

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

*К защите допустить:*

Заведующая кафедрой ПОИТ

\_\_\_\_\_ Н. В. Лапицкая

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту  
на тему:

**ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ  
РАЗРАБОТКИ ФОРМ ОПРОСНИКОВ**  
БГУИР ДП 1-40 01 01 03 072 ПЗ

Студент

В.В. Левшиц

Руководитель

В.Л. Богомаз

Консультанты:

*от кафедры ПОИТ  
по экономической части*

В.А. Леванцевич  
Т. А. Рыковская

Нормоконтролер

С. В. Болтак

Рецензент

Минск 2017

## РЕФЕРАТ

Пояснительная записка 122с., 46 рис., 3 табл., 14 источников

### ПРОГРАММНОЕ СРЕДСТВО, СОЦИАЛЬНЫЕ ИССЛЕДОВАНИЯ, МЕТОД ОПРОСА, АНКЕТА, ОПРОСНИК

Объектом исследования данного дипломного проекта является автоматизация процессов проведения социальных или маркетинговых исследований.

Цель работы – проектирование и разработка программного средства, решающего задачу автоматизации процессов разработки анкеты опросника, распространения анкеты, сбора и анализа результатов заполнения.

Разработка и использование описанного программного средства позволяет упростить процесс проведения социальных исследований, а также сократить временные и экономические затраты, связанные с этим процессом.

В процессе работы над проектом были изучены подходы к формализации социальных исследований, основные тенденции и направления развития в этой области, были изучены подходы к реализации программных средств по теме дипломного проектирования, разработана и реализована программная архитектура с возможностью дальнейшего масштабирования.

В разделе технико-экономического обоснования были произведены расчеты затрат, связанных с реализацией проекта, а также рентабельности разработки проекта. Проведенные расчеты показали экономическую целесообразность проекта.





## СОДЕРЖАНИЕ

Введение .....	7
1 Аналитический обзор программных продуктов, методов и подходов по теме дипломного проектирования.....	8
1.1 Основные понятия и определения в области анкетирования .....	8
1.2 Обзор тенденций в области проведения анкетирования.....	10
1.3 Анализ существующих программных решений по теме дипломного проектирования .....	11
1.4 Постановка целей и задач дипломного проектирования .....	22
2 Со моделирование предметной области, разработка функциональных требований и составление их спецификации .....	23
2.1 Общие сведения и требования к работе программного средства .....	23
2.2 Описание функциональности программного средства .....	23
2.3 Разработка информационной модели.....	31
2.4 Разработка модели взаимодействия пользователя с интерфейсом .....	33
2.5 Разработка спецификации функциональных требований.....	34
2.6 Разработка технических требований к программному средству .....	35
3 Проектирование архитектуры программного средства .....	36
3.1 Разработка архитектуры программного средства.....	36
3.2 Проектирование архитектуры базы данных .....	40
3.3 Проектирование алгоритмов ПС для проведения онлайн-исследований ....	40
4 Разработка программного обеспечения .....	45
4.1 Выбор и обоснование языка и среды разработки программного средства ..	45
4.2 Диаграмма классов программного средства.....	48
4.3 Описание компонент клиентской части программного средства .....	53
6 Руководство пользователя .....	59
6.1 Развертывание сервера приложения .....	59
6.2 Использование программного средства.....	60
7 Техничко-экономическое обоснование разработки и использования программного обеспечения.....	70
7.1 Характеристика программного обеспечения разработки форм опросников	70
7.2 Расчет затрат на разработку программного средства.....	70
7.3 Оценка экономического эффекта у разработчика программного средства .	73
Заключение.....	76
Список использованной литературы .....	77
Приложение А Текст программного средства .....	78

## ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

*Авторизация* – предоставление определённому лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

*Аутентификация* – проверка подлинности предъявленного пользователем идентификатора.

*Инициализация* – приведение областей памяти в состояние, исходное для последующей обработки или размещения данных.

*Интерпретатор* – программа или техническое средство, выполняющие интерпретацию.

*Программа* – данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определённого алгоритма.

*Программное обеспечение* – совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

*Программирование* – научная и практическая деятельность по созданию программ.

*Программный модуль* – программа или функционально завершённый фрагмент программы, предназначенный для хранения, трансляции, объединения с другими программными модулями и загрузки в оперативную память.

*Спецификация программы* – формализованное представление требований, предъявляемых к программе, которые должны быть удовлетворены при ее разработке, а также описание задачи, условия и эффекта действия без указания способа его достижения.

*Фреймворк* – программное обеспечение, облегчающее разработку и объединение различных компонентов большого программного проекта.

ООП – объектно-ориентированное программирование

БД – база данных

ПС – программное средство

ОС – операционная система

API – application programming interface (Интерфейс программирования приложений)

JSON – javascript object notation

URL – uniform resource locator (единообразный локатор ресурса)

## ВВЕДЕНИЕ

Вопросы и ответы являются неотъемлемой частью жизни каждого человека. С самого детства мы задаем огромное количество вопросов. Таким образом мы постигаем окружающий мир. Очень редко, для получения корректного ответа, нам необходимо задать один или два вопроса, в большинстве случаев для этого нам потребуется задать десятки самых разных вопросов, которые зависят друг от друга. Во многих ситуациях задать вопрос – это единственный возможный способ получения информации в нашей жизни.

Разного рода опросники плотно проникли в наш быт и на данный момент используется повсеместно. Мы заполняем тесты при приеме на работу, при поступлении в учебное заведение, при покупке того или иного товара. Самые разные анкеты и тесты используются при проверке знаний, для выполнения различных социологических исследований, при исследованиях рынка, для выяснения личностных качеств. Крайне важным этапом анкетирования является не только подготовка вопросов, но и анализ результатов опроса.

Еще несколько лет назад основную массу анкет составляли бумажные анкеты, которые заполнялись ручкой или карандашом, и, чаще всего, анализировались вручную. Однако на данный момент всё большую популярность набирает электронное анкетирование. Данный тип анкетирования обладает целым рядом преимуществ: сравнительная простота подготовки анкет, легкость их распространения, и самое главное низкая стоимость.

Ещё одним немаловажным преимуществом электронного анкетирования можно назвать возможность автоматического анализа заполненной анкеты. В этом случае у человека, который заполнил анкету, нет необходимости ждать, пока специалист проанализирует результаты заполнения и выдаст результат. В процессе составления анкеты задаются правила, по которым впоследствии вычисляются результаты опроса. Таким образом анкетирование можно свести к подготовке анкеты. Возможности повторного использования уже существующих анкет, их частичного изменения без необходимости полностью выполнить всю работу заново, позволяют повысить эффективность работы специалиста.

На данный момент в этой области существует достаточное количество приложений, ориентированных на проведение анкетирования в различных областях применения. Однако функциональность большинства из них предлагает лишь формирование списка вопросов, исключая возможность задания правил формирования результатов, а также правил отображения вопросов в зависимости от ранее введенных ответов.

Исходя из вышесказанного, целью данной работы является анализ существующих решений в этой области, а также проектирование и разработка приложения для формирования форм опросников.

# **1 АНАЛИТИЧЕСКИЙ ОБЗОР ПРОГРАММНЫХ ПРОДУКТОВ, МЕТОДОВ И ПОДХОДОВ ПО ТЕМЕ ДИПЛОМНОГО ПРОЕКТИРОВАНИЯ**

## **1.1 Основные понятия и определения в области анкетирования**

Анкетирование – психологический вербально-коммуникативный метод, в котором в качестве средства для сбора сведений от респондента используется специально оформленный список вопросов – анкета [1].

В социологии под анкетированием понимают метод опроса, который используется для составления статических (при однократном анкетировании) или динамических (при многократном анкетировании) статистических представлений о состоянии общественного мнения, состоянии политической, социальной и прочей напряженности для того, чтобы спрогнозировать дальнейшие действия или события [2].

Анкетирование в психологии используется для получения психологической информации. В данном случае социологические и демографические данные играют лишь вспомогательную роль. Непосредственное взаимодействие специалиста с респондентом в этом случае сведено к минимуму, в отличие от, например, интервьюирования.

Использование анкетирования позволяет с наименьшими затратами на проведение достичь высокий уровень массовости исследования, что крайне важно для обеспечения репрезентативности выборки. Особенностью данного метода является возможность обеспечить анонимность респондента. Анкетирование проводится в случаях, когда необходимо выяснить мнение по какому-либо вопросу, при этом охватив большое число людей [3-4].

Анкетирование можно классифицировать по следующим признакам:

- по числу респондентов;
- по полноте охвата;
- по типу контактов с респондентом.

По числу респондентов в анкетировании выделяют следующие виды:

- индивидуальное – опрашивается один респондент;
- групповое – опрашиваются несколько респондентов;
- аудиторное – методическая и организационная разновидность анкетирования, состоящая в одновременном заполнении анкет группой людей, собранных в одном помещении в соответствии с правилами выборочной процедуры;
- массовое – участвуют от сотни до нескольких тысяч респондентов (на практике работа трудоёмкая, а результаты менее корректны).



По полноте охвата в анкетировании выделяют следующие разновидности:

- сплошное – опрос всех представителей выборки;
- выборочное – опрос части представителей выборки.

По типу контактов с респондентом анкетирование можно разделить на следующие типы:

- очное – проводится в присутствии исследователя-анкетёра;
- заочное – анкетёр отсутствует.

В более общем случае мы можем рассматривать анкетирование как одну из разновидностей опросов.

Метод опроса – психологический вербально-коммуникативный метод, суть которого заключается в осуществлении взаимодействия между интервьюером и респондентами (людьми, участвующими в опросе), посредством получения от субъекта ответов на заранее сформулированные вопросы. Иными словами, опрос представляет собой общение интервьюера и респондента, в котором главным инструментом выступает заранее сформулированный вопрос.

Опрос можно рассматривать как один из самых распространённых методов получения информации о субъектах – респондентах опроса.

Опросы разделяют на стандартизированные и не стандартизированные. Стандартизированные опросы можно рассматривать как строгие опросы, дающие, прежде всего общее представление об исследуемой проблеме. Не стандартизированные опросы менее строгие в сравнении со стандартизированными, в них отсутствуют жёсткие рамки. Они позволяют варьировать поведение исследователя в зависимости от реакции респондентов на вопросы [5].

Выделяют следующие виды опросов:

- анкетирование;
- метод лестницы;
- свободный;
- устный;
- письменный;
- стандартизованный.

Существует целый ряд правил составления вопросов для создания опроса:

- каждый вопрос должен быть логичным и отделенным от остальных;
- запрещено употребление малораспространенных слов и понятий;
- вопрос должен быть кратким;
- формулировка вопроса должно оставаться лаконичной, но при необходимости вопрос может сопровождаться пояснением.
- вопросы должны быть максимально конкретны;
- вопрос не должен содержать подсказки;

- формулировка вопроса должна предотвращать получение шаблонных ответов;

- недопустимы вопросы внушающего характера.

Вопросы, входящие в состав опросников, можно отнести к одному из следующих типов:

- закрытые (структурированные) предполагают выбор ответа из списка. закрытые вопросы могут быть дихотомическими («да/нет») или же с множественным выбором, то есть предоставлять более двух вариантов ответа. ответы на закрытые вопросы легко поддаются обработке; недостатком же можно считать высокую вероятность необдуманности ответов, случайный их выбор, автоматизм у респондента;

- открытые (неструктурированные) оправданы на стадии проб, пилотажа, определения области исследования и в качестве контроля. ответы на открытый вопрос позволяют выявить динамику мнений, оценок, настроений, ценностных ориентаций и пр.

При проведении опросов специалисты сталкиваются с определенными ошибками и неточностями: неполучение ответов, ошибки в ответах, неправильные формулировки вопросов, неправильный охват зоны наблюдения [6].

## **1.2 Обзор тенденций в области проведения анкетирования**

Каждая компания в определенный момент своей деятельности проводит различные социальные исследования. Невозможно опираться только на свое мнение при старте нового проекта или же принятии решения о необходимости развития уже существующих решений. Собственное мнение не всегда правильное, но зачастую не учитывает всех аспектов, вариантов развития событий. Зачастую точка зрения других людей отличается от нашей по одним и тем же вопросам.

На помощь компаниям в этом случае приходят онлайн-исследования. На данный момент – это самый быстрый способ собрать информацию из первых уст. Провести исследование можно за один день, миновав длительные разборы результатов, их структурирование и анализ.

Разумеется, что метод проведения онлайн-исследований применим не всегда, и все еще много областей, где необходимо использование проверенных временем методов. Однако существует целый ряд опросов, которые намного удобнее проводить онлайн. И стоит отметить, что таких исследований довольно много.

Современные сервисы позволяют сделать многое. Теперь не важно, есть ли у вас собственная база респондентов. Вы можете воспользоваться уже собранной кем-то, отобрав по определенным параметрам свою собственную аудито-

рию. В одном месте вы можете и создать электронную анкету, и провести исследование, и проанализировать его результаты. При всем этом процесс сбора информации как никогда прозрачен.

Кроме всего сказанного такой формат позволяет использовать более широкий географический охват. Распространение интернета делают онлайн-исследования все более привлекательными для конечных пользователей.

Такой формат обладает целым рядом преимуществ:

- гибкие возможности управления отображением вопросов;
- скорость составления анкеты;
- скорость получения ответов на формы опроса;
- сравнительно низкая стоимость проведения исследования;
- высокая скорость обработки результатов
- простота исправления ошибок, возникающих в процессе разработки формы опроса.

Однако стоит отметить, что у данного метода исследований есть и недостатки:

- не все респонденты имеют доступ к сети интернет;
- связанные с идентификацией респондента;
- выборка респондентов может быть не репрезентативна.

Учитывая вышеописанное важно отметить, что, несмотря на возрастающую популярность метода онлайн-исследований, нельзя говорить о том, что данный метод способен полностью заметить другие методы проведения социальных исследований. На данный момент времени данный метод широко используется в комплексе с более традиционными методами.

Исходя из вышесказанного, можно прийти к такому выводу: онлайн исследования на данный момент являются менее затратной альтернативой маркетинговым агентствам. Однако с рядом ограничений, связанных с малой глубиной проводимый исследований.

### **1.3 Анализ существующих программных решений по теме дипломного проектирования**

Наличие спроса на сервисы для проведения онлайн исследований привело к тому, что был разработан целый ряд программных средств, которые призваны удовлетворить потребность. На данный момент число таких сервисов не ограничивается одним десятком и среди них мы сможем найти как максимально универсальные – Google Forms, так и сервисы, ориентированные на решение узкоспециализированных задач – Kahoot!.

### 1.3.1 Google Forms

Одним из самых известных сервисов, которые позволяют конструировать опросники является сервис от компании Google. Его популярность обусловлена целым рядом факторов, среди которых и простота создания формы, и отсутствие необходимости оплаты его функционирования.

На рисунке 1.1 представлен один из основных экранов, с которым осуществляется забота при создании формы будущего опросника. Легко заметить, что внешний вид формы в режиме редактирования практически не отличается от готового опросника. Единственным отличием в данном случае является наличие дополнительной панели, предназначенной для добавления новых вопросов различных типов.

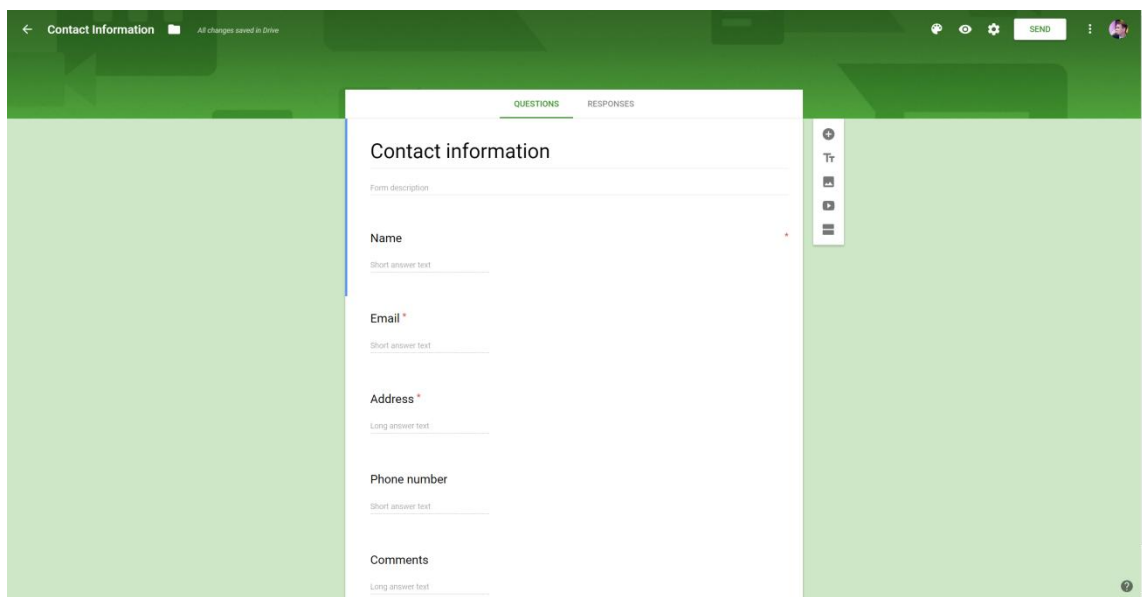


Рисунок 1.1 – Пример опросника Google Forms открытого в режиме редактирования.

Google Forms позволяет создавать вопросы следующих типов:

- текст (строка) – тип вопроса, который предполагает текстовый ответ в одну строку;
- текст (абзац) – тип вопроса, на который следует дать развернутый ответ;
- один из списка – тип вопроса, предоставляющий пользователю выбор одного варианта из ряда предложенных;
- несколько из списка – респонденту предлагается выбрать один или большее количество предложенных вариантов;
- раскрывающийся список – еще один тип вопроса, предполагающий выбор одного ответа из списка;

- шкала – тип вопроса, ответ на который отражает степень согласия респондента с утверждением в вопросе;
- сетка – тип вопроса, позволяющий пользователю выбрать несколько ответов на вопрос;
- дата – вопросы этого типа, ожидают ответ в формате даты;
- время – данный тип вопроса предполагает ответ в формате времени.

На рисунке 1.2 представлен диалог выбора типа вопроса в процессе конструирования формы опросника.

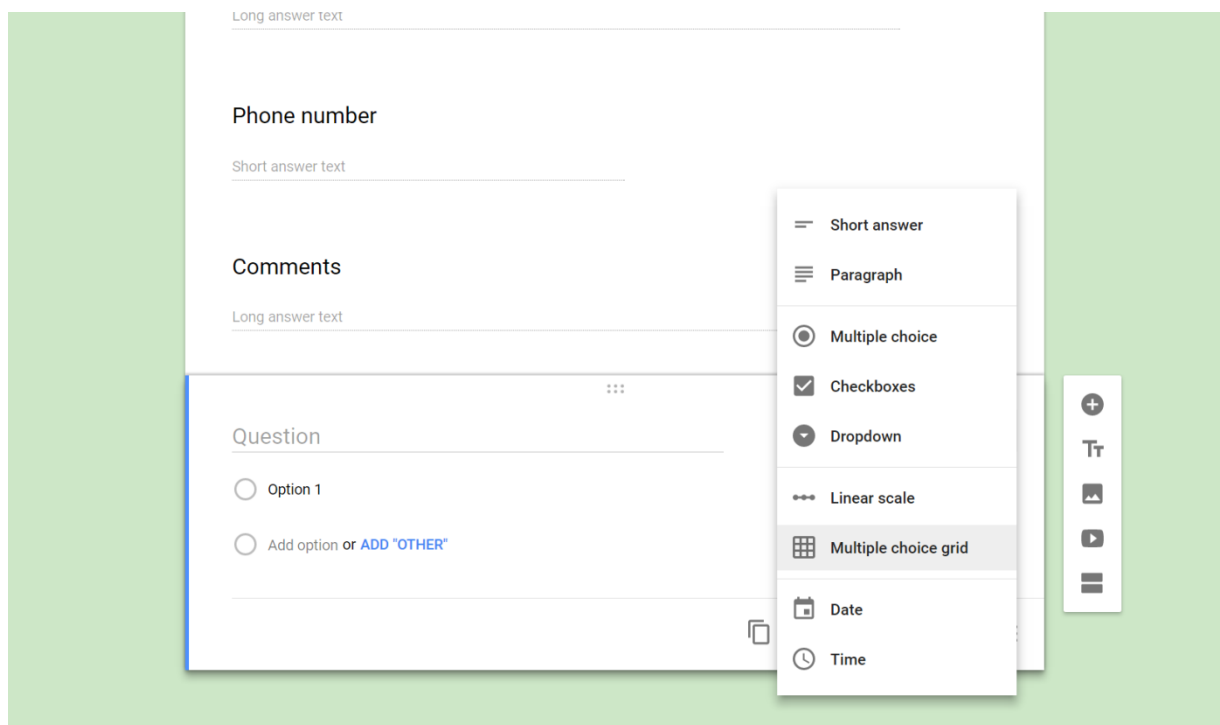


Рисунок 1.2 – Выбор типа вопроса в Google Forms опроснике

Каждый из перечисленных типов вопросов может быть помечен как обязательный для заполнения. Кроме этой характеристики, разные типы вопросов позволяют задать некоторые дополнительные параметры, среди которых: правила валидации введенного ответа, действие, которое необходимо выполнить в зависимости от выбранного ответа, способ отображение вариантов ответов и многие другие.

Одной из важных отличительных черт данного сервиса можно назвать строго выдержанное оформление, как элементов редактора, так и элементов готовой формы. Google Forms позволяют как выбрать одну из стандартных тем оформления, так и доработать предложенное отображение формы до необходимого, меняя цветовое оформление, добавляя медиа-контент, гиперссылки и т.д. Все

элементы управления выполнены в соответствии с правилами Material Design, что позволяет получить унифицированное отображение формы на самых разных типах устройств. Благодаря этому, нет необходимости в доработке формы для обеспечения корректного отображения на мобильных устройствах.

При работе с сервисом пользователь может задать настройки заполнения для каждой отдельной формы. Пример диалога задания глобальных настроек отображен на рисунке 1.3. Здесь мы можем увидеть возможность ограничения количества вариантов ответов на экземпляр формы, а также ряд параметров, которые позволяют пользователю редактировать свой ответ после сохранения результатов.

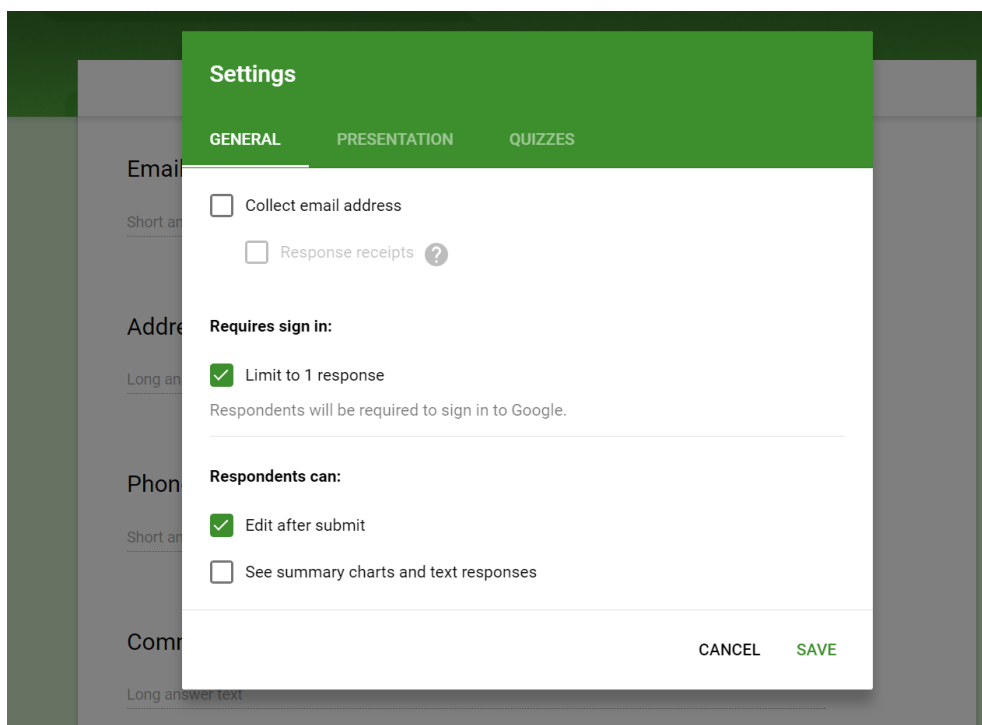


Рисунок 1.3 – Окно глобальных настроек формы

Важным преимуществом данного сервиса перед остальными является широкая интеграция с другими сервисами от компании Google. Тем самым многие задачи, которые необходимо решить в процессе проведения онлайн исследования, значительно упрощаются. Ярким примером такой интеграции может служить возможность синхронизации ответов, полученных в процессе исследования с таблицей на Google Docs.

Помимо этого, администратор имеет возможность добавлять на формы контент из своих каналов на медиасервисе YouTube, а также разного рода файлы, хранящиеся на Google Drive.

Еще одним несомненным достоинством данного сервиса является то, что доступ к нему предоставляется без необходимости внесения платы за услуги. При этом стоит также отметить, что пользователь получает весь функционал, без каких-либо ограничений.

Для разрешения вопросов, которые могут возникнуть в процессе администрирования опросника можно воспользоваться исчерпывающим руководством пользователя. Однако стоит отметить, что интерфейс администраторской части спроектирован таким образом, что большинство средств управления находятся в том месте, где их ожидает найти пользователь.

Несмотря на то, что данный сервис обладает огромным количеством достоинств стоит отметить и ряд недостатков:

- относительно низкие возможности задания действий на выбранные пользователем варианты ответов;
- отсутствие возможности использовать введенные пользователем данные для отображения дальнейших вопросов на форме;
- Google forms не предоставляют возможности для анализа результатов заполнения формы;
- отсутствие возможности интеграции со сторонними системами.

В некоторых случаях стоит учесть еще одну особенность данного сервиса: администратор формы не может выбирать местоположение серверов, где будут храниться результаты заполнения опросника. Данный аспект стоит учитывать при проведении онлайн-исследований, так как в некоторых странах, необходимость хранения пользовательских данных строго на территории государства прописана на законодательном уровне. Исходя из этого, в некоторых случаях данный аспект может перевесить все преимущества данного сервиса.

### 1.3.2 Survey Monkey

Еще одним решением, существующим в данной предметной области, является Survey Monkey. Данный сервис ориентирован на различные бизнес-приложения онлайн-исследований. Его использование возможно при проведении исследования мнения клиентов, маркетинговых исследованиях, планировании мероприятий, а также разного рода исследованиях в области рынка труда.

На данный момент данный сервис насчитывает порядка 30 миллионов активных пользователей, среди которых такие компании как Facebook, Samsung, SalesForce.

Самое первое отличие, которое бросается в глаза после рассмотрения Google Forms является то, что данный сервис предполагает внесение платы за его использование. Хотя существует и бесплатная подписка, ее возможности сильно урезаны, что делает ее использование практически невозможным для проведения серьезных онлайн исследований.

Второе отличие, которое становится очевидным при создании самого первого опросника – чрезвычайно перегруженный пользовательский интерфейс. Сервис предоставляет порядка 15 различных типов вопросов, среди которых и привычные: однострочный текст, многострочный текст, дата, время и т.д. – так и не так часто встречающиеся разновидности: рэнкинг, контактная информация, скоринг вопросы.

Несмотря на то, что интерфейс приложения содержит области с возможностью сворачивания, однако это не помогает скрыть все неиспользуемые элементы интерфейса и сконцентрироваться на основной рабочей области приложения. На рисунке 1.4 представлено главное окно конструктора новой формы.

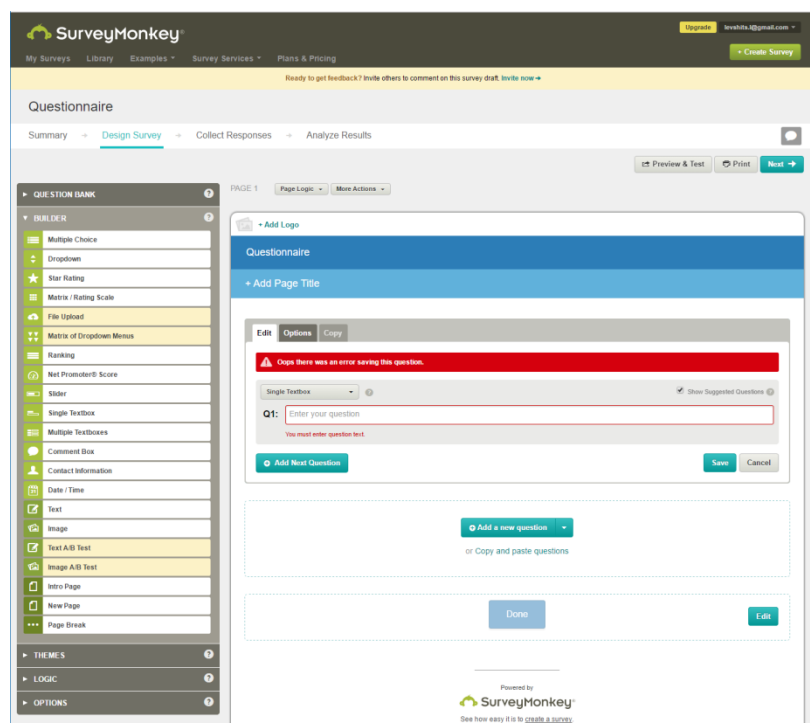


Рисунок 1.4 – Окно создания новой формы на Survey Monkey

Возможности создать формы «с нуля» есть также возможности создать дубликат уже существующей формы или же создать новую форму на основе одного из множества предлагаемых шаблонов. Предлагаемые шаблоны разбиты на группы по предметным областям. Данный подход позволяет сэкономить время, необходимое на подготовку опросника. Кроме возможности выбора одного из существующих шаблонов, администратор формы может добавлять на форму отдельные вопросы из существующей библиотеки вопросов. Пример окна выбора вопроса из библиотеки размещен ниже, на рисунке 1.5.



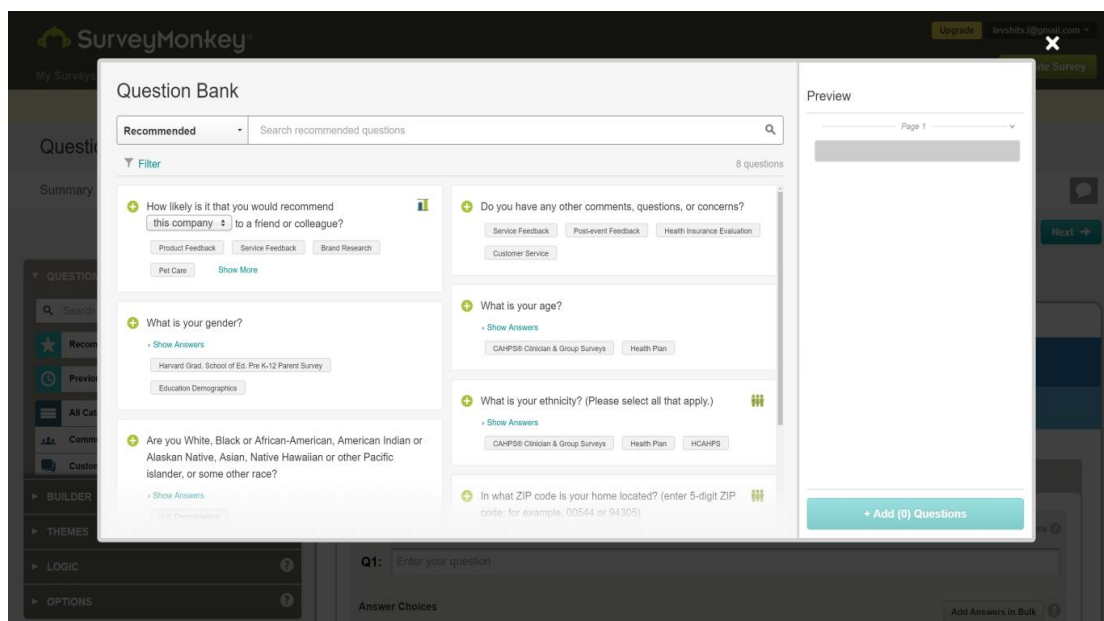


Рисунок 1.5 – Добавление нового вопроса из библиотеки в Survey Monkey

Данное приложение, как и Google Forms, реализует клиент-серверную архитектуру. Клиентская часть приложения реализована в виде SPA, что позволяет повысить отзывчивость интерфейса на действия пользователя. Такая реализация позволяет избегать ненужных перезагрузок страницы, а также снижает количество времени, затрачиваемого на ожидание при работе с медленным интернет соединением.

Survey Monkey позволяют осуществить экспорт результатов опроса в одном из следующих форматов: .xls, .pdf, .csv, .pptx, .spss. Однако для того, чтобы воспользоваться данной функциональностью, придется купить подписку на данный сервис.

Данное приложение ориентировано в первую очередь на англоязычную аудиторию. Несмотря на то, что сервисом заявлена поддержка русского языка, в процессе использования приходится мириться с тем, что локализованы далеко не все формы. Так, например, типы вопросов, а также некоторые страницы руководства пользователя не имеют версий на русском языке. Верстка отдельных страниц также не адаптирована под русскоязычную версию.

Для каждой формы можно просмотреть статистику активности ее использования. На рисунке 1.6 изображена главная страница опросника, на которой администратор может получить наиболее общие сведения о статусе формы, а также активности респондентов.

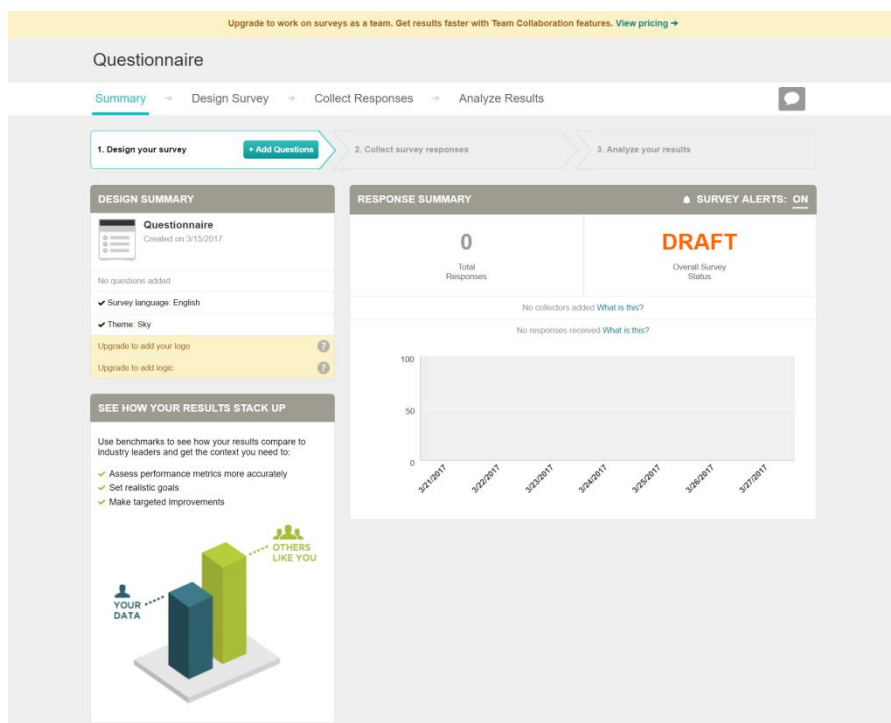


Рисунок 1.6 – Окно статистики опросника в Survey Monkey

Резюмируя вышесказанное, стоит отметить, что в процессе использования не возникает претензий к функционированию приложения, однако можно найти небольшие огрехи, касающиеся скорее возможных нефункциональных требований к подобного рода сервисам.

### 1.3.3 Survio

Очередным сервисом, существующим в области проведения онлайн исследований, является Survio. Данный сервис предоставляет как платную, так и бесплатную версию. На бесплатной версии можно создать 5 анкет с неограниченным количеством вопросов и собрать максимум 100 ответов в месяц. Для того чтобы воспользоваться дополнительными возможностями придется приобрести платную версию. Минимальный тариф (29 \$/мес. при разовой покупке либо 14 \$/мес. за годовой тариф на момент написания работы) позволяет создать 100 опросов и получить 1 000 ответов, при этом ограничения на количество возможных вопросов в анкете отсутствуют.

Сервис оптимизирован для работы на мобильных устройствах. По способу взаимодействия с пользователем данное приложение имеет сходства с Google Forms: интерфейс минималистичен, в каждый момент времени отображаются только активные средства управления. Такой способ взаимодействия с пользователем позволяет сконцентрировать внимание на выполняемых им действиях.

Чуть ниже, на рисунке 1.7 можно увидеть реализацию списковой формы всех опросников, созданных пользователем.

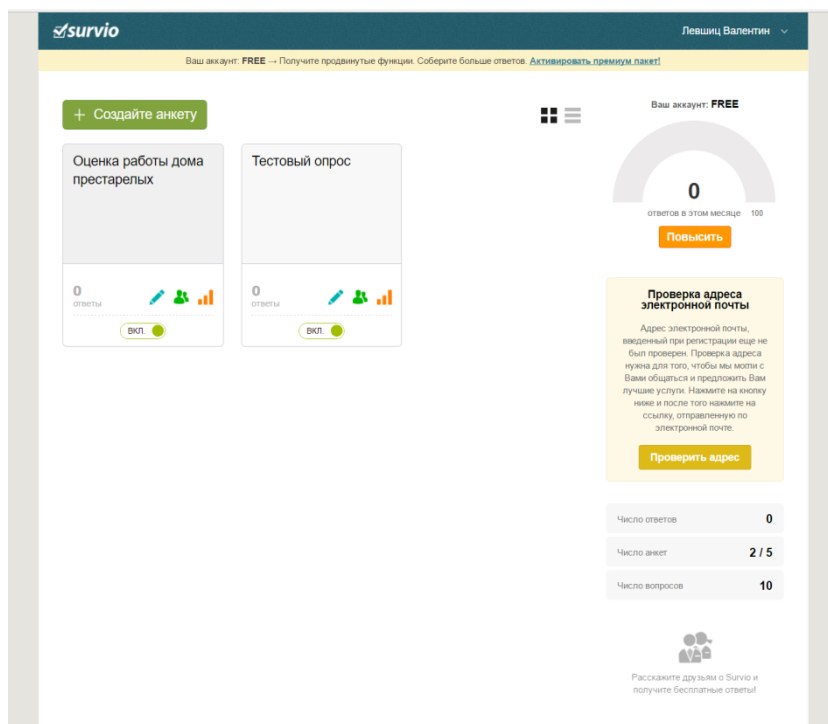


Рисунок 1.7 – Главная страница приложения Survio со списком созданных опросников.

Возможности настройки отображения достаточно бедны. Сервис предлагает только добавление логотипа, смену цветовой схемы, или же использование готовых фонов.

Survey предлагает ряд настроек заполнения для каждой анкеты. Администратор может выбрать период времени, в течение которого будут приниматься ответы на анкету; разрешить или запретить повторное заполнение; ограничить регион ip-адресов, с которых возможно заполнение. Возможна реализация парольного доступа к анкете, таким образом, ее смогут заполнить только те резиденты, обладающие ключом.

Также, при создании анкеты, администратор может указать адрес, на который будет перенаправлен пользователь сразу после заполнения или же ввести текст, который будет отображен пользователю.

Как и ранее рассмотренные сервисы, Survey предоставляет возможности создания анкеты на базе шаблона, однако их количество достаточно мало. Также пользователю предлагается создать копию ранее созданной пользователем фор-

мы. Кроме этого стоит отметить отсутствие возможности для повторного использования отдельных вопросов.

В процессе создания анкеты сервис предлагает задать правила переходов между страницами, в зависимости от ответов пользователя на отдельные вопросы. К сожалению, данная функциональность имеет целый ряд ограничений: от ограничения количества таких вопросов на страницу (не более одного), до ограничений, накладываемых на возможный тип вопроса (данное правило может быть применено только к типу вопроса «одиночный выбор»).

Существует несколько способов распространения анкеты. Помимо стандартного распространения опроса по прямой ссылке, есть еще встраивание опроса на сайт и всплывающее окно, однако обе этих возможности не имеют дополнительных настроек. Отсутствует возможность синхронизации данных заполнения анкеты со сторонними системами. Окно настроек распространения формы изображено на рисунке 1.8.

The screenshot shows the 'Survey Settings' interface. On the left is a sidebar with three items: 'Составление опроса' (Survey Creation), 'Сбор ответов' (Collecting Answers), and 'Анализ результатов' (Analyzing Results). The main area is titled 'Как вы предпочитаете собирать ответы?' (How do you prefer to collect answers?). Below this is a section for the 'URL адрес анкеты' (Survey URL), showing a red URL: <https://www.survio.com/survey/d/A5L5Q7U3U7G7R1T7R>, with an 'Изменить' (Change) button. A 'Поделиться' (Share) icon is also present. Below the URL section are three distribution methods: 1. 'E-mail приглашение' (Email Invitation) with a description, an 'Отправить' (Send) button, and an icon of an envelope. 2. 'Панель респондентов' (Respondent Panel) with a description, a 'Купить' (Buy) button, and an icon of a group of people. 3. 'Сайт' (Website) with three options: 'Веб-ссылка' (Web link), 'Опрос на сайте' (Survey on site), and 'Pop-up окно' (Pop-up window), each with a corresponding icon.

Рисунок 1.8 – Настройка способов распространения анкеты

Присутствует возможность экспорта данных в .xls и .pdf. Однако для того, чтобы воспользоваться данной возможностью придется купить платную подписку. В бесплатной версии приложения возможен только просмотр результатов на сайте приложения.

Данные результатов заполнения анкет хранятся на сервере в Чехии. Так же, как и в остальных случаях, у пользователя нет возможности выбрать месторасположения сервера, что в некоторых случаях способно сделать использование данного продукта невозможным.

В остальном данное программное средство схоже с ранее рассмотренными. Список типов вопросов характерен для программных решений в данной области. На рисунке 1.9 изображено окно выбора типа при создании нового вопроса.

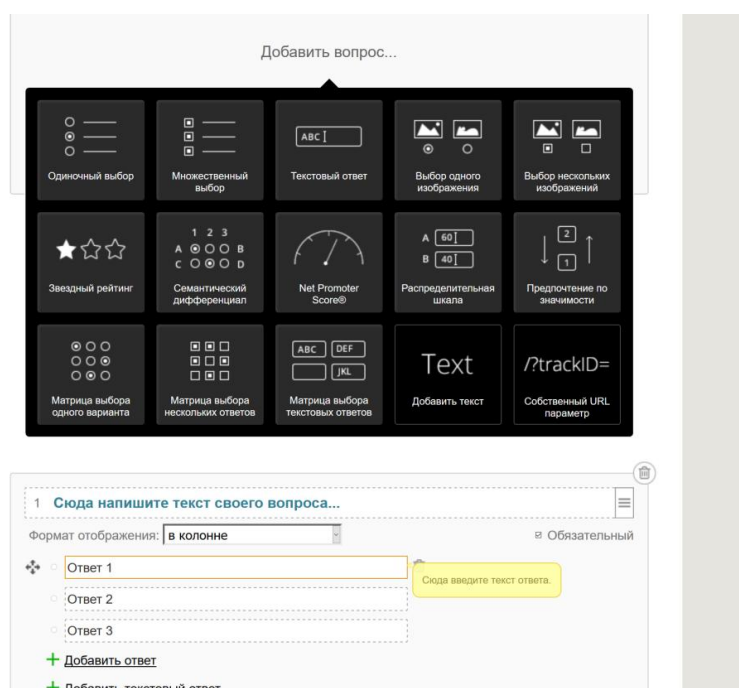


Рисунок 1.9 – Форма выбора типа вопроса, предлагаемая Survio

Подводя итог, к преимуществам данного сервиса стоит отнести:

- поддержка мобильных устройств;
- возможность пропуска вопросов в зависимости от ответов пользователя;
- интеграция со сторонними системами;
- реализация парольного доступа к анкете;
- экспорт результатов заполнения.

Однако не обошлось и без недостатков:

- бедные возможности настройки отображения;
- отсутствие коллективного администрирования;
- отсутствие синхронизации данных со сторонними системами;
- отсутствие возможности использования введенных ответов при формировании текста дальнейших вопросов.

## 1.4 Постановка целей и задач дипломного проектирования

Как показал анализ программной области, основными недостатками существующих программных решений являются следующие:

- отсутствие поддержки работы на мобильных устройствах;
- отсутствие возможности интеграции со сторонними системами;
- низкие возможности анализа результатов заполнения;
- высокая стоимость;
- отсутствие возможности многопользовательского администрирования формы;
- отсутствие возможности выбрать расположение серверов, на которых будет осуществляться хранение результатов заполнения.

Исходя из этого, целью дипломного проектирования является разработка программного средства, способного устранить вышеперечисленные недостатки, а также реализовать необходимый набор функций, характерный для программных средств в данной предметной области.

Для достижения поставленных целей следует решить следующие задачи:

- определить требования к разрабатываемому программному средству и составление спецификации, включающей их;
- осуществить выбор технологии и языка программирования для реализации программного средства;
- провести проектирование архитектуры программного средства;
- разработка пользовательских интерфейсов;
- разработка моделей данных;
- разработка алгоритмов задания правил переходов между страницами и отображения отдельных вопросов;
- программирование и тестирование отдельных программных модулей;
- тестирование готового программного средства.

## **2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ И СОСТАВЛЕНИЕ ИХ СПЕЦИФИКАЦИИ**

### **2.1 Общие сведения и требования к работе программного средства**

Функциональным назначением разрабатываемого программного решения является предоставление пользователю возможностей для проведения онлайн-исследований.

В качестве пользователя программного средства может выступать любой человек, который имеет доступ к персональному компьютеру с доступом к сети интернет. Для использования программного решения не требуется специальная подготовка или обучение пользователей.

Предполагается возможность одновременной эксплуатации разрабатываемого решения широким кругом пользователей. При этом отсутствуют какие-либо ограничения, накладываемые на предметную область, в рамках которой возможно его применение.

Исходя из предполагаемого использования, можно заключить, что проектируемое программное решение должно реализовывать следующие три группы функций:

- управление формами;
- управление результатами заполнения;
- управление пользователями.

### **2.2 Описание функциональности программного средства**

В качестве способа представления функциональности программного средства будем использовать диаграмму вариантов использования. Данный тип UML диаграмм позволяет описать функциональность системы на концептуальном уровне через взаимосвязи между прецедентами и актерами.

Каждый прецедент Use-Case диаграммы отображает один из вариантов использования программного средства конкретным пользователем. Средства UML позволяют установить отношения обобщения для актеров на диаграмме, таким образом отсутствует необходимость в дублировании одинаковых прецедентов на диаграмме использования.

На рисунке 2.1 представлена укрупненная диаграмма вариантов использования разрабатываемого программного средства, на которой отражены обобщенные группы функций, доступные для действующих лиц внутри системы.

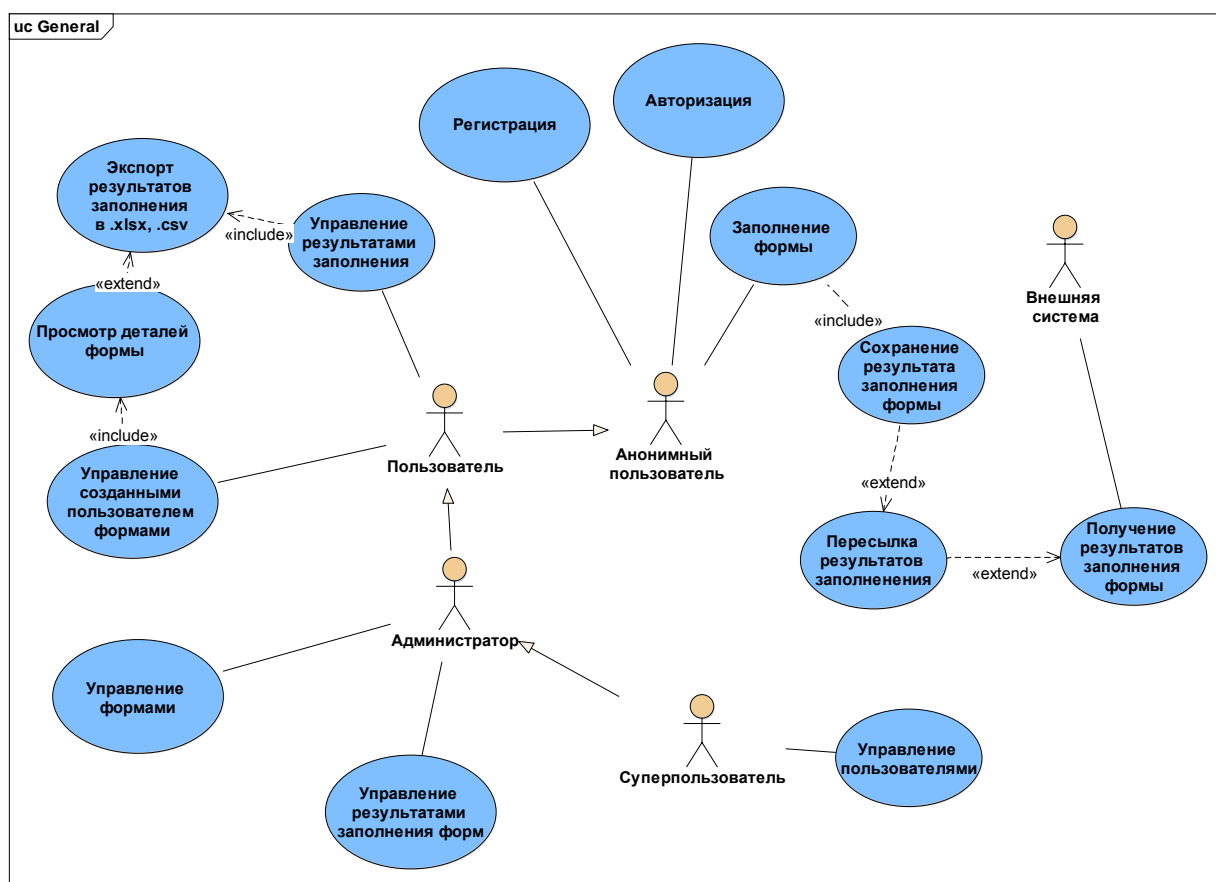


Рисунок 2.1 – Обобщенная диаграмма вариантов использования

В процессе анализа литературы по теме дипломного проектирования, а также в течение обзора существующих программных решений в данной предметной области был определён список действующих лиц и ролей, которые будут взаимодействовать с разрабатываемой системой, и для которых, в свою очередь, данная система должна будет предоставлять ряд функций. Среди таких действующих лиц мы можем выделить следующие:

- анонимный пользователь;
- пользователь;
- администратор;
- суперпользователь;
- внешняя система.

Рассмотрим всех действующих лиц проектируемого решения в порядке их следования. Анонимный пользователь – базовая роль для остальных актеров на диаграмме прецедентов, к данному типу мы относим всех пользователей, о которых разрабатываемое решение не имеет никакой информации, которая бы позволила идентифицировать в них зарегистрированного пользователя. Однако



стоит отметить, что данный тип действующих лиц является одним из основных в рамках программного средства. Многие из пользователей, которые используют системы подобного типа, не имеют персонального аккаунта.

На рисунке 2.2 отображены функциональные возможности анонимного пользователя.

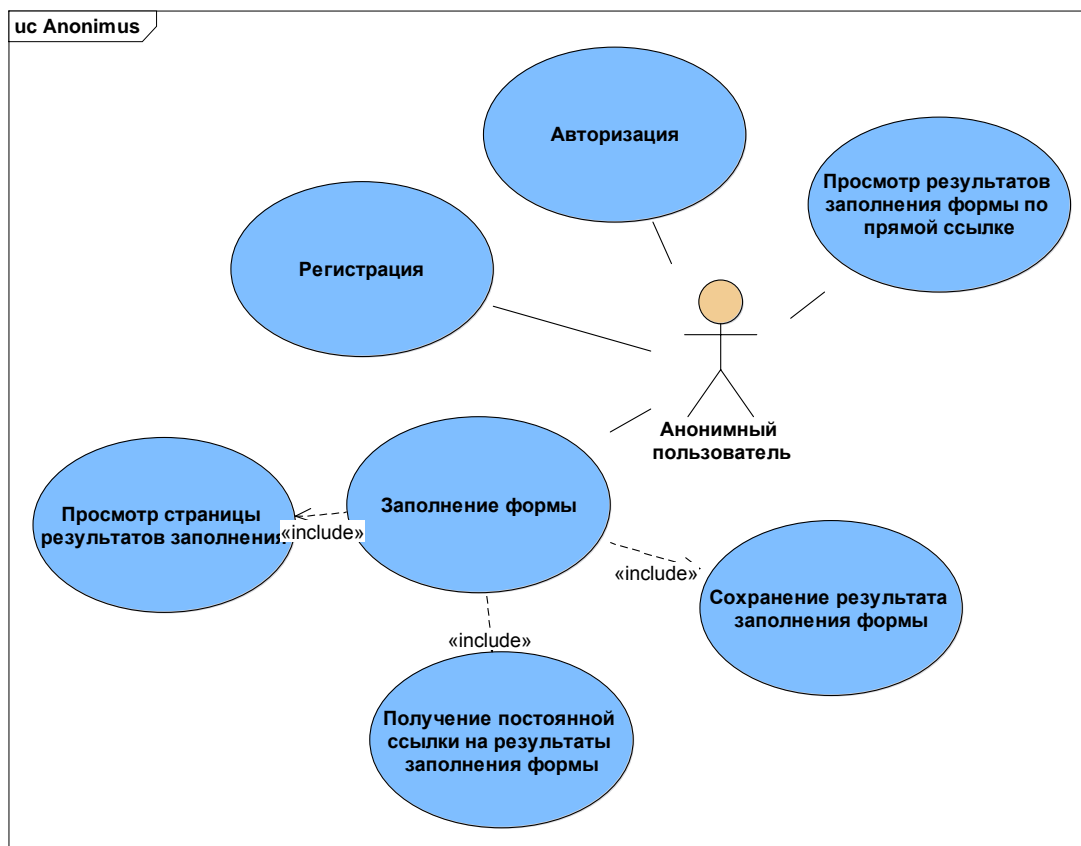


Рисунок 2.2 – Функциональные возможности Анонимного пользователя

Во многом наличии отдельного пользовательского профайла в такой системе не имеет смысла. Для этого есть целый ряд причин: подобного рода системы предоставляют возможность создания анонимных опросов, где пользователю нет необходимости идентифицировать себя, с другой стороны многие опросники не представляют какую-либо ценность для пользователя, который заполнил анкету.

Онлайн-исследование может являться одним из этапов более глобального исследования, и, как следствие, не могут быть предоставлены осмысленные результаты по данным предоставленным в течение анкетирования. С другой стороны, существуют предметные области, в которых, для предоставления результатов необходима работа специалиста в рамках данной предметной области, так

как на данном этапе развития техники отсутствует возможность для автоматического вычисления результатов. В этом случае онлайн-исследование используется только для сбора первичной информации.

Данному типу пользователей предоставляются следующие функции в рамках системы:

- регистрация;
- авторизация;
- заполнение формы;
- просмотр результатов заполненной формы по прямой ссылке.

Регистрация пользователя в системе подразумевает создание персонального аккаунта, а также предоставление пользователям прав роли типа Пользователь. В процессе регистрации пользователю необходимо ввести персональные данные среди которых: адрес электронной почты, пароль, подтверждение пароля, фамилия и имя. Впоследствии адрес электронной почты и персональный пароль будут использоваться при осуществлении идентификации пользователей системы.

Под авторизацией в системе будем понимать идентификацию пользователя по предоставленному адресу электронной почты и персональному паролю зарегистрированного в системе пользователя и дальнейшее предоставление ему прав доступа в соответствии с занимаемой в системе ролью.

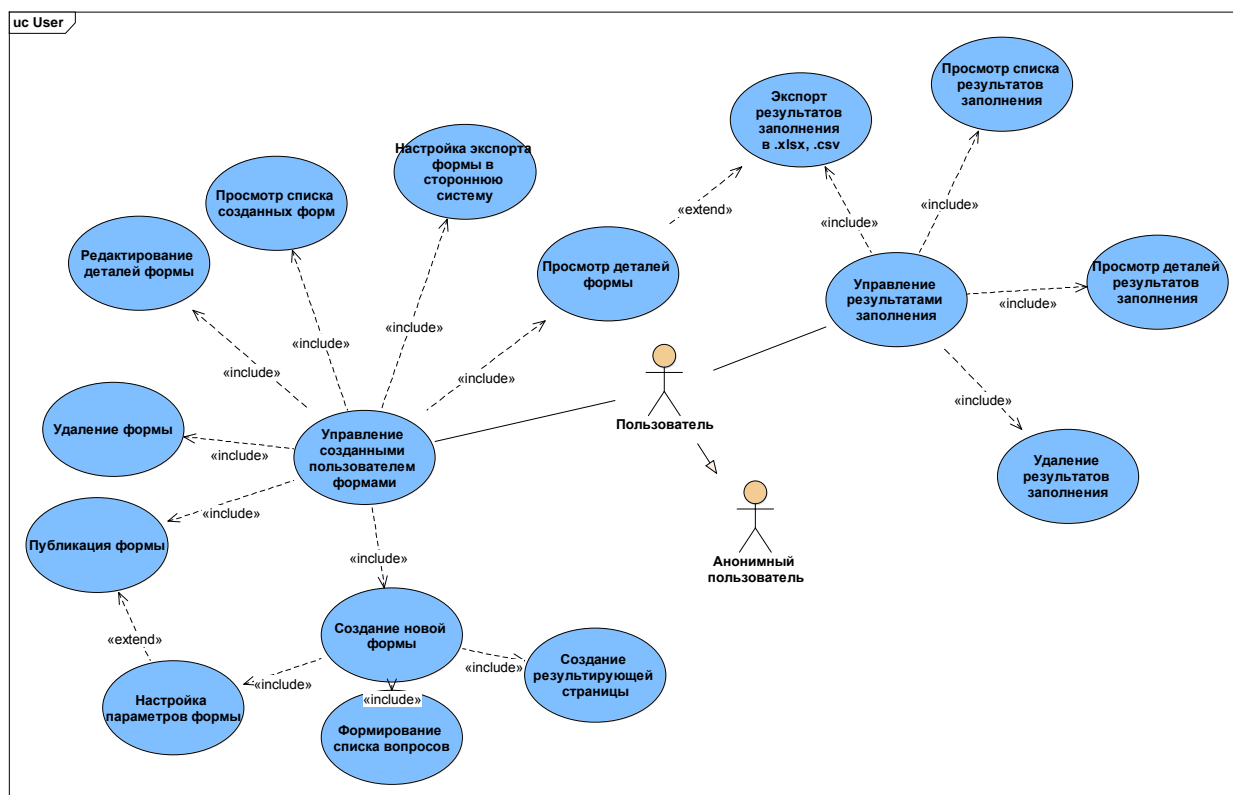
Основной функцией, предоставляемой программным решением является возможность заполнения формы опросника. Данная функция подразумевает открытие существующей формы на заполнение, осуществление логики переходов между страницами в процессе работы пользователя с программным средством, формирование результирующей страницы, которая основывается на данных приведенных в процессе работы с сервисом, сохранение введенных пользователем данных по окончании заполнения опросника, а также генерация ссылки, по которой будет осуществляться прямой доступ на результаты анкетирования.

В процессе заполнения опросника должны осуществляться все необходимые проверки введенных значений, осуществляться отображение элементов управления, а также следует обеспечить корректное функционирование логики переходов между отдельными страницами опросника и правил отображения вопросов, заданных администратором данной формы.

В случае, когда настройки формы подразумевают необходимость отправки результатов заполнения в стороннюю систему, после сохранения введенных значений необходимо осуществить пересылку результатов заполнения в формате JSON на указанный создателем формы адрес.

Следующим видом актера, изображенным на диаграмме прецедентов, является Пользователь. Данный тип роли наследует все функциональные возможности анонимного пользователя. Кроме уже рассмотренных функций, пользовате-

лю также предоставляется возможность управлять созданными им формами опросников, а также результатами их заполнения.



Все вышеперечисленные функции можно разделить на две группы:

Рассмотрим более подробно некоторые из функциональных возможностей актеров данного типа.

Разрабатываемая система должна позволять создавать вопросы следующих типов:

- многострочный текст;
- целое число;
- дробное значение;
- дата;
- время.

При формировании списка вопросов должна присутствовать возможность разбить весь список вопросов на отдельные страницы. При этом каждая страница должна иметь настраиваемый заголовок и название.

Для закрытых вопросов, то есть вопросов с заранее определенным списком возможных ответов, должна предоставляться возможность введения стоимости каждого ответа. Перед отображением результирующей страницы стоимости, выбранный пользователем, вариантов ответов суммируются, образуя значение скоринга.

В процессе создания результирующих страниц пользователь должен иметь возможность выбрать диапазон значений скоринга, при которых будет отображаться текущая результирующая страница.

Созданная форма должна иметь уникальный адрес внутри системы, переход на который приводил бы к инициализации заполнения формы. Помимо этого, для интеграции созданных форм в сторонние системы необходимо предусмотреть возможность встраивания модуля заполнения формы в веб-страницы.

Все вышеперечисленные функции могут быть применены для форм, созданных текущим пользователем, а также результатам заполнения форм текущего пользователя.

Функция настройки параметров формы подразумевает, конфигурирование параметров экспорта результатов заполнения формы в сторонние системы, а также изменение статуса публикации выбранной формы. Настройки экспорта результатов заполнения должны включать URL адрес сервиса, на который должна быть осуществлена отправка результатов заполнения.

Все формы, не имеющие результатов заполнения, могут быть удалены пользователем, который их создал. В случае, если выбранная форма уже содержит результаты заполнения, ее удаление возможно лишь после удаления всех результатов заполнения.

Очередным типом роли является – Администратор. Данный тип роли введен в систему для обеспечения поддержки работы остальных пользователей. Отличительной особенностью данной роли является возможность видеть формы всех пользователей.

Как и в предыдущем случае, роль типа Администратор наследует все возможности ранее рассмотренных типов. С точки зрения реального применения программного средства данный тип роли может быть предоставлен сотрудникам службы поддержки сервиса. Возможности пользователей этого типа призваны

помогать в устранении возникающих в процессе использования вопросов, а также незаменимы при проведении тестирования функциональности.

В остальном данный тип роли мало отличается от роли Пользователь. Однако функционирование программного средства без пользователей данного типа не представляется возможным. Полный список возможностей, внесенных для пользователей данного типа, изображен на рисунке 2.4.

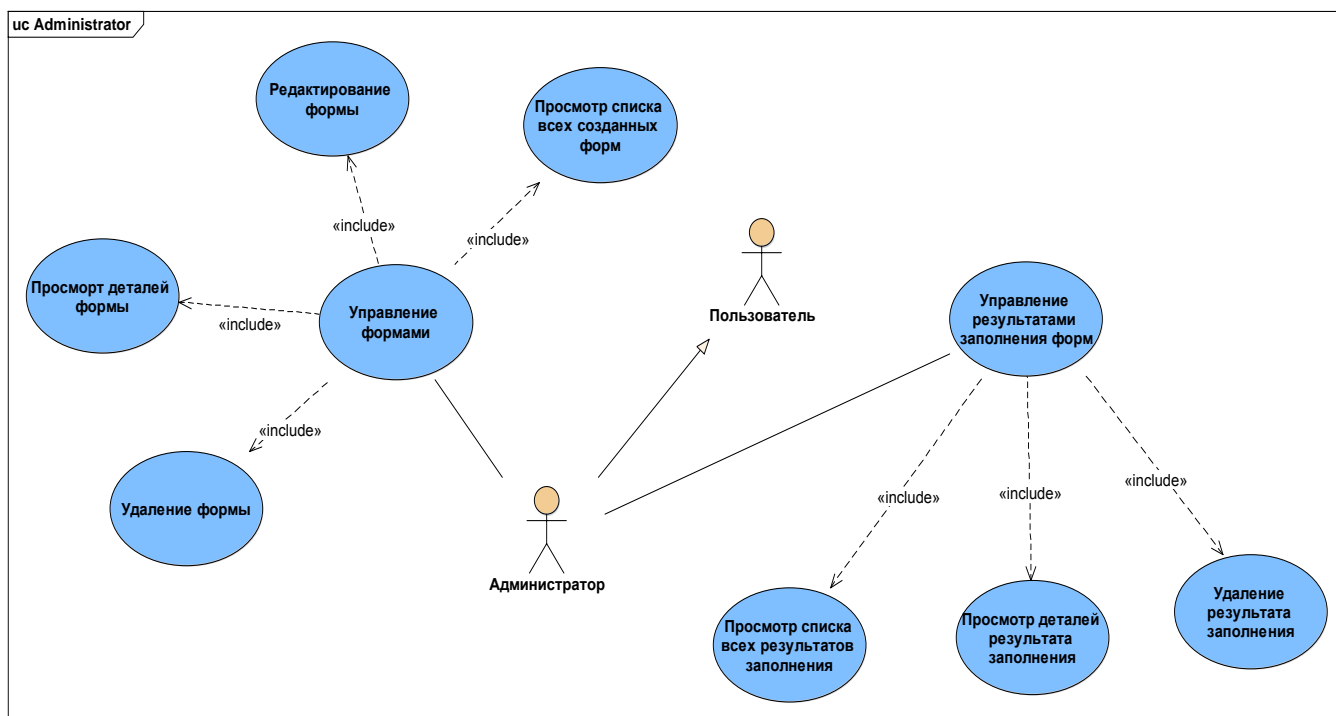


Рисунок 2.4 – Функциональные возможности администратора.

Последним типом пользовательских ролей, представленных на диаграмме прецедентов, является суперпользователь. Суперпользователь – тип роли, предназначенный для администраторов системы. Пользователи с данной ролью должны обладать максимальным набором функций из списка возможных. Суперпользователь включает в себя возможности остальных типов ролей.

Основное предназначение данной роли – администрирование системы в целом и контроль над всеми процессами, происходящими внутри системы. Исходя из этого, пользователи данного типа агрегируют возможности остальных ролей.

При развертывании программного средства в системе должен существовать пользователь с ролью данного типа.

Отличительный набор функциональных возможностей пользователей данного типа отображен на рисунке 2.5.

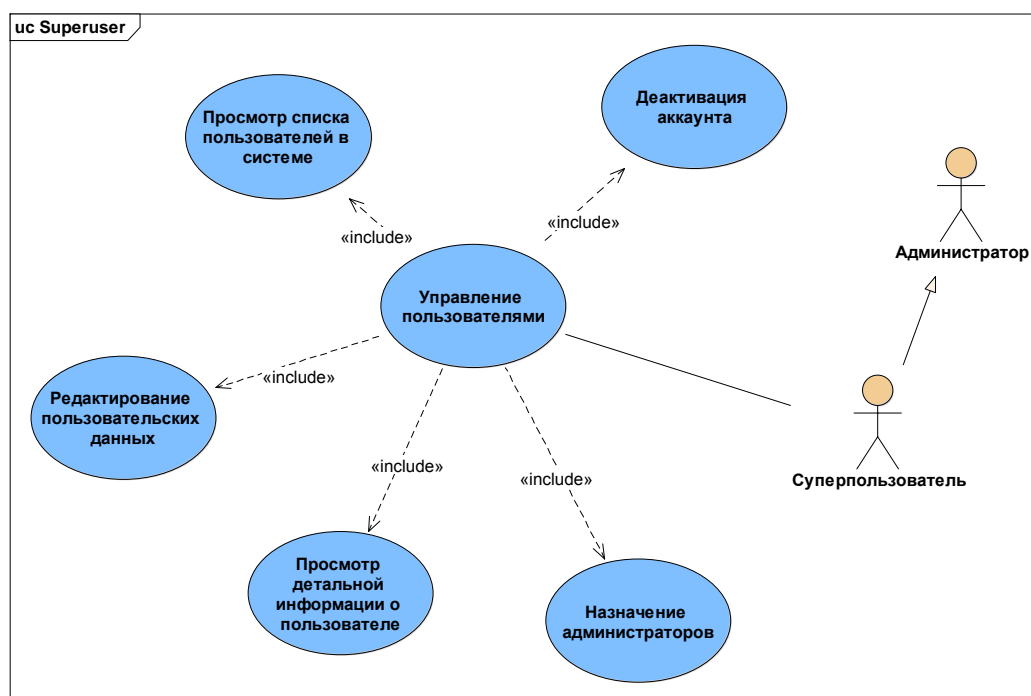


Рисунок 2.5 – Функциональные возможности суперпользователя.

Для обеспечения работы сервиса важно знать количество существующих пользователей, иметь возможность просмотреть список пользователей. Но вместе с тем, стоит учитывать, что доступ к персональным данным пользователей должен быть у строго ограниченного числа лиц. В рамках системы только пользователи с ролью типа Суперпользователь имеют доступ к персональным данным пользователя.

Несколько ранее в этой главе был описан тип роли «Администратор», однако не был рассмотрен способ создания пользователей этого типа. Суперпользователь должен иметь возможность назначения администраторов из списка существующих пользователей.

С течением времени в подобного рода сервисах скапливается достаточное количество пользователей, которые не осуществляют работу с системой, или же которые по разным причинам утратили доступ к своему аккаунту. Для повышения производительности системы и очистки неактуальных данных, суперпользователь может воспользоваться деактивацией старых аккаунтов. При деактивации аккаунта происходит удаление пользователя из системы, а также удаление всех форм и связанных с ними.

Одновременное наличие в программном решении ролей типов Администратор и Суперпользователь, позволяет разделять зоны ответственности и снизить нагрузку, связанную с обеспечением поддержки пользователей, которая изначально возлагается на суперпользователя.

## 2.3 Разработка информационной модели

На основании функциональной модели была разработана информационная модель программного решения. На рисунке 2.6 представлена информационная модель проектируемой системы [8].

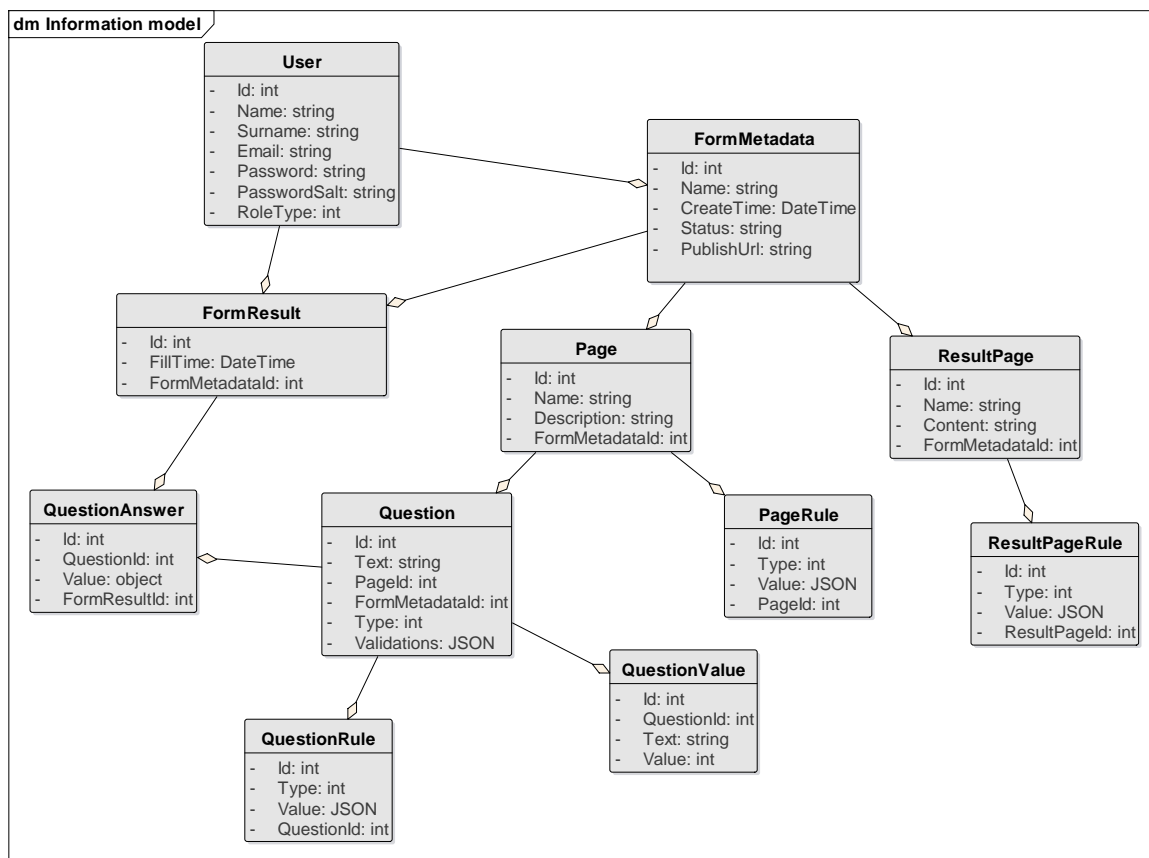


Рисунок 2.6 – Информационная модель проектируемой системы

В процессе анализа предметной области были выделены следующие типы сущностей:

- User – сущность хранящая представляющая собой все типы пользователей в рамках системы. Атрибут Id – идентификатор пользователя, Name – имя пользователя, Surname – фамилия пользователя, Email – электронный адрес пользователя, Password – хэш пароля пользователя, PasswordSalt – случайное значение, использованное при хешировании пароля, RoleType – тип роли пользователя.

- FormMetadata – главная сущность, содержащая общие настройки для всей формы и статус ее публикации. Сущность обладает следующими атрибутами: Id – идентификатор формы, Name – имя формы, CreateTime – время создания фор-

мы, Status – статус публикации формы, PublishUrl – адрес, на который следует отослать результаты заполнения формы.

- Page – каждая форма может содержать целый ряд страниц, на которых будут содержаться вопросы. Атрибут Id – идентификатор страницы, Name – имя страницы, Description – детальное описание страницы, FormMetadataId – идентификатор формы к которой принадлежит текущая страница.

- PageRule – правило отображение страницы. Атрибут Id – идентификатор правила отображения, Type – тип правила, Value – описание критерия правила, PageId – идентификатор страницы к которой относится правило отображения.

- Question – сущность, описывающая вопрос формы опросника в системе. Атрибут Id – идентификатор вопроса, Text – текст вопроса, PageId – идентификатор страницы, на которой располагается вопрос, Type – тип вопроса, Validations – список правил проверки для значений ответов на вопрос.

- QuestionValue – закрытые вопросы в системе могут содержать несколько вариантов ответа. Атрибут Id – идентификатор вопроса, QuestionId – идентификатор вопроса, Text – значение варианта ответа, Value – заданная стоимость ответа.

- QuestionRule – для каждого вопрос можно задать несколько правил, по которым будет определяться необходимость отображать вопрос. Атрибут Id – идентификатор правила отображения, Type – тип правила, Value – описание критерия правила, QuestionId – идентификатор вопроса к которому относится правило отображения.

- ResultPage – после заполнения формы, пользователю должна быть отображена страница с результатом. Атрибут Id – идентификатор результирующей страницы, Name – имя результирующей страницы, Content – ее содержимое, FormMetadataId – идентификатор формы к которой относится текущая страница.

- ResultPageRule – пользователь может задать правило по которому будет выбрана результирующая страница. Атрибут Id – идентификатор правила отображения, Type – тип правила, Value – описание критерия правила, ResultPageId – идентификатор результирующей страницы к которой относится правило отображения.

- FormResult – сущность, представляющая собой общие сведения о результатах опроса. Атрибут Id – идентификатор результирующей страницы, FillTime – время сохранения результата, FormMetadataId – идентификатор формы, к которой относится данный результат.

- QuestionAnswer – каждый результат заполнения содержит в себе список ответов пользователя на вопросы формы. Атрибут Id – идентификатор ответа на вопрос, Value – значение ответа, FormResultId – идентификатор результата заполнения, к которому принадлежит данный ответы.



## 2.4 Разработка модели взаимодействия пользователя с интерфейсом

В процессе анализа программных решений в области дипломного проектирования, отмечалась необходимость обеспечения высокого уровня интерактивности при взаимодействии с пользователем. На рисунке 2.7 изображена модель взаимодействия с пользователем, базирующаяся на реактивном подходе к разработке пользовательских интерфейсов.

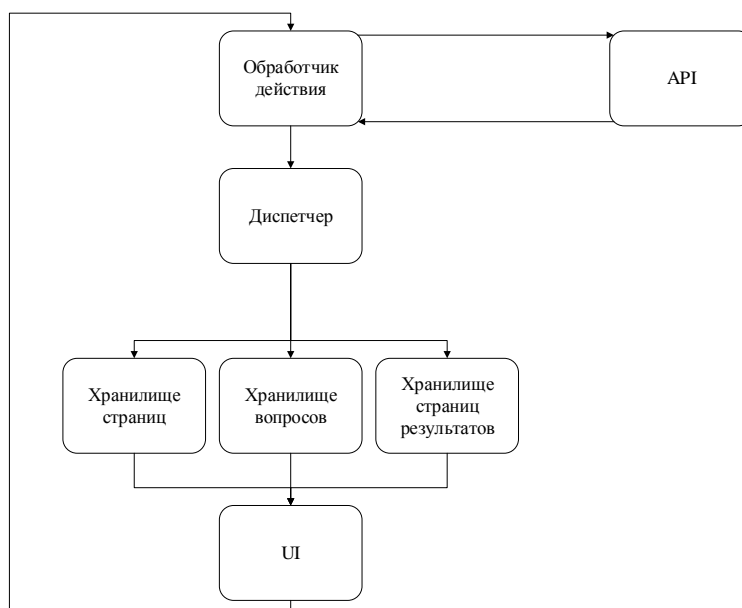


Рисунок 2.7 – Модель взаимодействия пользователя с интерфейсом.

На данной диаграмме отражены классические элементы метода Flux. Flux – это архитектура, которую команда Facebook использует при работе с React. Это не фреймворк, или библиотека, это новый архитектурный подход, который дополняет React и принцип однонаправленного потока данных.

На данной модели отражены следующие компоненты:

- действия – вспомогательные методы и классы, упрощающие передачу данных Диспетчеру и осуществляющие взаимодействие с серверной частью приложения;
- диспетчер – принимает объекты действия и рассылает нагрузку зарегистрированным обработчикам;
- хранилища – контейнеры для состояния приложения и бизнес-логики в обработчиках, зарегистрированных в диспетчере;

– представления – компоненты клиентских модулей, которые собирают состояние хранилищ и передают его дочерним компонентам через свойства и реализуют их дальнейшее отображение.

Модуль диспетчера, изображенный на модели является стандартным и имеет ряд готовых реализаций для различных языков программирования. Исходя из этого, следование данному подходу увеличивает скорость реализации отдельных модулей.

Наличие однонаправленного потока данных также положительно сказывается на общей реализации логики взаимодействия. Однонаправленный проход упрощает дальнейшее проектирование программной архитектуры, при этом не снижая ее производительность.

Использование данного подхода и реализация клиентских модулей в соответствии с данной моделью позволяет достичь улучшить отзывчивость пользовательского интерфейса, а также повысить интерактивность средств управления.

## **2.5 Разработка спецификации функциональных требований**

Исходя из результатов анализа исходных данных для проектируемого программного средства, можно выделить следующее: основной целью работы является создание программного продукта, который позволил бы разрешить существующие несовершенства программных средств в области, рассматриваемой в процессе дипломного проектирования [7]. Среди недостатков программных решений стоит отметить следующие:

- отсутствие поддержки работы на мобильных устройствах;
- отсутствие возможности интеграции со сторонними системами;
- высокая стоимость;
- отсутствие возможности выбрать расположение серверов, на которых будет осуществляться хранение результатов заполнения.

В ходе проектирования и разработки необходимо продумать и реализовать следующие функциональные возможности:

- функциональность управления формами;
- функциональность заполнения форм опросников;
- функциональность управления результатами заполнения;
- функциональность экспорта результатов заполнения в популярные форматы работы с данными;
- функциональность по пересылке результатов заполнения в сторонние системы;
- функциональность по управлению пользователями;
- функциональность по настройке и администрированию системы;

- возможность идентификации и аутентификации субъектов доступа.

Разрабатываемое решение представляет собой программное средство, предназначенное для создания форм опросников и последующей обработки результатов их заполнения. Основными функциями данного программного средства являются следующие:

- функции идентификации и аутентификации;
- система должна поддерживать создание форм опросников;
- система должна позволять осуществлять заполнение форм опросников анонимным пользователям;
- система должна позволять экспорт результатов заполнения формы;
- возможность назначения администраторов системы.

Идентификация и аутентификация должны быть выполнены в соответствии со следующими требованиями:

- авторизация должна быть осуществлена по адресу электронной почты и персональному паролю;
- следует предусмотреть возможность аутентификации одного и того же пользователя с нескольких устройств.

Функциональность заполнения должна отвечать следующим требованиям:

- необходимо обеспечить поддержку браузеров, указанных в технических требованиях;
- необходимо обеспечить возможность повторного заполнения формы;
- необходимо обеспечить возможность сохранить результаты заполнения.

## **2.6 Разработка технических требований к программному средству**

Разрабатываемое программное решение должно обеспечивать корректное функционирование при развертывании программных модулей на сервере со следующими техническими характеристиками:

- Xeon 2.2 ГГц или более быстродействующий процессор;
- оперативная память 4 Гбайт или более;
- доступный объем дискового пространства 10 Гбайт.

Для нормального функционирования клиентской части программного комплекса должны выполняться следующие технические требования:

- Pentium 2 ГГц или более быстродействующий процессор;
- оперативная память 2 Гбайт или более;
- 32- или 64-битная версия операционных систем Windows 7, 8, 10;
- Браузер IE версии 11.x, Google Chrome версии 46.x, Opera версии 12.x, Mozilla Firefox 41.x

### **3 ПРОЕКТИРОВАНИЕ      АРХИТЕКТУРЫ      ПРОГРАММНОГО СРЕДСТВА**

#### **3.1 Разработка архитектуры программного средства**

После того, как были сформулированы функциональные требования к разрабатываемой системе, а также исходя из результатов анализа существующих программных решений, можно определить основные моменты организации системы, в которой будет функционировать разрабатываемое программное решение.

Процесс проектирования архитектуры программного обеспечения включает в себя сбор требований клиентов, их анализ и создание проекта для компонента программного обеспечения в соответствии с требованиями. Успешная разработка ПО должна обеспечивать баланс неизбежных компромиссов вследствие противоречащих требований; соответствовать принципам проектирования и рекомендованным методам, выработанным со временем; и дополнять современное оборудование, сети и системы управления.

Архитектуру программного обеспечения можно рассматривать как сопоставление между целью компонента ПО и сведениями о реализации в коде. Правильное понимание архитектуры обеспечит оптимальный баланс требований и результатов. Только программное обеспечение с хорошо продуманной архитектурой способно выполнять указанные задачи с параметрами исходных требований, одновременно обеспечивая максимально высокую производительность.

Программные средства в данной предметной области построены на базе клиент-серверной архитектуры. Использование других подходов приводит к сложностям связанным, со сбором результатов заполнения форм, а также синхронизацией моделей форм между отдельными клиентами внутри системы.

На рисунке 3.1 представлена диаграмма развертывания программного средства. На данной диаграмме нашел свое отражение клиент-серверный подход. Помимо блоков, отражающих рабочие устройства пользователей, на диаграмме представлен сервер, на котором развернуто приложение.

Сервер приложений представляет собой устройство, на котором развернут веб-сервер и серверная часть приложения. В рамках клиент-серверной архитектуры веб-сервер представляет собой посредника между логикой программного средства и внешним миром. Его основная задача: принимать запросы от пользователей и вызывать соответствующие методы обработки данных запросов.

Помимо прочего, среди функций, реализуемых веб-сервером, есть и возможность возврата статического контента. Статический контент, в свою очередь, включает: изображения, стили, клиентские модули приложения.

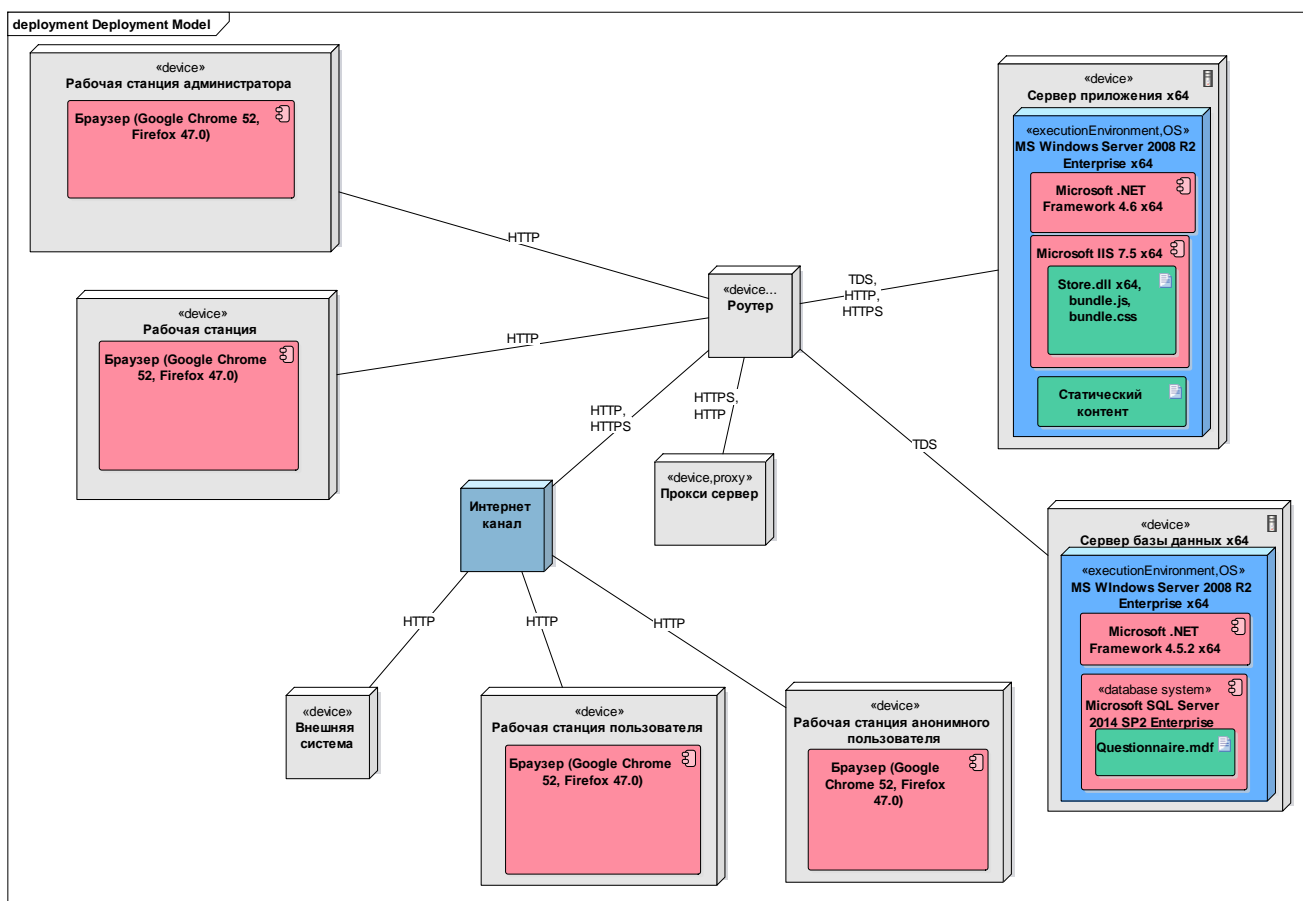


Рисунок 3.1 – Диаграмма развертывания

В процессе исследования программных решений в области проведения онлайн-исследований было отмечено факт того, что зачастую клиентская часть приложения реализуется в виде SPA (Single Page Application). Клиентский модуль проектируемого решения также должен быть выполнен в соответствии с этой парадигмой. Отражение этого факта на диаграмме заключается в наличии `bundle.js` файла на сервере приложения.

В качестве хранилища данных необходимых для функционирования системы используется Sql Server. Сервер базы данных может быть размещен как на сервере, на котором располагается само приложение, так и на других серверах, доступных в рамках данной сети. Помимо прочего существует возможность использования кластера для обеспечения работы базы данных.

В качестве архитектурного подхода в котором следует выполнить само программное средство выбран микросервисный подход. Данная архитектура сервисной части программного решения обладает рядом преимуществ, среди которых масштабируемость, гибкость, относительная простота внесения изменений. Как результат, все функции, реализуемые на серверной части приложения разби-

ты на ряд самостоятельных компонент. Основные архитектурные компоненты программного средства отображены на рисунке 3.2.

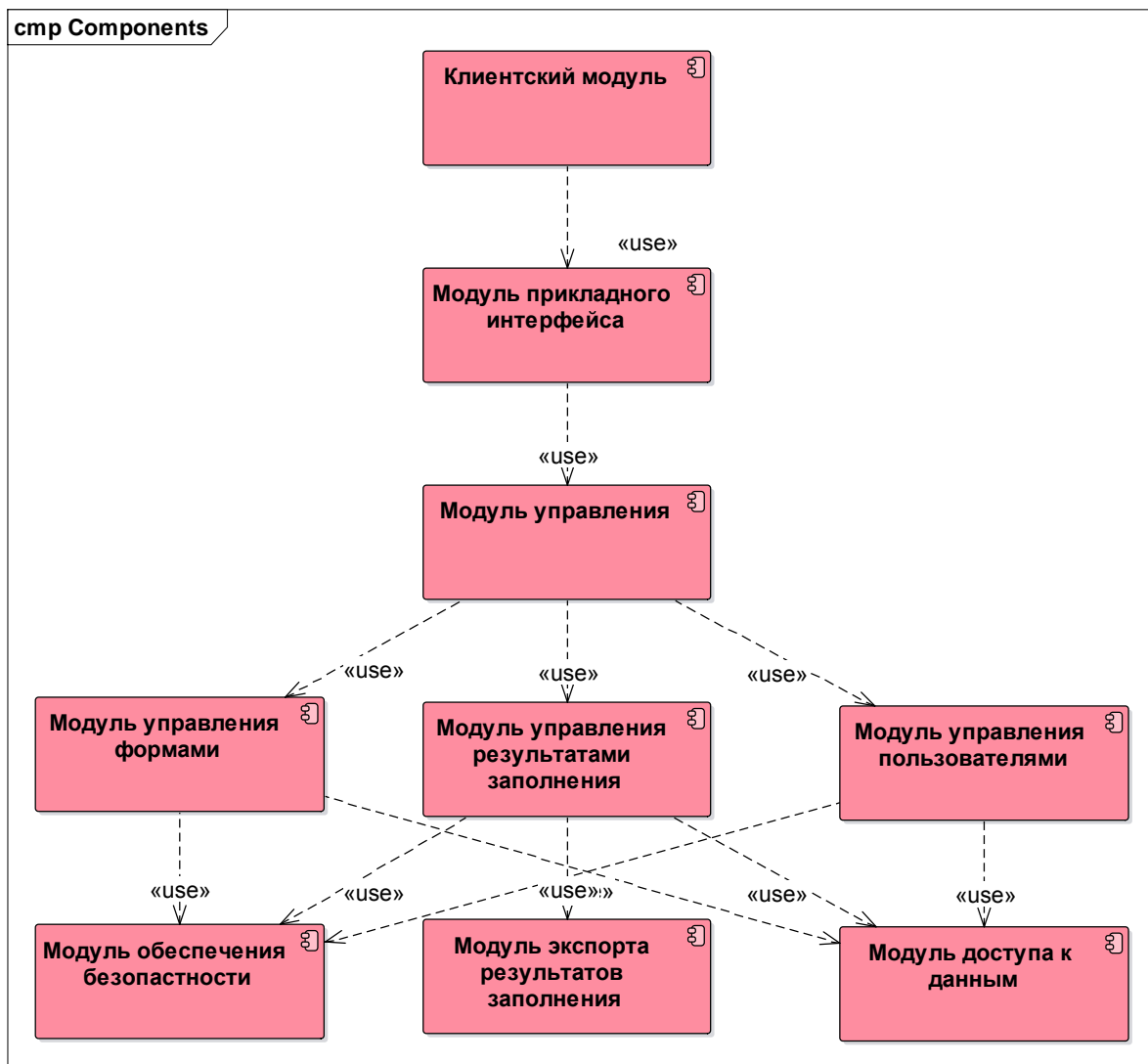


Рисунок 3.2 – Модель архитектуры программного средства для проведения онлайн исследований

Использование микросервисной архитектуры также позволяет размещать отдельные компоненты системы на разных устройствах, таким образом позволяя конфигурировать экземпляр приложения под потребности каждого конкретного клиента.

На рисунке 3.3 представлена схема работы программы в режиме работы суперпользователя. На данной диаграмме наиболее полно отражены функции, реализуемые программным средством.

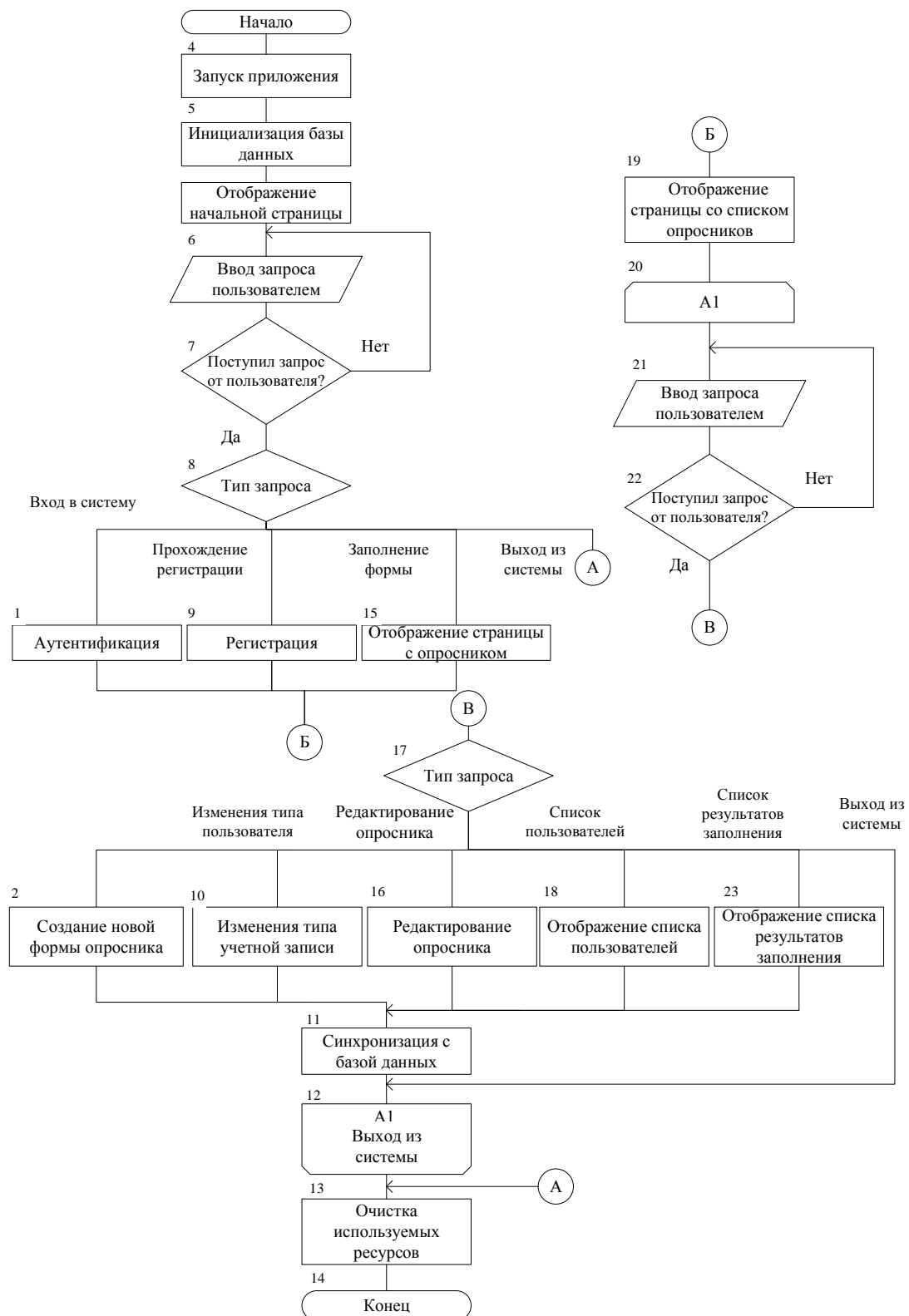


Рисунок 3.3 – Схема работы программы в режиме суперпользователя

## 3.2 Проектирование архитектуры базы данных

Разработанная во второй главе информационная модель нашла свое отражение в физической модели базы данных. При проектировании архитектуры было отмечено, что не все логические сущности необходимы на уровне базы данных. В результате анализа, была проведена денормализация базы данных и как результат, хранение части сущностей осуществляется в полях формата XML. Модель базы данных приняла вид, изображенный на рисунке 3.4 [8-9].

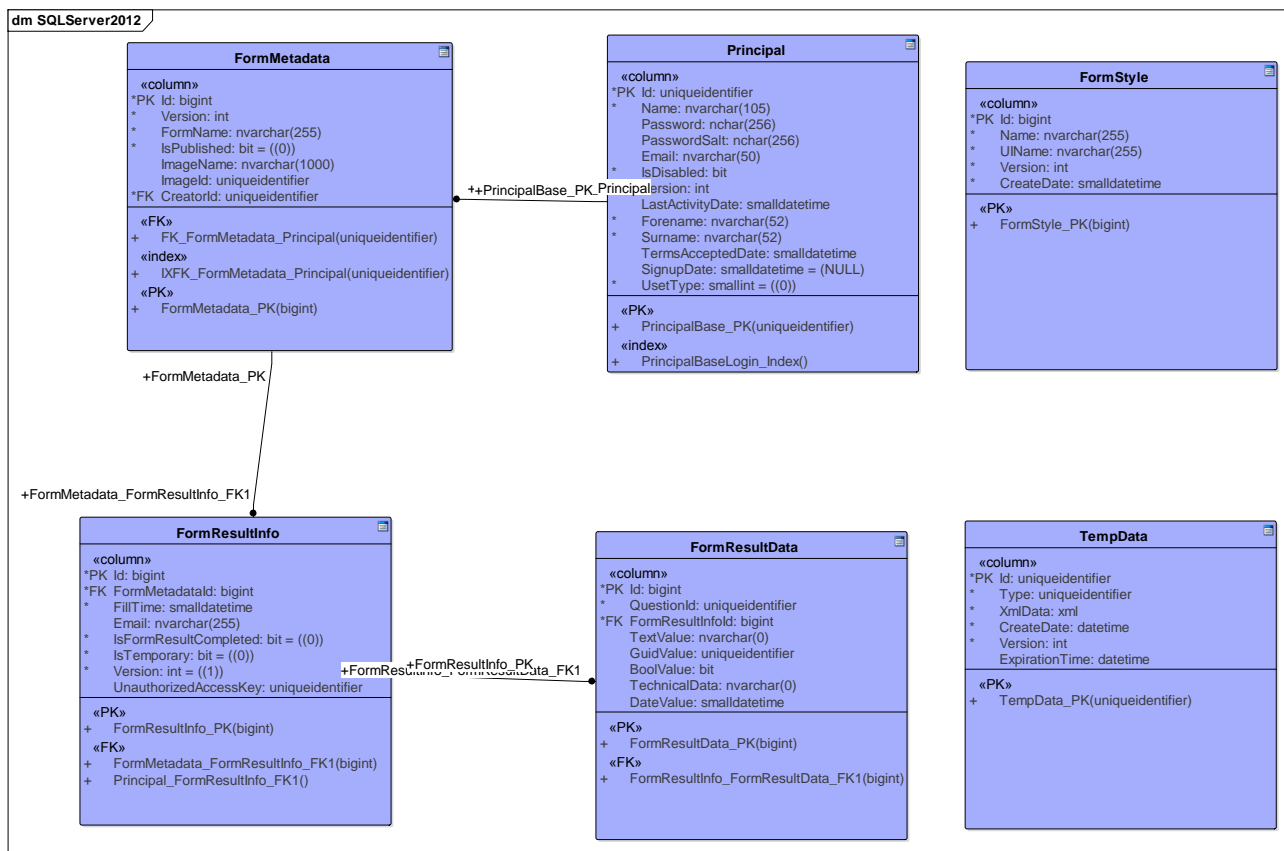


Рисунок 3.4 – Физическая модель базы данных

## 3.3 Проектирование алгоритмов ПС для проведения онлайн-исследований

В процессе проектирования архитектуры был определен детализированный ряд алгоритмов реализации логики функций приложения. Более детально были исследованы функции реализации условной логики для вопросов, страниц и результирующих страниц, алгоритмы обновления текста вопроса, и алгоритм вычисления результирующей страницы.



### 3.3.1 Разработка алгоритма реализации условной логики отображения вопросов

Одной из функциональных возможностей проектируемого приложения является возможность задания правил отображения вопросов на странице. Схема алгоритма, применяемого при реализации данной функциональности, приведена на рисунке 3.5.

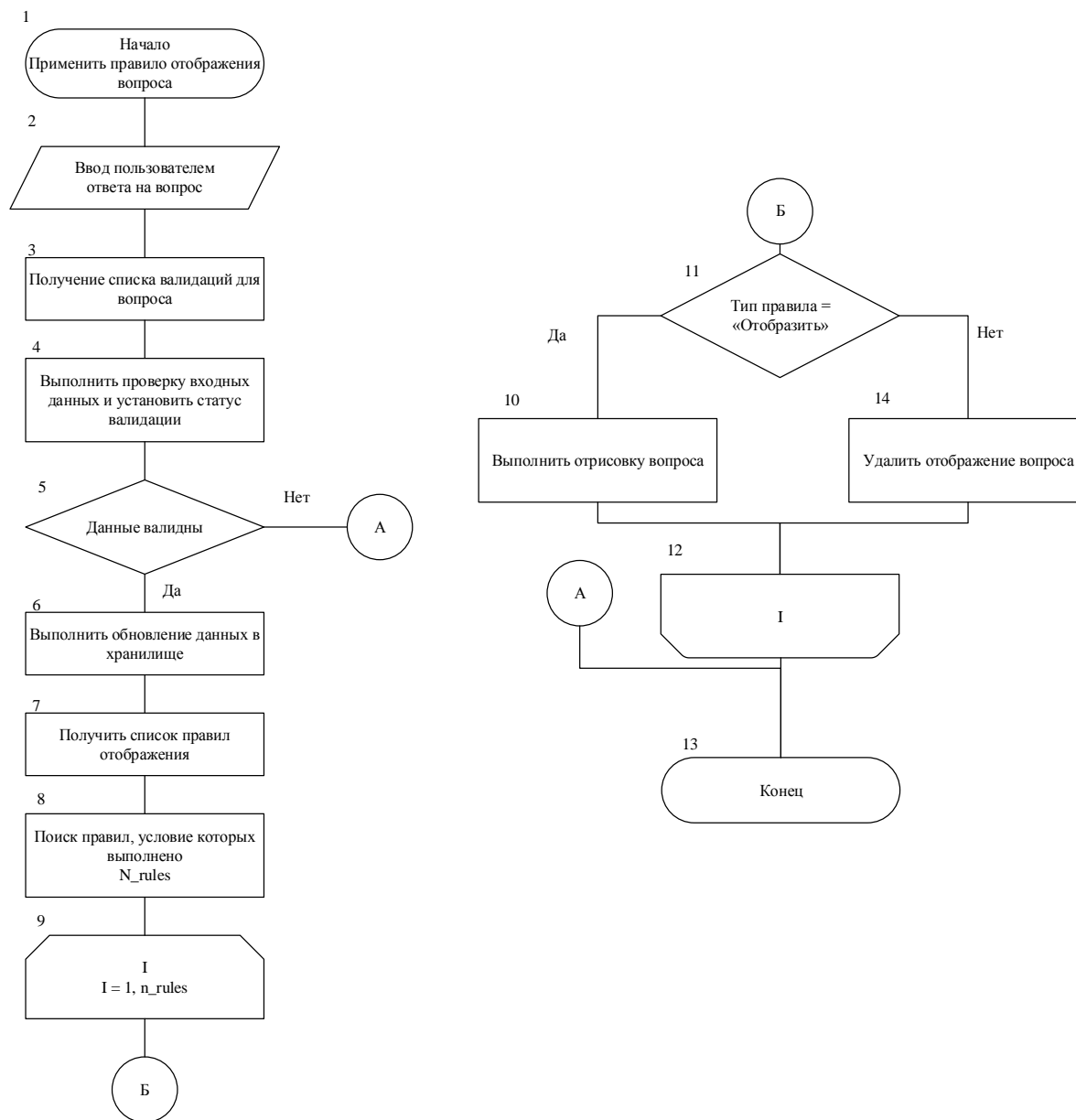


Рисунок 3.5 – Схема алгоритма применения правил отображения вопросов

Как мы видим из схемы алгоритма, логика работы клиентского модуля основывается на принципах реактивного программирования. После ввода пользователем ответа на один из вопросов и выполнения заданных проверок корректности введенных данных, мы выполняем соответствующее действие из вспомогательного класса.

В свою очередь действия вызывают диспетчер, что приводит к обновлению уже отрисованных компонент. Списковая компонента запрашивает правила отображения для каждого элемента вопроса на странице, после чего пробегается по полученному списку правил в поисках такого, условие которого выполняется.

В случае, если найдено более одного правила система должна использовать первое найденное правило. Таким образом мы имеем возможность однозначно определить необходимое поведение системы.

В зависимости от типа правила отображения, делается вывод о необходимости отображения данного вопроса на странице.

Аналогичным образом работают механизмы плавил переходов между отдельными страницами формы.

В процессе реализации данного алгоритма следует использовать функциональный подход. Использование чистых функций положительно сказывается на реактивных интерфейсах, так как сам подход предполагает отсутствие состояний.

### 3.3.2 Разработка алгоритма поиска результирующей страницы

Еще одной возможностью, схожей с логикой правил отображения является логика выбора результирующей страницы. Принципиальное отличие в данном случае заключается в том, что критерием выбора результата является скоринг.

Для каждого ответа на вопрос закрытого типа может быть задана стоимость. При переходе с последней страницы на страницу результата происходит вычисление так называемого скоринга. Скоринг – сумма стоимостей выбранных ответов. Если для какого-либо ответа не задана стоимость – она принимает значение по умолчанию равное 0.

Правила отображения страниц результатов основываются на попадании значения скоринга в заданный в правиле выбора диапазон.

Как и в случае с вопросами, в случае, если критерий отображения выполнен в более чем одном правиле, будет выбрана первая страница, правила выбора которой были выполнены.

В ситуации, когда не был выполнен ни один критерий, будет отображаться страница результатов по умолчанию. Алгоритм поиска результирующей страницы приведен на диаграмме 3.6.



Рисунок 3.6 – Схема алгоритма поиска результирующей страницы

### 3.3.3 Разработка алгоритма формирования текста вопроса.

Одним из недостатков рассмотренных программных решения является невозможность использования ответов пользователя в процессе выполнения. Для устранения этого недостатка спроектирован алгоритм обновления текста вопроса, приведенный на рисунке 3.7.

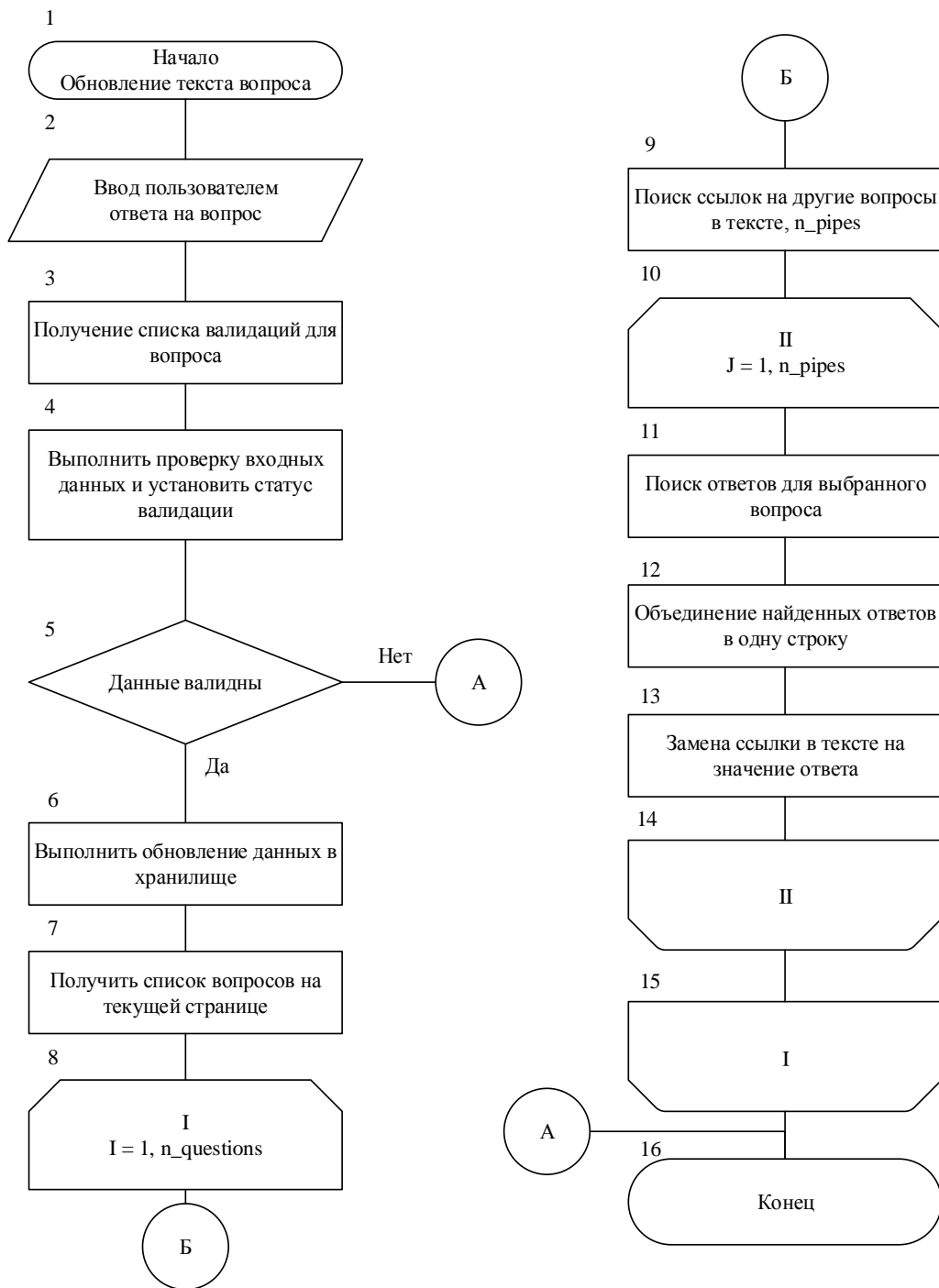


Рисунок 3.7 – Схема алгоритма обновления текста вопроса

В текст вопроса, могут быть вставлены ссылки на другие вопросы. В процессе заполнения, отображаемый вариант текста вопроса будет зависеть от ответов пользователя в процессе работы с программным средством. Все ссылки должны заменяться ответами пользователя на указанный вопрос [11].

## 4 РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

На основе спецификации функциональных требований и спроектированной архитектуры программного средства, а также требований к техническим характеристикам аппаратного обеспечения был произведен выбор наиболее подходящих технологий для разработки программного решения.

### 4.1 Выбор и обоснование языка и среды разработки программного средства

Программное средство можно разделить на две основные части: серверная часть, обеспечивающая работу с базой данных и реализующая сохранение результатов работы с программой; клиентская часть, реализующая интерфейс для работы с программным средством.

Серверная часть программного средства реализована с использованием языка программирования C# и кроссплатформенной версии фреймворка .NET – ASP.Net Core.

.NET Core – это универсальная платформа разработки, которая поддерживается корпорацией Майкрософт и сообществом .NET на сайте GitHub. Она является кроссплатформенной, поддерживает Windows, Mac OS и Linux и может использоваться на устройствах, в облаке, во внедренных системах и в сценариях Интернета вещей.

Перечисленные ниже особенности наиболее полно определяют платформу .NET Core:

- гибкая разработка: может включаться в приложение или устанавливаться параллельно на уровне пользователя или компьютера.
- кроссплатформенность: работает в Windows, Mac OS и Linux; может переноситься в другие операционные системы. Спектр поддерживаемых операционных систем (ОС), ЦП и приложений будет со временем расширяться благодаря усилиям корпорации Майкрософт, других компаний и отдельных лиц.
- программы командной строки: любые сценарии использования продукта можно реализовать посредством командной строки.
- совместимость: платформа .NET Core совместима с .NET Framework, Xamarin и Mono благодаря библиотеке .NET Standard.
- открытый исходный код: платформа .NET Core имеет открытый исходный код и распространяется по лицензиям MIT и Apache 2. Документация распространяется по лицензии CC-BY. .NET Core является проектом .NET Foundation.

Данная версия фреймворка была переработана значительным образом по сравнению с классическим .Net Framework. В том числе изменения коснулись и его структуры (рисунок 4.1).

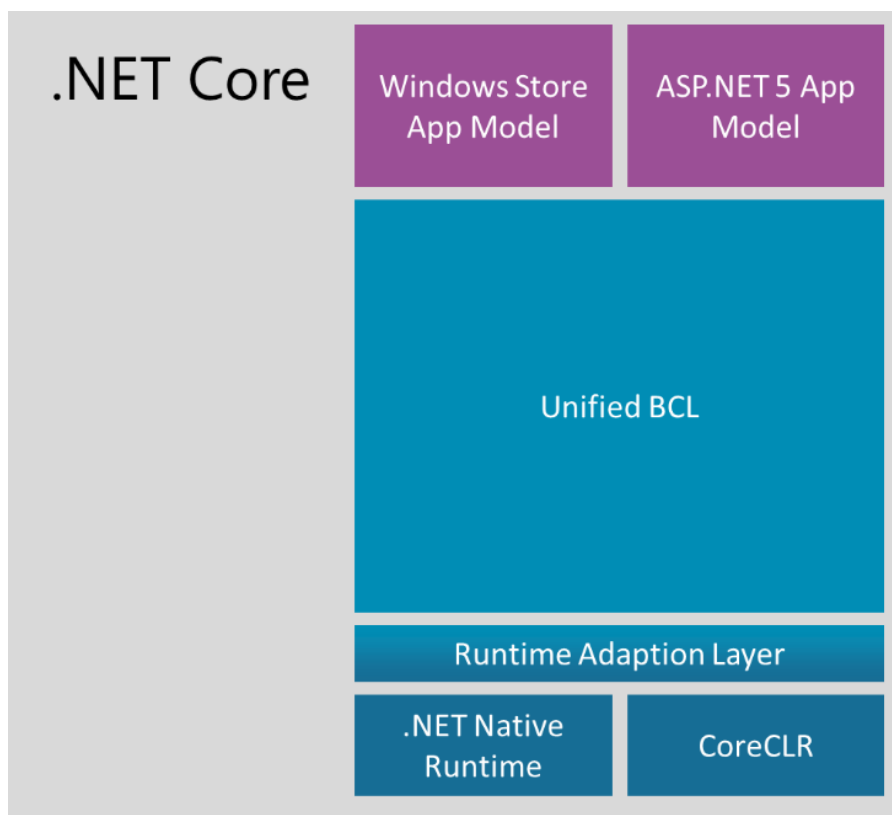


Рисунок 4.1 – Структура .Net Core

Кроссплатформенный подход нашел свое отражение в структуре всего фреймворка. На нижнем уровне мы можем видеть интерпретаторы промежуточного кода для различных платформ. По своей сути .NET Core – это практически полная перезагрузка стека .NET Framework. Из новой платформы по разным причинам был исключён ряд технологий. Следует понимать, что платформа .NET Core рассчитана в первую очередь на разработку для серверных и облачных решений. Для десктопных приложений лучше подходят классический .NET для Windows (с поддержкой WPF и Windows Forms) и Mono для Linux и Mac OS X (с поддержкой Windows Forms).

Главной особенностью языка C# является его ориентированность на платформу Microsoft .NET – создатели C# ставили своей целью предоставление разработчикам естественных средств доступа ко всем возможностям платформы .NET. Видимо, это решение можно считать более или менее вынужденным, так как платформа .NET изначально предлагала значительно большую функцио-

нальность, чем любой из существовавших на тот момент языков программирования.

Кроме того, создатели С# хотели скрыть от разработчика как можно больше незначительных технических деталей, включая операции по упаковке/распаковке типов, инициализации переменных и сборке мусора. Благодаря этому программист, пишущий на С#, может лучше сконцентрироваться на содержательной части задачи.

Многие существующие языки программирования обладают весьма запутанным синтаксисом и конструкциями с неочевидной семантикой – достаточно вспомнить перегруженную значениями открывающую фигурную скобку в С++.

С# занимает некоторую промежуточную позицию: из стандарта языка убраны наиболее неприятные и неоднозначные особенности С++, но в то же время язык сохранил мощные выразительные возможности, присущие для таких языков, как С++, Java или VB.

В качестве языка и фреймворка, на которых велась разработка клиентского модуля приложения были выбраны язык программирования JavaScript и ReactJS.

JavaScript – мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией языка ECMAScript. JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам. Однако на данный момент данный язык может использоваться для реализации самых разных типов приложений, начиная с десктопных и заканчивая приложениями для мобильных устройств.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

JavaScript является объектно-ориентированным языком, но используемое в языке прототипирование обуславливает отличия в работе с объектами по сравнению с традиционными класс-ориентированными языками. Кроме того, JavaScript имеет ряд свойств, присущих функциональным языкам – функции как объекты первого класса, объекты как списки, карринг, анонимные функции, замыкания – что придаёт языку дополнительную гибкость.

Несмотря на схожий с Си синтаксис, JavaScript по сравнению с языком Си имеет коренные отличия:

- объекты с возможностью интроспекции;
- функции как объекты первого класса;
- автоматическое приведение типов;
- автоматическая сборка мусора;
- анонимные функции.

Для реализации клиентской части клиент-серверных приложений данный язык предлагает множество библиотек и фреймворков. В данном дипломном проекте использован React JS для построения клиентских форм.

React JS – это библиотека написанная Facebook, для того, чтобы ускорить взаимодействие с DOM. В первую очередь использование данной библиотеки призвано ускорить процесс ведения разработки программных модулей.

Синтаксис, поддерживаемый библиотекой прост и понятен даже людям, не работавшим ранее с языками программирования. Использование механизмов декомпозиции отображения на отдельные компоненты позволяет логично разбить определенные страницу на отдельные элементы и повторно использовать часто повторяющийся код более эффективно.

Данный фреймворк позволяет разработчикам создавать большие web-страницы, которые используют данные, которые изменяются в течение времени, без необходимости осуществлять перезагрузку страницы. Основными задачами, которые реализует данный фреймворк заключается в том, чтобы быть быстрым, максимально простым, легким для дальнейшего масштабирования. Однако это привело к наличию дополнительных ограничений, связанных с тем что данная библиотека представляет собой только лишь отображение. Однако на данный момент существует огромное количество библиотек, которые позволяют расширить возможности React JS.

Клиентская часть данного дипломного проекта было написано с использованием реактивной архитектуры. Данная разновидность архитектуры предполагает однонаправленный поток данных, который упрощает расширение существующей функциональности, а также упрощает процесс внесения новых изменений. Использование данного подхода положительно сказывается как на времени разработки отдельных модулей, так и на интеграция этих модулей и в систему.

## **4.2 Диаграмма классов программного средства**

Для наглядной демонстрации всех классов, разработанных в процессе работы над серверной частью программного средства, была построена диаграмма классов (рисунок 4.2).

На данной диаграмме изображены имена классов, их местоположение в иерархии программного средства. Помимо этого, здесь также можно найти имена методов, поля и свойства разработанных классов, а также взаимосвязи между ними.



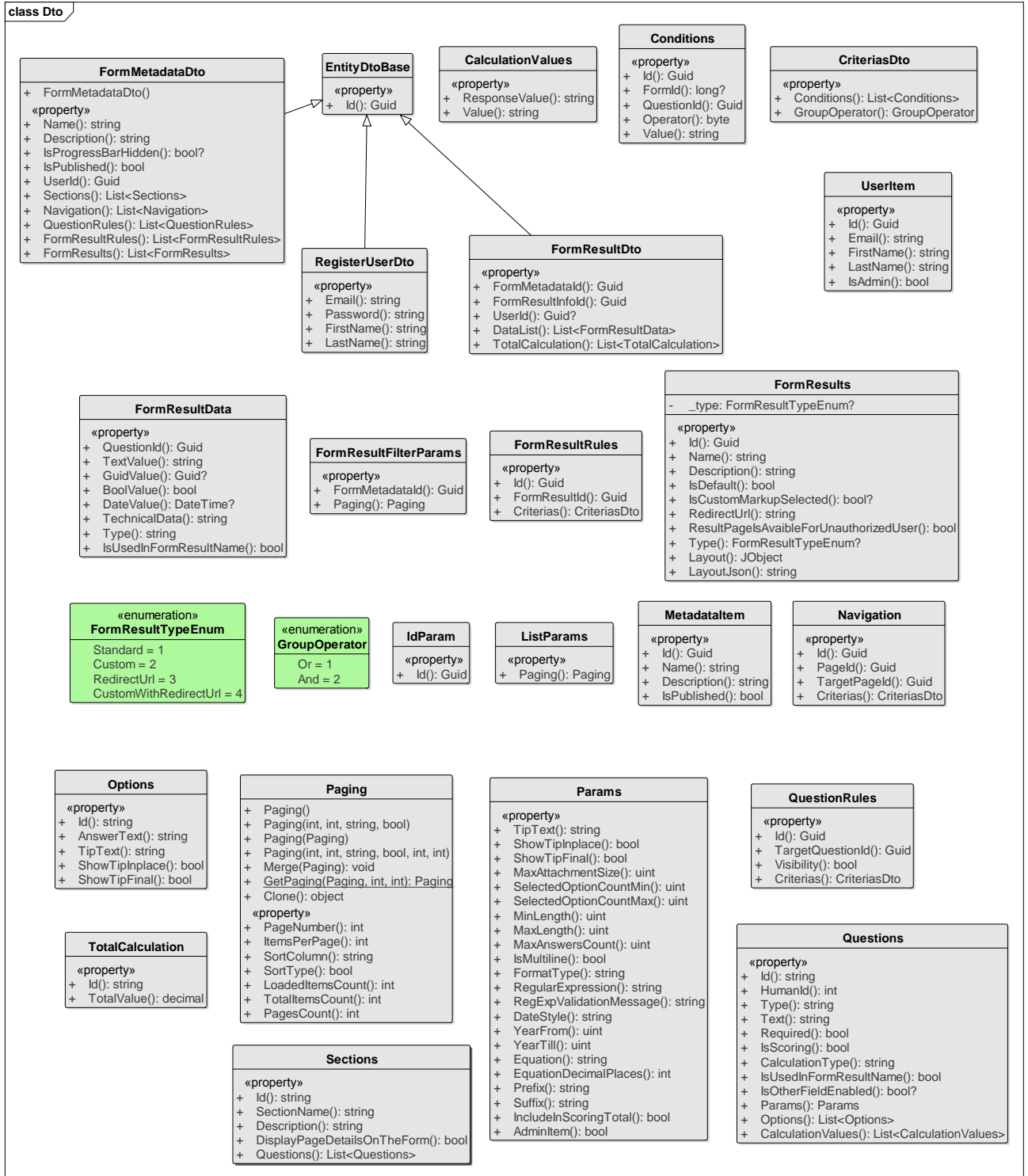


Рисунок 4.2 – Диаграмма классов связующего модуля приложения

На данной модели перечислены все классы, которые используются для передачи данных при взаимодействии отдельных частей приложения. Большая часть классов представляет собой типы данных описывающих модели предметной области, а также перечисления, которые используются, для реализации логики в предметной области приложения.

На рисунке 4.3 изображена диаграмма классов прикладного программного интерфейса. Данные классы осуществляют обработку пользовательских запросов, приходящих с клиентского модуля.

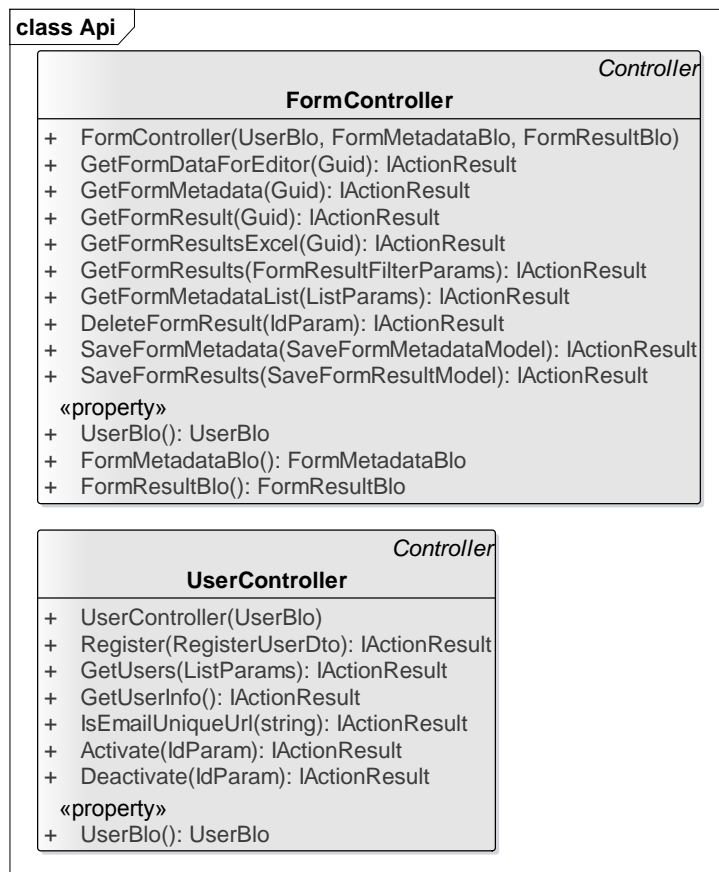


Рисунок 4.3 – Диаграмма классов прикладного программного интерфейса

Классы, изображенные на диаграмме, содержат зависимости на классы логики приложения. Так как приложение разработано с использованием классического подхода к построению клиент серверных приложений в нем реализована слоистая архитектура. При этом работа с хранилищами данных инкапсулирована в классах логики. Таким образом прикладной интерфейс осуществляет взаимодействие только лишь со слоем логики приложения, без привязки к конкретным способам работы с источниками данных.

На рисунке 4.4 отражены взаимосвязи между классами логики, реализованными в приложении. Как можно видеть на диаграмме все классы реализующие программную логику наследуются от общего класса. Данный подход позволяет повторно использовать повторяющиеся алгоритмы работы с данными.

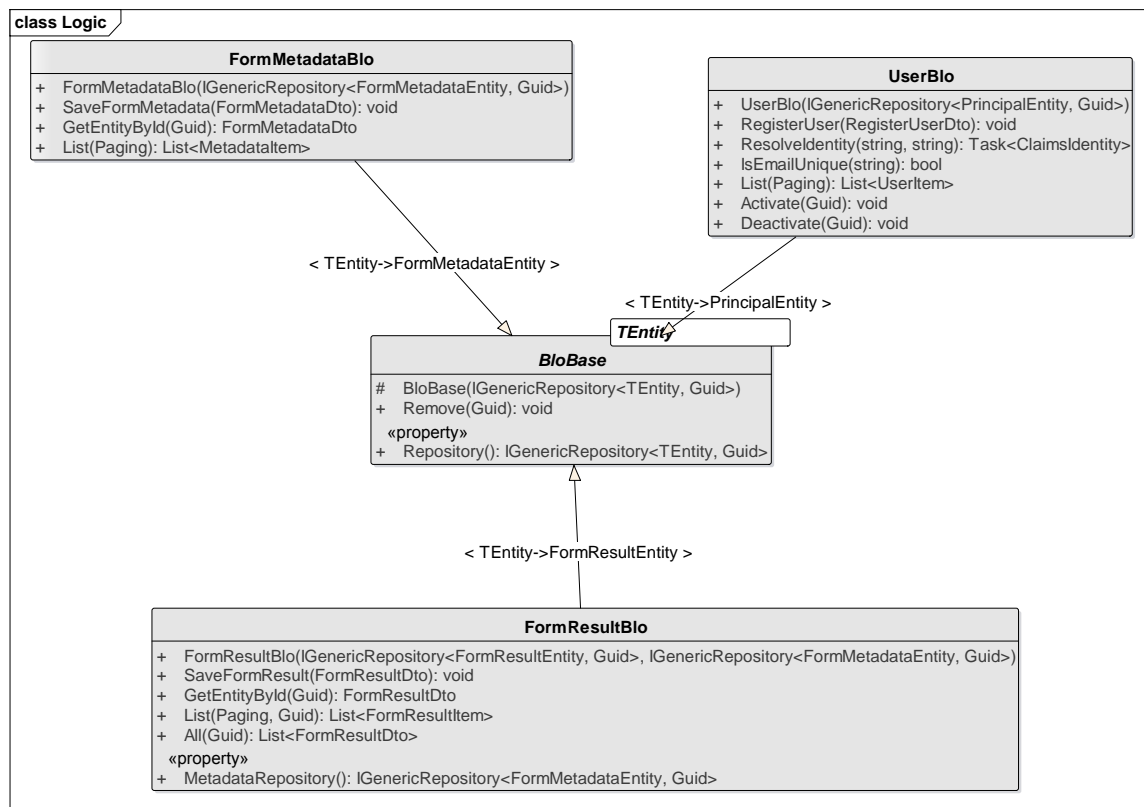


Рисунок 4.4 – Диаграмма классов слоя логики приложения

Использование обобщений при написании программного кода позволяет повысить скорость его выполнения, а также безопасность при работе со структурами данных, за счет того, что в этом случае выполняется проверка типов данных на этапе компиляции программного модуля, а не на этапе выполнения.

Диаграмма на рисунке 4.5 содержит классы инициализации приложения, а также классы, реализующие загрузку конфигураций. Класс Startup в котором осуществляется конфигурирование всех частей приложений реализован с использованием частичных классов. Разбиение данного класса на отдельные части позволяет получить четкое разграничение реализованной в нем логики, что упрощает дальнейшее внесение изменений, а также действия, связанные с поддержкой и покрытием тестами.

Изображенные на диаграмме классы являются стандартными для приложений, написанных с использованием ASP.NET Core. Однако содержат методы,

переопределенные в соответствии со спецификой и логикой разрабатываемого приложения.

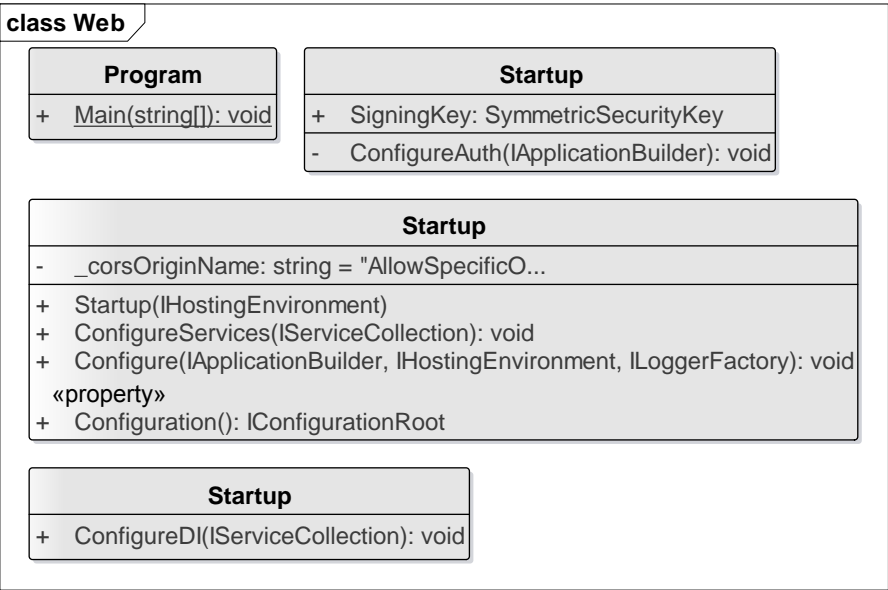


Рисунок 4.5 – Диаграмма классов модуля инициализации

На рисунке 4.6 изображены классы, реализующие авторизацию пользователя в системе. Логика авторизации реализована с использованием паттерна посредник. Данный подход позволяет использовать разработанную компоненту при реализации аналогичного модуля в любом другом приложении, реализованном с использованием ASP.NET Core. Изображенный на диаграмме класс осуществляет проверку корректности ключевого значения, а также генерацию токена при входе пользователя в систему.

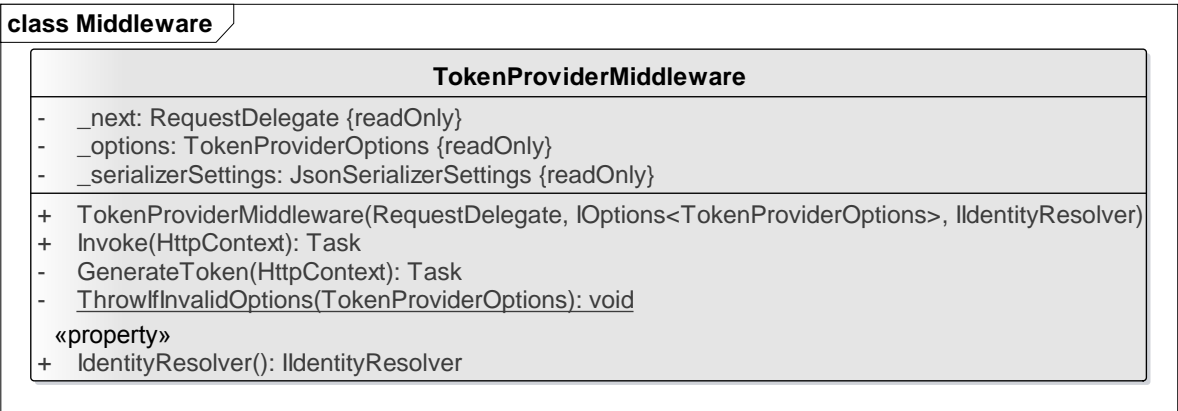


Рисунок 4.6 – Диаграмма классов компоненты авторизации

Для реализации внедрения зависимостей в процессе разработки активно применялось использование интерфейсов для описания функциональных возможностей разрабатываемых модулей. На рисунке 4.7 изображены интерфейсы классов, реализующих работу с базой данных.

IGenericRepository – интерфейс обобщенного репозитория, который отображает основные возможные операции по работе с записями в соответствующей конкретному типу таблице. В процессе разработки был разработан класс, реализующий логику работы с базой данных в соответствии с данным интерфейсом.

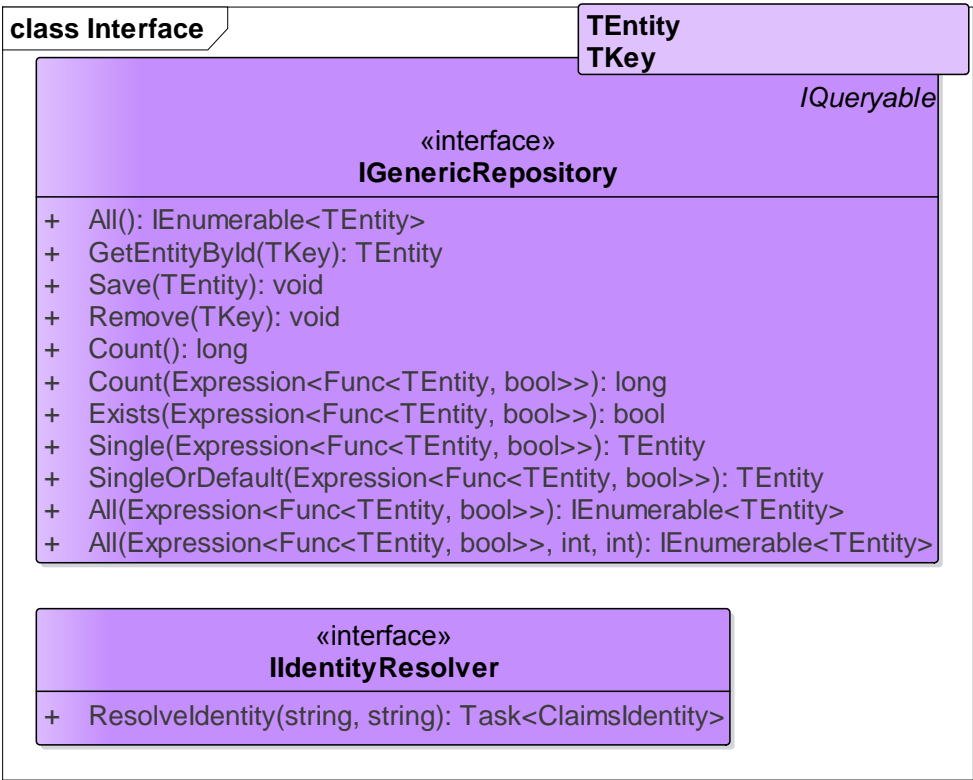


Рисунок 4.7 – Диаграмма классов компоненты интерфейсов

### 4.3 Описание компонент клиентской части программного средства

Для реализации клиентской части на React JS все элементы управления были разбиты на отдельные компоненты. Каждая отдельная компонента предназначена для реализации отображения одного атомарного элемента.

Некоторые разработанные компоненты представляют собой сложные и используют другие. Такие высокоуровневые компоненты реализуют целые страницы или формы приложения.

Все разработанные компоненты можно отнести к одной из следующих групп: компоненты форм ввода, компоненты вопроса для модуля редактора формы опросника, вопросы опросника для модуля заполнения формы опросника, вспомогательные компоненты.

Каждая описанная компонента содержит ряд методов, описывающих ее работу. Среди них можно выделить следующие: `getInitialState()`, `ComponentDidMount()`, `componentWillMount()`, `ComponentWillUnmount()`, `render()`, `getProps()`.

Метод `getInitialState` содержит программный код, который осуществляет инициализацию состояния компоненты перед первым отображением компоненты на странице. Данный метод вызывается один раз и позволяет задать начальные условия отображения, или же осуществить подготовку данных.

Метод `componentWillMount()` также вызывается один раз перед внедрением компоненты в дерево узлов на веб-странице. В данном методе как правило осуществляется подписка на события, по которым будет осуществляться дальнейшая перерисовка компоненты. Как правило в данном методе осуществляется подписка на обновление значений в хранилище данных.

Единственным обязательным для каждой компоненты методом является метод `render()`. В данном методе осуществляется генерация кода HTML, который будет отображаться в месте подключения генерируемой компоненты. Внутри данного метода могут использоваться как другие пользовательские компоненты, так и стандартные для React JS компоненты.

Передача параметров между компонентами осуществляется через использование свойств. Согласно лучшим практикам написания кода, данные свойства должны быть описаны в методе `getProps()`. Данный метод возвращает список свойств, поддерживаемых данным компонентом.

## 5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СОЗДАНИЯ ФОРМ ОПРОСНИКОВ

Для проверки правильности работы программного обеспечения и реализации функциональных требований проведено тестирование модулей разработанного программного обеспечения.

На ранних этапах тестирование проводилось при помощи юнит-тестирования. Юнит-тесты позволяли еще на начальном этапе разработки выявлять ошибки в программном средстве, а также позволяли более полно и детально проверить самые основные и важные модули программного средства.

На более поздних этапах в качестве метода тестирования программного обеспечения выбран метод функционального тестирования.

Функциональное тестирование является основным видом тестирования программного обеспечения. Каждая функция программы тестируется и при этом делается вывод об ее правильности. Очевидно, что по всей области определения проверить функцию невозможно и поэтому каждая функция проверяется на правильность в некоторых точках области её определения.

Данный метод тестирования позволяет:

- обнаружить некорректные или отсутствующие функции;
- обнаружить ошибки интерфейса;
- обнаружить ошибки во внешних структурах данных (файлы, базы данных).

В рамках разработки дипломного проекта функциональное тестирование реализовано с помощью набора тест-кейсов, собранного в тестовый сценарий, представленный в таблице 5.1.

Таблица 5.1 – Результаты тестирования приложения

№	Тестовый случай	Ожидаемый результат	Фактический результат
1	1)Открыть приложение. 2)Нажать на кнопку Register.	Форма регистрации нового пользователя	Совпадает с ожидаемым.
2	1)Открыть приложение. 2)Ввести данные учетной записи суперпользователя. 3)Нажать кнопку Login.	Пользователь успешно авторизован, открыта главная страница пользователя со списком всех созданных в системе форм опросников	Совпадает с ожидаемым.

Продолжение таблицы 5.1 – Результаты тестирования приложения

№	Тестовый случай	Ожидаемый результат	Фактический результат
3	1)Открыть приложение повторно после авторизации в системе, в течение 5 минут с момента первой авторизации.	Пользователь по-прежнему авторизован в системе	Совпадает с ожидаемым.
4	1)Открыть приложение. 2)Ввести данные учетной записи суперпользователя. 3)Нажать кнопку Login. 4)Нажать кнопку Log out.	Переход на страницу авторизации	Совпадает с ожидаемым.
5	1)Перейти в защищенную область приложения по прямому адресу.	Переход на страницу No permission	Совпадает с ожидаемым.
6	1)Перейти на страницу заполнения формы по прямому адресу.	Переход на страницу заполнения формы	Совпадает с ожидаемым.
7	1)Перейти на страницу заполнения формы по прямому адресу. 2) В адресе указать идентификатор еще не созданной формы опросника.	Переход на страницу Object not found	Совпадает с ожидаемым.
8	1)Открыть приложение. 2)Нажать на кнопку Register. 3)Заполнить поля формы 4)Нажать кнопку Register	Произошла регистрация пользователя. Переход на страницу авторизации.	Совпадает с ожидаемым.
9	1)Открыть приложение. 2)Ввести данные учетной записи суперпользователя. 3)Нажать кнопку Login. 4)Нажать кнопку Create form	Переход в модуль создания новой формы опросника	Совпадает с ожидаемым.
10	1)Авторизоваться как пользователь системы 2)Нажать кнопку Login. 3)Нажать кнопку Create form 4)Нажать Save changes	Срабатывание валидации со списком обязательных полей	Совпадает с ожидаемым.



Продолжение таблицы 5.1 – Результаты тестирования приложения

№	Тестовый случай	Ожидаемый результат	Фактический результат
11	1)Авторизоваться как пользователь системы 2)Нажать кнопку Login. 3)Нажать кнопку Create form 4)Ввести все обязательные для заполнения поля 5)Нажать Save changes.	Форма сохранена. Переход на страницу со списковой формой.	Совпадает с ожидаемым.
12	1)Открыть приложение. 2)Ввести данные учетной записи суперпользователя. 3)Нажать кнопку Login. 4) Нажать кнопку Users.	Переход на страницу со списком пользователей системы.	Совпадает с ожидаемым.
13	1)Открыть приложение. 2)Ввести данные учетной записи суперпользователя. 3)Нажать кнопку Login. 4) Нажать кнопку Users. 5) Для неадминистративного пользователя нажать кнопку Activate.	Обновление страницы. Тип пользователя изменен на административный	Совпадает с ожидаемым.
14	1)Открыть приложение. 2)Ввести данные учетной записи суперпользователя. 3)Нажать кнопку Login. 4) Нажать кнопку Users. 5) Для неадминистративного пользователя нажать кнопку Deactivate	Обновление страницы. Тип пользователя изменен на обыкновенный пользователь.	Совпадает с ожидаемым.
15	1)Открыть приложение. 2)Ввести данные учетной записи суперпользователя. 3)Нажать кнопку Login. 4) Нажать кнопку Users. 5) Найти учетную запись суперпользователя.	Учетная запись суперпользователя отсутствует.	Совпадает с ожидаемым.

Продолжение таблицы 5.1 – Результаты тестирования приложения

№	Тестовый случай	Ожидаемый результат	Фактический результат
16	1) Авторизоваться как пользователь системы. 2) Открыть по прямому адресу страницу формы регистрации	Переход на страницу Object not found	Совпадает с ожидаемым.
17	1) Авторизоваться как пользователь системы. 2) Открыть по прямому адресу страницу формы авторизации	Переход на страницу Object not found	Совпадает с ожидаемым.
18	1) Перейти на страницу заполнения формы по прямому адресу. 2) Заполнить форму до конца и нажать Finish	Ответ сохранен. Переход на страницу с результатами заполнения.	Совпадает с ожидаемым.
19	1) Авторизоваться как пользователь системы. 2) Открыть форму с ответами 3) Перейти на вкладку с ответами	Отображается список всех ответов на форму	Совпадает с ожидаемым.
20	1) Перейти на страницу заполнения формы по прямому адресу. 2) Заполнить первую страницу формы 3) Перейти на вторую страницу 4) Нажать кнопку Previous	Отображается первая страница формы. Поля ввода заполнены ответами пользователя.	Совпадает с ожидаемым.

В результате итогового тестирования серьезных дефектов выявлено не было. Все тестовые случаи были пройдены успешно. Фактический результат совпал с ожидаемым результатом в каждом из тестовых случаев.

## 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для корректной работы программного средства предъявляются следующие минимальные требования к техническим характеристикам сервера:

- операционная система: Linux (Ubuntu 14.04, Mint 17) Windows (версии 7, 8 или 10)
- Xeon 2.2 ГГц или более быстродействующий процессор;
- оперативная память 4 Гбайт или более;
- сетевая карта Ethernet 1 Гбит.
- доступный объем дискового пространства 10 Гбайт.

Дополнительно необходимо наличие развернутой системы управления базами данных MondoDB, ASP.NET Core, пакетные менеджер npm.

### 6.1 Развертывание сервера приложения

Готовые модули приложения могут быть поставлены для развертывания в виде пакета для развертывания с использованием WebDeploy, а также в виде архива с файлами проекта.

Для развертывания с использованием WebDeploy необходимо запустить следующую команду:

```
msdeploy -verb:sync -source:metakey=/lm/w3svc/1  
dest:metakey=/lm/w3svc/2 -verbose
```

Выполнение данной команды осуществит развертывание пакета приложения на сервере. Перед началом работы с приложением рекомендуется изменить параметры учетной записи суперпользователя, которая будет создана в процессе развертывания приложения.

Так как приложение для своей работы требует развернутую СУБД, необходимо указать имя базы данных и URL для установления соединения с сервером базы данных. Установка этих параметров производится в файле конфигурирования приложения appsettings.json.

Для более детальной настройки приложения следует использовать параметры командной строки утилиты MsDeploy.

После развертывания приложение по умолчанию будет доступно по адресу 127.0.0.1. Одновременно с запуском веб-сервера будет производиться старт приложения.

Клиентский доступ к приложению может быть осуществлен при помощи веб-браузера как персональных компьютеров, так и мобильных устройств.

## 6.2 Использование программного средства

### 6.2.1 Авторизация пользователя

При обращении к системе пользователя не имеющего аккаунта в системе или не прошедшего авторизацию, ему будет отображена страница входа в систему (рисунок 6.1).

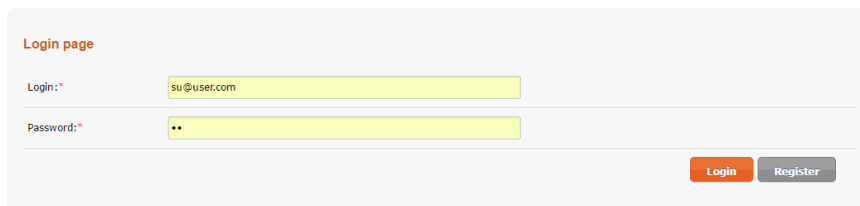


Рисунок 6.1 – Форма авторизации пользователя в системе

Для входа пользователя в систему необходимо указать адрес электронной почты и пароль, введенные при регистрации.

### 6.2.2 Регистрация в системе

В случае, если новый пользователь, не имеющий своей учетной записи, желает создать новую форму опросника необходимо зарегистрироваться в системе, нажав кнопку «Register». Страница регистрации нового пользователя представлена на рисунке 6.2.

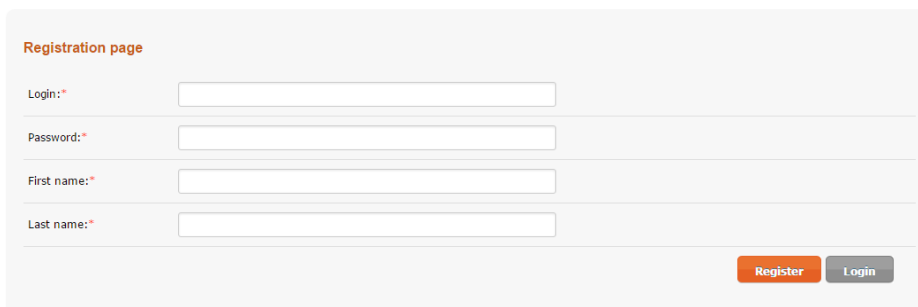


Рисунок 6.1 – Форма регистрации пользователя в системе

В процессе развертывания системы создается учетная запись суперпользователя, для того чтобы иметь возможность сконфигурировать систему с ее использованием. Параметры учетной записи суперпользователя могут быть сконфигурированы в файле конфигурации приложения.

Для регистрации в системе, необходимо ввести адрес электронной почты, пароль, имя и фамилию пользователя. По нажатию на кнопку «Register» будет осуществлено создание новой учетной записи. В случае успешного завершения процессе будет осуществлен переход на страницу авторизации в системе.

### 6.2.3 Управление формами опросников

При успешном прохождении авторизации пользователя в системе будет отображена главная страница приложения. На данной странице расположены ссылки на основные части приложения. Пример главной страницы суперпользователя отображен на рисунке 6.3.

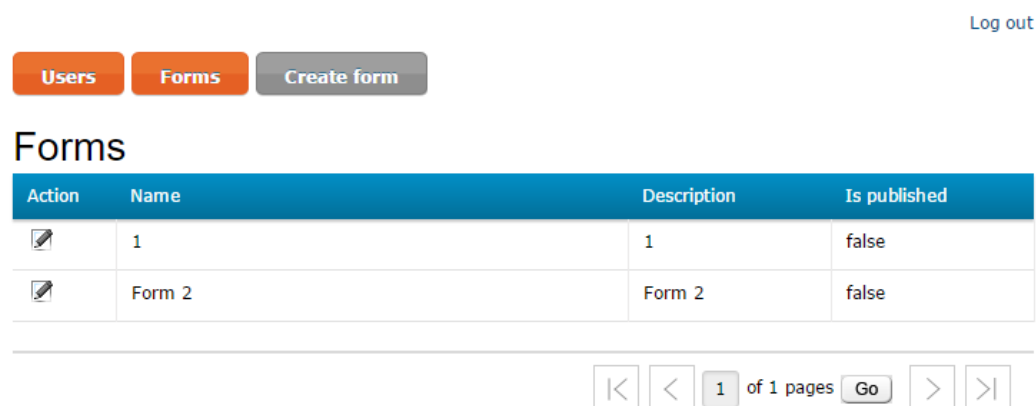


Рисунок 6.3 – Главная страница суперпользователя

Главные страницы остальных типов пользователей схожи с главной страницей суперпользователя. Главным отличием данной страницы является наличие ссылки на страницу со списком пользователей системы.

На главной странице приложения размещена списковая форма с формами опросников, созданных в системе. Административные пользователи имеют доступ к списку всех форм опросников, созданных в системе, обычные же пользователи имеют доступ.

Кроме этого на странице размещена кнопка «Log out», нажатие на которую приводит к завершению работы пользователя с системой.

### 6.2.4 Управление пользователями

Нажатие на кнопку «Users» приводит к переходу на страницу со списковой формой пользователей. Данная форма содержит перечисление, всех пользователей, зарегистрированных в системе. Для каждого из существующих пользователей отображаются его адрес электронной почты, фамилия и имя, а также тип учетной записи.

Суперпользователь может осуществить наделение любого из этих пользователей правами администратора, нажав на кнопку «Activate». На этой же форме возможно и осуществление обратного текущего действия. Внешний вид формы доступен на рисунке 6.4.

Action	First name	Last name	Email	Is admin
<a href="#">Deactivate</a>	John	Buyer	buyer@user.com	true

Рисунок 6.4 – Списковая форма пользователей в системе.

### 6.2.5 Модуль разработки формы опросника

Доступ к модулю осуществляется по нажатию на кнопку «Create form» или открытие ранее созданной формы на редактирование. Общий вид модуля представлен на рисунке 6.5.

Form name:\*

Description:

Publish form: ☐

Hide Progress bar: ☐

Рисунок 6.5 – Модуль разработки формы опросника

Для упрощения взаимодействия с модулем разработки новой формы, функциональность разделена на отдельные группы, для каждой из которых выделена отдельная секция. На рисунке 6.5 расположены главные детали формы: имя, описание, статус публикации, и опция определяющая необходимость отображения прогресса заполнения формы. Поле имени формы является обязательным для заполнения. Поле статуса публикации, не активно, в случае, если форма опросника не удовлетворяет правилам валидации.

При внесении изменений на форме, отображается кнопка сохранения изменений. Нажатие на кнопку приведет к подготовке изменений для сохранения на сервере.

Клик по следующей вкладке приводит к открытию формы со списком страниц опросника (рисунок 6.6). Порядок строк в таблице можно менять, перетаскивая строки, а также по нажатию на соответствующие кнопки.

Action	Move	Number	Name	Number of questions
	↓	1	<u>1</u>	0
	↓ ↑	2	<u>2</u>	0
	↑	3	<u>3</u>	0

Рисунок 6.6 – Списковая форма страниц опросника

Для каждой страницы доступен ряд действий: открыть на просмотр, открыть на редактирование, удалить. Кроме этого чуть выше расположена кнопка создания новой страницы.

При создании новой страницы пользователю необходимо указать ее имя, а также есть возможность ввести более подробное описание. Ввод описания осуществляется при помощи редактора TinyMCE, который позволяет ввести и от-

форматировать описание под потребности любого пользователя в системе. На рисунке 6.7 можно увидеть пример добавления новой страницы на форму.

Также администратор формы может задать правила переходов для каждой отдельно взятой страницы опросника. Задание логики переходов можно осуществить посредством нажатия на кнопку «Add rule», а также на вкладке с правилами отображения.

The screenshot shows a web application interface for editing a form. At the top, there is a horizontal tab bar with five tabs: 'Main details', 'Pages' (which is currently selected), 'Rules', 'Result page', and 'Results'. Below the tabs, on the right side, are three orange buttons: 'Copy', 'Save', and 'Cancel'. The main content area is divided into two sections. The upper section is for editing a page and includes a 'Page name:' label with a red asterisk and an empty text input field. Below this is a 'Description:' label followed by a rich text editor. The rich text editor has a toolbar with icons for undo, redo, bold, italic, underline, bulleted list, numbered list, decrease indent, increase indent, link, unlink, image, eye (visibility), code, and fullscreen. Below the text editor is a checkbox labeled 'Display page details on the form'. The lower section is labeled 'Rule(s):' and contains an orange 'Add rule' button.

Рисунок 6.7 – Форма создания новой страницы

Каждая страница содержит один или более вопросов. Клик на имя страницы на списковой форме приведет к отображению списка вопросов на странице. Описанная списковая форма полностью аналогична ранее рассмотренной.

Система позволяет осуществлять добавление различных типов вопросов на страницу. Для осуществления работы с вопросами пользователь должен перейти на списковую форму вопросов страницы и открыть вопрос на редактирование, или же создать новый вопрос.

Каждый вопрос в системе имеет текст вопроса, тип и флаг обязательности ввода ответа на вопрос при заполнении формы. Кроме того, для каждого отдельного вопроса можно задать правила его отображения в зависимости от ответов на другие вопросы. Базовый вид формы создания вопроса отображен на рисунке 6.8 и включает в себя поля ввода, для ранее описанных параметров. Для упро-



щения навигации между вопросами вод формой редактирования вопроса отображена таблица со списком вопросов на текущей странице.

Page 1 Question ID 2

Save Cancel

Question options:

☐ Mandatory question

Select the type of the question: \* Please select...

Question text: \*

Rule(s):

Add rule

Action	Move	Number	Question	Type	Is mandatory?	Use in saved form result name
--------	------	--------	----------	------	---------------	-------------------------------

Рисунок 6.8 – Базовая форма создания нового вопроса

При выборе типа вопроса, будут отображены дополнительные параметры, соответствующие выбранному типу вопроса. Примером таких параметров могут быть дополнительные валидации пользовательского ввода, список вариантов ответа на вопрос (рисунок 6.9), сообщение, которое будет отображено пользователю, при вводе ответа на вопрос, стоимость ответа для реализации вычисления скоринга и др.

Options:

Add new

Use "Other" field ☐

The last answer choice will have a text field next to it.

^ No

^ Yes

Рисунок 6.9 – Форма управления опциями для закрытых вопросов

Некоторые типы вопросов позволяют администратору формы указать стоимость вариантов ответа. Данные стоимости используются при вычислении скоринга, который в свою очередь может использоваться при вычислении результирующей страницы. Задание стоимости ответам изображено на рисунке 6.10.

Calculation values:

☒ Enable scoring

Assign value for

☐ Entire item

☒ Each choice

Enter a value for each choice.

Values:

10	No
5	Yes

Рисунок 6.10 – Форма задания стоимости вариантов ответов.

Возможность управлять ходом заполнения опросника одна из важнейших функций всего приложения. Просмотр и редактирование правил осуществляется как на страницах редактирования элементов формы, так и на отдельной вкладке с правилами. Здесь сведены вместе правила отображения вопросов, логика переходов между страницами и логика выбора страницы результата. Пример создания правила перехода отображен на рисунке 6.11.

1

Delete all New rule

▼

Delete

Number: 1

After 1

Skip to: 3 when ANY of its criteria match:

Are you student? Is Yes + -

Рисунок 6.11 – Форма создания нового правила перехода между страницами

Внешний вид формы создания правил несколько отличается в зависимости от типа правила, типа выбранного критерия выполнения правила и типа вопроса.

Одной из последних вкладок, размещенных на странице, является вкладка управления страницами с результатом. Каждый опросник должен иметь как минимум одну такую страницу, которая будет отображаться пользователем по умолчанию. Списковая форма имеет массу сходств с другими аналогичными формами и представлена на рисунке 2.12. В случае если создано более одной результирующей страницы, на списковой форме пользователь может выбрать одну из них, как страницу результата по умолчанию.

Рисунок 6.12 – Списковая форма страниц результатов заполнения

Детальная форма не нуждается в дополнительном описании так как, полностью аналогична форма создания новой страницы с вопросами.

Еще одной интересной возможностью, при использовании конструктора форм опросников, является возможность использования введенных пользователем ответов в тексте вопроса, или описании станицы. Для того, чтобы воспользоваться этой возможностью администратор формы может кликнуть на кнопку «Insert pipe» вверху страницы. Данное действие приведет к отображению диалогового окна, в котором можно выбрать вопрос, ответ на который необходимо внедрить в текстовку на форме. Форма диалога изображена на рисунке 2.13.

Вставка значения, отображаемого в поле кода в описание страницы, или же текст вопроса приведет к отображению ответа в тексте в процессе заполнения опросника. Данный функционал особенно полезен при формировании страницы с результатом, так как отображение пользовательских ответов на этой страницы

позволяет повысить интерактивность взаимодействия с конечным пользователем системы.

**Pipe chooser**

To pipe the response of an item, copy its piping code and insert it into any text

**Item:**  
Please select...

**Pipe Id:**  
\_\_\_\_\_

**Code:**  
[pipe:]

Close

Рисунок 6.13 – Форма генерации кода инъекции

Последняя вкладка предназначена для управления результатами заполнения опросника. На данной вкладке также расположена кнопка экспорта результатов заполнения опросника.

**Forms**  
Enter details of the form below, create pages and rules

Insert pipe Save changes Close

Main details Pages Rules Result page **Results**

Export

Action	Form result name	Submitted / Saved date
✕ ↻	Test form	2017-05-13T14:16:03.54Z
✕ ↻	Test form	2017-05-13T14:16:18.283Z

1 of 1 pages Go

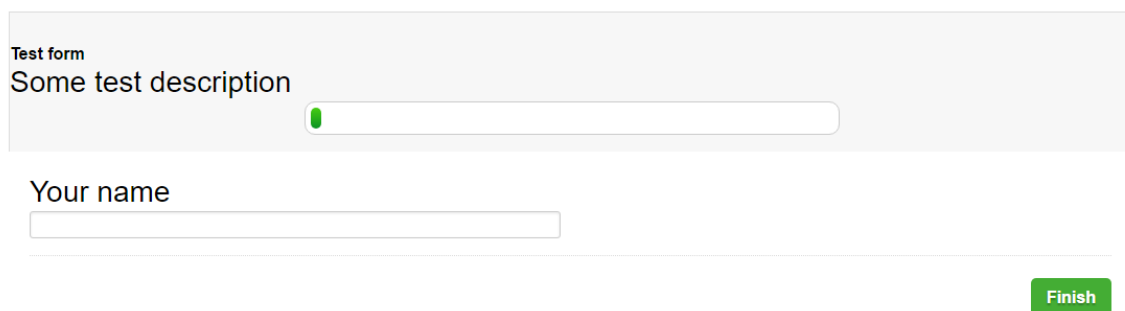
Рисунок 6.14 – Списковая форма результатов заполнения

Нажатие на кнопку «Export» приводит к открытию новой вкладки с загрузкой файла с результатами заполнения. Экспортированные результаты содержат все введенные пользователями ответы на все вопросы.

Администратор формы со списковой формы результатов заполнения может осуществлять удаление результатов заполнения, а также просмотр результирующей страницы, для конкретного результата.

#### 6.2.6 Заполнение форм

Для перевода приложения в режим заполнения формы достаточно перейти по соответствующей ссылке. Распространение формы осуществляется через размещение данной ссылки на страницах в сети интернет. Пример страницы заполнения формы размещен на рисунке 6.15.

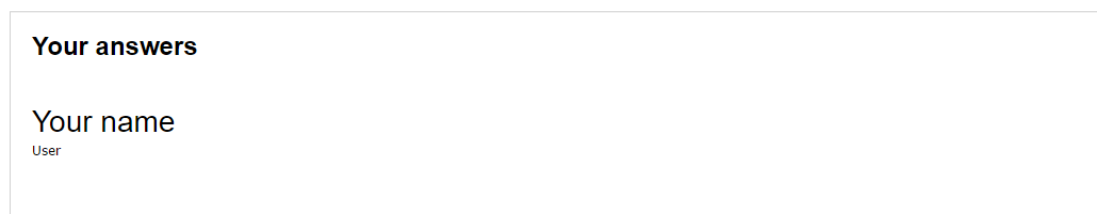


The screenshot shows a web form titled "Test form" with the subtitle "Some test description". Below the title is a progress bar with a small green segment on the left. Underneath the progress bar is a text input field labeled "Your name". At the bottom right of the form is a green button labeled "Finish".

Рисунок 6.15 – Опросник в режиме заполнения

Переход между страницами в режиме заполнения осуществляется посредством нажатия на кнопки «Next», «Previous», «Finish». Нажатие этих кнопок приводит к запуску соответствующих проверок на корректность введенных пользователем данных. Переход осуществляется, только если были введены корректные данные.

Нажатие на кнопку «Finish» приводит к сохранению результатов заполнения и отображению результирующей страницы (рисунок 6.16).



The screenshot shows a results page titled "Your answers". Below the title, it displays "Your name" followed by a small "User" label, indicating the user's identity.

Рисунок 6.16 – Страница результатов заполнения формы

## **7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

### **7.1 Характеристика программного обеспечения разработки форм опросников**

Разрабатываемое в дипломном проекте программное средство разработки форм опросников предназначено для применения широким кругом пользователей и позволяет автоматизировать процесс проведения социальных исследований методом опроса. Данное программное средство предназначено упростить процессы распространения и сбора результатов анкеты, которые на текущий момент зачастую выполняются вручную. Использование программного средства позволяет снизить затраты на привлечение специалистов для проведения опросов, а также на процессы распространения анкеты и сбора результатов заполнения.

Программное средство разрабатывается для свободной реализации на рынке информационных технологий [12].

### **7.2 Расчет затрат на разработку программного средства**

Затраты на разработку программного средства включают в себя следующие статьи:

- затраты на основную заработную плату разработчиков;
- затраты на дополнительную заработную плату разработчиков;
- отчисления на социальные нужды;
- прочие затраты (амортизация оборудования, расходы на электроэнергию, командировочные расходы, накладные расходы и т.п.)ц

#### **7.2.1 Расчет затрат на основную заработную плату разработчиков**

Затраты на основную заработную плату определяются составом команды, которая занимается разработкой программного средства, месячным окладом специалистов и трудоемкостью процесса разработки и рассчитываются по формуле:

$$З_0 = \sum_{i=1}^n T_{чi} * t_i, \quad (7.1)$$

где  $n$  – количество исполнителей, занятых разработкой конкретного ПО;  
 $T_{чi}$  – часовая заработная плата  $i$ -го исполнителя, руб.;  
 $t_i$  – трудоемкость работ, выполняемых  $i$ -м исполнителем, ч.

Для реализации программного средства разработки форм опросников была выбрана команда разработчиков в составе двух инженеров-программистов. Причиной этого является то, что конечный продукт должен состоять из двух частей: веб-приложения, непосредственно с которым будут работать пользователи, а также серверная часть на которой осуществляется долговременное хранение результатов работы с программным средством. Является целесообразным вести параллельную разработку обеих частей программного средства и поручить разработку пользовательского приложения веб-программисту, а вторую часть отдать на выполнение специалисту в области разработки серверных решений. Такое разделение позволит закончить проект вовремя с учетом рисков, связанных с разработкой, и выполнить его качественнее благодаря специализации разработчиков.

В качестве размера месячной тарифной ставки 1-го разряда для расчетов заработной платы выбирается значение, принятое в организации, которая занимается разработкой проекта, и равное 300 руб.

Среднемесячное количество рабочих дней при пятидневной рабочей неделе в 2017 году составляет 21,1 дн.

В таблице 7.1 сведены данные о команде разработчиков, их окладе и назначенном объеме работ для каждого.

Таблица 7.1 – Расчет затрат на основную заработную плату команды разработчиков

№	Участник команды	Разряд	Тариф-ный коэффициент	Месячный оклад, руб	Дневной оклад, руб	Трудоемкость работ, чел./дн.	Основная заработная плата, руб
1	Инженер-программист	12	2,84	852	40,37	5	201,85
2	Инженер-программист	13	3,04	912	43,23	20	864,6
Премия (100% от основной заработной платы)							1066,45
Итого затраты на основную заработную плату разработчиков							2132,9

### 7.2.2 Расчет затрат на дополнительную заработную плату

Дополнительная заработная плата исполнителей проекта  $Z_o$ , руб., определяется по формуле:

$$Z_o = \frac{Z_o \cdot H_o}{100}, \quad (7.2)$$

где  $H_o$  – норматив дополнительной заработной платы (20%).

Дополнительная заработная плата составит:

$$Z_o = 2132,9 \cdot 20 / 100 = 426,58 \text{ руб.}$$

### 7.2.3 Расчет отчислений на социальные нужды

Отчисления в фонд социальной защиты населения и на обязательное страхование  $Z_{сз}$ , руб., определяются в соответствии с действующими законодательными актами по формуле

$$Z_{сз} = \frac{(Z_o + Z_o) \cdot H_{сз}}{100}, \quad (7.3)$$

где  $H_{сз}$  – норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34,6 %).

Отчисления в фонд социальной защиты населения и на обязательное страхование составят:

$$Z_{сз} = (2132,9 + 426,58) \cdot 34,6 / 100 = 885,58 \text{ руб.}$$

### 7.2.4 Расчет прочих затрат

Прочие затраты включают затраты, связанные с разработкой конкретного программного обеспечения напрямую, а также связанные с функционированием организации-разработчика в целом. Расчет прочих затрат выполняется в процентах от затрат на основную заработную плату команды разработчиков с учетом премии по формуле:

$$Z_{пз} = \frac{Z_o \cdot H_{пз}}{100}, \quad (7.4)$$

где  $H_{пз}$  – норматив прочих затрат (100–150%).



Примем  $H_{пз}=120\%$  и рассчитаем сумму прочих затрат:

$$З_{пз} = 2132,9 \cdot 120/100 = 2559,48 \text{ руб.}$$

Полная сумма затрат на разработку программного обеспечения находится путем суммирования всех рассчитанных статей затрат. Расчет приведен в таблице 7.2.

Исходя из вышесказанного, таблица приняла следующий вид:

Таблица 7.2 – Затраты на разработку программного обеспечения

Статья затрат	Сумма, руб.
Основная заработная плата команды разработчиков	2132,9
Дополнительная заработная плата команды разработчиков	426,58
Отчисления на социальные нужды	885,58
Прочие затраты	2559,48
Общая сумма затрат на разработку	6004,54

Рассчитанное значение полной себестоимости, которая составила 6004,54 руб., будет использоваться в дальнейшем, для определения цены ПС.

### 7.3 Оценка экономического эффекта у разработчика программного средства

#### 7.3.1 Экономический эффект у разработчика

Экономический эффект организации-разработчика программного обеспечения в данном случае заключается в получении прибыли от его продажи множеству потребителей. Прибыль от реализации в данном случае напрямую зависит от объемов продаж, цены реализации и затрат на разработку данного ПО.

Цена формируется на рынке под воздействием спроса и предложения. Тогда расчет прибыли от продажи одной копии (лицензии) ПО осуществляется по формуле:

$$П_{ед} = Ц - \frac{З_p}{N} - \text{НДС}, \quad (7.5)$$

где,  $Ц$  – цена реализации одной копии (лицензии) ПО (руб.);

$З_p$  – сумма расходов на разработку и реализацию ПО;

$N$  – количество копий (лицензий) ПО, которое будет куплено клиентами за год;

$\Pi_{\text{ед}}$  – прибыль, получаемая организацией-разработчиком от реализации одной копии программного продукта (руб.);

НДС – сумма налога на добавленную стоимость (руб.).

Проанализировав схожие программные решения на рынке [13-14], была получена средняя стоимость одной лицензии в размере 20 руб., количество реализуемых в год копий 500 в 2017, 250 – в 2018, 150 – в 2019, 2500 – в 2020. Общее количество реализованных копий равно 1000.

Сумма налога на добавленную стоимость в таком случае рассчитывается по формуле 6.6.

$$\text{НДС} = \frac{Ц * \% \text{НДС}}{100\% + \% \text{НДС}} \quad (7.6)$$

где НДС – ставка налога на добавленную стоимость, (20%).

$$\text{НДС} = \frac{20 * 20\%}{100\% + 20\%} = 3,33 \text{ руб.},$$

$$\Pi_{\text{ед}} = 20 - \frac{6004,54}{1000} - 3,33 = 10,66 \text{ руб}$$

Суммарная годовая прибыль по проекту в целом будет равна:

$$\Pi = \Pi_{\text{ед}} * N, \quad (7.7)$$

$$\Pi = 10,66 * 1000 = 10660$$

Далее рассчитываем рентабельность затрат на разработку ПО по следующей формуле:

$$P = \frac{\Pi}{Z_p} * 100\%, \quad (7.8)$$

$$P = \frac{10660}{6004,54} * 100\% = 177,53$$

Для расчета чистой прибыли воспользуемся формулой 6.10.

$$\text{ЧП} = \Pi - \frac{\Pi * H_{\text{приб}}}{100\%}, \quad (7.9)$$

где  $H_{\text{приб}}$  – ставка налога на прибыль, %.

$$\text{ЧП} = 10660 - \frac{10660 * 18\%}{100\%} = 8741,20$$

Таким образом, при реализации 1000 копий программного продукта в течение 4 лет по цене 20 руб. разработчик получит экономический эффект в размере 8741,20 руб. Рентабельность затрат на разработку программного средства равна 177,53 %.

### 7.3.2 Оценка эффекта у пользователя

Эффект от использования программного средства не связан напрямую с экономическими результатами деятельности компании. Данный показатель отражает опосредованное влияние на показатели деятельности пользователя.

Внедрение разработанного программного средства у пользователя приведет к единовременным затратам на покупку копии программного средства в размере 20 руб., а также к возникновению дополнительных постоянных затрат связанных с оплатой услуг доступа к сети Интернет.

Данное программное средство в первую очередь ориентировано на корпоративного пользователя. Использование разрабатываемого программного решения позволяет, в первую очередь, снизить временные затраты на проведение социальных исследований. Автоматизация процессов распространения, сбора и анализа результатов заполнения форм опросников дают ощутимое снижение затрачиваемого времени. Кроме этого, автоматизация рутинных процессов, требующих внимания позволяет облегчить работу соответствующих специалистов.

Несмотря на то, что основная аудитория пользователей программного средства представляет собой специалистов в сфере проведения социальных исследований, однако само программное средство разрабатывалось с целью предоставления предсказуемого интерфейса, что в свою очередь снижает требования к навыкам пользователя, необходимым для работы с программным средством.

Помимо этого, внедрение программного средства в работу компании позволяет ускорить сбор данных от пользователей компании, что позволяет повысить эффективность управления.

## ЗАКЛЮЧЕНИЕ

В ходе дипломного проектирования проведен анализ литературы по теме дипломного проекта, изучены наиболее известные программные решения для конструирования форм опросников и проведения онлайн-исследований. В процессе работы были исследованы современные тенденции в области проведения онлайн-исследований. Для каждого из упомянутых в отчете программных средств выявлены достоинства и недостатки использования, которые учитывались при разработке спецификации требований к программному обеспечению.

Также в ходе моделирования предметной области была разработана функциональная и информационная модели программного обеспечения. На основе разработанных моделей сформулирована функциональная спецификация к программному обеспечению.

Исходя из полученных на этапе моделирования данных, была спроектирована архитектура программного решения, которая смогла бы позволить точно реализовать требования, указанные в функциональной спецификации. Также на данном этапе были определены архитектурные принципы, в соответствии с которыми должна производиться разработка программного решения.

На основании информационной модели была спроектирована модель базы данных. Данная модель призвана покрыть все необходимые аспекты, связанные с хранением сущностей, используемых внутри системы.

Большую часть времени дипломного проектирования заняло моделирование программной области и проектирование архитектуры программного средства.

Таким образом, задачи, поставленные в рамках индивидуального задания, были выполнены. Знания и опыт полученные в процессе прохождения дипломного проектирования будут полезны при дальнейшей работе по специальности.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

- [1] Dillman, Don A., Smyth, Jolene D., Christian, Leah Melani. 2014. Internet, Phone, Mail and Mixed-Mode Surveys: The Tailored Design Method, 4th edition. John Wiley: Hoboken, NJ
- [2] Lord, F. and Novick, M. R.(1968). Statistical theories of mental test scores. Addison – Wesley.
- [3] Heise, D. R.(1969). Separating reliability and stability in test-retest correlation. American Sociological Review, 34, 93-101.
- [4] Andrews, F. M. (1984). Construct validity and error components of survey measures: a structural modelling approach. Public Opinion Quarterly, 48, 409-442.
- [5] Saris, W. E. and Gallhofer, I. N. (2014). Design, evaluation and analysis of questionnaires for survey research. Second Edition. Hoboken, Wiley.
- [6] Timothy R. Graeff, 2005. "Response Bias," Encyclopedia of Social Measurement, pp. 411-418. ScienceDirect.
- [7] ГОСТ 34.003-90. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения. // Информационная технология. Комплекс стандартов и руководящих документов на автоматизированные системы. – М.: Изд-во стандартов, 1991.
- [8] NoSQL – Wikipedia, the free encyclopedia [Электронный ресурс]. – 2016 – Режим доступа: <https://ru.wikipedia.org/wiki/NoSQL> – Дата доступа: 20.03.2016
- [9] Реляционные базы данных обречены? [Электронный ресурс]. – 2016 – Режим доступа: [habrahabr.ru/post/103021/](http://habrahabr.ru/post/103021/)
- [10] Хассан, Г. UML-проектирование систем реального времени параллельных и распределенных приложений / Г. Хассан – М.: ДМК Пресс, 2011 – 704с.
- [11] Дипломные проекты (работы) общие требования СТП 01–2010 [Электронный ресурс]: стандарт предприятия / БГУИР – Электронные данные. – Режим доступа: СТП П2010 бгуир.pdf
- [12] Техничко-экономическое обоснование дипломных проектов: Методическое пособие для студентов всех специальностей БГУИР. В 4-ч. Ч. 4: Проекты программного обеспечения / В.А. Палицын. – Минск: БГУИР, 2006 г. – 76 с.
- [13] Formsit - Pricing [Электронный ресурс]. – 2017 – Режим доступа: <https://www.formsite.com/pricing.html/>
- [14] Survey Monkey - Pricing [Электронный ресурс]. – 2017 – Режим доступа: [https://www.surveymonkey.com/pricing/upgrade/?ut\\_source=header](https://www.surveymonkey.com/pricing/upgrade/?ut_source=header)

## ПРИЛОЖЕНИЕ А (обязательное)

### Текст программного средства

```
var React = require('react');
var assign = require('object-assign');
var Section = require('./Section.jsx');
var Finish = require('./Finish.jsx');
var FormDataStore = require('../data/FormDataStore');
var FormMetadata = require('../data/FormMetadata');
var NavigationLogic = require('../logic/NavigationLogic');
var      NotificationsLogic      =      re-
quire('../logic/NotificationsLogic');
var Config = require('../Config');
var ActionButton = require('../ui/buttons/ActionButton.jsx');
var      GlobalErrorMessage      =      re-
quire('../ui/controls/GlobalErrorMessage.jsx');
var Constants = require('../editor/data/Constants');
var Gen = require('../utils/Gen');
var localStorage = require('../utils/LocalStorage');
var ScoringCalculation = require('../data/ScoringCalculation');
var PipingLogic = require('../data/PipingLogic');
var      NotificationPopup      =      re-
quire('../controls/NotificationPopup.jsx');
import ProgressBar from '../controls/ProgressBar';

var Mode = {
  Index: 'Index',
  SuccessfullySaved: 'SuccessfullySaved',
  SavingFailed: 'SavingFailed'
};

var Form = React.createClass({

  propTypes: {
    isPrintView: React.PropTypes.bool
  },

  getDefaultProps() {
    return {
      isPrintView: false
    }
  },

  getInitialState: function () {
```

```

    var isFinish = false;
    if (this.props.isFinish !== undefined) {
        isFinish = this.props.isFinish;
    }

    var {formSectionId} = this.props;
    var navigationHistory = [];
    if (formSectionId){
        var currentSection =
this.props.metadata.Sections.find(
        (section) => (section.Id == formSectionId)
    );

        if (formSectionId !=
this.props.metadata.Sections[0].Id){

this.props.metadata.Sections.every(function(section){
            if (section.Id == formSectionId)
            {
                return false;
            }
            navigationHistory.push(section);
        });
    }
    var state = {
        currentSection: currentSection ||
this.props.metadata.Sections[0],
        isFinish: isFinish,
        sessionId: null,
        navigationHistory: navigationHistory,
        formKey: Gen.elementId(),
        mode: Mode.Index,
        isProductForm: Config.getConfig().finishAction !=
'' && Config.getConfig().finishAction != undefined
    };
    if (this.props.tempFormId) {
        var tempState =
this.processTempFormResultData(Constants.LocalStorageKeys.FillFormD
ata);
        state = assign({}, state, tempState);
    } else if
(localStorage.getItem(Constants.LocalStorageKeys.FillFormNavigation
Data)){
        var navigationState =
this.processTempFormResultData(Constants.LocalStorageKeys.FillFormN
avigationData);

```

```

        state = assign({}, state, navigationState);
    }
    return state;
},

componentWillMount() {
    if
    (!localStorage.getItem(Constants.LocalStorageKeys.FillFormData)) {
        return;
    }
    if
    (Config.getConfig().isAuthenticated    &&
    !this.props.tempFormId
    && Config.getConfig().mode    !=    Con-
    stants.Mode.NavigationView) {
        this.saveFormResult();
    }
},

processTempFormResultData: function(localStorageKey) {
    let                formResult                =
JSON.parse(localStorage.getItem(localStorageKey));
    localStorage.removeItem(localStorageKey);
    if(formResult.FormResultInfo.DataList) {

FormDataStore.convertFromResultDataToStore(formResult.FormResultInf
o.DataList);
    }
    var state = {
        navigationHistory: formResult.NavigationHistory,
        isFillForm: true,
        formKey: Gen.elementId(),
        mode: Mode.Index,
        isFinish: formResult.IsFinish || false
    };

    if(formResult.CurrentSection) {
        state = assign({}, state, {currentSection:
formResult.CurrentSection});
    }

    if(formResult.Mode) {
        Config.setConfig({mode: formResult.Mode});
    }

    return state;
},

```



```

        componentDidUpdate: function() {
            if(this.state.editQuestionId !== undefined){
                this.setState({
                    editQuestionId: undefined
                });
                document.getElementById(this.state.editQuestionId).scrollIntoView(true)
            }
        },

        onEditQuestion: function (section, navigationHistory,
            questionMetadataId) {
            this.setState({
                formKey: Gen.elementId(),
                isFinish: false,
                currentSection: section,
                navigationHistory: navigationHistory,
                editQuestionId: questionMetadataId
            });
        },

        prevButtonClick: function () {
            this.scrollToTop();
            var navigationHistory = this.state.navigationHistory;
            if (navigationHistory.length < 1)
                return;

            var prevSection = navigationHistory.pop();
            this.setState({currentSection: prevSection,
                navigationHistory: navigationHistory});
        },

        nextButtonClick: function () {
            var isValid = FormDataStore.isValid();
            if (isValid) {
                this.scrollToTop();
                var currentSection = this.state.currentSection;
                this.state.navigationHistory.push(currentSection);
                var nextSectionId =
                    NavigationLogic.getNextSectionId(this.props.metadata,
                        currentSection.Id);
                var nextSection =
                    this.props.metadata.Sections.find(section => section.Id ==
                        nextSectionId);
                if (!nextSection) {
                    this.setState({resultPage: nextSectionId});
                }
            }
        }
    }
}

```

```

        this.finishButtonClick(true);
    } else {
        this.setState({isFinish: false, currentSection:
nextSection});
    }
    }else{
        this.scrollFirstError();
    }
},

    handleGoToSection: function (nextSectionId, openInNewTab) {

        var isValid = FormDataStore.isValid();
        if (isValid) {
            var nextSection =
this.props.metadata.Sections.find(section => section.Id ==
nextSectionId);
            if (openInNewTab){

                localStorage.setItem(Constants.LocalStorageKeys.FillFormNavigationD
ata,
                JSON.stringify(this.getFillFormDataForNewTabNavigation(nextSection
)));
                var cs = this.state.currentSection;
                var finish = this.state.isFinish;
                this.setState({isFinish: false, currentSection:
nextSection});
                window.open(window.location.href, "_blank");
                this.setState({isFinish: finish,
currentSection: cs})
            } else {
                this.scrollTop();
                var currentSection = this.state.currentSection;

                this.state.navigationHistory.push(currentSection);

                if (!nextSection) {
                    this.setState({resultPage: nextSectionId});
                    this.finishButtonClick(true);
                } else {
                    this.setState({isFinish: false,
currentSection: nextSection});
                }
            }
        }else{
            this.scrollFirstError();
        }
    }
}

```

```

    },

    onSaveSuccess: function() {
        let currentSection = this.state.currentSection;
        let nextSectionId =
NavigationLogic.getNextSectionId(this.props.metadata,
currentSection.Id);
        let nextSection =
this.props.metadata.Sections.find(section => section.Id ==
nextSectionId);
        if (!nextSection) {
            this.setState({resultPage: nextSectionId});
        }
        let resultPage = this.getResultPage();
        let mode = Config.getConfig().mode;
        let totalScore =
ScoringCalculation.scoringTotalCalculation();
        if (resultPage &&
            (resultPage.Type == Constants.FormResultTypes.redirectUrl
            || resultPage.Type == Constants.FormResultTypes.customWithRedirectUrl
            && mode.toLowerCase() != Constants.Mode.Edit.toLowerCase()
            && mode.toLowerCase() != Constants.Mode.EditViaAssessment.toLowerCase())) {
            window.location.href = resultPage.RedirectUrl;
        } else {
            if (resultPage && (!mode || mode.toLowerCase() !=
Constants.Mode.Edit.toLowerCase() && mode.toLowerCase() !=
Constants.Mode.EditViaAssessment.toLowerCase())) {
                {
                    Config.updateConfig({closeUrl:
resultPage.RedirectUrl || Config.getConfig().closeUrl});
                }
                this.setState({isFinish: true, totalScore:
totalScore});
            }
        }
    },

    finishButtonClick: function (isSavingNecessary) {
        var isValid = FormDataStore.isValid();
        if (isValid) {
            this.scrollTop();
            var isResultPageAllowed =
Config.getConfig().finishAction == '' ||
Config.getConfig().finishAction == undefined;

```

```

        if(isResultPageAllowed) {
            if (isSavingNecessary) {
                this.submitFormData(true,          ()          =>
this.onSaveSuccess());
            }else{
                this.onSaveSuccess();
            }
        }else{
            this.submitFormData(true);
        }
    }else{
        this.scrollFirstError();
    }
},

scrollTop: function () {
    $("html, body").animate({ scrollTop: 0 });
},
scrollFirstError: function(){
    setTimeout( function() {
        var          element          =$("span.field-validation-
error:first");
        var topOffset = element.length ? $("span.field-
validation-error:first").parent().parent().parent().offset().top:
0;
        $("html, body").animate({scrollTop: topOffset});
    }, 100);
},

submitFormData: function (isFinish, cb) {
    var that = this;
    var          dataForSave          =
FormDataStore.prepareDataStoreForSave(NavigationLogic);
    var formResultId = Config.getConfig().formResultId;
    var version = Config.getConfig().formResultVersion ||
0;

    var url = Config.getSaveFormResultsUrl();
    if (formResultId || that.state.sessionId != null) {
        url = Config.getEditFormResultsUrl();
    }

    if (Config.getConfig().mode != Constants.Mode.TestRun)
{
        var totalCalculation = [];
        Ob-
ject.keys(FormMetadata.TotalCalculation).forEach(function(i) {

```

```

        totalCalculation.push({ Id:i, TotalValue:
ScoringCalculation.getCalculationQuestionValue(i)});
    });
    var data = {
        FormMetadataId: Config.getFormMetadataId(),
        DataList: dataForSave,
        FormResultInfoId: formResultId,
        CurrentSectionId: that.state.currentSection.Id,
        IsFormResultCompleted: isFinish,
        SessionId: that.state.sessionId,
        IsFillForm: that.state.isFillForm,
        TotalCalculation: totalCalculation,
        IsTemporary: Config.getConfig().isTemp,
        Version: version,
        NotificationsIdsToSend:
NotificationsLogic.getNotificationsIdsToSend(this.props.metadata.No
tifications, this.props.metadata),
        NotificationsIdsToRemind:
NotificationsLogic.getNotificationsIdsToRemind(this.props.metadata.
Notifications, this.props.metadata),
        ResultPageIsAvaibleForUnauthorizedUser:
this.getResultPage().ResultPageIsAvaibleForUnauthorizedUser
    };

    var jqxhr = $.ajax({ url: url, type: 'POST', data:
JSON.stringify(data), dataType: "json", contentType: "applica-
tion/json",
        success: function (result) {
            console.log('Success saving form data');
            if(!data.IsFormResultCompleted){
                that.setState({mode:
Mode.SuccessfullySaved });
            }
            if(!data.IsFormResultCompleted ||
Config.getConfig().finishAction == '' ||
Config.getConfig().finishAction == undefined){
                if (result != '') {
                    if (result.sessionId != null) {
                        that.setState({sessionId: re-
sult.sessionId});
                    }
                    if (result.formResultId != null) {
                        Config.updateConfig({formResultId: result.formResultId});
                    }
                    if (result.formResultVersion !=
null) {

```

```

Config.updateConfig({formResultVersion: result.formResultVersion});
    }
    if (result.isFillForm != undefined)
{
        that.setState({isFillForm:
true});
    }
    }
    that.setState({globalErrorMessages:
null});
    }else{
        window.location.href
Config.getConfig().finishAction + '?formResultInfoId=' + re-
sult.formResultId;
    }
    if(cb){
        cb();
    }
    }
});
jqxhr.fail((jqXHR, textStatus, errorThrown) => {
    if (jqXHR.status == 403) {
        var shouldAuthenticate =
jqXHR.getResponseHeader('x-s4s-should-authenticate') == "true";
        if (shouldAuthenticate) {

localStorage.setItem(Constants.LocalStorageKeys.FillFormData,
JSON.stringify(this.getFillFormData()));

that.saveSaveFormResultToSession(Config.getAuthUrl());
        } else {
            location.href
Config.getNoPermissionUrl();
            that.setState({loadError: true,
errorMessage: errorThrown});
        }
    }else if(jqXHR.status === 500){
        that.setState({globalErrorMessages: ['This
object has not been found as it has been modified by another us-
er.'],mode: Mode.SavingFailed});
    }
    else{
        console.error('Fail saving form data',
textStatus, errorThrown, jqXHR.responseText);
        that.setState({globalErrorMessages:
[jqXHR.responseText], mode: Mode.SavingFailed});
    }
}

```

```

        }
    });
},

addToFavorite: function() {
    var data = this.getFillFormData();
    var url = Config.getAddToFavoriteFormResultUrl();
    var jqxhr = $.post(url, data);

    jqxhr.fail((jqXHR, textStatus, errorThrown) => {
        console.error('Fail saving form data', textStatus,
errorThrown, jqXHR.responseText);
        this.setState({globalErrorMessages:
[jqXHR.responseText]});
    });

    return jqxhr;
},
loadFormResult: function (formResult, mode) {
    this.setState({
        currentSection: formResult.CurrentSection,
        navigationHistory: formResult.NavigationHistory,
        isFillForm: true,
        formKey: Gen.elementId(),
        mode: mode
    });
},
saveFormResult: function () {
    var that = this;
    var formResult =
JSON.parse(localStorage.getItem(Constants.LocalStorageKeys.FillForm
Data));
    var formResultId =
formResult.FormResultInfo.FormResultInfoId ||
formResult.FormResultInfo.Id;
    var version = formResult.FormResultInfo.Version;
    var url = Config.getSaveFormResultsUrl();
    var data = {
        FormMetadataId: Config.getFormMetadataId(),
        DataList: formResult.FormResultInfo.DataList,
        NotificationsIdsToSend:
formResult.FormResultInfo.NotificationsIdsToSend,
        NotificationsIdsToRemind:
formResult.FormResultInfo.NotificationsIdsToRemind,
        ResultPageIsAvaibleForUnauthorizedUser:
this.getResultPage().ResultPageIsAvaibleForUnauthorizedUser,

```

```

        CurrentSectionId: formResult.CurrentSection.Id
    };
    if (formResultId) {
        url = Config.getEditFormResultsUrl();
        data.isFillForm = true;
        data.FormResultInfoId = formResultId;
        data.Version = version;
    }
    var jqxhr = $.ajax({ url: url, type: 'POST', data:
JSON.stringify(data), dataType: "json", contentType: "applica-
tion/json",
        success: function (result) {
            if (result != '') {
                if (result.formResultId != null) {
                    Config.updateConfig({formResultId: re-
sult.formResultId});
                }
                if (result.formResultVersion != null) {
                    Config.updateConfig({formResultVersion:
result.formResultVersion});
                }
            }

            localStorage.removeItem(Constants.LocalStorageKeys.FillFormData);

            FormDataStore.loadFormResult(result.formResultId,           re-
sult.unauthorizedAccessKey, null, () => {
                console.log('Success      saving      form
data');
                that.loadFormResult(formResult,
Mode.SuccessfullySaved);
            }, (jqXHR) => {
                that.ajaxErrorHandler(jqXHR);
            });
        }
    });

    jqxhr.fail((jqXHR, textStatus, errorThrown) => {
        console.error('Fail saving form data', textStatus,
errorThrown, jqXHR.responseText);

        localStorage.removeItem(Constants.LocalStorageKeys.FillFormData);
        that.setState({globalErrorMessages:
[jqXHR.responseText]});
    });
},

```



```

        getResultPage: function () {
            if(this.state != null && this.state.resultPage != unde-
fined) {
                var resultPage =
this.props.metadata.FormResults.find(r => r.Id ==
this.state.resultPage);
                return resultPage;
            }

            var matchedRule = this.findMatchingResultPageRule();

            var resultPage = matchedRule ?
                this.props.metadata.FormResults.find(r => r.Id ==
matchedRule.FormResultId)
                : this.props.metadata.FormResults.find(r =>
r.IsDefault);

            return resultPage;
        },

        findMatchingResultPageRule: function() {
            var matchedRule = null;
            if (this.props.metadata.FormResultRules.length) {
                var scoringTotal =
parseFloat(this.state.totalScore);
                matchedRule =
this.props.metadata.FormResultRules.find(rule => {
                    if (!rule.CriteriaConditions ||
!rule.CriteriaConditions.length)
                        return false;

                    var isAnd = rule.CriteriaConditions.GroupOperator ==
Constants.GroupOperator.And;
                    var aggregateFunction = isAnd ? Ar-
ray.prototype.every : Array.prototype.some;
                    var result =
aggregateFunction.call(rule.CriteriaConditions, condition => {
                        var parsedValue =
parseFloat(condition.Value);
                        switch (condition.Operator) {
                            case Constants.OperationTypes.is:
                                if (parsedValue == scoringTotal) {
                                    return true;
                                }
                                break;
                            case Constants.OperationTypes.isNot:
                                if (parsedValue != scoringTotal) {

```

```

        return true;
    }
    break;
    case Constants.OperationTypes.greaterThan:
        if (parsedValue < scoringTotal) {
            return true;
        }
        break;
    case Constants.OperationTypes.lessThan:
        if (parsedValue > scoringTotal) {
            return true;
        }
        break;
    default:
        break;
    }
    return false;
});
return result;
});
}
return matchedRule;
},

ajaxErrorCommonHandler: function (jqXHR) {
    if (jqXHR.status == 403) {
        var shouldAuthenticate =
jqXHR.getResponseHeader('x-s4s-should-authenticate') == "true";
        if (shouldAuthenticate) {
            window.location.href = Config.getAuthUrl();
        } else {
            window.location.href =
Config.getNoPermissionUrl();
        }
    }
    else if (jqXHR.status == 404) {
        window.location.href = Config.getPageNotFoundUrl();
    }
},

onSaveClick: function () {
    var isValid = FormDataStore.isValid(true);
    if (isValid) {
        this.scrollTop();
        if (Config.getConfig().isAuthenticated == undefined
|| Config.getConfig().isAuthenticated) {

```

```

        this.submitFormData(false);
    } else {

localStorage.setItem(Constants.LocalStorageKeys.FillFormData,
JSON.stringify(this.getFillFormData()));

this.saveSaveFormResultToSession(Config.getAuthUrl());
    }
    }else{
        this.scrollFirstError();
    }
},

saveSaveFormResultToSession: function (redirectUrl) {
    var data = this.getFillFormData();
    var url = Config.getSaveFormResultToSessionUrl();
    var jqxhr = $.post(url, data, () => {
        window.location.href = redirectUrl;
    });

    jqxhr.fail((jqXHR, textStatus, errorThrown) => {
        console.error('Fail saving form data', textStatus,
errorThrown, jqXHR.responseText);
        this.setState({globalErrorMessages:
[jqXHR.responseText]});
    });
},

getFillFormDataForNewTabNavigation: function(nextSection){
    var data = this.getFillFormData();

    delete data.FormResultInfo.FormResultInfoId;
    delete data.FormResultInfo.Version;
    delete data.FormResultInfo.FormMetadataId;

    data.FormResultInfo.DataList
FormDataStore.prepareDataStoreForSave(NavigationLogic,
nextSection);
    data.CurrentSection = nextSection;
    data.Mode = Constants.Mode.NavigationView;
    data.metadata = this.props.metadata;

    return data;
},

getFillFormData: function () {
    return {

```

```

        FormResultInfo: {
            FormResultInfoId:
Config.getConfig().formResultId,
            Version: Config.getConfig().formResultVersion,
            FormMetadataId: Config.getConfig().formId,
            DataList:
FormDataStore.prepareDataStoreForSave(NavigationLogic)
        },
        CurrentSection: this.state.currentSection,
        NavigationHistory: this.state.navigationHistory,
        IsFinish: this.state.isFinish,
        PageName:
this.state.currentSection.DisplayPageDetailsOnTheForm      &&
!this.state.isFinish ? this.state.currentSection.SectionName :
this.props.metadata.Name
    };
},

onCloseButtonClick: function(){
    var closeBtnUrl = Config.getConfig().appBaseUrl;
    if(Config.getConfig().closeUrl){
        closeBtnUrl = Config.getConfig().closeUrl;
    }
    window.location.href = closeBtnUrl;
},

render: function () {
    var {isPrintView} = this.props;
    var isNavigationView = Config.getConfig().mode == Con-
stants.Mode.NavigationView;
    if (!this.props.metadata.Sections ||
this.props.metadata.Sections.length <= 0) {
        return <div>No content. Form sections collection is
empty</div>
    }
    var currentSection = this.state.currentSection;
    var
        currentSectionIndex
this.props.metadata.Sections.findIndex(s => s.Id ==
currentSection.Id);

    var isEditAllowed = Config.getConfig().IsEditAllowed;
    var isViewMode = Config.getConfig().mode == Con-
stants.Mode.View || Config.getConfig().mode == Con-
stants.Mode.ViewAnswers;
    var isFinish = this.state.isFinish
        || (isEditAllowed !== undefined && !isEditAllowed)
        || (isViewMode !== undefined && isViewMode);

```

```

        var progressBarWidth = isFinish ? (100) :
(currentSectionIndex * 100 / this.props.metadata.Sections.length);
        var progressBarWidthPercent = Math.max(2,
progressBarWidth) + '%';

        var closeBtn = Config.getConfig().mode !== Con-
stants.Mode.TestRun ? <ActionButton label="Close"
onClick={this.onCloseButtonClick}
/> : null;
        var buttons = isFinish ? <ul className="actionButtons
alt" id="buttons">
        {closeBtn}
        <ActionButton label="Print"
onClick={() => {
            var element = $("#app")[0];
            $('#buttons').hide();
            var doc = new jsPDF('p', 'pt', [ele-
ment.clientHeight, element.clientWidth]);
            var name =
`${this.props.metadata.Name}.pdf`;
            doc.addHTML(element, 0, 0, {
                pagesplit: true
            }, function () {
                doc.save(name);
                $('#buttons').show();
            });
        }}
        />
        </ul> : null;

        if(Config.getConfig().mode == Constants.Mode.Edit){
            buttons = <ul className="actionButtons alt"
id="buttons">
                <ActionButton label="Save"
onClick={()=>this.onSaveClick()} />
                {closeBtn}
            </ul>
        }

        var headerImage = null;
        var titleStyles = {clear: 'both'};
        if (!isFinish && currentSection.ImagePath != null &&
currentSection.ImagePath != undefined && currentSection.ImagePath
!= '') {

```

```

        headerImage = <div className="photo-proportions"
style={{
            marginLeft: '90px',
            float: 'left',
            width: '200px',
            height: '200px',
            display: 'inline'
        }}>
            <img id="fImg" src={currentSection.ImagePath}
alt="" style={{maxWidth: '144px', maxHeight: '100px'}}/>
        </div>;
        titleStyles = {marginRight: '290px'};
    }

    let progressBar =
this.props.metadata.IsProgressBarHidden === true ? undefined :
<ProgressBar percent={progressBarWidthPercent}/>;

    var header = Config.getConfig().mode == Constants.Mode.ViewAnswers ? null :
        <div className="s4s-pathway-header-container">
            <div className="mainheader tint3">
                {buttons}
                {headerImage}
                <h1 style={titleStyles}>
                    <div
                        dangerouslySetInnerHTML={{__html:
currentSection.DisplayPageDetailsOnTheForm && !isFinish ?
currentSection.SectionName : this.props.metadata.Name}}></div>
                    </h1>
                    <h2 style={titleStyles}>
                        <div dangerouslySetInnerHTML={{
__html:
currentSection.DisplayPageDetailsOnTheForm && !isFinish ?
currentSection.Description : this.props.metadata.Description || ''
                        }}></div>
                    </h2>
                    {!isPrintView && progressBar}
                </div>
            </div>;

    var saveButton = !isFinish && Config.getConfig().mode
!= Constants.Mode.TestRun
        && (Config.getConfig().mode == Constants.Mode.EditViaAssessment || Config.getConfig().isAuthenticated
|| this.props.metadata.IsSavingForUnauthorizedUsersAllowed)

```

```

        ? <button onClick={()=>this.onSaveClick()}
className='blockBtns small alt textAfterInput'>Save</button> : un-
defined;
        var closeButton = Config.getConfig().mode == Con-
stants.Mode.EditViaAssessment ? <button className='blockBtns small
alt textAfterInput' onClick={() => {window.location.href =
Config.getConfig().closeUrl}}>Close</button> : null;
        var prevButton = this.state.navigationHistory.length >
0 ? <button onClick={this.prevButtonClick} className='blockBtns
small alt textAfterInput' >Previous</button> : undefined;
        var nextButton = currentSectionIndex <
this.props.metadata.Sections.length - 1 ? <button
onClick={this.nextButtonClick} className='blockBtns small alt green
textAfterInput'>Next</button> : undefined;
        var finishButtonLabel = this.state.isProductForm ?
'Next': (FormMetadata.FinishButtonAlternativeName || 'Finish');
        var finishButton = currentSectionIndex ==
this.props.metadata.Sections.length - 1 ?
        <button onClick={()=>this.finishButtonClick(true)}
        className='blockBtns small alt green
textAfterInput'>{finishButtonLabel}</button> : undefined;

        var isFinishWithoutSavingAllowed =
FormMetadata.IsFinishWithoutSavingAllowed === true;
        var finishButtonWithoutSavingLabel =
(FormMetadata.FinishWithoutSavingButtonAlternativeName || 'Finish
(without saving)');
        var finishWithoutSavingButton =
(isFinishWithoutSavingAllowed && currentSectionIndex ==
this.props.metadata.Sections.length - 1) ?
        <button onClick={()=>this.finishButtonClick(false)}
        className='blockBtns small alt green
textAfterInput'>{finishButtonWithoutSavingLabel}</button> : unde-
fined;

        var content = null;
        if (!isFinish) {
            content = <div className="section">
                <div className="pad20">
                    <Section
                        isTextView={isPrintView ||
isNavigationView}
                        displayNavigation={!isPrintView &&
!isNavigationView}
                        sectionMetadata={currentSection}
                        metadata={this.props.metadata}

```

```

goToSectionHandler={this.handleGoToSection}
    />
    {!isPrintView && !isNavigationView &&
      <div className="section">
        <div className="col span_12_of_12">
          <div      className="answer-btns
section rhs">
                                {saveButton}
                                {closeButton}
                                {prevButton}
                                {nextButton}
                                {finishButton}
                                {finishWithoutSavingButton}
          </div>
        </div>
      </div>
    }
  </div>
</div>
} else {
  var resultPage = this.getResultPage();
  content = <Finish
showDetails={this.props.showDetails}
showResult={this.props.showResult}

onEditQuestion={this.onEditQuestion} isViewMode={isViewMode}

navigationHistory={this.state.navigationHistory}
resultPage={resultPage}></Finish>
}

    var notificationPopupMarkup = (this.state.mode ==
Mode.SuccessfullySaved && Config.getConfig().mode != Con-
stants.Mode.Edit || this.state.mode == Mode.SavingFailed)?
    <NotificationPopup      onClose={()=>
this.setState({mode: Mode.Index})} message={this.state.mode ==
Mode.SuccessfullySaved? __localisation.SaveFormSuccessMessage : "The
form saving has been failed."} />: null;

    if(this.state.mode == Mode.SuccessfullySaved &&
Config.getConfig().mode == Constants.Mode.Edit){
      this.onCloseButtonClick();
    }

    return <div key={this.state.formKey}>

```



```

        <GlobalErrorMessage
es={this.state.globalErrorMessages} />
        {notificationPopupMarkup}
        {header}
        {content}
    </div>;
    }
});

module.exports = Form;

var React = window.React = require('react');
var ReactDOM = window.ReactDOM = require('react-dom');
var Form = require("../ui/Form.jsx");
var FormMetadata = require('../data/FormMetadata');
var FormDataStore = require('../data/FormDataStore');
var Config = require('../Config');
var Constants = require('../editor/data/Constants');
var Gen = require('../utils/Gen');
var localStorage = require('../utils/LocalStorage');
var Select = require('../ui/controls/Select.jsx');
var RulesHelper = require('../data/RulesHelper');

var FillForm = React.createClass({
    propTypes: {
        isPrintView: React.PropTypes.bool,
        apiUrl: React.PropTypes.string,
        appBaseUrl: React.PropTypes.string,
        closeUrl: React.PropTypes.string,
        signUpUrl: React.PropTypes.string,
        formId: React.PropTypes.oneOfType([
            React.PropTypes.string,
            React.PropTypes.number
        ]).isRequired,
        formSectionId: React.PropTypes.oneOfType([
            React.PropTypes.string,
            React.PropTypes.number
        ]),
        metadata: React.PropTypes.object,
        mode: React.PropTypes.string,
        unauthorizedAccessKey: React.PropTypes.string
    },

    addToFavorite(){
        return this.refs.Form.addToFavorite();
    },

```

```

    getInitialState: function () {
        var that = this;

        var      apiUrl      =      this.props.apiUrl      ||
$.s4s.globalParams.apiUrl;
        var      appBaseUrl  =      this.props.appBaseUrl  ||
$.s4s.globalParams.appBaseUrl;

        Config.setConfig({
            apiUrl: apiUrl,
            appBaseUrl: appBaseUrl,
            metadata: this.props.metadata,
            formId: this.props.formId,
            formResultId: this.props.formResultId,
            closeUrl: this.props.closeUrl,
            mode: this.props.mode,
            isTemp: this.props.isTemp,
            finishAction: this.props.finishAction,
            isAuthenticated: this.props.isAuthenticated,
            signUpUrl: this.props.signUpUrl,
            unauthorizedAccessKey:
this.props.unauthorizedAccessKey,
        });

        var      isTestRun    =      this.props.mode    ==    Con-
stants.Mode.TestRun;
        var metadata = null;
        var formStyles = null;
        var isFillFormLoaded = false;

        if (isTestRun) {
            var      metadataJson      =
localStorage.getItem(Constants.LocalStorageKeys.TestRunMetadata);
            var      formStylesJson    =
localStorage.getItem(Constants.LocalStorageKeys.TestRunFormStyles);
            metadata = JSON.parse(metadataJson);
            formStyles = JSON.parse(formStylesJson);
        } else if (this.props.metadata) {
            metadata = this.props.metadata;
        }

        if (metadata) {
            FormMetadata.setMetadata(metadata);
            FormMetadata.setFormStyles(formStyles);
            return {      metadata:      FormMetadata,      formKey:
Gen.elementId() };

```

```

        }
        if(this.props.isPrintView){
            var jqxhr =
FormMetadata.loadFormMetadataSync(this.props.formId);
            this.ajaxErrorHandler(jqxhr);
            var data = JSON.parse(jqxhr.responseText);
            metadata = data.FormMetadata;
            FormMetadata.setMetadata(metadata);
            FormMetadata.setFormStyles(data.FormStyles);

        }else {
            FormMetadata.loadFormMetadata(this.props.formId,
function () {
                that.onLoadMetadata();
            }, function (jqXHR, textStatus, errorThrown) {
                that.ajaxErrorHandler(jqXHR);
                that.setState({loadError: true, errorMessage:
errorThrown});
            });
        }

        if(this.props.isFillFormData){
            var url =
Config.getGetFormResultFromStore(this.props.tempFormId);

            var jqxhr = $.ajax({
                type: "GET",
                dataType: 'json',
                url: url,
                async: false
            });

            this.ajaxErrorHandler(jqxhr);
            var data = JSON.parse(jqxhr.responseText);

            localStorage.setItem(Constants.LocalStorageKeys.FillFormData,
JSON.stringify(data));
            isFillFormLoaded = true;

        }

        return {metadata: metadata, formKey: Gen.elementId(),
isFillFormLoaded: isFillFormLoaded};
    },
    onLoadMetadata: function(){
        if (RulesHelper.isAnyRulesForMainForm(FormMetadata)) {
            var url = Config.getGetFormResultOptionsUrl();

```

```

        var params = {formId: this.props.formId};
        var that = this;
        var jqxhr = $.getJSON(url, params, function (data)
{
            var mainFormResultId = null;
            var mainFormResultOptions = da-
ta.MainFormResultOptions;
            if (mainFormResultOptions &&
mainFormResultOptions != null) {
                if (mainFormResultOptions.length == 1) {
                    mainFormResultId =
mainFormResultOptions[0].Value;
                }
                if (mainFormResultOptions.length > 1) {
                    that.setState({
                        mainFormResultOptions:
mainFormResultOptions,
                        formKey: Gen.elementId(),
                        metadata: FormMetadata,
                        mainFormName: data.MainFormName
                    });
                } else {
                    if (mainFormResultId) {
                        FormDataStore.loadMainFormResult(mainFormResultId,
that.props.formId, () => {
                            this.forceUpdate()
                        }, (jqXHR) => {
                            this.ajaxErrorCommonHandler(jqXHR);
                        });
                    } else {
                        RulesHelper.removeUnusedCriteriaas(FormMetadata);
                        that.setState({metadata:
FormMetadata});
                    }
                }
            }
        } else {
            this.setState({metadata: FormMetadata});
        }
        if (this.props.formResultId) {
            FormDataStore.loadFormResult(this.props.formResultId,
this.props.unauthorizedAccessKey,

```

```

        this.props.mode, () => {
            this.forceUpdate()
        }, (jqXHR) => {
            this.ajaxErrorCommonHandler(jqXHR);
        });
    },
    onChange: function (e) {
        FormDataStore.loadMainFormResult(e,    this.props.formId,
    () => {
        this.forceUpdate();
        this.setState({mainFormResultOptions: null});
    }, (jqXHR) => {
        this.ajaxErrorCommonHandler(jqXHR);
    });
    },

    ajaxErrorCommonHandler: function (jqXHR) {
        if (jqXHR.status == 403) {
            var shouldAuthenticate =
jqXHR.getResponseHeader('x-s4s-should-authenticate') == "true";
            if (shouldAuthenticate) {
                window.location.href = Config.getAuthUrl();
            } else {
                window.location.href =
Config.getNoPermissionUrl();
            }
        }
        else if (jqXHR.status == 404) {
            window.location.href = Config.getPageNotFoundUrl();
        }
    },

    render: function () {
        if (this.state.loadError) {
            return <h2>{'Error    has    occurred:    '    +
this.state.errorMessage}</h2>;
        }

        if (!this.state.metadata) {
            return <h2>Form is loading</h2>;
        }

        if (this.state.mainFormResultOptions != null) {
            return <Formsy.Form    key={this.state.formKey}
ref="refForm" mapping={Formsy.Custom.properModelMapper}

```

```

                                style={{textAlign:      'center',
marginTop: 100}}}>
        <h3>Please select the answers of the related
form '{this.state.mainFormName}'.</h3>
        <div style={{marginTop: '2%'}}>
            <Select name="MainFormResultOptions"
                op-
tions={this.state.mainFormResultOptions} /*label="Form results"*/
                value={null}
pleaseSelectOption={true} onChange={this.onChange}/>
            </div>
        </Formsy.Form>;
    }

    var {formSectionId} = this.props;

    var isFormRender = (this.props.isFillFormData) ?
this.state.isFillFormLoaded : true;

    return (
        <div key={this.state.formKey} ref="Form2">
            {isFormRender &&
                <Form
                    tempFormId={this.props.tempFormId}
                    isPrintView={this.props.isPrintView}
                    ref="Form"
                    metadata={this.state.metadata}
                    formSectionId={formSectionId}
                    isFinish={this.props.isFinish}
                    showDetails={this.props.showDetails}
                    showResult={this.state.showResult}
                />
            }
        </div>
    );
}

});

module.exports = FillForm;
var React = require('react');
var assign = require('object-assign');
var Section = require('./Section.jsx');
var Finish = require('./Finish.jsx');
var FormDataStore = require('../data/FormDataStore');
var FormMetadata = require('../data/FormMetadata');
var NavigationLogic = require('../logic/NavigationLogic');

```

```

    var NotificationsLogic = require('.../logic/NotificationsLogic');
    var Config = require('.../Config');
    var ActionButton = require('.../ui/buttons/ActionButton.jsx');
    var GlobalErrorMessage = require('.../ui/controls/GlobalErrorMessage.jsx');
    var Constants = require('.../editor/data/Constants');
    var Gen = require('.../utils/Gen');
    var localStorage = require('.../utils/LocalStorage');
    var ScoringCalculation = require('.../data/ScoringCalculation');
    var PipingLogic = require('.../data/PipingLogic');
    var NotificationPopup = require('.../controls/NotificationPopup.jsx');
    import ProgressBar from '.../controls/ProgressBar';

    var Mode = {
      Index: 'Index',
      SuccessfullySaved: 'SuccessfullySaved',
      SavingFailed: 'SavingFailed'
    };

    var Form = React.createClass({
      propTypes: {
        isPrintView: React.PropTypes.bool
      },
      getDefaultProps() {
        return {
          isPrintView: false
        }
      },
      getInitialState: function () {
        var isFinish = false;
        if (this.props.isFinish !== undefined) {
          isFinish = this.props.isFinish;
        }

        var {formSectionId} = this.props;
        var navigationHistory = [];
        if (formSectionId) {
          var currentSection = this.props.metadata.Sections.find(
            (section) => (section.Id == formSectionId)
          );

```

```

        if (formSectionId !==
this.props.metadata.Sections[0].Id) {

this.props.metadata.Sections.every(function(section) {
    if (section.Id == formSectionId)
    {
        return false;
    }
    navigationHistory.push(section);
});
}
}
var state = {
    currentSection: currentSection ||
this.props.metadata.Sections[0],
    isFinish: isFinish,
    sessionId: null,
    navigationHistory: navigationHistory,
    formKey: Gen.elementId(),
    mode: Mode.Index,
    isProductForm: Config.getConfig().finishAction !==
'' && Config.getConfig().finishAction !== undefined
};
if (this.props.tempFormId) {
    var tempState =
this.processTempFormResultData(Constants.LocalStorageKeys.FillFormD
ata);
    state = assign({}, state, tempState);
}
else if
(localStorage.getItem(Constants.LocalStorageKeys.FillFormNavigation
Data)){
    var navigationState =
this.processTempFormResultData(Constants.LocalStorageKeys.FillFormN
avigationData);
    state = assign({}, state, navigationState);
}
return state;
},

componentWillMount(){
    if
(!localStorage.getItem(Constants.LocalStorageKeys.FillFormData)) {
        return;
    }
    if (Config.getConfig().isAuthenticated &&
!this.props.tempFormId

```



```

        && Config.getConfig().mode !== Constants.Mode.NavigationView) {
            this.saveFormResult();
        },

        processTempFormResultData: function(localStorageKey) {
            let formResult =
JSON.parse(localStorage.getItem(localStorageKey));
            localStorage.removeItem(localStorageKey);
            if(formResult.FormResultInfo.DataList) {
                FormDataStore.convertFromResultDataToStore(formResult.FormResultInfo.DataList);
            }
            var state = {
                navigationHistory: formResult.NavigationHistory,
                isFillForm: true,
                formKey: Gen.elementId(),
                mode: Mode.Index,
                isFinish: formResult.IsFinish || false
            };

            if(formResult.CurrentSection){
                state = assign({}, state, {currentSection:
formResult.CurrentSection});
            }

            if(formResult.Mode){
                Config.setConfig({mode: formResult.Mode});
            }

            return state;
        },

        componentDidUpdate: function() {
            if(this.state.editQuestionId !== undefined){
                this.setState({
                    editQuestionId: undefined
                });
                document.getElementById(this.state.editQuestionId).scrollIntoView(true)
            }
        },
    },

```

```

        onEditQuestion: function (section, navigationHistory,
questionMetadataId) {
            this.setState({
                formKey: Gen.elementId(),
                isFinish: false,
                currentSection: section,
                navigationHistory: navigationHistory,
                editQuestionId: questionMetadataId
            });
        },

        prevButtonClick: function () {
            this.scrollTop();
            var navigationHistory = this.state.navigationHistory;
            if (navigationHistory.length < 1)
                return;

            var prevSection = navigationHistory.pop();
            this.setState({currentSection: prevSection,
navigationHistory: navigationHistory});
        },

        nextButtonClick: function () {
            var isValid = FormDataStore.isValid();
            if (isValid) {
                this.scrollTop();
                var currentSection = this.state.currentSection;
                this.state.navigationHistory.push(currentSection);
                var nextSectionId =
NavigationLogic.getNextSectionId(this.props.metadata,
currentSection.Id);
                var nextSection =
this.props.metadata.Sections.find(section => section.Id ==
nextSectionId);
                if (!nextSection) {
                    this.setState({resultPage: nextSectionId});
                    this.finishButtonClick(true);
                } else {
                    this.setState({isFinish: false, currentSection:
nextSection});
                }
            } else {
                this.scrollFirstError();
            }
        },

        handleGoToSection: function (nextSectionId, openInNewTab) {

```

```

        var isValid = FormDataStore.isValid();
        if (isValid) {
            var nextSection =
this.props.metadata.Sections.find(section => section.Id ==
nextSectionId);
            if (openInNewTab){

localStorage.setItem(Constants.LocalStorageKeys.FillFormNavigationD
ata,
JSON.stringify(this.getFillFormDataForNewTabNavigation(nextSection)
));
                var cs = this.state.currentSection;
                var finish = this.state.isFinish;
                this.setState({isFinish: false, currentSection:
nextSection});
                window.open(window.location.href, "_blank");
                this.setState({isFinish: finish,
currentSection: cs})
            } else {
                this.scrollTop();
                var currentSection = this.state.currentSection;

this.state.navigationHistory.push(currentSection);

                if (!nextSection) {
                    this.setState({resultPage: nextSectionId});
                    this.finishButtonClick(true);
                } else {
                    this.setState({isFinish: false,
currentSection: nextSection});
                }
            }
        } else {
            this.scrollFirstError();
        }
    },

    onSaveSuccess: function(){
        let currentSection = this.state.currentSection;
        let nextSectionId =
NavigationLogic.getNextSectionId(this.props.metadata,
currentSection.Id);
        let nextSection =
this.props.metadata.Sections.find(section => section.Id ==
nextSectionId);
        if (!nextSection) {

```

```

        this.setState({resultPage: nextSectionId});
    }
    let resultPage = this.getResultPage();
    let mode = Config.getConfig().mode;
    let totalScore =
    ScoringCalculation.scoringTotalCalculation();
    if (resultPage &&
        (resultPage.Type == Constants.FormResultTypes.redirectUrl
        || resultPage.Type == Constants.FormResultTypes.customWithRedirectUrl
        && mode.toLowerCase() != Constants.Mode.Edit.toLowerCase()
        && mode.toLowerCase() != Constants.Mode.EditViaAssessment.toLowerCase())) {
        window.location.href = resultPage.RedirectUrl;
    } else {
        if (resultPage && (!mode || mode.toLowerCase() !=
        Constants.Mode.Edit.toLowerCase() && mode.toLowerCase() !=
        Constants.Mode.EditViaAssessment.toLowerCase()) )
        {
            Config.updateConfig({closeUrl:
            resultPage.RedirectUrl || Config.getConfig().closeUrl});
        }
        this.setState({isFinish: true, totalScore:
        totalScore});
    }
    },

    finishButtonClick: function (isSavingNecessary) {
        var isValid = FormDataStore.isValid();
        if (isValid) {
            this.scrollTop();
            var isResultPageAllowed =
            Config.getConfig().finishAction == '' ||
            Config.getConfig().finishAction == undefined;
            if(isResultPageAllowed) {
                if (isSavingNecessary) {
                    this.submitFormData(true, () =>
                    this.onSaveSuccess());
                }else{
                    this.onSaveSuccess();
                }
            }else{
                this.submitFormData(true);
            }
        }else{
    
```

```

        this.scrollFirstError();
    },
    scrollTop: function () {
        $("html, body").animate({ scrollTop: 0 });
    },
    scrollFirstError: function() {
        setTimeout( function() {
            var element = $("span.field-validation-
error:first");
            var topOffset = element.length ? $("span.field-
validation-error:first").parent().parent().parent().offset().top:
0;
            $("html, body").animate({scrollTop: topOffset});
        }, 100);
    },
    submitFormData: function (isFinish, cb) {
        var that = this;
        var dataForSave =
FormDataStore.prepareDataStoreForSave(NavigationLogic);
        var formResultId = Config.getConfig().formResultId;
        var version = Config.getConfig().formResultVersion ||
0;
        var url = Config.getSaveFormResultsUrl();
        if (formResultId || that.state.sessionId != null) {
            url = Config.getEditFormResultsUrl();
        }
        if (Config.getConfig().mode != Constants.Mode.TestRun)
        {
            var totalCalculation = [];
            Object.keys(FormMetadata.TotalCalculation).forEach(function(i) {
                totalCalculation.push({ Id:i, TotalValue:
ScoringCalculation.getCalculationQuestionValue(i)});
            });
            var data = {
                FormMetadataId: Config.getFormMetadataId(),
                DataList: dataForSave,
                FormResultInfoId: formResultId,
                CurrentSectionId: that.state.currentSection.Id,
                IsFormResultCompleted: isFinish,
                SessionId: that.state.sessionId,
                IsFillForm: that.state.isFillForm,
                TotalCalculation: totalCalculation,

```

```

        IsTemporary: Config.getConfig().isTemp,
        Version: version,
        NotificationsIdsToSend:
NotificationsLogic.getNotificationsIdsToSend(this.props.metadata.No
tifications, this.props.metadata),
        NotificationsIdsToRemind:
NotificationsLogic.getNotificationsIdsToRemind(this.props.metadata.
Notifications, this.props.metadata),
        ResultPageIsAvaibleForUnauthorizedUser:
this.getResultPage().ResultPageIsAvaibleForUnauthorizedUser
    };

    var jqxhr = $.ajax({ url: url, type: 'POST', data:
JSON.stringify(data), dataType: "json", contentType: "applica-
tion/json",
        success: function (result) {
            console.log('Success saving form data');
            if(!data.IsFormResultCompleted){
                that.setState({mode:
Mode.SuccessfullySaved });
            }
            if(!data.IsFormResultCompleted ||
Config.getConfig().finishAction == '' ||
Config.getConfig().finishAction == undefined){
                if (result != '') {
                    if (result.sessionId != null) {
                        that.setState({sessionId: re-
sult.sessionId});
                    }
                    if (result.formResultId != null) {
Config.updateConfig({formResultId: result.formResultId});
                    }
                    if (result.formResultVersion !=
null) {
Config.updateConfig({formResultVersion: result.formResultVersion});
                    }
                    if (result.isFillForm != undefined)
{
                        that.setState({isFillForm:
true});
                    }
                }
                that.setState({globalErrorMessages:
null});
            }else{

```

```

                                window.location.href =
Config.getConfig().finishAction + '?formResultInfoId=' + re-
sult.formResultId;
                                }
                                if(cb){
                                    cb();
                                }
                            }
                        });
                        jqxhr.fail((jqXHR, textStatus, errorThrown) => {
                            if (jqXHR.status == 403) {
                                var shouldAuthenticate =
jqXHR.getResponseHeader('x-s4s-should-authenticate') == "true";
                                if (shouldAuthenticate) {

localStorage.setItem(Constants.LocalStorageKeys.FillFormData,
JSON.stringify(this.getFillFormData()));

that.saveSaveFormResultToSession(Config.getAuthUrl());
                                } else {
                                    location.href =
Config.getNoPermissionUrl();
                                    that.setState({loadError: true,
errorMessage: errorThrown});
                                }
                                }else if(jqXHR.status === 500){
                                    that.setState({globalErrorMessages: ['This
object has not been found as it has been modified by another us-
er.'],mode: Mode.SavingFailed});
                                }
                                else{
                                    console.error('Fail saving form data',
textStatus, errorThrown, jqXHR.responseText);
                                    that.setState({globalErrorMessages:
[jqXHR.responseText], mode: Mode.SavingFailed});
                                }
                            });
                        },

addToFavorite: function(){
    var data = this.getFillFormData();
    var url = Config.getAddToFavoriteFormResultUrl();
    var jqxhr = $.post(url, data);

    jqxhr.fail((jqXHR, textStatus, errorThrown) => {

```

```

        console.error('Fail saving form data', textStatus,
errorThrown, jqXHR.responseText);
        this.setState({globalErrorMessages:
[jqXHR.responseText]});
    });

    return jqxhr;
},
loadFormResult:function (formResult, mode) {
    this.setState({
        currentSection: formResult.CurrentSection,
        navigationHistory: formResult.NavigationHistory,
        isFillForm: true,
        formKey: Gen.elementId(),
        mode: mode
    });
},
saveFormResult: function () {
    var that = this;
    var formResult =
JSON.parse(localStorage.getItem(Constants.LocalStorageKeys.FillForm
Data));
    var formResultId =
formResult.FormResultInfo.FormResultInfoId ||
formResult.FormResultInfo.Id;
    var version = formResult.FormResultInfo.Version;
    var url = Config.getSaveFormResultsUrl();
    var data = {
        FormMetadataId: Config.getFormMetadataId(),
        DataList: formResult.FormResultInfo.DataList,
        NotificationsIdsToSend:
formResult.FormResultInfo.NotificationsIdsToSend,
        NotificationsIdsToRemind:
formResult.FormResultInfo.NotificationsIdsToRemind,
        ResultPageIsAvaibleForUnauthorizedUser:
this.getResultPage().ResultPageIsAvaibleForUnauthorizedUser,
        CurrentSectionId: formResult.CurrentSection.Id
    };
    if (formResultId) {
        url = Config.getEditFormResultsUrl();
        data.isFillForm = true;
        data.FormResultInfoId = formResultId;
        data.Version = version;
    }
    var jqxhr = $.ajax({ url: url, type: 'POST', data:
JSON.stringify(data), dataType: "json", contentType: "applica-
tion/json",

```



```

        success: function (result) {
            if (result != '') {
                if (result.formResultId != null) {
                    Config.updateConfig({formResultId: re-
sult.formResultId});
                }
                if (result.formResultVersion != null) {
                    Config.updateConfig({formResultVersion:
result.formResultVersion});
                }

localStorage.removeItem(Constants.LocalStorageKeys.FillFormData);

FormDataStore.loadFormResult(result.formResultId, re-
sult.unauthorizedAccessKey, null, () => {
                    console.log('Success saving form
data');
                    that.loadFormResult(formResult,
Mode.SuccessfullySaved);
                }, (jqXHR) => {
                    that.ajaxErrorHandler(jqXHR);
                });
            }
        });

jqxhr.fail((jqXHR, textStatus, errorThrown) => {
    console.error('Fail saving form data', textStatus,
errorThrown, jqXHR.responseText);

localStorage.removeItem(Constants.LocalStorageKeys.FillFormData);
    that.setState({globalErrorMessages:
[jqXHR.responseText]});
});

},

getResultPage: function () {
    if(this.state != null && this.state.resultPage != unde-
fined){
        var resultPage =
this.props.metadata.FormResults.find(r => r.Id ==
this.state.resultPage);
        return resultPage;
    }

    var matchedRule = this.findMatchingResultPageRule();

```

```

        var resultPage = matchedRule ?
            this.props.metadata.FormResults.find(r => r.Id ==
matchedRule.FormResultId)
            : this.props.metadata.FormResults.find(r
=>
r.IsDefault);

        return resultPage;
    },

    findMatchingResultPageRule: function() {
        var matchedRule = null;
        if (this.props.metadata.FormResultRules.length) {
            var scoringTotal =
parseFloat(this.state.totalScore);
            matchedRule =
this.props.metadata.FormResultRules.find(rule => {
                if (!rule.Criterias ||
!rule.Criterias.Conditions || !rule.Criterias.Conditions.length)
                    return false;

                var isAnd = rule.Criterias.GroupOperator ==
Constants.GroupOperator.And;
                var aggregateFunction = isAnd ? Ar-
ray.prototype.every : Array.prototype.some;
                var result =
aggregateFunction.call(rule.Criterias.Conditions, condition => {
                    var parsedValue =
parseFloat(condition.Value);
                    switch (condition.Operator) {
                        case Constants.OperationTypes.is:
                            if (parsedValue == scoringTotal) {
                                return true;
                            }
                            break;
                        case Constants.OperationTypes.isNot:
                            if (parsedValue != scoringTotal) {
                                return true;
                            }
                            break;
                        case Constants.OperationTypes.greaterThan:
                            if (parsedValue < scoringTotal) {
                                return true;
                            }
                            break;
                        case Constants.OperationTypes.lessThan:
                            if (parsedValue > scoringTotal) {

```

```

        return true;
    }
    break;
    default:
    break;
}
return false;
});
return result;
});
}
return matchedRule;
},

ajaxErrorCommonHandler: function (jqXHR) {
    if (jqXHR.status == 403) {
        var shouldAuthenticate =
jqXHR.getResponseHeader('x-s4s-should-authenticate') == "true";
        if (shouldAuthenticate) {
            window.location.href = Config.getAuthUrl();
        } else {
            window.location.href =
Config.getNoPermissionUrl();
        }
    }
    else if (jqXHR.status == 404) {
        window.location.href = Config.getPageNotFoundUrl();
    }
},

onSaveClick: function () {
    var isValid = FormDataStore.isValid(true);
    if (isValid) {
        this.scrollTop();
        if (Config.getConfig().isAuthenticated == undefined
|| Config.getConfig().isAuthenticated) {
            this.submitFormData(false);
        } else {

localStorage.setItem(Constants.LocalStorageKeys.FillFormData,
JSON.stringify(this.getFillFormData()));

this.saveSaveFormResultToSession(Config.getAuthUrl());
        }
    }else{
        this.scrollFirstError();
    }
}

```

```

    },

    saveSaveFormResultToSession: function (redirectUrl) {
        var data = this.getFillFormData();
        var url = Config.getSaveFormResultToSessionUrl();
        var jqxhr = $.post(url, data, () => {
            window.location.href = redirectUrl;
        });

        jqxhr.fail((jqXHR, textStatus, errorThrown) => {
            console.error('Fail saving form data', textStatus,
errorThrown, jqXHR.responseText);
            this.setState({globalErrorMessages:
[ jqXHR.responseText ]});
        });
    },

    getFillFormDataForNewTabNavigation: function(nextSection){
        var data = this.getFillFormData();

        delete data.FormResultInfo.FormResultInfoId;
        delete data.FormResultInfo.Version;
        delete data.FormResultInfo.FormMetadataId;

        data.FormResultInfo.DataList
FormDataStore.prepareDataStoreForSave(NavigationLogic,
nextSection);
        data.CurrentSection = nextSection;
        data.Mode = Constants.Mode.NavigationView;
        data.metadata = this.props.metadata;

        return data;
    },

    getFillFormData: function () {
        return {
            FormResultInfo: {
                FormResultInfoId:
Config.getConfig().formResultId,
                Version: Config.getConfig().formResultVersion,
                FormMetadataId: Config.getConfig().formId,
                DataList:
FormDataStore.prepareDataStoreForSave(NavigationLogic)
            },
            CurrentSection: this.state.currentSection,
            NavigationHistory: this.state.navigationHistory,
            IsFinish: this.state.isFinish,

```

```

        PageName:
this.state.currentSection.DisplayPageDetailsOnTheForm      &&
!this.state.isFinish ? this.state.currentSection.SectionName :
this.props.metadata.Name
    };
    },

    onCloseButtonClick: function(){
        var closeBtnUrl = Config.getConfig().appBaseUrl;
        if(Config.getConfig().closeUrl){
            closeBtnUrl = Config.getConfig().closeUrl;
        }
        window.location.href = closeBtnUrl;
    },

    render: function () {
        var {isPrintView} = this.props;
        var isNavigationView = Config.getConfig().mode == Constants.Mode.NavigationView;
        if (!this.props.metadata.Sections ||
this.props.metadata.Sections.length <= 0) {
            return <div>No content. Form sections collection is
empty</div>
        }
        var currentSection = this.state.currentSection;
        var
            currentSectionIndex
this.props.metadata.Sections.findIndex(s => s.Id ==
currentSection.Id);

        var isEditAllowed = Config.getConfig().IsEditAllowed;
        var isViewMode = Config.getConfig().mode == Constants.Mode.View ||
Config.getConfig().mode == Constants.Mode.ViewAnswers;
        var isFinish = this.state.isFinish
            || (isEditAllowed !== undefined && !isEditAllowed)
            || (isViewMode !== undefined && isViewMode);

        var progressBarWidth = isFinish ? (100) :
(currentSectionIndex * 100 / this.props.metadata.Sections.length);
        var progressBarWidthPercent = Math.max(2,
progressBarWidth) + '%';

        var closeBtn = Config.getConfig().mode != Constants.Mode.TestRun ? <ActionButton label="Close"
            onClick={this.onCloseButtonClick}
        /> : null;

```

```

        var buttons = isFinish ? <ul className="actionButtons
alt" id="buttons">
    {closeBtn}
    <ActionButton label="Print"
        onClick={() => {
            var element = $("#app")[0];
            $('#buttons').hide();
            var doc = new jsPDF('p', 'pt', [ele-
ment.clientHeight, element.clientWidth]);
            var name =
`${this.props.metadata.Name}.pdf`;
            doc.addHTML(element, 0, 0, {
                pagesplit: true
            }, function () {
                doc.save(name);
                $('#buttons').show();
            });
        }}
    />
</ul> : null;

    if(Config.getConfig().mode == Constants.Mode.Edit){
        buttons = <ul className="actionButtons alt"
id="buttons">
            <ActionButton label="Save"
onClick={()=>this.onSaveClick()} />
            {closeBtn}
        </ul>
    }

    var headerImage = null;
    var titleStyles = {clear: 'both'};
    if (!isFinish && currentSection.ImagePath != null &&
currentSection.ImagePath != undefined && currentSection.ImagePath
!= '') {
        headerImage = <div className="photo-proportions"
style={{
            marginLeft: '90px',
            float: 'left',
            width: '200px',
            height: '200px',
            display: 'inline'
        }}>
            <img id="fImg" src={currentSection.ImagePath}
alt="" style={{maxWidth: '144px', maxHeight: '100px'}}/>
        </div>;
        titleStyles = {marginRight: '290px'};

```

```

    }

    let progressBar =
this.props.metadata.IsProgressBarHidden === true ? undefined :
<ProgressBar percent={progressBarWidthPercent}/>;

    var header = Config.getConfig().mode == Constants.Mode.ViewAnswers ? null :
    <div className="s4s-pathway-header-container">
      <div className="mainheader tint3">
        {buttons}
        {headerImage}
        <h1 style={titleStyles}>
          <div
            dangerouslySetInnerHTML={{__html:
currentSection.DisplayPageDetailsOnTheForm && !isFinish ?
currentSection.SectionName : this.props.metadata.Name}}></div>
          </h1>
          <h2 style={titleStyles}>
            <div dangerouslySetInnerHTML={{
              __html:
currentSection.DisplayPageDetailsOnTheForm && !isFinish ?
currentSection.Description : this.props.metadata.Description || ''
            }}></div>
          </h2>
          {!isPrintView && progressBar}
        </div>
      </div>;

    var saveButton = !isFinish && Config.getConfig().mode
!= Constants.Mode.TestRun
    && (Config.getConfig().mode == Constants.Mode.EditViaAssessment || Config.getConfig().isAuthenticated
|| this.props.metadata.IsSavingForUnauthorizedUsersAllowed)
    ? <button onClick={()=>this.onSaveClick()}
className='blockBtns small alt textAfterInput'>Save</button> : un-
defined;

    var closeButton = Config.getConfig().mode == Con-
stants.Mode.EditViaAssessment ? <button className='blockBtns small
alt textAfterInput' onClick={() => {window.location.href =
Config.getConfig().closeUrl}}>Close</button> : null;

    var prevButton = this.state.navigationHistory.length >
0 ? <button onClick={this.prevButtonClick} className='blockBtns
small alt textAfterInput' >Previous</button> : undefined;

    var nextButton = currentSectionIndex <
this.props.metadata.Sections.length - 1 ? <button

```

```

onClick={this.nextButtonClick} className='blockBtns small alt green
textAfterInput'>Next</button> : undefined;
    var finishButtonLabel = this.state.isProductForm ?
'Next': (FormMetadata.FinishButtonAlternativeName || 'Finish');
    var finishButton = currentSectionIndex ==
this.props.metadata.Sections.length - 1 ?
    <button onClick={()=>this.finishButtonClick(true)}
        className='blockBtns small alt green
textAfterInput'>{finishButtonLabel}</button> : undefined;

    var isFinishWithoutSavingAllowed =
FormMetadata.IsFinishWithoutSavingAllowed === true;
    var finishButtonWithoutSavingLabel =
(FormMetadata.FinishWithoutSavingButtonAlternativeName || 'Finish
(without saving)');
    var finishWithoutSavingButton =
(isFinishWithoutSavingAllowed && currentSectionIndex ==
this.props.metadata.Sections.length - 1) ?
    <button onClick={()=>this.finishButtonClick(false)}
        className='blockBtns small alt green
textAfterInput'>{finishButtonWithoutSavingLabel}</button> : unde-
fined;

    var content = null;
    if (!isFinish) {
        content = <div className="section">
            <div className="pad20">
                <Section
                    isTextView={isPrintView ||
isNavigationView}
                    displayNavigation={!isPrintView &&
!isNavigationView}
                    sectionMetadata={currentSection}
                    metadata={this.props.metadata}

goToSectionHandler={this.handleGoToSection}
                />
                <div className="section">
                    <div className="col span_12_of_12">
                        <div className="answer-btns
section rhs">
                            {saveButton}
                            {closeButton}
                            {prevButton}
                            {nextButton}

```



```

                                {finishButton}
                                {finishWithoutSavingButton}
                            </div>
                        </div>
                    </div>
                }
            </div>
        </div>
    } else {
        var resultPage = this.getResultPage();
        content = <Finish
showDetails={this.props.showDetails}
showResult={this.props.showResult}

onEditQuestion={this.onEditQuestion} isViewMode={isViewMode}

navigationHistory={this.state.navigationHistory}
resultPage={resultPage}></Finish>
    }

    var notificationPopupMarkup = (this.state.mode ==
Mode.SuccessfullySaved && Config.getConfig().mode != Con-
stants.Mode.Edit || this.state.mode == Mode.SavingFailed)?
        <NotificationPopup onClose={()=>
this.setState({mode: Mode.Index})} message={this.state.mode ==
Mode.SuccessfullySaved? __localisation.SaveFormSuccessMessage : "The
form saving has been failed."} />: null;

        if(this.state.mode == Mode.SuccessfullySaved &&
Config.getConfig().mode == Constants.Mode.Edit){
            this.onCloseButtonClick();
        }

        return <div key={this.state.formKey}>
            <GlobalErrorMessage message=
es={this.state.globalErrorMessages} />
            {notificationPopupMarkup}
            {header}
            {content}
        </div>;
    }
});

module.exports = Form;

```