**Министерство науки и высшего образования Российской Федерации**

федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

(ФГАОУ ВО «СПбПУ»)

**Институт среднего профессионального образования**

**ОТЧЕТ**

**по преддипломной практике (по профилю специальности)**

Специальность09.02.07 Информационные системы и программирование

(код и наименование специальности)

Студент(ка) 4 курса 42919/4 группы

Марков Данил Петрович

(Фамилия, имя, отчество)

Место прохождения практики: ООО «СЛ Маркет», Санкт-Петербург, наб. Обводного канала, д. 93а литер а, помещ. 5н офис 1449

(наименование и адрес организации)

Период прохождения практики

с «22» апреля 2024 г. по «18» мая 2024 г.

Руководитель практики

от учебной организации Иванова Д.В \_\_

(подпись) (расшифровка подписи)

Руководитель с места

прохождения практики

(подпись) (расшифровка подписи)

Итоговая оценка по практике \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

М.П.

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на преддипломную практику (по профилю специальности)**

Специальность09.02.07 Информационные системы и программирование

(код и наименование специальности)

Студент(ка) 4 курса 42919/4 группы

Марков Данил Петрович

(Фамилия, имя, отчество)

Место прохождения практики: ООО «СЛ Маркет», Санкт-Петербург, наб. Обводного канала, д. 93а литер а, помещ. 5н офис 1449

(наименование и адрес организации)

Период прохождения практики

с «22» апреля 2024 г. по «18» мая 2024 г.

**Виды работ, обязательные для выполнения**:

1. Ознакомление со структурой и характером деятельности предприятия.
2. Сбор материалов для составления технического задания по теме дипломного проекта.
3. Разработка программного обеспечения на основе технического задания дипломного проекта.
4. Проведение испытаний, отладка и внедрение программного продукта на предприятии.
5. Расчет показателей экономической эффективности программного продукта.
6. Оформление отчета о прохождении производственной практики (преддипломной).

Задание выдал «22» апреля 2024 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Иванова Д.В.

(подпись)  (Ф.И.О.)

Задание получил «22» апреля 2024 г. \_\_\_\_\_\_\_\_\_\_\_\_ Марков Д.П (подпись) (Ф.И.О.)

**СОГЛАСОВАНО**

Председатель ПЦК «ИС»

\_\_\_\_\_\_\_\_\_\_/В.А. Андреев

«22» апреля 2024 г.

**Министерство науки и высшего образования Российской Федерации**

федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

(ФГАОУ ВО «СПбПУ»)

**Институт среднего профессионального образования**

**ДНЕВНИК**

**прохождения преддипломной практики**

**(по профилю специальности)**

Специальность09.02.07 Информационные системы и программирование

(код и наименование специальности)

Студент(ка) 4 курса 42919/4 группы

Марков Данил Петрович

(Фамилия, имя, отчество)

Место прохождения практики: ООО «СЛ Маркет», Санкт-Петербург, наб. Обводного канала, д. 93а литер а, помещ. 5н офис 1449

(наименование и адрес организации)

Период прохождения практики

с «22» апреля 2024 г. по «18» мая 2024 г.

Руководитель практики

от учебной организации Иванова Д.В.

(подпись) (расшифровка подписи)

Руководитель с места

прохождения практики

(подпись) (расшифровка подписи)

Итоговая оценка по практике \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

М.П.

Санкт-Петербург

2024

**Содержание дневника**

|  |  |  |
| --- | --- | --- |
| Дата | Виды выполненных работ и заданий по программе практики | Подпись руководителя практики |
| **1** | **2** | **3** |
| 10.04.24 | Разработка программ по типовым алгоритмам |  |
| 13.04.24 |
| 14.04.24 | Использование функций |  |
| 17.04.24 |
| 18.04.24 | Указатели и динамическое распределение памяти |  |
| 20.04.24 |
| 22.04.24 |
| 23.04.24 | Обработка строчных величин |  |
| 25.04.24 |
| 26.04.24 |
| 27.04.24 | Работа с файлами |  |
| 28.04.24 |
| 30.04.24 | Динамические структуры данных |  |
| 01.05.24 |
| 02.05.24 | Работа с файловой системой |  |
| 03.05.24 | Организация многопоточной обработки данных |  |
| 04.05.24 |
| …. | …… |  |

**СОДЕРЖАНИЕ**

[Введение 6](#_Toc165830032)

[1 Общая часть 7](#_Toc165830033)

[1.1 Анализ предметной области 7](#_Toc165830034)

[1.2 Постановка задачи 7](#_Toc165830035)

[1.3 Функции проектируемого приложения 8](#_Toc165830036)

[1.4 Анализ рынка существующих решений 8](#_Toc165830037)

[1.5 Архитектура приложения 12](#_Toc165830038)

[1.6 Обоснование и выбор методики, технологии и инструментальных средств проектирования и разработки 16](#_Toc165830039)

[1.7 Вывод по общей части 28](#_Toc165830040)

[2 Специальная часть 29](#_Toc165830041)

[2.1 Описание структуры приложения 29](#_Toc165830042)

[2.2 Проектирование программы 30](#_Toc165830043)

[2.3 Разработка программы 34](#_Toc165830044)

[2.4 Вывод по специальной части 53](#_Toc165830046)

[Заключение 54](#_Toc165830047)

[Список использованных источников 55](#_Toc165830048)

[Приложение А (обязательное) Описание моделей данных 56](#_Toc165830049)

[Приложение Б (обязательное) Программный код 58](#_Toc165830050)

# **ВВЕДЕНИЕ**

С развитием информационных технологий интернет стал неотъемлемой частью нашей жизни, а использование программных решений стало обязательным, для улучшения бизнеса. Таким образом, использование программных решений в маркетинге становится все более актуальным и востребованным.

Одним из инновационных направлений в этой области является использование приложений, которые подстраиваются под поведения и запросы пользователей.

Таким образом, целью работы выступает проектирование и разработка веб-приложения для создания и использования умных опросов в маркетинге. Веб-приложение, предоставляющее маркетинговому отделу компании, возможность создания сценариев опроса, исходя из различных факторов объекта опроса.

Основными задачами в проектирование и разработки являются:

* анализ предметной области;
* постановка задачи
* анализ существующих решений;
* анализ и выбор средств и методов разработки;
* разработка различных диаграмм проектирования;
* разработка серверной части веб приложения;
* разработка клиентской части веб приложения.

# **1 Общая часть**

# **1.1 Анализ предметной области**

Предприятие «СЛ Маркет» было зарегистрировано 29 августа 2013 года. Отрасль данного предприятия – это оптовая торговля пищевыми продуктами, напитками и табачными изделиями. Основным видом деятельности предприятия по ОКВЭД является оптовая торговля шоколадом и сахаристыми кондитерскими изделиями. Данное предприятие занимает 675 место в России по выручке за 2022 год [1]. В данном предприятии насчитывается 20 человек. Отделами предприятия являются: бухгалтерия, маркетинговый отдел, отдел кадров, отдел снабжения, финансовый отдел и юридический отдел.

В ходе преддипломной практики необходимо разработать веб-приложение для создания и использования умных опросов для маркетингового отдела предприятия.

Пользователями данного приложения являются специалисты телемаркетинга в отделе маркетинга, которые непосредственно общаются с клиентами предприятия.

# **1.2 Постановка задачи**

Основной задачей данной преддипломной практики является проектирование и разработка веб-приложения для создания и использования умных опросов в маркетинговом отделе, для автоматизации и улучшения взаимодействия с клиентами предприятия. Данное приложение должно предоставлять пользователям системы, удобный конструктор для создания опросов, который наглядно показывает создаваемый опрос.

Для достижения данной цели в веб-приложении должен присутствовать функционал, выполняющий следующие задачи:

* создание и редактирование опроса в конструкторe;
* прохождение опроса;
* редактирование пройденного опроса;
* просмотр пройденных опросов;
* удаление пройденных опросов;
* выгрузка данных в Excel;
* добавление пользователей в систему.

Исходя из поставленных задач можно выделить следующие этапы разработки:

* описание предметной области;
* проектирование и разработка базы данных;
* разработка веб приложения;
* тестирование.

# **1.3 Функции проектируемого приложения**

Исходя из поставленной задачи и анализа предметной области в веб-приложение предполагается наличие следующего базового функционала:

* получение / создание / редактирование опросов;
* прохождение / удаление / редактирование опроса;
* выгрузка данных о пройденных опросах в Excel;
* создание / удаление пользователей в системе.

# **1.4 Анализ рынка существующих решений**

Для описанного выше веб-приложения существуют некоторое количество аналогов на рынке. Каждое из этих решений имеет ряд особенностей, плюсов и минусов.

**1.4.1 SurveyMonkey**

SurveyMonkey – это платформа для проведения опросов с 16 миллионами вопросов ежедневно, которая позволяет создавать опросы, собирать информацию у респондентов и анализировать статистику. Данное решение содержит как бесплатную, так и расширенную платную версию, доступную по подписке. Сервис позволяет рассылать созданные формы, подключать инструменты, делится статистикой. Также платформа содержит возможность создания ветвления при прохождении опроса, но не содержит в себе блоки вопросов, что усложняет работу при большом количестве вопросов и ветвлений в программе. Также в связи с санкциями, данное приложение не доступно для работы как в бесплатном, так и в платном варианте [2]. Данное веб приложение имеет перегруженный и неудобный конструктор создания опросов, который представлен на рисунке 1.

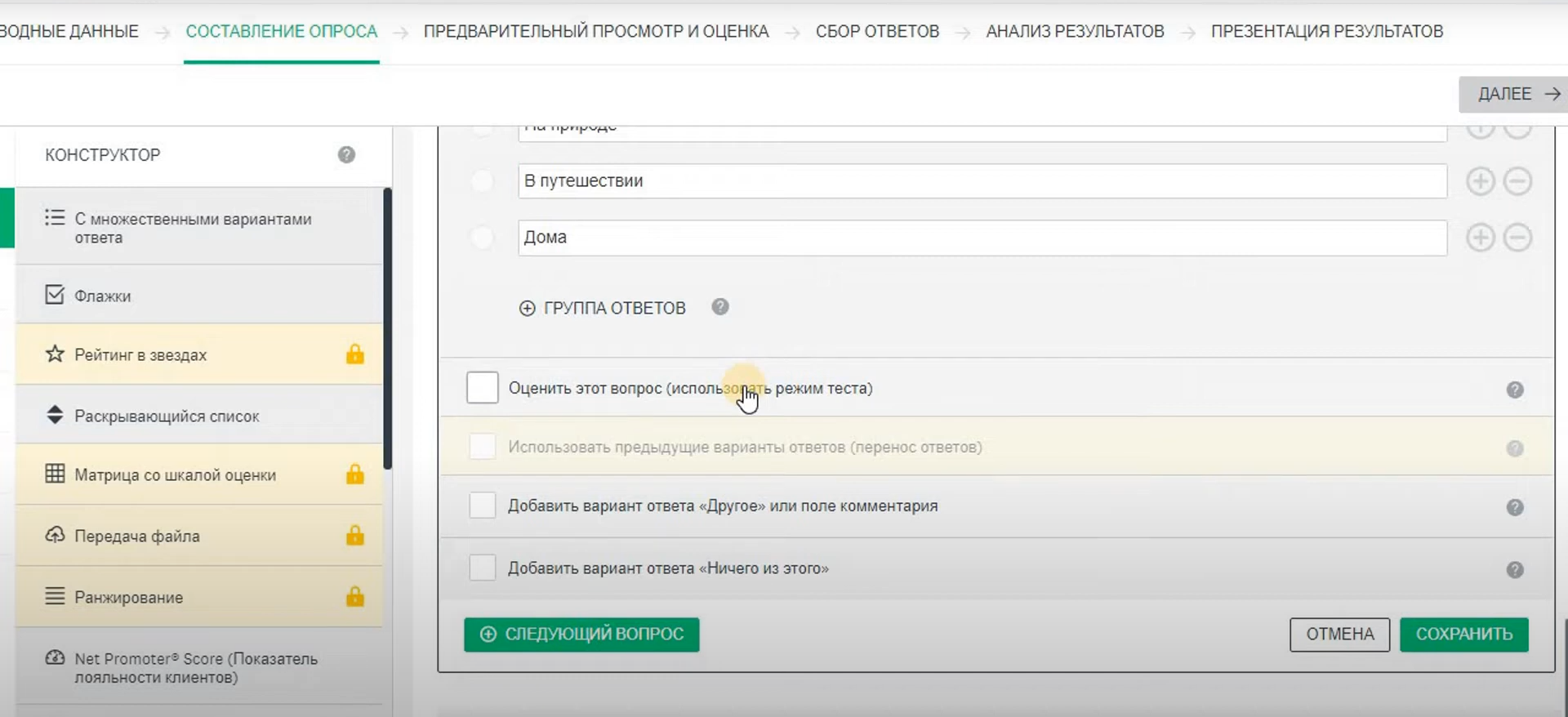


Рисунок 1 – Создание опроса в SurveyMonkey

**1.4.2 Qualtrics**

Qualtrics – это решение для онлайн опросов и способ проведения исследований компаний всех размеров. Сервис подходит для создания анкет и выявления удовлетворенности клиентов, а также ключевых пунктов маркетинговых исследований и показателей. Приложение Qualtrics поставляется с базой знаний, онлайн поддержкой и видео уроками. Qualtrics предлагает сложный анализ для профессиональных исследований. Данное приложение также содержит в себе возможность ветвления при прохождении опросов. Qualtrics не имеет пробного и бесплатного тарифа, а доступ к приложению осуществляется по подписке. Данное приложение доступно на территории Российской Федерации, но имеет проблемы с оплатой подписки, так как не принимает карты российских банков [3]. Данное приложение имеет сложный и не наглядный конструктор опросов, который представлен на рисунке 2.

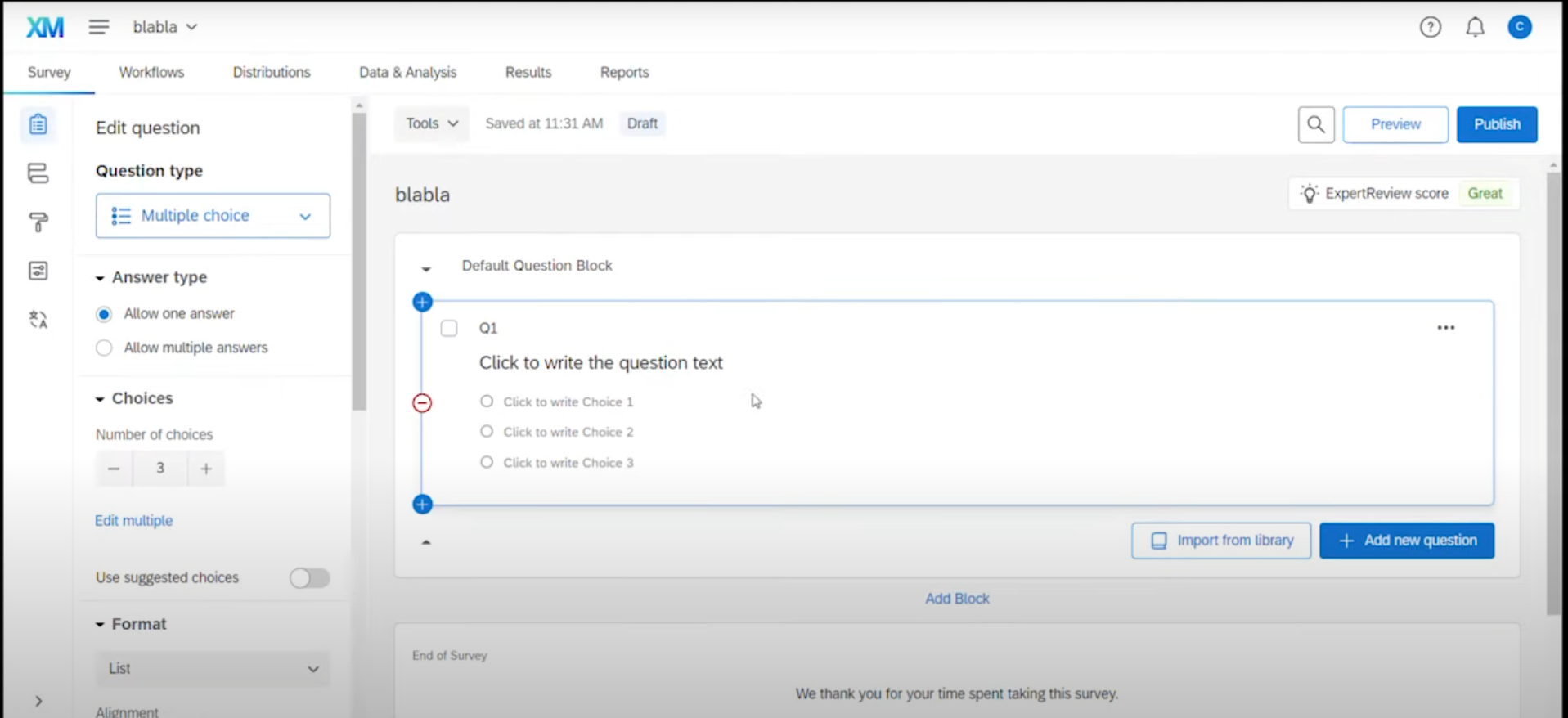


Рисунок 2 – Создание опроса в Qualtrics

**1.4.3 Google Forms**

Google Forms – это облачное решение для создания и проведения анкетирования. Google Forms также служат инструментом для создания приглашений, сбора данных. Не содержит в себе каких-либо инструментов для отображения или анализа статистики. Является упрощенным и быстрым решением при создании опросов. Имеет большую базу знаний и постоянную поддержку. Google Forms является бесплатной программой и доступна для любого пользователя. Google Forms имеют стандартный, почти аналогичный другим решениям конструктор, который, конечно, является более удобным в использовании, но все также является не наглядным [4]. Интерфейс создания опроса в Google Forms представлен на рисунке 3.

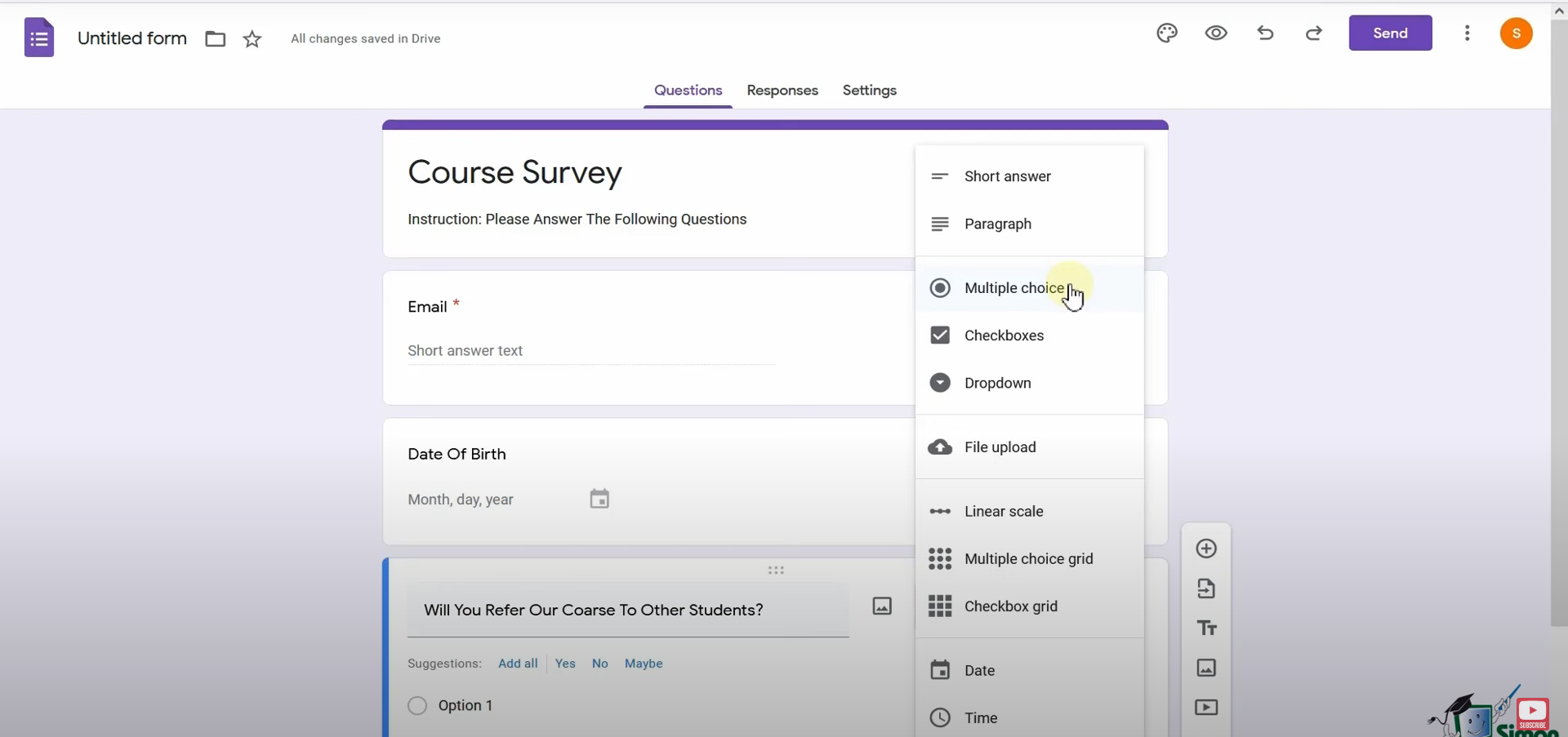


Рисунок 3 – Создание опроса в Google Forms

**1.4.4 Вывод анализа рынка существующих решений**

После анализа различных существующих решений можно составить таблицу, которая будет отражать все плюсы и минусы уже разработанных программ. Плюсы и минусы каждого решения отдельно представлены в таблице 1.

Таблица 1 – Таблица сравнения решений

| **Отличия** | **SurveyMonkey** | **Qualtrics** | **Google Forms** |
| --- | --- | --- | --- |
| Возможность создания ветвления. | **+** | **+** | **+** |
| Возможность анализа пройденных опросов | **+** | **+** | **-** |
| Содержание подробного руководства по использованию | **+** | **+** | **+** |
| Возможность прохождения опроса непосредственно из приложения | **-** | **-** | **-** |
| Наглядный конструктор опросов | **-** | **-** | **-** |
| Возможность использования без вложений | **+** | **-** | **+** |

Таким образом, анализ рынка существующих решений позволяет выявить, что большинство решений имеет крайнее сложные конструкторы создания опросов, а сами опросы плохо визуализированы. Разрабатываемое веб-приложение, в свою очередь, имеет удобный и интуитивно понятный интерфейс, который визуализирует опрос ввиде древовидной структуры, что позволяет создавать большие опросы за более короткий промежуток времени.

# **1.5 Архитектура приложения**

Для разрабатываемого веб приложения существует большое количество вариантов архитектурных решений. Каждое из этих решений имеет ряд плюсов и минусов. Для решения поставленной задачи стоит грамотно подойти к выбору архитектуры приложения.

**1.5.1 Монолитная архитектура**

Монолитная архитектура представляет собой классический подход к разработке приложений. При такой архитектуре вся функциональность находится в одном проекте. Визуальное представление монолитной архитектуры представлено на рисунке 4.

Главный плюс монолитной архитектуры – простота и легкость разработки. Компоненты монолитной системы тесно связаны, поэтому писать и тестировать такой код сравнительно легко. Также стоит отметить среду совместно используемых данных, у всех компонентов системы есть прямой доступ к той же базе данных, что позволяет беспрепятственно обмениваться этими данными. Тем самым устраняется необходимость в сложных механизмах синхронизации.

Однако, монолитная архитектура не позволяет достаточно просто масштабировать приложения, что является огромным минусом, при разработке больших приложений. Также монолитная архитектура потребляет большое количество ресурсов, что приводит к снижению производительности и увеличению затрат [5].

Изображение выглядит как текст, снимок экрана, диаграмма, дизайн

Автоматически созданное описание

Рисунок 4 – Монолитная архитектура

**1.5.2 Клиент-серверная архитектура**

Клиент-серверная архитектура — это один из наиболее распространенных подходов к построению распределенных систем. В данной архитектуре функциональность приложения разделяется между клиентскими и серверными компонентами, которые взаимодействуют между собой посредством сетевых запросов. Визуальное представление клиент-серверной архитектуры представлено на рисунке 5.

Основным преимуществом клиент-серверной архитектуры является возможность эффективного масштабирования системы. За счет разделения функциональности на клиентскую и серверную части, возможно горизонтальное масштабирование серверов, что позволяет обрабатывать увеличенные нагрузки без необходимости модификации всей системы. Кроме того, такая архитектура способствует улучшению отказоустойчивости, так как отказ одного сервера не приводит к полной недоступности всей системы.

Однако, клиент-серверная архитектура также имеет свои ограничения. Например, необходимость поддержания и обновления серверной инфраструктуры может быть сложной и затратной задачей. Кроме того, взаимодействие между клиентом и сервером может быть замедлено из-за сетевых задержек, что может сказаться на производительности системы.

Тем не менее, клиент-серверная архитектура остается популярным выбором для многих приложений благодаря своей гибкости, масштабируемости и относительной простоте реализации [6].

Изображение выглядит как текст, диаграмма, снимок экрана, План

Автоматически созданное описание

Рисунок 5 – Клиент-серверная архитектура

**1.5.3 Микросервисная архитектура**

Микросервисная архитектура — это подход к построению приложений, при котором функциональность разделяется на небольшие, автономные сервисы, каждый из которых отвечает за определенный аспект бизнес-логики или функциональность. Визуальное представление микросервисной архитектуры представлено на рисунке 6.

Основным преимуществом микросервисной архитектуры является высокая гибкость и независимость компонентов. Каждый сервис может быть разработан, развернут и масштабирован независимо от других, что позволяет командам разработчиков работать параллельно над различными частями системы.

Другим важным преимуществом является улучшенная масштабируемость. Микросервисы могут быть масштабированы горизонтально, то есть путем добавления новых экземпляров сервисов, что позволяет легко управлять увеличивающейся нагрузкой на систему. Это также улучшает отказоустойчивость, так как отказ одного сервиса не приводит к полной недоступности всей системы.

Однако, микросервисная архитектура также имеет свои вызовы и ограничения. Например, управление большим количеством микросервисов может быть сложным и требовать дополнительных инструментов и процессов [7].

Изображение выглядит как текст, снимок экрана, диаграмма, круг

Автоматически созданное описание

Рисунок 6 – Микросервисная архитектура

# **1.6 Обоснование и выбор методики, технологии и инструментальных средств проектирования и разработки**

Для разработки веб-приложения для создания и использования умных опросов стоит также правильно выбрать технологии разработки. Для разработки приложения стоит учитывать языки программирования, библиотеки, фреймворки, базы данных.

**1.6.1 Обзор и выбор языка программирования**

**1.6.1.1 Python**

Python — это высокоуровневый, интерпретируемый язык программирования, который широко используется для разработки веб-приложений благодаря своей простоте, читаемости и мощным фреймворкам.

Python имеет ряд плюсов:

* достаточно простой синтаксис, что делает его простым для изучения и понимания;
* богатая стандартная библиотека, которая покрывает множество задач, упрощающих разработку;
* широко применяем, благодаря чему имеет огромное сообщество и множество доступных ресурсов.

Однако данному языку присуще и минусы:

* относительная медленность выполнения, особенно при работе с большими объемами данных;
* потребление ресурсов, Python известен своим потреблением ресурсов, особенно при выполнении на стороне сервера. Это может привести к увеличенным требованиям к памяти и процессору при обработке больших нагрузок или при работе с высоконагруженными веб-приложениями.

**1.6.1.2 С#**

С# — это объектно-ориентированный язык программирования, разработанный Microsoft, который часто используется для создания приложений на платформе .NET.

Плюсами C# можно выделить:

* интеграция с платформой .NET, C# тесно интегрирован с платформой .NET, что предоставляет доступ к множеству библиотек и инструментов для разработки;
* простота использования: C# имеет понятный и интуитивно понятный синтаксис, что делает его отличным выбором для разработчиков, знакомых с другими языками, такими как Java или JavaScript;
* высокая производительность: Код на C# может быть скомпилирован в машинный код, что обеспечивает высокую производительность приложений.

Минусами С# можно выделить:

* ограниченность платформы, использование C# часто связано с платформой .NET, что может ограничить перенос приложений на другие платформы;
* закрытость, поскольку C# разработан Microsoft, некоторые разработчики могут предпочесть более открытые языки и платформы.

**1.6.1.3 JavaScript**

JavaScript – мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Языком JavaScript не владеет какая-либо организация или компания, что отличает его от ряда языков программирования, используемых в веб-разработке [8].

JavaScript обладает следующими плюсами:

* универсальность, является стандартным языком для веб-разработки, широко используется как на клиентской, так и на серверной стороне;
* фреймворки, существуют мощные фремворки для клиентской и серверной части веб приложений;
* большое сообщество, JavaScript обладает огромным и активным сообществом разработчиков, что обеспечивает доступ к обширной базе знаний, библиотекам и ресурсам;
* быстрое выполнение, современные JavaScript движки обеспечивают высокую производительность выполнения кода, что позволяет создавать быстрые и отзывчивые веб-приложения.

JavaScript обладает следующими минусами:

* браузерная совместимость, JavaScript может вести себя по-разному в различных браузерах, что может привести к несоответствиям и ошибкам в работе вашего приложения;
* асинхронность и функции обратного вызова, использование асинхронных операций и функций обратного вызова может привести к сложностям в управлении потоком выполнения кода, что может привести к появлению ошибок и трудностям в отладке;
* недостатки языка, JavaScript имеет некоторые недостатки в языковом уровне, такие как неявное преобразование типов данных, глобальные переменные, сложности с обработкой ошибок и другие, которые могут привести к неожиданному поведению и ошибкам в приложении.

**1.6.1.4 Выбор языка программирования**

Исходя из рассмотренных выше факторов, выбор в пользу JavaScript для разработки веб-приложений обоснован. JavaScript является одним из самых популярных и востребованных языков программирования в мире веб-разработки по ряду причин.

Во-первых, JavaScript является универсальным языком программирования, который широко применяется как на клиентской, так и на серверной стороне. Это делает его идеальным выбором для создания интерактивных веб-приложений, которые могут взаимодействовать с пользователем и обрабатывать данные на стороне клиента, а также взаимодействовать с сервером для получения и отправки данных.

Во-вторых, JavaScript обладает огромным и активным сообществом разработчиков, что обеспечивает доступ к обширной базе знаний, библиотекам, фреймворкам и ресурсам. Это делает процесс разработки более эффективным и ускоренным, а также обеспечивает поддержку и помощь в случае возникновения проблем или вопросов.

Кроме того, JavaScript обладает широким выбором клиентских и серверных фреймворков и библиотек, которые упрощают создание современных и интерактивных веб-приложений. Эти фреймворки предоставляют готовые компоненты, инструменты для управления состоянием, маршрутизацию и другие возможности, что значительно ускоряет и упрощает разработку.

Наконец, JavaScript обеспечивает быстрое выполнение кода, что позволяет создавать отзывчивые веб-приложения.

Исходя из всех этих преимуществ, выбор в сторону JavaScript для разработки веб-приложений является обоснованным и позволит создать современные, интерактивные и эффективные приложения для пользователей.

**1.6.2 Обзор и выбор базы данных**

**1.6.2.1 PostgreSQL**

PostgreSQL — это мощная объектно-реляционная система управления базами данных с открытым исходным кодом. Ее особенности включают в себя поддержку сложных запросов, транзакций, процедур и триггеров.

PostgreSQL обеспечивает высокую степень надежности и целостности данных благодаря механизмам транзакций и средствам обеспечения целостности данных.

PostgreSQL позволяет создавать пользовательские типы данных, функции и операторы, что делает ее гибкой и расширяемой для различных потребностей проекта.

С последними версиями PostgreSQL появилась поддержка JSON, что делает ее подходящей для разработки приложений, работающих с таким форматом.

**1.6.2.2 MS SQL**

MS SQL (Microsoft SQL Server) — это реляционная СУБД, разработанная Microsoft. Она широко используется в корпоративной среде благодаря своим возможностям управления данными и интеграции с другими продуктами Microsoft.

MS SQL легко интегрируется с другими продуктами Microsoft, такими как .NET Framework, SharePoint, Microsoft Azure и многими другими.

MS SQL предлагает решения для масштабирования как вертикально (путем добавления ресурсов на одном сервере), так и горизонтально (путем добавления дополнительных серверов).

**1.6.2.3 MongoDB**

MongoDB — это документо-ориентированная NoSQL база данных, которая хранит данные в формате JSON-подобных документов. Она позволяет разработчикам быстро создавать и масштабировать приложения с гибкой структурой данных.

MongoDB не требует строгой схемы данных и позволяет хранить различные типы данных в одной коллекции.

MongoDB обеспечивает высокую производительность при работе с большими объемами данных и обеспечивает горизонтальное масштабирование.

MongoDB имеет встроенную поддержку для геопространственных запросов, что делает ее подходящей для приложений, работающих с географическими данными.

**1.6.2.4 Выбор базы данных**

После анализа различных факторов, включая требования проекта и его цели, был выбран PostgreSQL в качестве базы данных для приложения. Одним из ключевых факторов, является поддержка формата JSON в PostgreSQL.

Поддержка JSON в PostgreSQL открывает широкие возможности для работы с полуструктурированными данными, что идеально соответствует потребностям приложения. Использование JSON позволяет хранить и обрабатывать данные в гибком формате, что особенно полезно для разработки веб-приложений, где данные могут иметь различные структуры и требования.

Благодаря поддержке JSON в PostgreSQL, приложение сможет эффективно работать с опросами в формате JSON, обмениваться ими с клиентской частью приложения и обеспечивать удобный доступ в базе данных. Это позволит упростить разработку и расширение функциональности приложения, обеспечивая при этом высокую производительность и надежность работы с данными.

**1.6.3 Обзор и выбор фреймворка для серверной части приложения**

Учитывая выбранный язык программирования, стоит рассмотреть самые популярные серверные фреймворки на JavaScript.

**1.6.3.1 Nest.Js**

Nest.js — это прогрессивный фреймворк на базе Express.js для построения масштабируемых и эффективных серверных приложений на Node.js. Он построен на основе принципов SOLID и вдохновлен Angular, что делает его модульным и легким в поддержке.

Nest.js предлагает модульную архитектуру, которая позволяет организовать код приложения на отдельные модули, обеспечивая лучшую читаемость и масштабируемость.

Nest.js предоставляет встроенную поддержку TypeScript, что делает кодирование более безопасным и производительным за счет статической типизации.

Фреймворк предоставляет мощные инструменты для обработки запросов.

**1.6.3.2 Express.Js**

Express.js — это минималистичный и гибкий веб-фреймворк для Node.js, который широко используется для разработки веб-приложений и API.

Express.js предоставляет минимальный набор функций, что делает его простым и интуитивно понятным для начинающих разработчиков.

Фреймворк предоставляет гибкую архитектуру, которая позволяет легко расширять его функциональность.

Express.js имеет широкую и активную поддержку в сообществе разработчиков, что обеспечивает доступ к богатой базе знаний и ресурсов.

**1.6.3.3 Выбор фреймворка для серверной части приложения**

После внимательного рассмотрения особенностей обоих фреймворков был выбран Express.js, для серверной части приложения. Этот фреймворк известен своей стабильностью, простотой использования и обширной поддержкой, что делает его идеальным выбором для средних проектов.

Express.js обеспечит приложение надежной и эффективной серверной частью, что позволит сосредоточиться на разработке функционала и достижении поставленных целей.

**1.6.4 Обзор и выбор фреймворка для клиентской части приложения**

Исходя из выбранного языка, стоит обратить внимание на самые популярные фреймворки для клиентской части веб приложения.

**1.6.4.1 Angular**

Angular — это мощный фреймворк, разработанный и поддерживаемый Google. Он предлагает полный набор инструментов для создания сложных веб-приложений.

Angular основан на компонентах, что делает код более модульным и повторно используемым.

Angular включает в себя множество встроенных функций, таких как маршрутизация, HTTP-запросы, формы и многое другое, что упрощает разработку сложных приложений.

Angular также известен своей простотой в использовании благодаря строгой структуре и четким правилам разработки. Он предлагает обширную документацию и активное сообщество разработчиков, что делает процесс обучения и разработки более удобным и эффективным.

Кроме того, Angular обеспечивает высокую степень безопасности благодаря встроенным инструментам защиты от уязвимостей.

**1.6.4.2 React**

React — это библиотека для создания пользовательских интерфейсов, разработанная Facebook. Она позволяет строить эффективные и масштабируемые интерфейсы с использованием компонентов.

React использует виртуальный DOM для оптимизации производительности и быстрой отрисовки интерфейса.

React предлагает гибкий подход к разработке интерфейсов и может использоваться вместе с другими библиотеками и фреймворками.

Одним из ключевых преимуществ React является его модульная архитектура, основанная на компонентах. Это позволяет разбить интерфейс на небольшие и независимые компоненты, что упрощает разработку, тестирование и поддержку кода. Кроме того, React обладает большим количеством инструментов, таких как Redux для управления состоянием приложения, React Router для навигации и маршрутизации, а также множество других библиотек.

**1.6.4.3 Vue**

Vue — это прогрессивный фреймворк для создания пользовательских интерфейсов. Он отличается простотой использования и небольшим размером.

Vue имеет небольшой размер, что облегчает его внедрение в проекты и ускоряет загрузку приложения.

Vue предлагает интуитивно понятный синтаксис и хорошо документирован, что делает его привлекательным для новичков и опытных разработчиков.

Одним из ключевых преимуществ Vue является его реактивный подход к управлению состоянием приложения. Благодаря использованию реактивных данных и компонентов, Vue автоматически обновляет интерфейс при изменении данных, что делает разработку более интуитивной и продуктивной.

**1.6.4.4 Выбор фреймворка для клиентской части приложения**

Исходя из рассмотренных факторов и требований проекта, был выбран React для клиентской части приложения. Этот фреймворк обеспечит удобное и эффективное развитие интерфейса, а также обеспечит высокую производительность и гибкость в разработке.

Одной из главных причин выбора React является его высокая гибкость и расширяемость. React позволяет разработчикам создавать сложные интерфейсы с использованием компонентной архитектуры, что делает код более модульным и повторно используемым. Благодаря виртуальному DOM, React обеспечивает оптимальную производительность при работе с динамическими данными и частыми обновлениями интерфейса.

**1.6.5 Выбор дополнительных инструментов разработки**

Современная веб-разработка требует использования различных дополнительных инструментов и библиотек для создания мощных и эффективных приложений. Нельзя просто так написать приложение без учета требований к производительности, безопасности, масштабируемости и пользовательского опыта. Вот почему использование дополнительных инструментов становится необходимостью в современной разработке

**1.6.5.1 Серверные инструменты разработки**

**1.6.5.1.1 Sequelize**

Sequelize был выбран в качестве ORM для работы с базой данных из-за его удобства и мощных инструментов. Он поддерживает различные типы баз данных, что делает его гибким инструментом для работы с различными проектами. Sequelize предоставляет простой и интуитивно понятный интерфейс для выполнения запросов к базе данных, а также обеспечивает безопасность данных благодаря возможности использования подготовленных запросов и защиты от SQL-инъекций.

**1.6.5.1.2 Bcrypt**

Bcrypt выбран для хеширования паролей в приложении в целях обеспечения безопасного хранения пользовательских данных. Эта библиотека предоставляет мощные алгоритмы хеширования, которые защищают пароли пользователей от атак по подбору и обеспечивают конфиденциальность данных.

**1.6.5.1.3 JsonWebToken**

Jsonwebtoken был выбран для создания и верификации JWT в приложении для обеспечения безопасной аутентификации пользователей и обмена данными между сервером и клиентом. Эта библиотека обеспечивает надежный способ аутентификации пользователей.

**1.6.5.1.4 Exceljs**

Exceljs был выбран для работы с файлами Excel в приложении для создания, редактирования и чтения файлов Excel. Это позволяет экспортировать данные из приложения в формате Excel и генерировать отчеты для удобного анализа и представления информации. Exceljs обеспечивает высокую производительность и гибкость при работе с файлами Excel, что делает его отличным выбором для обработки данных.

**1.6.5.2 Клиентские инструменты разработки**

**1.6.5.2.1 MobX**

MobX был выбран для управления состоянием приложения из-за своей простоты и эффективности. Благодаря концепции реактивного программирования, MobX позволяет легко организовать реактивные данные и автоматически обновлять интерфейс при изменении состояния. Это делает его отличным выбором для создания динамических и отзывчивых пользовательских интерфейсов.

**1.6.5.2.2 Axios**

Axios — это библиотека JavaScript для выполнения HTTP-запросов из браузера или Node.js. Она предоставляет простой и однородный интерфейс для выполнения запросов и работы с ответами. Axios поддерживает междоменные запросы, отмену запросов, преобразование данных и многое другое. Он широко используется в разработке веб-приложений для взаимодействия с серверами и получения или отправки данных.

**1.6.5.2.3 MUI**

MUI (также известный как Material-UI) — это популярная библиотека пользовательского интерфейса для React-приложений. Она реализует компоненты в соответствии с принципами Material Design, предоставляя разработчикам готовые стилизованные компоненты, которые могут быть легко настроены и использованы в приложении. MUI предлагает широкий набор компонентов, таких как кнопки, текстовые поля, модальные окна и другие, которые обеспечивают согласованный и эстетически приятный внешний вид приложения.

**1.6.5.2.4 React Flow**

React Flow — это библиотека, которая предоставляет компоненты и инструменты для создания визуальных диаграмм и потоков работы в React-приложениях. Она позволяет разработчикам строить интерактивные графы, соединять элементы с помощью связей и добавлять пользовательские логики взаимодействия с диаграммой. React Flow предоставляет гибкий API и настраиваемый внешний вид, что делает ее популярным выбором для разработки приложений, связанных с визуализацией и управлением потоками данных.

**1.6.5.2.5 React Router Dom**

React Router Dom — это библиотека маршрутизации для React-приложений, которая обеспечивает навигацию между различными компонентами приложения в зависимости от текущего URL. Она позволяет определить маршруты и связанные с ними компоненты, а затем автоматически обновляет интерфейс при переходе пользователя по URL. React Router Dom предоставляет механизмы для создания ссылок, перехода на другие страницы и управления историей браузера, что делает ее неотъемлемой частью разработки одностраничных приложений на React.

# **1.7 Вывод по общей части**

Исходя из анализа существующих решений, были выделены плюсы и минусы уже готовых решений. Таким образом мы можем сравнить их и принять взвешенное решение о том, какие решения лучше всего подходят для нашего проекта. Исходя из этого решения, мы выбрали подходящие для нас технологии разработки, такие как язык программирования, фреймворки и библиотеки.

# **2 Специальная часть**

# **2.1 Описание структуры приложения**

В приложении используется клиент-серверная архитектура с использованием фреймворка React для клиентской части и фреймворка Express для серверной части. Клиентская часть отвечает за отображение пользовательского интерфейса и взаимодействие с пользователем, в то время как серверная часть обрабатывает запросы от клиента и выполняет бизнес-логику. В целом, приложение следует архитектурному шаблону MVC (Model-View-Controller), где клиентская часть играет роль представления (View), а серверная часть отвечает за модель (Model) и контроллер (Controller). Взаимодействие между клиентом и сервером осуществляется через обмен данных посредством сетевых запросов. Визуальное представление архитектурного шаблона MVC представлено на рисунке 7.

Изображение выглядит как текст, снимок экрана, Шрифт, ожерелье

Автоматически созданное описание

Рисунок 7 – Архитектура MVC

# **2.2 Проектирование программы**

**2.2.1 Проектирование базы данных**

Схема базы данных является структурой, которая определяет организацию и отношения между данными в базе данных. Она определяет, какие таблицы существуют, какие столбцы они содержат, а также ограничения и связи между таблицами. Данная схема представлена на рисунке 8.

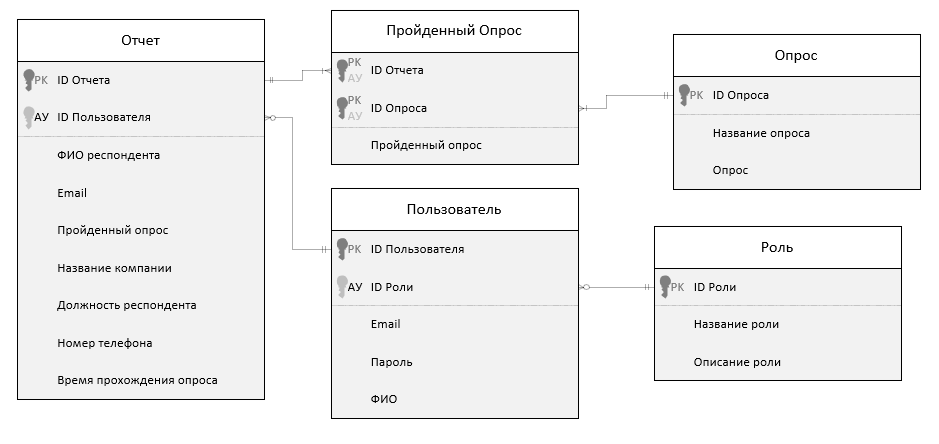


Рисунок 8 – Схема базы данных

**2.2.2 Построение диаграммы последовательностей**

Диаграмма последовательностей — это вид диаграммы в рамках языка моделирования UML (Unified Modeling Language), который используется для визуализации взаимодействий и последовательности сообщений между объектами или компонентами в системе. Она помогает описать динамическое поведение системы [9].

На рисунке 9 представлена диаграмма последовательностей, которая отображает процесс авторизации с использованием JWT (JSON Web Token). Диаграмма показывает последовательность сообщений между различными компонентами системы, включая пользователя, клиентского и серверного приложения.



Рисунок 9 – Диаграмма последовательностей для авторизации с использованием JWT токена

**2.2.3 Построение диаграммы активности**

Диаграмма активности — это вид диаграммы в рамках языка моделирования UML (Unified Modeling Language), который используется для визуализации последовательности действий, процессов и потоков управления в системе или в рамках определенной активности. Она помогает описать структуру и поведение системы на уровне бизнес-процессов [10].

На рисунках 10 и 11 представлена диаграмма активности, которая отображает процесс редактирования пройденного опроса. Диаграмма показывает последовательность шагов, которые необходимо выполнить для редактирования информации об опросе.



Рисунок 10 – Диаграмма активности для процесса редактирования пройденного опроса



Рисунок 11 – Диаграмма активности для процесса редактирования пройденного опроса

**2.2.4 Карта навигации**

Для наглядности структуры приложения была создана карта навигации (Рисунок 12). Карта навигации представляет собой схему, отображающую связь между страницами приложения.

Изображение выглядит как текст, диаграмма, снимок экрана, План

Автоматически созданное описание

Рисунок 12 – Карта навигации

# **2.3 Разработка программы**

**2.3.1 Разработка серверной части веб приложения**

Разработка серверной части веб приложения велась исходя из ранее выбранного языка, фреймворка и библиотек.

**2.3.1.1 Разработка базы данных с помощью подхода сode first**

Для разработки базы данных в проекте был использован подход “code first”. Этот подход предполагает создание моделей данных и их описание в коде до создания физической базы данных. Это позволило определить структуру базы данных и ее отношения на основе бизнес-логики и требований приложения, а затем автоматически сгенерировать соответствующую базу данных.

Для описания моделей данных и взаимодействия с базой данных была использована библиотека Sequelize. Sequelize предоставляет ORM (Object-Relational Mapping) для Node.js. С помощью Sequelize стало возможно определить модели данных, их атрибуты и связи, а затем использовать методы библиотеки для взаимодействия с базой данных.

В проекте был создан файл с моделями данных, где каждая модель была описана с использованием кода. Описание моделей данных представлено в приложении А. После этого было настроено подключение к базе данных. Настройка подключения представлена на рисунке 13 и 14.

При запуске приложения или при необходимости обновления структуры базы данных, Sequelize автоматически синхронизировал модели данных с физической базой данных.

Таким образом, использование подхода "code first" с помощью Sequelize позволило удобно разрабатывать структуру базы данных, опираясь на модели данных, описанные в коде, и автоматически синхронизировать изменения с физической базой данных при необходимости.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 13 – Создание конфигурации подключения к базе данных

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 14 – Создание подключения к базе данных

**2.3.1.2 Разработка серверной части**

При разработке были использованы стандартные компоненты для создания серверной части. Таким образом структура серверной части веб приложения принимает вид, представленный на рисунке 15.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 15 – Структура серверной части приложения

В архитектуре MVC, сущность "controller" представляет собой модули или классы, которые содержат логику обработки запросов от клиента. Они принимают запросы от клиента, взаимодействуют с моделью (Model) и вызывают соответствующие сервисы для выполнения бизнес-логики приложения. Контроллеры также могут обрабатывать входные данные, валидировать их и формировать ответы для клиента.

Сущность "services" представляет собой модули или классы, которые содержат бизнес-логику приложения. Они предоставляют функциональность, необходимую для выполнения операций, связанных с обработкой данных и выполнением конкретных задач. Сервисы могут взаимодействовать с базой данных, внешними API или другими сервисами, чтобы получать и обрабатывать данные.

Сущность "routes" определяет маршруты (URL-пути) и их соответствующие обработчики в приложении. Они определяют, какие действия должны быть выполнены при получении определенного запроса от клиента. Маршруты могут быть связаны с определенными контроллерами, которые обрабатывают запросы и вызывают соответствующие методы для выполнения требуемых действий.

Сущность "middleware" представляет собой промежуточное программное обеспечение, которое выполняет дополнительные операции между получением запроса от клиента и обработкой его соответствующим контроллером. Данная сущность может выполнять различные задачи, такие как аутентификация, авторизация, обработка ошибок, запись журналов и многое другое. Она предоставляют расширенные возможности для обработки запросов, прежде чем они достигнут контроллера.

В целом эти сущности - "controller", "services", "routes" и "middleware" - являются ключевыми компонентами архитектуры приложения, следующего паттерну MVC. Они взаимодействуют друг с другом, чтобы обеспечить правильную обработку запросов, выполнение бизнес-логики и формирование ответов для клиента.

**2.3.1.3 Разработка авторизации с использованием Json Web Token**

JSON Web Token — это открытый стандарт для создания токенов доступа, основанный на формате JSON. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях.

В данном случае токен доступа прикрепляется к заголовку HTTP запроса, всякий раз, когда приложению нужен доступ к данным только авторизированного пользователя, это означает, что пока жив токен – есть доступ к приложению. Для получения этого токена на сервер отправляется запрос с паролем и логином, при удачном сравнении пароля формируется токен и он отправляется на клиент.

Визуальное представление авторизации с использованием Json Web Token представлена на рисунке 16. Программная реализация “middleware” для авторизации представлена на рисунке 17.

Изображение выглядит как текст, чек, снимок экрана, диаграмма

Автоматически созданное описание

Рисунок 16 – Авторизация с помощью Json Web Token

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 17 – Программная реализация авторизации

**2.3.2 Разработка клиентской части веб приложения**

**2.3.2.1.1 Управление состоянием c помощью MobX**

MobX — это библиотека управления состоянием, которая позволяет вам легко управлять и отслеживать изменения состояния в приложении.

В MobX состояние представляется в виде объектов и обычных JavaScript-классов, называемых "наблюдаемыми". Когда свойства объекта помечаются как наблюдаемые, MobX автоматически отслеживает изменения этих свойств и уведомляет о них все зависимые части приложения.

Для доступа к состоянию в приложении, были созданы функции обертки, которые создают тем самым контекст. Это означает, что все компоненты нижнего уровня будут иметь доступ к состоянию. Одним из таких состояний, является состояние пользователя. Данный объект описывает состояние пользователя в приложении и меняет его в зависимости от обстоятельств. Объект состояния пользователя представлен на рисунке 18. Создание контекста с помощью функции обертки представлен на рисунке 19.

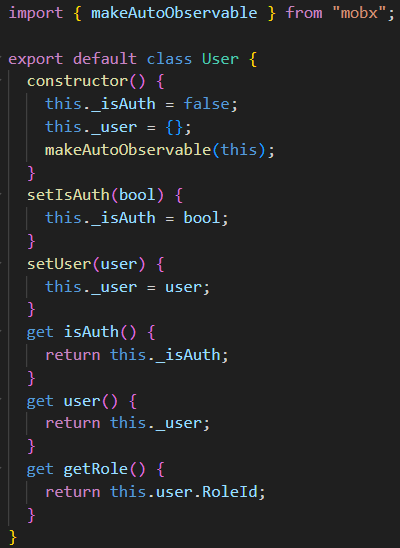


Рисунок 18 – Состояние пользователя

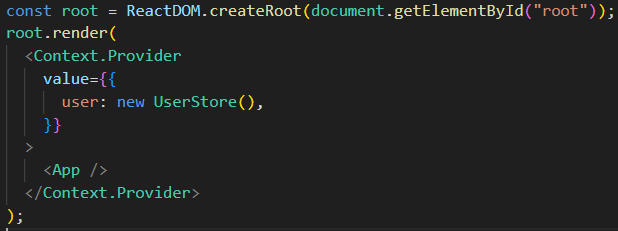


Рисунок 19 – Функция обертка

**2.3.2.1.2 Маршрутизация c помощью React Router Dom**

React Router Dom — это библиотека, которая позволяет добавить маршрутизацию в приложение на основе React. Она позволяет определить различные маршруты в приложении и связать их с соответствующими компонентами.

Для реализации маршрутизации был создан файл с маршрутами, где определены различные маршруты и связанные с ними компоненты. Эти маршруты могут быть доступны в зависимости от роли пользователя и состоянии.

Далее, используя различные функции из библиотеки React Router Dom, маршруты были определены и связаны с соответствующими компонентами. Перечень маршрутов представлен на рисунке 20. Создание маршрутизации представлено на рисунке 21.

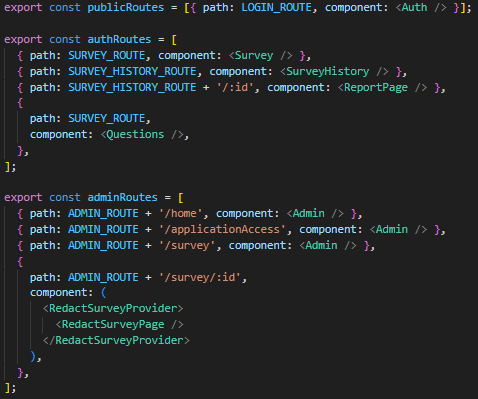


Рисунок 20 – Маршруты приложения

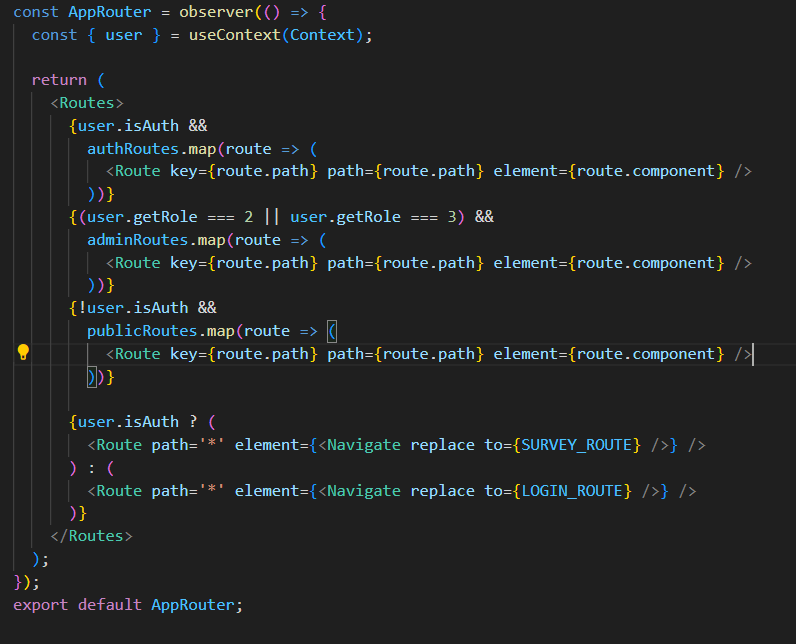


Рисунок 21 – Создание маршрутизации

**2.3.2.1.3 Работа с библиотекой React Flow**

React Flow — это библиотека для создания визуальных диаграмм и потоковых графов в React-приложениях. Она предоставляет компоненты и инструменты, которые позволяют создавать интерактивные и гибкие диаграммы с помощью React.

В React Flow основными понятиями являются узлы и связи Узлы представляют собой элементы диаграммы, такие как блоки, формы, символы или любые другие объекты, которые могут быть отображены на графической диаграмме. Узлы могут иметь различные свойства, такие как размер, цвет, содержимое и стиль.

Связи представляют собой соединения между узлами на диаграмме. Они определяют отношения и потоки данных между узлами. Связи обычно имеют начальную и конечную точку, которые указывают, какие узлы они соединяют. Связи также могут иметь различные свойства, такие как тип, стиль линии и направление.

React Flow предоставляет возможности для создания, редактирования и взаимодействия с узлами и связями. Существует возможность добавлять новые узлы на диаграмму, перемещать их, изменять их размеры и стиль. А также существует возможность создавать связи между узлами, указывая начальную и конечную точки связи. Пользователи могут взаимодействовать с диаграммой, перетаскивая узлы, изменяя связи и выполняя другие действия.

React Flow также предоставляет набор инструментов для управления и обработки событий в диаграмме, таких как нажатие мыши, перемещения и изменения размера элементов.

Таким образом для правильной работы библиотеки React Flow нужно разработать функции управления узлами и связями, которые будут предоставлять приложению перечень имеющихся узлов, их связи, координаты, а также компоненты для отрисовки. Данные функции написаны не только предоставляемыми функциями библиотеки React Flow, но и стандартными функциями библиотеки React. Данные функции представлены в приложении Б.

**2.3.2.1.4 Взаимодействие с серверной частью**

Для взаимодействия с серверной частью была использована библиотека Axios, которая предоставляет удобные инструменты для работы с HTTP-запросами в JavaScript.

В приложении были созданы два экземпляра Axios. Оба экземпляра были настроены на базовый URL, указанным в переменной окружения. Один из экземпляров используется для обычных HTTP-запросов. Другой настроен для отправки запросов с авторизационным заголовком.

Для добавления авторизационного заголовка к каждому запросу был создан перехватчик. Он добавляет заголовок Authorization со значением токена, полученного из локального хранилища, к каждому запросу, отправляемому через авторизированный экземпляр Axios. Это позволяет осуществлять авторизованные запросы к серверу.

Далее, оба экземпляра Axios экспортируются, чтобы они могли быть использованы в других частях приложения для отправки запросов к серверу.

После создания экземпляров Axios и настройки перехватчика быои созданы классы или функции, которые используют эти экземпляры для отправки запросов к серверу.

**2.3.2.2 Разработка пользовательского интерфейса**

**2.3.2.2.1 Разработка страницы авторизации**

Первой страницей для неавторизированного пользователя является страница авторизации (Рисунок 22). На данной странице, после ввода электронной почты и пароля, происходит отправка данных на сервер, после получения данных от сервера, пользователь получает доступ к приложению или сообщение об ошибке (Рисунок 23).

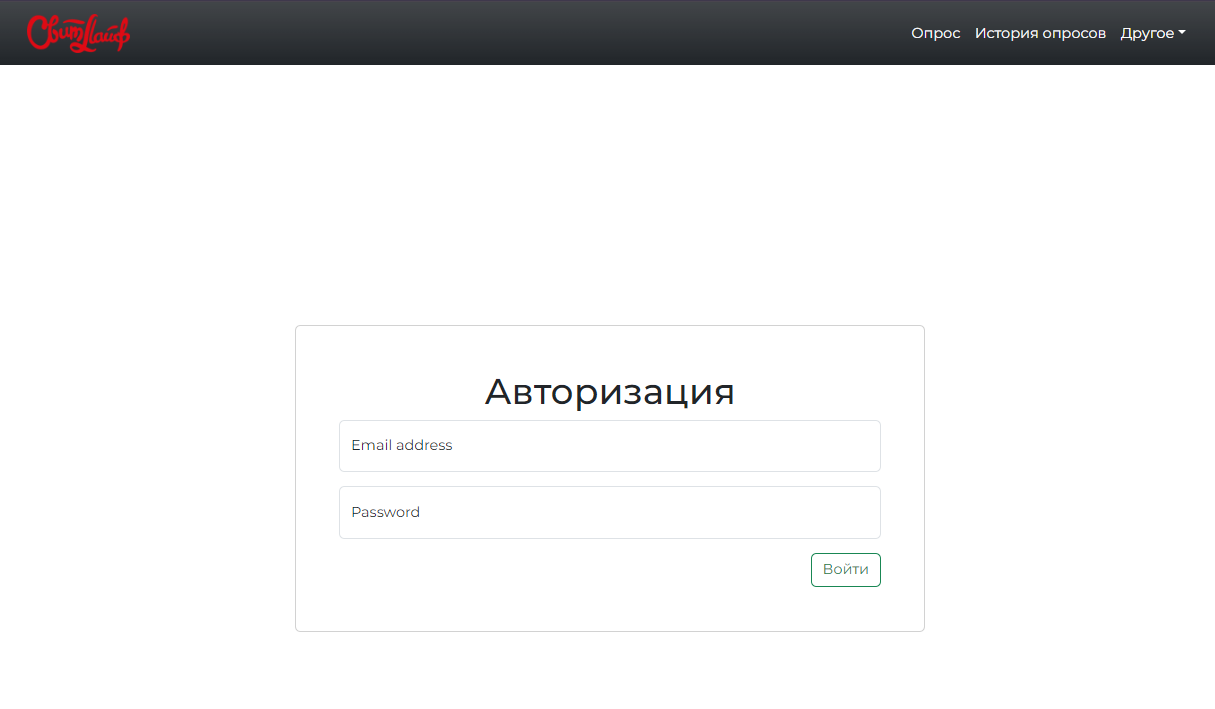


Рисунок 22 – Страница авторизации

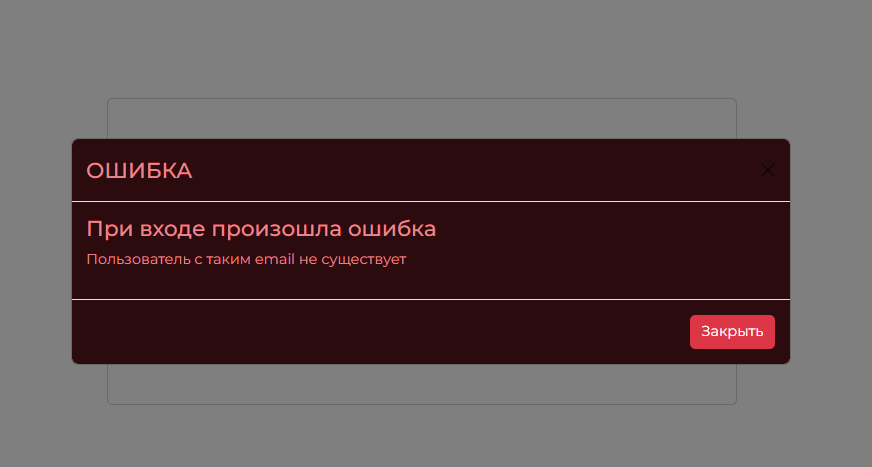


Рисунок 23 – Сообщение об ошибке

**2.3.2.2.2 Разработка модуля прохождения опроса**

После авторизации пользователя, его встречает модуль прохождения опроса. На первой странице модуля содержится ввод информации о респонденте и выбор опроса для прохождения (Рисунок 24).

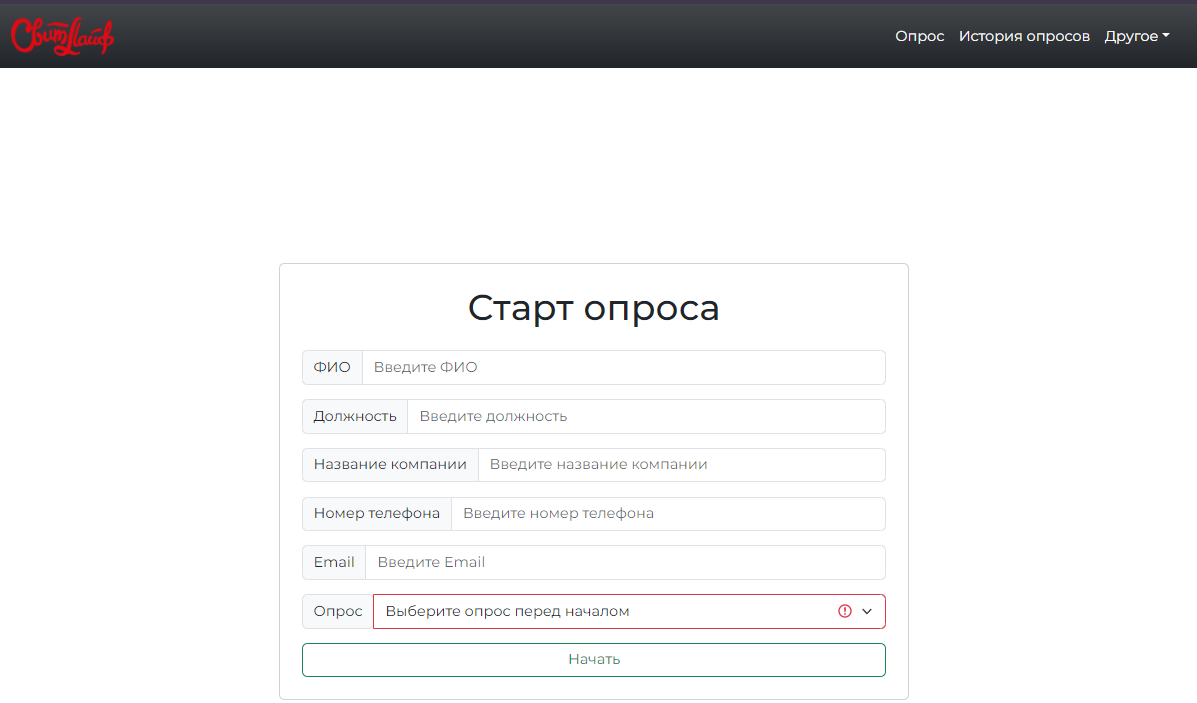


Рисунок 24 – Страница ввода информации о респонденте

После ввода информации и выбора опроса, пользователь начинает прохождение опроса. На данной странице, появляются вопросы. Которые могут содержать или не содержать поле для произвольного ответа, а также иметь возможность одного или нескольких ответов. На рисунке 25 представлен пример вопроса с возможностью произвольного ответа.

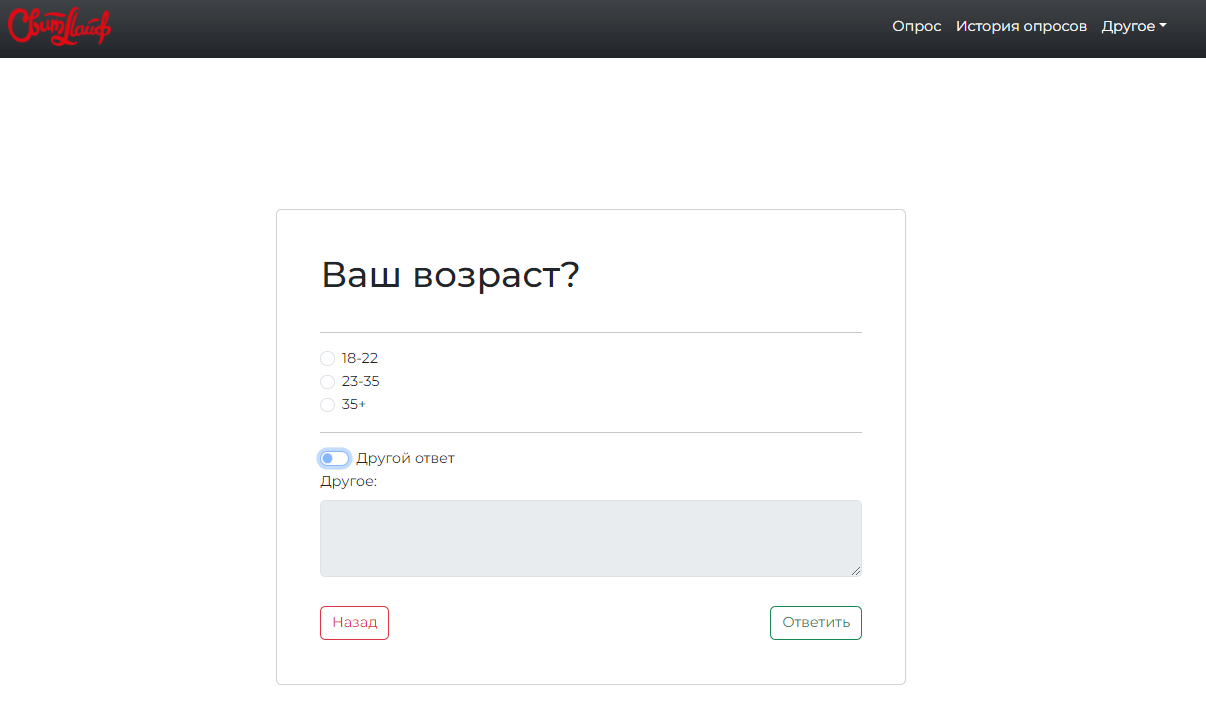


Рисунок 25 – Компонент вопроса с возможностью произвольного ответа

После прохождения опроса пользователь попадает на страницу сохранения опроса (рисунок 26), в котором он может посмотреть ответы, информацию о респонденте, время прохождения опроса, а также отредактировать их.

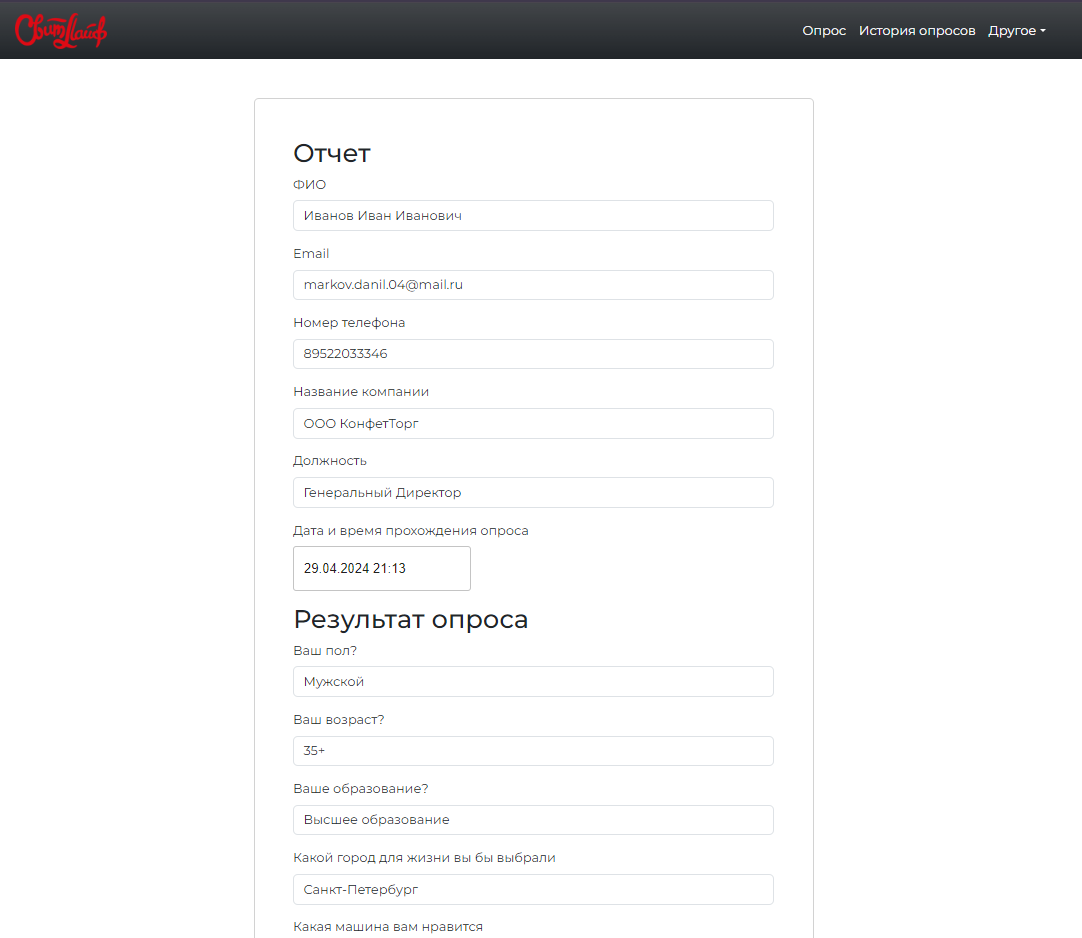


Рисунок 26 – Страница сохранения опроса

**2.3.2.2.3 Разработка модуля управления опросами**

Пользователю программы также доступен модуль управления опросами. На первой странице данного модуля представлены карточки с краткой информацией о пройденном опросе (рисунок 27). Каждая карточка содержит кнопку для выделения и редактирования. Страница редактирования пройденного опроса дублирует страницу на рисунке 26. Выделенные опросы можно удалить или выгрузить в Excel. Полученный файл представлен на рисунке 28. Сама страница реализует постраничный вывод информации. Содержание же страницы формируется из параметров URL адреса страницы.

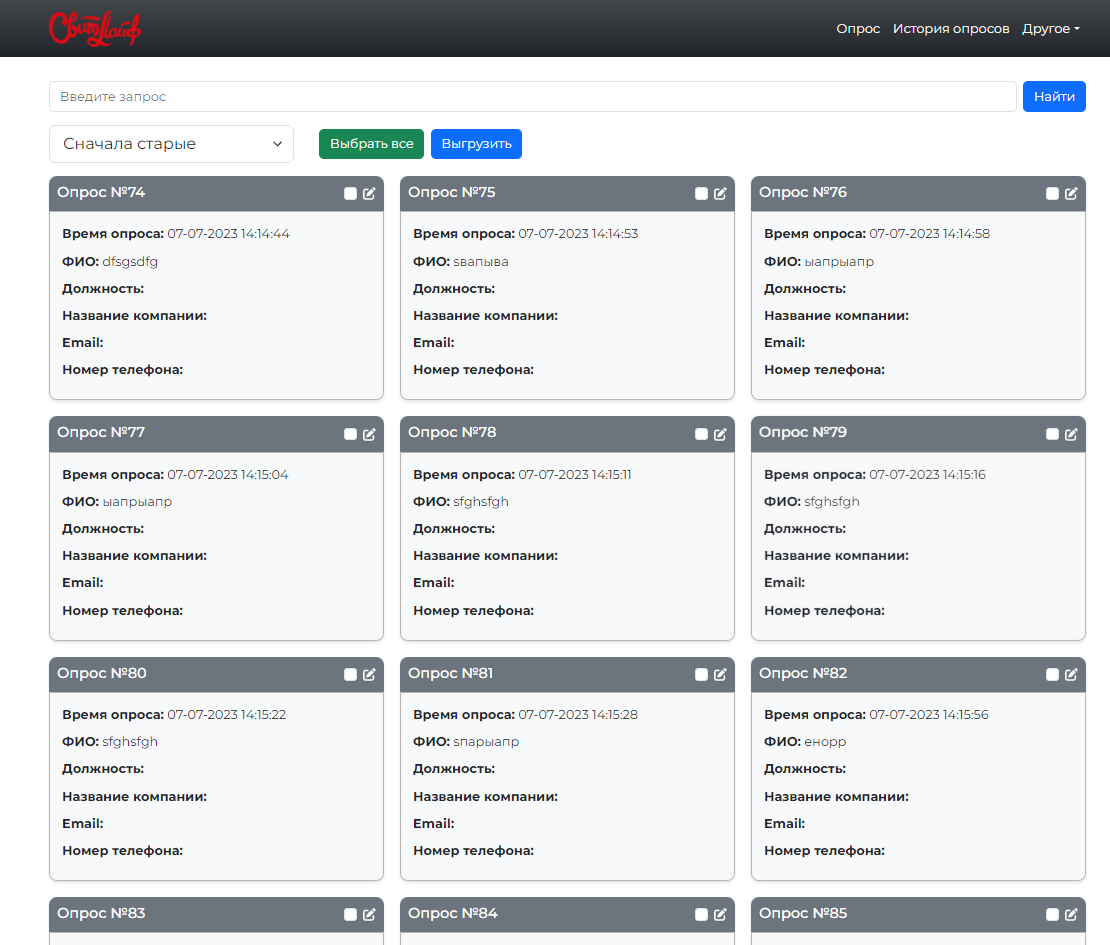


Рисунок 27 – Главная страница модуля управления опросами

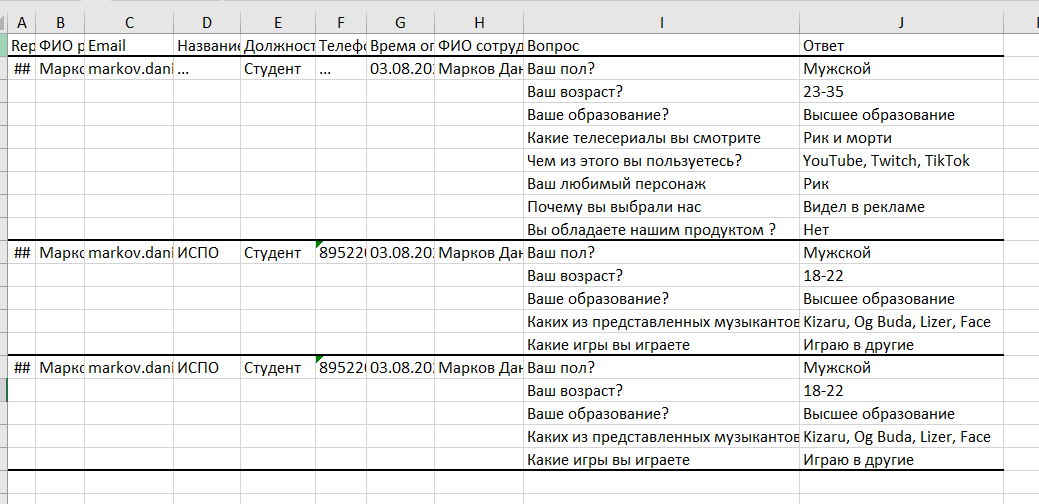


Рисунок 28 – Выгруженный отчет о выбранных опросах

В данном отчете содержится информация о респонденте, времени начала опроса и данных пользователем ответов на вопросы. Каждый опрос имеет свой идентификатор, а сами опросы визуально отделены друг от друга черной линией.

**2.3.2.2.4 Разработка модуля администратора**

Пользователь с ролью администратор имеет доступ к модулю администрирования. Данный модуль содержит две функции. Первая функция — это создание и редактирование пользователей. Вторая функция — создание опросов и управление ими.

Страница управления пользователями представлена на рисунке 29.

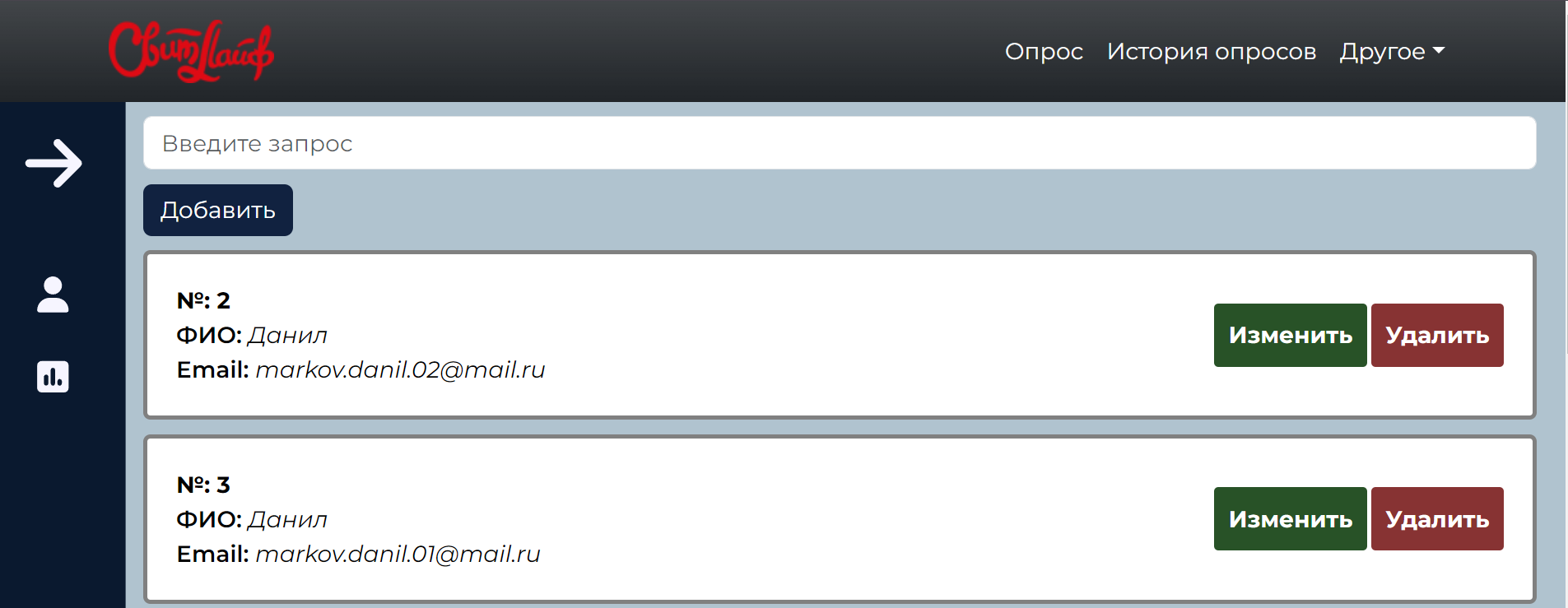


Рисунок 29 – Страница управления пользователями

На данной странице пользователь может добавить пользователя, редактировать информацию о нем или удалить. Окно добавления пользователя представлено на рисунке 30. Окно редактирования информации о пользователе представлено на рисунке 31.

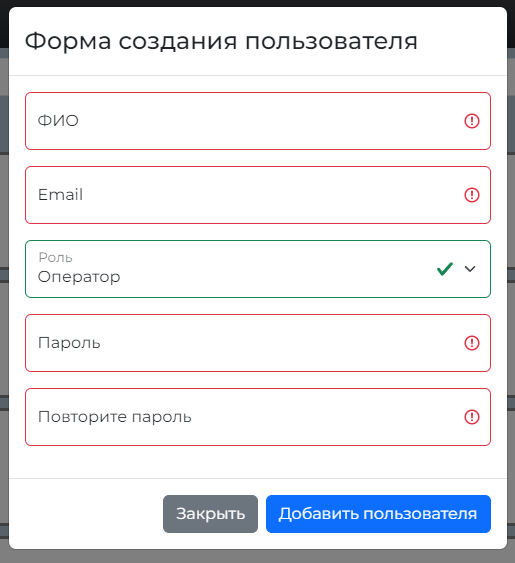


Рисунок 30 – Окно добавления пользователя

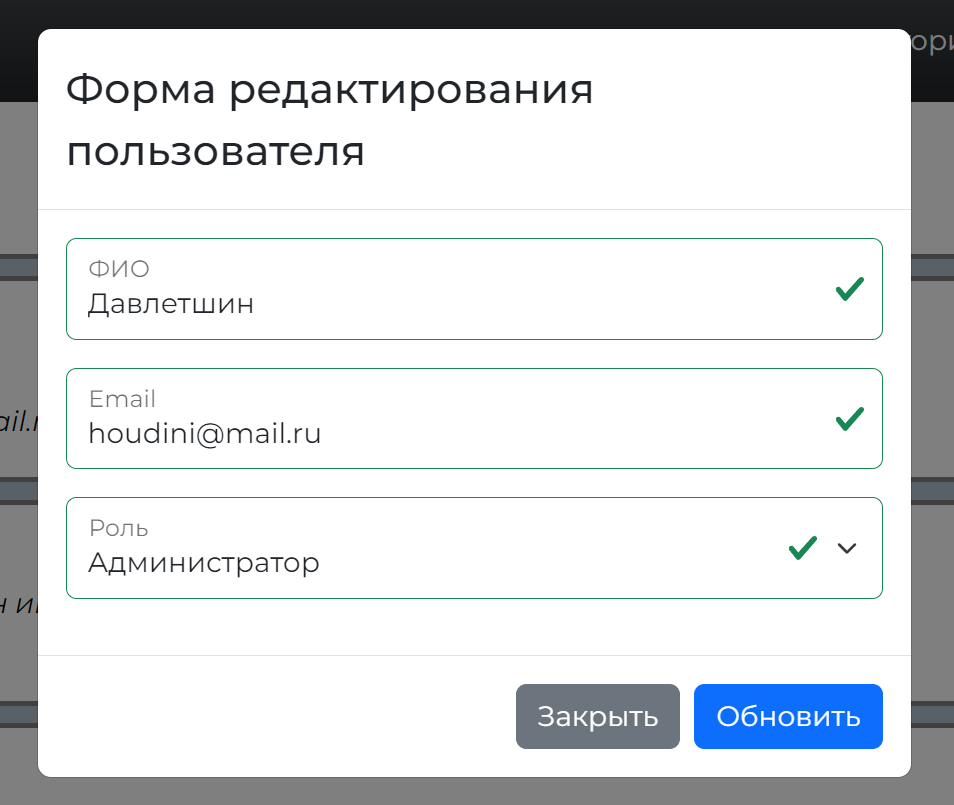


Рисунок 31 – Окно редактирования пользователя

Администратору также доступно окно управления опросами. Данная страница представлена на рисунке 32. Окно создания опроса представлено на рисунке 33.

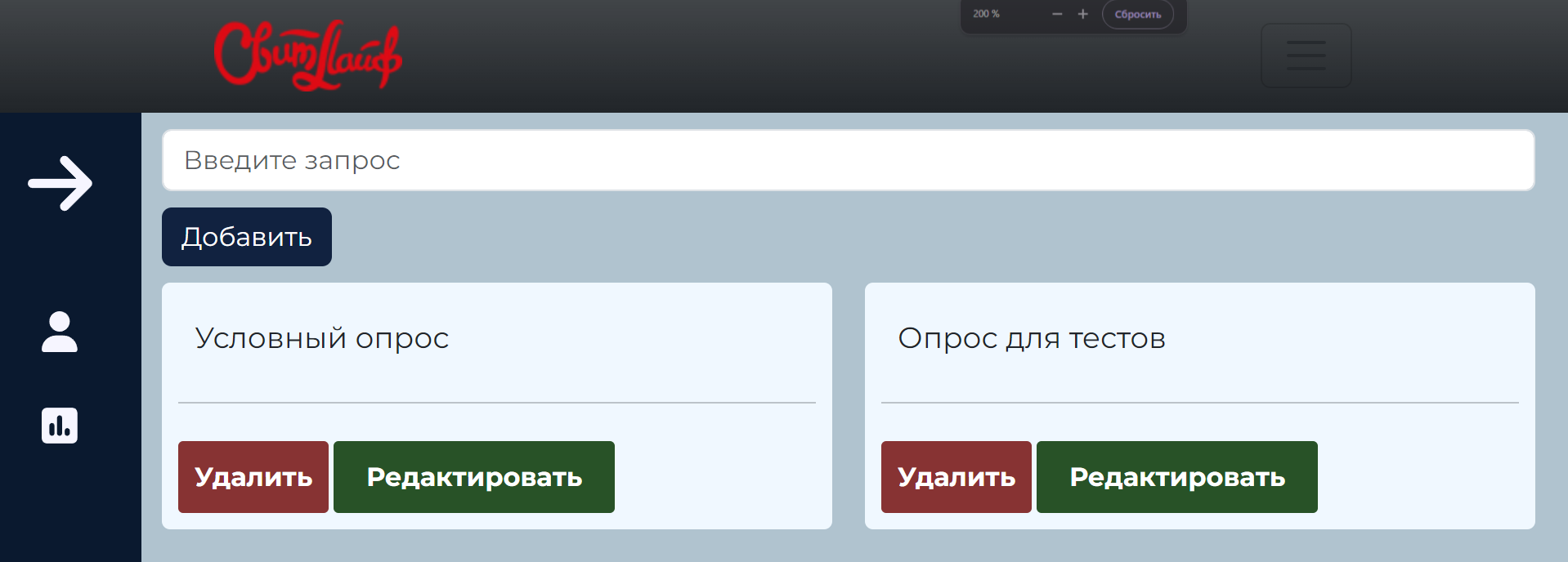


Рисунок 32– Страница управления опросами

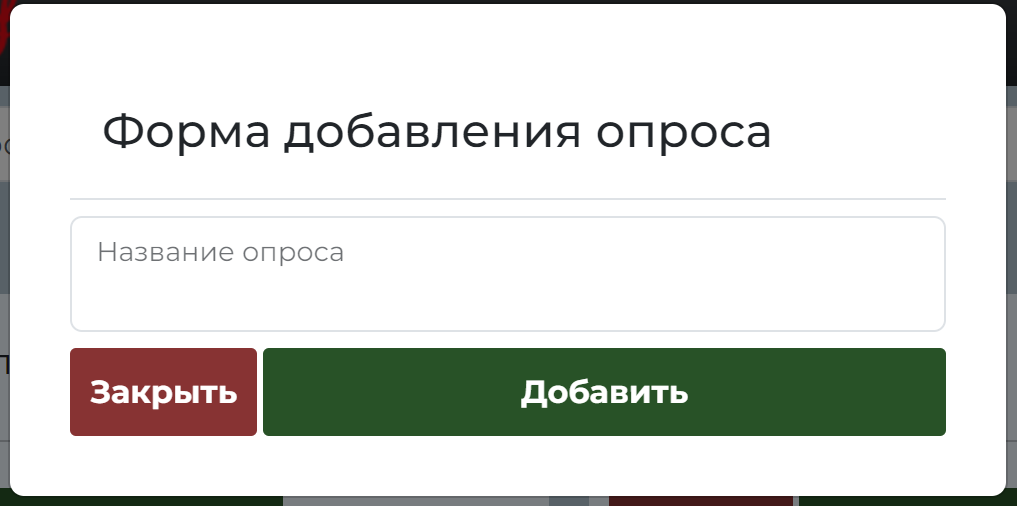


Рисунок 33 – Форма добавления опроса

При нажатии на кнопку редактировать администратор попадает в конструктор редактирования опроса. Данный конструктор представлен на рисунке 34.

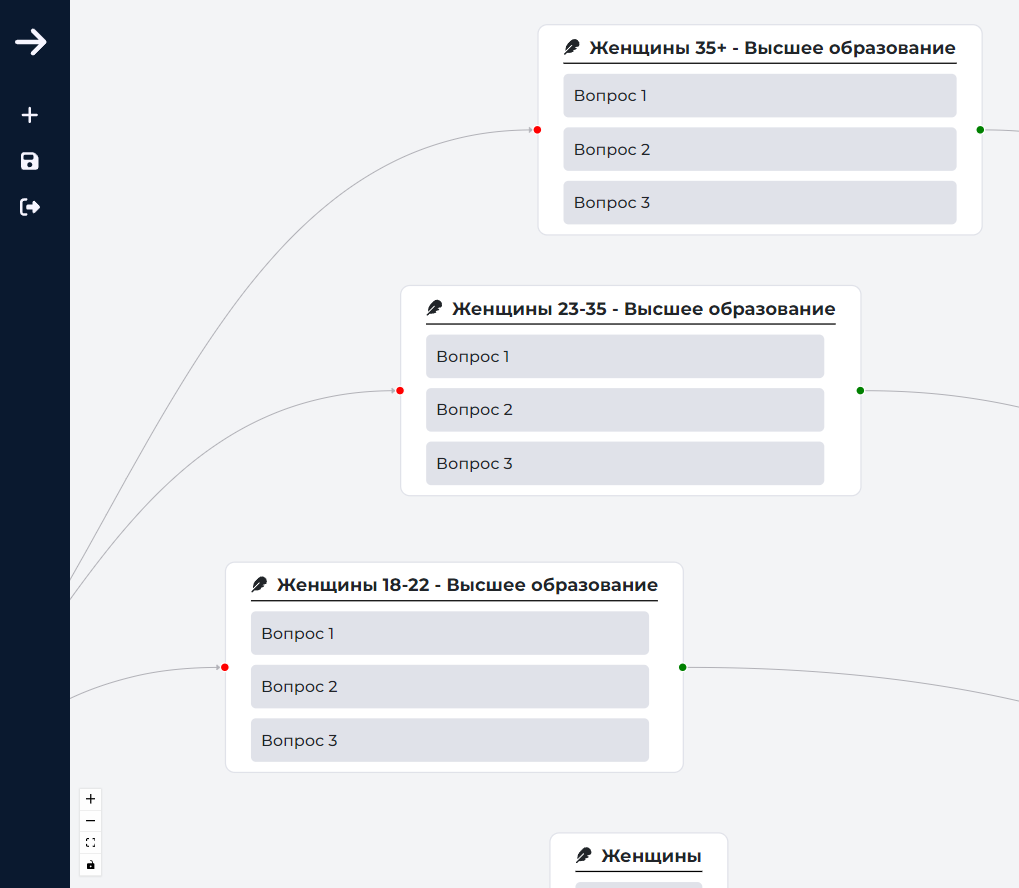


Рисунок 34 – Конструктор опросов

Данный конструктор поддерживает добавление блоков вопросов, добавление вопросов в блок, добавление условий переходов. На рисунке 35 представлено общее меню редактирования блока.

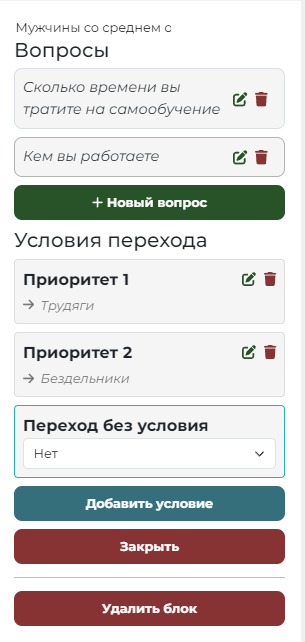


Рисунок 35 – Общее меню редактирования блока

Данное меню предоставляет возможность управления вопросами в блоке, их добавление, редактирование и удаление. Меню управления вопросом представлено на рисунке 36.

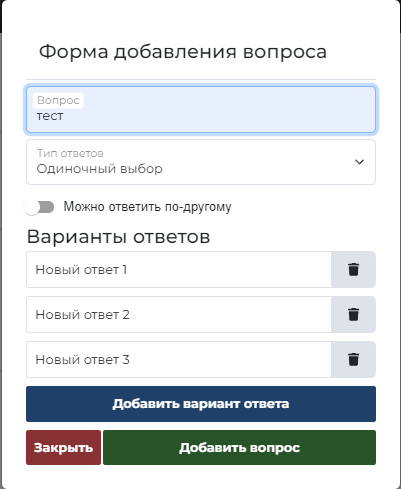


Рисунок 36 – Форма управление вопросом

Также меню предоставляет возможность создания, изменения и удаления условий переходов, форма управления условием перехода представлена на рисунке 37.

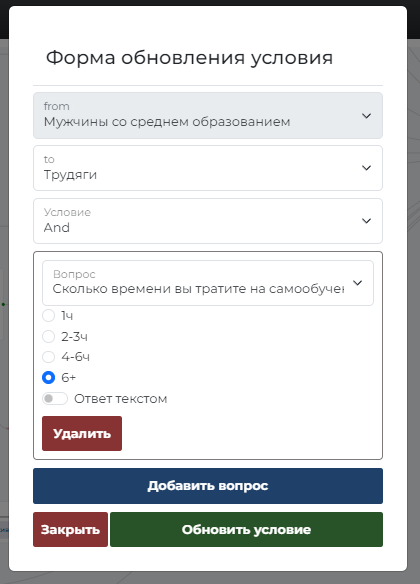


Рисунок 37 – Форма управления условием перехода

# **2.4 Тестирование приложения**

# **2.4 Вывод по специальной части**

В результате специальной части проекта была выполнена проектировка системы, включающая разработку диаграммы активности, диаграммы последовательностей, схемы базы данных и карты навигации. Для реализации системы было разработано серверное приложение с использованием фреймворка Express, а также клиентское приложение на базе React, React Router DOM, MobX и React Flow. Был реализован механизм авторизации с использованием JWT, обеспечивающий безопасность и надежность процесса аутентификации. В результате была достигнута цель по разработке веб приложения для создания и использования умных опросов в маркетинге.

# **ЗАКЛЮЧЕНИЕ**

В результате проделанной работы была достигнута цель. Было спроектировано и разработано веб приложение для создания и использования умных опросов в маркетинге.

В ходе работы все поставленные задачи были выполнены.

Разработанное решение позволяет автоматизировать и оптимизировать процесс создания и проведения опросов, что повышает их эффективность и упрощает работу маркетингового отдела. Внедрение данного веб-приложения может принести значительные выгоды для компаний, заинтересованных в повышении качества обратной связи от своих клиентов.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. РусПрофайл [Электронный ресурс] / ООО "РусПрофайл". – Электрон. дан. – Режим доступа : https://www.rusprofile.ru/id/6945587. – Загл. с экрана.
2. SurveyMonkey [Электронный ресурс]. – Электрон. дан. – Режим доступа: <https://ru.surveymonkey.com/>. – Загл. с экрана.
3. Qualtrics [Электронный ресурс]. – Электрон. дан. – Режим доступа: <https://www.qualtrics.com/uk/>. – Загл. с экрана.
4. Google Forms [Электронный ресурс]. – Электрон. дан. – Режим доступа: <https://www.google.com/forms/about/>. – Загл. с экрана.
5. Отличия монолитной и микросервисной архитектуры [Электронный ресурс]. – Режим доступа: https://www.atlassian.com/ru/microservices/microservices-architecture/microservices-vs-monolith. – Загл. с экрана.
6. Клиент-серверная архитектура в картинках [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/495698/>. – Загл. с экрана.
7. Принципы микросервисной архитектуры [Электронный ресурс]. – Режим доступа: <https://www.atlassian.com/ru/microservices>. – Загл. с экрана.
8. Крокфорд, Д. Как устроен JavaScript / Д. Крокфорд, - СПб. : Питер, 2019. – 304 с.
9. Документация для построения диаграммы последовательности [Электронный ресурс]. – Режим доступа: <https://intuit.ru/studies/courses/32/32/lecture/1014>, свободный. – Загл. с экрана.
10. Создание диаграммы активности. Учебное пособие [Электронный ресурс]. – Режим доступа: <https://anylogic.help/ru/advanced/actionchart/action-chart-tutorial.html>. – Загл. с экрана.

# Приложение А

(обязательное)

**Описание моделей данных**

const Quiz = sequelize.define('Quiz', {

  QuizId: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },

  Name: { type: DataTypes.STRING, primaryKey: true, unique: true },

  Survey: { type: DataTypes.JSON },

});

const Report = sequelize.define('Report', {

  ReportId: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },

  RespondentName: { type: DataTypes.STRING },

  Email: { type: DataTypes.STRING },

  CompanyName: { type: DataTypes.STRING },

  JobTitle: { type: DataTypes.STRING },

  PhoneNumber: { type: DataTypes.STRING, allowNull: false },

  QuizTime: { type: DataTypes.DATE, allowNull: false },

  UserId: { type: DataTypes.INTEGER, allowNull: true },

});

const User = sequelize.define('User', {

  UserId: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },

  FullName: { type: DataTypes.STRING, allowNull: false },

  Email: { type: DataTypes.STRING, unique: true },

  Password: { type: DataTypes.STRING, allowNull: false },

  RoleId: { type: DataTypes.INTEGER },

});

const Role = sequelize.define('Role', {

  RoleId: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },

  Name: { type: DataTypes.STRING, unique: true },

  Description: { type: DataTypes.STRING },

});

const PassedSurvey = sequelize.define('PassedSurvey', {

  QuizId: { type: DataTypes.INTEGER, primaryKey: true },

  ReportId: { type: DataTypes.INTEGER, primaryKey: true },

  Survey: { type: DataTypes.JSON, allowNull: false },

});

//Установка связи между таблицами Role и User

Role.hasMany(User, { foreignKey: 'RoleId' });

User.belongsTo(Role, { foreignKey: 'RoleId' });

//Установка связи между таблицами Report и User

Report.belongsTo(User, { foreignKey: 'UserId' });

User.hasMany(Report, { foreignKey: 'UserId', onDelete: 'SET NULL' });

// Установка связи между таблицей "Опросы" и "Пройденные опросы"

Quiz.hasMany(PassedSurvey, { foreignKey: 'QuizId' });

PassedSurvey.belongsTo(Quiz, { foreignKey: 'QuizId' });

// Установка связи между таблицей "Отчеты" и "Пройденные опросы"

Report.hasMany(PassedSurvey, { foreignKey: 'ReportId' });

PassedSurvey.belongsTo(Report, { foreignKey: 'ReportId' });

module.exports = {

  Quiz,

  Report,

  Role,

  User, PassedSurvey};

# Приложение Б

(обязательное)

**Программный код**

Программный код файла quizController.js

const ApiError = require('../error/ApiError');

const quizServices = require('../services/quizServices');

class QuizController {

  async addQuiz(req, res, next) {

    try {

      const { Survey } = req.body;

      const stringSurvey = JSON.stringify(Survey);

      const reqBody = req.body;

      const createdQuiz = await quizServices.createQuiz({

        ...reqBody,

        Survey: stringSurvey,

      });

      return res.json(createdQuiz);

    } catch (e) {

      return next(e);

    }

  }

  async getAllQuiz(req, res, next) {

    const getedQuizzes = await quizServices.getAllQuiz();

    if (!getedQuizzes)

      return next(

        ApiError.badRequest('Поиск по всем опросам не дал результатов'),

      );

    return res.json(getedQuizzes);

  }

  async getOneQuiz(req, res, next) {

    const { id } = req.params;

    console.log('id', id);

    const getedQuiz = await quizServices.getQuizById(id);

    if (!getedQuiz)

      return next(ApiError.badRequest('Опроса с таким id не существует'));

    return res.json(getedQuiz);

  }

  async deleteQuiz(req, res, next) {

    const { id } = req.params;

    const isAlive = await quizServices.getQuizById(id);

    if (!isAlive)

      return next(ApiError.badRequest('Опроса с таким id не существует'));

    const deletedQuiz = await quizServices.deleteQuizById(id);

    if (!deletedQuiz) return next(ApiError.badRequest('Ошибка при удалении'));

    return res.json(deletedQuiz);

  }

  async updateOneQuiz(req, res, next) {

    const { QuizId, Survey } = req.body;

    const isAlive = await quizServices.getQuizById(QuizId);

    if (!isAlive)

      return next(

        ApiError.badRequest(`Не существует такого опроса с id = ${QuizId}`),

      );

    const stringSurvey = JSON.stringify(Survey);

    const reqBody = req.body;

    const updatedQuiz = await quizServices.updateQuiz({

      ...reqBody,

      Survey: stringSurvey,

    });

    if (!updatedQuiz)

      return next(

        ApiError.badRequest('При обновлении опроса произошла ошибка'),

      );

    return res.json(updatedQuiz);

  }

}

module.exports = new QuizController();

Программный код файла userController.js

const ApiError = require('../error/ApiError');

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const { User } = require('../models/models');

const userServices = require('../services/userServices');

generateJwtToken = (UserId, Email, RoleId, FullName) => {

  return jwt.sign({ UserId, Email, RoleId, FullName }, process.env.SECRET\_KEY, {

    expiresIn: '24h',

  });

};

class UserController {

  async registration(req, res, next) {

    const { Email, Password, RoleId, FullName } = req.body;

    const senderRoleId = req.user.RoleId;

    if (senderRoleId < RoleId)

      return next(

        ApiError.badRequest(

          'Нельзя создать пользователя, уровень доступа которого выше вашего',

        ),

      );

    if (!Email || !Password || !RoleId || !FullName) {

      return next(ApiError.badRequest('Не все поля заполнены'));

    }

    const candidate = await userServices.checkEmail(Email);

    if (candidate) {

      return next(

        ApiError.badRequest('Пользователь с таким email уже зарегистрирован'),

      );

    }

    const hashPassword = await bcrypt.hash(Password, 5);

    const user = await userServices.create({

      Email,

      Password: hashPassword,

      RoleId,

      FullName,

    });

    const jwtToken = generateJwtToken(

      user.UserId,

      user.Email,

      user.RoleId,

      user.FullName,

    );

    return res.json({ jwtToken });

  }

  async login(req, res, next) {

    const { Email, Password } = req.body;

    const candidate = await userServices.checkEmail(Email);

    if (!candidate) {

      return next(

        ApiError.internal('Пользователь с таким email не существует'),

      );

    }

    const comparePassword = bcrypt.compareSync(Password, candidate.Password);

    if (!comparePassword) {

      return next(ApiError.badRequest('Неверный пароль'));

    }

    const jwtToken = generateJwtToken(

      candidate.UserId,

      candidate.Email,

      candidate.RoleId,

      candidate.FullName,

    );

    return res.json({ jwtToken });

  }

  async check(req, res, next) {

    const newData = await userServices.getUserById(req.user.UserId);

    const token = generateJwtToken(

      req.user.UserId,

      newData.Email,

      newData.RoleId,

      newData.FullName,

    );

    return res.json({ token });

  }

  async delete(req, res, next) {

    const { id } = req.params;

    const user = await userServices.getUserById(id);

    const senderRoleId = req.user.RoleId;

    if (senderRoleId <= user.RoleId)

      return next(

        ApiError.badRequest(

          'Нельзя удалить пользователя, уровень доступа которого равен или выше вашего',

        ),

      );

    const deleteUser = await userServices.deleteUserById(id);

    return res.json(deleteUser);

  }

  async update(req, res, next) {

    const { UserId, Email, RoleId, FullName } = req.body;

    const getedUser = await userServices.getUserById(UserId);

    const checkEmail = await userServices.checkEmail(Email);

    if (!getedUser)

      return next(ApiError.badRequest('Пользователя не существует'));

    if (req.user.RoleId < RoleId)

      return next(

        ApiError.badRequest(

          'Ваш уровень доступа меньше, чем тот который вы хотите присвоить',

        ),

      );

    if (

      req.user.RoleId <= getedUser.RoleId &&

      req.user.UserId !== getedUser.UserId

    )

      return next(

        ApiError.badRequest(

          'Нельзя редактировать, уровень доступа выше или равен вашему',

        ),

      );

    if (checkEmail && Email !== getedUser.Email)

      return next(

        ApiError.badRequest('Пользователя с таким Email уже существует'),

      );

    const updatedUser = await userServices.updateUser(req.body);

    console.log('updatedUser', updatedUser);

    return res.json(updatedUser);

  }

  async getUsers(req, res, next) {

    const RoleId = req.user.RoleId;

    if (RoleId === 1) return next(ApiError.badRequest('Доступ запрещен'));

    const getedUsers = await userServices.getUsers();

    return res.json(getedUsers);

  }

  async getUser(req, res, next) {

    const RoleId = req.user.RoleId;

    const { id } = req.params;

    if (RoleId === 1) return next(ApiError.badRequest('Доступ запрещен'));

    const getedUser = await userServices.getUser(id);

    return res.json(getedUser);

  }

}

module.exports = new UserController();

Функция управления узлами

import React, { useState, useEffect } from 'react';

export const useSurveyNodes = survey => {

  const [nodes, setNodes] = useState([]);

  useEffect(() => {

    // if (survey.length === 0) return;

    const extractedNodes = survey.Survey.blocks.map(node => ({

      id: node.id + '',

      type: 'questionBlock',

      data: { block: node },

      position: {

        x: parseFloat(node.position.x),

        y: parseFloat(node.position.y),

      },

    }));

    setNodes(extractedNodes);

  }, [survey]);

  return [nodes, setNodes];

};

Функция управления связями

import { useState, useEffect } from 'react';

import { MarkerType } from 'reactflow';

export const useSurveyEdges = survey => {

  const [edges, setEdges] = useState([]);

  useEffect(() => {

    const extractedEdges = survey.Survey.blocks.reduce((arr, block) => {

      const currentId = block.id + '';

      if (!block.nextBlock) return arr;

      block.nextBlock.condition.forEach(condition => {

        const nextBlockId = condition[0].blockId + '';

        const edgeId = currentId + '-' + nextBlockId;

        const edge = {

          id: edgeId,

          source: currentId,

          target: nextBlockId,

          markerEnd: {

            type: MarkerType.ArrowClosed,

          },

        };

        if (arr.find(condition => condition.id === edgeId)) return;

        arr.push(edge);

      });

      if (block.nextBlock.unconditionallyJump !== -1) {

        const unconditionallyJump = block.nextBlock.unconditionallyJump;

        const edgeId =

          currentId + '-' + unconditionallyJump + '-unconditionally';

        const edge = {

          id: edgeId,

          source: currentId,

          target: unconditionallyJump,

          markerEnd: {

            type: MarkerType.ArrowClosed,

          },

        };

        arr.push(edge);

      }

      return arr;

    }, []);

    setEdges(extractedEdges);

  }, [survey]);

  return [edges, setEdges];

};

**АТТЕСТАЦИОННЫЙ ЛИСТ**

**ПО ПРЕДДИПЛОМНОЙ ПРАКТИКЕ (ПО ПРОФИЛЮ СПЕЦИАЛЬНОСТИ)**

Специальность09.02.07 Информационные системы и программирование

(код и наименование специальности)

Студент(ка) 4 курса 42919/4 группы

Марков Данил Петрович

(Фамилия, имя, отчество)

Место прохождения практики: ООО «СЛ Маркет», Санкт-Петербург, наб. Обводного канала, д. 93а литер а, помещ. 5н офис 1449

(наименование и адрес организации)

Период прохождения практики

с «22» апреля 2024 г. по «18» мая 2024 г.

**Виды и качество выполнения работ**

|  |  |  |
| --- | --- | --- |
| **Виды выполненных работ обучающимся**  **во время практики** | **Объем работ, час.** | **Качество выполнения работ (оценка по пятибалльной системе)** |
| **Тема 1.** Ознакомление со структурой и характером деятельности предприятия. Организационные вопросы оформления на предприятии, установочная лекция, инструктаж по охране труда и технике безопасности, распределение по рабочим местам | 30 |  |
| **Тема 2**. Сбор материалов для составления технического задания по теме дипломного проекта | 26 |  |
| **Тема 3.** Разработка программного обеспечения на основе технического задания дипломного проекта | 30 |  |
| **Тема 4.** Проведение испытаний, отладка и внедрение программного продукта на предприятии | 16 |  |
| **Тема 5.** Расчет показателей экономической эффективности программного продукта | 22 |  |
| **Тема 6.** Оформление отчета о прохождении производственной практики (преддипломной) | 20 |  |

**Характеристика учебной/профессиональной деятельности обучающегося во время учебной практики (по профилю специальности):**

Общие и профессиональные компетенции, предусмотренные программой практики, **освоены /** не освоены.

(нужное подчеркнуть)

Итоговая оценка по практике \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель практики Иванова Д.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О.) (подпись)

Дата «18» мая 2024 г.