

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 СРЕДСТВО МЕНЕДЖМЕНТА
ПЕРСОНАЛА ПРЕДПРИЯТИЯ С ИСПОЛЬЗОВАНИЕМ
ТЕХНОЛОГИИ RUBY ON RAILS**

БГУИР ДП 1-40 01 01 01 024 ПЗ

Студент

Д. В. Голубко

Руководитель

О. Г. Смолякова

Консультанты:

от кафедры ПОИТ

О. Г. Смолякова

по экономической части

В. А. Палицын

Нормоконтролёр

И. М. Марина

Рецензент

Минск 2019

РЕФЕРАТ

Пояснительная записка 136 с., 23 рис., 7 табл., 16 формул, 28 источников.

ПРОГРАММНОЕ СРЕДСТВО МЕНЕДЖМЕНТА ПЕРСОНАЛА ПРЕДПРИЯТИЯ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ RUBY ON RAILS: дипломный проект / Д.В. Голубко — Минск: БГУИР, 2019.

Цель настоящего дипломного проекта состоит в разработке программной системы, предназначенной для эффективной автоматизации процесса менеджмента персонала и генерации резюме.

При разработке и внедрении приложения используется следующий стек технологий: HTML5, CSS3, JavaScript, React.js, Ruby, Ruby on Rails, PostgreSQL.

В первом разделе проводится обзор существующих программных средств, менеджмента персонала, генерации резюме, выделяются положительные и отрицательные особенности существующих аналогов. Выдвигаются общие требования к созданию программного средства.

Во втором разделе проводится моделирование программного средства, а также разработка функциональных требований.

Третий раздел посвящён разработке архитектуры программного средства, а также модели базы данных. Также третий раздел описывает необходимые технологии для разработки программного средства и процесс разработки ПС.

В четвертом разделе содержится информация о тестировании разработанного приложения на соответствие функциональным требованиям.

Пятый раздел содержит руководство пользователя.

В шестом разделе приведено технико-экономическое обоснование разработки и внедрения программного средства.

Заключение содержит краткие выводы по дипломному проекту.

Дипломный проект является завершённым, поставленная задача решена в полной мере. Планируется дальнейшее развитие программного средства и расширение его функциональности. Проект выполнен самостоятельно, проведён анализ оригинальности в системе «Антиплагиат». Процент оригинальности составляет 81,92%. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1 Анализ литературных источников, прототипов и формирование требований к проектируемому программному средству	9
1.1 Аналитический обзор литературных источников	9
1.2 Обзор существующих аналогов	16
1.3 Требования к проектируемому программному средству	24
2 Моделирование программного средства и разработка функциональных требований	31
2.1 Функциональная модель программного средства	31
2.2 Разработка спецификации функциональных требований	36
3 Проектирование и разработка программного средства	39
3.1 Разработка архитектуры программного средства	39
3.2 Разработка даталогической и физической моделей базы данных	40
3.3 Проектирование и разработка серверной части программного средства	43
3.4 Проектирование и разработка клиентской части программного средства	54
3.5 Развертывание программного средства	62
4 Тестирование и проверка работоспособности программного средства	65
5 Руководство по использованию программного средства	73
6 Техничко-экономическое обоснование разработки и внедрения программного средства менеджмента персонала предприятия	79
6.1 Общая характеристика разрабатываемого средства менеджмента персонала предприятия	79
6.2 Расчет затрат на разработку программного обеспечения	79
6.3 Расчет стоимостной оценки результата	83
6.4 Расчет экономического эффекта	85
ЗАКЛЮЧЕНИЕ	88
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	90
Приложение А. Исходный код	93

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

Спецификация – документ, который желательно полно, точно и верифицируемо определяет требования, дизайн, поведение или другие характеристики компонента или системы, и, часто, инструкции для контроля выполнения этих требований [1].

Веб-приложение – клиент-серверное приложение, в котором клиентом выступает браузер, а сервером – веб-сервер.

Кроссплатформенность – способность программного обеспечения работать более чем на одной аппаратной платформе и (или) операционной системе.

CSS – Cascading Style Sheets – каскадные таблицы стилей – формальный язык описания внешнего вида документа, написанного с использованием языка разметки

HTML – HyperText Markup Language – язык гипертекстовой разметки – стандартизированный язык разметки документов во Всемирной паутине.

API – Application Programming Interface – программный интерфейс приложения – описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой.

JSON – текстовый формат обмена данными, основанный на JavaScript.

ПС – программное средство

ПО – программное обеспечение.

БД – база данных.

СУБД – система управления базами данных.

HR-менеджмент – управление персоналом.

ВВЕДЕНИЕ

Развитие информационных технологий повлияло и на такую сферу деятельности компании, как управление персоналом. Нельзя не отметить, что управление персоналом является одним из наиболее важных составляющих компании, так как грамотное управление персоналом способно повысить эффективность деятельности сотрудников компании и увеличить прибыль. В последнее время количество компаний, которые хотят автоматизировать процесс управления персоналом при помощи информационных систем, значительно увеличилось. Такие информационные системы помогают достичь целей компании в более короткие сроки и без дополнительных финансовых вложений.

На сегодняшний день использование информационных технологий в управлении персоналом - это необходимое условие для того, чтобы обеспечить эффективную работу любой компании. Как показывает практика, одному менеджеру кадровой службы при помощи автоматизированных систем под силу вести дела сотни сотрудников компании.

Программы, которые автоматизируют определенные участки деятельности кадровой службы, дают возможность проводить отбор, аттестацию и учет работников; разрабатывать штатное расписание; рассчитывать заработную плату; составлять аналитические отчеты тенденций предприятия. Данный вид программ используется в небольших по размерам компаниях для того, чтобы решать отдельные задачи [2].

В данном дипломном проекте реализуется программное средство менеджмента персонала, которое способно отслеживать активные вакансии в компании, текущую занятость сотрудников на проектах, их способности, а также на основе полученных данных автоматически генерировать резюме сотрудника для дальнейшего использования в отделе маркетинга и продаж.

Целью дипломного проекта является разработка и реализация кроссплатформенного веб-приложения, сообщающегося с сервером в сети Интернет с целью предоставления пользователям верной и актуальной информации о сотрудниках компании. В первую очередь разрабатываемое программное средство предназначается для работников отдела маркетинга, поскольку они наиболее часто запрашивают актуальную информацию о сотрудниках в удобном для представления формате с целью передачи данных заказчикам.

В пояснительной записке к дипломному проекту излагаются детали поэтапной разработки приложения менеджмента персонала. В первом разделе приведены результаты анализа литературных источников по теме дипломного проекта, рассмотрены особенности существующих систем-аналогов, вы-

двинуты требования к проектируемому ПС. Во втором разделе приведено описание функциональности проектируемого ПС, представлена спецификация функциональных требований. В третьем разделе приведены детали проектирования и конструирования ПС. Результатом этапа конструирования является функционирующее программное средство. В четвертом разделе представлены доказательства того, что спроектированное ПС работает в соответствии с выдвинутыми требованиями спецификации. В пятом разделе приведены сведения по развертыванию и запуску ПС, указаны требуемые аппаратные и программные средства. Обоснование целесообразности создания программного средства с технико-экономической точки зрения приведено в шестом разделе. Итоги проектирования, конструирования программного средства, а также соответствующие выводы приведены в заключении.

1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ, ПРОТОТИПОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

Конечный успех программного проекта во многом определяется до начала конструирования: на этапе подготовки, которая проводится с учетом всех особенностей проекта.

Первое предварительное условие, которое нужно выполнить перед конструированием, – ясное формулирование проблемы, которую система должна решать. Общая цель подготовки — снижение риска: адекватное планирование позволяет исключить главные аспекты риска на самых ранних стадиях работы, чтобы основную часть проекта можно было выполнить максимально эффективно.

Главный факторы риска в создании ПО — неудачная выработка требований. Требования подробно описывают, что должна делать программная система. Внимание к требованиям помогает свести к минимуму изменения системы после начала разработки [3].

Перед формулированием требований необходимо изучить ряд вопросов, которые напрямую влияют на все дальнейшие этапы разработки. В частности, необходимо рассмотреть вопросы выбора платформ, архитектуры. По результатам анализа можно будет составить техническое задание к проектируемому программному средству, которое станет основой для составления функциональных требований.

1.1 Аналитический обзор литературных источников

Далее приводится анализ сведений, которые влияют на формулирование требований, выбор архитектуры и дальнейшее проектирование и разработку программного средства.

1.1.1 Кроссплатформенность приложений

Как несколько десятилетий назад, так и в настоящее время выбор платформы является серьезным ограничением для всех последующих этапов разработки. Однако уже начали появляться технологии, которые позволяют использовать однажды написанный код на многих платформах. Сегодня все больше приложений создается сразу для нескольких платформ, а приложения, созданные изначально для одной платформы, активно адаптируются под другие [4]. Разработчик, изучающий какую-либо из таких технологий, получает конкурентное преимущество, поскольку за счет расширения количества платформ расширяется круг задач, над которыми он может работать. Поэто-

му кроссплатформенность – реальная или потенциальная – является одним из факторов, который необходимо учитывать при выборе технологий реализации проекта.

1.1.2 Обзор целевых платформ

Несмотря на планируемое использование кроссплатформенных технологий, поддержка всех платформ может затребовать значительно больших средств и времени, чем есть в наличии для выполнения дипломного проектирования. Поэтому, необходимо осуществить выбор одной основной платформы с расчетом на продолжение разработки и реализацию проекта для других платформ. Рассмотрим достоинства и недостатки основных.

Настольное приложение – программное средство, которое запускается локально на компьютере пользователя. При его создании появляется возможность использования всех преимуществ аппаратного обеспечения, которым оснащен компьютер, например: прямой доступ к видеокарте, внешним устройствам. Кроме того, появляется возможность взаимодействия с другими установленными приложениями.

Тем не менее у настольных приложений есть и ряд недостатков. Когда пользователь работает удаленно, возникают проблемы, связанные с сетью, соединениями, сетевыми экранами. Один из самых больших недостатков заключается в огромной сложности развертывания приложения на десятки или сотни машин, их конфигурирования, периодического обновления [5].

Веб-приложения, в отличие от настольных, работают на удаленном аппаратном обеспечении и поставляются пользователю через браузер [6]. При их использовании разработчик избавляется от необходимости поддерживать установку большого числа зависимостей, он всегда может быть уверен, что все пользователи используют самую последнюю версию приложения. Вычислительные операции могут производиться на мощном сервере, а результаты вычислений поставляться пользователю – так называемая концепция «тонкого клиента» [7].

Несмотря на то, что ресурсоемкие вычисления производятся на сервере, задержки при передаче, особенно при нестабильном соединении, сама потребность в постоянном интернет соединении могут значительно снизить удобство пользования приложением. Кроме того, размер загружаемого при каждом запуске кода и ресурсов может значительно увеличить траты пользователя, особенно если он использует дорогое мобильное подключение к интернету.

Для мобильных приложений актуальны ограничения платформ, на которых они запускаются, такие как меньшие размеры экранов, более медлен-

ные процессоры, ограниченное энергопотребление. Несмотря на это, мобильные устройства часто находятся рядом с пользователями, появляется возможность использования мгновенных оповещений [8]. Вместе с этим, большинство смартфонов оснащено модулями, такими как GPS, камера, NFC, что предоставляет разработчику новые возможности по их использованию.

На основании рассмотренных характеристик различных платформ можно осуществить выбор одной из них, которая и станет целевой для разработки.

1.1.3 Обзор архитектурных стилей

Далее необходимо рассмотреть применяющиеся на практике архитектурные стили, провести их анализ и по результатам осуществить выбор архитектуры, которая затем будет применяться при проектировании программного средства.

Под разработкой архитектуры понимают специфицирование структуры всей системы: глобальную организацию и структуру управления, протоколы коммуникации, синхронизации и доступа к данным, распределение функциональности между компонентами системы, физическое размещение, состав системы, масштабируемость и производительность [9]. Набор принципов, используемых в архитектуре, формирует архитектурный стиль. Применение архитектурных стилей упрощают решение целого класса абстрактных проблем [10].

При проектировании архитектуры программной системы почти никогда не ограничиваются единственным архитектурным стилем, поскольку они могут предлагать решение каких-либо проблем в различных областях. В таблице 1.1 приведен вариант категоризации архитектурных стилей [11].

Таблица 1.1 – Категоризация архитектурных стилей

Категория	Архитектурный стиль
Связь	SOA (Service-oriented architecture – архитектура, ориентированная на сервисы), Шина сообщений
Развертывание	Клиент-серверный, трехуровневый, N-уровневый
Предметная область	DDD (Domain-driven design – проблемно-ориентированное проектирование)
Структура	Компонентный, объектно-ориентированный, многоуровневый

Сервис-ориентированная архитектура позволяет приложениям предоставлять некоторую функциональность с помощью набора слабосвязанных автономных сервисов; связь между сервисами обеспечивается с помощью за-

ранее определенных контрактов. Данный стиль предоставляет следующие преимущества [11]:

- повторное использование сервисов снижает стоимость разработки;
- автономность и использование формальных контрактов способствует слабой связанности и повышает уровень абстракции;
- сервисы могут использовать возможность автоматического обнаружения и определения интерфейса;
- сервисы и использующие их приложения могут быть развернуты на различных платформах.

Архитектура шины сообщений описывает принципы построения систем, которые используют обмен сообщениями как способ связи. Наиболее часто при реализации данной архитектуры используется модель маршрутизатора сообщений или шаблон издатель-подписчик. Главные преимущества использования данного архитектурного стиля [11]:

- расширяемость, которая заключается в возможности добавлять и удалять приложения без влияния на другие;
- снижается сложность приложений, так как единственный интерфейс, который они должны поддерживать – интерфейс общей шины;
- гибкость, которая заключается в возможности подстраиваться под бизнес-требования или желания пользователей через изменения конфигурации или параметров маршрутизации сообщений;
- слабая связанность, поскольку единственное, чем связаны приложения – интерфейс общей шины;
- масштабируемость, которая заключается в возможности в случае необходимости присоединения к шине нескольких экземпляров одного и того же приложения.

Клиент-серверная архитектура описывает распределенную систему, которая включает независимые системы сервера, клиента и соединяющую их сеть. Иногда данную архитектуру называют двухзвенной. Из преимуществ выделяют следующие [10]:

- безопасность: все данные хранятся на сервере, обеспечивающем больший уровень безопасности, чем отдельные клиенты;
- централизованный доступ к данным, который предоставляет возможность более легкого управления, чем в других архитектурах;
- устойчивость и легкость поддержки: роль сервера могут выполнять несколько физических компьютеров, объединённых в сеть; благодаря этому клиент не замечает сбоев или замены отдельного серверного компьютера.

Многоуровневый архитектурный стиль заключается в группировании схожей функциональности приложения по уровням, которые выстроены в

вертикальную структуру. Уровни связаны слабо, связь между ними осуществляется по явно установленным протоколам. Строгий вариант архитектуры предполагает, что компоненты какого-либо уровня могут взаимодействовать только с компонентами одного нижележащего уровня; ослабленный вариант разрешает взаимодействие с компонентами любого из нижележащих уровней. Использование данного архитектурного стиля предлагает следующие преимущества [11]:

- есть возможность осуществлять изменения на уровне абстракций;
- изолированность: изменения на каких-либо уровнях не влияют на другие, что снижает риск и минимизирует воздействие на всю систему;
- разделение функциональности помогает управлять зависимостями, что приводит к большей управляемости всего кода;
- независимые уровни предоставляют возможность повторного использования компонентов;
- строго-определенные интерфейсы способствуют повышению тестируемости компонентов.

Многозвенная архитектура предлагает схожее с многоуровневой архитектурой разбиение функциональности, отличие же заключается в предлагаемом размещении звеньев на физически обособленной машине [10]. Предлагается использовать данный стиль в случаях, когда компоненты одного звена могут проводить дорогие в ресурсном отношении вычисления, так что это может сказаться на других уровнях, или когда некоторую чувствительную информацию переносят со звеньев уровня представления на уровень бизнес-логики приложения. Далее приведены главные преимущества использования многозвенной архитектуры:

- удобство сопровождения, которое заключается в возможности внесения изменений и обновлений в некоторые компоненты при минимальном влиянии на всё приложение;
- масштабируемость, которая возникает из распределенного развертывания;
- гибкость, которая появляется благодаря предыдущим двум пунктам;
- масштабируемость также приводит к повышению доступности приложения.

Проблемно-ориентированный архитектурный стиль основывается на предметной области, ее элементах, их поведении и связях между ними. Для применения данного стиля необходимо иметь хорошее понимание предметной области или людей, которые бы смогли объяснить ее специфику разработчикам. Первое, что нужно сделать при принятии данного стиля – разрабо-

тать единый язык, который бы знала вся команда разработчиков, который бы был избавлен от технических жаргонизмов и содержал только термины предметной области – только так можно избежать проблемы непонимания между участниками [12]. Преимущества, которые может предложить данный стиль:

- упрощение коммуникации между участниками процесса разработки благодаря выработке единого языка;
- модель предметной области обычно является расширяемой и гибкой при изменениях условий и бизнес-требований;
- хорошая тестируемость.

Компонентная архитектура основывается на декомпозиции системы в отдельные функциональные или логические компоненты, которые раскрывают другим компонентам только заранее определенные интерфейсы [11]. Преимущества данного подхода:

- легкость развертывания, которая заключается в обновлении компонентов без влияния на другие;
- использование компонентов сторонних разработчиков позволяет снизить расходы на разработку и поддержку;
- поддержка компонентами заранее определенных интерфейсов позволяет упростить разработку;
- возможность переиспользования компонентов также оказывает влияние на снижение стоимости.

Объектно-ориентированный архитектурный стиль выражается в разделении функциональности системы на множество автономных объектов, каждый из которых содержит некоторые данные и набор методов поведения, свойственных объекту. Обычной практикой является определение классов, соответствующих объектам предметной области. Применение данного стиля предоставляет следующие преимущества [11]:

- соотнесение классов программы и объектов реального мира делает программное средство более понятным;
- полиморфизм и принцип абстракции предоставляет возможность повторного их использования;
- инкапсуляция повышает тестируемость объектов;
- улучшение расширяемости, которое возникает благодаря тому, что изменения в представлении данных не оказывают влияния на внешний интерфейс объекта, что не ограничивает его способность взаимодействовать с другими объектами;
- высокая сцепленность объектов, которая достигается использованием разных объектов для разного набора действий.

Таким образом, применение рассмотренных архитектур на этапе про-

ектирования окажет большое влияние на успешность всего процесса разработки; какие бы стили не были применены, можно быть уверенным в правильности выбора за счет того, что у всех из них есть определенные и зачастую разные преимущества.

1.1.4 Проектирование баз данных

В настоящее время сложно представить сложные приложения, которые бы не использовали специальные средства для хранения информации.

База данных – представленная в объективной форме совокупность самостоятельных материалов, систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью ЭВМ. Модель базы данных – описание базы данных с помощью определенного (в т.ч. графического) языка на некотором уровне абстракции.

Основные задачи проектирования баз данных:

- обеспечение хранения в БД всей необходимой информации;
- обеспечение возможности получения данных по всем необходимым запросам;
- сокращение избыточности и дублирования данных;
- обеспечение целостности базы данных.

Выделяют следующие уровни моделирования [13]:

- инфологический уровень: описание предметной области без привязок к каким-либо средствам реализации: языкам программирования, СУБД, и т.д.;
- даталогический уровень: модель предметной области в привязке к средствам реализации;
- физический уровень: описывает конкретные таблицы, связи, индексы, методы хранения, настройки производительности, безопасности и т.д.

При ошибках моделирования могут возникать аномалии операций с БД. Аномалия – противоречие между моделью предметной области и моделью данных, поддерживаемой средствами конкретной СУБД [13].

Выделяют следующие виды аномалий:

- аномалия вставки: при добавлении данных, часть которых у нас отсутствует, мы вынуждены или не выполнять добавление или подставлять пустые или фиктивные данные;
- аномалия обновления – при обновлении данных мы вынуждены обновлять много строк и рискуем часть строк «забыть обновить»;
- аномалия удаления – при удалении части данных мы теряем другую часть, которую не надо было удалять.

Для устранения аномалий существует процесс нормализации БД.

Нормализация – группировка и/или распределение атрибутов по отношениям с целью устранения аномалий операций с БД, обеспечения целостности данных и оптимизации модели БД.

Отношение находится в первой нормальной форме, если все его атрибуты являются атомарными. Атрибут считается атомарным, если в предметной области не существует операции, для выполнения которой понадобилось бы извлечь часть атрибута.

Отношение находится во второй нормальной форме, если оно находится в первой нормальной форме, и при этом любой атрибут, не входящий в состав первичного ключа, функционально полно зависит от первичного ключа.

Отношение находится в третьей нормальной форме, если оно находится во второй нормальной форме, и при этом любой его неключевой атрибут нетранзитивно зависит от первичного ключа [13].

Как правило, третья нормальная формы принимается достаточной и, поскольку дальнейшая нормализация может породить избыточность, то она обычно не проводится.

1.2 Обзор существующих аналогов

Для решения задач менеджмента персонала могут использоваться различные средства. Рассмотрим каждую из задач в отдельности.

1.2.1 Управление кадрами

Прибыльная и стабильная деятельность любого предприятия в первую очередь зависит от правильно выбранного системного подхода к рабочему коллективу. Для достижения наилучших результатов работы сотрудников были разработаны системы, благодаря которым обеспечивается автоматизация управления персоналом.

Данные системы состоят из набора мероприятий, целью проведения которых является организация эффективного руководства работниками компании и соответственно достижение наилучшего качества управления кадровыми ресурсами. При наличии необходимого количества профессиональных сотрудников в штате организации, автоматизированная система управления персоналом гарантирует быстроту достижения бизнес-целей предприятия без дополнительных финансовых расходов. Кроме этого, привлечение HR-менеджмента к рабочему процессу какой-либо компании позволяет:

- повышать уровень продуктивной деятельности всего кадрового состава;
- значительно расширять клиентскую базу;

- увеличивать спрос на товары или услуги;
- сохранять целостность всех процессов в бизнесе;
- обеспечивать удовлетворенность клиентов.

Первой и самой основной функцией системы управления сотрудниками стало планирование персонала. Ее цель заключается в тщательной разработке политики кадрового отдела и составление стратегического плана руководства рабочим составом предприятия. Также в планирование персонала вошел профессиональный анализ потенциала работников, составление прогноза потребностей в привлечении новых соискателей и поддержание связи с внешними источниками, которые помогают привлекать в организацию новых кандидатов.

На втором месте стоит функция, отвечающая за оценку уровня знаний работников и обеспечение их дальнейшего развития. В основном в этот процесс входит осуществление обучения новых сотрудников, переподготовка и повышение уровня профессионализма всего рабочего коллектива в целом. Также специалисты отвечают за введение в должность и помогают адаптироваться новичкам в кадровом составе. Кроме этого, функция оценки, обучения и развития сотрудников помогает правильно организовывать мероприятия, позволяющие управлять карьерным ростом персонала.

На данный момент существует множество систем управления персоналом персоналом с широким набором возможностей. В процессе проектирования системы были изучены следующие программные решения:

- E-Staff;
 - 1С:Зарплата и управление персоналом.
- а) E-Staff.

Система E-Staff существует на рынке с 2000 года. Система предназначена для HR-служб компаний, осуществляющих подбор сотрудников, и для кадровых агентств. E-Staff – система полного цикла, автоматизирующая большинство рутинных операций в рекрутинге. Стартовая страница системы представлена на рисунке 1.1.

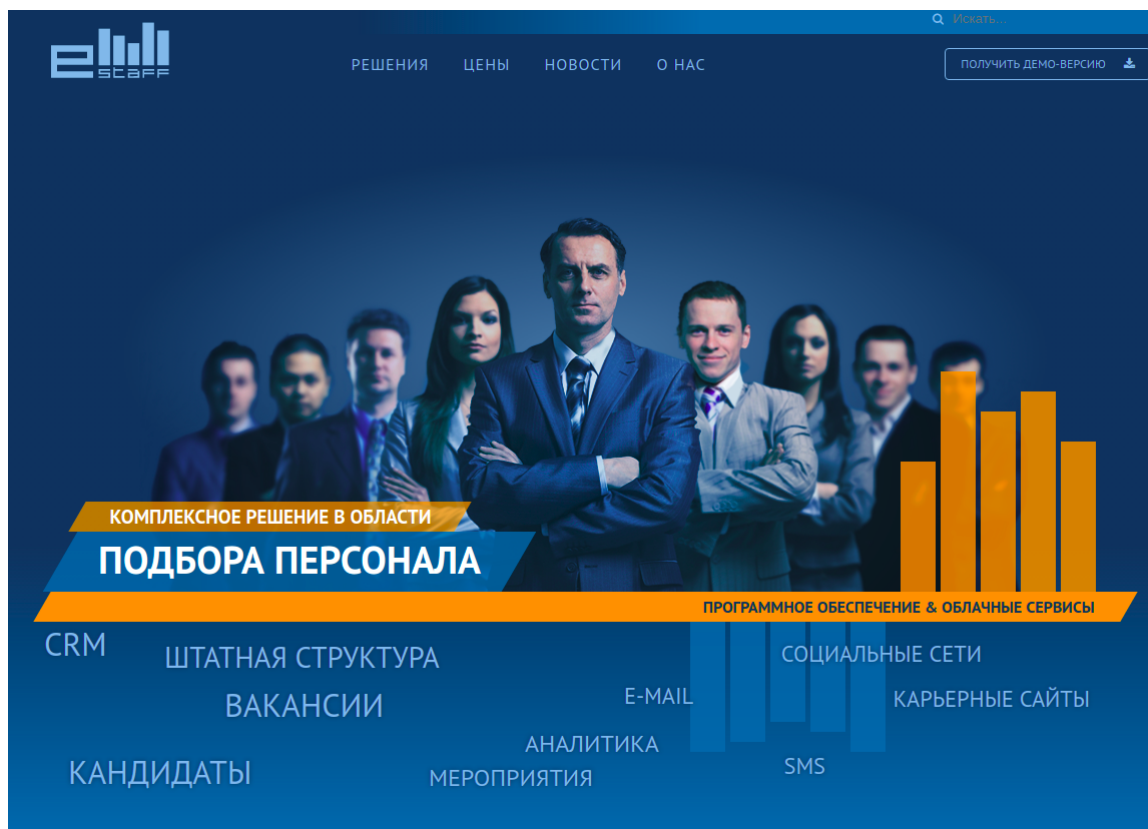


Рисунок 1.1 – Стартовая страница E-Staff

Структурные подразделения могут быть автоматически загружены из системы кадрового учета компании (1С, БОСС-Кадровик, SAP и др.). Также их можно завести вручную, если бизнес-структура компании не совпадает с формальной штатной структурой. Процесс управления подразделениями показан на рисунке 1.2.

Также как и список подразделений, список сотрудников компании может автоматически синхронизироваться с системой кадрового учета компании (1С, БОСС-Кадровик, SAP и др.). Процесс управления сотрудниками показан на рисунке 1.3.

Преимущества системы:

- хранение данных о структуре и сотрудниках компании;
- работа с вакансиями компании;
- импорт данных из сторонних систем;
- поиск резюме в сети Интернет.

Недостатки системы:

- ПО требует приобретения;
- устаревший интерфейс;
- отсутствие кроссплатформенности.

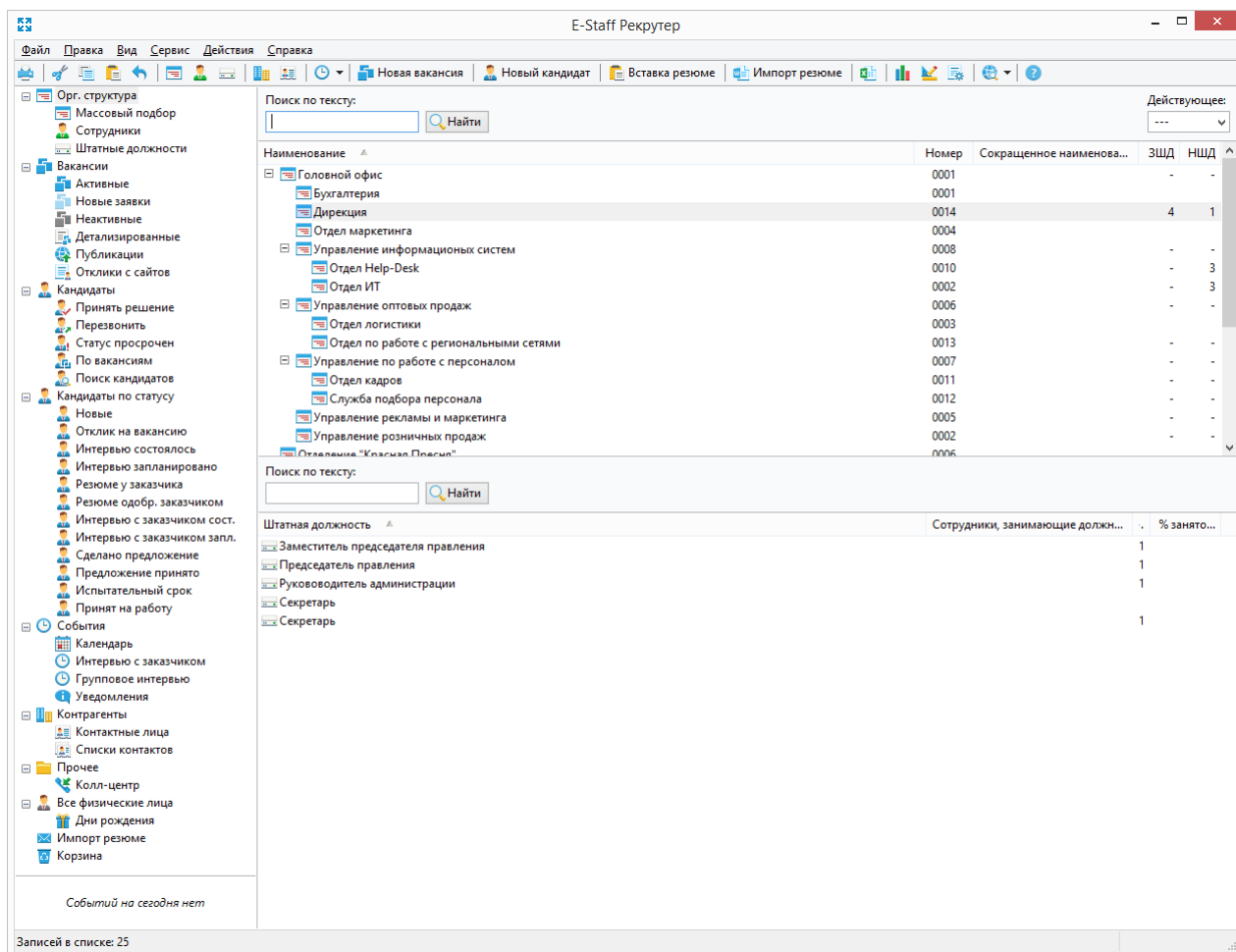


Рисунок 1.2 – Управление подразделениями в системе E-Staff

б) 1С:Зарплата и управление персоналом.

1С:Зарплата и управление персоналом – программа массового назначения, позволяющая в комплексе автоматизировать задачи, связанные с расчетом заработной платы персонала и реализацией кадровой политики, с учетом требований законодательства и реальной практики работы предприятий. Она может успешно применяться в службах управления персоналом и бухгалтериях предприятий, а также в других подразделениях, заинтересованных в эффективной организации работы сотрудников, для управления человеческими ресурсами коммерческих предприятий различного масштаба. Основные функции системы показаны на рисунке 1.4.

В 1С:Зарплате и управлении персоналом поддерживаются все основные процессы управления персоналом, а также процессы кадрового учета, расчета зарплаты, исчисления налогов, формирования отчетов и справок в государственные органы и социальные фонды, планирования расходов на оплату труда. Учтены требования законодательства, реальная практика работы предприятий и перспективные мировые тенденции развития подходов к управлению персоналом.

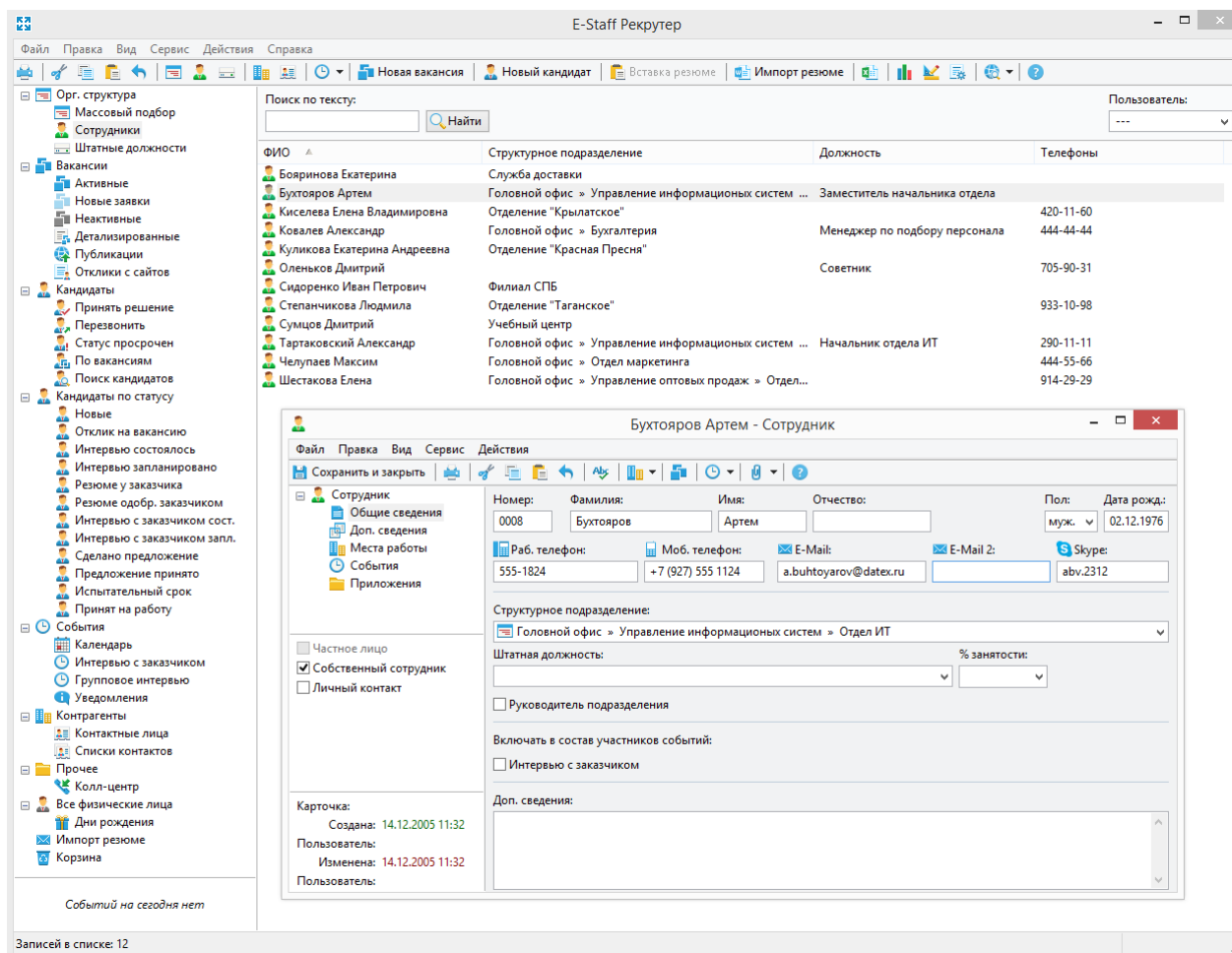


Рисунок 1.3 – Управление сотрудниками в системе E-Staff

Кадровый учет обязателен для любой компании. С каждым годом он все более регламентируется законодательством, поэтому возможность его автоматизации важна для многих компаний, особенно с большим штатом сотрудников. Кадровый учет показан на рисунке 1.5.

«1С:Зарплата и управление персоналом 8» позволяет:

- снизить временные затраты на выполнение кадровой работы;
- учитывать движение кадров;
- вести учет персональных данных работников;
- осуществлять воинский учет;
- вести учет рабочего времени;
- вести штатное расписание.

Преимущества системы:

- возможность автоматизации расчета зарплаты;
- учет движения кадров;
- учет рабочего времени;
- возможность учета персональных данных работников.

Недостатки системы:



Рисунок 1.4 – Основные функции системы 1С:Зарплата и управление персоналом

Работа с кадрами

Сотрудник: Краснова Раиса Захаровна

[Справка о зарплате \(произв. форма\)](#) [Справка с места работы](#) [Справка 2-НДФЛ](#)

Дата рожд.: 18 ноября 1960 г.

ИНН: <ИНН не заполнен>

СНИЛС: <СНИЛС не заполнен>

<документ, удостоверяющий личность не заполнен>

<телефон не заполнен>

<адрес не заполнен>

Приказы по сотруднику

Оформить документ

Приказ	Дата
Прием на работу	01.01.2015
Отпуск	13.04.2015
Отпуск	01.07.2015

Прочие приказы

Рисунок 1.5 – Кадровый учет в системе 1С:Зарплата и управление персоналом

- ПО требует приобретения;
- устаревший интерфейс;
- отсутствие кроссплатформенности;
- сложность использования;

- необходимость предварительного обучения персонала.

1.2.2 Генерация резюме

В настоящее время практика направления кандидатом, ведущим поиск работы, своего резюме работодателю или компании по подбору персонала в ответ на его объявление о вакансии находит все большее распространение не только среди многонациональных, но и белорусских компаний. Представление резюме потенциальному работодателю или агентству по подбору персонала становится одним из основных методов трудоустройства для квалифицированных кандидатов.

Резюме – это документ, в котором в краткой, но емкой форме изложены основные сведения о профессиональных умениях и навыках, трудовой биографии и личных данных кандидата. Оно позволяет работодателю заочно, с минимальными затратами времени ознакомиться в первом приближении с деловыми и личностными качествами кандидатов, произвести первичный отбор наиболее достойных и дать первую оценку их соответствия имеющейся вакансии.

Практика показывает, что в большинстве случаев специалист по подбору персонала или менеджер по работе с персоналом при принятии решения о том, пригласить кандидата на интервью или нет, основывается на содержании и форме резюме [14].

Принято считать, что профессионально составленное резюме обычно содержит следующие разделы:

- общие данные о кандидате — возраст, адрес проживания, контактная информация;
- образование — как основное, так и дополнительное;
- профессиональные умения и навыки;
- трудовая биография в обратном хронологическом порядке (последнее место работы — сначала);
- личные данные о кандидате, его увлечения;
- пожелания кандидата относительно будущей должности, уровня компенсации, возможного района или региона работы.

Существует несколько сервисов, автоматизирующих процесс генерации резюме. В процессе проектирования системы были изучены следующие существующие аналоги:

- система генерации резюме CVMaker;
- система визуализации резюме Vizualize.

а) Система генерации резюме CVMaker.

Система позволяет пользователям автоматически генерировать резюме


	<h2>Thomas Anderson</h2>															
	956, 31st Street NYC - 10001 United States	E-mail: thomas.a@thecompany.com Website: http://cvmkr.com Phone: (123) 456 789 (456) 789 125														
COMPUTER SKILLS	Microsoft Word, Excel, Access, PowerPoint, Outlook Express, Microsoft Windows XP and Microsoft Office XP Professional															
OBJECTIVE	"I can work independently using my own initiative or as part of a team."															
EDUCATION	<table border="0"> <tr> <td>MS in Accounting</td> <td>Sep 1997 – Sep 2001</td> </tr> <tr> <td colspan="2">University of Washington</td> </tr> <tr> <td colspan="2">Obtained the MS degree summa cum laude, with GPA 4.0 - http://google.com</td> </tr> <tr> <td>BS in Accounting</td> <td>Sep 1993 – Sep 1996</td> </tr> <tr> <td colspan="2">Columbia University</td> </tr> <tr> <td>BS in Computer Science</td> <td>Sep 1989 – Sep 1992</td> </tr> <tr> <td colspan="2">Columbia University</td> </tr> </table>		MS in Accounting	Sep 1997 – Sep 2001	University of Washington		Obtained the MS degree summa cum laude, with GPA 4.0 - http://google.com		BS in Accounting	Sep 1993 – Sep 1996	Columbia University		BS in Computer Science	Sep 1989 – Sep 1992	Columbia University	
MS in Accounting	Sep 1997 – Sep 2001															
University of Washington																
Obtained the MS degree summa cum laude, with GPA 4.0 - http://google.com																
BS in Accounting	Sep 1993 – Sep 1996															
Columbia University																
BS in Computer Science	Sep 1989 – Sep 1992															
Columbia University																
WORK EXPERIENCE	<table border="0"> <tr> <td>MyOffice Inc, Boston</td> <td>Oct 2005 – Present</td> </tr> <tr> <td colspan="2">Administrator</td> </tr> <tr> <td colspan="2"> <ul style="list-style-type: none"> Performed general office duties and administrative tasks. Prepared weekly confidential sales reports for presentation to management. </td> </tr> </table>		MyOffice Inc, Boston	Oct 2005 – Present	Administrator		<ul style="list-style-type: none"> Performed general office duties and administrative tasks. Prepared weekly confidential sales reports for presentation to management. 									
MyOffice Inc, Boston	Oct 2005 – Present															
Administrator																
<ul style="list-style-type: none"> Performed general office duties and administrative tasks. Prepared weekly confidential sales reports for presentation to management. 																

Рисунок 1.6 – Пример готового резюме в системе CVMaker

ме. Пример показан на рисунке 1.6. CVMaker дает на выбор 6 бесплатных шаблонов для резюме, выполненных в строгом классическом стиле. Однако система никак не связана с компанией и не позволяет управлять уже созданными резюме.

Преимущества:

- бесплатность;
- возможность выбора шаблона;
- возможность выбора формата загрузки.

Недостатки:

- отсутствие связи с компанией;
- невозможность проверки заполненного резюме сотрудником компании встроенными средствами;


– невозможность корректировки уже сгенерированного резюме.

б) Система визуализации резюме Vizualize.

Скучные списки о прошлых местах работы, заслугах и навыках можно

Visualize your resume in one click

Create your infographic resume for free

 Sign in with LinkedIn

Sign up with email

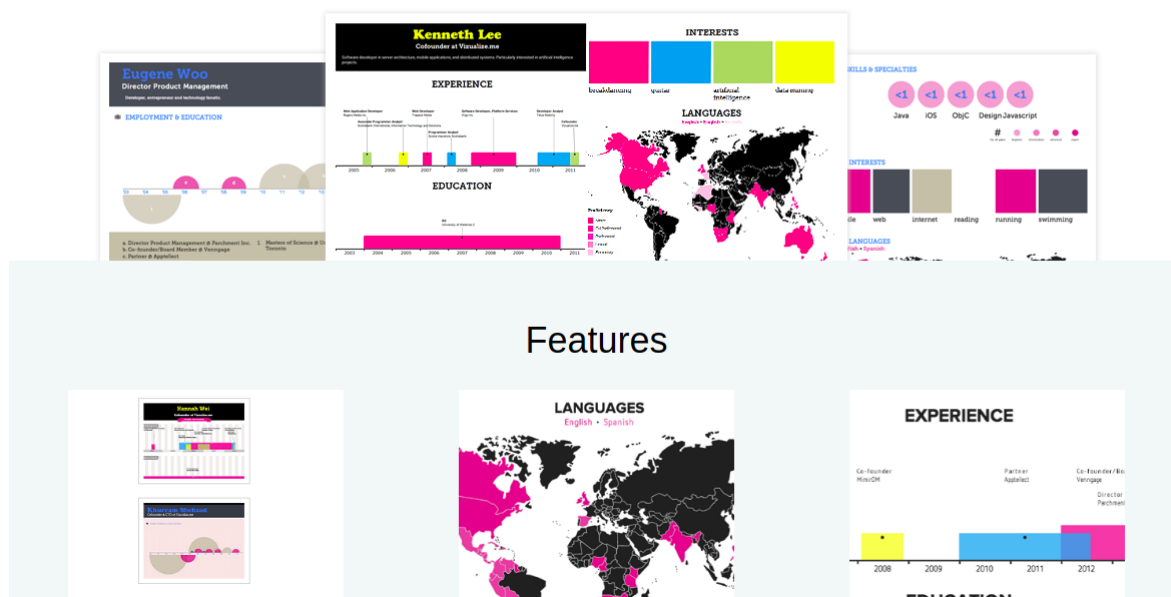


Рисунок 1.7 – Стартовая страница Visualize

превратить в интересную инфографику на Visualize. Стартовая страница системы показана на рисунке 1.7. Чтобы создать ее, нужен профиль в LinkedIn. Информацию можно импортировать также из Twitter, Facebook и Foursquare.

Преимущества:

- бесплатность;
- визуализация резюме;
- возможность импорта данных из социальных сетей.

Недостатки:

- отсутствие связи с компанией;
- невозможность проверки заполненного резюме сотрудником компании встроенными средствами;
- невозможность вывода резюме на бумажный носитель.

1.3 Требования к проектируемому программному средству

По результатам изучения предметной области, анализа литературных источников и обзора существующих систем-аналогов сформулируем требования к проектируемому программному средству.

1.3.1 Назначение проекта

Назначением проекта является разработка программного средства, обеспечивающего отслеживание активных вакансий в компании, текущую занятость сотрудников на проектах, их способности, а также автоматическая генерация на основе полученных данных резюме сотрудника для дальнейшего использования в отделе маркетинга и продаж.

1.3.2 Основные функции

Программное средство должно поддерживать следующие основные функции:

- регистрация и аутентификация;
- возможность приглашения пользователей;
- поддержка системы ролей;
- управление проектами организации;
- управление отделами организации;
- управление должностями;
- управление профилями пользователей;
- управление вопросами для генерации резюме;
- генерация резюме;
- возможность проверки резюме другими сотрудниками организации;
- уведомление сотрудников о необходимости проверки резюме.

1.3.3 Требования к входным данным

Входные данные для программного средства должны быть представлены в виде вводимого пользователем с помощью клавиатуры текста и выбора доступных опций пользовательского интерфейса.

Должны быть реализованы проверки вводимых данных на корректность с отображением информации об ошибках в случае их некорректности.

1.3.4 Требования к выходным данным

Выходные данные программного средства должны быть представлены посредством отображения информации с помощью различных элементов пользовательского интерфейса.

1.3.5 Требования к временным характеристикам

Производительность программно-аппаратного комплекса должна обеспечивать следующие временные характеристики: время реакции на запрос пользователя не должно превышать 2 секунд при минимальной скорости соединения 10 МБит/с. Допускается невыполнение данного требования в слу-

чае, когда невозможность обеспечить заявленную производительность обусловлена объективными внешними причинами.

1.3.6 Требования к надежности

Надежное функционирование программы должно быть обеспечено выполнением следующих организационно-технических мероприятий:

- организация бесперебойного питания;
- выполнение рекомендаций Министерства труда и социальной защиты РБ, изложенных в Постановлении от 23 марта 2011 г. «Об утверждении Норм времени на работы по обслуживанию персональных электронно-вычислительных машин, организационной техники и офисного оборудования»;
- выполнение требований ГОСТ 31078-2002 «Защита информации. Испытания программных средств на наличие компьютерных вирусов»;
- необходимым уровнем квалификации пользователей.

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), нефатальным сбоем операционной системы, не должно превышать времени, необходимого на перезагрузку операционной системы и запуск программы, при условии соблюдения условий эксплуатации технических и программных средств. Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

Отказы программы возможны вследствие некорректных действий пользователя при взаимодействии с операционной системой. Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу конечного пользователя без предоставления ему административных привилегий.

1.3.7 Требования к аппаратному обеспечению серверной части

ЭВМ, на которой должна функционировать серверная часть программного средства, должна обладать следующими минимальными характеристиками:

- процессор Intel Core i5 с тактовой частотой 2 ГГц;
- жесткий диск объемом 100 Гб;
- оперативная память 4 Гб;
- сетевая карта Ethernet 100 МБит/с.

Также для функционирования серверной части требуется установленный Ruby on Rails сервер, который работает на операционной системе Linux.

1.3.8 Требования к аппаратному обеспечению клиентской части

Клиентская часть программного средства должна функционировать на ЭВМ со следующими минимальными характеристиками:

- процессор Intel Celeron с тактовой частотой 2 ГГц и более;
- оперативная память 4 Гб и более;
- возможность выхода в сеть Интернет.

Для корректной работы программного средства необходим один из следующих браузеров с соответствующей минимальной версией:

- Google Chrome 70;
- Opera 58;
- Mozilla Firefox 66;
- Apple Safari 12.0;
- Microsoft Edge 44.

1.3.9 Выбор технологий программирования

Язык программирования, на котором будет реализована система, заслуживает большого внимания, так как вы будете погружены в него с начала конструирования программы до самого конца. Исследования показали, что выбор языка программирования несколькими способами влияет на производительность труда программистов и качество создаваемого ими кода. Если язык хорошо знаком программистам, они работают более производительнее. Данные, полученные при помощи модели оценки Сосото II, показывают, что программисты, использующие язык, с которым они работали три года или более, примерно на 30% более продуктивны, чем программисты, обладающие аналогичным опытом, но для которых язык является новым [15]. В более раннем исследовании, проведенном в IBM, было обнаружено, что программисты, обладающие богатым опытом использования языка программирования, были более чем втрое производительнее программистов, имеющих минимальный опыт [16].

JavaScript ("JS" для краткости) — это полноценный динамический язык программирования, который применяется к HTML документу, и может обеспечить динамическую интерактивность на веб-сайтах.

JavaScript сам по себе довольно компактный, но очень гибкий. Разработчиками написано большое количество инструментов поверх основного языка JavaScript, которые разблокируют огромное количество дополнительных функций с очень небольшим усилием. К ним относятся:

- программные интерфейсы приложения (API), встроенные в браузеры, обеспечивающие различные функциональные возможности, такие как динамическое создание HTML и установку CSS стилей, захват и манипуляция

видеопотоком, работа с веб-камерой пользователя или генерация 3D графики;

- сторонние API позволяют разработчикам внедрять функциональность в свои сайты от других разработчиков, таких как Twitter или Facebook;
- также есть возможность применить к HTML сторонние фреймворки и библиотеки, что позволит ускорить создание сайтов и приложений [17].

Выбранный язык программирования является средством для программирования клиентской части приложения. Поскольку для приложения в любом случае понадобится база данных, то есть два варианта:

- осуществлять запросы к БД напрямую с клиентского приложения;
- реализовать серверную прослойку между клиентской частью и базой данных.

Первый подход крайне небезопасен: очень опасно предоставлять открытый доступ к БД. В это же время, второй способ, помимо ее сокрытия, предоставляет возможность проверки подлинности и предоставления прав пользователям. В связи с этим появляется проблема выбора технологий для серверной части. Основным влияющим фактором является имеющийся опыт команды разработки, в связи с чем была выбрана технология Ruby on Rails и язык программирования Ruby.

Язык программирования Ruby, является кроссплатформенным языком программирования. Ruby – это тщательно сбалансированный язык. Его создатель Юкиhiro Мацумото, объединил части его любимых языков (Perl, Smalltalk, Eiffel, Ada и Lisp) чтобы сформировать новый язык, в котором парадигма функционального программирования сбалансирована принципами императивного программирования.

В Ruby всё – объект. Для каждой частицы информации или кода могут быть определены собственные свойства и действия. В объектно-ориентированном программировании свойства называются переменными объекта, а действия – методами. Чистейший объектно-ориентированный подход Ruby может быть продемонстрирован парой строк кода, в которых производится действие над числом.

Во многих языках числа и другие примитивные типы данных не являются объектами. Ruby под влиянием языка Smalltalk позволяет задать методы и переменные объекта всем типам данных. Это упрощает использование Ruby, так как правила применимые к объектам – применимы ко всему Ruby [18].

Ruby on Rails - фреймворк для веб-разработки, написанный на языке программирования Ruby. Он разработан, чтобы сделать программирование веб-приложений проще, так как использует ряд допущений о том, что нужно каждому разработчику для создания нового проекта. Он позволяет писать

меньше кода в процессе программирования, в сравнении с другими языками и фреймворками. Ruby on Rails реализует архитектурный шаблон Model-View-Controller, представленный на рисунке 1.8 для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером баз данных.

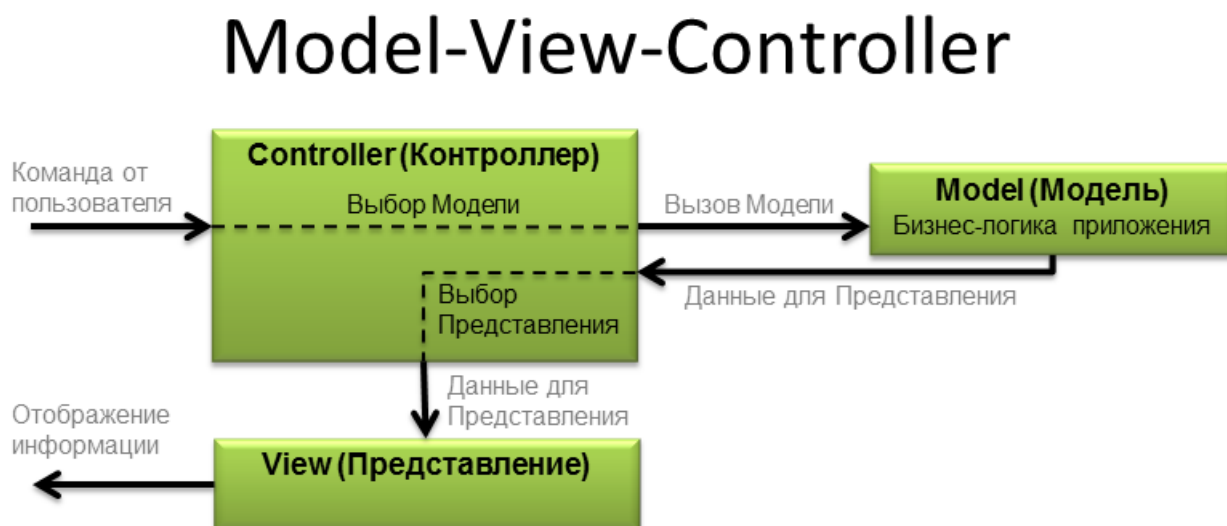


Рисунок 1.8 – архитектурный шаблон Model-View-Controller

Философия Rails включает два важных принципа:

- Don't Repeat Yourself: DRY – это принцип разработки ПО, который гласит, что "Каждый кусочек информации должен иметь единственное, неизбыточное, авторитетное представление в системе." Не пишите одну и ту же информацию снова и снова, код будет легче поддерживать, и он будет более расширяемым и менее ошибочным;
- Convention Over Configuration: у Rails есть мнения о наилучших способах делать множество вещей в веб-приложении, и по умолчанию выставлены эти соглашения, вместо того, чтобы заставлять вас по мелочам править многочисленные конфигурационные файлы [19].

По результатам обзора возможных платформ, представленных в пункте 1.1.2, было принято решение выбрать основной для разработки платформу веб-приложений.

В качестве СУБД было принято решение использовать PostgreSQL – свободно распространяемую объектно-реляционную систему управления базами данных, которая является наиболее развитой из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.

Надежность PostgreSQL является проверенным и доказанным фактом и обеспечивается следующими возможностями:

- полное соответствие принципам ACID – атомарность, непротиворечивость, изолированность, сохранность данных. Атомарность – транзакция

рассматривается как единая логическая единица, все ее изменения или сохраняются целиком, или полностью откатываются. Непротиворечивость – транзакция переводит базу данных из одного непротиворечивого состояния (на момент старта транзакции) в другое непротиворечивое состояние (на момент завершения транзакции). Непротиворечивым считается состояние базы, когда выполняются все ограничения физической и логической целостности базы данных, при этом допускается нарушение ограничений целостности в течение транзакции, но на момент завершения все ограничения целостности, как физические, так и логические, должны быть соблюдены. Изолированность – изменения данных при конкурентных транзакциях изолированы друг от друга на основе системы версионности. Сохранность данных – PostgreSQL заботится о том, что результаты успешных транзакций гарантировано сохраняются на жесткий диск вне зависимости от сбоев аппаратуры;

- многоверсионность используется для поддержания согласованности данных в конкурентных условиях, в то время как в традиционных базах данных используются блокировки. многоверсионность означает, что каждая транзакция видит копию данных (версию базы данных) на время начала транзакции, несмотря на то, что состояние базы могло уже измениться;

- репликация также повышает надежность PostgreSQL;

- открытость кодов PostgreSQL означает их абсолютную доступность для любого, а либеральная BSD лицензия не накладывает никаких ограничений на использование кода.

Производительность PostgreSQL основывается на использовании индексов, интеллектуальном планировщике запросов, тонкой системы блокировок, системе управления буферами памяти и кэширования, превосходной масштабируемости при конкурентной работе [20].

Сформулированные требования позволят осуществить успешное проектирование и разработку программного средства.

2 МОДЕЛИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

2.1 Функциональная модель программного средства

Функциональная модель программного средства представлена в виде диаграммы вариантов использования и информационной модели предметной области. Варианты использования отражают функциональность системы в ответ на внешние воздействия с точки зрения получения значимого результата для пользователей. Информационная модель предметной области в дальнейшем будет использоваться при проектировании базы данных для программного средства.

2.1.1 Варианты использования программного средства

Проектируемое программное средство предполагает, что пользователи делятся на обычных и менеджеров.

Возможности обычного пользователя и менеджера представлены на рисунке 2.1 в виде диаграммы вариантов использования, разработанной с использованием нотации UML 2.1.

Рассмотрим подробно представленные на рисунке прецеденты.

Регистрация, аутентификация и авторизация – функции, которые доступны для роли «Гость» (пользователь, не зарегистрированный в системе). В первой версии приложения планируется реализация собственной системы авторизации; в дальнейшем будет добавлена возможность регистрации с помощью внешних поставщиков данных (Google).

После регистрации пользователь получает доступ к функциям заполнения профиля. Среди функций заполнения профиля стоит отметить:

- редактирование личной информации. В эту функцию входит возможность заполнения имени, фамилии, даты рождения, адреса электронной почты, места жительства, номера телефона;
- редактирование данных об образовании. В этот блок входит возможность ввода учебного заведения или названия курсов, годов начала и конца обучения, возможность указания специализации и ученой степени. Есть возможность добавления нескольких мест учебы. Также пользователю предлагается указать свой уровень владения английским языком, что является крайне важным фактором при работе в сфере информационных технологий;
- редактирование профессиональных данных. В этой секции пользователю предлагается ответить на несколько вопросов, связанных с его профессией. Например, разработчику серверной части программного обеспечения будут заданы следующие вопросы: "В каких технологиях серверной части

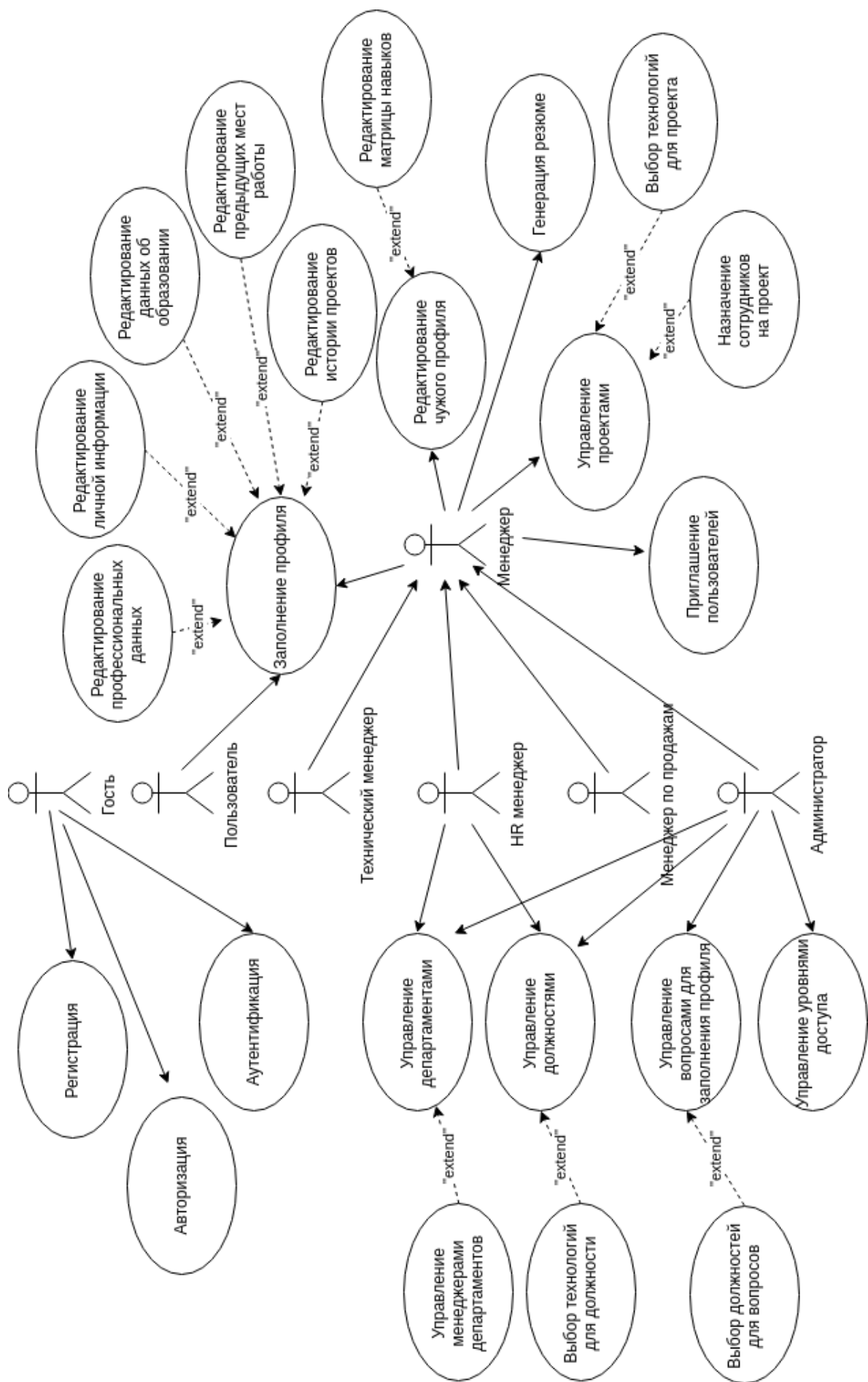


Рисунок 2.1 – Диаграмма вариантов использования ПС

ПО вы хороши? "В каких технологиях клиентской части ПО вы хороши? "С какими базами данных вы имели опыт работы? "Какие системы управления проектами вы использовали? "С какими методологиями разработки ПО вы знакомы? "Какие другие инструменты вы использовали в вашей работе?";

- редактирование предыдущих мест работы. Здесь можно добавлять предыдущие места работы с указанием начала и окончания работы. Также есть возможность добавить в свое резюме предыдущие рабочие проекты. Здесь можно указать название проекта, краткое его описание, зону ответственности, время начала и окончания работы над проектом, роль в команде, а также выбрать из списка технологии, использовавшиеся на данном проекте;

- редактирование истории проектов в текущей организации.

Кроме обычных пользователей, в системе присутствуют менеджеры, которые обладают этим же функционалом. Кроме того, все менеджеры имеют следующий набор функций:

- менеджеры имеют возможность редактировать профили других пользователей. Кроме того, они могут редактировать матрицу навыков пользователя, чего не может делать обычный пользователь;

- менеджеры могут генерировать из заполненного профиля резюме для последующей отправки заказчику;

- менеджеры могут управлять проектами в компании. Среди функций управления проектами стоит отметить возможность выбора технологий, которые будут использоваться на проекте и возможность назначения сотрудников на проект. Таким образом осуществляется отслеживание занятости персонала на проектах;

- менеджер может приглашать в систему новых пользователей посредством отправки писем на указанную электронную почту. Условием приглашения пользователя является наличие электронной почты в корпоративном домене.

В системе представлены следующие типы менеджеров:

- технический менеджер;
- HR менеджер;
- менеджер по продажам.

Технический менеджер и менеджер по продажам имеют возможности обычного менеджера, которые описаны выше, HR менеджер, кроме вышеперечисленных функций также имеет возможность управления департаментами в компании(создание, редактирование, удаление), с возможностью назначения менеджеров(каждый департамент должен иметь своего HR менеджера, технического менеджера и менеджера по продажам). HR менеджер также имеет доступ к управлению должностями компании с определением спис-

ка технологий, о знании которых может указывать работник при заполнении своего профиля.

Кроме менеджеров, гостей и обычных пользователей, в системе существует еще один вид пользователя – администратор. Помимо всех функций обычного пользователя и менеджеров, администратор также имеет возможность управлять списком вопросов для каждой должности, которые будут в дальнейшем задаваться при заполнении профессиональных данных пользователя, и управлять уровнями доступа для менеджеров (можно разрешать или запрещать те или иные действия определенному типу менеджеров).

2.1.2 Разработка инфологической модели базы данных

Исходя из необходимости использования в проектируемом приложении базы данных, разработаем ее инфологическую модель.

Предметная область разрабатываемого проекта включает следующие сущности и атрибуты:

а) пользователь:

- 1) адрес электронной почты;
- 2) хешированный пароль;
- 3) количество авторизаций;
- 4) время текущей авторизации;
- 5) время последней авторизации;
- 6) имя;
- 7) идентификатор позиции;

б) вопрос:

- 1) название;
- 2) описание;
- 3) краткий ответ;

в) позиция:

- 1) название;
- 2) идентификатор департамента;
- 3) наличие матрицы навыков;

г) департамент:

- 1) название;
- 2) офис;

д) категория:

- 1) название;

е) навык:

- 1) название;
- 2) идентификатор категории;

- ж) технология:
 - 1) идентификатор технологии;
 - 2) тип технологии;
 - 3) идентификатор навыка;
- з) список прав:
 - 1) права;
 - 2) идентификатор роли;
- и) журнал:
 - 1) идентификатор пользователя;
 - 2) тип журнала;
 - 3) идентификатор журнала;
 - 4) исходное состояние;
 - 5) конечное состояние;
 - 6) данные;
- к) роль:
 - 1) название;
- л) пользователь департамента:
 - 1) идентификатор пользователя;
 - 2) идентификатор департамента;
 - 3) идентификатор роли;
- м) профиль:
 - 1) идентификатор пользователя;
 - 2) информация;
 - 3) статус;
 - 4) идентификатор родителя;
 - 5) ссылка на фотографию;
- н) пользователь проекта:
 - 1) идентификатор пользователя;
 - 2) идентификатор проекта;
 - 3) роль;
 - 4) зона ответственности;
- о) проект:
 - 1) название;
 - 2) описание;
 - 3) статус;

- п) вопрос позиции:
 - 1) идентификатор вопроса;
 - 2) идентификатор позиции.

2.2 Разработка спецификации функциональных требований

С учетом требований, определенных в подразделе 1.3, представим детализацию функций проектируемого ПС.

2.2.1 Функция регистрации

Функция регистрации должна быть реализована с учетом следующих требований:

- а) процесс регистрации инициируется пользователем системы (на рисунке 2.1 представлен в виде роли «Гость»);
- б) для регистрации пользователь обязан предоставить адрес электронной почты и установить пароль;
- в) правильность предоставленного адреса электронной почты должна проверяться путем отправки письма со ссылкой, переход по которой означает подтверждение пользователя;
- г) хранение пароля допускается только в хешированном виде; применяющийся алгоритм должен по криптостойкости быть равным или превосходить алгоритмы семейства SHA-2. Использование соли обязательно;
- д) должна быть предусмотрена возможность смены пароля и после регистрации;
- е) должна быть предусмотрена возможность регистрации по приглашению.

2.2.2 Функция аутентификации

Функция аутентификации должна быть реализована с учетом следующих требований:

- а) инициатором является пользователь, при этом ему необходимо предоставить адрес электронной почты и пароль, заданные при регистрации;
- б) должна быть реализована возможность повторной аутентификации пользователя без необходимости ввода какой-либо информации;
- в) должна быть реализована возможность восстановления пароля:
 - 1) для восстановления пароля пользователь должен предоставить адрес электронной почты, зарегистрированный в системе;
 - 2) на предоставленный адрес высылается уникальная ссылка;
 - 3) после перехода пользователем по данной ссылке ему предоставляется возможность установить новый пароль.

2.2.3 Система ролей

При реализации системы ролей следует учесть требования:

- а) должны быть реализованы следующие роли:
 - 1) обычный пользователь;
 - 2) технический менеджер;
 - 3) менеджер по продажам;
 - 4) HR менеджер;
 - 5) администратор;
- б) для роли администратор должна быть реализована возможность назначения любого пользователя менеджером;
- в) для роли администратор должны быть реализована возможность управления списком прав каждого вида менеджеров
- г) должна быть реализована возможность наличия у одного пользователя нескольких разных ролей.

2.2.4 Функция заполнения профиля

Заполнение профиля входит в число основных функций разрабатываемого приложения. При реализации функции заполнения профиля следует учесть требования:

- а) необходимо обеспечить внесение всей необходимой информации в удобном и понятном виде;
- б) необходимо обеспечить редактирование уже заполненного профиля;
- в) перед подтверждением правильности внесенных данных профиль должен пройти проверку трех менеджеров департамента;
- г) Из заполненного и подтвержденного профиля можно создать копию.

2.2.5 Функция генерации резюме

Генерация резюме также входит в число основных функций разрабатываемого приложения. При реализации генерации резюме следует учесть требования:

- а) необходимо генерировать резюме в формате, пригодном для вывода на бумажный носитель;
- б) сгенерированное резюме должно содержать всю необходимую информацию из заполненного профиля пользователя;
- в) генерировать резюме могут только пользователи с ролью технический менеджер, менеджер по продажам, HR менеджер и администратор.

2.2.6 Функция управления проектами

При реализации управления проектами следует учесть требования:

- а) управлять проектами могут только менеджеры и администратор;
- б) при необходимости возможность управления проектами может быть отключена у любого вида менеджеров;
- в) необходимо обеспечить создание, редактирование, удаление проектов, их описания;
- г) необходимо обеспечить выбор технологий, используемых на проекте;
- д) необходимо обеспечить добавление, редактирование, удаление пользователей, работающих на проекте, описание их роли и зоны ответственности;
- е) необходимо обеспечить автоматическое добавление выбранных технологий в профили пользователей, работающих на проекте.

2.2.7 Функция управления департаментами

При реализации управления департаментами следует учесть требования:

- а) управлять департаментами могут только менеджеры и администратор;
- б) при необходимости возможность управления департаментами может быть отключена у любого вида менеджеров;
- в) необходимо обеспечить создание, редактирование, удаление департаментами, их описания;
- г) необходимо обеспечить выбор менеджеров департамента для дальнейшей проверки заполненных профилей;
- д) необходимо обеспечить добавление, редактирование, удаление пользователей, работающих в департаменте, их позицию в департаменте.

3 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Разработка архитектуры программного средства

Как было указано в пункте 1.3.9, на основании анализа вариантов проектирования приложения для различных платформ, проведенного в пункте 1.1.2, было принято решение выбрать основной для разработки платформу веб-приложений.

Классической архитектурой для приложений данного типа является двузвенная клиент-серверная архитектура. В простейшем случае единственная задача сервера – возврат по запросу статической страницы с некоторой информацией. Однако в настоящее время данный подход считается несовременным и используется крайне редко: в основном для информационных сайтов [21]. Абсолютное большинство ресурсов сети Интернет реализованы с высокой степенью интерактивности, то есть предоставляют некоторые элементы, с которыми пользователь может взаимодействовать. Таким образом, клиентская часть приложения, помимо простого отображения информации, должна быть управляемой с помощью пользовательских воздействий. Выбранная в пункте 1.3.9 программная платформа позволяет нам это реализовать.

Вследствие интерактивности появляется необходимость в хранилище данных, которая обычно решается с помощью реляционных баз данных. В пункте 1.3.9 указано, что в приложении, описываемом в данном дипломном проекте будет использоваться СУБД PostgreSQL, которая может запускаться на физически отдельной ЭВМ. Таким образом, приходим к трехзвенной архитектуре, схема которой представлена на рисунке 3.1.



Рисунок 3.1 – Вариант трехзвенной архитектуры

Появляется вопрос о необходимости серверной части приложения, ведь клиентская часть с помощью, например, протокола HTTP может сама без промежуточных звеньев обращаться к базе данных. При выборе данного варианта

исчезает необходимость в дублировании сущностей модели (в базе, на клиенте и на сервере), а также в создании кода, который бы просто перенаправлял запросы от клиента к базе и переупаковывал данные из одних структур в другие при отправке клиенту. Однако от данной модификации архитектуры было решено отказаться, поскольку ему свойственны существенные риски нарушения требований безопасности: потребовалось бы наличие открытого API для взаимодействия клиента с БД, которое могло бы легко использоваться злоумышленником. Серверная часть же может использоваться для реализации аутентификации и авторизации.

3.2 Разработка даталогической и физической моделей базы данных

Как было упомянуто ранее, в программном средстве, описываемом данным дипломным проектом, будет использоваться СУБД PostgreSQL. Описание схемы БД для нее может производиться в том числе с помощью механизма миграций и моделей в веб-фреймворке Ruby on Rails. Тем не менее, разрабатывать модель схемы можно с помощью любых средств, однако затем разработанную модель нужно вручную перевести в формат, используемый СУБД.

На даталогическом уровне модель предметной области представляется в привязке к конкретной СУБД и описывает способ организации данных безотносительно их физического размещения. Описывать модель можно с помощью специальных графических нотаций [13].

Модель даталогического уровня разработаем на основании инфологической модели, описание которой приведено в пункте 2.1.2.

СУБД PostgreSQL не просто реляционная, а объектно-реляционная СУБД. Это даёт ему некоторые преимущества над другими SQL базами данных с открытым исходным кодом, такими как MySQL, MariaDB и Firebird.

Фундаментальная характеристика объектно-реляционной базы данных — это поддержка пользовательских объектов и их поведения, включая типы данных, функции, операции, домены и индексы. Это делает PostgreSQL невероятно гибким и надежным. Среди прочего, он умеет создавать, хранить и извлекать сложные структуры данных.

Существует обширный список типов данных, которые поддерживает PostgreSQL. Кроме числовых, с плавающей точкой, текстовых, булевых и других ожидаемых типов данных (а также множества их вариаций), PostgreSQL может похвастаться поддержкой uuid, денежного, перечисляемого, геометрического, бинарного типов, сетевых адресов, битовых строк, текстового поиска, xml, json, массивов, композитных типов и диапазонов, а

также некоторых внутренних типов для идентификации объектов и местоположения логов. Справедливости ради стоит сказать, что MySQL, MariaDB и Firebird тоже имеют некоторые из этих типов данных, но только PostgreSQL поддерживает их все.

Поддержка JSON в PostgreSQL позволяет перейти к хранению schema-less данных в SQL базе данных. Это может быть полезно, когда структура данных требует определённой гибкости: например, если в процессе разработки структура всё ещё меняется или неизвестно, какие поля будет содержать объект данных.

Тип данных JSON обеспечивает проверку корректности JSON, который позволяет использовать специализированные JSON операторы и функции, встроенные в Постгрес для выполнения запросов и манипулирования данными. Также доступен тип JSONB — двоичная разновидность формата JSON, у которой пробелы удаляются, сортировка объектов не сохраняется, вместо этого они хранятся наиболее оптимальным образом, и сохраняется только последнее значение для ключей-дубликатов. JSONB обычно является предпочтительным форматом, поскольку требует меньше места для объектов, может быть проиндексирован и обрабатывается быстрее, так как не требует повторного синтаксического анализа.

Для создания даталогической модели базы данных, используемой в разрабатываемом приложении будем использовать расширение диаграммы классов UML 2.1, предназначенное для моделирования баз данных. Полученная диаграмма, представленная на рисунке 3.2, будет являться моделью базы данных даталогического уровня.

Физический уровень моделирования БД описывает конкретные таблицы, связи, индексы, методы хранения, настройки производительности, безопасности. Описывать модель можно с помощью средств, уместных для предметной области [13].

Индексы – это специальные структуры, применяемые для ускорения операций взаимодействия с данными. Целесообразно реализовать следующие индексы:

- по названиям проектов, позиций, департаментов, навыков;
- по электронной почте и имени пользователей.

Для идентификации используется специальное поле id, которое внедряется во все сущности.

Остальные настройки будут применяться непосредственно при развёртывании программной системы, поэтому в данном разделе не рассматриваются.

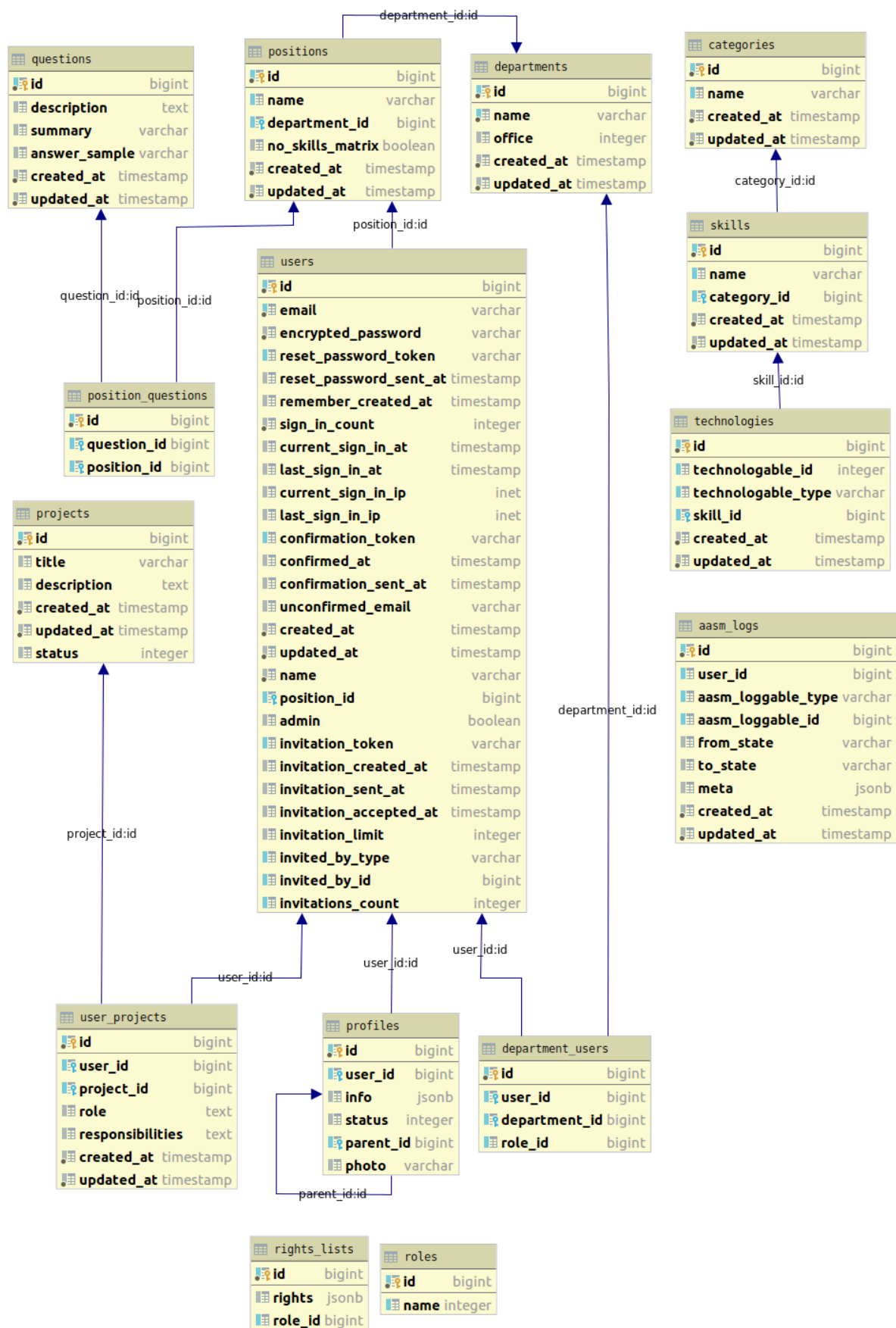


Рисунок 3.2 – Даталогическая модель базы данных программного средства

3.3 Проектирование и разработка серверной части программного средства

3.3.1 Этапы разработки программного средства

Архитектурный стиль разделения функциональности программной системы на уровни поднимает сразу несколько проблем. Их решение в типичном случае заключается в выполнении следующих этапов [11]:

- а) определение стратегии разбиения на уровни;
- б) определение уровней;
- в) распределение функциональности по уровням и компонентам;
- г) уточнение количества уровней: при необходимости некоторые из них объединяются;
- д) установление правил взаимодействия уровней;
- е) идентификация функциональности, которая используется на всех уровнях (cross cutting concerns);
- ж) определение интерфейсов уровней;
- з) выбор стратегии развертывания;
- и) выбор конкретных протоколов взаимодействия.

К данному этапу разработки дипломного проекта выполнены шаги с 3.3.1а по 3.3.1д. Шаг 3.3.1е вследствие использования различных программных средств, применяемых на уровнях, будет выполняться позднее – на соответствующих уровнях.

Таким образом, следующий этап – это определение интерфейсов уровней. СУБД имеет заранее определенный интерфейс, установленный ее разработчиком. Предполагается, что из остальных двух частей программной системы инициатором любых действий будет являться клиентская. Следовательно, единственный межуровневый интерфейс, который следует определить – это интерфейс серверной части.

Веб-приложения, в стиле которых планируется создание клиентской части, обычно запускаются большим числом пользователей. Таким образом очевидна автономность и независимость частей проектируемой программной системы. Исходя из этих соображений, целесообразным является применение сервис-ориентированного архитектурного стиля.

3.3.2 Программный интерфейс серверной части

Предварительным условием для проектирования серверной части приложения является определение его интерфейса в высокоуровневых терминах предметной области. Таким образом, далее приведены методы его API, которые основаны на функциональных требованиях, определенных

в подразделе 2.2:

- метод регистрации пользователя системы, принимающий адрес электронной почты, пароль, повторный пароль;
- метод смены пароля, также принимающий старый пароль, новый пароль, повторный новый пароль;
- метод аутентификации, принимающий пароль и адрес электронной почты;
- метод восстановления пароля, принимающий адрес электронной почты;
- метод приглашения нового пользователя, принимающий адрес электронной почты;
- метод принятия приглашения пользователя, принимающий пароль, повторный пароль;
- метод, возвращающий список пользователей;
- метод, возвращающий список копий профилей;
- метод создания копии профиля;
- метод, возвращающий копию профиля для редактирования;
- метод просмотра копии профиля;
- метод обновления данных копии профиля;
- метод удаления копии профиля;
- метод, возвращающий список статусов профилей;
- метод, возвращающий список навыков;
- метод, возвращающий список ролей системы;
- метод, возвращающий список прав для ролей;
- метод обновления прав;
- метод создания департамента;
- метод просмотра департамента;
- метод обновления департамента;
- метод удаления департамента;
- метод создания вопроса;
- метод просмотра вопроса;
- метод обновления вопроса;
- метод удаления вопроса;
- метод, возвращающий список проектов;
- метод создания нового проекта;
- метод, возвращающий проект для редактирования;
- метод просмотра проекта;
- метод обновления проекта;
- метод удаления проекта;

- метод, возвращающий список позиций;
- метод создания новой позиции;
- метод, возвращающий позицию для редактирования;
- метод просмотра позиции;
- метод обновления позиции;
- метод удаления позиции;
- метод, возвращающий список категорий;
- метод, возвращающий категорию для редактирования;
- метод просмотра категории;
- метод обновления категории;
- метод удаления категории;
- метод назначения менеджеров на департаменты;
- метод, возвращающий менеджеров департаментов;
- метод, возвращающий списки прав;
- метод, возвращающий информацию о текущем пользователе;
- метод просмотра информации о пользователе;
- метод, возвращающий список технологий;
- метод, возвращающий список департаментов;
- метод, возвращающий список вопросов для позиции;
- метод, возвращающий список позиций;
- метод, возвращающий список проектов;
- метод, возвращающий список вопросов;
- метод создания профиля пользователя;
- метод просмотра профиля пользователя;
- метод обновления профиля пользователя;
- метод отправки профиля на модерацию;
- метод генерации резюме;
- метод обновления фотографии профиля.

На рисунке 3.3 приведена схема алгоритма обработки запросов перечисленных методов программного интерфейса серверной части приложения. Далее приведены архитектурные решения, которые были применены при реализации данной части программной системы.

Поскольку в ПС проектируется реализация сложных независимых систем ролей и пользовательских функций, которые находят своё отражение в серверной части приложения, то встаёт вопрос об их сопряжении. А поскольку данное сопряжение вследствие особенностей предметной области является достаточно нетривиальным: часть функций уникальна для некоторых ролей, часть разделяется ими всеми. Поэтому целесообразным является разбиение их на следующие группы:

- а) функции, доступные без аутентификации;
- б) функции, доступные для всех аутентифицированных пользователей;
- в) функции менеджеров;
- г) функции администраторов.

Серверная проверка аутентификации пользователей будет осуществляться с помощью технологии создания и проверки специального токена.

Большое количество блоков, в которых осуществляется доступ к БД, подтверждает назначение всей серверной части приложения: прослойка между клиентским веб-приложением и базой данных с добавлением проверок аутентификации и авторизации.

Реализовав перечисленные методы API, можно будет достигнуть очень важной цели: внутренняя бизнес-логика, логика доступа к данным и сами данные будут надежно защищены.

3.3.3 Выбор протоколов коммуникации

В пункте 3.3.1 приведены этапы, в соответствии с которыми рекомендуется производить проектирование программного системы [11]. Этап 3.3.1з будет рассмотрен позднее вследствие неоднородности и отсутствия схожести в применяющихся технологий между уровнями. Таким образом, следующий актуальный этап – 3.3.1и: выбор конкретных протоколов взаимодействия.

Можно выделить два основных стиля, которые (с необходимыми модификациями в каждом конкретном случае) применяются в информационных системах, содержащих веб-сервисы [11]: REST (Representational State Transfer – протокол передачи состояния представления) и SOAP (Simple Object Access Protocol – простой протокол доступа к объектам). Технически REST является архитектурным шаблоном проектирования, построенным на основе использования простых глаголов (называемых запросами или методами). На практике данный архитектурный стиль применяется в связке с протоколом прикладного уровня модели OSI HTTP. SOAP же является XML-ориентированным протоколом, причем стандарт не специфицирует применяемые протоколы передачи данных.

Основное отличие между данными протоколами состоит в способе манипулирования состоянием. SOAP предусматривает осуществление переходов между различными состояниями с помощью взаимодействия с единственной конечной точкой ввода информации, через которую предоставляется доступ к функциональности сервиса. Протокол REST предполагает использование ограниченного набора операций, которые могут применяться к ресурсам, адресуемым с помощью уникальных адресов URI.

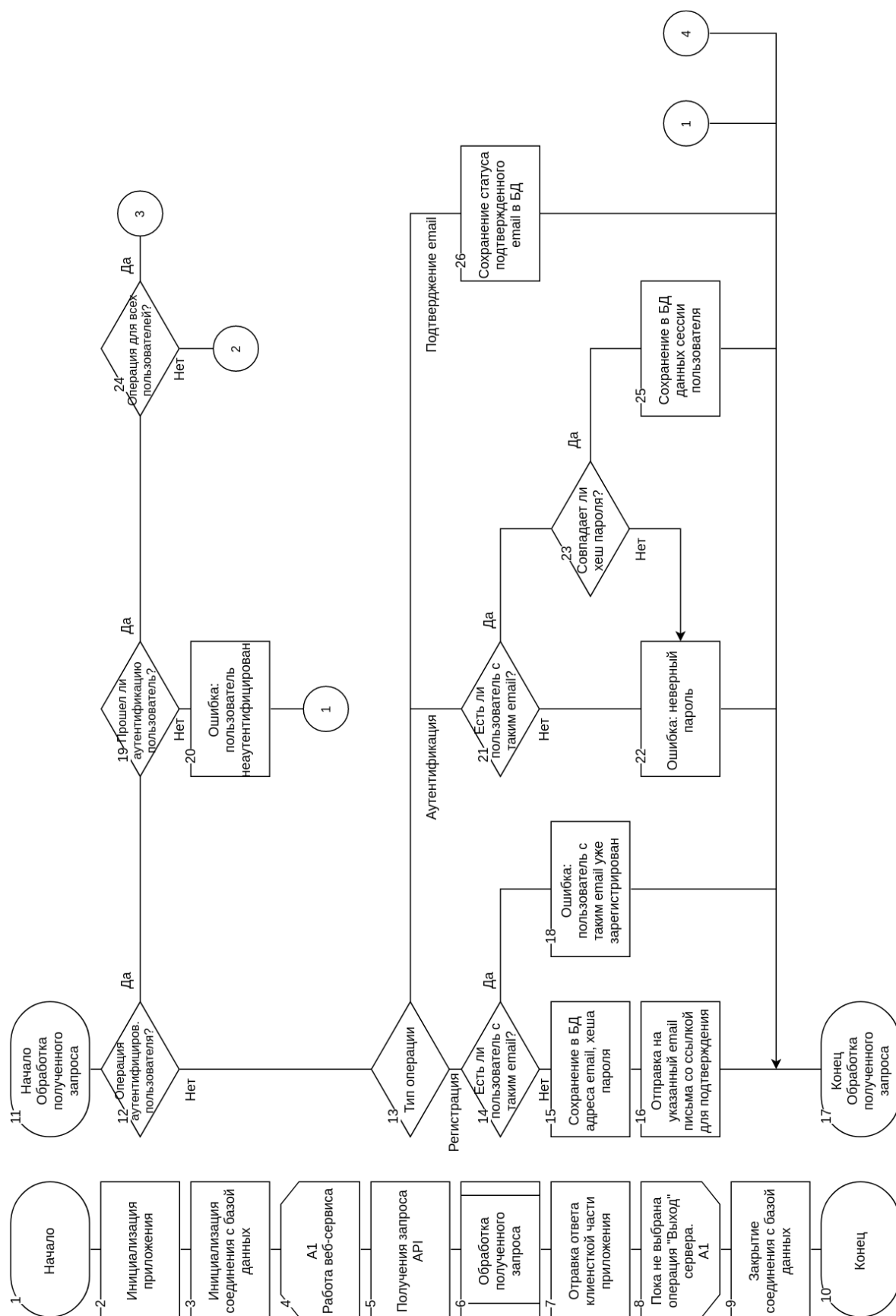


Рисунок 3.3 – Схема программы серверной части программного средства

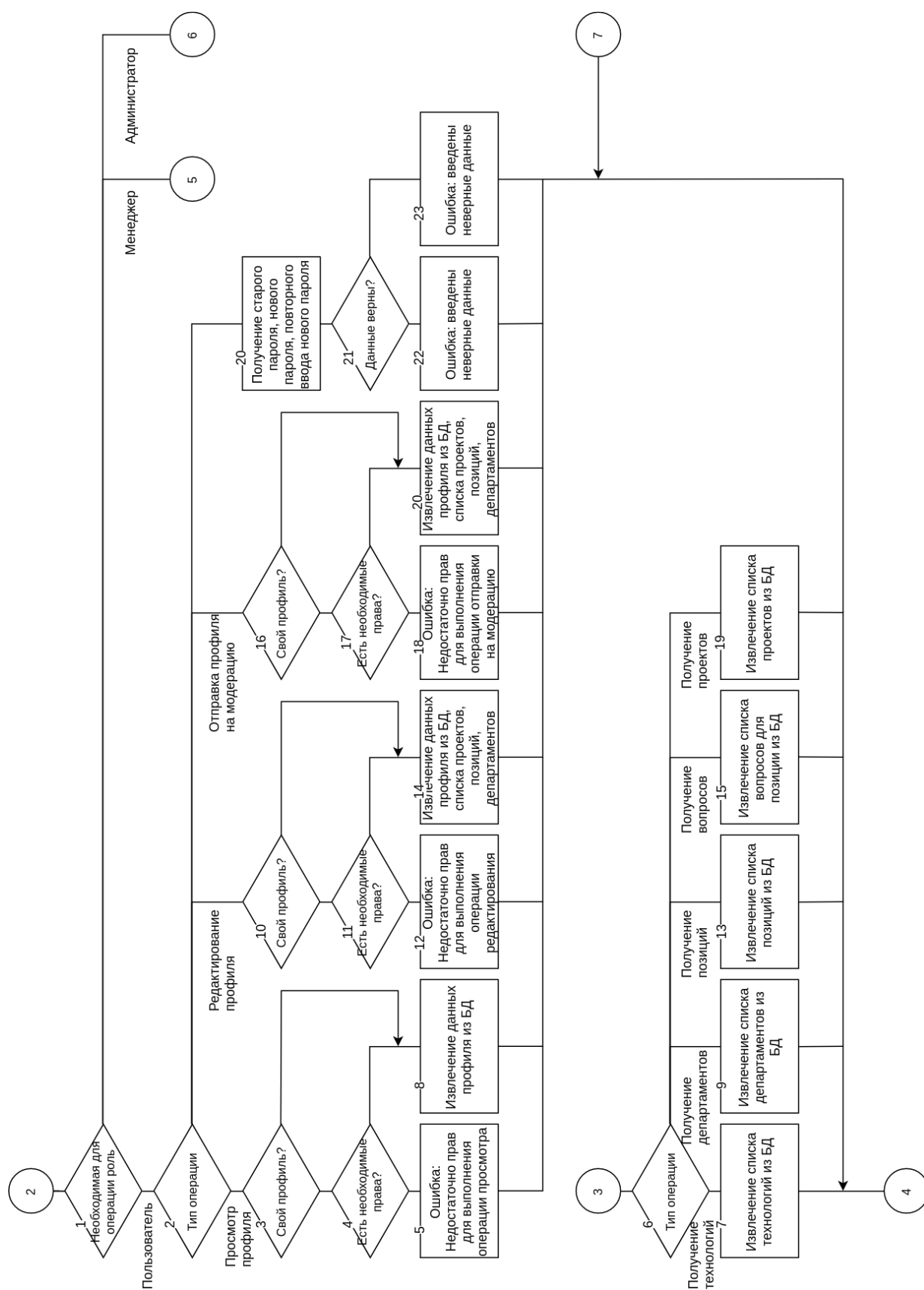


Рисунок 3.3 – Схема программы серверной части программного средства (продолжение)

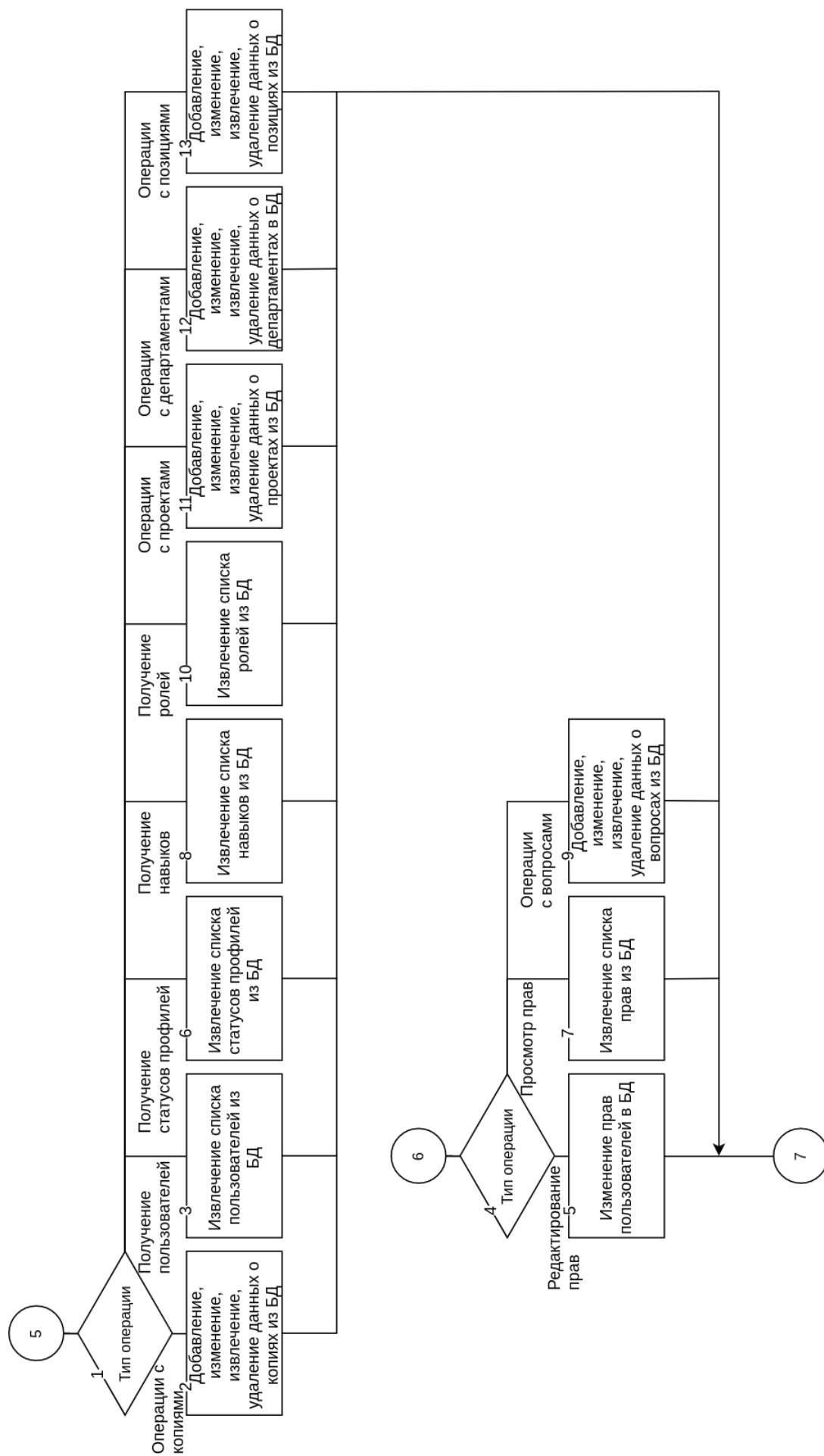


Рисунок 3.3 – Схема программы серверной части программного средства (окончание)

Данный протокол в большей степени соответствует распределенной архитектуре клиент-серверных приложений, поскольку серверная часть публично доступна для множества заведомо неизвестных клиентских приложений. REST обладает свойством отсутствия состояния, что означает, что каждый запрос должен содержать в себе достаточный набор данных для его осуществления. Кроме того, данный протокол не запрещает использование различных форматов передачи данных, например, JSON, что оказывается огромным преимуществом перед SOAP вследствие планируемого использования в клиентской части языка Javascript, который нативно поддерживает данный формат.

3.3.4 Настройка окружения

Перед установкой Rails необходимо проверить, чтобы в системе были установлены необходимые предварительные зависимости. К ним относятся Ruby и PostgreSQL.

Для установки Rails используется команду `gem install`, представленная RubyGems:

```
$ gem install rails
```

Rails поставляется с рядом скриптов, названных генераторами, разработанных для облегчения жизни разработчика, создавая все, что необходимо для начала работы над определенной задачей. Одним из них является генератор нового приложения, предоставляющий основу приложения Rails, таким образом, не нужно писать его самостоятельно.

Для использования этого генератора создания приложения, необходимо в терминале выполнить следующую команду:

```
$ rails new diploma
```

Это создаст приложение на Rails с именем Diploma в директории `diploma` и установит гемы, зависимости от которых упомянуты в Gemfile при использовании `bundle install`.

Как и в большинстве языков программирования, в Ruby можно использовать широкий набор сторонних библиотек. Большая часть из них реализована в форме гема. RubyGems – менеджер пакетов Ruby, созданный для упрощения процесса создания, распространения и установки библиотек.

В директории `diploma` имеется несколько автоматически сгенерированных файлов и папок, задающих структуру приложения на Rails. Рассмотрим подробнее функции каждой папки, которые создает Rails в новом приложении по умолчанию в таблице 3.1:

Таблица 3.1 – Назначение папок, используемых в проекте

Название папки/файла	Назначение
app/	Содержит контроллеры, модели, представления, хелперы, рассыльщики, каналы, задания и стили приложения.
bin/	Содержит Rails скрипты которые стартуют приложение, также директория может содержать другие скрипты которые используются для настройки, обновления, деплоя или запуска.
config/	Конфигурации маршрутов, базы данных приложения, и т.д.
config.ru	Конфигурация Rack для серверов, основанных на Rack, используемых для запуска приложения.
db/	Содержит текущую схему базы данных, а также миграции базы данных.
Gemfile Gemfile.lock	Эти файлы позволяют указать, какие зависимости от гемов нужны для приложения на Rails. Эти файлы используются гемом Bundler.
lib/	Внешние модули для приложения.
package.json	Этот файл позволяет указать, какие зависимости npm необходимы для приложения Rails. Этот файл используется Yarn.
public/	Единственная папка, которая доступна извне как есть. Содержит статичные файлы и скомпилированные стили.
Rakefile	Этот файл находит и загружает задачи, которые могут быть запущены в командной строке. Определенная задача доступна во всех компонентах Rails. Вместо изменения Rakefile, можно добавить свои собственные задачи, добавив файлы в директорию lib/tasks приложения.
test/	Юнит-тесты, и прочий аппарат тестирования
vendor/	Место для кода сторонних разработчиков. В типичном приложении на Rails включает внешние геммы.

Для запуска веб-сервера на машине, необходимо запустить следующую команду из директории diploma:

```
$ bin/rails server
```

Это запустит Puma, веб-сервер, распространяющийся с Rails по умолчанию. Чтобы увидеть приложение в действии, необходимо открыть окно

браузера и пройти по адресу `http://localhost:3000`.

3.3.5 Конструирование серверной части приложения

Наконец, по завершению этапов проектирования и подготовки можно приступить к собственно созданию исходных кодов приложения. Ключевые фрагменты кода, описание которых приведено в данном пункте, приведены в приложении А.

Чтобы добавить в Rails логику, нужно создать, как минимум, контроллер и представление.

Назначением контроллера является получение определенных запросов к приложению. Роутинг решает, какой контроллер получит какие запросы. Часто имеется более одного маршрута к каждому контроллеру, и различные маршруты могут быть обработаны различными экшнами. Назначением каждого экшна является сбор информации для предоставления ее на представление.

Назначением представления является отображение этой информации в удобочитаемом формате. Необходимо отметить важное различие, что местом, в котором собирается информация, является контроллер, а не представление. Представление должна только лишь отображать эту информацию. По умолчанию шаблоны представления пишутся на языке, названном eRuby (Embedded Ruby), который конвертируется циклом запросов в Rails до отправки пользователю.

Для создания нового контроллера, нужно запустить генератор "controller" и сказать, что необходим контроллер с именем "Welcome" с экшном по имени "index" вот так:

```
$ bin/rails generate controller Welcome index
```

Контроллер – это просто класс, унаследованный от ApplicationController. В этом классе необходимо определить методы, которые станут экшнами для этого контроллера. Эти экшны будут выполнять операции CRUD с сущностями в системе.

Модели в Rails используют имя в единственном числе, а их соответствующая таблица в базе данных – имя во множественном числе. Rails предоставляет генератор для создания моделей, которым пользуются большинство разработчиков на Rails для создания новых моделей. Для создания новой модели, необходимо эту команду в терминале:

```
$ bin/rails generate model Article title:string text:text
```

С помощью этой команды Rails сообщают, что необходима модель Article с атрибутом title строкового типа и атрибутом text текстового типа. Эти атрибуты автоматически добавятся в таблицу articles и привяжутся к модели Article.

bin/rails generate model создал файл миграции базы данных в директории db/migrate. Миграции – это класс Ruby, разработанный для того, чтобы было просто создавать и модифицировать таблицы базы данных. Rails использует команды rake для запуска миграций, и возможна отмена миграции после того, как она была применена к базе данных. Имя файла миграции включает временную метку, чтобы быть уверенным, что они выполняются в той последовательности, в которой они создавались.

чтобы запустить миграцию, необходимо использовать команду bin/rails:

```
$ bin/rails db:migrate
```

Rails выполнит эту команду миграции и сообщит, что он создал таблицу Articles.

Роутер Rails распознает URL и направляет его в экшн контроллера или в приложение Rack. Он также может генерировать пути и URL, избегая необходимость жестко прописывать строки в представлениях.

Маршруты для приложения или engine располагаются в файле config/routes.rb и обычно выглядят так:

```
Rails.application.routes.draw do
  Rails.application.routes.draw do
    resources :products, only: [:index, :show]
  end
  resource :basket, only: [:show, :update, :destroy]
  resolve("Basket") route_for(:basket)
end
```

Ресурсный роутинг позволяет быстро объявлять все общие маршруты для заданного ресурсного контроллера. Вместо объявления отдельных маршрутов для экшнов index, show, new, edit, create, update и destroy, ресурсный маршрут объявляет их одной строчкой кода.

Таким образом, разработаем исходные коды серверной части приложения.

3.4 Проектирование и разработка клиентской части программного средства

Задачи клиентской части приложения включают отображение пользовательского интерфейса и обработку действий пользователя. Типичными этапами его проектирования являются следующие [11]:

- идентификация типа клиентской части приложения, которое удовлетворяет установленным требованиям. Осуществление данного этапа осуществлено в подразделе 3.1;
- выбор технологии пользовательского интерфейса. Осуществляется на основе анализа требуемой для реализации функциональности;
- проектирование UI. Хорошей практикой является реализация модульности, а также принципа разделения ответственности компонентов;
- определение стратегии и протоколов обмена информацией между уровнями. Поскольку рассматриваемый уровень является самым верхним, а его протоколы связи с серверной частью приложения были рассмотрены в пункте 3.3.3, то данный вопрос считается решенным и рассматриваться не будет.

Ранее был проведен анализ и осуществлен выбор целевой платформы для реализации клиентской части приложения, был выбран язык программирования. Однако, существует огромное число специализированных технологий и фреймворков по созданию пользовательских интерфейсов. Представляется целесообразным провести уточнение средств разработки.

3.4.1 Уточнение выбора технологий программирования

Одной из широко используемых в настоящее время библиотек по созданию интерактивных интерфейсов веб-приложения является библиотека React. Выбор в ее пользу был осуществлен вследствие наличия опыта по ее использованию у членов команды. Особенности данной библиотеки являются следующие [22]:

- кроссплатформенность. Переиспользование существующего кода возможно даже на других платформах благодаря проекту React Native;
- декларативность: использование элементов (стандартных для React объектов, которые представляют собой HTML-теги) и компонентов (объекты, создаваемые разработчиком);
- JSX: техника создания и использования компонентов с помощью HTML-подобного синтаксиса, который применяется прямо в Javascript коде;
- виртуальный DOM: дерево React элементов, которое отрисовывается в браузере, причем изменения в нём не требуют полной перерисовки всего интерфейса. Позволяет значительно улучшить быстродействие при использо-

вании библиотеки.

Строго говоря, использование JSX является опциональным и может как не использоваться при разработке с React, так и использоваться при разработке с помощью других библиотек. Однако, он значительно упрощает исходный код компонентов. Например, следующий React код

```
return <div>Hello{this.props.children}</div>;
```

после компиляции будет преобразован в следующий

```
return React.createElement(  
  "div",  
  null,  
  "Hello ",  
  this.props.children  
);
```

Лаконичность использования JSX очевидна. Некоторым кажется сомнительным данный подход, в котором смешивается исполняемый код и код разметки. Однако, данное смешение формирует исходный код представления. Кроме того, в других фреймворках используются специальные конструкции по реализации, например, циклов, условий, в то время как при использовании JSX нужды в новых конструкциях нет, ведь используются стандартные операторы языка Javascript.

Одна из классификаций компонентов React предполагает их разделение на pure (простые) и stateful (имеющие внутреннее состояние) [23]. Простые компоненты, реализованные в виде классов, унаследованных от `React.Component`, переопределяют метод `render()`, который принимает некоторый набор исходных данных и возвращает элемент, предназначенный для отображения. Компоненты другого класса имеют внутри себя некоторые данные, образующие их состояние. Чтобы React узнал, что данные изменились, и осуществил перерисовку измененных компонентов, требуется использовать специальный метод `this.setState(...)`.

Для упрощения управления состоянием компонентов могут применяться различные техники. Одной из них является использование специальной библиотеки `Redux`. Он предназначен для управления потоком данных и их связями. В React связь между двумя компонентами, не имеющими отношения родитель-потомок (дочерний элемент), не рекомендуется. React обращает внимание, что если такое сделать (создать связь), можно вполне создать собственную глобальную систему событий по шаблону `Flux`; и именно в этот момент и появляется `Redux`.

С Redux у нас есть хранилище, в котором можно сохранять все состояния приложения. Если в компоненте А происходит изменение состояния, оно затем передается в хранилище, а другие компоненты В и С, которые должны знать об этом изменении состояния в компоненте А, могут получать эту самую информацию об этом изменении из хранилища.

Сравнение работы приведено на рисунке 3.4

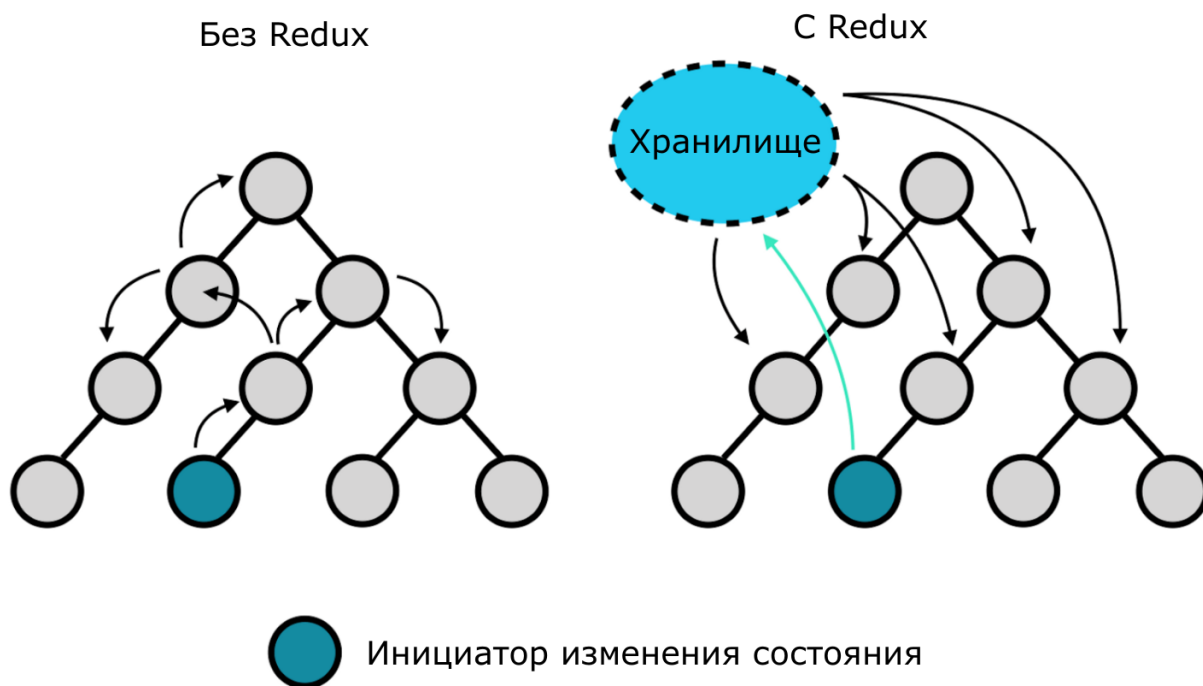


Рисунок 3.4 – Сравнение работы с Redux и без него

Redux состоит из следующих составных частей:

- действия(actions) – Это просто события, созданные с помощью функций для отправки данных из приложения в хранилище. Данные могут быть отправлены различными способами, такими как отправка формы, вызов API или обычного взаимодействия с пользователем. Каждое действие в Redux имеет свойство type, которое описывает тип действия, а также «важную» информацию, отправляемую в хранилище;

- редюсеры (reducer). Поскольку Redux не позволяет приложению вносить изменения в состояния компонентов, сохраняемых в хранилище, он использует dispatch() для этого. Функция dispatch() просто указывает на намерение изменить данное состояние, но на самом деле не меняет его, вот почему и нужны редюсеры (Reducer). Редюсеры (Reducer) – это функции, которые считывают из хранилища текущее состояние приложения через отправленное действие, а затем возвращают новое состояние;

- хранилище похоже на сердце фреймворка Redux. Это единственный источник истины, в котором находятся все состояния приложения и который

обеспечивает доступ к состоянию с помощью нескольких методов, действий отправки данных и регистрации записей. Любое отправленное действие возвращает новое состояние данных в хранилище с помощью редюсеров [24].

Однако, Redux представляет собой только библиотеку. Ограничения на архитектуру приложения не накладываются, однако сохраняется необходимость ее реализации программистом. Одним из вариантов является разбиение всех компонентов на контейнерные и презентационные [25]. Особенности создания презентационных компонентов следующие:

- их задача заключается в определении, как должны выглядеть элементы;
- не зависят от других частей приложения;
- получают данные исключительно в качестве входных параметров;
- не изменяют данные;
- редко имеют собственное состояние (в таком случае это состояние UI, а не собственно данные);
- обычно содержат JSX разметку и имеют относящиеся к ним стили.

В это же время особенности контейнерных компонентов заключаются в следующем:

- их задача заключается в определении, как элемент должны работать;
- предоставляют данные и методы их обработки другим компонентам;
- как правило имеют некоторые данные в виде состояния;
- обычно не содержат JSX разметки и никогда не имеют стилей.

Использование данного подхода имеет следующие преимущества:

- достигается выполнение принципа разделения ответственности;
- появляется возможность легкого переиспользования компонентов с различными источниками данных;
- одни и те же презентационные компоненты могут использоваться повсеместно в приложении;
- презентационные компоненты формируют «палитру». Их внешний вид может изменяться без влияния на логику приложения.

Еще один важный момент относительно проектирования UI, помимо содержания и интерактивных пользовательских взаимодействий, касается стилизации внешнего вида компонентов. Существует несколько подходов по ее исполнению [26]: использование внутренних (inline) и внешних стилей.

Первый подход имеет множество недостатков:

- невозможно использование медиа-запросы, чтобы, например, реагировать на изменения ширины экрана или отличать мобильное устройство от настольного компьютера;
- невозможно использование псевдоклассов и псевдоэлементов, напри-

мер: `:hover`, `:active`, `:before`, `:after`;

- значительное увеличение размера разметки и снижение производительности;

- невозможно переопределение стилей по более сложным селекторам.

Тем не менее, Facebook, как компания разработчик React, несмотря на описанные недостатки, выступает за использование `inline` стилей. Причиной этому является стремление обеспечить единообразие с React Native, поскольку там возможность стилизации обеспечивается только данным подходом.

Второй подход – использование внешних стилей – является более традиционным для веб-разработки. Его эволюционное развитие включает несколько этапов:

- создание одного файла `.css` со стилями, который подключается один раз глобально в главный `html` файл. В React компонентах используется тег `className` со значениями имен классов из этого файла;

- разбиение единого файла стилей на множество файлов в соответствии с реализованными компонентами. В файлы компонентов подключаются лишь те стили, которые там необходимы. Преимущества данного подхода заключаются в простоте поиска файла, в который необходимо внести изменения, и в простоте удаления компонентов и соответствующих им файлов стилей;

- использование различных препроцессоров стилей, таких как SASS/SCSS, LESS и прочих.

Использование SASS по сравнению с обычным CSS предоставляет следующие преимущества [27]:

- переменные, в которых можно хранить значения цветов, шрифтов, а также любые другие значения;

- вложенность, что приводит к большей наглядности файлов стилевых таблиц за счет соответствия иерархии HTML кода;

- фрагментирование, то есть обеспечение модульности;

- импортирование других файлов стилей, которое, в отличие от CSS, вместо создания новых HTTP запросов подставляет указанный файл в тот, где он вызывается, таким образом на выходе получается единственный файл стилей;

- примеси (`mixins`), которые представляют собой группы деклараций, используемые по несколько раз;

- наследование, позволяющее приносить наборы свойств от одного селектора к другому;

- использование математических операторов.

Таким образом, на основании проведенного уточнения технологий для

использования выбираем следующие: React в связке с Redux и SCSS, кроме того, для разметки в коде повсеместно будет использоваться JSX.

3.4.2 Проектирование клиентской части приложения

Далее рассмотрим вопрос взаимодействия пользователя с клиентской частью приложения. Известно, что удобство пользования программным средством может во многом определять успешность проекта в целом [3].

Схема работы клиентской части программной системы представлена на рисунке 3.5. Среди ее особенностей можно выделить в высокой степени соответствие диаграмме прецедентов, которая была составлена и рассмотрена в пункте 2.1.1. Кроме того, данный чертеж представляет собой отображение чертежа схемы серверной части с отличием в том, что там делается акцент на взаимодействие с базой данных, а в данном чертеже – на отображении данных и интерактивном взаимодействии с пользователем.

Таким образом, составленная схема клиентской части ПС будет использована при разработке навигации (routing) по страницам веб-приложения.

3.4.3 Конструирование клиентской части приложения

Наконец, по завершению этапов проектирования и подготовки можно приступить к собственно созданию исходных кодов приложения. Ключевые фрагменты кода, описание которых приведено в данном пункте, приведены в приложении А.

Для начала необходимо создать корневой файл index.html. Ключевая его особенность состоит в следующем теге

```
<div id="root"></div>
```

Несмотря на то, что он объявляется пустым, именно в него библиотека React подставит всё приложение.

Кроме этого, в данном файле подключается основная зависимость приложения – непосредственно сама библиотека React и библиотека по управлению виртуальным деревом элементов браузера, а также пакет bundle.js, в который будет собран весь созданный исходный код. Помимо библиотек, подключаются некоторые файлы стилей, необходимые для корректной работы некоторых компонентов, например, специальных шрифтов, предоставляющих возможность использования большого количества специальных иконок.

Алгоритм генерации резюме пользователя представлен на рисунке 3.6.

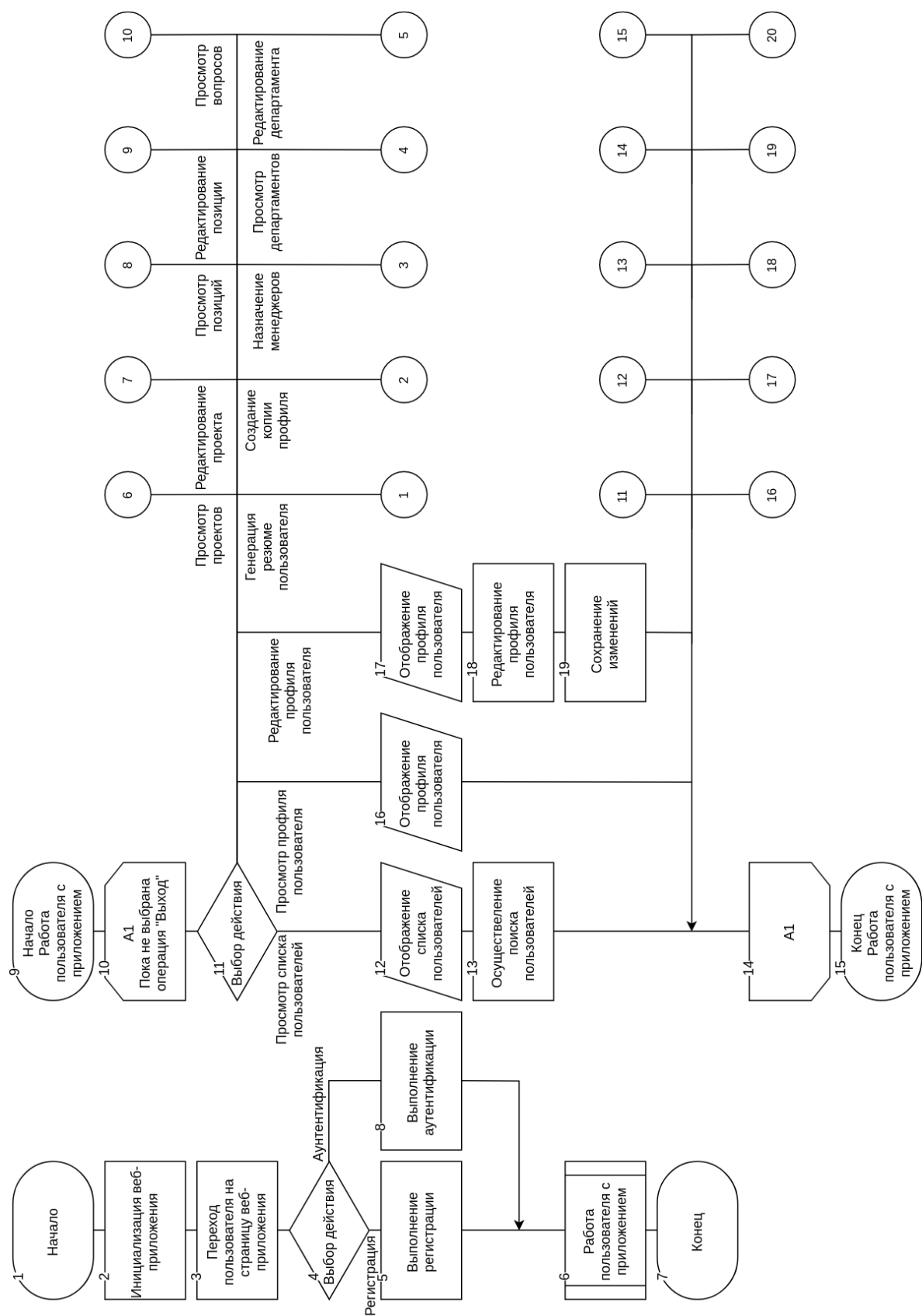


Рисунок 3.5 – Схема программы клиентской части программного средства

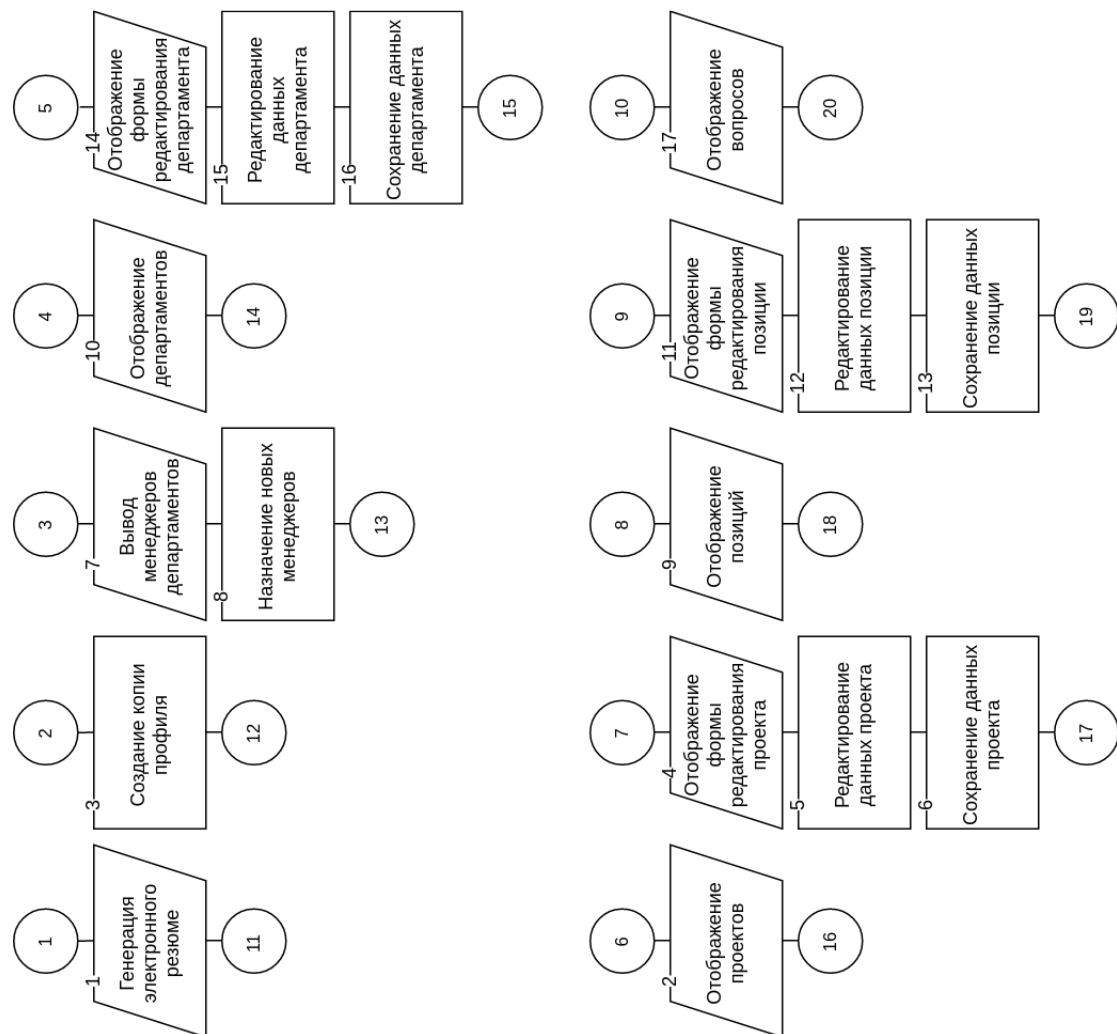


Рисунок 3.5 – Схема программы клиентской части программного средства (окончание)

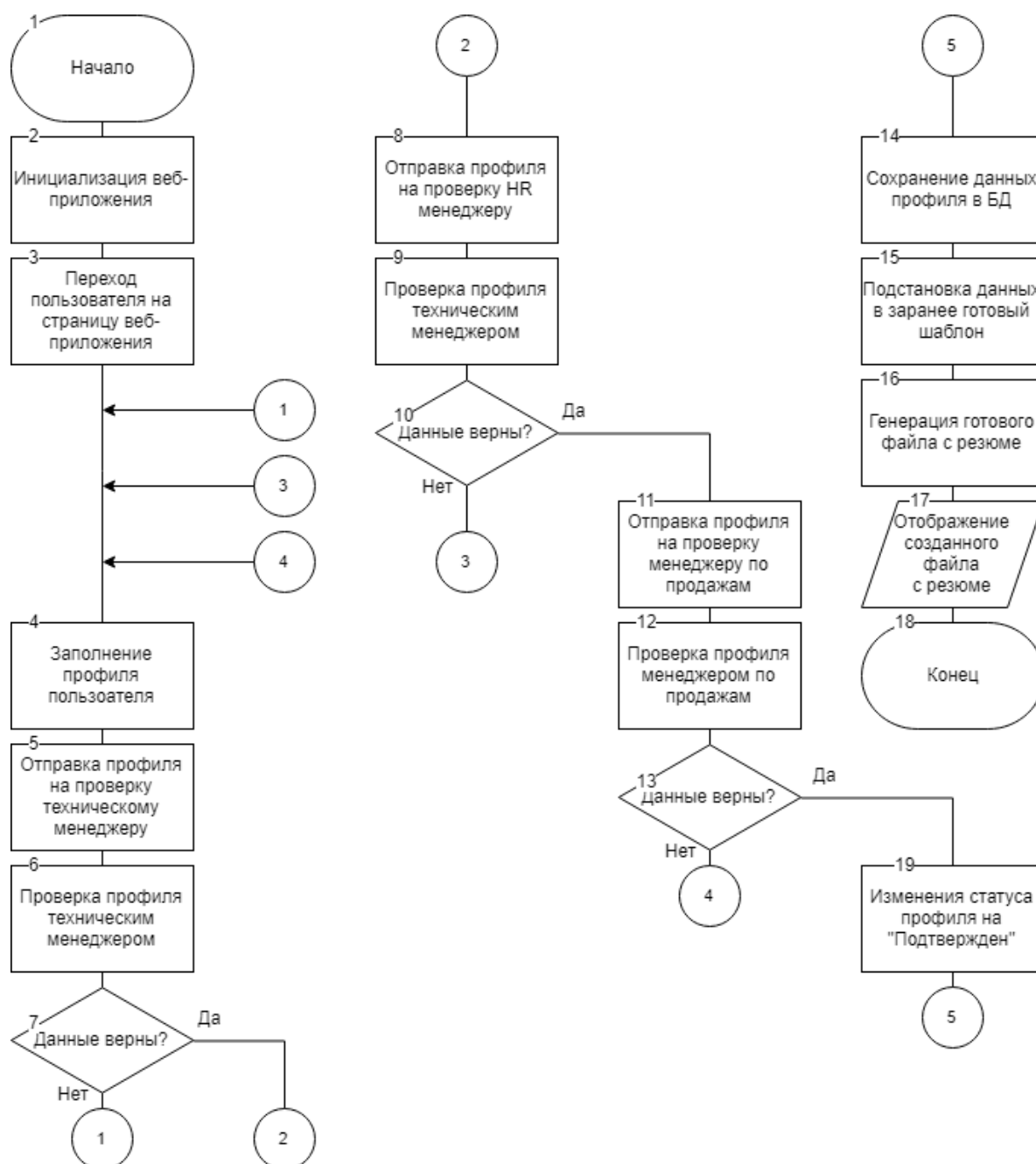


Рисунок 3.6 – Алгоритм генерации резюме пользователя

3.5 Развертывание программного средства

После выявления, а также завершения проектирования всех компонентов программного средства появляется вопрос о планировании развертывания всей системы. Необходимо составить описание требуемых аппаратно-программных комплексов, которые понадобятся для обеспечения функционирования распределенного приложения. Для этих целей целесообразным выглядит составление диаграммы развертывания стандарта UML 2.1.

Данная диаграмма представлена на рисунке 3.7. Она отражает следующие особенности развертывания:

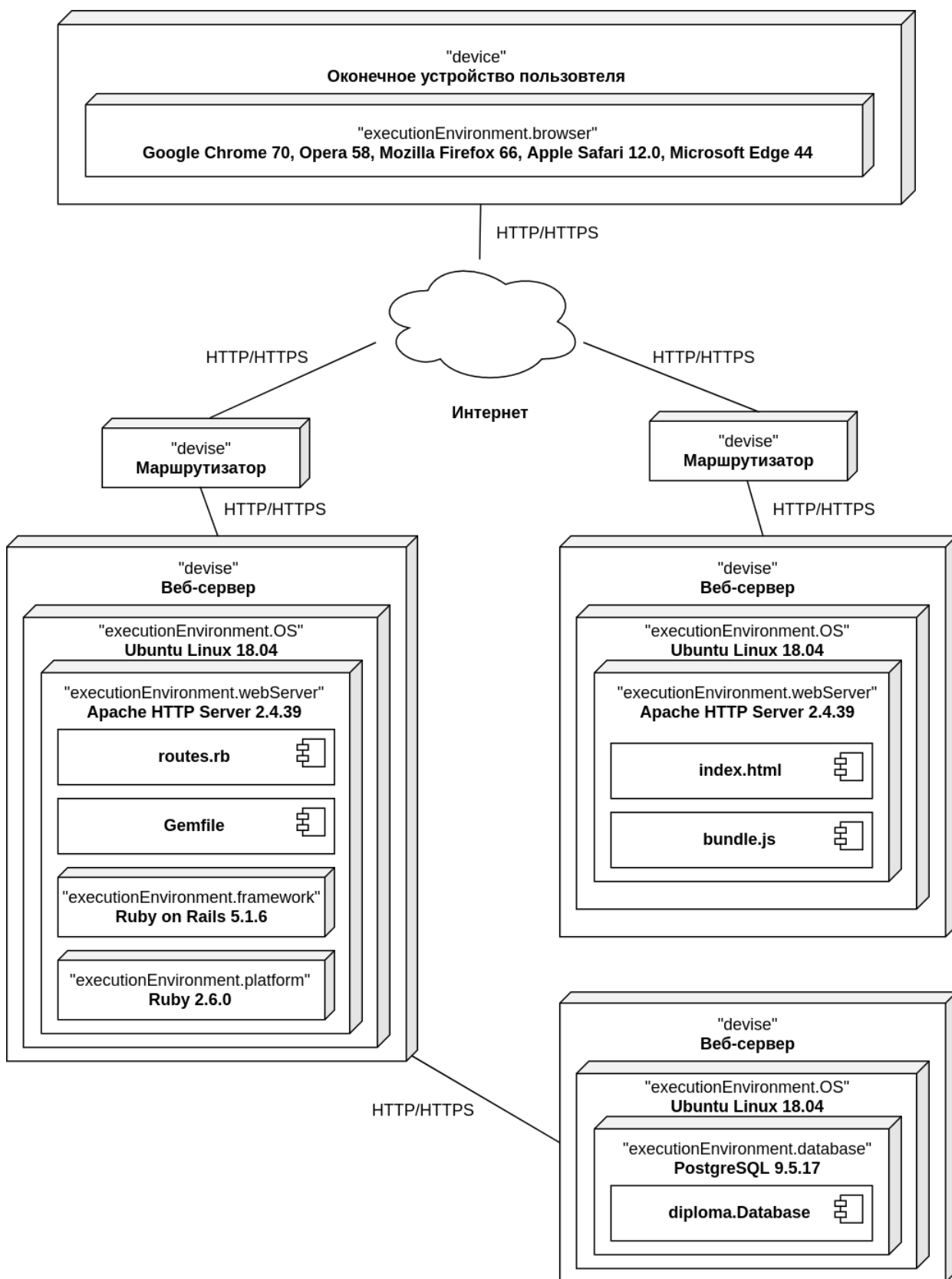


Рисунок 3.7 – Диаграмма развертывания ПС

– на узле оконечного устройства в качестве среды выполнения перечислен список браузерных программных средств, с помощью которых можно использовать клиентскую часть приложения;

- в качестве операционной системы для сервера клиентской части перечислен список поддерживаемых веб-сервером ОС;
 - при необходимости Apache HTTP Server может быть заменен другим HTTP-сервером;
 - для серверной части программного средства и базы данных показано их развертывание на отдельных узлах. При самом развертывании в зависимости от условий поставщика вычислительных мощностей данные элементы программной системы могут быть объединены на одном узле;
 - в свою очередь, помимо упрощения, возможно и усложнение схемы развертывания, например, база данных будет развернута на нескольких узлах. Тем не менее, все узлы должны удовлетворять отображенным условиям;
 - все серверы: и клиентской части, и серверной, и базы данных, – могут быть физически расположены в различных дата-центрах;
 - предполагается, что пользовательское оконечное устройство значительно удалено от серверов программной системы, доступ осуществляется через сеть Интернет, что и упрощенно показано на диаграмме.
- Таким образом, после развертывания программного средства пользователи могут уже начать им пользоваться.

4 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

Тестирование программного обеспечения – процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта [28]. Вот уже несколько десятков лет его стабильно включают в планы разработки как одна из основных работ, причем выполняемая практически на всех этапах проектов. Важность своевременного выявления дефектов подчеркивается выявленной эмпирически зависимостью между временем допущения ошибки и стоимостью ее исправления: график данной функции круто возрастает.

Тестирование можно классифицировать по очень большому количеству признаков. Основные виды классификации включают следующие [28]:

- а) по запуску кода на исполнение:
 - 1) статическое тестирование – без запуска программного средства;
 - 2) динамическое тестирование – с запуском;
- б) по степени автоматизации:
 - 1) ручное тестирование – тестовые случаи выполняет человек;
 - 2) автоматизированное тестирование – тестовые случаи частично или полностью выполняет специальное инструментальное средство;
- в) по принципам работы с приложением:
 - 1) позитивное тестирование – все действия с приложением выполняются строго в соответствии с требованиями без недопустимых действий или некорректных данных;
 - 2) негативное тестирование – проверяется способность приложения продолжать работу в критических ситуациях недопустимых действий или данных.

В данном разделе проведем динамическое ручное тестирование. Его целью является подтверждение соответствия работы программного средства установленным в начале разработки требованиям. Успешное выполнение приведенных в данном разделе тестовых случаев должно подтвердить работоспособность программного средства в основных сценариях использования, а также устойчивость к неверным входным данным. В таблице 4.1 приведен список тестовых случаев, относящихся к позитивному тестированию, в таблице 4.2 – к негативному.

Таблица 4.1 – Тестовые случаи позитивного тестирования

Модуль (экран)	Описание тестового случая	Ожидаемые результаты	Тестовый случай пройден?
1	2	3	4
Аккаунт	<p>1. Регистрация. Предусловие: необходим существующий ящик электронной почты.</p> <p>1) Нажать кнопку «Регистрация» на главной странице ПС.</p> <p>2) Ввести корпоративный адрес электронной почты.</p> <p>3) Ввести пароль 12345678.</p> <p>4) Ввести пароль из предыдущего пункта в поле подтверждения пароля.</p> <p>5) Нажать кнопку «Зарегистрироваться».</p> <p>6) Проверить ящик электронной почты, дожидаясь получения электронного письма.</p> <p>7) Перейти по ссылке из полученного письма.</p>	Отображается страница регистрации. На указанный адрес электронной почты приходит письмо со ссылкой. При переходе по ссылке появляется сообщение «Аккаунт подтвержден».	Да
Аккаунт	<p>2. Аутентификация. Предусловие: необходим зарегистрированный в системе аккаунт.</p> <p>1) Ввести адрес электронной почты и пароль аккаунта.</p> <p>2) Нажать кнопку «Войти».</p>	Отображается страница аутентификации. По нажатию кнопки «Войти» открывается страница заполнения профиля пользователя.	Да

Продолжение таблицы 4.1

1	2	3	4
Профиль	3. Заполнение профиля. Предусловие: необходим аутентифицированный в системе аккаунт. 1) Ввести данные для заполнения профиля. 2) Нажать кнопку «отправить на проверку».	Отображается страница профиля. Профиль отображается для проверки технического менеджера. Появляется всплывающее сообщение.	Да
Аккаунт	4. Проверка профиля менеджера. Предусловие: необходим аутентифицированный в системе аккаунт, профиль, требующий проверки, соответствующие права пользователя. 1) Открыть профиль, требующий проверки. 2) Проверить правильность заполнения профиля. 3) Если необходимо, внести правки. 4) Нажать кнопку «отправить на проверку».	Отображается страница профиля. Профиль отображается для проверки следующему менеджеру. Появляется всплывающее сообщение. Меняется статус профиля на тот, какая проверка требуется.	Да
Менеджеры	5. Назначение менеджеров. Предусловие: необходим аутентифицированный в системе аккаунт, права администратора. 1) Выбрать из выпадающего списка менеджеров для соответствующих департаментов. 2) Нажать кнопку «Сохранить».	Отображается страница назначения менеджеров. Выбранные пользователи назначаются менеджерами департаментов. Появляется всплывающее сообщение.	Да

Продолжение таблицы 4.1

1	2	3	4
<p>Департаменты</p>	<p>6. Создание департамента. Предусловие: необходим аутентифицированный в системе аккаунт, соответствующие права пользователя. 1) Нажать на кнопку «Создать департамент». 2) Ввести название департамента. 3) Выбрать тип офиса для нового департамента из выпадающего списка. 4) Нажать кнопку «Создать новый департамент».</p>	<p>Отображается страница департаментов. В списке появится созданный департамент. Появляется всплывающее сообщение.</p>	<p>Да</p>
<p>Проекты</p>	<p>7. Создание проекта. Предусловие: необходим аутентифицированный в системе аккаунт, соответствующие права пользователя. 1) Нажать на кнопку «Создать проект». 2) Ввести название департамента. 3) Ввести описание проекта. 4) Выбрать статус проекта. 5) Выбрать технологии, используемые на проекте. 6) Нажать кнопку «Добавить пользователя». 7) Выбрать пользователя из списка. 8) Ввести его роль и зону ответственности 9) Нажать кнопку «Создать новый проект».</p>	<p>Отображается страница созданного проекта. Появляется всплывающее сообщение.</p>	<p>Да</p>

Продолжение таблицы 4.1

1	2	3	4
Позиции	<p>8. Создание позиции.</p> <p>Предусловие: необходим аутентифицированный в системе аккаунт, соответствующие права пользователя.</p> <ol style="list-style-type: none"> 1) Нажать на кнопку «Создать позицию». 2) Ввести название позиции. 3) Выбрать департамент для позиции. 4) Выбрать технологии для позиции. 5) Нажать кнопку «Создать новую позицию». 	Отображается страница позиции. Появляется всплывающее сообщение.	Да
Вопросы	<p>9. Создание вопроса.</p> <p>Предусловие: необходим аутентифицированный в системе аккаунт, соответствующие права пользователя.</p> <ol style="list-style-type: none"> 1) Нажать на кнопку «Создать вопрос». 2) Ввести вопрос. 3) Ввести краткое описание для резюме. 4) Ввести пример ответа. 5) Выбрать позиции, для которых будет доступен создаваемый вопрос. 6) Нажать кнопку «Создать новый вопрос». 	Отображается страница вопроса. Появляется всплывающее сообщение.	Да
Профиль	<p>10. Генерация резюме.</p> <p>Предусловие: необходим аутентифицированный в системе аккаунт, соответствующие права пользователя.</p> <ol style="list-style-type: none"> 1) Открыть профиль пользователя. 2) Нажать на кнопку «Генерировать резюме».. 	Отображается сгенерированная страница резюме.	Да

Таблица 4.2 – Тестовые случаи негативного тестирования

Модуль (экран)	Описание тестового случая	Ожидаемые результаты	Тестовый случай пройден?
1	2	3	4
Аккаунт	<p>1. Повторная регистрация одного email.</p> <p>Предусловие: необходим существующий ящик электронной почты.</p> <ol style="list-style-type: none"> 1) Произвести регистрацию в системе. 2) Подтвердить email с помощью ссылки, полученной в электронном письме, отправленном на указанный адрес. 3) Выйти из системы. 4) Произвести регистрацию с тем же email. 	Регистрация производится успешно, письмо приходит, при переходе по ссылке отображается сообщение об успешности общения. При попытке регистрации с тем же email появляется сообщение о невозможности регистрации.	Да
Аккаунт	<p>2. Аутентификации с неправильными данными.</p> <p>Предусловие: необходимо наличие зарегистрированного пользователя.</p> <ol style="list-style-type: none"> 1) Открыть главную страницу приложения. 2) В поле email ввести адрес электронной почты зарегистрированного пользователя. 3) В поле пароля ввести заведомо неверный пароль. 	Во всех случаях отображается одинаковое сообщение о неверном email или пароле.	Да

Продолжение таблицы 4.2

1	2	3	4
Аккаунт	<p>3. Аутентификация без подтверждения.</p> <p>Предусловие: необходим существующий адрес электронной почты.</p> <p>1) Произвести регистрацию в системе с произвольным паролем и существующим адресом электронной почты.</p> <p>2) Открыть страницу входа</p> <p>3) Ввести указанные при регистрации email и пароль.</p> <p>4) Нажать кнопку «Вход».</p>	Регистрация происходит успешно. При аутентификации появляется сообщение "Данный аккаунт не подтвержден".	Да
Менеджеры	<p>4. Назначение не всех менеджеров.</p> <p>Предусловие: необходимо наличие зарегистрированного пользователя, обладающего соответствующими правами.</p> <p>1) Открыть страницу входа</p> <p>2) Ввести указанные при регистрации email и пароль.</p> <p>3) Нажать кнопку «Вход».</p> <p>4) Открыть страницу назначения менеджеров.</p> <p>5) Выбрать не всех менеджеров для департаментов.</p> <p>6) Нажать кнопку «Сохранить».</p>	Аутентификация происходит успешно. При назначении менеджеров появляется всплывающее сообщение об ошибке, изменения не сохраняются.	Да

Продолжение таблицы 4.2

1	2	3	4
<p>Профиль</p>	<p>5. Заполнение личных данных неверной информацией. Предусловие: необходим существующий адрес электронной почты.</p> <ol style="list-style-type: none"> 1) Произвести регистрацию в системе с произвольным паролем и существующим адресом электронной почты. 2) Подтвердить адрес электронной почты путем перехода по ссылке из письма. 3) Аутентифицироваться в системе. 4) Нажать кнопку «Редактировать профиль». 5) Проверить, чтобы поля имени и фамилии были пустыми. 6) Оставить секцию ответа на вопросы пустой. Не заполнять данные об уровне английского. 7) Нажать кнопку «Сохранить». 	<p>Регистрация проходит успешно. На указанный адрес электронной почты приходит письмо с подтверждающей ссылкой. Аутентификация происходит успешно. Открывается страница профиля. Открывается страница редактирования профиля. Появляется сообщение "Имя и фамилия не могут быть пустыми". Подсветят поля ответа на вопросы, уровень английского языка.</p>	<p>Да</p>

5 РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА

В данном разделе приведены основные сведения по работе с программным средством.

Приложение данного дипломного проекта (а точнее, его клиентская часть) не требует установки и настройки на конечных устройствах пользователя, поскольку представляет собой веб-приложение. Как и было заявлено в требованиях, для корректной работы программного средства необходим один из следующих браузеров с соответствующей минимальной версией:

- Google Chrome 70;
- Opera 58;
- Mozilla Firefox 66;
- Apple Safari 12.0;
- Microsoft Edge 44.

Далее рассмотрены основные функции, предоставляемые приложением пользователям.

После авторизации открывается главная страница приложения, представленная на рисунке 5.1. Весь профиль заполняется с помощью формы на 4 страницы. Результат после заполнения каждой страницы сохраняется.

The screenshot displays the SumatoSoft web application's profile creation interface. On the left, a sidebar shows the user's name 'Dmitry Golubko' and title 'Backend / Full-Stack Developer' next to a group photo. The main area features a progress bar at the top with five steps: Step 1 (Personal Information), Step 2 (Previous Experience), Step 3 (SumatoSoft Experience), Step 4 (Skills Matrix), and Step 5 (Profile Preview). The 'Personal Information' form contains input fields for Name (filled with 'Dmitry'), Surname (filled with 'Golubko'), Date of birth (filled with '02/14/1994'), Location (filled with 'Minsk'), Email (filled with 'd.golubko@sumatosoft.com'), and Phone number (filled with '+375293240378'). A 'Choose File' button is present for the photo, and a 'Photo preview' section shows a group photo. Below the form is an 'Education' section with a button 'Add University/Courses' and a dropdown for 'Level of English' (filled with 'Upper Intermediate (B2)'). At the bottom, there is a 'Professional Profile' section with a 'Personment' field.

Рисунок 5.1 – Экран заполнения профиля

Если авторизоваться в качестве администратора, то будет доступно меню функций, показанное на рисунке 5.2

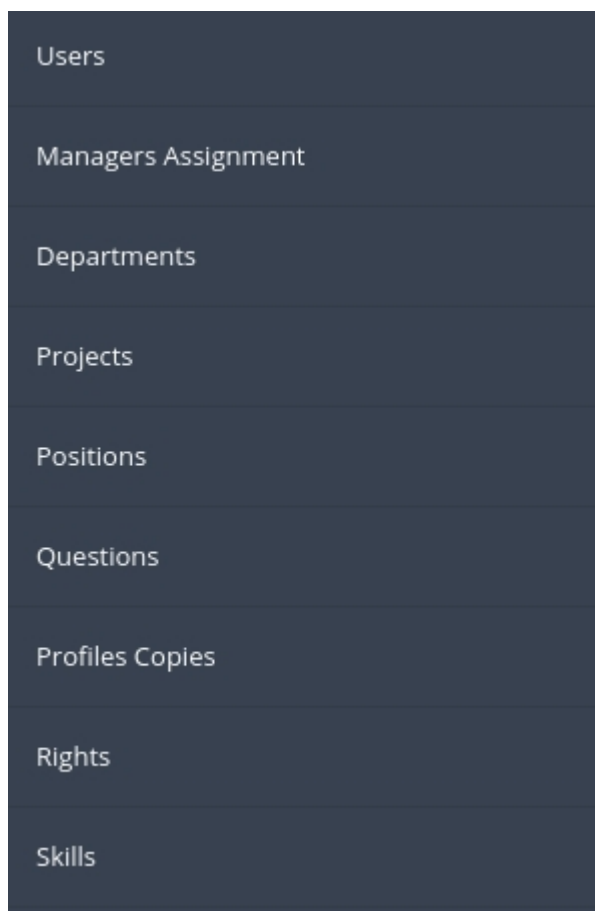


Рисунок 5.2 – Меню функций администратора

На рисунке 5.4 представлена страница назначения менеджеров. Страница выполнена в виде таблицы, где строками являются департаменты, а столбцами - тип менеджера. Назначение менеджеров происходит путем выбора из выпадающего списка пользователей для назначения в качестве соответствующего менеджера в соответствующем департаменте.

При открытии менеджером страницы профиля появляются функции, показанные на рисунке 5.3

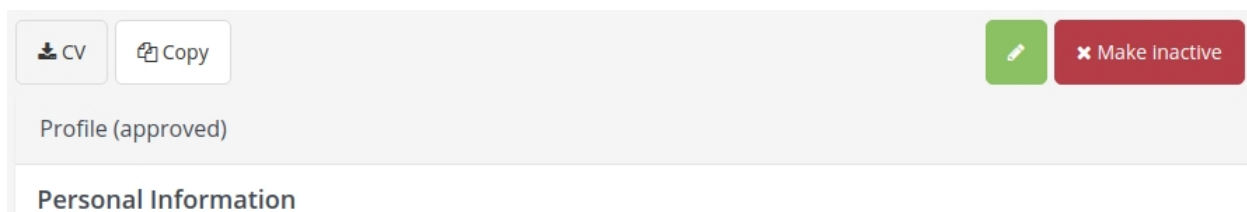


Рисунок 5.3 – Функции с профилем, доступные администратору

Назначение функций профиля следующее:

- кнопка «CV» - кнопка генерации резюме, генерирует из профиля резюме в формате pdf;
- кнопка «Сору» - кнопка создания копии профиля, позволяет делать копии профиля для дальнейшего редактирования и создания резюме без изменения основного профиля. Все копии профилей находятся в отдельном меню, доступ к ним имеют только менеджеры, их редактирование никак не влияет на основной профиль;
- кнопка «Редактировать» - кнопка, открывающая профиль для редактирования;
- кнопка «Make Inactive» - кнопка, деактивирующая профиль.

Сгенерированное из профиля резюме представлено на рисунке 5.5

Managers Assignment			
DEPARTMENT / ROLE	SALES MANAGER	TECH MANAGER	HR MANAGER
SOFTWARE DEVELOPMENT	HR manager ▾	Tech manager ▾	HR manager ▾
DESIGN	Katerina Merzlova Vladimir Shidlovsky Shidlovsky	ADmin ADmin ▾	ADmin ADmin ▾
QA DEPT	ADmin ADmin ▾	ADmin ADmin ▾	ADmin ADmin ▾
PROJECT MANAGEMENT	Whitley Leuschke ▾	ADmin ADmin ▾	HR manager ▾
BUSINESS ANALYSIS	ADmin ADmin ▾	HR manager ▾	Rickie Goyette III ▾
DATA SCIENCE	Sales manager ▾	Dr. Daria Sporer ▾	Dr. Daria Sporer ▾
PRODUCT MANAGEMENT	Dr. Daria Sporer ▾	HR manager ▾	Sales manager ▾
BDSM	ADmin ADmin ▾	Tech manager ▾	HR manager ▾
HR DEPARTMENT	HR manager ▾	Ned Boehm II ▾	HR manager ▾
TOP MANAGEMENT	Tech manager ▾	ADmin ADmin ▾	Rickie Goyette III ▾

Рисунок 5.4 – Страница назначения менеджеров



Katerina Merzlova, Head of Sales & Marketing

Katerina Merzlova, Head of Sales & Marketing



2006-2011 Minsk State Linguistics University, English
2015-2016 European Marketing Confederation, Marketing Director

Head of Sales & Marketing

Raina Sanford

Period	Description
February 2019 - current	Ruby on Rails, Ruby on Rails, Ruby on Rails, Ruby on Rails, Ruby on Rails, Ruby on Rails, Ruby on Rails, Ruby on Rails, Ruby on Rails, Ruby on Rails, Releinds ut commodi quis.
Role	Bootstrap, CSS/SCSS, HTML/JHTML, JavaScript, jQuery, React Native
Jermaine Rath	

February 2019 - current	
Role	Ruby on Rails, Ruby on Rails, Ruby on Rails,
Description	Cum qui explicabo quibusdam.
Technologies	AngularJS, Bootstrap, CSS/SCSS

Skills matrix

Professional Skills	Experience, in years	Level	Last used, year
TECHNOLOGIES			
AngularJS	1	novice	2019
Bootstrap	1	novice	2019
CSS/SCSS	2	intermediate	2019
HTML/JHTML	1	novice	2019
JavaScript	2	intermediate	2019
jQuery	2	intermediate	2019
React Native	1	novice	2019

На рисунке 5.6 представлена форма создания нового проекта. На форме представлены поля для ввода названия проекта, его описания, выбора статуса проекта, выпадающий список для выбора технологий. Есть возможность выбора нескольких технологий. Есть отдельная кнопка для добавления пользователя, которая добавляет дополнительные поля для задания пользователя, его роли и зоны ответственности. Можно добавить несколько пользователей на проект.

New project

Create

Title:

Description:

Status:

Select...

Used Technologies

What technologies did you use? Please, name all the tools & technologies used:

Select...

Users

x

User name:

User Role:

User responsibilities:

Рисунок 5.6 – Форма создания проекта

На рисунке 5.7 представлена страница управления правами. Данная страница доступна только администраторам системы и позволяет управлять уровнями доступа менеджеров. Так, можно включать или отключать доступ к редактированию определенных секций в профилях пользователей, доступ к генерации резюме, созданию копий, доступ к менеджменту департаментов, позиций, проектов.

Rights			
RIGHT / ROLE	SALES MANAGER	TECH MANAGER	HR MANAGER
PROFILE MODERATION: PERSONAL INFORMATION	+ Grant	+ Grant	— Take
PROFILE MODERATION: EDUCATION	+ Grant	+ Grant	— Take
PROFILE MODERATION: PROFESSIONAL PROFILE	+ Grant	— Take	— Take
PROFILE MODERATION: ADDITIONAL SKILLS	+ Grant	— Take	— Take
PROFILE MODERATION: COMPANIES USER WORKED IN	+ Grant	+ Grant	— Take
PROFILE MODERATION: PROJECTS USER WORKED ON PREVIOUSLY	— Take	— Take	— Take
PROFILE MODERATION: PROJECTS USER WORKED ON IN SUMATOSOFT	— Take	— Take	— Take
PROFILE MODERATION: SKILL MATRIX	+ Grant	— Take	— Take
PROFILES MANAGEMENT: PROFILES LIST	— Take	— Take	— Take

Рисунок 5.7 – Страница управления правами

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ВНЕДРЕНИЯ ПРОГРАММНОГО СРЕДСТВА МЕНЕДЖМЕНТА ПЕРСОНАЛА ПРЕДПРИЯТИЯ

6.1 Общая характеристика разрабатываемого средства менеджмента персонала предприятия

Разработанное средство менеджмента предназначено для автоматизации и упрощения процесса менеджмента персонала на предприятии ООО «СуматоСофт». Система позволяет управлять текущими должностями, проектами организации, отслеживать движение сотрудников по проектам, департаментам, позициям, их профессиональную подготовку, автоматически генерировать резюме для передачи клиентам.

Разработанное средство менеджмента обеспечивает контроль менеджерам или администраторам над информацией о проектах, сотрудниках, департаментах и т. д.

Разработка программного средства осуществлялась с помощью фреймворка Ruby on Rails.

Данное программное средство в первую очередь ориентировано на менеджеров компании, поэтому разработан максимально простой и дружелюбный пользовательский интерфейс.

Применение программного средства позволит снизить затраты на обработку аналитических данных и ускорить процесс подбора работников для новых проектов.

Средство менеджмента персонала разрабатывалось сотрудниками предприятия.

6.2 Расчет затрат на разработку программного обеспечения

В соответствии с «Рекомендациями по применению «Единой тарифной сетки» рабочих и служащих народного хозяйства» и тарифными разрядами и коэффициентами должностей каждому исполнителю устанавливается разряд и тарифный коэффициент.

Месячная тарифная ставка каждого исполнителя определяется путем умножения действующей месячной тарифной ставки 1-го разряда на тарифный коэффициент, соответствующий установленному тарифному разряду и рассчитывается по формуле:

$$З_{зм} = З_{зм}^1 \cdot К_{т}, \quad (6.1)$$

где $З_{зм}$ — тарифная ставка за месяц, руб.;
 $З_{зм}^1$ — тарифная ставка 1-го разряда за месяц, руб.;
 $К_{т}$ — тарифный коэффициент, ед.

$$З_{зм} = 290,97 \cdot 2,74 = 797,26 \text{ руб.}$$

Основная заработная плата исполнителей на конкретное ПС рассчитывается по формуле:

$$З_{оз} = З_{зд} \cdot Т_{о} \cdot К_{п} \cdot К_{пр}, \quad (6.2)$$

где $З_{оз}$ — основная заработная плата, руб.;
 $З_{зд}$ — тарифная ставка за день ($З_{зм}$ разделить на 21,25), руб.;
 $Т_{о}$ — общая трудоемкость ПС, человеко-дней;
 $К_{п}$ — коэффициент естественных потерь рабочего времени, ед.;
 $К_{пр}$ — коэффициент премирования, $К_{пр} = 1,2$;

$$З_{зд} = \frac{797,26}{21,25} = 37,52 \text{ руб.,}$$

$$З_{оз} = 37,52 \cdot 39,62 \cdot 1,3 \cdot 1,2 = 2319 \text{ руб.}$$

Дополнительная заработная плата на конкретное ПС включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по нормативу в процентах к основной заработной плате:

$$З_{дз} = \frac{З_{оз} \cdot Н_{дз}}{100}, \quad (6.3)$$

где $З_{дз}$ — дополнительная заработная плата на конкретное ПС, руб.;
 $Н_{дз}$ — норматив дополнительной заработной платы, (22%);

$$З_{дз} = \frac{2319 \cdot 22}{100} = 510,18 \text{ руб.}$$

Отчисления на социальные нужды включают в предусмотренные законодательством отчисления в фонд социальной защиты (34%) и фонд обязательного страхования (0,6%) в процентах от основной и дополнительной заработной платы и рассчитываются по формуле:

$$З_{\text{соц}} = \frac{(З_{\text{оз}} + З_{\text{дз}}) \cdot Н_{\text{соц}}}{100}, \quad (6.4)$$

$$З_{\text{соц}} = \frac{(2319 + 510,18) \cdot 34,6}{100} = 978,90 \text{ руб.}$$

Расходы по статье «Машинное время» ($P_{\text{мв}}$) включают оплату машинного времени, необходимого для разработки и отладки ПС, и определяются по формуле:

$$P_{\text{мв}} = Ц_{\text{м}} \cdot T_{\text{ч}} \cdot C_{\text{р}}, \quad (6.5)$$

где $Ц_{\text{м}}$ — цена одного машино-часа;
 $T_{\text{ч}}$ — количество часов работы в день;
 $C_{\text{р}}$ — длительность проекта.

Стоимость машино-часа на предприятии составляет 1,5 руб. Разработка проекта займет 40 дней. Определим затраты по статье «Машинное время»:

$$P_{\text{мв}} = 1,5 \cdot 8 \cdot 40 = 480 \text{ руб.}$$

Расчет прочих затрат осуществляется в процентах от затрат на основную заработную плату разработчиков с учетом премии по формуле:

$$З_{\text{пз}} = \frac{З_{\text{оз}} \cdot Н_{\text{пз}}}{100}, \quad (6.6)$$

где $Н_{\text{пз}}$ — норматив прочих затрат, берется в пределах, равный 100% от основной заработной платы.

$$З_{\text{пз}} = \frac{2319 \cdot 100}{100} = 2319 \text{ руб.}$$

Затраты по статье «Накладные расходы» определяются по формуле:

$$P_{\text{н}} = \frac{З_{\text{оз}} \cdot Н_{\text{рн}}}{100}, \quad (6.7)$$

где $Н_{\text{рн}}$ — процент накладных расходов, (55%).

$$P_{\text{н}} = \frac{2319 \cdot 55}{100} = 1275,45 \text{ руб.}$$

Общая сумма расходов по смете на ПО составит:

$$C_p = Z_{\text{оз}} + Z_{\text{дз}} + Z_{\text{соц}} + P_{\text{мв}} + Z_{\text{пз}} + P_{\text{н}}, \quad (6.8)$$

$$C_p = 2319 + 510,18 + 978,90 + 480 + 2319 + 1275,45 = 6882,53 \text{ руб.}$$

Полная сумма затрат на разработку программного обеспечения приведена в таблице 6.1.

Таблица 6.1 – Исходные данные

Статья затрат	Сумма, руб
Основная заработная плата команды разработчиков	2319
Дополнительная заработная плата команды разработчиков	510,18
Затраты на машинное время	480
Отчисления на социальные нужды	978,90
Прочие затраты	2319
Накладные расходы	1275,45
Общая сумма затрат на разработку	6882,53

Расходы на освоение разработчиком ПО рассчитываются по формуле:

$$P_o = \frac{C_p \cdot H_o}{100}, \quad (6.9)$$

где H_o — норматив расходов на освоение, 15%.

$$Z_{\text{пз}} = \frac{6882,53 \cdot 15}{100} = 1032,38 \text{ руб.}$$

Полная себестоимость разработанного ПО определяется по следующей формуле:

$$C_{\text{п}} = C_p + P_o, \quad (6.10)$$

$$C_{\text{п}} = 6882,53 + 1032,38 = 7914,91 \text{ руб.}$$

Таким образом, общие затраты предприятия на разработку системы управления бизнес-процессами составят 7914,91 руб.

6.3 Расчет стоимостной оценки результата

В качестве экономического эффекта выступает общая экономия всех видов ресурсов при использовании разработанной системы относительно того, как процесс менеджмента персонала осуществлялся ранее.

Данные для расчета экономии ресурсов в связи с применением разработанной системы менеджмента персонала отображены в таблице 6.2.

Таблица 6.2 – Исходные данные

Наименование показателя	Условное обозначение	Значение в базовом варианте	Значение в новом варианте
Капитальные вложения, руб.	$K_{пр}$	—	7914,91
Среднемесячная заработная плата одного специалиста, руб.	$З_{см}$	1500	1500
Среднемесячное число рабочих дней	$Д_p$	18	18
Количество задач, решаемых за год	$З_{т1}, З_{т2}$	50	30
Объём работ, выполняемый при решении одной задачи	A_1, A_2	16	10
Средняя трудоемкость работ в расчете на 1 задачу, чел.-час на задачу	T_{c1}, T_{c2}	1	2
Количество часов работы в день	$T_ч$	8	8

Так как ПО разрабатывалось для внутреннего использования ООО «СуматоСофт», то общие капитальные затраты на разработку соответствуют полной себестоимости данного ПО и составят 7914,91 руб.

Определим общую годовую экономию затрат при использовании системы менеджмента персонала.

Экономия затрат на заработную плату при использовании нового ПО в расчете на количество задач, выполняемых в год рассчитывается по формуле:

$$C_3 = \frac{З_{см} \cdot (T_{c1} \cdot З_{т1} \cdot A_1 - T_{c2} \cdot З_{т2} \cdot A_2)}{T_ч \cdot Д_p}, \quad (6.11)$$

где $Z_{зм}$ — среднемесячная заработная плата одного специалиста, руб.;

$Z_{т1}, Z_{т2}$ — количество задач, решаемых в год в базовом варианте и при использовании нового ПО соответственно, задач;

$T_{ч}$ — количество часов работы в день, часов;

$T_{с1}, T_{с2}$ — средняя трудоемкость работ в расчете на 1 задачу в базовом варианте и при использовании нового ПО соответственно, человеко-часов;

A_1, A_2 — объем выполняемых работ в расчете на 1 задачу в базовом варианте и при использовании нового ПО соответственно, обращений;

D_p — среднемесячное количество рабочих дней.

$$C_3 = \frac{1500 \cdot (1 \cdot 50 \cdot 16 - 2 \cdot 30 \cdot 10)}{8 \cdot 18} = 2083,33 \text{ руб.}$$

Экономия с учетом начисления на заработную плату определяется по формуле:

$$C_n = C_3 \cdot K_{нз}, \quad (6.12)$$

где $K_{нз}$ — коэффициент начислений на заработную плату, $K_{нз} = 1,55$.

$$C_n = 2083,33 \cdot 1,55 = 3229,16 \text{ руб.}$$

Общая годовая экономия в результате сокращения текущих затрат, связанных с использованием нового ПО:

$$C_o = C_n \cdot C_3, \quad (6.13)$$

$$C_o = 3229,16 + 2083,33 = 5312,49 \text{ руб.}$$

Для пользователя в качестве экономического эффекта выступает чистая прибыль – дополнительная прибыль, остающаяся в его распоряжении, которая определяется по формуле:

$$\Delta\Pi_{ч} = C_o - \frac{C_o \cdot H_{п}}{100}, \quad (6.14)$$

где $H_{п}$ — ставка налога на прибыль, $H_{п} = 18\%$.

Годовой прирост чистой прибыли от полученной экономии ресурсов при использовании разработанной системы управления продажами составит:

$$\Delta\Pi_q = 5312,49 - (5312,49 \cdot 18)/100 = 4356.24 \text{ руб.}$$

6.4 Расчет экономического эффекта

В процессе использования нового ПС чистая прибыль в конечном итоге возмещает капитальные затраты. Однако, полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени – расчетному году (за расчетный год принят год разработки ДП) путем умножения результатов и затрат за каждый год на коэффициент приведения ($ALFA_t$), который рассчитывается по формуле:

$$ALFA_t = (1 + E_n)^{t_p - t}, \quad (6.15)$$

где E_n — норматив приведения разновременных затрат и результатов;

t_p — расчетный год, $t_p = 1$;

t — номер года, результаты и затраты которого приводятся к расчетному.

$$ALFA_1 = (1 + 0,2)^{1-1} = 1;$$

$$ALFA_2 = (1 + 0,2)^{1-2} = 0,833;$$

$$ALFA_3 = (1 + 0,2)^{1-3} = 0,690;$$

$$ALFA_4 = (1 + 0,2)^{1-4} = 0,579.$$

Данные расчета экономического эффекта целесообразно свести в таблицу 6.3.

Таблица 6.3 – Расчет экономического эффекта от использования нового ПС

Показатели	Ед. измерения	Методика расчета	2019	2020	2021	2022
Прирост прибыли за счет экономии затрат	руб.	$\Delta\Pi_q$	4356,24	4356,24	4356,24	4356,24

Продолжение таблицы 6.3

Показатели	Ед. измерения	Методика расчета	2019	2020	2021	2022
Сумма прибыли с учетом фактора времени	руб.	$\Delta\Pi_q \cdot ALFA_t$	4356,24	3628,75	3005,81	2522,26
Сумма затрат	руб.	K_o	7914,91	—	—	—
Сумма затрат с учетом фактора времени	руб.	$K_o \cdot ALFA_t$	7914,91	—	—	—
Эконом. эффект	руб.	$\Delta\Pi_q \cdot ALFA_t - K_o \cdot ALFA_t$	-3558,67	3628,75	3005,81	2522,26
Эконом. эффект с нарастающим итогом	руб.		-3558,67	70,08	3075,89	5598,15
Коэффициент приведения	ед.	$ALFA_t$	1	0,833	0,690	0,579

Таким образом, все затраты на разработку системы менеджмента персонала полностью окупятся на второй год.

Рентабельность инвестиций в разработку и внедрение программного средства по следующей формуле:

$$P_{\text{и}} = \frac{\Pi_{\text{ср}}}{K_{\text{пр}}} \cdot 100\%, \quad (6.16)$$

где $\Pi_{\text{ср}}$ — среднегодовая величина чистой прибыли за расчетный период, руб.

$$P_n = \frac{\frac{\sum_{i=1}^4 \Delta\Pi_{\text{чи}}}{4}}{K_o} \cdot 100\% = \frac{3378,27}{7914,91} \cdot 100\% = 42\%.$$

В результате технико-экономического обоснования применения программного продукта были получены следующие показатели:

- чистый дисконтированный доход за четыре года составит 5598,15 руб.;
- затраты на разработку программного продукта окупятся на второй год использования;
- рентабельность инвестиций составляет 42%.

Таким образом, применение программного продукта является эффективным и инвестиции в его разработку целесообразно осуществлять.

ЗАКЛЮЧЕНИЕ

Результатом работы над дипломным проектом стало программное средство менеджмента персонала предприятия, задачей которого является отслеживание текущих проектов предприятия, текущей занятости сотрудников на проектах, их способностей, а также на основе полученных данных автоматически генерировать резюме сотрудника для дальнейшего использования в отделе маркетинга и продаж. Были проанализированы существующие сервисы генерации резюме, системы управления сотрудниками, исследованы разные направления и подходы к решению подобного рода задач.

На основании проведенного анализа предметной области были выдвинуты требования к программному средству. В качестве технологий разработки были выбраны наиболее современные существующие на данный момент средства, широко применяемые в индустрии.

Разработано программное средство, целевой платформой которого является веб-приложение и которое поддерживает следующие функции:

- создание и управление проектами предприятия;
- создание и управление текущих позиций и департаментов предприятия;
- заполнение и проверка профиля пользователя;
- создание копий профиля пользователя с возможностью их редактирования без изменения базового профиля;
- генерация резюме пользователя на основе заполненного профиля;
- возможность управления правами доступа для разграничения функций между менеджерами;
- возможность приглашения незарегистрированных пользователей в систему.

Согласно объявленным требованиям был разработан набор тест-кейсов, которые были успешно пройдены в ходе тестовых испытаний программного средства. Успешность прохождения тестов показывает корректность работы программы и соответствие функциональным требованиям.

Подробно описана методика использования программного средства, которая позволяет за достаточно короткие сроки освоить работу с программой.

Также в ходе работы над дипломным проектом рассмотрена экономическая сторона проектирования и разработки программного средства, рассчитан экономический эффект от использования программного средства у пользователя. В результате расчётов подтвердилась целесообразность разработки. Так как разработанная система предоставляет новый способ гене-

рации резюме сотрудников и использует современные технологии создания программных средств, рентабельность её разработки составила 42%, а инвестиции, вложенные в разработку, окупаются на второй год использования программного средства.

Основной целью разработки программы было упрощение процессов генерации резюме сотрудников, отслеживания текущих проектов и занятости сотрудников на них. В ходе работы над дипломным проектом эта цель была успешно достигнута.

В будущем планируется добавить возможность обмена сообщениями между сотрудниками, систему учета рабочего времени, более детальное управление проектами, его текущие задачи, их описание и статус. Для генерации резюме добавить возможность выбора информации для генерации, а также реализовать конструктор шаблонов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] ISTQB glossary – Specification [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://glossary.astqb.org/search/specification>. — Дата доступа: 18.04.19.

[2] Использование информационных технологий в системе управления персоналом [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://cyberleninka.ru/article/n/ispolzovanie-informatsionnyh-tehnologiy-v-sisteme-upravleniya-personalom>. — Дата доступа: 18.04.19.

[3] Макконнелл, С. Совершенный код. Мастер-класс / Пер. с англ. / С. Макконнелл. — М. : Издательско-торговый дом «Русская редакция», 2010. — 896 с.

[4] Как правильно переходить границу: кроссплатформенность в мобильном приложении [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://habrahabr.ru/company/parallels/blog/254325/>. — Дата доступа: 18.04.19.

[5] Sheriff, Paul D. Designing for Web or Desktop? [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://msdn.microsoft.com/en-us/library/ms973831.aspx>. — Дата доступа: 18.04.19.

[6] Shaaban, Sam. Web-Based vs “Desktop” Software [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://nurelm.com/web-based-vs-desktop-software/>. — Дата доступа: 18.04.19.

[7] Bychkov, Dmitriy. Desktop vs. Web Applications: A Deeper Look and Comparison [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://www.seguetech.com/desktop-vs-web-applications/>. — Дата доступа: 18.04.19.

[8] Summerfield, Jason. Mobile Website vs. Mobile App: Which is Best for Your Organization? [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://www.hswsolutions.com/services/mobile-web-development/mobile-website-vs-apps/>. — Дата доступа: 18.04.19.

[9] Garlan, David. An Introduction to Software Architecture / David Garlan, Mary Shaw / Ed. by V Ambriola, G Tortora. — New Jersey : World Scientific Publishing Company, 1994. — Vol. I. — Режим доступа: https://www.cs.cmu.edu/afs/cs/project/vit/ftp/pdf/intro_softarch.pdf. — Дата доступа: 18.04.19.

[10] Волосевич, А. А. Архитектура программного обеспечения: Курс лекций для студентов специальности 1-40 01 03 Информатика и технологии программирования / А. А. Волосевич. — Минск : БГУИР, 2013.

- [11] Team, Microsoft Patterns & Practices. Microsoft® Application Architecture Guide (Patterns & Practices) / Microsoft Patterns & Practices Team. — Microsoft Press, 2009.
- [12] Marinescu, Floyd. Domain-Driven Design Quickly / Floyd Marinescu, Abel Avram. — LULU PR, 2007.
- [13] Куликов, С. С. Базы данных. Рабочая тетрадь. / С. С. Куликов. — Минск .
- [14] Резюме. О чем оно может рассказать [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://www.rabotka.ru/resume/st6.php>. — Дата доступа: 18.04.19.
- [15] Software Cost Estimation with Cocomo II / Barry W. Boehm [et al.]. — Prentice Hall, 2000.
- [16] Walston, Claude E. A Method of Programming Measurement and Estimation. / Claude E. Walston, Charles P. Felix // IBM Systems Journal. — 1977. — Vol. 16, no. 1. — Pp. 54–73.
- [17] Основы Javascript [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://developer.mozilla.org>. — Дата доступа: 18.04.19.
- [18] О Ruby [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://www.ruby-lang.org/ru/about/>. — Дата доступа: 18.04.19.
- [19] Rails [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://rusrails.ru/getting-started-with-rails>. — Дата доступа: 18.04.19.
- [20] PostgreSQL [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.sai.msu.su/megera/postgres/talks/what_is_postgresql.html — Дата доступа: 18.04.19.
- [21] МакЛафлин, Бретт. От Web-сайтов к Web-приложениям: Часть 1. Web-сайт или Web-приложение? [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://www.ibm.com/developerworks/ru/library/wa-websiteapp/>. — Дата доступа: 18.04.19.
- [22] Краткое руководство по React JS [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://habrahabr.ru/post/248799/>. — Дата доступа: 18.04.19.
- [23] React. A javascript library for building user interfaces [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://facebook.github.io/react/>. — Дата доступа: 18.04.19.
- [24] Когда, зачем и почему может быть полезным Redux [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://wayup.in/blog/when-and-why-redux-can-be-useful>. — Дата доступа: 18.04.19.

[25] Abramov, Dan. Presentational and Container Components [Электронный ресурс]. — Электронные данные. — Режим доступа: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0. — Дата доступа: 18.04.19.

[26] Жуков, Антон. Стилизация React-компонентов [Электронный ресурс]. — Электронные данные. — Режим доступа: habrahabr.ru/company/devexpress/blog/283314/. — Дата доступа: 18.04.19.

[27] Основы Sass [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://sass-scss.ru/guide/>. — Дата доступа: 18.04.19.

[28] Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. — EPAM Systems, RD Dep., 2017. — http://svyatoslav.biz/software_testing_book/.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код

```
# categories_controller.rb

module Api
  module V1
    module Admin
      class CategoriesController < BaseController
        before_action :find_category, only: %i[update show destroy]

        def index
          @categories = Category.includes(:skills)
          render json: @categories
        end

        def create
          authorize_api Category do
            @category = Category.create(category_params)

            create_or_update_response
          end
        end

        def update
          authorize_api(@category, error_msg: "Can't delete skills. Some of them in use.",
            options: { skills_attributes: params[:category][:
              skills_attributes] }) do
            @category.update(category_params)

            create_or_update_response
          end
        end

        def show
          authorize_api @category do
            render json: @category
          end
        end

        def destroy
          authorize_api @category, error_msg: "Can't delete category. Some skills may be
            used." do
            @category.destroy

            render json: @category
          end
        end

        private

        def find_category
          @category = Category.includes(skills: [:technologies]).find(params[:id])
        end
      end
    end
  end
end
```

```

end

def create_or_update_response
  if @category.valid?
    render json: @category
  else
    render_model_errors(:unprocessable_entity, @category)
  end
end

def category_params
  params.require(:category).permit(:id, :name,
    skills_attributes: %i[id name _destroy])
end
end
end
end

# department_users_controller.rb

module Api
  module V1
    module Admin
      class DepartmentUsersController < BaseController
        def create_managers
          authorize_api DepartmentUser do
            DepartmentUser.where.not(role: Role.employee).destroy_all
            @managers = DepartmentUser.create!(department_managers_params)
            render json: @managers
          end
        end

        def managers
          authorize_api DepartmentUser do
            @managers = DepartmentUser.includes(user: :position).where.not(role: Role.
              employee)
            render json: @managers
          end
        end

        private

        def department_managers_params
          params.permit(departmentManagers: %i[department_id role_id user_id]).require(:
            departmentManagers)
        end
      end
    end
  end
end

# departments_controller.rb

module Api
  module V1

```

```

module Admin
  class DepartmentsController < BaseController
    before_action :find_department, only: %i[update show destroy]

    def create
      authorize_api Department do
        @department = Department.create(department_params)
        create_or_update_response
      end
    end

    def show
      authorize_api @department do
        render json: @department, serializer: DepartmentWithAssociationsSerializer
      end
    end

    def update
      authorize_api @department do
        @department.update(department_params)
        create_or_update_response
      end
    end

    def destroy
      authorize_api @department, error_msg: "Can't delete department, please delete it
        's positions before." do
        @department.destroy
        render json: @department
      end
    end

    private

    def create_or_update_response
      if @department.valid?
        render json: @department
      else
        render_model_errors(:unprocessable_entity, @department)
      end
    end

    def find_department
      @department = Department.includes(department_users: { user: :position }).find(
        params[:id])
    end

    def department_params
      params.require(:department).permit(:name, :office)
    end
  end
end
end
end

```

```

# positions_controller.rb

module Api
  module V1
    module Admin
      class PositionsController < BaseController
        before_action :find_position, only: %i[update show destroy]

        def create
          authorize_api @position do
            @position = Position.create(position_params)
            create_or_update_response
          end
        end

        def update
          authorize_api @position do
            @position.update(position_params)
            create_or_update_response
          end
        end

        def show
          authorize_api @position do
            render json: @position
          end
        end

        def destroy
          authorize_api @position, error_msg: "Can't delete position, it has related users" do
            @position.destroy
            render json: @position
          end
        end

        private

        def create_or_update_response
          if @position.valid?
            render json: @position
          else
            render_model_errors(:unprocessable_entity, @position)
          end
        end

        def find_position
          @position = Position.find(params[:id])
        end

        def position_params
          params.require(:position).permit(:name, :department_id, technologies_attributes:
            %i[id skill_id technologable_id _destroy])
        end
      end
    end
  end
end

```

```

    end
  end
end

# profile_copies_controller.rb

module Api
  module V1
    module Admin
      class ProfileCopiesController < BaseController
        before_action :find_profile, only: %i[show destroy]

        def create
          authorize_api Profile do
            @parent = Profile.find(params[:id])
            @profile = @parent.dup
            @profile.update(parent_id: @parent.id)
            @profile.copy!

            if @profile.valid?
              render json: @profile
            else
              render_model_errors(:unprocessable_entity, @profile)
            end
          end
        end

        def index
          authorize_api Profile do
            @profiles = Profile.copies.includes(user: :position)
            render json: @profiles, include: ['user.position']
          end
        end

        def show
          authorize_api @profile do
            render json: @profile
          end
        end

        def destroy
          authorize_api @profile do
            @profile.destroy
            render json: @profile
          end
        end

        private

        def find_profile
          @profile = Profile.find(params[:id])
        end
      end
    end
  end
end

```

```

    end
  end

# projects_controller.rb

module Api
  module V1
    module Admin
      class ProjectsController < BaseController
        before_action :find_project, only: %i[update show destroy]

        def create
          authorize_api Project do
            @project = Project.create(project_params)
            if @project.valid?
              ProfilePacker.new.pack_project(@project, [], @project.users.to_a)
              render_project
            else
              render_model_errors(:unprocessable_entity, @project)
            end
          end
        end

        # rubocop:disable Metrics/AbcSize
        def update
          authorize_api @project do
            users_was = @project.users.to_a
            @project.update(project_params)
            errors = []
            errors = @project.errors.full_messages if @project.invalid?
            @project.reload
            ProfilePacker.new.pack_project(@project, users_was, @project.users.to_a) if
              errors.empty?

            if errors.present?
              render_error(:unprocessable_entity, errors)
            else
              render_project
            end
          end
        end

        # rubocop:enable Metrics/AbcSize

        def show
          authorize_api @project do
            render_project
          end
        end

        def destroy
          authorize_api @project do
            @project.destroy
            render_project
          end
        end
      end
    end
  end
end

```



```

private

def render_project
  render json: @project, serializer: ProjectWithAssociationsSerializer,
    include: 'technologies.skill.category,user_projects.technologies'
end

def find_project
  @project = Project.includes(technologies: { skill: :category }).find(params[:id
  ])
end

def project_params
  params.require(:project).permit(
    :title,
    :description,
    :status,
    user_projects_attributes: %i[id user_id responsibilities role _destroy],
    technologies_names: []
  )
end
end
end
end
end

# questions_controller.rb

module Api
  module V1
    module Admin
      class QuestionsController < BaseController
        before_action :find_question, only: %i[update show destroy]

        def create
          authorize_api Question do
            @question = Question.create(question_params)
            create_or_update_response
          end
        end

        def update
          authorize_api @question do
            @question.update(question_params)
            create_or_update_response
          end
        end

        def show
          render_response
        end

        def destroy
          authorize_api @question do

```

```

        @question.destroy
        render_response
      end
    end

  private

  def question_params
    params.require(:question).permit(
      :description,
      :answer_sample,
      :summary,
      position_ids: []
    )
  end

  def find_question
    @question = Question.find(params[:id])
  end

  def create_or_update_response
    if @question.valid?
      render_response
    else
      render_model_errors(:unprocessable_entity, @question)
    end
  end

  def render_response
    render json: @question, serializer: QuestionWithAssociationsSerializer, include:
      ['positions.department.name']
  end
end
end
end
end
end

#rights_controller.rb

module Api
  module V1
    module Admin
      class RightsController < BaseController
        def index
          authorize_api RightsList do
            @rights = RightsList.all
            render json: @rights
          end
        end

        def update
          authorize_api @right do
            @right = RightsList.find(params[:id])
            @right.update(rights_params)
          end
        end
      end
    end
  end
end

```

```

        @rights = RightsList.all
        render json: @rights
      end
    end

    def rights_all
      authorize_api RightsList do
        render json: RightsList::RIGHTS.map { |right| [right, I18n.t(right)] }
      end
    end

    private

    def rights_params
      params.require(:roleRights).permit(:role_id, rights: [])
    end
  end
end
end
end

# cv_pdf_generator.rb

require 'render_anywhere'
require 'tempfile'

class CvPdfGenerator
  include RenderAnywhere

  def initialize(cv_info)
    @cv = cv_info
  end

  def to_pdf
    header = header_file
    kit = PDFKit.new(
      cv_body,
      header_html: header.path,
      margin_bottom: '0mm',
      margin_left: '0mm',
      margin_right: '10mm',
      margin_top: '45mm'
    )
    cv = Tempfile.new(['cv', '.pdf'])
    kit.to_file(cv.path)
    cv
  end

  private

  def cv_body
    render template: 'cv/show', layout: 'cv', locals: { cv: @cv }
  end

  def cv_header

```

```

    render template: 'cv/_header', layout: 'cv', locals: { cv: @cv }
end

def header_file
  header = Tempfile.new(['header', '.html'])
  header.write(cv_header)
  header.close
  header
end
end

# profile_moderation_service.rb

class ProfileModerationService
  def initialize(params)
    @user = params[:user]
    @profile = params[:profile]
    @base_url = params[:base_url]
  end

  def assign_department
    dep_name = @profile.info['professional_profile_department']['label']
    @department = Department.find_by(name: dep_name)
  end

  def moderate
    @status = @profile[:status]
    assign_department
    # return department manager due to status profile status
    manager = @department.public_send(@status)
    ModerationMailer.notify_manager(manager, @user, @base_url).deliver_later
  end
end

# profile_packer.rb

class ProfilePacker
  def pack_project(project, users_was, users)
    if users.count > users_was.count
      users_add_to = users - users_was
      users_add_to.each do |user|
        add_project_to_profile(user, project)
      end
    elsif users.count < users_was.count
      users_remove_from = users_was - users
      users_remove_from.each do |user|
        remove_project_from_profile(user, project)
      end
    end
  end

  private

  def add_project_to_profile(user, project)
    profile = user.base_profile
    user_project = UserProject.find_by(user: user, project: project)
  end
end

```

```

profile.info['sumato_projects'] ||= []
profile.info['sumato_projects'].delete_if { |proj| proj['name'] == project.title }

profile_proj = {
  name: project.title,
  role: user_project.role,
  company: 'SumatoSoft',
  project: {
    label: project.title,
    value: project.id
  },
  # TODO: add to project create/edit page datepickers for users
  end_date: '',
  start_date: Time.now.to_s,
  description: project.description,
  responsibilities: user_project.responsibilities
}

profile_proj[:technologies] = project.technologies.includes(:skill).map do |
  technology|
    { value: technology.skill.name, label: technology.skill.name }
  end

profile.info['sumato_projects'] << profile_proj
profile.save!
end

def remove_project_from_profile(user, project)
  profile = user.base_profile
  profile.info['sumato_projects'].delete_if { |proj| proj['name'] == project.title }
  profile.save!
end

# profile_unpacker.rb

class ProfileUnpacker

  def initialize(params)
    @profile = params[:profile]
    @info = @profile.info
    @user = @profile.user
    @department = Department.find(@info['professional_profile_department']["value"])
    @position = Position.find(@info['professional_profile_position']["value"])
  end

  def unpack
    update_user
    update_departments
    update_projects
  end

  def update_user
    name = "#{@info['personal_information_name']} #{@info['personal_information_surname']}

```

```

    ']]"'
    @user.update_attributes(position: @position, name: name)
  end

  def update_departments
    DepartmentUser.where(user: @user, role: Role.employee.first).delete_all
    DepartmentUser.create!(user: @user, department: @department, role: Role.employee.
      first)
  end

  def update_projects
    projects_info = @info['sumato_projects']
    UserProject.where(user: @user).delete_all

    projects_info.each do |project_info|
      project = Project.find_by(title: project_info['name'])
      UserProject.create!(
        user: @user,
        project: project,
        role: project_info['role'],
        responsibilities: project_info['responsibilities']
      )
    end
  end

end

# routes.rb

Rails.application.routes.draw do
  devise_for :users

  mount LetterOpenerWeb::Engine, at: '/letter_opener' if Rails.env.development?

  authenticated :user, lambda { |user| user.admin? } do
    require 'sidekiq/web'
    mount Sidekiq::Web => '/sidekiq'
  end

  root to: 'home#index'
  get 'front/*path', to: 'home#index'
  namespace :api do
    namespace :v1 do
      namespace :admin, defaults: { format: :json } do
        resources :users, only: %i[index]
        resources :profile_copies, except: %i[new]
        resources :statuses, only: %i[index]
        resources :skills, only: %i[index]
        resources :roles, only: %i[index]
        resources :rights, only: %i[index update]
        resources :departments, only: %i[create show update destroy]
        resources :questions, only: %i[create update show destroy]
        resources :projects
        resources :positions, except: %i[new]
        resources :categories, except: %i[new]
      end
    end
  end
end

```

```

    post 'departments_managers', to: 'department_users#create_managers'
    get 'managers', to: 'department_users#managers'
    get 'rights/rights_all', to: 'rights#rights_all'
  end
  resources :current_user, only: %i[index]
  resources :users, only: %i[show]
  resources :technologies, only: [:index]
  resources :departments, only: %i[index]
  resources :positions, only: %i[index] do
    resources :questions, only: %i[index], controller: 'positions/questions'
  end
  resources :projects, only: [:index]
  resources :profiles, only: %i[create update show]
  resources :questions, only: %i[index]
  resources :profiles_moderation, only: %i[update]
  resources :cv, only: %i[show]
  resources :profile_photo, only: %i[update]
end
end
end

# schema.rb

ActiveRecord::Schema.define(version: 20190225071319) do

  # These are extensions that must be enabled in order to support this database
  enable_extension "plpgsql"

  create_table "aasm_logs", force: :cascade do |t|
    t.bigint "user_id"
    t.string "aasm_loggable_type"
    t.bigint "aasm_loggable_id"
    t.string "from_state"
    t.string "to_state"
    t.jsonb "meta"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["aasm_loggable_type", "aasm_loggable_id"], name: "index_aasm_logs_on_aasm_loggable_type_and_aasm_loggable_id"
    t.index ["user_id"], name: "index_aasm_logs_on_user_id"
  end

  create_table "categories", force: :cascade do |t|
    t.string "name"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["name"], name: "index_categories_on_name", unique: true
  end

  create_table "department_users", force: :cascade do |t|
    t.bigint "user_id"
    t.bigint "department_id"
    t.bigint "role_id"
    t.index ["department_id"], name: "index_department_users_on_department_id"
    t.index ["role_id"], name: "index_department_users_on_role_id"
  end

```

```

t.index ["user_id", "department_id", "role_id"], name: "
    index_department_users_on_user_id_and_department_id_and_role_id", unique: true
t.index ["user_id"], name: "index_department_users_on_user_id"
end

create_table "departments", force: :cascade do |t|
  t.string "name", default: "", null: false
  t.integer "office", default: 0
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
  t.index ["name"], name: "index_departments_on_name", unique: true
end

create_table "position_questions", force: :cascade do |t|
  t.bigint "question_id"
  t.bigint "position_id"
  t.index ["position_id"], name: "index_position_questions_on_position_id"
  t.index ["question_id"], name: "index_position_questions_on_question_id"
end

create_table "positions", force: :cascade do |t|
  t.string "name"
  t.bigint "department_id"
  t.boolean "no_skills_matrix", default: false
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
  t.index ["department_id"], name: "index_positions_on_department_id"
  t.index ["name"], name: "index_positions_on_name"
end

create_table "profiles", force: :cascade do |t|
  t.bigint "user_id"
  t.jsonb "info"
  t.integer "status", default: 0
  t.bigint "parent_id"
  t.string "photo"
  t.index ["parent_id"], name: "index_profiles_on_parent_id"
  t.index ["user_id"], name: "index_profiles_on_user_id"
end

create_table "projects", force: :cascade do |t|
  t.string "title"
  t.text "description"
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
  t.integer "status", default: 0
end

create_table "questions", force: :cascade do |t|
  t.text "description"
  t.string "summary"
  t.string "answer_sample"
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
end

```



```

create_table "rights_lists", force: :cascade do |t|
  t.jsonb "rights"
  t.bigint "role_id"
  t.index ["role_id"], name: "index_rights_lists_on_role_id"
end

create_table "roles", force: :cascade do |t|
  t.integer "name", default: 0
  t.index ["name"], name: "index_roles_on_name", unique: true
end

create_table "skills", force: :cascade do |t|
  t.string "name"
  t.bigint "category_id"
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
  t.index ["category_id"], name: "index_skills_on_category_id"
  t.index ["name"], name: "index_skills_on_name", unique: true
end

create_table "technologies", force: :cascade do |t|
  t.integer "technologable_id"
  t.string "technologable_type"
  t.bigint "skill_id"
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
  t.index ["skill_id", "technologable_id", "technologable_type"], name: "
    index_technologies_on_skill_id_and_technologable", unique: true
  t.index ["skill_id"], name: "index_technologies_on_skill_id"
end

create_table "user_projects", force: :cascade do |t|
  t.bigint "user_id"
  t.bigint "project_id"
  t.text "role"
  t.text "responsibilities"
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
  t.index ["project_id"], name: "index_user_projects_on_project_id"
  t.index ["user_id", "project_id"], name: "
    index_user_projects_on_user_id_and_project_id"
  t.index ["user_id"], name: "index_user_projects_on_user_id"
end

create_table "users", force: :cascade do |t|
  t.string "email", default: "", null: false
  t.string "encrypted_password", default: "", null: false
  t.string "reset_password_token"
  t.datetime "reset_password_sent_at"
  t.datetime "remember_created_at"
  t.integer "sign_in_count", default: 0, null: false
  t.datetime "current_sign_in_at"
  t.datetime "last_sign_in_at"
  t.inet "current_sign_in_ip"

```

```

t.inet "last_sign_in_ip"
t.string "confirmation_token"
t.datetime "confirmed_at"
t.datetime "confirmation_sent_at"
t.string "unconfirmed_email"
t.datetime "created_at", null: false
t.datetime "updated_at", null: false
t.string "name", default: "", null: false
t.bigint "position_id"
t.boolean "admin", default: false
t.string "invitation_token"
t.datetime "invitation_created_at"
t.datetime "invitation_sent_at"
t.datetime "invitation_accepted_at"
t.integer "invitation_limit"
t.string "invited_by_type"
t.bigint "invited_by_id"
t.integer "invitations_count", default: 0
t.index ["confirmation_token"], name: "index_users_on_confirmation_token", unique:
  true
t.index ["email"], name: "index_users_on_email", unique: true
t.index ["invitation_token"], name: "index_users_on_invitation_token", unique: true
t.index ["invitations_count"], name: "index_users_on_invitations_count"
t.index ["invited_by_id"], name: "index_users_on_invited_by_id"
t.index ["invited_by_type", "invited_by_id"], name: "
  index_users_on_invited_by_type_and_invited_by_id"
t.index ["position_id"], name: "index_users_on_position_id"
t.index ["reset_password_token"], name: "index_users_on_reset_password_token", unique
  : true
end

add_foreign_key "department_users", "departments"
add_foreign_key "department_users", "users"
add_foreign_key "position_questions", "positions"
add_foreign_key "position_questions", "questions"
add_foreign_key "positions", "departments"
add_foreign_key "profiles", "profiles", column: "parent_id"
add_foreign_key "profiles", "users"
add_foreign_key "skills", "categories"
add_foreign_key "technologies", "skills"
add_foreign_key "user_projects", "projects"
add_foreign_key "user_projects", "users"
add_foreign_key "users", "positions"
end

//app.jsx

import React from 'react';
import { BrowserRouter, Route, Redirect, Switch } from 'react-router-dom';
import { connect } from 'react-redux';
import _ from 'lodash';
import { ToastContainer, ToastStore } from 'react-toasts';
import UserList from './components/admin/User/UserList';
import Sidebar from './components/layouts/Sidebar';
import Navbar from './components/layouts/Navbar';

```

```

import {
  getDepartmentsList,
  getCurrentUser,
  getPositionsList,
  getStatusesList,
  getProjectsList,
  getQuestionsList,
  getCategoriesList,
  getTechnologiesList,
} from './actions';
import UserPage from './components/pages/UserPage';
import UserEditPage from './components/pages/UserEditPage';
import CurrentUserEditPage from './components/pages/CurrentUserEditPage';
import CurrentUserPage from './components/pages/CurrentUserPage';
import ProfileCopiesList from './components/admin/ProfileCopy/ProfilesCopiesList';
import ProfileCopy from './components/admin/ProfileCopy/ProfileCopy';
import ProjectList from './components/admin/Project/ProjectList';
import Project from './components/admin/Project/Project';
import ProjectForm from './components/admin/Project/ProjectForm';
import { managerOrAdmin } from './auth';
import ManagersAssignment from './components/admin/ManagersAssignment';
import PositionsList from './components/admin/Position/PositionsList';
import DepartmentsList from './components/admin/Department/DepartmentsList';
import Department from './components/admin/Department/Department';
import Position from './components/admin/Position/Position';
import PositionForm from './components/admin/Position/PositionForm';
import DepartmentForm from './components/admin/Department/DepartmentForm';
import ProfileCopyEdit from './components/admin/ProfileCopy/ProfileCopyEdit';
import { PROFILE_STATUS } from './constants/constants';
import QuestionsList from './components/admin/Question/QuestionsList';
import Question from './components/admin/Question/Question';
import QuestionForm from './components/admin/Question/QuestionForm';
import Rights from './components/admin/Rights';
import Skills from './components/admin/Category/Skills';
import NotFound from './components/pages/NotFound';
import withAuthorization from './Authorize';
import { withFormEdit, withFormNew } from './components/Form';
import CategoryForm from './components/admin/Category/CategoryForm';

```

```

class App extends React.Component {

  componentWillMount() {
    const { dispatch } = this.props;
    dispatch(getCurrentUser());
    dispatch(getDepartmentsList());
    dispatch(getPositionsList());
    dispatch(getProjectsList());
    dispatch(getQuestionsList());
    dispatch(getTechnologiesList());
  }

  componentDidUpdate() {
    const { currentUser, dispatch } = this.props;
    if (managerOrAdmin(currentUser)) {

```

```

        dispatch(getStatusesList());
        dispatch(getCategoriesList());
    }

    if (!_isEmpty(currentUser) && currentUser.profile_status === PROFILE_STATUS.DRAFT) {
        ToastStore.success('Please, edit your profile.');
```

```
    }
}
```

```

render() {
    const { currentUser } = this.props;

    return !_isEmpty(currentUser) && (
        <BrowserRouter basename="/front">
            <div className="page-container">
                <ToastContainer store={ToastStore} />
                <Sidebar />
                <div className="page-content">
                    <Navbar />
                    <Switch>

                        <Route
                            exact
                            path="/"
                            render={() => {
                                let route = null;
                                if (currentUser.profile_status === PROFILE_STATUS.DRAFT) {
                                    route = <Redirect to="/current_user/edit" />;
                                } else if (managerOrAdmin(currentUser)) {
                                    route = <Redirect to="/admin/users" />;
                                } else {
                                    route = <Redirect to="/current_user" />;
                                }
                                return route;
                            }}
                        />

                        { managerOrAdmin(currentUser)
                            && (
                                <Route
                                    exact
                                    path="/admin/users"
                                    component={
                                        withAuthorization(currentUser, ['profiles_management_profiles_list'])(
                                            UserList)
                                    }
                                />
                            )
                        }

                        { managerOrAdmin(currentUser)
                            && (
                                <Route exact path="/users/:userId" component={UserPage} />
                            )
                        }

                        { managerOrAdmin(currentUser)

```

```

    && (
      <Route exact path="/users/:userId/edit" component={UserEditPage} />
    )
  }
  { managerOrAdmin(currentUser)
    && (
      <Route exact path="/admin/copies" component={ProfileCopiesList} />
    )
  }
  { managerOrAdmin(currentUser)
    && (
      <Route exact path="/admin/profile_copies/:profileId" component={ProfileCopy
        } />
    )
  }
  { managerOrAdmin(currentUser)
    && (
      <Route exact path="/admin/profile_copies/:profileId/edit" component={
        ProfileCopyEdit} />
    )
  }
  { managerOrAdmin(currentUser)
    && (
      <Route
        exact
        path="/admin/projects"
        component={
          withAuthorization(currentUser, ['project_management_list_of_projects'])
            (ProjectList)}
      />
    )
  }
  { managerOrAdmin(currentUser)
    && (
      <Route
        exact
        path="/admin/projects/:projectId"
        component={
          withAuthorization(currentUser, ['project_management_list_of_projects'])
            (Project)}
      />
    )
  }
  { managerOrAdmin(currentUser)
    && (
      <Route
        exact
        path="/admin/project/new/"
        component={
          withAuthorization(currentUser, ['
            project_management_add_edit_project_details'])(
            withFormNew(ProjectForm)
          )}
      />
    )
  }

```

```

}
{ managerOrAdmin(currentUser)
  && (
    <Route
      exact
      path="/admin/projects/:projectId/edit"
      component={
        withAuthorization(currentUser, ['
          project_management_add_edit_project_details'])(
          withFormEdit(ProjectForm)
        )}
    />
  )
}
{ currentUser.admin
  && (
    <Route exact path="/admin/managers_assignment" component={
      ManagersAssignment} />
  )
}
{ managerOrAdmin(currentUser)
  && (
    <Route exact path="/admin/departments" component={DepartmentsList} />
  )
}
{ managerOrAdmin(currentUser)
  && (
    <Route exact path="/admin/departments/:departmentId" component={Department}
    />
  )
}
{ managerOrAdmin(currentUser)
  && (
    <Route
      exact
      path="/admin/department/new/"
      component={
        withAuthorization(currentUser, ['
          departments_positions_management_departments_management'])(
          withFormNew(DepartmentForm)
        )}
    />
  )
}
{ managerOrAdmin(currentUser)
  && (
    <Route
      exact
      path="/admin/departments/:departmentId/edit"
      component={
        withAuthorization(currentUser, ['
          departments_positions_management_departments_management'])(
          withFormEdit(DepartmentForm)
        )}
    />
  )
}

```

```

    )
  }
  { managerOrAdmin(currentUser)
    && (
      <Route exact path="/admin/positions" component={PositionsList} />
    )
  }
  { managerOrAdmin(currentUser)
    && (
      <Route exact path="/admin/positions/:positionId" component={Position} />
    )
  }
  { managerOrAdmin(currentUser)
    && (
      <Route
        exact
        path="/admin/position/new"
        component={
          withAuthorization(currentUser, ['
            departments_positions_management_positions_management'])(
            withFormNew(PositionForm)
          )
        }
      />
    )
  }
  { managerOrAdmin(currentUser)
    && (
      <Route
        exact
        path="/admin/positions/:positionId/edit"
        component={
          withAuthorization(currentUser, ['
            departments_positions_management_positions_management'])(
            withFormEdit(PositionForm)
          )
        }
      />
    )
  }

  { managerOrAdmin(currentUser)
    && (
      <Route exact path="/admin/questions" component={QuestionsList} />
    )
  }
  { managerOrAdmin(currentUser)
    && (
      <Route exact path="/admin/questions/:questionId" component={Question} />
    )
  }
  { managerOrAdmin(currentUser)
    && (
      <Route
        exact
        path="/admin/question/new/"
        component={

```

```

        withAuthorization(currentUser, ['
            departments_positions_management_positions_management'])(
            withFormNew(QuestionForm)
        )}
    />
)
}
{ managerOrAdmin(currentUser)
  && (
    <Route
      exact
      path="/admin/questions/:questionId/edit"
      component={
        withAuthorization(currentUser, ['
            departments_positions_management_positions_management'])(
            withFormEdit(QuestionForm)
        )}
    />
  )
}

{ managerOrAdmin(currentUser)
  && (
    <Route
      exact
      path="/admin/rights"
      component={
        withAuthorization(currentUser, ['rights_management_rights_management'])
        (Rights)}
    />
  )
}

{ managerOrAdmin(currentUser)
  && (
    <Route exact path="/admin/skills" component={Skills} />
  )
}
{ managerOrAdmin(currentUser)
  && (
    <Route exact path="/admin/category/new" component={withFormNew(CategoryForm)} />
  )
}
{ managerOrAdmin(currentUser)
  && (
    <Route exact path="/admin/categories/:categoryId/edit" component={
      withFormEdit(CategoryForm)} />
  )
}

<Route exact path="/current_user/edit" component={CurrentUserEditPage} />
<Route exact path="/current_user" component={CurrentUserPage} />

<Route component={NotFound} />

```



```

        </Switch>
      </div>
    </div>
  </BrowserRouter>
);
}

}

const mapStateToProps = (state) => {
  return {
    currentUser: state.currentUser,
  };
};

export default connect(mapStateToProps)(App);

//profile.js

import moment from 'moment'
import validator from 'validator'
import _ from 'lodash'

export class Company {
  constructor() {
    this.start_year = {value: '', label: ''}
    this.end_year = {value: '', label: ''}
    this.name = ''
  }
}

export class Project {
  constructor({company=''}) {
    this.name = ''
    this.description = ''
    this.responsibilities = ''
    this.start_date = moment()
    this.end_date = ''
    this.period = ''
    this.role = ''
    this.technologies = []
    this.company = company
    this.project = {value: '', label: ''}
  }
}

export class Education {
  constructor() {
    this.start_year = {value: '', label: ''}
    this.end_year = {value: '', label: ''}
    this.universiity_courses = ''
    this.specialization = ''
    this.degree_acquired = ''
  }
}

```

```

    }
}

export class Skill {
  constructor({name='', group='', experience='', level='', last_used=''}) {
    this.name = name
    this.group = group
    this.experience = experience
    this.level = level
    this.last_used = last_used
  }
}

export class Profile {
  constructor({email=''}) {
    this["personal_information_name"] = '',
    this["personal_information_surname"] = '',
    this["personal_information_birth_date"] = moment('14/02/1994', 'DD/MM/YYYY'),
    this["personal_information_living_place"] = '',
    this["personal_information_phone"] = '',
    this["personal_information_email"] = email,

    this["education_study"] = [],
    this["education_level_of_english"] = {value: '', label: ''},

    this["professional_profile_department"] = {value: '', label: ''},
    this["professional_profile_position"] = {value: '', label: ''},
    this["professional_profile_office"] = {value: '', label: ''},
    this["professional_profile_questions_answers"] = [],
    this["additional_skills_skills"] = '',

    this.companies = [],
    this.previous_projects = [],
    this.sumato_projects = []

    this.skills_matrix = {}
  }
}

export class ProfileRulesBuilder {

  constructor(profile, step, technologiesList, profileStatus) {
    this.profile = profile;
    this.step = step;
    this.technologiesList = technologiesList;
    this.profileStatus = profileStatus;
  }

  rules() {
    switch(this.step) {
      case 0:
        return [...this.general(), ...this.education(), ...this.questions()]
      case 1:
        return [...this.companies(), ...this.previous_projects()]
      case 2:

```

```

        return [...this.sumato_projects()]
    case 3:
        return [...this.skills_matrix()]
    }
}

general() {
    return [
        {
            field: 'personal_information_name',
            method: (field) => { return !_isEmpty(field) || this.profileStatus === 'copy'; },
            validWhen: true,
            message: 'Name is required.'
        },
        {
            field: 'personal_information_surname',
            method: (field) => { return !_isEmpty(field) || this.profileStatus === 'copy'; },
            validWhen: true,
            message: 'Surname is required.'
        },
        {
            field: 'personal_information_phone',
            method: (val) => { return validator.isMobilePhone(val, 'any') || this.
                profileStatus === 'copy'; },
            validWhen: true,
            message: 'Phone is invalid.'
        },
        {
            field: 'education_level_of_english',
            method: (field) => !_isEmpty(field.value),
            validWhen: false,
            message: 'Level of english is required.'
        },
        {
            field: 'professional_profile_department',
            method: (field) => !_isEmpty(field.label),
            validWhen: false,
            message: 'Department is required.'
        },
        {
            field: 'professional_profile_position',
            method: (field) => !_isEmpty(field.label),
            validWhen: false,
            message: 'Position is required.'
        }
    ]
}

education() {
    return _.flatten(this.profile["education_study"].map((edu, ind) => {
        return [
            {
                field: 'education_study[${ind}].start_year',
                method: (field) => !_isEmpty(field.value),
                validWhen: false,

```

```

      message: 'Start year is required.'
    },
    {
      field: 'education_study[${ind}].end_year',
      method: (field) => {
        return !_.isEmpty(field.value) &&
          this.profile["education_study"][ind].start_year.value <= this.profile["
            education_study"][ind].end_year.value
      },
      validWhen: true,
      message: 'End year is invalid.'
    },
    {
      field: 'education_study[${ind}].university_courses',
      method: 'isEmpty',
      validWhen: false,
      message: 'Field is required.'
    },
    {
      field: 'education_study[${ind}].specialization',
      method: 'isEmpty',
      validWhen: false,
      message: 'Specialization is required.'
    },
    {
      field: 'education_study[${ind}].degree_acquired',
      method: 'isEmpty',
      validWhen: false,
      message: 'Degree Acquired is required.'
    }
  ]
})))
}

questions() {
  return _.flatten(this.profile["professional_profile_questions_answers"].map((qa, ind)
    => {
      return [
        {
          field: 'professional_profile_questions_answers[${ind}].answer',
          method: 'isEmpty',
          validWhen: false,
          message: 'Question answer is required.'
        }
      ]
    }
  ))
}

companies() {
  return _.flatten(this.profile["companies"].map((company, ind) => {
    return [
      {
        field: 'companies[${ind}].start_year',
        method: (field) => _.isEmpty(field.value),
        validWhen: false,

```

```

        message: 'Start year is required.'
      },
      {
        field: 'companies[${ind}].end_year',
        method: (field) => {
          return !_.isEmpty(field.value) &&
            this.profile["companies"][ind].start_year.value <= this.profile["companies"]
              [ind].end_year.value
        },
        validWhen: true,
        message: 'End year is invalid.'
      },
      {
        field: 'companies[${ind}].name',
        method: 'isEmpty',
        validWhen: false,
        message: 'Name is required.'
      }
    ]
  }
))
}

```

```

previous_projects() {
  return _.flatten(this.profile["previous_projects"].map((project, ind) => {
    return [
      {
        field: 'previous_projects[${ind}].name',
        method: 'isEmpty',
        validWhen: false,
        message: 'Name is required.'
      },
      {
        field: 'previous_projects[${ind}].description',
        method: 'isEmpty',
        validWhen: false,
        message: 'Description is required.'
      },
      {
        field: 'previous_projects[${ind}].responsibilities',
        method: 'isEmpty',
        validWhen: false,
        message: 'Responsibilities is required.'
      },
      {
        field: 'previous_projects[${ind}].role',
        method: 'isEmpty',
        validWhen: false,
        message: 'Role is required.'
      },
      {
        field: 'previous_projects[${ind}].technologies',
        method: (field) => {
          return !_.isEmpty(this.technologiesList) ? !_.isEmpty(field) : true
        },
        validWhen: true,

```

```

        message: 'Technologies are required.'
      },
      {
        field: 'previous_projects[${ind}].company',
        method: (field) => _.isEmpty(field.value),
        validWhen: false,
        message: 'Company is required.'
      },
      {
        field: 'previous_projects[${ind}].end_date',
        method: (field) => {
          return (_.isEmpty(field) || moment(field).isAfter(moment(this.profile["
            previous_projects"][ind].start_date)))
        },
        validWhen: true,
        message: 'End date must be after start date.'
      }
    ]
  }
}))
}

```

```

sumato_projects() {
  return _.flatten(this.profile["sumato_projects"].map((project, ind) => {
    return [
      {
        field: 'sumato_projects[${ind}].project',
        method: field => _.isEmpty(field.label),
        validWhen: false,
        message: 'Project is required.'
      },
      {
        field: 'sumato_projects[${ind}].responsibilities',
        method: 'isEmpty',
        validWhen: false,
        message: 'Responsibilities is required.'
      },
      {
        field: 'sumato_projects[${ind}].role',
        method: 'isEmpty',
        validWhen: false,
        message: 'Role is required.'
      },
      {
        field: 'sumato_projects[${ind}].technologies',
        method: field => {
          return !_.isEmpty(this.technologiesList) ? !_.isEmpty(field) : true
        },
        validWhen: true,
        message: 'Technologies are required.'
      },
      {
        field: 'sumato_projects[${ind}].end_date',
        method: (field) => {
          return (_.isEmpty(field) || moment(field).isAfter(moment(this.profile["
            sumato_projects"][ind].start_date)))
        }
      }
    ]
  })
}

```

```

    },
    validWhen: true,
    message: 'End date must be after start date.'
  }
]
}))
}

skills_matrix() {
  return _.flatten(this.profile["skills_matrix"].list.map((skill, ind) => {
    return [
      {
        field: 'skills_matrix.list[${ind}].name',
        method: field => {
          return _.filter(this.profile["skills_matrix"].list.map((skill, ind) => {return
            skill.name}), field).length > 1
        },
        validWhen: false,
        message: 'Skills must be unique.'
      },
      {
        field: 'skills_matrix.list[${ind}].name',
        method: field => _.isEmpty(field.value) ,
        validWhen: false,
        message: 'Name is required.'
      },
      {
        field: 'skills_matrix.list[${ind}].level',
        method: field => _.isEmpty(field.value),
        validWhen: false,
        message: 'Level is required.'
      },
      {
        field: 'skills_matrix.list[${ind}].experience',
        method: (field) => {
          return validator.isInt(field.toString(), {min: 0, max: 100})
        },
        validWhen: true,
        message: 'Experience must be a number.'
      },
      {
        field: 'skills_matrix.list[${ind}].last_used',
        method: field => {
          return _.isEmpty(field.label.toString())},
        validWhen: false,
        message: 'Last used year is required.'
      }
    ]
  }))
}

}

// ProfileWizard.jsx

```

```

import React from 'react';
import { withRouter } from 'react-router-dom';
import { connect } from 'react-redux';
import { Wizard, Steps, Step } from 'react-albus';
import moment from 'moment';
import classNames from 'classnames';
import { ToastStore } from 'react-toasts';
import _ from 'lodash';
import ProfileWizardStep1 from './ProfileWizardStep1';
import ProfileWizardStep2 from './ProfileWizardStep2';
import ProfileWizardStep3 from './ProfileWizardStep3';
import ProfileWizardStep4 from './ProfileWizardStep4';
import ProfilePreview from '../pages/ProfilePreview';
import { Profile, Skill, ProfileRulesBuilder } from './Profile';
import { getUsersList } from '../../actions';
import { updateProfile, uploadProfilePhoto } from '../../api/profiles';
import { updateProfileAndModerate } from '../../api/profiles_moderation';
import { FormValidator } from '../validations/FormValidator';
import Transition from './Transition';
import WizardNav from './WizardNav';
import { fetchPositionQuestions } from '../../api/positions';

class ProfileWizard extends React.Component {

  constructor(props) {
    super(props);

    const steps = [
      {
        title: 'Step 1',
        description: 'Personal Information',
        id: 'personal_info',
      },
      {
        title: 'Step 2',
        description: 'Previous Experience',
        id: 'previous_experience',
      },
      {
        title: 'Step 3',
        description: 'SumatoSoft Experience',
        id: 'sumato_experience',
      },
      {
        title: 'Step 4',
        description: 'Skills Matrix',
        id: 'skills_matrix',
      },
      {
        title: 'Step 5',
        description: 'Profile Preview',
        id: 'profile_preview',
      }
    ];
  }

```



```

const { profile, email } = this.props;
const state = {
  profileInfo: !_.isEmpty(profile.info) ? profile.info : new Profile({ email }),
  steps,
  validation: null,
  currentStepInd: 0,
  visitedStepsMap: steps.map((_step, ind) => { return ind === 0; }),
  photo: profile.photo,
  file: null,
};

this.state = state;

this.onNext = this.onNext.bind(this);
this.onPrevious = this.onPrevious.bind(this);

this.onInputChange = this.onInputChange.bind(this);
this.onArrayExtend = this.onArrayExtend.bind(this);
this.onArrayItemChange = this.onArrayItemChange.bind(this);
this.setQuestionsAnswers = this.setQuestionsAnswers.bind(this);
this.onDepartmentChange = this.onDepartmentChange.bind(this);
this.generateSkillsMatrix = this.generateSkillsMatrix.bind(this);
this.onSkillsMatrixItemChange = this.onSkillsMatrixItemChange.bind(this);
this.onArrayItemRemove = this.onArrayItemRemove.bind(this);

this.sumatoProjectsDataToProfile = this.sumatoProjectsDataToProfile.bind(this);

this.technologiesOptions = this.technologiesOptions.bind(this);
this.positionsOptions = this.positionsOptions.bind(this);
this.departmentsOptions = this.departmentsOptions.bind(this);

this.isFrontOffice = this.isFrontOffice.bind(this);
this.onGoTo = this.onGoTo.bind(this);
this.markStateAsVisited = this.markStateAsVisited.bind(this);

this.addSkill = this.addSkill.bind(this);
this.deleteSkill = this.deleteSkill.bind(this);

this.onPhotoSelected = this.onPhotoSelected.bind(this);
this.onCrop = this.onCrop.bind(this);
}

onPhotoSelected(files) {
  if (files && files[0]) {
    const [file] = files;
    file.blob = URL.createObjectURL(files[0]);
    this.setState({ file, photo: null });
  }
}

onCrop(cropper) {
  cropper.getCroppedCanvas().toBlob((blob) => {
    const formData = new FormData();
    const { file } = this.state;

```

```

const { profile } = this.props;
formData.append('photo', blob, `${file.name.split('.')[0]}_${moment().format()}.jpg`);
uploadProfilePhoto(profile.id, formData)
  .then((res) => {
    this.setState({ photo: res.data.photo, file: null });
  });
}, 'image/jpeg');
}

static getSkillLevel(experience) {
  switch (experience) {
    case '0':
    case '1':
      return 'novice';
    case '2':
      return 'intermediate';
    case '3':
      return 'advanced';
    default:
      return 'expert';
  }
}

onNext(context) {
  const { currentStepInd, steps } = this.state;
  this.validate(() => {
    const { validation } = this.state;
    const { editedByOwner } = this.props;
    if (validation.isValid) {
      if (editedByOwner && this.isLastStep()) {
        this.sendToModeration();
      } else if (!editedByOwner && this.isLastStep()) {
        this.managerProfileUpdate();
      } else {
        this.saveProfile();
      }
    }
    const transition = new Transition({
      context,
      steps,
      position: this.getPosition(),
      isFrontOffice: this.isFrontOffice(),
      currentStepInd,
    });

    const nextStepId = transition.next();
    this.markStateAsVisited(nextStepId);
    this.setState({ currentStepInd: nextStepId });
  });
}

onPrevious(context) {
  const { currentStepInd, steps } = this.state;
  const transition = new Transition({

```

```

    context,
    steps,
    position: this.getPosition(),
    isFrontOffice: this.isFrontOffice(),
    currentStepInd,
  });

  const nextStepId = transition.prev();
  this.setState({ currentStepInd: nextStepId });
}

onGoTo(context, stepId) {
  const { currentStepInd, steps, visitedStepsMap } = this.state;
  const transition = new Transition({
    context,
    steps,
    position: this.getPosition(),
    isFrontOffice: this.isFrontOffice(),
    currentStepInd,
    visitedStepsMap,
  });
  const nextStepId = transition.goTo(stepId);
  this.setState({ currentStepInd: nextStepId });
}

onInputChange(event, callback) {
  const { profileInfo } = this.state;
  const newProfileInfo = { ...profileInfo, [event.target.name]: event.target.value };

  if (!_isFunction(callback)) {
    this.setState({ profileInfo: newProfileInfo }, callback);
  } else {
    this.setState({ profileInfo: newProfileInfo });
  }
}

onArrayExtend(arr, newItem) {
  const { profileInfo } = this.state;
  const newProfileInfo = { ...profileInfo };
  newProfileInfo[arr] = [...newProfileInfo[arr], newItem];

  this.setState({ profileInfo: newProfileInfo });
}

onArrayItemRemove(arr, ind) {
  const { profileInfo } = this.state;
  const newProfileInfo = { ...profileInfo };
  newProfileInfo[arr].splice(ind, 1);
  this.setState({ profileInfo: newProfileInfo });
}

onArrayItemChange(arr, ind, field, value) {
  const { profileInfo } = this.state;
  const newProfileInfo = { ...profileInfo };
  newProfileInfo[arr][ind][field] = value;
}

```

```

    this.setState({ profileInfo: newProfileInfo });
  }

onSkillsMatrixItemChange(ind, field, value) {
  const { profileInfo } = this.state;
  const newProfileInfo = { ...profileInfo };
  const skill = newProfileInfo.skills_matrix.list[ind];
  skill[field] = value;

  if (field === 'name') {
    const technology = _.find(this.chosenPositionTechnologiesList(), (tech) => {
      return tech.skill.name === value.value;
    });
    skill.group = technology.skill.group;
    newProfileInfo.skills_matrix.groups = _.uniq(newProfileInfo.skills_matrix.list.map((
      skill) => {
        return skill.group;
      }
    ));
  }

  if (field === 'experience' && !_.isEmpty(value)) {
    const level = ProfileWizard.getSkillLevel(skill.experience);
    skill.level = { value: level, label: level };
  }
  this.setState({ profileInfo: newProfileInfo });
}

// onTechnologyCreate(_projectInd, newTechnology) {
// const { dispatch } = this.props;
// dispatch(createTechnology({ group: '', name: newTechnology }));
// }

onDepartmentChange() {
  const { profileInfo } = this.state;
  const newProfileInfo = { ...profileInfo };
  const department = this.getDepartment();
  newProfileInfo.professional_profile_position = { label: '', value: '' };
  newProfileInfo.professional_profile_office = department.office;
  this.setState({ profileInfo: newProfileInfo }, () => { return this.
    setQuestionsAnswers(); });
}

getPosition() {
  const { profileInfo } = this.state;
  const { positionsList } = this.props;
  const positionId = _.get(profileInfo.professional_profile_position, 'value', '');
  return _.find(positionsList, { id: positionId });
}

getDepartment() {
  const { profileInfo } = this.state;
  const { departmentsList } = this.props;
  const departmentId = _.get(profileInfo.professional_profile_department, 'value', '');
  return _.find(departmentsList, { id: departmentId });
}

```

```

setQuestionsAnswers() {
  const { profileInfo } = this.state;
  const departmentId = _.get(profileInfo.professional_profile_department, 'value', null);
  const positionId = _.get(profileInfo.professional_profile_position, 'value', null);
  const newProfileInfo = { ...profileInfo };
  if (_.isNumber(departmentId) && _.isNumber(positionId)) {
    const { positionsList } = this.props;
    const position = _.find(positionsList, { id: positionId });
    if (position.no_skills_matrix) {
      newProfileInfo.skills_matrix = {};
    }

    if (this.isFrontOffice()) {
      newProfileInfo.previous_projects = [];
      newProfileInfo.sumato_projects = [];
    }
    let questionsAnswers = [];
    fetchPositionQuestions(positionId).then((res) => {
      questionsAnswers = res.data.map((question) => {
        return {
          question: question.description,
          answer: '',
          answer_sample: question.answer_sample,
          summary: question.summary,
        };
      });
      newProfileInfo.professional_profile_questions_answers = questionsAnswers;
      this.setState({ profileInfo: newProfileInfo });
    });
  } else {
    newProfileInfo.professional_profile_questions_answers = [];
    this.setState({ profileInfo: newProfileInfo });
  }
}

positionsOptions() {
  const { profileInfo } = this.state;
  const { positionsList } = this.props;
  const departmentId = _.get(profileInfo.professional_profile_department, 'value', '');
  return positionsList.filter((position) => { return position.department_id ===
    departmentId; })
    .map((position) => {
      return { value: position.id, label: position.name };
    });
}

sumatoProjectsDataToProfile() {
  const { profileInfo } = this.state;
  const newProfileInfo = { ...profileInfo };
  const { projectsList } = this.props;

  newProfileInfo.sumato_projects.forEach((sumatoProject) => {
    const projectInfo = _.find(projectsList, { id: sumatoProject.project.value });

```

```

    if (!_isEmpty(projectInfo)) {
      // eslint-disable-next-line no-param-reassign
      sumatoProject.name = projectInfo.title;
      // eslint-disable-next-line no-param-reassign
      sumatoProject.description = projectInfo.description;
    }
  });
  this.setState({ profileInfo: newProfileInfo });
}

deleteSkill(ind) {
  const { profileInfo } = this.state;
  const newProfileInfo = { ...profileInfo };
  newProfileInfo.skills_matrix.list.splice(ind, 1);
  newProfileInfo.skills_matrix.groups = _.uniq(newProfileInfo.skills_matrix.list.map((
    skill) => {
      return skill.group;
    }
  )));

  this.setState({ profileInfo: newProfileInfo });
}

addSkill() {
  const { profileInfo } = this.state;
  const newProfileInfo = { ...profileInfo };
  const group = 'UNCATEGORIZED';
  const skill = new Skill({
    name: { value: '', label: '' },
    group,
    experience: '',
    level: { value: '', label: '' },
    lastUsed: { value: '', label: '' },
  });
  newProfileInfo.skills_matrix.list.push(skill);
  newProfileInfo.skills_matrix.groups = _.union(newProfileInfo.skills_matrix.groups, [
    group]);

  this.setState({ profileInfo: newProfileInfo });
}

generateSkillsMatrix() {
  const { profileInfo } = this.state;
  const newProfileInfo = { ...profileInfo };
  const projects = [...newProfileInfo.sumato_projects, ...newProfileInfo.
    previous_projects];
  const technologiesNames = _.uniqBy(
    _.flatten(projects.map((project) => {
      return project.technologies.map((technology) => {
        return technology.value;
      }
    }
  ))
  );
  const technologies = this.chosenPositionTechnologiesList().filter((technology) => {
    return technologiesNames.includes(technology.skill.name);
  });

```

```

newProfileInfo.skills_matrix.groups = _.uniq(
  technologies.map((technology) => {
    return !_isEmpty(technology.skill.group) ? technology.skill.group : '
      UNCATEGORIZED';
  })
);
newProfileInfo.skills_matrix.list = [];
technologies.forEach((technology) => {
  const projectsWithTech = projects.filter((project) => {
    return _.find(project.technologies, { value: technology.skill.name });
  });
  const firstUsed = moment(moment.min(projectsWithTech.map((project) => {
    return moment(project.start_date);
  })));
  const lastUsed = moment(moment.max(projectsWithTech.map((project) => {
    return _.isEmpty(project.end_date) ? moment() : project.end_date;
  })));
  let experience = moment.duration(lastUsed.diff(firstUsed)).years() || 1;
  experience = experience.toString();

  const level = ProfileWizard.getSkillLevel(experience);

  const skill = new Skill({
    name: { label: technology.skill.name, value: technology.skill.name },
    group: technology.skill.group,
    experience,
    level: { value: level, label: level },
    last_used: { value: lastUsed.year(), label: lastUsed.year() },
  });
  newProfileInfo.skills_matrix.list.push(skill);
});
this.setState({ profileInfo: newProfileInfo });
}

nextStep() {
  const { currentStepInd, steps } = this.state;
  return steps[currentStepInd + 1];
}

saveProfile() {
  const { editedByOwner, profile } = this.props;
  const { profileInfo } = this.state;
  const status = editedByOwner ? 'draft' : profile.status;
  updateProfile(profile.id, profileInfo, status);
}

sendToModeration() {
  const { profile } = this.props;
  const { profileInfo } = this.state;
  updateProfileAndModerate(profile.id, profileInfo, 'tech')
    .then(() => {
      ToastStore.success('Profile sent to Tech manager');
      this.navigateAfterSave();
    });
}

```

```

navigateAfterSave() {
  const { dispatch, history, navigateAfter } = this.props;
  dispatch(getUsersList());
  history.push(navigateAfter);
}

managerProfileUpdate() {
  const { profile } = this.props;
  const { profileInfo } = this.state;
  if (profile.status === 'tech') {
    updateProfileAndModerate(profile.id, profileInfo, 'hr')
      .then(() => {
        ToastStore.success('Profile sent to HR manager');
        this.navigateAfterSave();
      });
  } else if (profile.status === 'hr') {
    updateProfileAndModerate(profile.id, profileInfo, 'sales')
      .then(() => {
        ToastStore.success('Profile sent to Sales manager');
        this.navigateAfterSave();
      });
  } else if (profile.status === 'sales') {
    updateProfile(profile.id, profileInfo, 'approved')
      .then(() => {
        ToastStore.success('Profile is approved');
        this.navigateAfterSave();
      });
  } else if (profile.status === 'copy') {
    updateProfile(profile.id, profileInfo, 'copy')
      .then(() => {
        ToastStore.success('Profile copy was updated');
        this.navigateAfterSave();
      });
  } else if (profile.status === 'approved') {
    updateProfile(profile.id, profileInfo, 'approved')
      .then(() => {
        ToastStore.success('Profile is updated');
        this.navigateAfterSave();
      });
  }
}

isFrontOffice() {
  const { profileInfo } = this.state;
  return profileInfo.professional_profile_office === 'Front Office';
}

isLastStep() {
  const { currentStepInd, steps } = this.state;
  return currentStepInd === steps.length - 1;
}

markStateAsVisited(stepInd) {
  const { visitedStepsMap } = this.state;

```



```

    visitedStepsMap[stepInd] = true;
    this.setState({ visitedStepsMap });
  }

  chosenPositionTechnologiesList() {
    const { profileInfo } = this.state;
    const { technologiesList } = this.props;

    return technologiesList.filter((technology) => {
      return technology.technologable.id === profileInfo.professional_profile_position.
        value;
    });
  }

  validate(callback) {
    const { profileInfo, currentStepInd } = this.state;
    const { profile } = this.props;
    const technologyList = this.chosenPositionTechnologiesList();
    const rulesBuilder = new ProfileRulesBuilder(profileInfo, currentStepInd,
      technologyList, profile.status);
    const validator = new FormValidator(rulesBuilder.rules());
    const validation = validator.validate(profileInfo);

    this.setState({ validation }, callback);
  }

  technologiesOptions() {
    const { profileInfo } = this.state;
    const { technologiesList } = this.props;
    return technologiesList.filter((technology) => {
      return technology.technologable.id === profileInfo.professional_profile_position.
        value;
    }).map((technology) => {
      return { value: technology.skill.name, label: technology.skill.name };
    });
  }

  departmentsOptions() {
    const { departmentsList } = this.props;
    return departmentsList.map((department) => {
      return { value: department.id, label: department.name };
    });
  }

  renderNavButtons(context) {
    const { currentStepInd } = this.state;

    return (
      <div className="row">
        {
          currentStepInd !== 0
          && (
            <div className="col-md-6">
              <button
                onClick={() => { this.onPrevious(context); }}

```

```

        type="button"
        className="btn btn-primary btn-lg pull-left"
      >
        Previous
      </button>
    </div>
  )
}
{
  <div className={classNames({ 'col-md-6': currentStepInd !== 0, 'col-md-12':
    currentStepInd === 0 })}>
    <button
      onClick={() => { this.onNext(context); }}
      type="button"
      className="btn btn-primary btn-lg pull-right"
    >
      Next
    </button>
  </div>
}
</div>
);
}

```

```

render() {
  const {
    technologiesList,
    projectsList,
    positionsList,
    departmentsList,
    currentUser,
  } = this.props;

  const {
    currentStepInd, visitedStepsMap, steps, profileInfo, validation, photo, file,
  } = this.state;
  const { editedByOwner, profile } = this.props;

  const propsReady = !_.isEmpty(technologiesList)
    && !_.isEmpty(projectsList)
    && !_.isEmpty(positionsList)
    && !_.isEmpty(departmentsList)
    && !_.isEmpty(currentUser);

  return propsReady && (
    <div className="row wizard-container">
      <div className="col-md-offset-2 col-md-8">
        <Wizard>
          <WizardNav
            currentStepId={currentStepInd}
            onGoTo={this.onGoTo}
            visitedStepsMap={visitedStepsMap}
            steps={steps}
          />

```

```

<Steps>
  <Step
    id="personal_info"
    render={(context) => {
      return (
        <div className="step-view">
          <ProfileWizardStep1
            profileInfo={profileInfo}
            lastChanges={profile.last_changes}
            onInputChange={this.onInputChange}
            onArrayExtend={this.onArrayExtend}
            onArrayItemChange={this.onArrayItemChange}
            onArrayItemRemove={this.onArrayItemRemove}
            setQuestionsAnswers={this.setQuestionsAnswers}
            positionsOptions={this.positionsOptions}
            departmentsOptions={this.departmentsOptions}
            validation={validation}
            onDepartmentChange={this.onDepartmentChange}
            editedByOwner={editedByOwner}
            photo={photo}
            onPhotoSelected={this.onPhotoSelected}
            onCrop={this.onCrop}
            blob={_.get(file, 'blob')}
          />
          {this.renderNavButtons(context)}
        </div>
      );
    }}
  />
  <Step
    id="previous_experience"
    render={(context) => {
      return (
        <div className="step-view">
          <ProfileWizardStep2
            profileInfo={profileInfo}
            lastChanges={profile.last_changes}
            onInputChange={this.onInputChange}
            onArrayExtend={this.onArrayExtend}
            onArrayItemChange={this.onArrayItemChange}
            onArrayItemRemove={this.onArrayItemRemove}
            onTechnologyCreate={this.onTechnologyCreate}
            technologiesOptions={this.technologiesOptions}
            validation={validation}
            isFrontOffice={this.isFrontOffice}
            editedByOwner={editedByOwner}
          />
          {this.renderNavButtons(context)}
        </div>
      );
    }}
  />
  <Step
    id="sumato_experience"
    render={(context) => {

```

```

return (
  <div className="step-view">
    <ProfileWizardStep3
      profileInfo={profileInfo}
      lastChanges={profile.last_changes}
      onChange={this.onChange}
      onArrayExtend={this.onArrayExtend}
      onArrayItemChange={this.onArrayItemChange}
      onArrayItemRemove={this.onArrayItemRemove}
      onTechnologyCreate={this.onTechnologyCreate}
      technologiesOptions={this.technologiesOptions}
      projectsList={projectsList}
      sumatoProjectsDataToProfile={this.sumatoProjectsDataToProfile}
      validation={validation}
      editedByOwner={editedByOwner}
    />
    {this.renderNavButtons(context)}
  </div>
);
}}
/>
<Step
  id="skills_matrix"
  render={(context) => {
    return (
      <div className="step-view">
        <ProfileWizardStep4
          profileInfo={profileInfo}
          lastChanges={profile.last_changes}
          onSkillsMatrixItemChange={this.onSkillsMatrixItemChange}
          generateSkillsMatrix={this.generateSkillsMatrix}
          technologiesOptions={this.technologiesOptions}
          addSkill={this.addSkill}
          deleteSkill={this.deleteSkill}
          editedByOwner={editedByOwner}
          validation={validation}
        />
        {this.renderNavButtons(context)}
      </div>
    );
  }}
/>
<Step
  id="profile_preview"
  render={(context) => {
    return (
      <div className="step-view">
        <ProfilePreview
          profile={
            {
              info: profileInfo,
              status: profile.status,
              photo,
            }
          }
        />
      </div>
    );
  }}
/>

```

```

        />
        {this.renderNavButtons(context)}
    </div>
    );
  }}
  />
</Steps>
</Wizard>
</div>
</div>
);
}

}

const mapStateToProps = (state) => {
  return {
    technologiesList: state.technologiesList,
    projectsList: state.projectsList,
    positionsList: state.positionsList,
    departmentsList: state.departmentsList,
    currentUser: state.currentUser,
  };
};

export default connect(mapStateToProps)(withRouter(ProfileWizard));

```