHECK ADD  CONSTRAINT
[FK_QuizResults_Users_UserId] FOREIGN KEY([UserId])
REFERENCES [dbo].[Users] ([Id])
ON DELETE CASCADE;

ALTER TABLE [dbo].[QuizResults] CHECK CONSTRAINT
[FK_QuizResults_Users_UserId];


CREATE TABLE [dbo].[Quizzes](
  [Id] [int] IDENTITY(1,1) NOT NULL,
  [GameName] [nvarchar](200) NOT NULL,
  [GameCoverImageUrl] [nvarchar](500) NOT NULL,
  [StartTime] [datetime2](7) NOT NULL,
  [EndTime] [datetime2](7) NOT NULL,
  CONSTRAINT [PK_Quizzes] PRIMARY KEY CLUSTERED
(
  [Id] ASC
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY];


CREATE TABLE [dbo].[Users](
  [Id] [int] IDENTITY(1,1) NOT NULL,
  [Email] [nvarchar](100) NOT NULL,
```

```
  [Username] [nvarchar](50) NULL,
  [Password] [nvarchar](100) NULL,
  [IsEmailVerified] [bit] NOT NULL,
  [IsActive] [bit] NOT NULL,
  [FirstName] [nvarchar](50) NULL,
  [LastName] [nvarchar](50) NULL,
  [ProfileImageUrl] [nvarchar](255) NULL,
  CONSTRAINT [PK_Users] PRIMARY KEY CLUSTERED
(
  [Id] ASC
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY];
```

ПРИЛОЖЕНИЕ Б Содержимое файла App.tsx

```
function App() {
    const [gameStarted, setGameStarted] = useState(false);
    const [quizStarted, setQuizStarted] = useState(false);
    const [gameFinished, setGameFinished] = useState(false);
    const [quizFinished, setQuizFinished] = useState(false);
    const [currentStartedGame, setCurrentStartedGame] =
useState<StartedGame>();
    const [currentStartedQuiz, setCurrentStartedQuiz] =
useState<StartedQuiz>();
    const [currentFinishedGame, setCurrentFinishedGame] =
useState<FinishedGame>();
    const [currentFinishedQuiz, setCurrentFinishedQuiz] =
useState<Quiz>();

    useEffect(() => AOS.init, []);

    const handleStart = (game: StartedGame) => {
        setCurrentStartedGame(game);
        setGameStarted(true);
    };

    const handleFinish = (game: FinishedGame) => {
        setCurrentFinishedGame(game);
        setGameFinished(true);
        setGameStarted(false);
    };

    const handleQuizStart = (quiz: StartedQuiz) => {
        setCurrentStartedQuiz(quiz);
        setQuizStarted(true);
    };

    const handleQuizFinish = (quiz: Quiz) => {
        setCurrentFinishedQuiz(quiz);
        setQuizFinished(true);
        setQuizStarted(false);
    };


    return (
        <>
            <AuthProvider>
                <Toaster/>
                <ThemeProvider defaultTheme="dark" storageKey="vite-
ui-theme">
                    <Routes>
                        <Route path='/' element={<BasicLayout/>}>
                            <Route path='settings'
element={<SettingsPage/>}/>
                            <Route path='contacts'
element={<ContactsPage/>}/>
```

```
                                <Route path='about'
element={<AboutPage/>}/>
                                <Route path='profile/:id'
element={<ProfilePage/>}/>
                                <Route path='games'
element={<GameListPage/>}/>
                                <Route path='games/:tabName'
element={<GameListPage/>}/>
                                <Route path='quizzes'
element={<QuizzesListPage/>}/>
                                <Route path='quizzes/:tabName'
element={<QuizzesListPage/>}/>
                                <Route path='notifications'
element={<NotifiactionsPage/>}/>

                                <Route element={<PrivateRoute/>}>

                                </Route>
                            </Route>
                            <Route path='/' element={<GameLayout/>}>
                                <Route index element={<MainPage/>}/>
                                <Route path='login'
element={<LoginPage/>}/>
                                <Route path='register'
element={<RegisterPage/>}/>
                                <Route path='forgot-password'
element={<ForgotPasswordPage/>}/>

                                <Route path='games/room/:id' element={
                                    gameFinished ? (
                                        currentFinishedGame ? (
                                            <GameFinishedPage
currentGame={currentFinishedGame}/>
                                        ) : (
                                            <div> Loading...
GameFinishedPage</div>  //todo add Skeleton
                                        )
                                    ) : gameStarted ? (
                                        currentStartedGame ? (
                                            <GameStartedPage
currentGame={currentStartedGame} onFinish={handleFinish}/>
                                        ) : (
                                            <div>Loading...
GameStartedPage</div>  //todo add Skeleton
                                        )
                                    ) : (
                                        <GameRoomPage
onStart={handleStart} isFinished={setGameFinished}

onFinish={handleFinish}/>
                                    )
                                }/>

                                <Route path='quizzes/room/:id' element={
```

```
                                quizFinished ? (
                                    currentFinishedQuiz ? (
                                        <QuizFinishedPage
currentQuiz={currentFinishedQuiz}/>
                                    ) : (
                                        <div> Loading...
QuizFinishedPage</div>  //todo add Skeleton
                                    )
                                ) : quizStarted ? (
                                    currentStartedQuiz ? (
                                        <QuizStartedPage
currentQuiz={currentStartedQuiz}

onQuizFinish={handleQuizFinish}/>
                                    ) : (
                                        <div>Loading...
QuizStartedPage</div>  //todo add Skeleton
                                    )
                                ) : (
                                    <QuizRoomPage
onQuizStart={handleQuizStart} isQuizFinished={setQuizFinished}

onQuizFinish={handleQuizFinish}/>
                                )
                            }/>

                        </Route>
                        <Route path="500"
element={<InternalServerErrorPage/>}/>
                        <Route path="404"
element={<NotFoundPage/>}/>
                        <Route path="*" element={<NotFoundPage/>}/>
                    </Routes>
                </ThemeProvider>
            </AuthProvider>
        </>
    )
}

export default App
```

ПРИЛОЖЕНИЕ В Содержимое компонента QuizStartedPage

```
import React, {useEffect, useState} from "react";
import {StartedQuiz} from "@/utils/interfaces/StartedQuiz.ts";
import {Button} from "@/components/ui/button.tsx";
import {Progress} from "@/components/ui/progress";
import {useAuth} from "@/utils/context/authContext.tsx";
import {serverFetch} from "@/utils/fetchs/serverFetch.ts";
import {toast} from "@/components/ui/use-toast.ts";
import {useNavigate, useParams} from "react-router-dom";
import {Quiz} from "@/utils/interfaces/Quiz.ts";
interface QuizStartedPageProps {
    currentQuiz?: StartedQuiz;
    onQuizFinish: (quiz: Quiz) => void;
}
export const QuizStartedPage: React.FC<QuizStartedPageProps> =
({currentQuiz, onQuizFinish}) => {
    const navigate = useNavigate();
    const {id} = useParams();
    const {isAuthenticated, user} = useAuth();
    const [countdown, setCountdown] = useState(3);
    const [showQuestion, setShowQuestion] = useState(false);
    const [currentQuestionIndex, setCurrentQuestionIndex] =
useState(0);
    const [selectedAnswer, setSelectedAnswer] = useState<number |
null>(null);
    const [userAnswers, setUserAnswers] = useState<{ questionId:
number; answerId: number }[]>([]);
    useEffect(() => {
        if (countdown > 0) {
            const timer = setTimeout(() => setCountdown(countdown -
1), 1000);
            return () => clearTimeout(timer);
        } else {
            setShowQuestion(true);
        }
    }, [countdown]);
    const handleAnswerSelect = (index: number) => {
        setSelectedAnswer(index);
        const updatedAnswers = [...userAnswers];
        updatedAnswers[currentQuestionIndex] = {
            questionId:
currentQuiz!.questions[currentQuestionIndex].id,
            answerId:
currentQuiz!.questions[currentQuestionIndex].answerOptions[index].id
        };
        setUserAnswers(updatedAnswers);
    };
    const handleNextQuestion = () => {
        setSelectedAnswer(null);
        setCurrentQuestionIndex(currentQuestionIndex + 1);
    };
    const handleFinishQuiz = async () => {
        try {
```

```
            if (!isAuthenticated || !user) {
                navigate('/login');
                return;
            }
            console.log("ANSWERS")
            console.log(userAnswers)
            const response = await serverFetch(`/quizzes/${id}/end`,
                method: "POST",
                headers: {"Content-Type": "application/json"},
                body: JSON.stringify({
                    userId: user.id,
                    userAnswers: userAnswers.map(answer => ({
                        questionId: answer.questionId,
                        answerId: answer.answerId,
                    })),
                }),
            });
            if (response.ok) {
                const responseData: Quiz = await response.json();
                onQuizFinish(responseData);
            } else {
                const responseData: unknown = await response.json();
                toast({
                    title: "Error",
                    description: (responseData as Error).message ||
"An error occurred.",
                    variant: "destructive",
                });
            }
        } catch (error: unknown) {
            toast({
                title: "Error",
                description: "An unexpected error occurred." +
(error as Error).message,
                variant: "destructive",
            });
        }
    };
    const progressValue = ((currentQuestionIndex + 1) /
currentQuiz?.questions.length!) * 100;
    return (
        <div className="quiz-container">
            {!showQuestion ? (
                <div className="countdown flex justify-center items-
center h-[78vh]">
                    <h1 className="animate-pulse text-[35rem] text-
roboto">{countdown}</h1>
                </div>
            ) : (
                <div className="question-section flex flex-col
items-center justify-center h-[78vh]">
                    <div className="absolute flex flex-wrap justify-
center items-center top-20 sm:w-1/2 w-[250px]">
```

```
                            <Progress value={progressValue}
className="h-2 mb-5"/>
                            <Button className="me-4"
onClick={currentQuestionIndex < currentQuiz!.questions.length - 1 ?
handleNextQuestion : handleFinishQuiz}
                                disabled={selectedAnswer === null}>
                                {currentQuestionIndex <
currentQuiz!.questions.length - 1 ? "Next" : "Finish"}
                            </Button>
                            <Button disabled={true} variant="outline">
                                {currentQuestionIndex + 1} /
{currentQuiz?.questions.length}
                            </Button>
                        </div>
                        <div className="flex flex-col items-center
lg:max-w-1/2">
                            <h1 className="text-xl w-full lg:w-2/3 mb-4
p-4 text-center font-
bold">{currentQuiz?.questions[currentQuestionIndex].text}</h1>
                            <div className="flex items-center sm:w-full
flex-wrap w-3/4">
{currentQuiz?.questions[currentQuestionIndex].answerOptions.map((opt
ion, index) => (
                               <Button className="m-2 sm:m-3 w-full text-md"
                                       variant={selectedAnswer !== null
&& selectedAnswer === index ? "default" : "outline"}
                                       key={index}
                                       onClick={() =>
handleAnswerSelect(index)}
                                       disabled={selectedAnswer !==
null}
                                         >
                                       {option.text}
                                   </Button>
                                ))}
                            </div>
                        </div>
                    </div>
                )}
            </div>
);};
```

ПРИЛОЖЕНИЕ Г Содержимое компонента GameStartedPage

```
import React, {useEffect, useState} from "react";
import {StartedGame} from "@/utils/interfaces/StartedGame.ts";
import {Button} from "@/components/ui/button";
import {Card, CardHeader, CardTitle, CardContent} from
"@/components/ui/card";
import {useAuth} from "@/utils/context/authContext.tsx";
import {Progress} from "@/components/ui/progress";
import {HubConnectionBuilder} from "@microsoft/signalr";
import * as signalR from "@microsoft/signalr";
import {Player} from "@/utils/interfaces/Player.ts";
import {FinishedGame} from "@/utils/interfaces/FinishedGame.ts";
import {serverFetch} from "@/utils/fetchs/serverFetch.ts";
interface GameStartedPageProps {
    currentGame?: StartedGame;
    onFinish: (game: FinishedGame) => void;
}
export const GameStartedPage: React.FC<GameStartedPageProps> =
({currentGame, onFinish}) => {
    const [connection, setConnection] = useState<any>(null);
    const [countdown, setCountdown] = useState(3);
    const [showQuestion, setShowQuestion] = useState(false);
    const [selectedAnswer, setSelectedAnswer] = useState<number |
null>(null);
    const [fadeIn, setFadeIn] = useState(false);
    const [currentQuestionIndex, setCurrentQuestionIndex] =
useState(0);
    const [userAnswers, setUserAnswers] = useState<{ questionId:
number; answerId: number }[]>([]);
    const {user} = useAuth();
    useEffect(() => {
        const newConnection = new HubConnectionBuilder()
            .withUrl("https://localhost:7292/gameHub", {
                transport:
signalR.HttpTransportType.ServerSentEvents,
                withCredentials: true
            })
            .withAutomaticReconnect()
            .build();
        setConnection(newConnection);
        if (countdown > 0) {
            const timer = setTimeout(() => {
                setCountdown((prev) => prev - 1);
            }, 1000);
            return () => clearTimeout(timer);
        } else {
            setShowQuestion(true);
            setTimeout(() => setFadeIn(true), 100);
        }
    }, [countdown]);
    useEffect(() => {
        if (connection) {
            connection.start().then(() => {
```

```
                console.log("Connected to SignalR hub");
                connection.invoke("JoinRunningGame",
currentGame?.id.toString())
                    .then(() => {
                        console.log("Joined running game room");
                    })
                    .catch((error: unknown) => {
                        console.error("Error joining running game:",
error);});});
                connection.on("NextQuestion", (nextQuestionIndex:
number) => {
                    setCurrentQuestionIndex(nextQuestionIndex);
                    alert(selectedAnswer);
                    setSelectedAnswer(null);
                    setFadeIn(true);
                });
                connection.on("GameCompleted", (gameId: string,
game: FinishedGame) => {
                    onFinish(game);
                    console.log("GameCompleted with id " + gameId);
                });
                connection.on("PlayersList", (playerList: Player[])
=> {console.log("Players in the game:", playerList);
                });
            }).catch((error: unknown) => {
                console.error("Connection failed:", error);
            });
            connection.onclose(async () => {
                console.warn("Connection closed, attempting to
reconnect...");
                try {
                    await connection.start();
                    console.log("Reconnected to the SignalR hub");
                } catch (error) {
                    console.error("Reconnection failed:", error);
                }
            });
        }
    }, [connection]);
    const handleSubmitAnswers = async (updatedAnswers: { questionId:
number; answerId: number }[] = userAnswers) => {
        console.log(currentGame?.id.toString() + "\n" + user?.id +
"\n" + userAnswers);
        if (connection && connection.state === "Connected" &&
userAnswers.length > 0) {
            try {
                await connection.invoke("SubmitAnswers",
currentGame?.id.toString(), user?.id.toString(), updatedAnswers);
            } catch (error) {
                console.error("Error submitting answers:", error);
            }
        } else {
            console.warn("Cannot send data, connection is not
established");
```

```
        }
    };
    const handleAnswerSelect = (index: number) => {
        if (selectedAnswer === null) {
            setSelectedAnswer(index);
            const updatedAnswers = [...userAnswers];
            updatedAnswers[currentQuestionIndex] = {
                questionId:
currentGame!.questions[currentQuestionIndex].id,
                answerId:
currentGame!.questions[currentQuestionIndex].answerOptions[index].id
,};
            setUserAnswers(updatedAnswers);
            handleSubmitAnswers(updatedAnswers);
        }
    };
    const handleNextQuestion = async () => {
        if (connection && connection.state === "Connected") {
            try {
                //await connection.invoke("SubmitAnswers",
currentGame?.id.toString(), user?.id.toString(), userAnswers);
                await connection.invoke("NextQuestion",
currentGame?.id.toString(), currentQuestionIndex + 1);
            } catch (error) {
                console.error("Error invoking NextQuestion or
SubmitAnswers:", error);
            }
        } else {
            console.warn("Cannot send data, connection is not
established");
            if (connection.state === "Disconnected") {
                try {
                    await connection.start();
                    console.log("Reconnected to the server");
                    await connection.invoke("SubmitAnswers",
currentGame!.id.toString(), user?.id, userAnswers);
                    await connection.invoke("NextQuestion",
currentGame!.id, currentQuestionIndex + 1);
                } catch (error) {
                    console.error("Failed to reconnect and send
NextQuestion or SubmitAnswers:", error);
                }
            }
        }
    };
    const handleEndGame = async () => {
        if (connection) {
            //connection.invoke("EndGame",
currentGame!.id.toString());
            const response = await
serverFetch(`/game/${currentGame?.id}/end`, {
                method: 'POST',
                headers: {'Content-Type': 'application/json'},
            });
```

```
            if (response.ok) {
            }
            else {
                const data = await response.json();
                console.warn(data);
            }
        }
    };
    const totalQuestions = currentGame ?
currentGame.questions.length : 1;
    const progressValue = ((currentQuestionIndex + 1) /
totalQuestions) * 100;
    return (
        <>
            {currentGame ? (
                <>
                    {!showQuestion ? (
                        <div className="flex justify-center items-
center h-[78vh]">
                            <h1 className="animate-pulse text-
[35rem] text-roboto">{countdown}</h1>
                        </div>
                    ) : (
                        <div className="flex flex-col items-center
justify-center h-[80vh] space-y-6">
                            <div
                                className="absolute flex flex-wrap
justify-center items-center top-20 sm:w-1/2 w-[250px]">
                                <Progress value={progressValue}
className="h-2 mb-5"/>
                                {currentQuestionIndex + 1} /
{currentGame.questions.length}
                                {currentGame.hostId.toString() ==
user?.id ? (
                                    <div className="flex">
                                        {currentQuestionIndex <
currentGame.questions.length - 1 ?
                                            <Button
variant="default" className="ms-5" size="sm"

onClick={handleNextQuestion}>Next question</Button> : null}
<Button variant="destructive" className="ms-5" size="sm"
onClick={handleEndGame}>
                                            End Game
                                        </Button>
                                    </div>
                                ) : null}
                            </div>
                            <Card
                                className={`w-full max-w-2xl border-
0 bg-transparent transition-opacity duration-1000 ${
                                    fadeIn ? 'opacity-100' :
'opacity-0'}`}>
```

```
                                <CardHeader>
                                    <CardTitle>
                                        <h1 className="text-center
mb-10">

{currentGame.questions[currentQuestionIndex].text}
                                        </h1>
                                    </CardTitle>
                                </CardHeader>
                                <CardContent className="flex flex-
wrap sm:flex-row flex-col"
>{currentGame.questions[currentQuestionIndex].answerOptions.map((opt
ion, index) => (<div
                                                key={index}
                                                className={`flex items-
center w-full sm:w-1/2 transition-opacity duration-1000 ${
                                                    fadeIn ? 'opacity-
100' : 'opacity-0'
                                                }`}
                                            >
                                                <Button

variant={selectedAnswer === index ? "default" : "outline"}
                                                    onClick={() =>
handleAnswerSelect(index)}

disabled={selectedAnswer !== null || currentGame.hostId.toString()
== user?.id}
                                                    className={`m-2
sm:m-3 w-full text-md ${option.isCorrect &&
currentGame.hostId.toString() == user?.id ? "border-green-900
animate-bounce" : ""}`}
                                                >
                                                    {option.text}
                                                </Button>
                                            </div>
                                        ))}
                                </CardContent>
                            </Card>
                        </div>
                    )}
                </>
            ) : null}
        </>
    );
};
```