

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Иркутский государственный университет»
(ФГБОУ ВО «ИГУ»)
Институт математики и информационных технологий
Кафедра алгебраических и информационных систем

**КУРСОВАЯ РАБОТА ПО ДИСЦИПЛИНЕ
«РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЙ»**

РАЗРАБОТКА МОДУЛЕЙ ВЕБ-ПРИЛОЖЕНИЯ «Приложение для
организации викторин»

Студента 3 курса очного отделения
группы 02361–ДБ
Петров Ивана Алексеевича

Руководитель:
преподаватель
Попова Виктория Алексеевна

Иркутск 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1. Предметная область и технологии разработки	4
1.1. Анализ предметной области	4
1.2. Технологии разработки веб-приложения	5
2. Проектирование и разработка веб-приложения	7
2.1. Требования к проекту	7
2.2. Схема базы данных	7
2.3. Определение обработчиков маршрутов	8
2.4. Разработка компонентов	10
2.5. Реализованная функциональность	11
ЗАКЛЮЧЕНИЕ	15
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	16
ПРИЛОЖЕНИЕ 1. Код контроллера quizController.js	17
ПРИЛОЖЕНИЕ 2. Код контроллера questionsController.js	23

ВВЕДЕНИЕ

На протяжении курса «Разработка веб-приложений» мною были освоены и изучены ключевые технологии, такие как Node.js, шаблонизаторы, базы данных, защита информации, разработка технических заданий, а также знакомство с СУБД MySQL, ORM Sequelize, фреймворком Vue.js, и процессом развёртывания веб-приложений. Для наглядной демонстрации усвоенного материала было реализовано веб-приложение для организации викторин.

Целью данной работы является разработка веб-приложения для организации викторин.

Задачами работы являются:

- 1) Разработка базы данных;
- 2) Создание основных модулей для перехода и просмотра объектов;
- 3) Разработка добавления и редактирования объектов;

1. Предметная область и технологии разработки

1.1. Анализ предметной области

Веб-приложение представляет собой платформу для организации викторин, разработанную с целью предоставить пользователям возможность легко создавать, управлять и участвовать в интерактивных викторинах. Оно обеспечивает понятный интерфейс для проведения викторин, позволяя пользователям находить темы и тестировать свои знания.

Проблемы/задачи, которые решает веб-приложение:

- 1) Регистрация пользователей;
- 2) Предоставление возможности добавления викторин;
- 3) Возможность просматривать результаты прохождения викторин.

Система ориентирована на пользователей, которые имеют равные права в приложении. Все участники могут создавать, управлять и участвовать в викторинах, что способствует активному взаимодействию и сотрудничеству. Каждому пользователю предоставляется возможность добавлять новые вопросы, обновлять существующий контент и оценивать викторины, создавая таким образом динамичную и увлекательную среду для тестирования знаний.

Аналоги приложения для организации викторин:

- 1) **Kahoot!** [2]. Платформа для создания и проведения викторин и опросов в реальном времени. Пользователи могут легко разрабатывать интерактивные викторины с использованием различных типов вопросов, а также участвовать в играх, что делает процесс обучения увлекательным и динамичным.
- 2) **Quizlet** [7]. Сервис, который позволяет создавать карточки и викторины для изучения различных тем. Quizlet предлагает множество функций, включая возможность совместной работы, что позволяет пользователям обмениваться своими викторинами и учиться друг у друга.

- 3) **Mentimeter** [3]. Платформа для создания интерактивных презентаций, викторин и опросов, которая позволяет пользователям активно взаимодействовать с аудиторией. Участники могут отвечать на вопросы в режиме реального времени с помощью своих мобильных устройств, что делает процесс увлекательным и интуитивно понятным.

1.2. Технологии разработки веб-приложения

Из рассмотренных выше технологий разработки вытекают две возможные архитектуры, а именно: применение Node.js + SQLite + PUG + jQuery + Bootstrap, или Node.js + MySQL + Sequelize + Vue.js + Bootstrap. После рассмотрения выбранного проекта и построения плана, была выбрана вторая архитектура: Node.js + MySQL + Sequelize + Vue.js + Bootstrap.

Выбор Node.js [5] в качестве серверной технологии обусловлен его отличной производительностью и способностью эффективно обрабатывать множество одновременных запросов. Это позволит создать быстрый и отзывчивый интерфейс для пользователей, что особенно важно в ситуациях, когда на платформе одновременно находится большое количество пользователей.

Для разработки клиентской части приложения был выбран Vue.js [6], что связано с его высокой гибкостью и простотой в создании интерфейсов. Эта библиотека даст возможность разработать интуитивно понятный и удобный интерфейс, тем самым улучшая пользовательский опыт и облегчая взаимодействие с онлайн-библиотекой.

В качестве системы управления базами данных была выбрана MySQL [4] благодаря ее широкой поддержке и эффективным возможностям масштабирования. Она предоставляет значительную гибкость в управлении данными и обладает обширным функционалом, что позволяет эффективно работать с разнообразными типами информации.

Sequelize [8] в качестве ORM для MySQL упрощает взаимодействие с базой данных и предлагает мощные инструменты для работы с данными. Он обеспечивает возможность создания связей между таблицами, что способствует эффективной структуризации данных и оптимизации запросов при работе с различными объектами, такими как книги и авторы.

Для фронтенда используется Bootstrap [1], который позволяет быстро и легко разрабатывать адаптивный и стильный интерфейс, поддерживающий мобильные устройства и обеспечивающий консистентность пользовательского опыта.

Все перечисленные технологии были выбраны на основе необходимости гарантировать стабильную работу приложения, высокую производительность и удобство использования для пользователей, а также простоту разработки и поддержки в дальнейшем.

В базе данных основная таблица Quiz хранит информацию о викторинах и связана с другими сущностями через различные таблицы. Таблица Result фиксирует ответы пользователей на викторины и связывается как с таблицей Quiz, так и с таблицей User. Таким образом, структура базы данных организована для эффективного хранения и извлечения информации о викторинах, пользователях и их результатах, позволяя легко управлять связями между этими сущностями.

2. Проектирование и разработка веб-приложения

2.1. Требования к проекту

Для полноценной работы проекта требовалось реализовать следующую функциональность:

1. Регистрация пользователей;
2. Предоставление возможности добавления викторин;
3. Просмотр пользователем всех доступных викторин и возможность пройти их.
4. Возможность просматривать результаты прохождения викторин.

2.2. Схема базы данных

Схема базы данных к проекту «Приложения для организации викторин» представлена на рисунке 1.

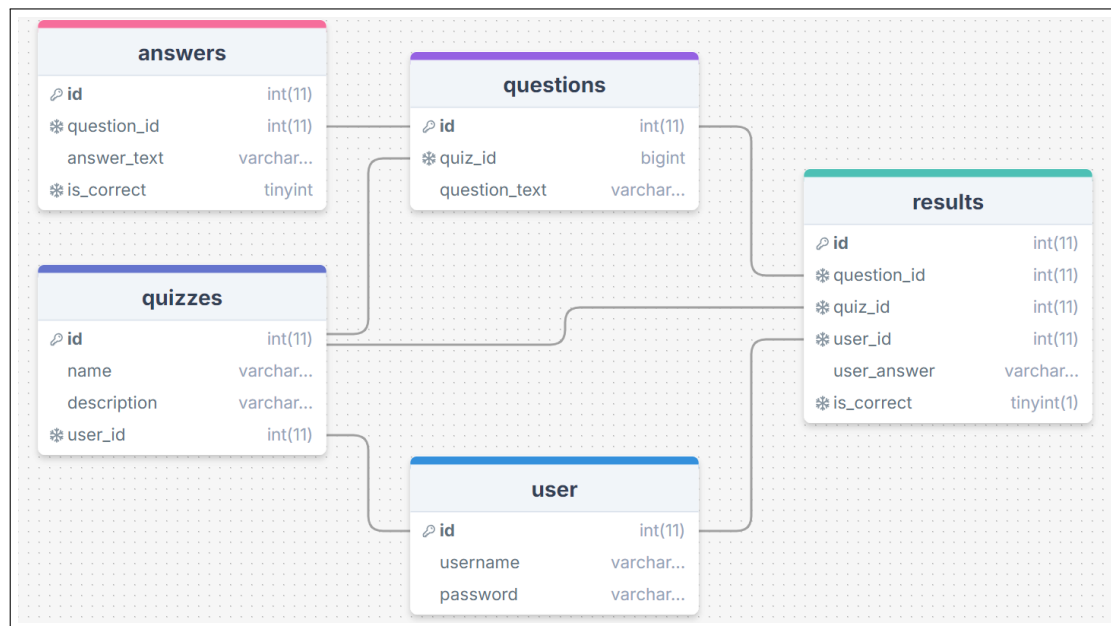


Рисунок 1. Схема базы данных

2.3. Определение обработчиков маршрутов

В данной работе реализованы следующие обработчики маршрутов:

- `\user\:id` — получение пользователя по `id`;
- `\login` — проверка данных пользователя;
- `\addQuiz` — добавление викторины;
- `\addQuestion\:quizId` — добавление вопросов к викторине;
- `\getQuestionsByQuizId\:QuizId` — получение вопрос по `id` викторины;
- `\deleteQuestion\:questionId` — удаление вопроса по `id`;
- `\getAnswersByQuestionId\:questionId` — получение ответов к вопросу по `id` вопроса;
- `\getQuizzesByUser\:userId` — получение викторин созданных пользователем по `id`;
- `\deleteQuiz\:id` — удаление викторины по `id`;
- `\quizList` — получение списка викторин;
- `\getQuiz\:id` — механизм прохождения викторины по `id`;
- `\saveAnswer` — сохранение ответа пользователя на вопрос;
- `\deleteUserAnswers` — удаление ответов пользователя на викторину;
- `\results\:quizId` — получение результатов прохождения викторины;

Обработчик маршрута для получения викторины по `id` представлен в листинге 1.

```
exports.getQuiz = async (req, res) => {
  try {
    const quizId = req.params.id;
    const quizz = await quiz.findByPk(quizId);
    const questions = await question.findAll({ where: { quiz_id: quizId } });
    const questionWithAnswers = await Promise.all(questions.map(async (question) => {
      const answers = await answer.findAll({ where: { question_id: question.id } });
      return {
        ...question.get(),
        answers
      };
    }));
  }
});
```



```

    res.status(200).json({ quizz, questions: questionWithAnswers });
  } catch (error) {
    console.error(error);
    res.status(500).send({ message: "Ошибка при получении викторины." });
  }
};

```

Листинг 1. Получение викторины по id

Обработчик маршрута для сохранения ответа пользователя 2.

```

exports.saveAnswer = async (req, res) => {
  try {
    const user_id = req.userId;
    console.log(user_id);
    const {quiz_id, question_id, user_answer, is_correct} = req.body;
    console.log(req.body)
    const newAnswer = await result.create({
      user_id,
      quiz_id,
      question_id,
      user_answer,
      is_correct
    });
    return res.status(201).json({
      message: 'Ответ успешно сохранен!',
      answer: newAnswer
    });
  } catch (error) {
    console.error('Ошибка при сохранении ответа:', error);
    return res.status(500).json({ message: 'Ошибка сервера при сохранении ответа.' });
  }
};

```

Листинг 2. сохранение ответа пользователя

Обработчик маршрута для получения вопросов викторины 3.

```

exports.getQuestionsByQuizId = (req, res) => {
  const quizId = req.params.QuizId;
  console.log(req.params.QuizId)

```

```

question.findAll({
  where: { quiz_id: quizId },
  include: [
    {
      model: answer,
      as: 'answers'
    }
  ]
})
.then(questions => {
  res.status(200).json(questions);
})
.catch(err => {
  globalFunctions.sendError(res, err);
});
};

```

Листинг 3. сохранение ответа пользователя

В прил. 1 приведен код контроллера quizController.js.

В прил. 2 приведен код для контроллера questionsController.js.

2.4. Разработка компонентов

При разработке приложения были созданы следующие компоненты:

- AddQuestions.vue — добавление вопросов к викторине;
- AddQuiz.vue — добавление викторины;
- GoQuiz.vue — прохождение викторины;
- MyQuiz.vue — получение списка викторин созданных пользователем;
- QuizList.vue — получение списка викторин;
- QuizResult.vue — получение результатов прохождения викторины;
- Login.vue — получение логин страницы;
- Register.vue — получение страницы регистрации.

2.5. Реализованная функциональность

Реализованная функциональность веб-приложения продемонстрирована далее в виде рисунков.

Осуществление авторизации происходит на странице, которая показана на рисунке 2.

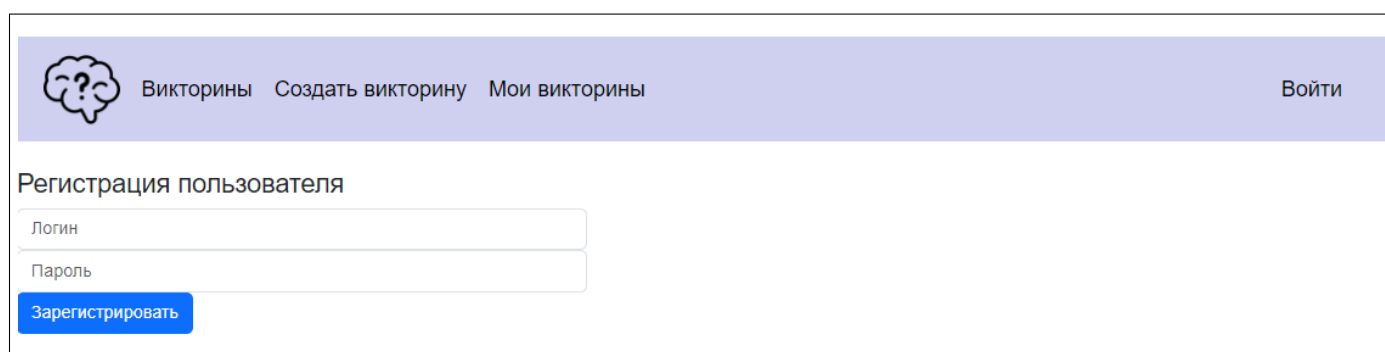
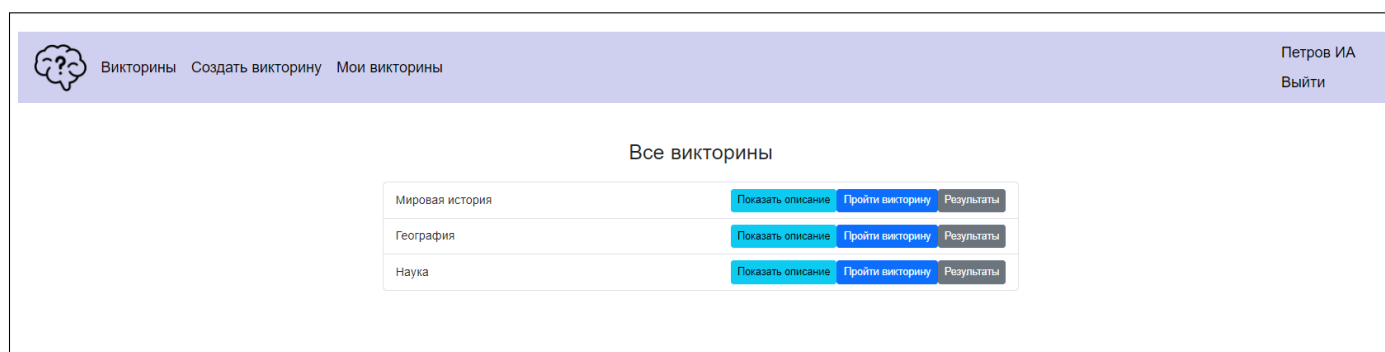


Рисунок 2. Страница авторизации

Просмотр списка викторин осуществляется на странице, которая показана на рисунке 3.



Все викторины			
Мировая история	Показать описание	Пройти викторину	Результаты
География	Показать описание	Пройти викторину	Результаты
Наука	Показать описание	Пройти викторину	Результаты

Рисунок 3. Список викторин

Прохождение викторины реализовано на странице, которая показана на рисунке 4 .

Информация о правильных ответах выводится на странице, которая показана на рисунке 5.

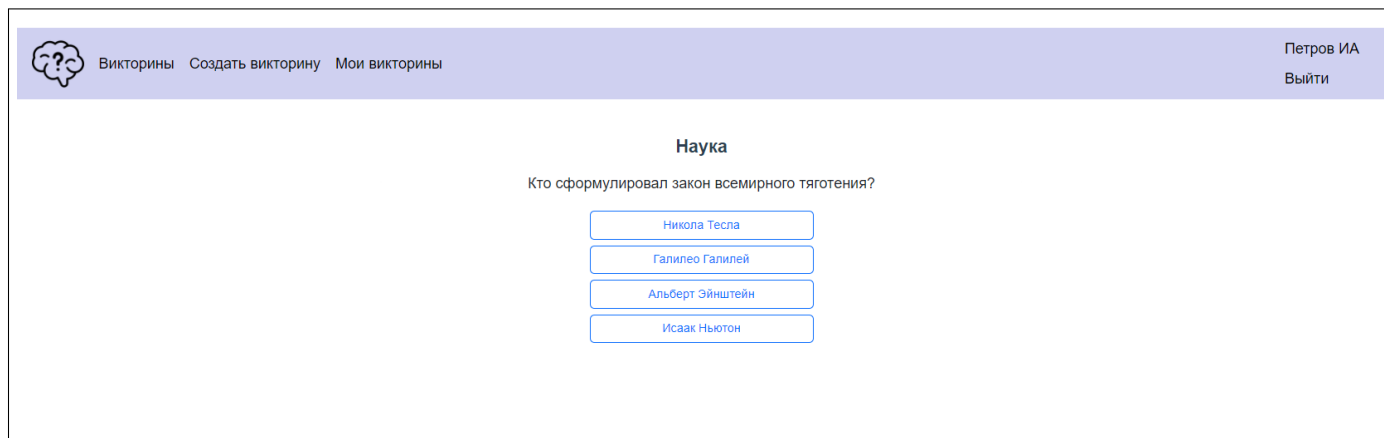


Рисунок 4. Прохождение викторины

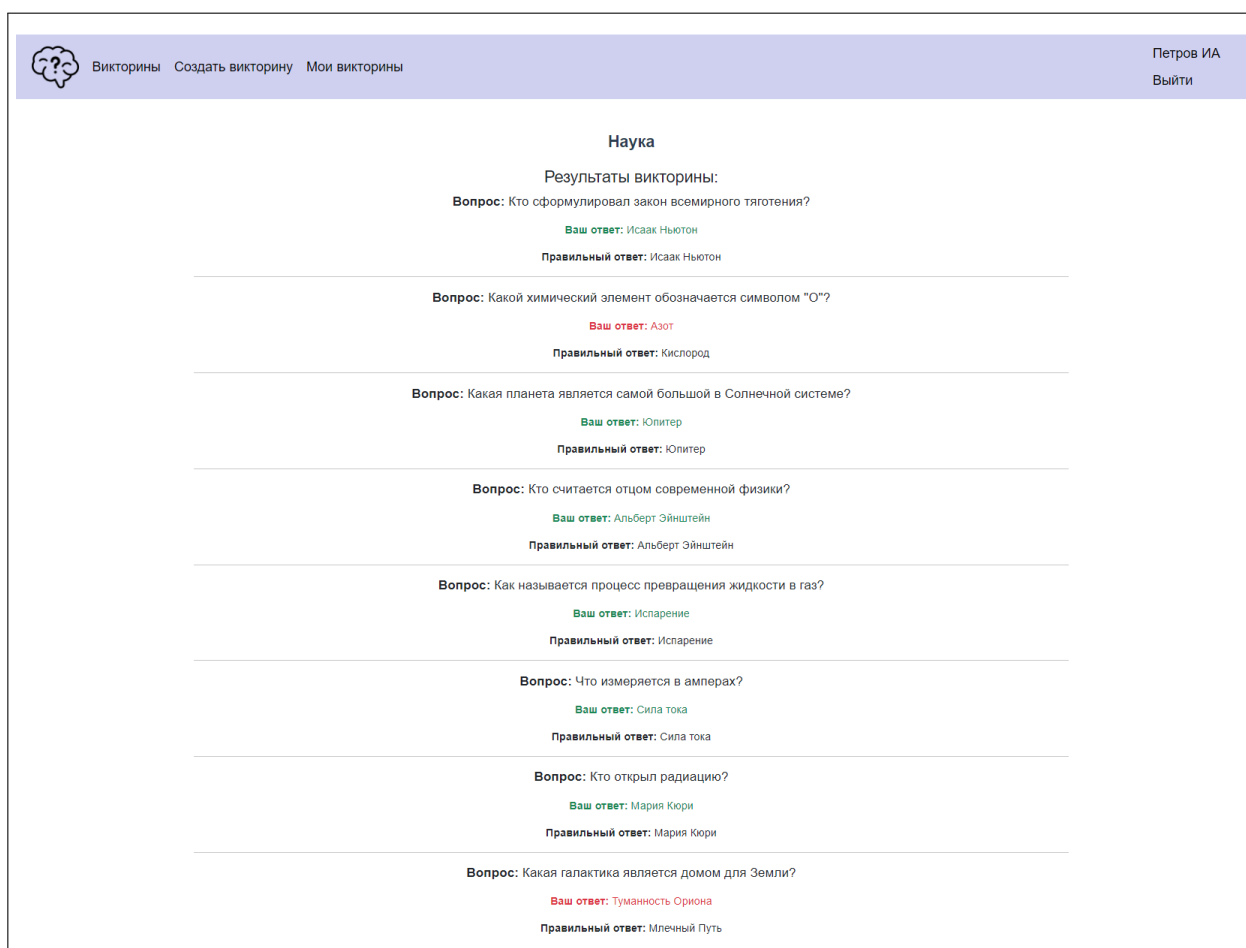


Рисунок 5. Итоги викторины

Ознакомиться с результатами прохождения викторины другими пользователями можно на странице, которая показана на рисунке 6.

Добавление викторины реализовано на странице, которая показана на ри-


<div>  Викторины Создать викторину Мои викторины </div> <div>Петров ИА Выйти</div>			
Результаты викторины			
Имя пользователя	Правильные ответы	Всего ответов	Процент правильных ответов
Дондукова ВЕ	9	9	100.00%
Петров ИА	7	9	77.78%
Сенькин ВЮ	3	9	33.33%
Рябец ЛВ	2	9	22.22%

Рисунок 6. Результаты викторины

сунке 7.


<div>  Викторины Создать викторину Мои викторины </div> <div>Петров ИА Выйти</div>
<div>Добавить викторину</div> <div> <div> <div>Наименование викторины</div> <div>Введите название викторины</div> </div> <div> <div>Описание викторины</div> <div>Введите описание викторины</div> </div> <div>Добавить</div> </div>

Рисунок 7. Создание викторины

Добавление вопросов в викторину реализовано на странице, которая показана на рисунке 8.

Викторины Создать викторину Мои викторины Петров ИА Выйти

Добавление вопросов к викторине

Текст вопроса:
Введите вопрос

Правильный ответ:
Введите правильный ответ

Неправильные ответы:
Неправильный ответ 1
Неправильный ответ 2
Неправильный ответ 3

Добавить вопрос

Список добавленных вопросов:

Какая река является самой длинной в мире?	Показать ответ	Удалить
Какая пустыня является самой большой на Земле?	Показать ответ	Удалить
Какой океан является самым большим по площади?	Показать ответ	Удалить
Какая гора является самой высокой в мире?	Показать ответ	Удалить
Какое из этих государств является самым большим по территории?	Показать ответ	Удалить
В какой стране находится Великая Китайская стена?	Показать ответ	Удалить
Какое озеро является самым глубоким в мире?	Показать ответ	Удалить
Какая страна имеет самую большую численность населения?	Показать ответ	Удалить
Какая из этих стран является архипелагом?	Показать ответ	Удалить
Какая страна известна как "страна тысячи озер"?	Показать ответ	Удалить

Рисунок 8. Создание вопроса

Список викторин созданных пользователем показан на странице, которая показана на рисунке 9.

Викторины Создать викторину Мои викторины Петров ИА Выйти

Ваши викторины

Мировая история	Описание	Изменить	Удалить
География	Описание	Изменить	Удалить

Рисунок 9. Ваши викторины

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано веб-приложение для организации викторин.

Выполнены задачи:

- 1) Разработана база данных;
- 2) Созданы основные модули для перехода и просмотра объектов;
- 3) Разработано добавление и редактирование объектов;
- 4) Разработано добавление викторин;
- 5) Разработан просмотр пользователем всех доступных викторин и возможность пройти их.
- 6) Разработан просмотр результатов прохождения викторин.

Выполнение перечисленных задач позволило достигнуть поставленную цель — создать веб-приложение для организации викторин.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Bootstrap [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <https://getbootstrap.com/> (дата обращения: 20.12.2023).
2. Kahoot! - , [Electronic resource]. — [S. l. : s. n.]. — URL: <https://kahoot.com/> (online; accessed: 20.12.2023).
3. Mentimeter - [Electronic resource]. — [S. l. : s. n.]. — URL: <https://www.mentimeter.com/> (online; accessed: 20.12.2023).
4. MySQL [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <https://www.mysql.com/> (дата обращения: 20.12.2023).
5. Node.js [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <https://nodejs.org/> (дата обращения: 19.05.2022).
6. The Progressive JavaScript Framework Vue.js [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <https://vuejs.org/> (дата обращения: 19.05.2022).
7. Quizlet - [Electronic resource]. — [S. l. : s. n.]. — URL: <https://quizlet.com/> (online; accessed: 20.12.2023).
8. Sequelize [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <https://sequelize.org/> (дата обращения: 20.12.2023).

ПРИЛОЖЕНИЕ 1.

Код контроллера quizController.js

```
var db = require('../config/db.config.js');
var globalFunctions = require('../config/global.functions.js');
var quiz = db.quiz1;
var question = db.question;
var answer = db.answer;
var result = db.result;
var user = db.user;
var result = db.result;
exports.create = (req, res) => {
  console.log(req.body);
  console.log(req.userId);
  quiz.create({
    id: req.body.id,
    name: req.body.name,
    description: req.body.description,
    user_id: req.userId,
  }).then(object => {
    globalFunctions.sendResult(res, object);
  }).catch(err => {
    globalFunctions.sendError(res, err);
  })
};
exports.findById = (req, res) => {
  quiz.findByIdPk(req.params.id)
    .then(object => {
      globalFunctions.sendResult(res, object);
    })
    .catch(err => {
      globalFunctions.sendError(res, err);
    })
};
exports.getQuizzesByUser = async (req, res) => {
  try {
    const userId = req.params.userId;
    if (!userId) {
```

```

        return res.status(400).json({ message: "User ID is required" });
    }

    const quizzes = await quiz.findAll({
        where: { user_id: req.userId }
    });

    res.status(200).json(quizzes);
} catch (error) {
    console.error(error);
    res.status(500).json({ message: "Server error" });
}
};

const deleteQuestion = async (questionId) => {
    return question.destroy({
        where: { id: questionId }
    });
};

const deleteResult = async (resultId) => {
    return result.destroy({
        where: { id: resultId }
    });
};

exports.deleteQuizWithQuestions = async (req, res) => {
    console.log(req.params);
    const quizId = req.params.id;
    try {
        const questions = await question.findAll({
            where: { quiz_id: quizId }
        });
        const results = await result.findAll({
            where: { quiz_id: quizId }
        });
        for (const question of questions) {
            await deleteQuestion(question.id);
        }
        for (const result of results) {
            await deleteResult(result.id);
        }
    }
};

```

```

    }
    await quiz.destroy({
      where: { id: quizId }
    });
    res.status(200).send({ message: "Викторина и все связанные с ней вопросы успешно удалены !" });
  } catch (err) {
    console.error(err);
    res.status(500).send({ message: "Произошла ошибка при удалении викторины и вопросов." });
  }
};

exports.getAllQuizzes = async (req, res) => {
  try {
    const quizzes = await quiz.findAll();
    res.status(200).json(quizzes);
  } catch (error) {
    console.error(error);
    res.status(500).send({ message: "Ошибка при получении викторин." });
  }
};

exports.getQuiz = async (req, res) => {
  try {
    const quizId = req.params.id;
    const quizz = await quiz.findByPk(quizId);
    const questions = await question.findAll({ where: { quiz_id: quizId } });
    const questionWithAnswers = await Promise.all(questions.map(async (question) => {
      const answers = await answer.findAll({ where: { question_id: question.id } });
      return {
        ...question.get(),
        answers
      };
    }));
    res.status(200).json({ quizz, questions: questionWithAnswers });
  } catch (error) {
    console.error(error);
    res.status(500).send({ message: "Ошибка при получении викторины." });
  }
};

```

```

exports.submitResults = async (req, res) => {
  console.log(req.body);
  const { userId, quizId, results } = req.body;
  try {
    const quizz = await quiz.findPk(quizId);
    if (!quizz) {
      return res.status(404).json({ message: 'Викторина не найдена' });
    }
    const saveResultsPromises = results.map(result => {
      return Result.create({
        user_id: userId,
        quiz_id: quizId,
        question_id: result.questionId,
        user_answer: result.userAnswer,
        is_correct: result.isCorrect
      });
    });
    await Promise.all(saveResultsPromises);
    res.status(200).json({ message: 'Результаты успешно сохранены' });
  } catch (error) {
    console.error('Ошибка при сохранении результатов:', error);
    res.status(500).json({ message: 'Ошибка сервера' });
  }
};

exports.saveAnswer = async (req, res) => {
  try {
    const user_id = req.userId;
    console.log(user_id);
    const {quiz_id, question_id, user_answer, is_correct } = req.body;
    console.log(req.body)
    const newAnswer = await result.create({
      user_id,
      quiz_id,
      question_id,
      user_answer,
      is_correct
    });
    return res.status(201).json({

```

```

        message: 'Ответ успешно сохранен!',
        answer: newAnswer
    });
} catch (error) {
    console.error('Ошибка при сохранении ответа:', error);
    return res.status(500).json({ message: 'Ошибка сервера при сохранении ответа.' });
}
};

exports.deleteUserAnswers = async (req, res) => {
    console.log("gggg");
    const userId = req.userId;
    const quizId = req.body.quizId;
    console.log(userId);
    console.log(quizId);
    try {
        await result.destroy({
            where: {
                user_id: userId,
                quiz_id: quizId
            }
        });
        res.status(200).json({ message: 'Ответы успешно удалены.' });
    } catch (error) {
        console.error('Ошибка при удалении ответов:', error);
        res.status(500).json({ error: 'Ошибка при удалении ответов.' });
    }
};

exports.getResults = async (req, res) => {
    const quizId = req.params.quizId;
    const currentUserId = req.userId;
    try {
        const results = await result.findAll({
            where: { quiz_id: quizId },
            attributes: ['user_id', 'is_correct'],
        });
        const userIds = [...new Set(results.map(result => result.user_id))];
        const users = await user.findAll({
            where: {

```

```

        id: userIds
      },
      attributes: ['id', 'username']
    });
    const userMap = {};
    users.forEach(user => {
      userMap[user.id] = user.username;
    });
    const userResults = {};
    results.forEach(answer => {
      const userId = answer.user_id;
      const userName = userMap[userId];
      if (!userResults[userId]) {
        userResults[userId] = {
          userId: userId,
          userName: userName,
          correctAnswers: 0,
          totalAnswers: 0
        };
      }
      userResults[userId].totalAnswers += 1;
      if (answer.is_correct) {
        userResults[userId].correctAnswers += 1;
      }
    });
    const responseResults = Object.values(userResults);
    responseResults.sort((a, b) => b.correctAnswers - a.correctAnswers);
    return res.status(200).json({ results: responseResults, currentUserId });
  } catch (error) {
    console.error('Ошибка при получении результатов:', error);
    res.status(500).json({ error: 'Ошибка при получении результатов.' });
  }
};

```

ПРИЛОЖЕНИЕ 2.

Код контроллера questionsController.js

```
var db = require('../config/db.config.js');
var globalFunctions = require('../config/global.functions.js');
var question = db.question;
var answer = db.answer;
var quiz = db.quiz1;
exports.create = async (req, res) => {
  try {
    console.log(req.body);
    const createdQuestion = await question.create({
      quiz_id: req.body.quiz_id,
      question_text: req.body.question_text
    });
    const questionId = createdQuestion.id;
    await answer.create({
      question_id: questionId,
      answer_text: req.body.correctAnswer,
      is_correct: true
    });
    await answer.bulkCreate([
      {
        question_id: questionId,
        answer_text: req.body.wrongAnswer1,
        is_correct: false
      },
      {
        question_id: questionId,
        answer_text: req.body.wrongAnswer2,
        is_correct: false
      },
      {
        question_id: questionId,
        answer_text: req.body.wrongAnswer3,
        is_correct: false
      }
    ]);
  }
```

```

    globalFunctions.sendResult(res, {
      message: "Вопрос и ответы успешно добавлены!"
    });
  } catch (err) {
    console.error("Ошибка при добавлении вопроса и ответов:", err);

    globalFunctions.sendError(res, err);
  }
};

exports.getQuestionsByQuizId = (req, res) => {
  const quizId = req.params.QuizId;
  console.log(req.params.QuizId)
  question.findAll({
    where: { quiz_id: quizId },
    include: [
      {
        model: answer,
        as: 'answers'
      }
    ]
  })
  .then(questions => {
    res.status(200).json(questions);
  })
  .catch(err => {
    globalFunctions.sendError(res, err);
  });
};

exports.deleteQuestion = (req, res) => {
  console.log(req.params);
  const questionId = req.params.questionId;
  console.log(req.params.questionId)
  question.destroy({
    where: { id: questionId }
  })
  .then(() => {
    res.status(200).send({ message: "Вопрос успешно удалён!" });
  });
};

```



```

    })
    .catch(err => {
        globalFunctions.sendError(res, err);
    });
};

exports.getAnswersByQuestionId = (req, res) => {
    const questionId = req.params.questionId;
    console.log(req.params.questionId)
    answer.findAll({
        where: { question_id: questionId }
    })
    .then(answers => {
        res.json(answers);
    })
    .catch(err => {
        console.error(err);
        res.status(500).json({ error: 'Ошибка при получении ответов.' });
    });
};

exports.findAll = (req, res) => {
    question.findAll()
    .then(objects => {
        globalFunctions.sendResult(res, objects);
    })
    .catch(err => {
        globalFunctions.sendError(res, err);
    })
};

```