Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«**СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**»

Хакасский технический институт – филиал ФГАОУ ВО
«Сибирский федеральный университет»

Кафедра прикладной информатики, математики и естественно-научных дисциплин

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Е. Н. Скуратенко
подпись
«____» _____ 2022 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.03 Прикладная информатика

Создание диалогового тренажера для студии веб-разработки «SibDev»

Руководитель _____ доцент, канд. пед. наук И. В. Янченко
подпись, дата

Выпускник _____ В. В. Олейник
подпись, дата

Консультанты
по разделам:

Экономический _____ Е. Н. Скуратенко
подпись, дата

Нормоконтролер _____ В. И. Кокова
подпись, дата

Абакан 2022

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«**СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**»

Хакасский технический институт – филиал ФГАОУ ВО
«Сибирский федеральный университет»

Кафедра прикладной информатики, математики и естественно-научных дисциплин

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Е. Н. Скуратенко
подпись
«____» _____ 2022 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме бакалаврской работы

Абакан 2022

Студенту Олейник Владимиру Викторовичу

Группа ХБ 18-03

Направление 09.03.03 Прикладная информатика

Тема выпускной квалификационной работы: Создание диалогового тренажера для студии веб-разработки «SibDev»

Утверждена приказом по институту № 208 от 14.04.2022 г.

Руководитель ВКР: И. В. Янченко, доцент, кандидат педагогических наук кафедры ПИМ и ЕД ХТИ – филиала СФУ

Исходные данные для ВКР: материалы полученные в ходе преддипломной практики о деятельности студии веб-разработки; данные о процессе обучения сотрудников и деятельности тренера; данные о требованиях к тренажеру (тип, форма заданий тренажера и др).

Перечень разделов ВКР:

1. Анализ предметной области.
2. Описание разработки диалогового тренажера.
3. Оценка экономической эффективности разработки и внедрения диалогового тренажера.

Перечень графического материала: нет

Руководитель ВКР _____ И. В. Янченко
подпись

Задание принял к исполнению _____ В. В. Олейник
подпись

«14» апреля 2022 г.

РЕФЕРАТ

Выпускная квалификационная работа по теме «Создание диалогового тренажера для студии веб-разработки SibDev» содержит 120 страниц текстового документа, 77 иллюстраций, 12 таблиц, 16 формул и 18 использованных источников.

ДИАЛОГОВЫЙ ТРЕНАЖЕР, МЕНЕДЖЕР ПО ПРОДАЖАМ, СЕРВЕР, БАЗА ДАННЫХ, ЯЗЫКИ ПРОГРАММИРОВАНИЯ, ВЕБ ПРИЛОЖЕНИЕ, REACT, GRAPHQL, GO, TYPESRIPT, IDEF0, IDEF3, КЛИЕНТ-СЕРВЕР, ТСО, КАПИТАЛЬНЫЕ ЗАТРАТЫ.

Объектом бакалаврской работы является деятельность студии веб-разработки «SibDev» по обучению менеджеров.

Целью бакалаврской работы является разработка диалогового тренажера для обучения менеджеров студии веб-разработки «SibDev».

Первый раздел посвящен вопросам анализа предметной области. В нем рассмотрены понятие диалогового тренажера, его предназначение и актуальность разрабатываемой информационной системы. Помимо этого, был проведен анализ средств разработки и на его основе были выбраны следующие средства: Go, TypeScript, React, MongoDB.

Во втором разделе описывается разработка диалогового тренажера, которая сопровождается скриншотами исходного кода и получившегося пользовательского интерфейса. Была разработана как серверная, так и клиентская части, которые вместе составили цельный диалоговый тренажер.

В третьем разделе выполнена оценка экономической эффективности разработки и внедрения диалогового тренажера. По её итогам стало ясно, что разработка экономически обоснована и срок окупаемости составит около 9 месяцев. Разработанный диалоговый тренажер внедрен в эксплуатацию в студии веб-разработки «SibDev».

SUMMARY

The theme of the graduation thesis is «Dialogue Simulator Design for SibDev Web Development Studio». It contains 120 pages of a text document, 77 figures, 12 tables, 18 references.

DIALOGUE SIMULATOR, SALES MANAGER, SERVER, DATABASE, PROGRAM LAGUAGES, WEB APPLICATION, REACT, GRAPHQL, GO, TYPESCRIPT, IDEF0, IDEF3, CLIENT-SERVER, TCO, CAPITAL COSTS.

The object of the graduation paper is the SibDev Web Development Studio performance for manager training.

The purpose of the graduation thesis: to develop a dialogue simulator for the SibDev Web Development Studio.

The first section of the thesis is devoted to the analysis of the subject area. It considers the concept of a dialogue simulator, its purpose and the relevance of the information system being developed. In addition, an analysis of development tools has been carried out, and the following tools have been selected: Go, TypeScript, React, MongoDB.

The second section describes the development of the dialogue simulator, which is accompanied by screenshots of the source code and the resulting user interface. Both a server and a client sides have been developed, which together have made up an integral dialogue simulator.

The third section assesses the economic efficiency of the development and implementation of the dialogue simulator. According to the results, it is clear that its development is economically feasible, and the payback period will be about 9 months. The developed dialogue simulator has been put into operation in the SibDev Web Development Studio.

English language supervisor: _____ N.V. Chezybaeva

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Диалоговые тренажеры применяются для отработки навыков у сотрудников организаций, основной деятельностью которых является общение и коммуникация. В студии веб-разработки «*SibDev*» это менеджеры по продажам. Менеджер по продажам – представитель студии, который осуществляет связь между заказчиком и студией. От его деятельности во многом зависит, заинтересуют ли предлагаемые услуги возможных заказчиков, поэтому обучение таких специалистов грамотному общению с заказчиками это важная задача. Диалоговый тренажер поможет в смоделированных условиях отвечать на подготовленные вопросы в течение ограниченного времени, научит формулировать шаблонные ответы на вопросы в соответствии со стандартами компании.

Объектом бакалаврской работы является деятельность студии веб-разработки «*SibDev*» по обучению менеджеров.

Предмет бакалаврской работы – технологии разработки веб-приложений.

Целью бакалаврской работы является разработка диалогового тренажера для обучения менеджеров студии веб-разработки «*SibDev*».

Для достижения цели бакалаврской работы необходимо решить задачи:

1. провести анализ деятельности студии веб-разработки «*SibDev*» по обучению менеджеров;

2. изучить функционал аналогичных программных продуктов;

3. спроектировать диалоговый тренажер методами структурного моделирования:

– модель «как будет»,

– модель «сущность-связь»,

– модель работы пользователя с информационной системой;

4. выполнить сравнительный анализ и определить средства разработки диалогового тренажера;

5. разработать серверную часть диалогового тренажера;

6. разработать клиентскую часть диалогового тренажера;

7. оценить экономическую эффективность разработки и внедрения диалогового тренажера;

8. сделать выводы.

При выполнении бакалаврской работы использованы общенаучные методы исследования: теоретические – анализ, синтез, сравнение, формализация информации; эмпирические методы – моделирование.

Практическая значимость состоит в том, что полученные результаты могут быть применены в деятельности студии веб-разработки *«SibDev»* по обучению сотрудников, а также другими компаниями.

Результаты бакалаврской работы были представлены на XVIII Международной конференции студентов, аспирантов и молодых ученых «Проспект Свободный–2022», посвященной Международному году фундаментальных наук в интересах устойчивого развития, г. Красноярск.

1 Анализ предметной области

1.1 Характеристика деятельности студии веб-разработки «SibDev»

SibDev – студия веб-разработки, офис которой расположен в городе Красноярске. Студия была создана в 2015 году. Основной деятельностью является разработка веб-сервисов (веб-приложений, сайтов) по заказу (заказная разработка). Миссией студии является обеспечение заказчика эффективным бизнес-инструментом в интернете, который будет приносить максимум пользы.

Студия гибко подходит к сотрудничеству с клиентами и на выбор предлагает несколько моделей сотрудничества:

1. *Разработка с нуля* – полный цикл разработки проекта «под ключ» от составления технического задания до полной реализации.

2. *Доработка и поддержка* – доработка и/или поддержка существующих систем.

3. *Аутсорс и аутстаф* – предоставление штатных веб-программистов на почасовую модель оплаты.

Студия в своей работе использует актуальное инструментальное программное обеспечение для разработки, которое позволяет создавать широкий спектр проектов: *Node JS*, *React*, *Vue*, *Python*, *Django*, *PostgreSQL*, *MongoDB* и другие.

Штат насчитывает около 60 сотрудников следующих специальностей:

1. менеджеры по персоналу;
2. менеджеры проектов;
3. менеджеры по продажам;
4. *backend* разработчики;
5. *frontend* разработчики;
6. тестировщики;
7. дизайнеры.

На рисунке 1.1 представлена организационная структура студии веб-разработки «SibDev».

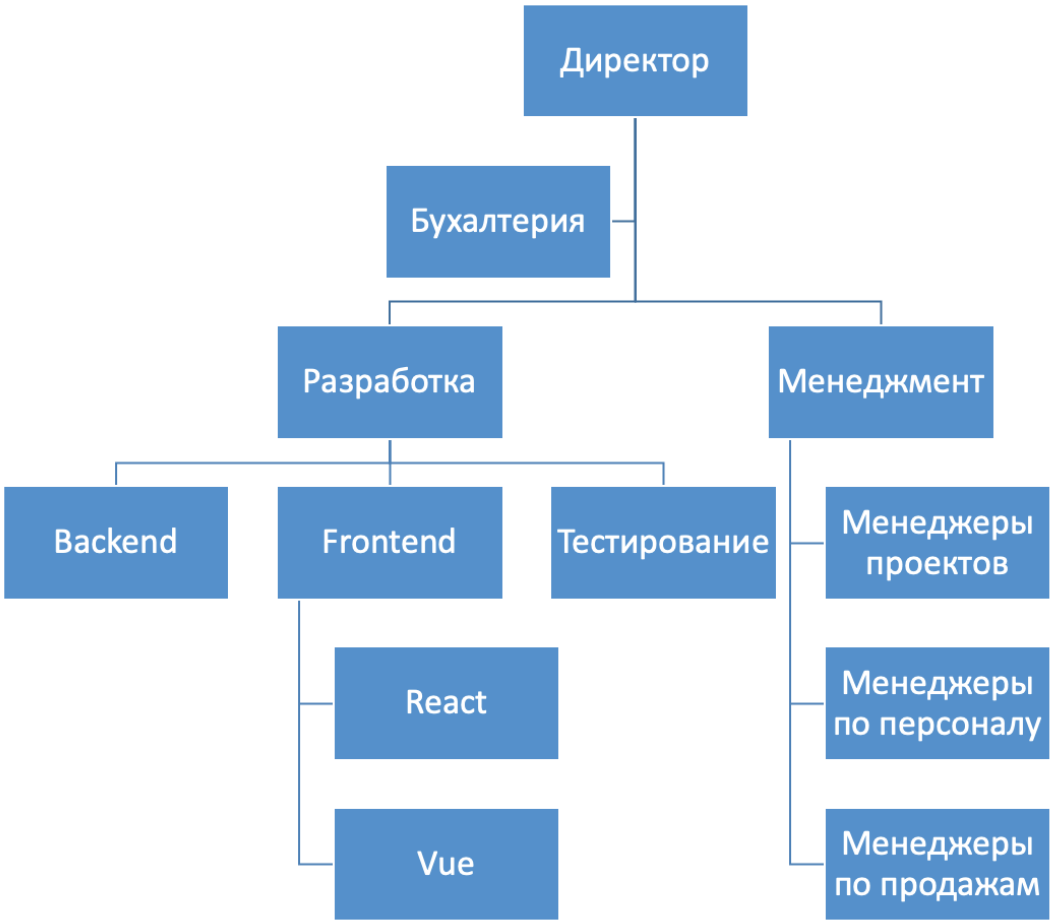


Рисунок 1.1 – Структурная схема студии веб-разработки «SibDev»

Выпускная квалификационная работа посвящена деятельности менеджеров и их тренингу, поэтому необходимо понимать их должностные обязанности:

1. Менеджер по продажам – основная задача такого менеджера – продажи услуг компании, расширение круга клиентов и поддержание партнерских отношений с ними. Большую часть своего рабочего времени специалист отдела продаж проводит в переговорах (письменных, телефонных или личных) [1]. От уровня подготовленности специалиста зависит количество полученных заказов.

Успешным результатом его работы является привлеченный клиент.

2. Менеджер проектов – связующее звено между заказчиком и командой разработки. Он отвечает за успешную реализацию идеи или задачи силами одной или нескольких команд, внутренних и/или внешних, в рамках бюджета и в установленные сроки. Для этого он организует работу команды, управляет ею – определяет краткосрочные цели и задачи, делегирует полномочия, взаимодействует с другими командами, заказчиками и подрядчиками [2]. Значительную часть рабочего времени проводит за общением с командой разработки и заказчиком. Результатом его успешной работы является сданный в срок заказчику программный продукт, отвечающий поставленным требованиям. Результат достигается за счет управления командой разработчиков.

3. Менеджер по персоналу (HR-специалист) – специалист, который осуществляет функции сопровождения процесса управления персоналом всех подразделений в организации: подбор и адаптация персонала, обучение, развитие, создание систем мотивации и стимулирования, осуществление комплекса работ по развитию корпоративной культуры, выстраивание эффективных бизнес-процессов и организация рабочих мест [3]. Значительную часть рабочего времени проводит за общением с работниками и/или соискателями свободных вакансий.

Рассмотрев обязанности нескольких менеджеров, можно сделать вывод, что всех их объединяет одно – успех их работы во многом зависит от степени развития коммуникативных навыков. Значительная часть их работы состоит из общения с другими людьми (заказчиками, клиентами и т.д.), поэтому, чем лучше они могут донести мысль, решить спорную ситуацию, чем больше у них опыта в типичных ситуациях, тем лучше они будут справляться с работой и тем лучше будут результаты у всей компании.

В улучшении этого навыка может помочь разрабатываемый диалоговый тренажер, который позволит менеджерам отрабатывать различные ситуации в

безопасной среде, не боясь, к примеру, потерять реальный заказ.

1.2 Предпосылки для разработки диалогового тренажера

На данный момент на рынке труда сохраняется дефицит IT-специалистов. По данным исследования портала по поиску работы *hh.ru* в начале 2021 года на одну IT-вакансию претендует всего до 2 кандидатов. Из того же исследования можно узнать, что IT-компании активно набирают специалистов из других сфер, например: маркетологов (13% от всех вакансий от IT-компаний) и менеджеров сферы продаж (11%) [4]. Очевидно, что в условиях нехватки кадров, компаниям достаточно сложно нанять менеджеров, которые бы обладали знаниями, специфичными для работы в IT-сфере. Поэтому, они готовы нанимать грамотных специалистов и обучать их специфике IT-сферы своими силами. Как было выяснено ранее, один из основных аспектов работы менеджеров – это общение и одним из инструментов, который позволяет развить коммуникативные навыки менеджера, является диалоговый тренажер.

Диалоговый тренажер – инструмент овладения коммуникативными навыками, позволяющий имитировать наиболее типичные ситуации, в которых может оказаться обучаемый в процессе общения с заказчиком. Пользователь выступает в роли некоего главного героя и вступает во взаимодействие, к примеру, с клиентом, покупателем, подчиненным или, наоборот, руководителем. Если сценарий для тренажера сделан достаточно качественно, пользователь отождествляет себя с главным героем и благодаря системе подкрепления и обратной связи осваивает требуемый коммуникативный навык. Именно поэтому диалоговые тренажеры считаются весьма эффективным инструментом, позволяющим «потренироваться», прежде чем вступать в реальное взаимодействие [5]. Основой сценария для задания диалогового тренажера обычно является конкретный бизнес-кейс из практики организации, в которой он применяется. Например, пользователю нужно выдать задание

подчиненному, ответить на вопросы недовольного клиента и др.

По данным исследования кадрового агентства «Рекадро» из опрошенных ими компаний, большинство, используют собственные разработки для обучения и развития сотрудников – корпоративные порталы (рис. 1.2).

Рисунок 1.2 – Методы, инструменты и технологии, используемые в обучении и развитии сотрудников

Отсюда можно сделать вывод, что существующие системы по каким-либо причинам не устраивает большинство компаний и они разрабатывают собственные решения. Помимо этого, государство считает приоритетной и актуальной задачей развитие отечественного рынка IT и специалистов в нём. Одним из направлений цифровой экономики РФ является развитие кадрового потенциала IT-отрасли.

1.3 Характеристика IT-инфраструктуры студии веб-разработки «SibDev»

Разрабатываемым диалоговым тренажером будут пользоваться менеджеры студии веб-разработки «SibDev». Основной их рабочий инструмент – компьютер или ноутбук, в зависимости от предпочтений. У компании нет стандартов для оборудования – каждый менеджер пользуется теми устройствами, которые он считает наиболее удобными в работе. Можно использовать как собственный компьютер, так и рабочий. Отсюда следует, что все сотрудники имеют разный набор устройств и их характеристики сильно варьируются. Разрабатываемая информационная система же должна работать на всех них полноценно и стабильно.

Так как имеется большой спектр устройств с различными операционными системами, установленными на них, оптимальным решением будет разрабатывать программный продукт как веб-приложение, ведь одно из главных преимуществ веб-приложений – совместимость со всеми существующими операционными системами «из коробки». Также необходимо определить перечень минимальных системных требований для работы диалогового тренажера. В таблице 1.1 представлен перечень минимальных требований, который был составлен с учетом анализа рынка компьютеров и ноутбуков, системных требования операционных систем и требуемого ПО для запуска веб-приложения.

Таблица 1.1 – Минимальные системные требования

Характеристика	Минимальные требования
Разрешение экрана	1280x720
Операционная система	Windows, Linux, MacOS
Оперативная память	2 ГБ
Процессор	не менее 1 ГГц
Браузер	Последние 5 версий браузеров основанных на Chromium (Google Chrome, Яндекс.Браузер и др.)

1.4 Анализ бизнес-процесса обучения менеджеров в студии веб-разработки «SibDev»

На данный момент обучение менеджеров в студии веб-разработки «SibDev» производится без средств автоматизации. Один из самых «больных» моментов – найм новых менеджеров. Дело в том, что несмотря на то, что все менеджеры должны иметь определенные навыки, уровень которых проверяется на собеседовании, после принятия на работу, их так или иначе все равно необходимо обучать. В каждой компании свой устоявшийся регламент по общению с клиентами и свои процессы. Так как речь идет об IT-сфере, то также разнится набор инструментов, языков программирования, фреймворков и т.д., применяющихся при работе над проектами. Все эти особенности необходимо знать менеджеру, хотя бы на базовом уровне. Обычно на процесс адаптации нового сотрудника выделяется один опытный коллега – тренер, который будет консультировать, отвечать на вопросы и помогать овладевать особенностями студии веб-разработки «SibDev». Очевидно, что будет выгодно, чтобы тренер тратил как можно меньше своего рабочего времени на нового сотрудника.

Бизнес-процесс обучения нового сотрудника «как есть» представлен на языке функционального моделирования в нотации *IDEF0* на рисунке 1.3.

Существующий бизнес-процесс обучения нового менеджера (рис. 1.4) имеет следующую последовательность:

1) Тренер выдает доступ новому менеджеру к базе знаний студии веб-разработки «SibDev» для ознакомления. В базе знаний описаны шаблонные конструкции всего того, что используется в компании: стиль общения, приемы коммуникации при ответах на вопросы клиентов и другое.

2) Менеджер изучает содержимое базы знаний. На этом этапе, после изучения информации, он не может быть уверен, что всё правильно понял или же что на практике сможет применить полученные знания верно.

3) После изучения материла базы знаний, менеджер связывается с

тренером для проверки знаний. Проверка происходит в форме устного общения с тренером. Тренер моделирует различные ситуаций, в которых может оказаться его новый коллега и смотрит на то, как он будет вести себя в них; задает вопросы по принятому стилю общения и прочим специфичным для студии моментам.

4) На основе этого общения дается обратная связь: что было хорошо, что нужно подтянуть.

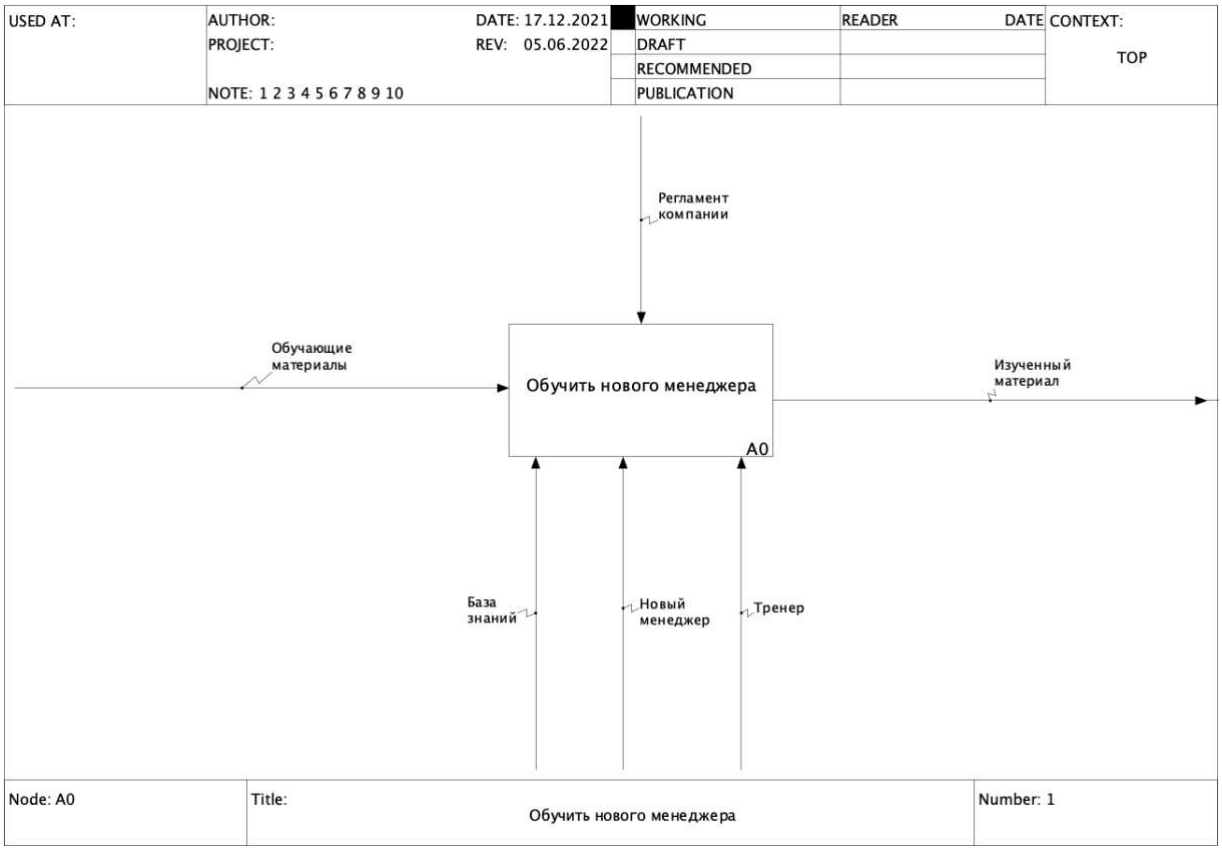


Рисунок 1.3 – *IDEF0* диаграмма обучения нового менеджера «как есть»

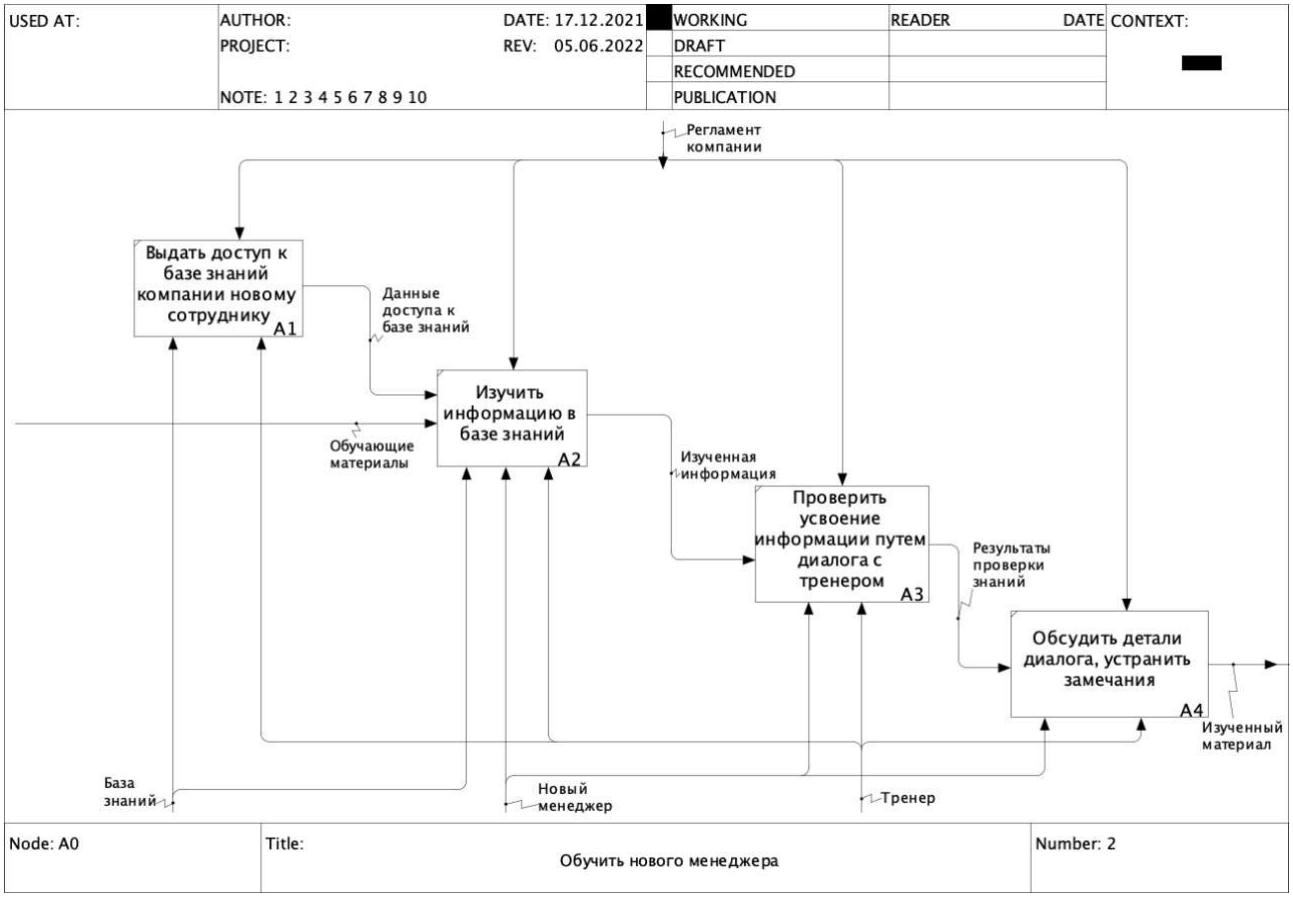


Рисунок 1.4 – Декомпозиция *IDEF0* диаграммы обучения нового сотрудника отдела продаж

После этого менеджер готов к работе и может приступать к выполнению своих обязанностей под надзором более опытных коллег.

1.5 Анализ аналогичных информационных систем

Разрабатываемая система не является полностью уникальной, на рынке уже представлены существующие диалоговые тренажеры. Есть как коммерческие, так и бесплатные решения. Анализ аналогов перед непосредственной разработкой позволяет выявить спектр типичного функционала для такого рода систем, посмотреть на положительные и отрицательные стороны существующих решений, а в некоторых случаях, даже

отказаться от разработки собственной реализации в пользу существующей. В данном разделе представлен анализ двух популярных решений:

– коммерческое от компании *iSpring*;

– бесплатное – *Online Test Pad*.

iSpring Suite – конструктор обучающих материалов, который позволяет создавать электронные курсы, тренажеры, тесты и т.д. в знакомой среде *PowerPoint* с последующим сохранением в *web*-формат.

Для создателя обучающих материалов система представляет из себя ряд вспомогательных инструментов в среде *PowerPoint* из офисного пакета *Microsoft*. В привычной для многих программе появляется дополнительная вкладка, которая представляет ряд инструментов, расширяющий функционал программы (рис. 1.5).

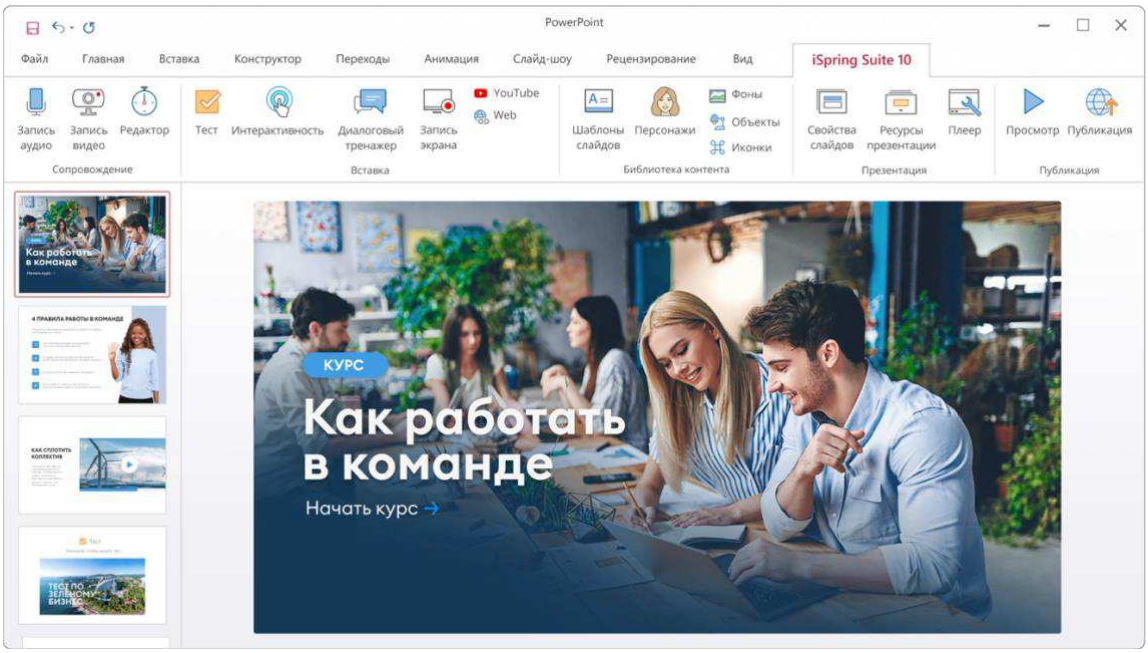


Рисунок 1.5 – Интерфейс *iSpring Suite*

iSpring Suite позволяет создавать: онлайн-курсы, тесты и опросы, видеокурсы, тренажеры и онлайн книги.

Для обучающегося же созданные материалы выглядят как отдельная веб-

страница. Ему не нужно ничего устанавливать дополнительно, достаточно только перейти по ссылке в браузере (рис. 1.6).

Из интересных особенностей можно отметить:

– каждая реплика может быть озвучена – выбранный вариант ответа и ответная реплика будут проиграны, если озвучка была запланирована при составлении тренинга;

– на рисунке 1.7 справа сверху можно заметить шкалу – это шкала настроения собеседника, по которой обучающийся может понимать в верную ли сторону он идет. Если шкала близится к зеленому – собеседник удовлетворён ответами, если она в красной зоне – собеседник сильно недоволен и нужно что-то с этим делать.

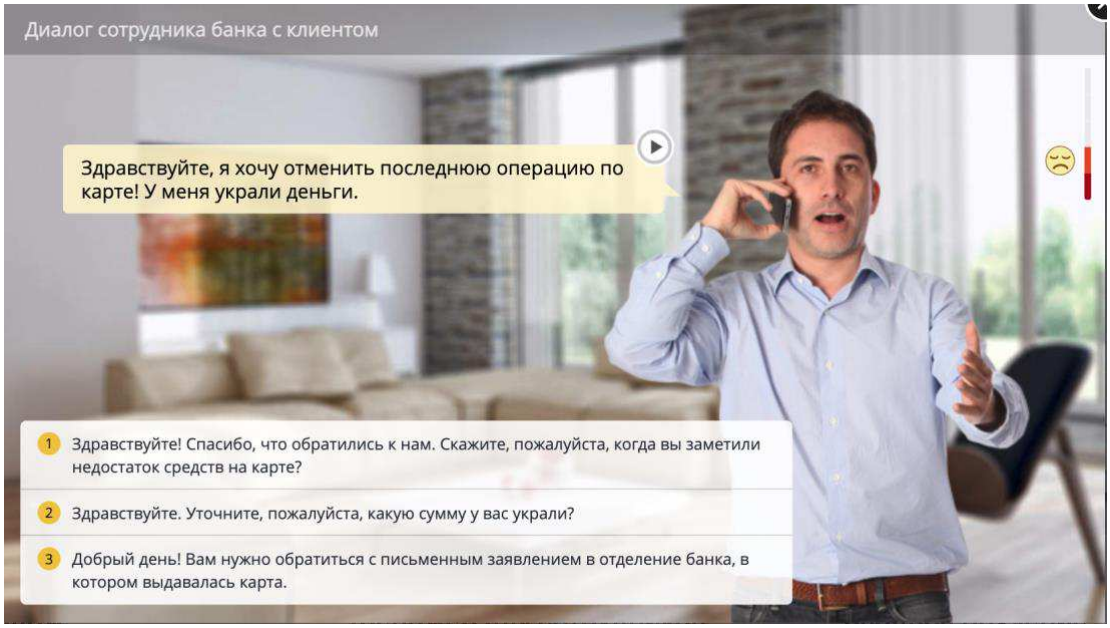


Рисунок 1.7 – Скриншот страницы прохождения тренажера *iSpring Suite*

Стоимость *iSpring Suite* составляет от 27000 до 47000 руб. в год (в зависимости от выбранной версии) за одного автора обучающих материалов. Помимо этого, нужно понимать, что также необходима лицензионная копия *Microsoft Office*, цена которого начинается от 13000 руб. в год для 5 устройств.

Также, одной из проблем является то, что *iSpring Suite* – это только конструктор обучающих материалов. Он позволяет создавать контент, но не предоставляет удобной платформы для его использования. Для этого у компании есть отдельный продукт – *iSpring Learn* – портал для онлайн-обучения сотрудников (рис. 1.8). Он предоставляет дополнительные возможности по отслеживанию статистики обучения, создания планов, управлению пользователями и т.д. Также он имеет глубокую интеграцию с конструктором *iSpring Suite*. Стоимость *iSpring Learn* составляет от 138 до 172 руб. в месяц за одного пользователя (цена зависит от общего количества пользователей – чем больше пользователей, тем меньше стоимость за одного пользователя).

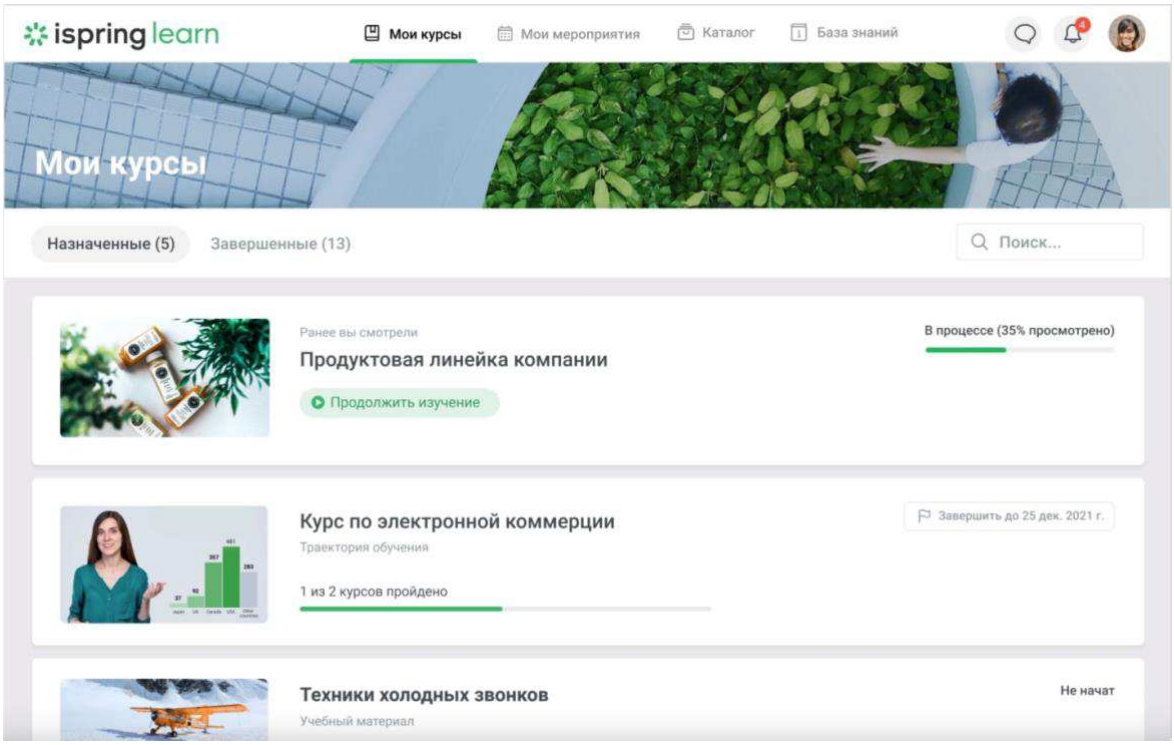


Рисунок 1.8 – Интерфейс *iSpring Learn*

Преимущества комплекса *iSpring*:

– конструктор обучающих материалов с обширными возможностями;

– наличие подробной документации и видео-уроков;

– корпоративный портал для онлайн-обучения с подробной статистикой;

– возможность добавления озвучки ко всем реплика тренинга;

– наличие шкалы настроения собеседника – она позволяет обучающему лучше понимать обстановку.

Недостатки комплекса *iSpring*:

– поддержка только *Microsoft PowerPoint* для создания материалов и необходимость его наличия для работы;

– поддержка только одной операционной системы для создания материалов – *Windows*;

– отсутствие единого тарифа, который бы включал в себя сразу 2 продукта – *iSpring Suite* и *iSpring Learn*.

Комплекс *iSpring* является хорошим профессиональным решением для создания платформы онлайн обучения сотрудников. Основной недостаток состоит в том, что при его выборе менеджеры, которые будут создавать обучающие материалы, будут вынуждены использовать только *Windows* и *Microsoft Office* в своей работе, что не так гибко, как могло бы быть.

Следующая аналогичная система – *Online Test Pad*. Это платформа по созданию тестов разных форматов, выполненная в виде веб-приложения.

Online Test Pad позволяет создавать: тесты; опросы; кроссворды; диалоговые тренажеры. В рамках обзора интересен только режим диалоговых тренажеров.

Платформа предлагает гибкий конструктор создания диалоговых тренажеров. Он состоит из нескольких этапов:

– начальная страница;

– сцены;

– последовательность.

Начальная страница. На этапе создания диалогового тренажера происходит заполнение основных данных о создаваемом тренажере, которая

показывается пользователю до начала его прохождения: название, инструкция к прохождению, можно добавить ввод дополнительных данных пользователя (например, имя и фамилию), а также автора и источник (рис. 1.9).

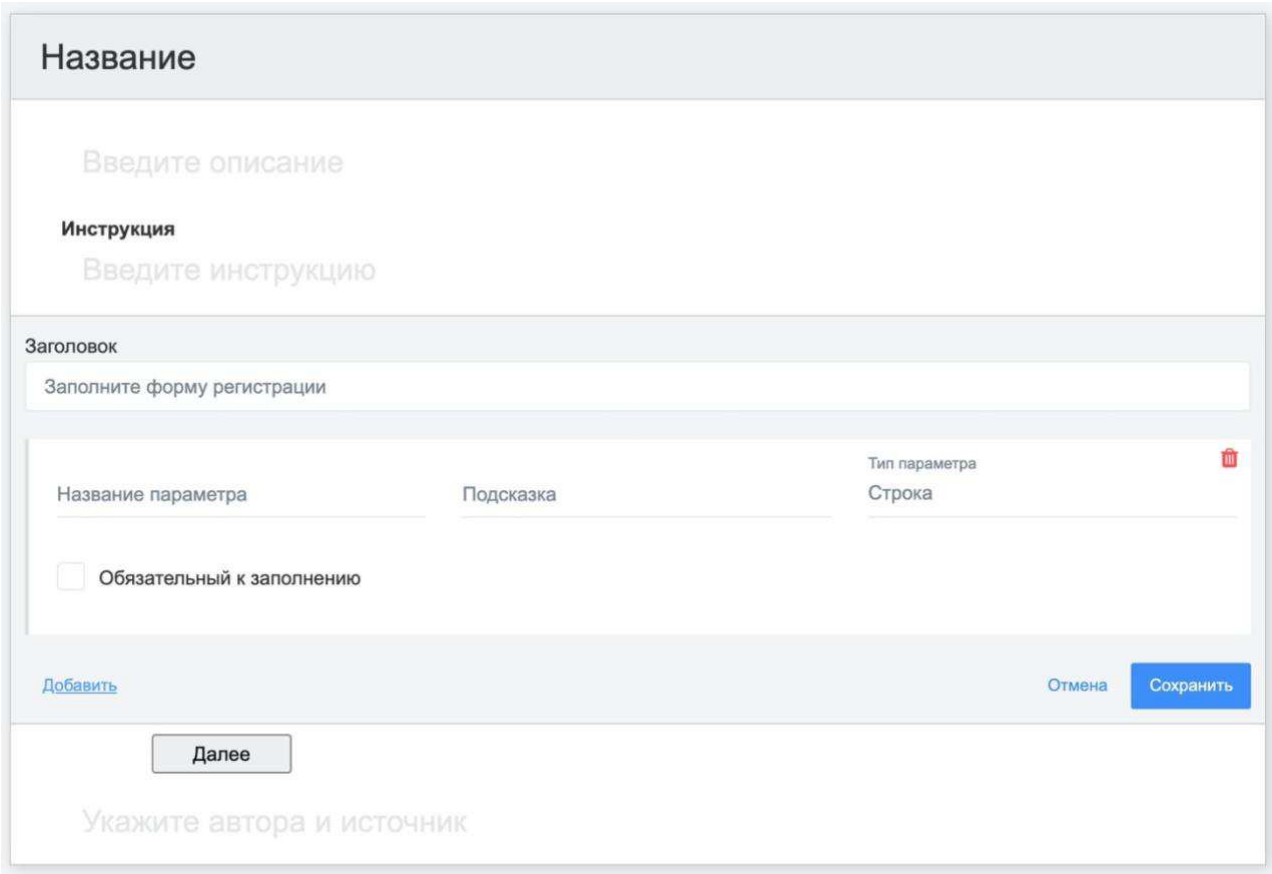


Рисунок 1.9 – Начальная страница создания диалогового тренажера

Сцены – это ситуации, происходящие в диалоге.

Конструктор позволяет задать:

– название сцены;

– текст фразы собеседника;

– варианты ответов на фразу.

За каждый из вариантов ответов можно выставлять баллы.

Помимо основного содержания есть возможность выбрать оформление – загрузить фоновую картинку и фотографию собеседника. Это делается для

повышения вовлеченности.

Также есть вкладка просмотра получившегося результата.

Скриншот страницы создания сцены в *Online Test Pad* представлен на рисунке 1.10.

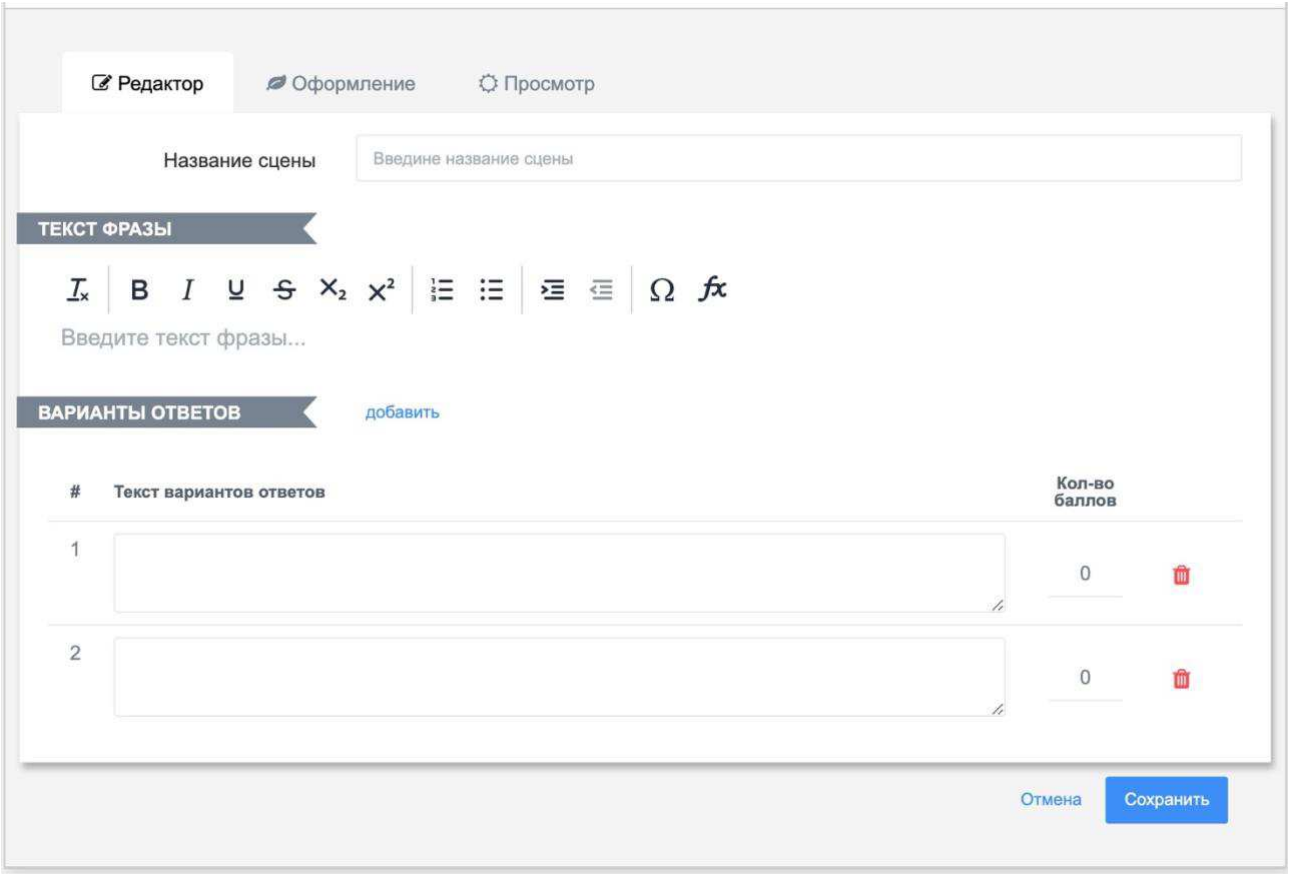


Рисунок 1.10 – Создание сцены

Последовательность. Данный пункт позволяет создать ветвление в «сюжете» диалога: можно добавить сцены и для каждого варианта ответа выбрать следующую сцену.

Возможность создания ветвления между сценариями добавляет нелинейности и позволяет перепроходить один тренажер несколько раз, чтобы попробовать пройти его разными путями. Это мощный инструмент в создании различных сценариев диалога (рис. 1.11).

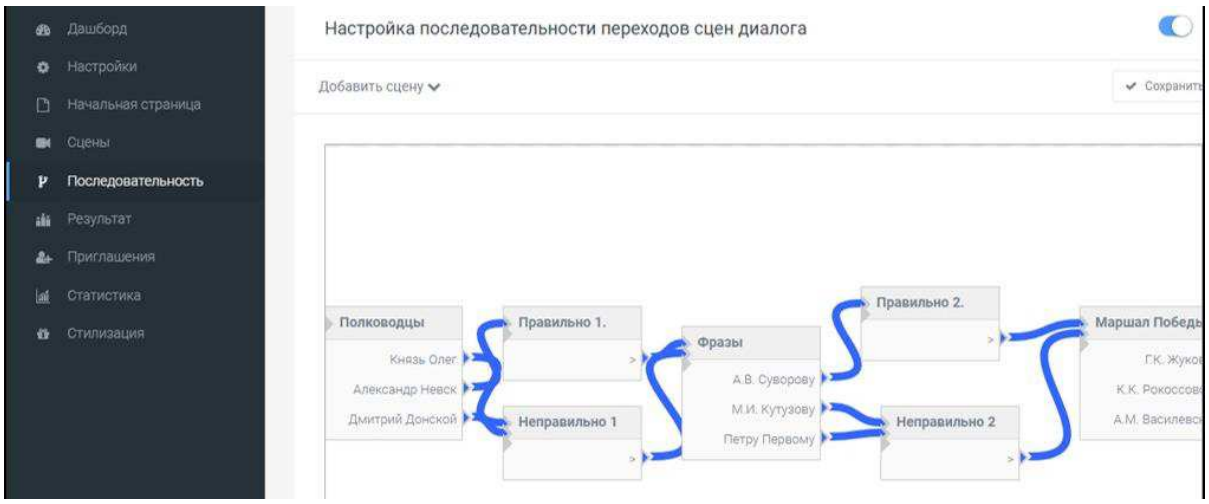


Рисунок 1.11 – Конструктор последовательности сцен

На этом создание тренажера в *Online Test Pad* завершено. Им можно поделиться по ссылке или открыть доступ для всех. При этом он предоставляет базовую статистику по прохождению тренажера. С точки зрения обучающегося, тренажер выглядит как веб-страница, на которой есть вопрос и несколько вариантов ответов (рис. 1.12).

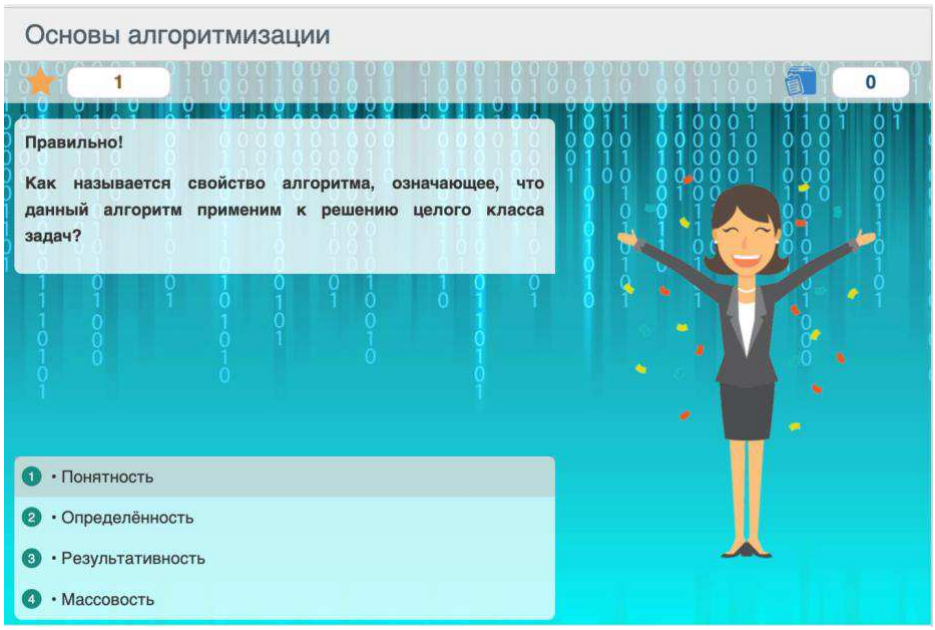


Рисунок 1.12 – Прохождение тренажера *Online Test Pad*

При работе с *Online Test Pad* можно выделить следующие преимущества:

1. простой и интуитивно понятный конструктор;
2. возможность создания тренажера с нелинейным сюжетом;
3. сервис предоставляется полностью бесплатно;
4. наличие статистики.

Но также есть и недостатки:

1. Сервис предлагает только один способ ограничения доступа к созданному тренажеру – доступ по специальной ссылке. Проблема заключается в том, что сервис не предоставляет возможности создать ссылку на список всех тренажеров, то есть у каждого тренажера будет своя отдельная ссылка. Поэтому придется где-то в отдельном месте хранить все ссылки и передавать их менеджерам, что не очень удобно.
2. Во время ознакомления сервис несколько раз был недоступен. Такая нестабильность может нарушить привычную работу компании, что однозначно не есть хорошо.
3. Нет какого-либо промежуточного результата – нельзя после выбора варианта ответа вывести какую-то подсказку. Также отсутствует шкала настроения собеседника.

Online Test Pad предоставляет продуманный конструктор диалоговых тренажеров, дает статистику прохождений и является бесплатным. Он отлично бы подошел для личного использования, но для корпоративного использования ему критически не хватает функционала по более точечному разграничению доступов. Также стоит учитывать нестабильность его работы.

Рассмотренные аналоги комплекс *iSpring* и *Online Test Pad* являются неплохими решениями, которые полностью или частично решают поставленную задачу, но при этом имеют ряд недостатков, которые не позволяют выбрать их в качестве диалогового тренажера в студии веб-разработки «SibDev». Комплекс *iSpring* построен на базе *Microsoft Office* и поддерживает только операционную систему *Windows* для создания

диалогового тренажера. Ограничение под одну операционную систему является критическим, так как менеджеры в студии используют устройства с различными операционными системами. *Online Test Pad* же на данном этапе своего развития не подходит для корпоративного использования – наблюдаемые перебои в работе и слабая система разграничения доступов к тренажерам не позволяют использовать его в студии веб-разработки «SibDev».

1.6 Моделирование информационной системы

Моделирование информационной необходимо для того, чтобы иметь четкое представление о разрабатываемой системе и о процессах, которые она автоматизирует. Моделирование позволит спроектировать и разобраться во взаимодействиях, происходящих внутри системы, потоках данных, протекающих через нее, и рассмотреть типичные пользовательские сценарии.

1.6.1 Модель «как будет»

В диалоговом тренажере будет присутствовать 2 типа пользователя: тренер – это старший менеджер, который проводит обучение и студент – менеджер, который проходит обучение.

Диаграмма в нотации *IDEF*0 «Как будет» строится для того, чтобы продемонстрировать заказчику и команде разработчиков, как будет выглядеть бизнес-процесс после внедрения разрабатываемой информационной системы (рис. 1.13). В функциональном блоке А0 отличий от диаграммы «как есть» практически нет, за исключением того, что добавилась стрелка механизма модели – «Диалоговый тренажер».

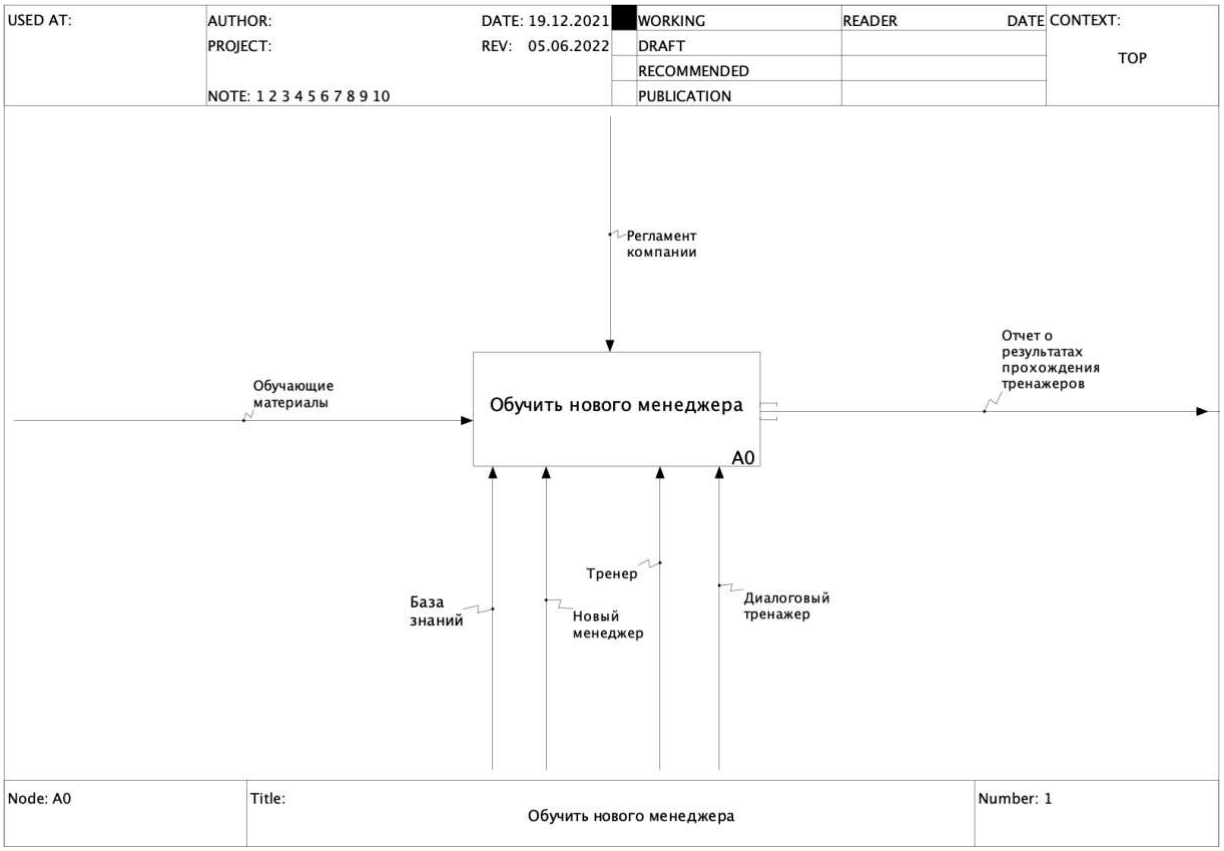


Рисунок 1.13 – Контекстная диаграмма «Как будет»

На рисунке 1.14 изображен декомпозированный процесс основного блока. Сразу можно заметить, что участие в процессе тренера значительно снизилось – он участвует только на первом и последнем этапах всего процесса. Первый этап «Выдать доступ к базе знаний и диалоговому тренажеру» предполагает создание тренером профиля нового сотрудника в базе знаний, передачу ему данных для входа и объяснение того, что он должен изучать и делать (обычно отправляется заготовленное заранее шаблонное сообщение, в котором заменяются данные для входа).

На втором этапе *A2* новый сотрудник (студент в рамках диалогового тренажера) изучает весь указанный тренером материал.

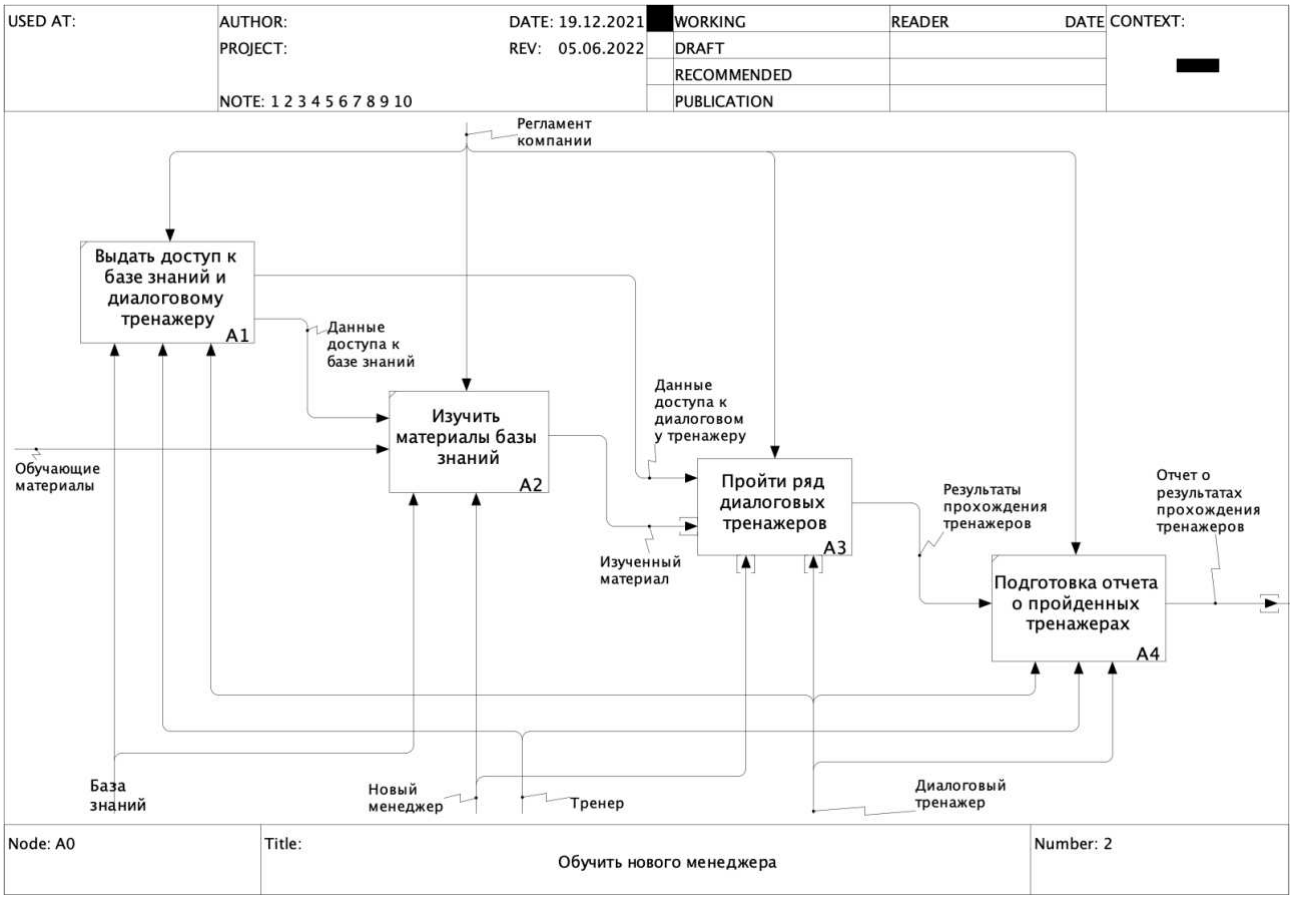


Рисунок 1.14 – Декомпозиция контекстной диаграммы «Как будет»

Более подробного рассмотрения требует блок A3 – «Пройти ряд диалоговых тренажеров» (рис. 1.15). Для начала студент должен пройти авторизацию в системе при помощи выданных ему ранее данных для входа. После того, как доступ к системе получен, он должен выбрать категорию тренажеров из списка. По плану, для адаптации новых менеджеров должна быть создана отдельная категория с тренажерами, пройдя которые, можно на практике закрепить все особенности работы в студии веб-разработки «SibDev». Затем из выбранной категории нужно выбрать конкретный тренажер и пройти его. Прохождение тренажера заключатся в выборе наиболее подходящих ответов в предлагаемом диалоге. После прохождения будут получены некие результаты. На основе результатов прохождения всех требуемых тренажеров формируется отчет по этому менеджеру. Тренер его анализирует и дает советы

новому менеджеру, получает обратную связь от него – что было не ясно и какие трудности возникли. На основе обратной связи контент тренажеров в дальнейшем может корректироваться.

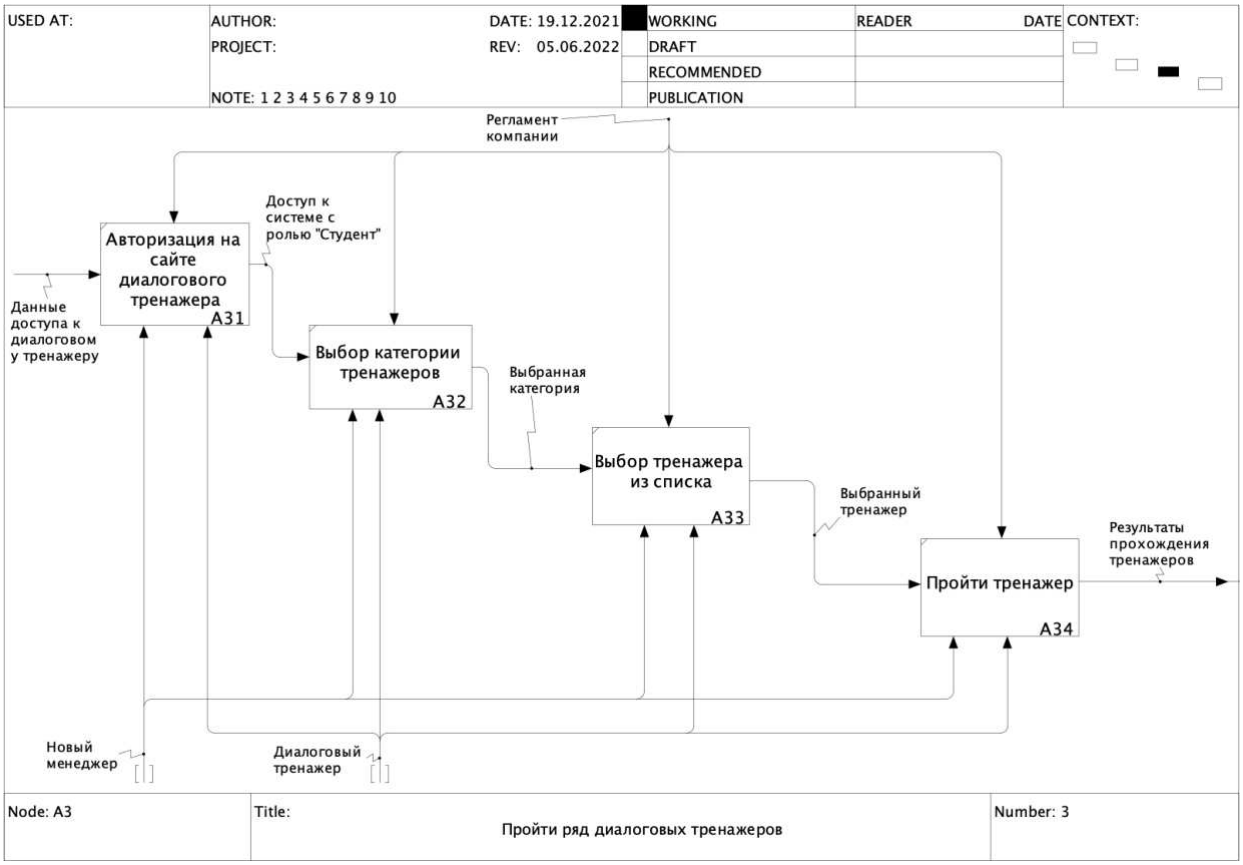


Рисунок 1.15 – Декомпозиция блока A4 диаграммы «Как будет»

На блоке A4 (рис. 1.14) тренер скачивает отчет, в котором будут все данные о результатах прохождения тренажеров студентом. На основе этих данных можно будет понять, с какими темами возникли трудности и что стоит в дальнейшем доработать.

1.6.2 Модель «сущность-связь»

Главное, чем оперирует каждая информационная система – это данные.

Поэтому важно знать, какие данные система будет обрабатывать и в каком виде их лучше всего хранить.

Если рассматривать структуру данных диалогового тренажера, то можно сделать вывод, что в общем случае она представляет из себя дерево, в котором родителем является начало диалога с вариантами ответов, а каждый из вариантов ответа может вести на другую ветвь диалога (рис. 1.16).

Рисунок 1.16 – Пример сценария диалогового тренажера

Для хранения древовидной структуры данных лучше подходят документоориентированные *NoSQL* СУБД (системы управления базами данных), ведь они специально предназначены для хранения иерархических структур данных.

На рисунке 1.17 изображена ER-диаграмма базы данных. Она использует особенности документоориентированных *NoSQL* СУБД, например, такую как встраивание.

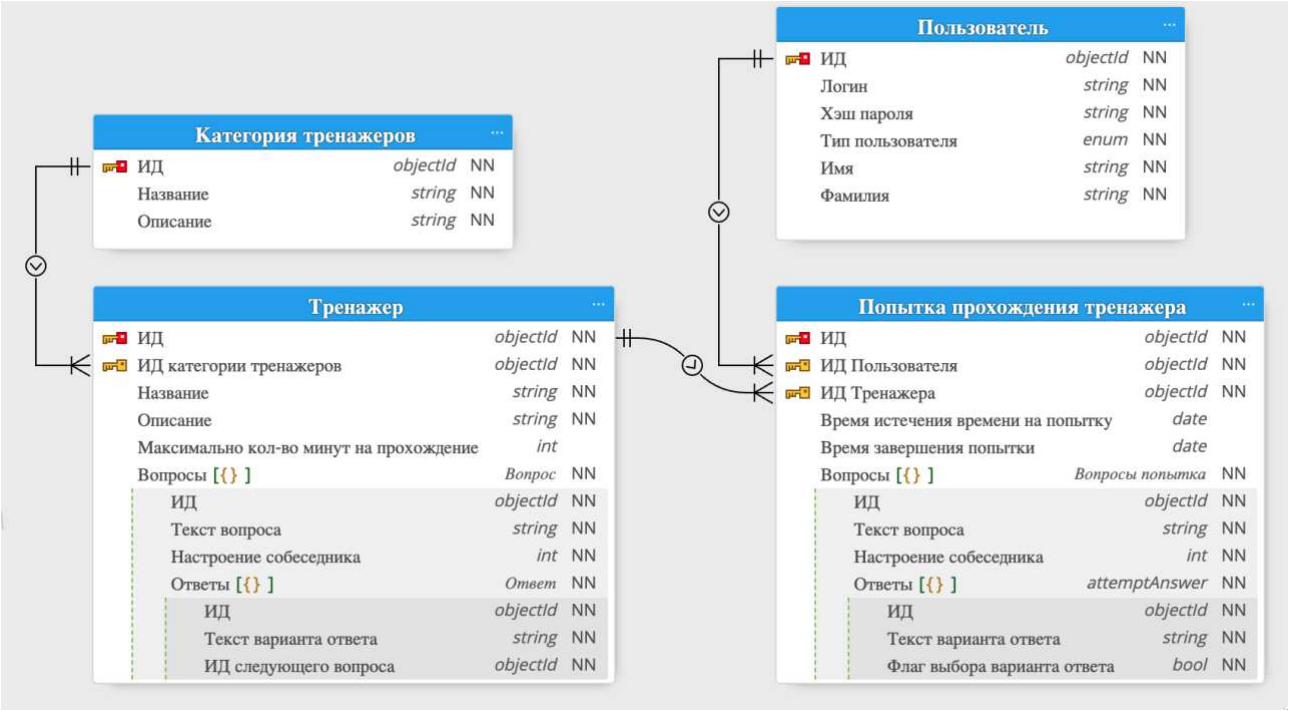


Рисунок 1.17 – ER-диаграмма

Если рассматривать диаграмму подробнее, то пояснений требуют сущности *Тренажер* и *Попытка прохождения тренажера*.

Тренажер хранит в себе помимо основной информации (названия, описания и др.) массив с вопросами. В каждом вопросе также хранится массив вариантов ответов на него. В варианте ответа есть одно функциональное поле, которое и позволяет выстраивать иерархию вопросов – *ИД следующего вопроса*. В нем хранится идентификатор следующего вопроса, который будет задан при выборе этого варианта ответа. Таким образом выстраивается некий односвязный список.

Попытка прохождения тренажера во многом копирует структуру документа *Тренажер* и это сделано намеренно. Дело в том, что для попытки прохождения нужно обязательно копировать текущие вопросы тренажера, ведь со временем сам тренажер могут редактировать – изменять вопросы и их последовательность. А в данном случае нужно сохранить состояние тренажера на момент его прохождения, чтобы в последующем, при просмотре этой

попытки, не было никаких изменений вне зависимости от сценария текущих вопросов тренажера, даже если он был изменен.

Также в вопросах тренажера есть одно поле, которое требует дополнительного внимания – это *Настроение собеседника*. Это целочисленный показатель, значения которого варьируются от 0 до 100. Это значение отображает насколько на текущем вопросе собеседника устраивают ответы обучаемого, где 100 – полностью устраивают, а 0 – полностью не устраивают. Это некая обратная связь, которая показывается студенту для понимания того, в правильном ли русле он движется. Он может видеть, какие ответы вызывают раздражения собеседников, а какие наоборот урегулируют ситуацию. Данный показатель также является результатом прохождения тренажера.

1.6.3 Модель работы пользователя с информационной системой

Каждая информационная система разрабатывается с тем учетом, что у нее будет свой пользователь. Важно при разработке информационной системы задумываться об удобстве создаваемого пользовательского интерфейса и основных сценариев использования. Для разработки пользовательских сценариев в системе служит диаграмма в нотации *IDEF3*. Так как система разрабатывается с учетом того, что у нее будет два типа пользователей: тренер и студент, то и сценариев использования будет тоже два.

Первая модель – модель работы тренера в системе (рис. 1.18). Первым этапом каждому из пользователей необходимо войти в систему. Без авторизации ничего сделать в системе нельзя. Регистрации также нет, доступ к системе выдается персонально для каждого менеджера. Роль тренера заключается в создании материалов для тренажеров. После входа пользователь попадает на страницу со списком созданных категорий тренажеров. Категории позволяют сгруппировать похожие по теме тренажеры вместе

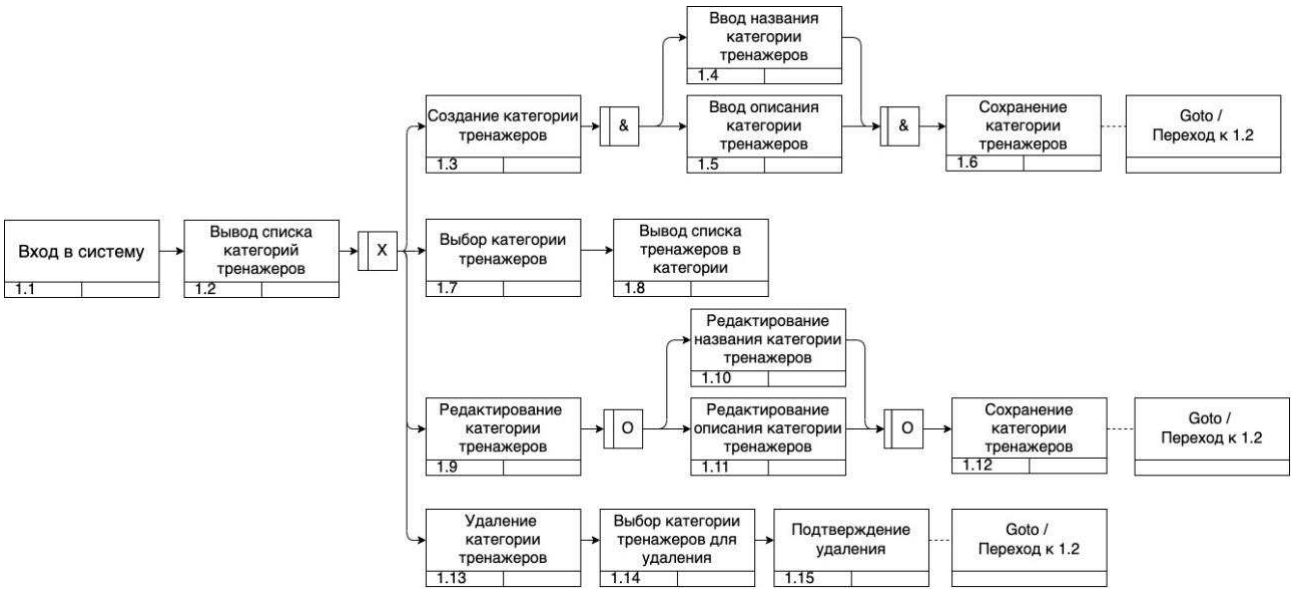


Рисунок 1.18 – Диаграмма работы тренера в информационной системе

Тренеру предлагается одно из следующих действий:

– создание категории тренажеров;

– редактирование категории тренажеров;

– удаление категории тренажеров;

– выбор категории тренажеров.

Для создания категории требуется:

– ввести название и описание категории;

– при редактировании можно изменить один или все эти параметры;

– при удалении выполняется удаление как самой категории, так и всех тренажеров в ней.

Выбор категории перенаправляет пользователя на страницу со списком тренажеров данной категории (рис. 1.19).

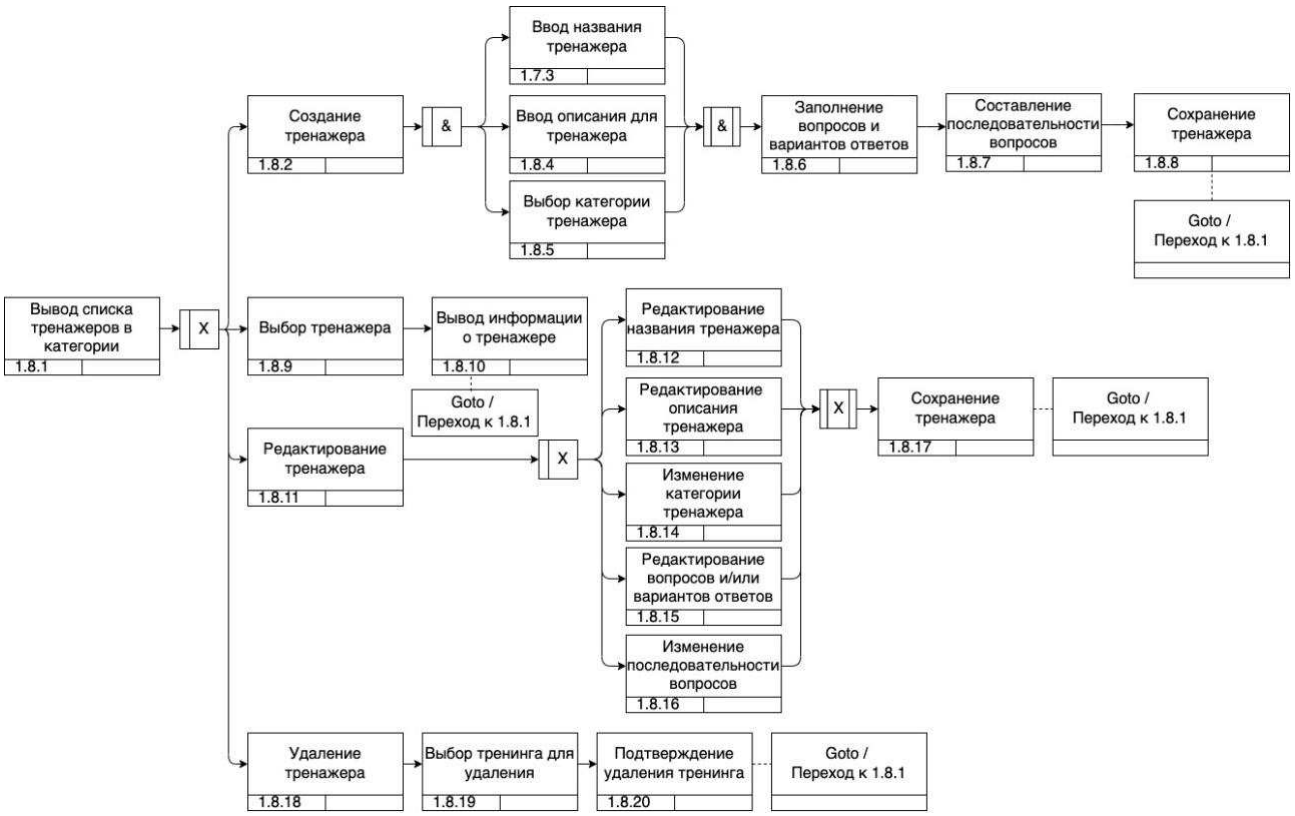


Рисунок 1.19 – Детализация блока тренинга «1.8.1 Вывод списка тренажеров в категории»

Здесь тренеру доступны все те же действия, что и с категориями, но теперь они относятся к самим тренажерам.

Отдельно стоит рассмотреть создание тренажера, так как это довольно нетривиальный процесс:

1) Тренеру необходимо заполнить название и описание тренажера, затем выбрать категорию, к которой он будет относиться.

2) Создается вопрос, варианты ответов к нему и задается уровень настроения собеседника на этом вопросе. После создания каждого следующего вопроса можно сразу задавать сценарий (последовательность) вопросов или же сделать это позже после создания некоторого перечня вопросов.

3) Составление сценария как последовательности вопросов имеет важное значение, так как при этом реализуется возможность задания нелинейности –

схемы, при которой при выборе одного варианта ответа следующим будет один вопрос, а при выборе другого варианта ответа – следующим будет другой вопрос. Таким образом, выполняется требование к диалоговому тренажеру по возможности создания нелинейных сценариев.

Редактирование позволяет изменить название, описание или вопросы тренажера и их последовательность.

С точки зрения второго типа пользователя – студента, система выглядит следующим образом (рис. 1.20). Его пользовательский сценарий гораздо более прямолинеен и сводится к одной главной цели – прохождению диалогового тренажера.

Рисунок 1.20 – Диаграмма работы студента в информационной системе

Для начала пользователю также необходимо войти в диалоговый тренажер. После этого открывается список доступных категорий тренажеров. Затем студент выбирает актуальную для него категорию и открывает список тренажеров категории. Среди них необходимо выбрать один. После этого открывается страница, в которой отображается подробная информация о тренажере и студенту предоставляется выбор – запустить тренажер или

посмотреть одну из своих предыдущих попыток прохождения (если таковые имеются). Прохождение тренажера заключатся во внимательном изучении вопросов и выборе на них наиболее подходящего вопроса. Это продолжается до тех пор, пока вопросы тренажера не закончатся.

1.7 Обоснование выбора средств разработки диалогового тренажера

Как было выяснено раньше, система должна работать на большом спектре устройств, которые могут иметь различные операционные системы. Исходя из этого, будет правильно разрабатывать систему как веб-приложение, ведь одно из главных преимуществ веб-платформы заключается в том, что одно приложение будет одинаково работать на всех существующих системах. Помимо этого, веб-приложения зачастую являются более удобными для пользователя – ведь не нужно ничего скачивать на свое устройство для работы, достаточно лишь перейти по ссылке, чтобы начать работу.

Веб-приложение – это клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются кроссплатформенными [6].

1.7.1 Выбор системы управления базами данных

Во время проектирования модели «сущность-связь» была проанализирована структура данных, которую необходимо хранить. В итоге было решено, что лучшим выбором системы управления базами данных (СУБД) будет использовать документоориентированную *NoSQL* СУБД.

Среди документоориентированных СУБД несомненным лидером является *MongoDB*, как самое популярное и наиболее используемое решение. По опросу портала *StackOverflow*, *MongoDB* является четвертой наиболее распространенной базой данных среди всех решений и первой среди *NoSQL* (рис. 1.21).

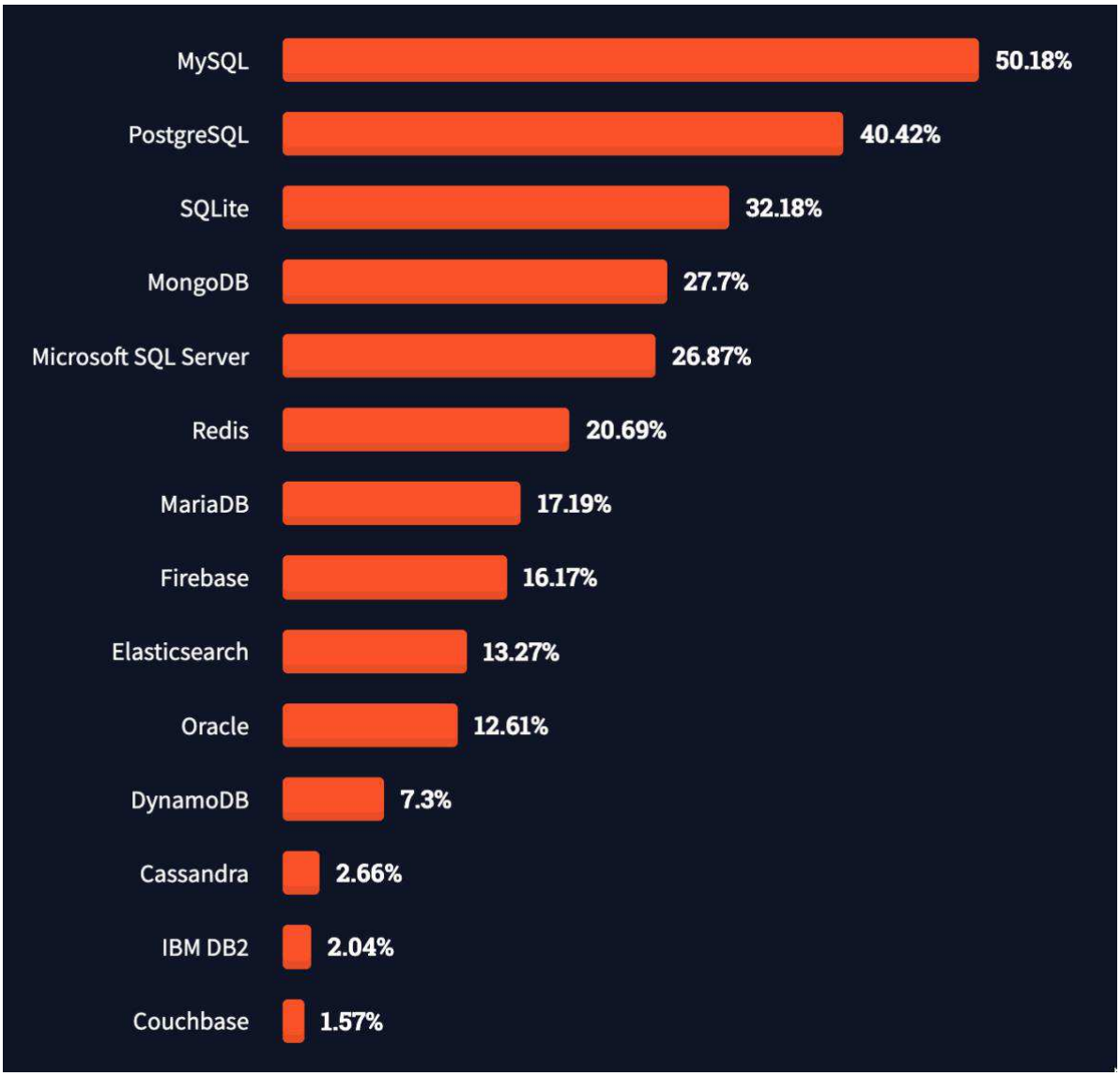


Рисунок 1.21 – Рейтинг СУБД по версии опроса *StackOverflow* 2021

Таким образом, системой управления базами данных для разрабатываемой системы была выбрана *MongoDB* как наиболее подходящая

для хранимых в ней данных.

1.7.2 Сравнительный анализ и выбор средств разработки клиентского приложения

Планируется разработка динамического веб-приложения, поэтому правильным решением будет выбрать разработку по принципу *Single Page Application* (одностраничное приложение). Для этих целей существуют три популярных и конкурирующих между собой решения: *React, Angular* и *Vue*.

Angular не рассматривается в качестве кандидата из-за того, что он имеет гораздо более высокий порог вхождения. Также он в своей основе построен на технологиях, которые не имели на момент создания утвержденной спецификации в языке. Сейчас спецификация в процессе утверждения, и она значительно отличается от того, что используется сейчас в *Angular*. Соответственно, в ближайшее время его ждут изменения и на данный момент не понятно как его разработчики будут решать эти несовместимости. Поэтому, в положении неопределенности, правильным решением будет не рассматривать *Angular* для выбора.

Vue – это фреймворк, который имеет собственную экосистему и предоставляет все необходимые инструменты для разработки современного веб-приложения «из коробки». Но на данный момент во *Vue* происходит переходный этап на новую версию, которая имеет несовместимости с прошлой. Если выбирать *Vue*, то есть дилемма: выбирать прошлую версию, но стабильную, где все из инструментов работают как нужно, но в итоге создать проблемы по переходу на новую версию в будущем; либо выбрать новую версию и мириться с тем, что не вся экосистема ещё полностью готова к новой версии и могут встречаться недоработки, ошибки и так далее. При таких вводных, выбирать *Vue* на данном его этапе развития не является хорошим решением.

Таким образом, в качестве технологии для разработки клиентского приложения был выбран *React*. *React* – это декларативная, эффективная и гибкая библиотека для создания пользовательских интерфейсов. Она позволяет создавать сложные интерфейсы из небольших и изолированных частей кода, называемых «компонентами» [7]. На данном этапе развития является самым популярным инструментов в своем роде (если судить по количеству скачиваний) и наиболее зрелым решением.

1.7.3 Сравнительный анализ и выбор средств разработки серверного приложения

Перед выбором языков для разработки серверного приложения, важно понимать специфику разрабатываемого решения. Так как это продукт для внутреннего использования, было бы неплохо свести затраты на его эксплуатацию к минимуму, ведь непосредственной прибыли в денежном эквиваленте он не приносит. Поэтому важно выбрать такой язык, программы на котором бы требовали для своей работы минимальных ресурсов, но при этом, чтобы это был высокоуровневый язык, чтобы разработка и поддержка не была излишне трудоемкой. При таких критериях выбора на ум сразу же приходит один кандидат – это язык программирования *Go*. Go (или *Golang*) – компилируемый язык программирования со статической типизацией созданный в *Google*. При его разработке большое внимание уделялось простоте, скорости работы и эффективности. Основное преимущество языка – его модель конкурентности, которая позволяет относительно легко создавать многозадачные приложения.

На рисунке 1.22 показано сравнение сравнить производительность *Go* с другими языками.

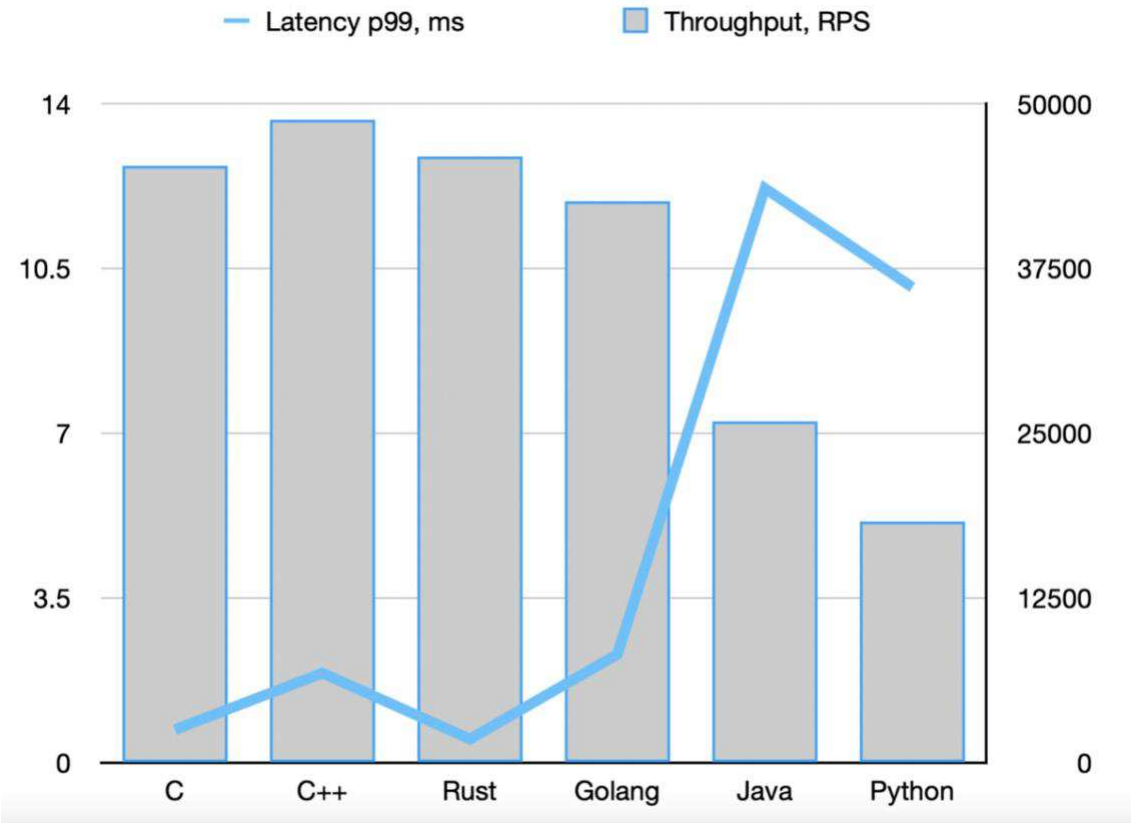


Рисунок 1.22 – Сравнение производительности языков программирования

Исходя из сравнения можно сделать вывод, что *Go* немного отстает от низкоуровневых языков программирования (*C*, *C++* и *Rust*), но при этом заметно опережает такие популярные высокоуровневые языки как *Java* и *Python*. При этом, если говорить о сложности разработки на низкоуровневых языках, то она несопоставимо выше того прироста в скорости, которые они дают по сравнению с *Go*. Таким образом, *Go* предоставляет несколько меньшую производительность, чем низкоуровневые языки программирования, но при этом является куда более простым для написания прикладных задач.

В процессе анализа средств проектных решений были выбраны следующие:

– *MongoDB* в качестве системы управления базами данных, как самая подходящая для хранения древовидной структуры данных.

– Язык *TypeScript* с библиотекой *React* для разработки клиентской части, как самое стабильное решение на данном этапе развития.

– Язык *Go* для разработки серверной части.

Вывод по разделу «Анализ предметной области»

В результате анализа предметной области было выяснено, что менеджеры выполняют важную роль в структуре компании и от их коммуникативных навыков во многом зависит насколько успешно будет выполняться работа. Для повышения этих навыков будет использоваться создаваемый диалоговый тренажер, который позволяет в смоделированных условиях отвечать на подготовленные вопросы в течение ограниченного времени, он научит формулировать шаблонные ответы на вопросы в соответствии со стандартами компании.

Был проведен анализ аналогичных решений – комплекс от компании *iSpring* и *Online Test Pad*. Оба продукта могли бы использоваться, но они имеют критичные недостатки. *iSpring* поддерживает только *Microsoft Office* и *Windows*, а *Online Test Pad* не подходит для корпоративного использования из-за отсутствия необходимых настроек ограничений доступов к создаваемым материалам. Исходя из этого необходима собственная разработка.

Изучив требования, были спроектированы модели на языках функционального моделирования IDEF0 и пользовательские сценарии в нотации IDEF3. При их помощи был определен основной функционал диалогового тренажера.

Исходя из требований были выбраны средств проектных решений. Ими стали: *Go* в качестве языка разработки серверной части, язык *TypeScript* и библиотека *React* для клиентской части, *MongoDB* в качестве системы управления базами данных.

2 Описание разработки диалогового тренажера

2.1 Разработка серверной части диалогового тренажера

Для обмена данными между сервером и клиентом будет использоваться *GraphQL API*. *GraphQL* — это язык запросов для API, а также среда выполнения для запросов с данными. [8]. Первый шаг в реализации *GraphQL API* – это описание схемы данных. Данная схема статически типизирована и является контрактом между клиентом и сервером. Сервер обязуется реализовать все, что заявлено в схеме, а клиент, при запросе данных, может быть уверен, что результат будет в точно таком же формате, который был заявлен. Для начала необходимо объявить типы данных предметной области (рис. 2.1).

```
type TrainingCategory {
  id: ObjectId!
  createdAt: DateTime!
  updatedAt: DateTime!
  title: String!
  description: String!
  trainings: [Training!]! @goField(forceResolver: true)
}

type Training {
  id: ObjectId!
  createdAt: DateTime!
  updatedAt: DateTime!
  categoryId: ObjectId!
  category: TrainingCategory! @goField(forceResolver: true)
  title: String!
  description: String!
  maxAttemptsCount: Int
  minutesToFinish: Int
  questions: [Question!]!
}

type Question {
  id: ObjectId!
  createdAt: DateTime!
  updatedAt: DateTime!
  text: String!
  answers: [Answer!]!
}

type Answer {
  id: ObjectId!
  createdAt: DateTime!
  updatedAt: DateTime!
  text: String!
}
```

Рисунок 2.1 – Объявление типов данных в *GraphQL* схеме

Синтаксис следующий – ключевое слово *type* для объявление типа, затем произвольное название на латинице без пробелов, в фигурных скобках перечисление полей и типов их данных через двоеточие, каждое поле с новой строки. Восклицательный знак после типа данных поля говорит о том, что оно должно обязательно присутствовать в ответе сервера, так как является обязательным. На рисунке 2.1 описаны типы данных для категории тренинга, самого тренинга, вопроса и ответа на вопрос.

Следующий шаг – описание доступных запросов. Для этого необходимо расширить встроенный тип *Query* и в нем описать название запроса, опционально входные данные и результат запроса (рис. 2.2).

```
type TrainingQuery

extend type TrainingQuery {
  "Список доступных тренингов"
  trainings: [Training!]! @goField(forceResolver: true)
  "Получение тренинга по идентификатору"
  training(id: ObjectId!): Training! @goField(forceResolver: true)
  "Список категорий тренингов"
  categories: [TrainingCategory]! @goField(forceResolver: true)
  "Получение категории по идентификатору"
  category(id: ObjectId!): TrainingCategory! @goField(forceResolver: true)
  "Список попыток прохождения тренинга"
  attempts: [TrainingAttempt!]! @goField(forceResolver: true)
  "Получение попытких прохождения тренинга по идентификатору"
  attempt(id: ObjectId!): TrainingAttempt! @goField(forceResolver: true)
}

extend type Query {
  training: TrainingQuery!
}
```

Рисунок 2.2 – Описание запросов в *GraphQL*

Последняя необходимая вещь – описание доступных мутаций. Мутация похожа на запрос, но с тем исключением, что её исполнение подразумевает изменение данных. Это может быть, к примеру, создание, редактирование,

удаление объекта или не такая явная операция, но которая, так или иначе, приводит к изменению состояния, например отправка ответа на вопрос в тренажере. На рисунке 2.3 показано описание мутаций для диалогового тренажера.

```
type TrainingMutation

extend type TrainingMutation {
  """Создание/редактирование категории тренажера"""
  putCategory(category: TrainingCategoryInput!): PutCategoryResponse! @goField(forceResolver: true)

  """Удаление категории тренажеров"""
  deleteCategory(categoryId: ObjectId!): DeleteCategoryResponse! @goField(forceResolver: true)

  """Создание/редактирование тренинга"""
  putTraining(categoryId: ObjectId!, training: TrainingInput!, trainingId: ObjectId): PutTrainingResponse! @goField(forceResolver: true)

  """Удаление тренажера"""
  deleteTraining(trainingId: ObjectId!): DeleteTrainingResponse! @goField(forceResolver: true)

  """Создание попытки выполнения тренажера"""
  startAttempt(trainingId: ObjectId!): StartAttemptResponse! @goField(forceResolver: true)

  """
  Отправка ответа `answerId` на вопрос `questionId` попытки `attemptId`.
  В `SubmitAnswerOkResult` возвращается следующий вопрос тренинга в рамках попытки
  """
  submitAnswer(attemptId: ObjectId!, questionId: ObjectId!, answerId: ObjectId!): SubmitAnswerResponse! @goField(forceResolver: true)
}
```

Рисунок 2.3 – Описание мутаций в *GraphQL*

Теперь, после того как схема готова, можно переходить к непосредственной её реализации.

В языке *Go* для реализации *GraphQL API* существует популярная библиотека *gqlgen*. Она использует популярный подход, который хорошо прижился в экосистеме *Go* – кодогенерация. Исходя из *GraphQL* схемы библиотека *gqlgen* генерирует типы данных в языке программирования, базовый функционал проверок данных, получения запросов и отправки ответов на них и прочие утилитарные вещи. В общем, делает всю «грязную» работу, программисту остается только реализовать функционал непосредственно самого приложения – получения данных для запросов, выполнения мутаций и

так далее. Для этого *gqlgen* также генерирует специальные функции-заглушки, в которых во входных параметрах уже находятся нужные типы данных, остается только реализовать заложенную предметной области логику и вернуть требуемый ответ (рис. 2.4).

```
func (r *trainingQueryResolver) Trainings(ctx context.Context, obj *model.TrainingQuery, filters *trainings.TrainingFilters) (model.TrainingsResponse, error) {
    panic( v: "Not implemented.")
}

func (r *trainingQueryResolver) Training(ctx context.Context, obj *model.TrainingQuery, id mongodb.ObjectId) (model.TrainingResponse, error) {
    panic( v: "Not implemented.")
}
```

Рисунок 2.4 – Функции, сгенерированные *gqlgen*

Теперь, когда заготовка готова, можно переходить к её наполнению. Основа любой информационной системы – это данные. Поэтому, следующий этап – реализация слоя работы с базой данных. Первым этапом создается интерфейс репозитория данных (рис. 2.5). Интерфейс – это контракт, в котором декларируется ряд методов. У интерфейсов может быть несколько реализаций, которые будут взаимозаменяемыми. В дальнейшем это пригодится во время написания тестов.

```
type Repository interface {
    db.Transaction

    // GetTraining получение тренажера по фильтрам
    GetTraining(ctx context.Context, filters TrainingFilters) (*Training, error)
    // GetTrainings получение списка тренажеров
    GetTrainings(ctx context.Context, filters TrainingFilters) ([]*Training, error)

    // GetCategory получение категории тренажеров по фильтрам
    GetCategory(ctx context.Context, filters TrainingCategoryFilters) (*TrainingCategory, error)
    // GetCategories получение списка категорий тренажеров по фильтрам
    GetCategories(ctx context.Context, filters TrainingCategoryFilters) ([]*TrainingCategory, error)
    // PutCategory создание/редактирование категории тренажеров
    PutCategory(ctx context.Context, category *TrainingCategory) (*TrainingCategory, error)
    // DeleteCategory удаление категории тренажеров
    DeleteCategory(ctx context.Context, categoryId db.ObjectId) error
```

Рисунок 2.5 – Интерфейс репозитория тренингов, лист 1

Рисунок 2.5, лист 2

Основная реализация будет с использованием базы данных *MongoDB*. Часть реализации приведена на рисунке 2.6.

```
type MongoRepo struct {
    *db.MongoTransaction

    trainingCollection         *mongo.Collection
    trainingCrud               *mongodb.CollectionCrud
    categoryCollection         *mongo.Collection
    categoryCrud               *mongodb.CollectionCrud
    trainingAttemptsCollection *mongo.Collection
    trainingAttemptsCrud       *mongodb.CollectionCrud

    logger logging.ContextLogger
}

func (r *MongoRepo) GetTraining(ctx context.Context, filters TrainingFilters) (*Training, error) {
    training := &Training{}
    if err := r.trainingCrud.FindOne(ctx, filters.GenerateFilters(), training); err != nil {...}

    return training, nil
}

func (r *MongoRepo) GetTrainingsByIds(ctx context.Context, ids ...db.ObjectId) ([]*Training, error) {
    var trainings []*Training
    if err := r.trainingCrud.FindManyByIds(ctx, ids, &trainings); err != nil : nil, err ↗

    return trainings, nil
}
```

Рисунок 2.6 – Реализация интерфейса репозитория для *MongoDB* в качестве базы данных

Далее необходимо реализовать сервисный слой. Он определяет для приложения границу и набор допустимых операций с точки зрения взаимодействующих с ним клиентских приложений. Он инкапсулирует бизнес-логику приложения, управляя транзакциями и управляя ответами в реализации этих операций [9]. В общем, он использует слой данных и другие модули для реализации логики приложения. На рисунке 2.7 представлена часть сервисного слоя для тренингов.

```
type Service struct {
    Repo   Repository
    Logger logging.ContextLogger
}

func (s *Service) StartAttempt(ctx context.Context, accountId, trainingId db.ObjectId) (*TrainingAttempt, error) {
    training, err := s.Repo.GetTraining(ctx, TrainingFilters{Id: &trainingId})
    if err != nil : nil, errors.WithStack(err) ⤴

    if len(training.Questions) == 0 : nil, ErrTrainingNotFound ⤴

    attemptsCount, err := s.Repo.GetTrainingAttemptCount(ctx, TrainingAttemptFilters{
        TrainingId: &trainingId,
        AccountId:  &accountId,
    })
    if err != nil : nil, errors.WithStack(err) ⤴

    if attemptsCount >= pointers.IntValue(training.MaxAttemptsCount) : nil, ErrMaxAttemptsReached ⤴

    attempt := training.ToNewAttempt(accountId)
    if attempt, err = s.Repo.CreateAttempt(ctx, attempt); err != nil : nil, errors.WithStack(err) ⤴

    return attempt, nil
}
```

Рисунок 2.7 – Сервисный слой для модуля тренингов

После написания основной логики приложения, важно понять, что она действительно работает так, как задумано. Проверить это призвано автоматическое тестирование. В данном случае будет использоваться юнит-тестирование сервисного слоя, в котором сосредоточена основа. Здесь пригодится созданный ранее интерфейс репозитория с данными. На его основе при помощи библиотеки *mock* создается мок-объект с репозиторием. Мок-

объекты (или заглушки, объекты-пустышки) – это объекты, которые имитируют поведение реальных объектов приложения, но в особенном, управляемом состоянии [10]. Таким образом, он не зависит от реальной базы данных, и при помощи библиотеки можно легко изменять поведение этого объекта, что дает гибкость и возможность легко протестировать различные сценарии.

На рисунке 2.8 изображен код структуры тестового набора. В методе *BeforeTest*, который выполняется перед запуском каждого теста, в переменную *s.repo* присваивается мок-объект репозитория. Это возможно благодаря тому, что сервис ожидает интерфейс и ему не важна конкретная реализация, поэтому он будет работать как с реализацией *MongoDB*, так и с реализацией мок-объекта. Главное то, что они реализуют один интерфейс и для сервиса являются взаимозаменяемыми.

```go
type ServiceTestSuite struct {
    suite.Suite
    service *Service
    repo    *MockRepository
    ctrl    *gomock.Controller
}

func (s *ServiceTestSuite) BeforeTest(suiteName, testName string) {
    var err error

    s.ctrl = gomock.NewController(s.T())
    s.repo = NewMockRepository(s.ctrl)
    s.service, err = NewService(logging.NewDummyLogger(), s.repo)
    require.NoError(s.T(), err)
}
```

Рисунок 2.8 – Структура тестового набора

На рисунке 2.9 изображен пример теста. Данный тест проверяет получение списка категорий тренингов и разбит на два подтеста:

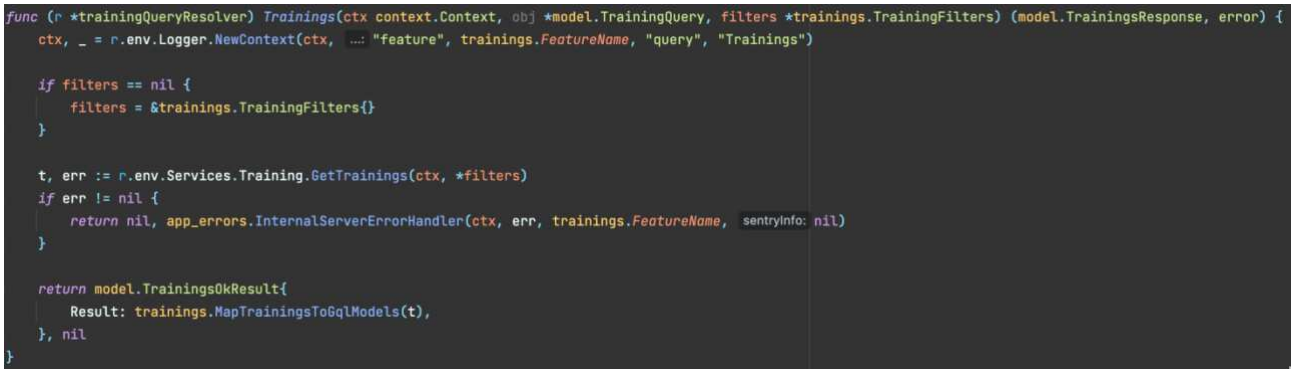
– в первом проверяется, что при передаче существующего идентификатора категории функция возвращает ожидаемый список категорий;

– во втором, что при передаче несуществующего идентификатора вернется ошибка о том, что такой категории нет.

```
func (s *ServiceTestSuite) TestGetCategoryTrainings() {
    s.Run( name: "Success", func() {
        trainings := []*Training{training}
        s.repo.EXPECT().GetTrainings(gomock.Any(), TrainingFilters{CategoryId: &category.ID}).
            Return(trainings, nil)

        receivedTrainings, err := s.service.GetTrainings(
            context.Background(),
            TrainingFilters{CategoryId: &category.ID},
        )
        require.NoError(s.T(), err)
        require.True(s.T(), len(receivedTrainings) == len(trainings))
    })
    s.Run( name: "category not found", func() {
        s.repo.EXPECT().GetTrainings(gomock.Any(), TrainingFilters{CategoryId: &training.ID}).
            Return( rets...: nil, ErrCategoryNotFound)

        receivedTrainings, err := s.service.GetTrainings(
            context.Background(),
            TrainingFilters{CategoryId: &training.ID},
        )
        require.Error(s.T(), err)
        require.True(s.T(), errors.Is(err, ErrCategoryNotFound))
        require.True(s.T(), errors.Is(err, app_errors.ErrNotFound))
        require.True(s.T(), app_errors.IsAppError(err))
        require.Nil(s.T(), receivedTrainings)
    })
}
```

Рисунок 2.9 – Модульный тест получения категорий тренингов

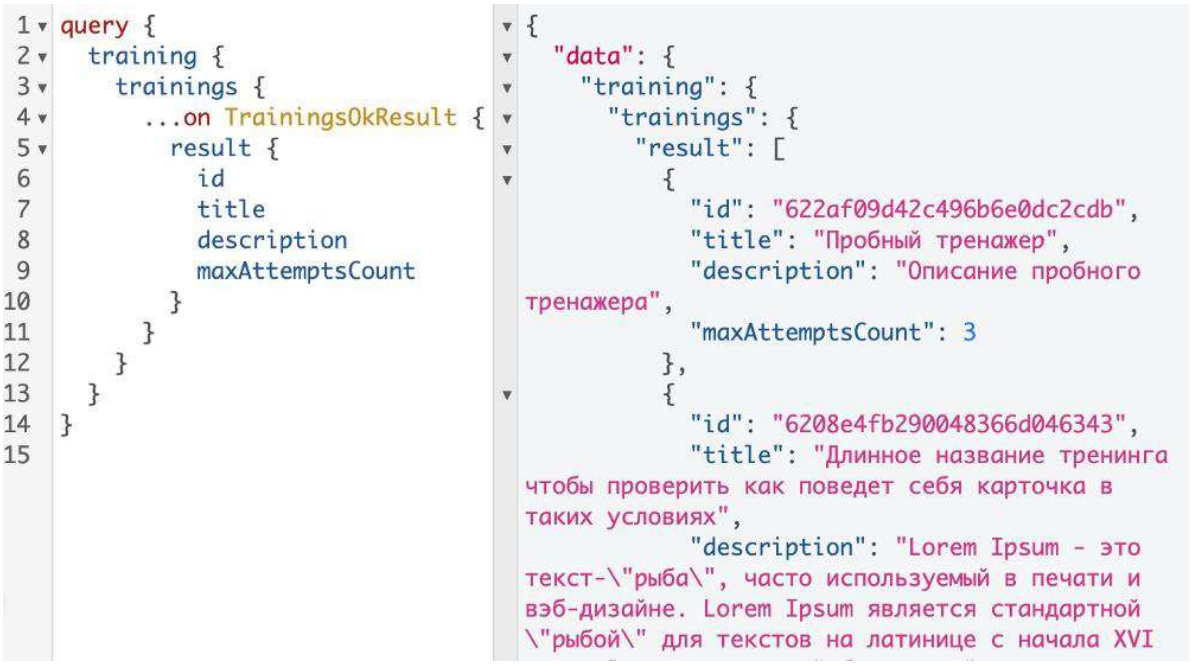
Последний этап – реализация функций «заглушек» для *GraphQL*. Все что нужно – передать входные данные нужному сервису и отправить полученный результат обратно клиенту, при этом обрабатывать возникающие ошибки (рис. 2.10).



```
func (r *trainingQueryResolver) Trainings(ctx context.Context, obj *model.TrainingQuery, filters *trainings.TrainingFilters) (model.TrainingsResponse, error) {
    ctx, _ = r.env.Logger.NewContext(ctx, ... "feature", trainings.FeatureName, "query", "Trainings")

    if filters == nil {
        filters = &trainings.TrainingFilters{}
    }

    t, err := r.env.Services.Training.GetTrainings(ctx, *filters)
    if err != nil {
        return nil, app_errors.InternalServerErrorHandler(ctx, err, trainings.FeatureName, sentryInfo: nil)
    }

    return model.TrainingsOkResult{
        Result: trainings.MapTrainingsToGqlModels(t),
    }, nil
}
```

Рисунок 2.10 – Реализация *GraphQL* запроса списка тренингов

На рисунке 2.11 показан результат работы *GraphQL* запроса списка тренажеров.



```
query {
  training {
    trainings {
      ...on TrainingsOkResult {
        result {
          id
          title
          description
          maxAttemptsCount
        }
      }
    }
  }
}
```

```
{
  "data": {
    "training": {
      "trainings": {
        "result": [
          {
            "id": "622af09d42c496b6e0dc2cdb",
            "title": "Пробный тренажер",
            "description": "Описание пробного
тренажера",
            "maxAttemptsCount": 3
          },
          {
            "id": "6208e4fb290048366d046343",
            "title": "Длинное название тренинга
чтобы проверить как поведет себя карточка в
таких условиях",
            "description": "Lorem Ipsum - это
текст-\"рыба\", часто используемый в печати и
вэб-дизайне. Lorem Ipsum является стандартной
\"рыбой\" для текстов на латинице с начала XVI
```

Рисунок 2.11 – Результат *GraphQL* запроса списка тренажеров

Отдельно стоит рассмотреть генерацию отчета по результатам прохождения тренажера. Так как это обучающий проект, важно понимать, как

справляются с тренажерами студенты. На данный момент в качестве показателя используется значение настроения собеседника. Это некий абстрактный параметр, который задается от 0 до 100, где 100 – это собеседник полностью удовлетворён теми ответами, который выбирает обучаемый, а 0 – полностью недоволен. Этот параметр задается тренером при создании наполнения тренажера. То есть, когда студент доходит до конца тренажера, он подходит с каким-то значением настроения собеседника – это и является результатом прохождения. Отчет представляет из себя *Excel*-таблицу в которой содержатся результаты пользователя по всем пройденным тренажерам.

Для начала нужно запросить необходимые данные, а именно все попытки прохождения тренажеров. В *MongoDB* для сложных запросов используется механизм *aggregation pipeline*. С помощью него можно задать список операций, которые база данных должна совершить над данными, в итоге можно получить необходимую структуру данных. На рисунке 2.12 показан запрос необходимых данных для составления отчета.

```
pipeline := []bson.M{
    {"$match": bson.M{"accountId": userId}},
    {
        "$lookup": bson.M{
            "from":         "trainings",
            "localField":   "trainingId",
            "foreignField": "_id",
            "as":           "training",
        },
    },
    {"$unwind": "$training"},
    {"$addFields": bson.M{"trainingTitle": "$training.title"}},
    {
        "$lookup": bson.M{
            "from":         "trainingCategories",
            "localField":   "training.categoryId",
            "foreignField": "_id",
            "as":           "category",
        },
    },
```

Рисунок 2.12 – Запрос данных для составления отчета, лист 1

```
    {"$unwind": "$category"},
    {"$addFields": bson.M{"categoryTitle": "$category.title"}},
    {
        "$addFields": bson.M{
            "attemptAverageMood": bson.M{
                "$avg": bson.A{"$questions.mood"},
            },
            "attemptMood": bson.M{
                "$last": "$questions.mood",
            },
        },
    },
    {
        "$sort": bson.M{
            "training.categoryId": 1,
            "training.trainingId": 1,
            "createdAt":           1,
        },
    },
}
```

Рисунок 2.12, лист 2

На первом этапе из всех попыток прохождения тренажеров оставляются только попытки запрашиваемого пользователя. Затем при помощи *$lookup* (аналог *JOIN* из мира *SQL*) присоединяются коллекции тренажеров и их категорий. К каждому документу попытки добавляются поля с названием тренажера и его категории, значение настроения собеседника на конец прохождения и средний показатель настроения за все прохождение. Последний этап – сортировка, чтобы в итоговой таблице попытки прохождения было проще группировать по категории и тренажеру.

Затем при помощи библиотеки *excelize* необходимо преобразовать полученные данные из *MongoDB* в *Excel*-таблицу. Из интересных особенностей *excelize* позволяет управлять условным форматированием, которое доступно в *Excel* (рис. 2.13). Благодаря этому итоговая таблица получилась намного более визуально информативной за счет цветовых акцентов на результатах прохождения.

```
resultRange := fmt.Sprintf( format: "F5:G%d", row-1)
f.SetConditionalFormat(sheet, resultRange, fmt.Sprintf( format: `[
    {
        "type": "cell",
        "criteria": ">=",
        "format": %d,
        "value": "66.6"
    }]`, goodFormat))
f.SetConditionalFormat(sheet, resultRange, fmt.Sprintf( format: `[
    {
        "type": "cell",
        "criteria": "between",
        "format": %d,
        "minimum": "33.3",
        "maximum": "66.5"
    }]`, okFormat))
f.SetConditionalFormat(sheet, resultRange, fmt.Sprintf( format: `[
    {
        "type": "cell",
        "criteria": "<=",
        "format": %d,
        "value": "33.2"
    }]`, badFormat))
```

Рисунок 2.13 – Использование условного форматирования таблицы

На рисунке 2.14 показана итоговая получившаяся таблица с отчетом.

Пользователь	Тестовый студент					
Категория тренажера	Тренажер	Попытка	Дата начала прохождения	Дата завершения прохождения	Настроение собеседника	Среднее настроение собеседника
Тестовая категория	Контрольный пример	1	03.05.22 13:06	03.05.22 13:14	54	42.33333333
		2	03.05.22 13:15	03.05.22 13:24	73	
		3	03.05.22 13:25	НЕ ЗАВЕРШЕН	0	
	Тестовый тренажер	1	02.05.22 11:14	02.05.22 11:26	64	69.5
		2	03.05.22 13:06	03.05.22 13:32	75	
Онбординг менеджеров по продажам	Работа с возражерниями	1	03.05.22 13:06	В ПРОЦЕССЕ	-	
	Работа с холодными клиентами	1	03.05.22 13:06	03.05.22 13:06	75	80
		2	03.05.22 13:06	03.05.22 13:06	85	

Рисунок 2.14 – Таблица с отчетом по пользователю

Таким образом разработка серверной части диалогового тренажера выполнена.

2.2 Разработка клиентской части диалогового тренажера

Вся клиентская часть будет написана на *TypeScript* (надмножество языка *JavaScript*, к которому добавили систему типов) и библиотеке *React*. В целом необходимо реализовать 6 основные страниц:

– страница категорий тренажеров;

– страница со списком тренажеров в категории;

– редактор тренажеров;

– страница с информацией о тренажере;

– страница прохождения тренажера;

– страница управления пользователями.

Как известно, веб-страница состоит из трех основных компонентов:

1) *HTML* разметка, которая задает каркас страницы;

2) *CSS* стилей, которые оформляют заданный каркас;

3) *JavaScript*, который привносит интерактивность на страницу.

Миссия библиотек и фреймворков, в том числе и *React* в том, чтобы максимально упростить работу с динамически изменяемыми данными на странице. Для этого в *React* отказались от написания *HTML* файлов и решили генерировать структуру из *JavaScript*.

Для удобства разработчиков был придуман специальный синтаксис – *JSX*. Этот синтаксис очень похож на *HTML*, но при этом он остается *JavaScript* с возможностью вставлять динамические данные. *JSX* – «синтаксический сахар» для функции *React.createElement(component, props, ...children)* [11]. Синтаксис *JSX* и то, во что он превращается под «капотом», показано на рисунке 2.15.

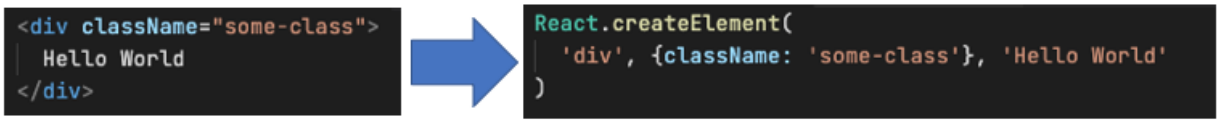


Рисунок 2.15 – Синтаксис *JSX*

React использует компонентный подход. Компоненты позволяют разбить интерфейс на независимые части, про которые легко думать в отдельности. Их можно складывать вместе и использовать несколько раз [12]. Хорошей практикой является разбиение элементов страницы на отдельные небольшие компоненты и сбор их в одном компоненте страницы. Компонент является обычной функцией, которая возвращает *JSX* разметку.

Этот подход можно рассмотреть на примере страницы списка категорий тренажеров. Задача компонента страницы – получить необходимые для отрисовки страницы данные с сервера, собрать из заранее подготовленных компонентов-блоков каркас страницы и передать в них полученные данные. Затем, эти компоненты-блоки, которые, зачастую, тоже собраны из более

простых компонентов, передают данные ещё ниже. Таких уровней вложенности может быть ещё несколько, в зависимости от сложности проекта.

Первый этап, как сказано выше, это получение данных. Для этого в проекте разработки диалогового тренажера используется *GraphQL*. Он предоставляет специальный язык запросов, благодаря которому клиент может запрашивать только необходимую информацию. На рисунке 2.16 показан запрос списка категорий тренингов.

Рисунок 2.16 – GraphQL запрос списка категорий тренингов

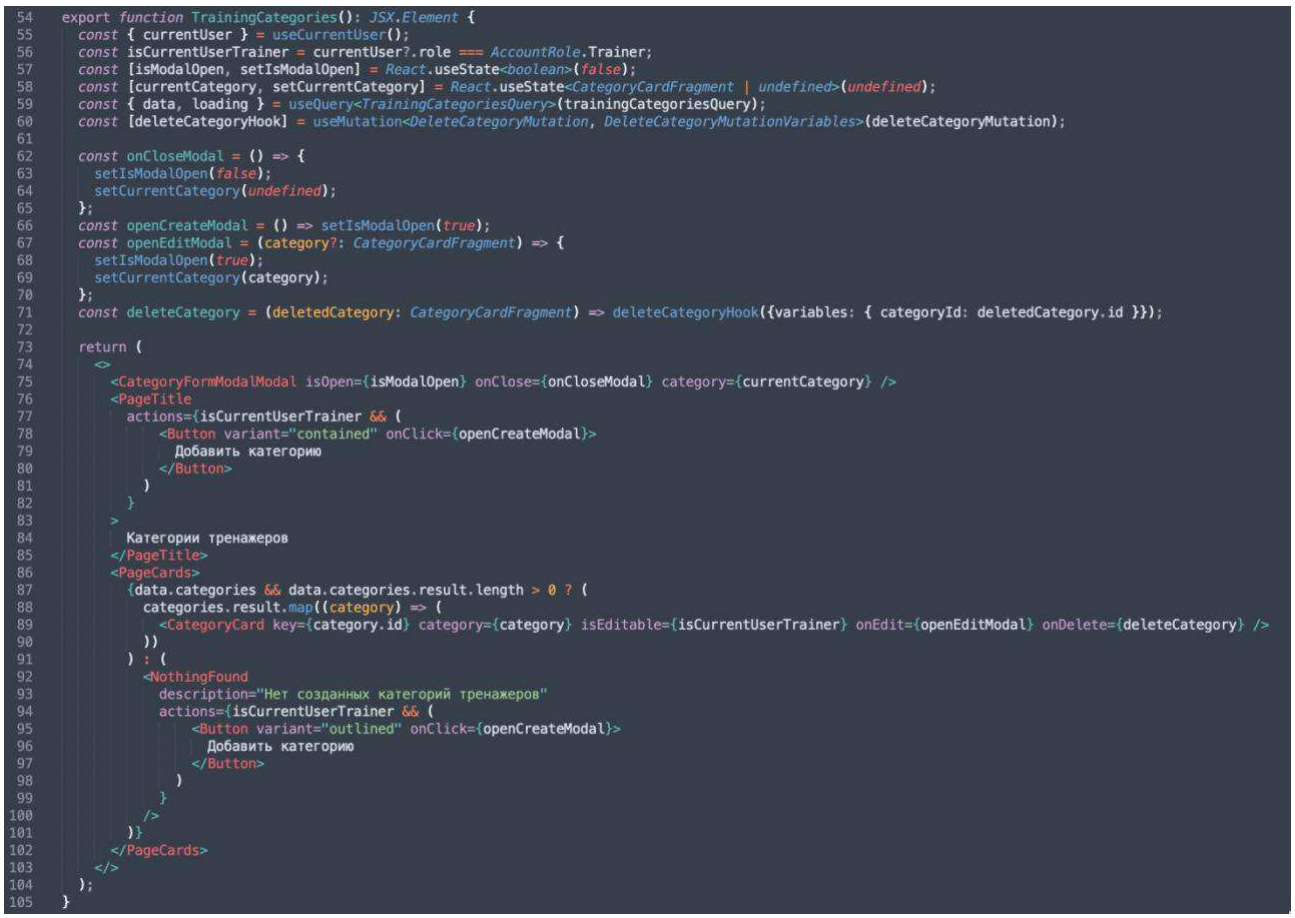
Троеточие и *CategoryCard* после него – это фрагмент – переиспользуемая структура, которая позволяют определять наборы полей, основанный на каком-

либо GraphQL-типе, и включать их в запрос. Таким образом, *CategoryCard* – это перечень полей, которые необходимо получить с сервера для отображения компонента карточки категории. Важным моментом является то, что этот фрагмент определяется вместе с компонентом карточки и, если в нем изменится набор необходимых данных, нужно будет изменить это только в одном месте – во фрагменте. При этом во всех местах, где он используется в проекте, запрос автоматически обновится. Это значительно упрощает внесение изменений в проект. На рисунке 2.17 показано определение фрагмента *CategoryCard*.

Рисунок 2.17 – Определение фрагмента *CategoryCard*

Теперь, когда запрос готов, его нужно отправить на сервер. Для этого используется библиотека *Apollo*.

Библиотека *Apollo* предоставляет набор *React*-хуков (от англ. *hook*) для коммуникации с сервером. Хуки – это функции, с помощью которых возможно «подцепиться» (от англ. *hook* – крючок) к состоянию и методам жизненного цикла *React* из функциональных компонентов [13]. В данном случае используется хук *useQuery* для отправки запроса (рис. 2.18).



```
export function TrainingCategories(): JSX.Element {
  const { currentUser } = useCurrentUser();
  const isCurrentUserTrainer = currentUser?.role === AccountRole.Trainer;
  const [isModalOpen, setIsModalOpen] = React.useState<boolean>(false);
  const [currentCategory, setCurrentCategory] = React.useState<CategoryCardFragment | undefined>(undefined);
  const { data, loading } = useQuery<TrainingCategoriesQuery>(trainingCategoriesQuery);
  const [deleteCategoryHook] = useMutation<DeleteCategoryMutation, DeleteCategoryMutationVariables>(deleteCategoryMutation);

  const onCloseModal = () => {
    setIsModalOpen(false);
    setCurrentCategory(undefined);
  };
  const openCreateModal = () => setIsModalOpen(true);
  const openEditModal = (category?: CategoryCardFragment) => {
    setIsModalOpen(true);
    setCurrentCategory(category);
  };
  const deleteCategory = (deletedCategory: CategoryCardFragment) => deleteCategoryHook({variables: { categoryId: deletedCategory.id }});

  return (
    <>
      <CategoryFormModalModal isOpen={isModalOpen} onClose={onCloseModal} category={currentCategory} />
      <PageTitle
        actions={isCurrentUserTrainer && (
            <Button variant="contained" onClick={openCreateModal}>
              Добавить категорию
            </Button>
          )
        }
      >
        Категории тренажеров
      </PageTitle>
      <PageCards>
        {data.categories && data.categories.result.length > 0 ? (
          categories.result.map((category) => (
            <CategoryCard key={category.id} category={category} isEditable={isCurrentUserTrainer} onEdit={openEditModal} onDelete={deleteCategory} />
          ))
        ) : (
          <NothingFound
            description="Нет созданных категорий тренажеров"
            actions={isCurrentUserTrainer && (
                <Button variant="outlined" onClick={openCreateModal}>
                  Добавить категорию
                </Button>
              )
            }
          />
        )}
      </PageCards>
    </>
  );
}
```

Рисунок 2.18 – Компонент страницы категорий тренажеров

В переменную *data* попадает ответ сервера, на строках 73-104 идет построение каркаса страницы из заранее подготовленных компонентов, где *PageTitle* – компонент заголовка страницы, в котором настроены все стили – шрифт, отступы, размер и т.д.; *PageCards* – контейнер для списка карточек, который ограничивает максимальную ширину и задает расстояние между карточками; *CategoryCard* компонент самой карточки категории, в которую при помощи параметров передаются данные о названии, заголовке и идентификаторе; *NothingFound* – компонент, который оповещает о том, что категорий нет, отображается только в том случае, когда сервер возвращает пустой список. На рисунке 2.19 показан итоговый результат страницы.

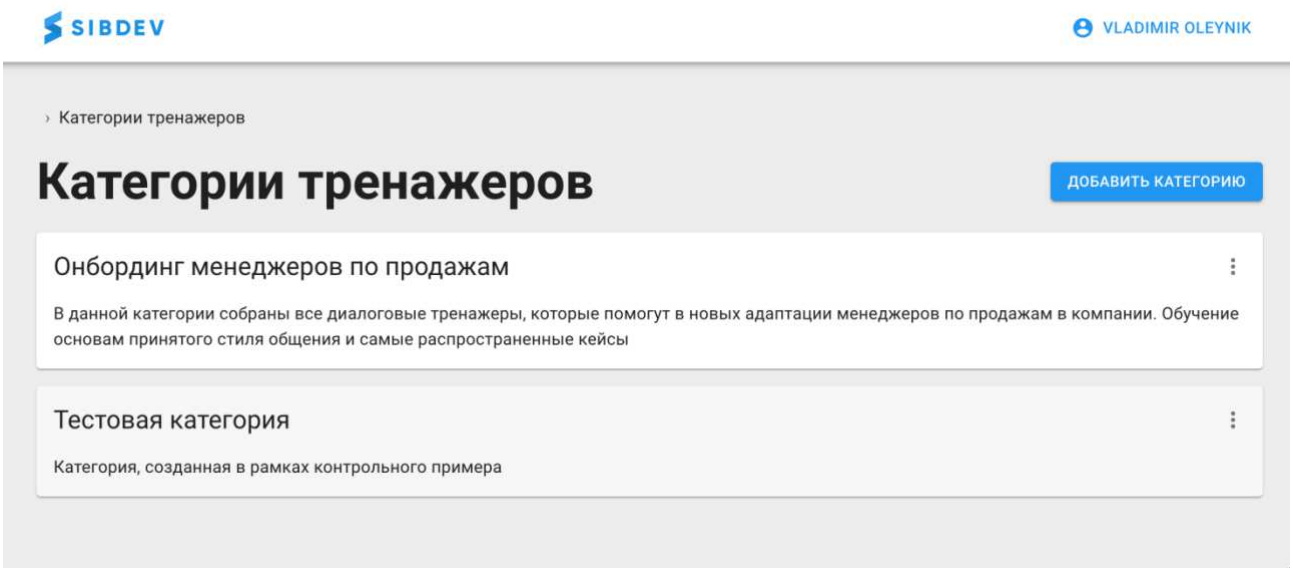


Рисунок 2.19 – Страница списка категорий тренажеров

Аналогичным образом создается страница со списком тренажеров в категории (рис. 2.20).

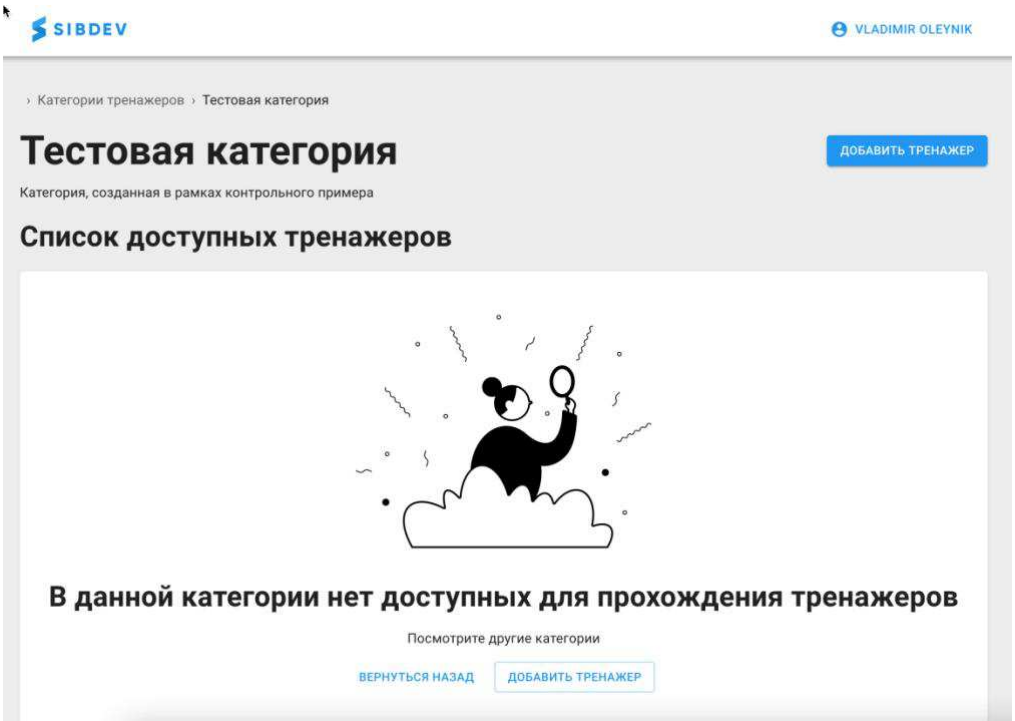


Рисунок 2.20 – Страница со списком тренажеров в категории

Следующая страница одна из важнейших – это редактор тренажеров. Редактор тренажеров позволяет создать новый или отредактировать существующий тренажер. Одна из стоящих задач – сделать эту страницу как можно удобнее для конечного пользователя – тренера, который будет заниматься наполнением. Основная сложность, которая возникла в процессе разработки – это придумать как визуализировать сценарий диалогового тренажера так, чтобы его было удобно просматривать и вносить в него изменения. Как было сказано ранее, диалоговый тренажер может иметь нелинейную структуру, то есть при выборе первого варианта ответа, следующим может быть задан один вопрос, а при выборе второго варианта – другой вопрос. На рисунке 2.21 показан получившийся вариант страницы.

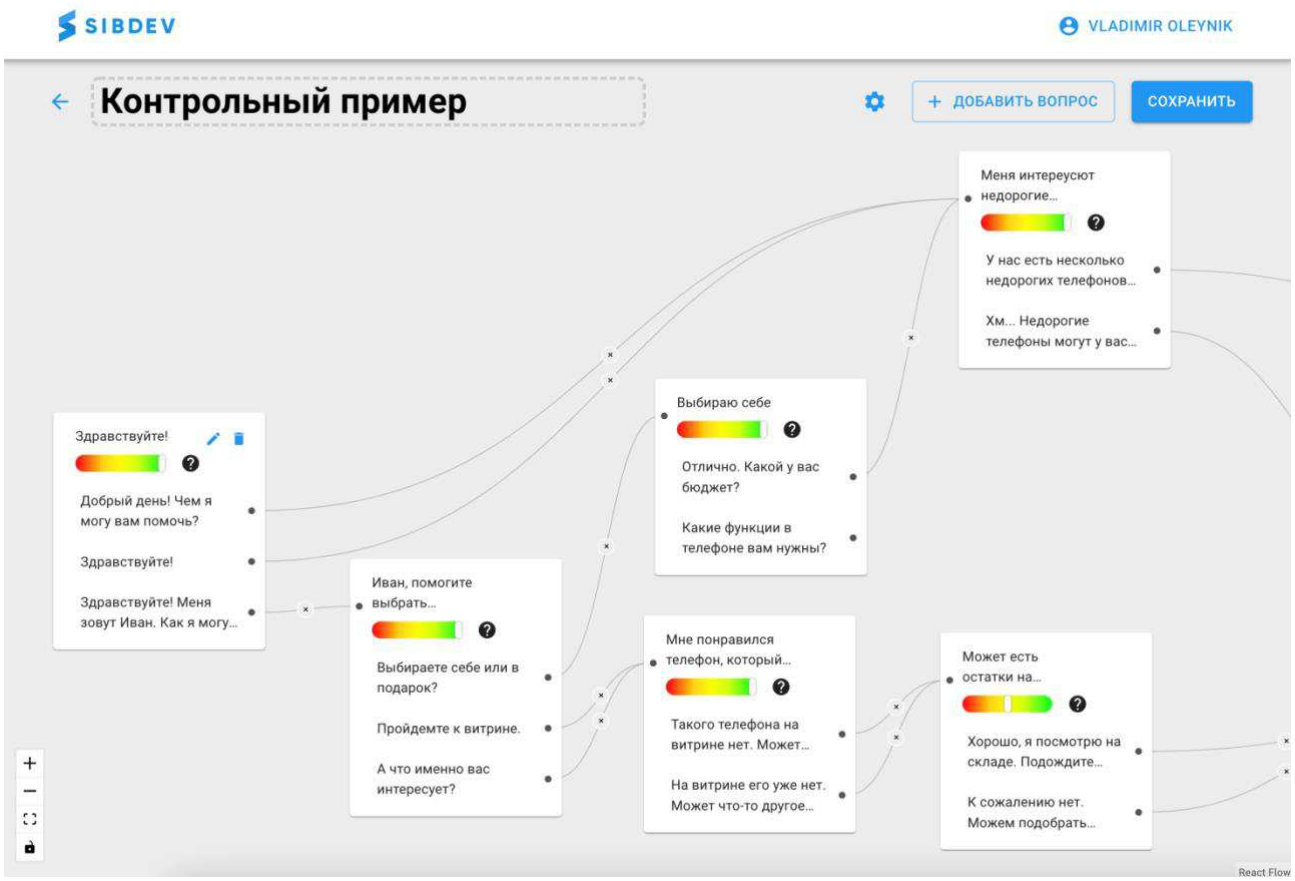


Рисунок 2.21– Страница редактирования тренажера

Концепция такая – вопрос представлен в виде карточки, в шапке которой написан текст вопроса и под ней список вариантов ответов на этот вопрос. Карточки размещаются на холсте и их можно по нему свободно перемещать. Каждый вариант ответа можно соединить с другой карточкой вопроса, таким образом и задается последовательность.

Данное дерево с карточками вопросов сделано при помощи библиотеки *React Flow*, которая позволяет строить подобные интерактивные диаграммы. При редактировании тренажеров с сервера приходит полная структура тренажера со всеми вопросами и ответам, они подгоняются под формат, который требует *React Flow*. Из этих данных при помощи *React Flow* строится дерево.

При нажатии на кнопку «Добавить вопрос» открывается модальное окно с формой создания нового вопроса (рис. 2.22).

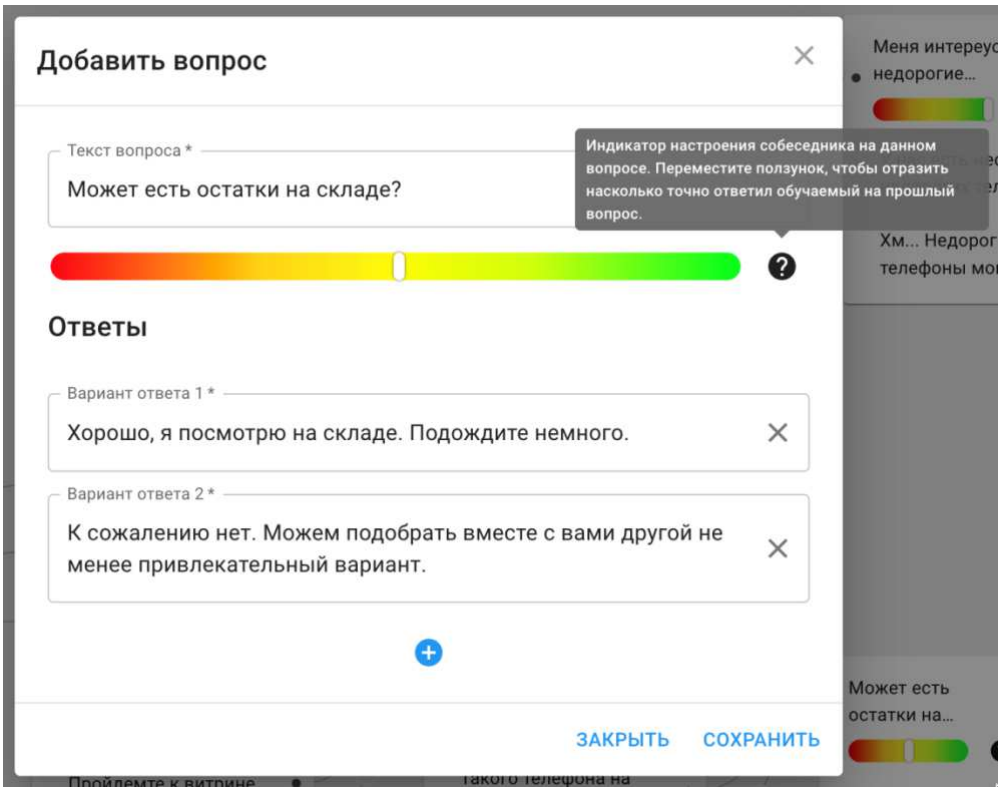


Рисунок 2.22 – Форма добавления вопроса

Необходимо заполнить текст вопроса, указать настроение собеседника и добавить ответы. Настроение собеседника помогает обучаемому получать обратную связь на свои ответы. Выполнена она в виде шкалы от 0 до 100, где 0 – красная зона, а 100 – зеленая зона. Чем ближе шкала к зеленой зоне, тем больше собеседник удовлетворён ответами обучаемого. Количество вариантов ответов на вопрос неограниченно, при помощи кнопки «+» можно добавить столько вариантов ответов, сколько необходимо. Максимальное количество вопросов в тренинге также не ограничено, минимальное количество – 2 вопроса.

На рисунке 2.23 показаны настройки тренажера, которые открываются при нажатии на «шестеренку». В них можно выбрать категорию тренажера, указать описание, максимальное количество попыток прохождения тренажера и максимальное количество минут, которое дается на каждое прохождение тренажера.

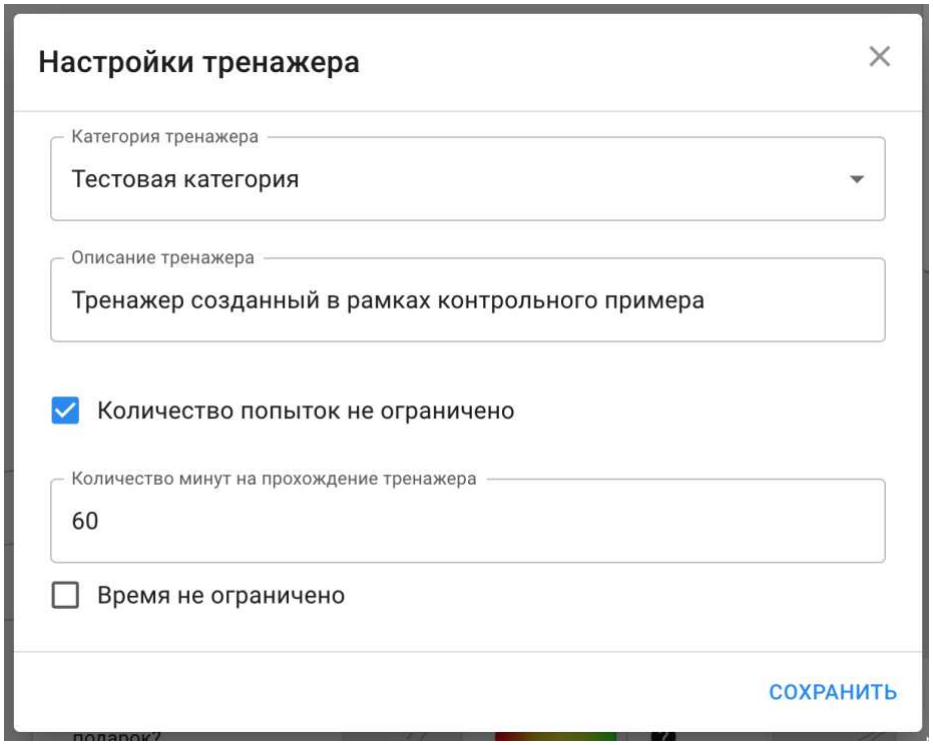


Рисунок 2.23 – Настройки тренажера

При сохранении тренажера все данные из формата *React Flow* преобразуются обратно в формат, который необходим серверу и в этом виде отправляются на него.

Следующая страница – страница тренажера. Она создается похожим образом, как и страницы со списками, но есть несколько особенностей: у тренажеров может быть ограничено число попыток выполнения, а также попытка выполнения может быть ограничена по времени. В связи с этим нужно предусмотреть 2 возможности – блокирование кнопки старта выполнения тренажера, если все попытки были израсходованы, а также вывод списка попыток с их текущим статусом (выполнен, в процессе выполнения, не выполнено вовремя).

На рисунке 2.24 компонент карточки с информацией о попытке прохождения тренажера принимает идентификатор, название, дату начала попытки, дату окончания попытки и дату, когда была завершена попытка (если тренажер выполнен раньше ограничения). Исходя из этих данных можно вывести следующее – завершена ли попытка (*isFinished)*, оставшееся время (*remainTime*), выполняется ли тренажер сейчас (*isInProgress*), закончилось ли время на выполнение тренажера (*isTimeOut*) и оставшееся время на выполнение (*timeLeft*).

```
export function AttemptCard({
  id,
  title,
  createdAt,
  expireAt,
  finishedAt,
}: AttemptCardProps): JSX.Element {
  const isFinished = !!finishedAt;
  const remainTime = expireAt && new Date(expireAt).getTime() - new Date().getTime();
  const isInProgress = !isFinished && remainTime && remainTime > 0;
  const isTimeOut = !isFinished && remainTime && remainTime <= 0;
  const [timeLeft, setTimeLeft] = React.useState<string | undefined>();
```

Рисунок 2.24 – Компонент карточки с информацией о попытке прохождения тренажера

Удобным для пользователя будет показ оставшегося времени на прохождение попытки. Для этого будет использоваться хук *useEffect*. Хук *useEffect* позволяет управлять различными сопутствующими действиями в функциональном компоненте или то, что называется «*side effects*» (побочные эффекты), например, извлечение данных, ручное изменение структуры *DOM*, использование таймеров, логгирование и т.д. [14]. В данном случае необходим таймер, который будет запускаться раз в секунду и обновлять значение оставшегося времени на выполнение. В *TypeScript* для таких действий используется функция *setInterval* (рис. 2.25).

```
React.useEffect(() => {
  if (isInProgress) {
    const interval = setInterval(() => {
      const remainTimestamp = new Date(expireAt).getTime() - new Date().getTime();
      setTimeLeft(formatMinSecDuration(remainTimestamp));
    }, 1000);

    return () => clearInterval(interval);
  }
}, []);
```

Рисунок 2.25 – Таймер оставшегося времени выполнения тренажера

Таким образом, используя все собранные данные, получается страница, изображенная на рисунке 2.26. Перед порядковым номером попытки отображается иконка, которая соответствует её текущему статусу: зеленая галочка – попытка пройдена; желтые аналоговые часы – попытка находится в процессе прохождения, красный крест – попытка не была закончена в установленный срок.

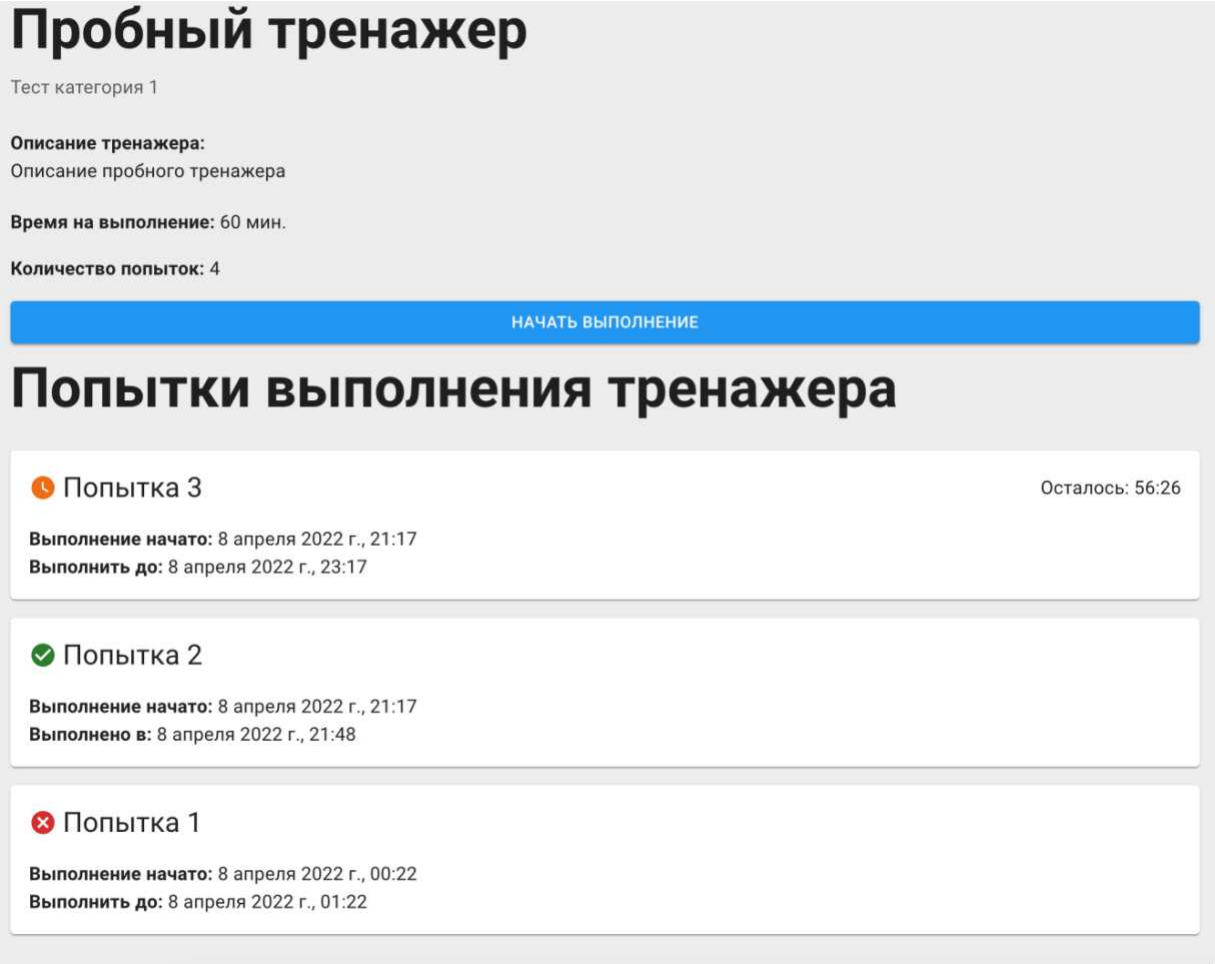


Рисунок 2.26 – Страница тренажера

Далее на очереди страница выполнения тренажера. Так как он используется в веб-студии, привычной средой для её сотрудников является мессенджеры. Многие коммуникации проходят с их помощью, поэтому, было принято решение выполнить тренажер в стиле мессенджера (рис. 2.27). В верхней панели есть 3 элемента: кнопка назад, индикатор настроения собеседника и таймер обратного отсчета, который показывает сколько времени осталось на выполнение тренажера. Индикатор настроения дает обучаемому обратную связь о том, в правильном ли направлении он движется. После ответа на вопрос, настроение собеседника может меняться – чем ближе индикатор к зеленой зоне, тем больше собеседник удовлетворён ответами. Его значения задаются тренером в редакторе тренажеров.

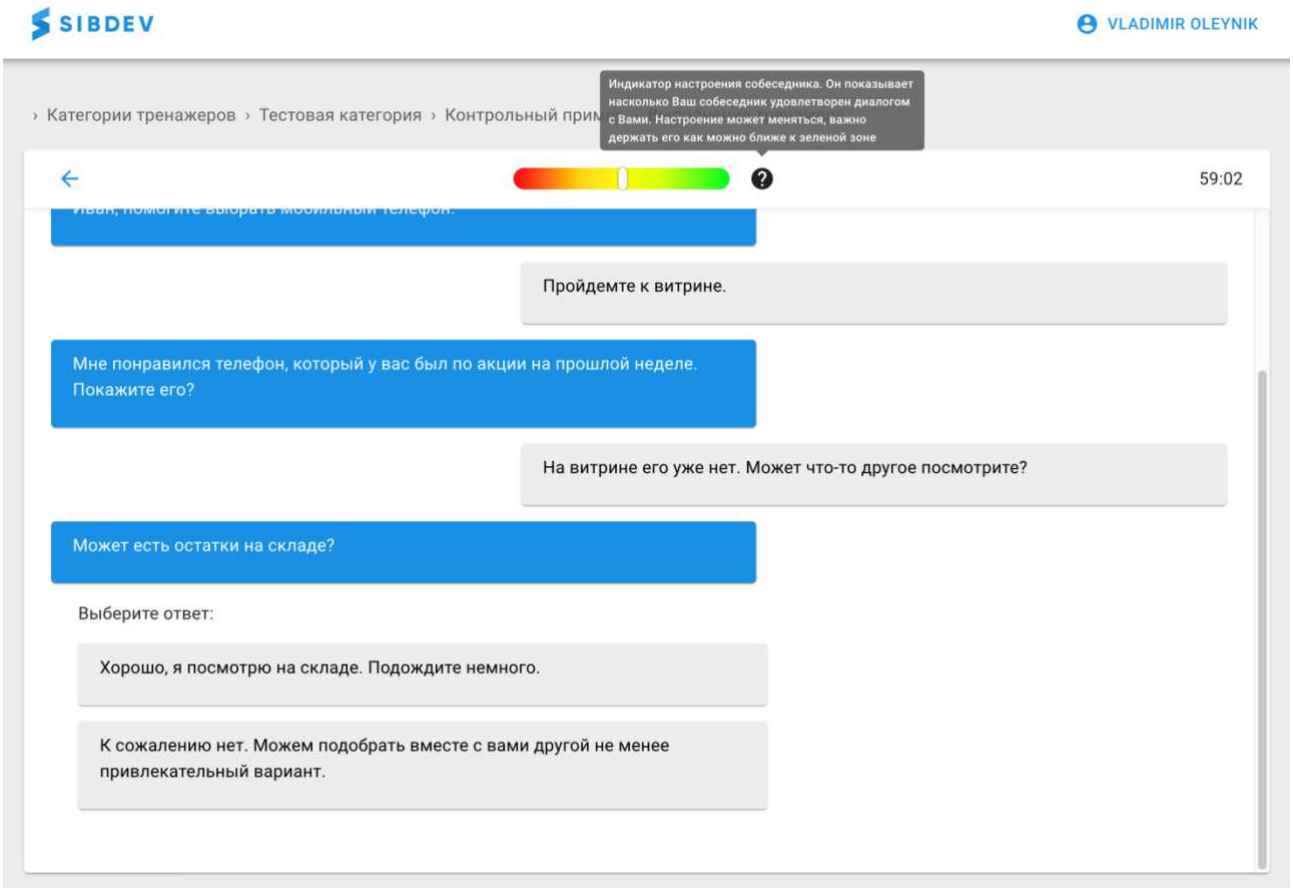


Рисунок 2.27 – Страница выполнения тренажера

Основная логика страницы – выбор студентом варианта ответа на поставленный вопрос. При нажатии на вариант ответа происходит следующее:

– информация о выбранном варианте ответ отправляется на сервер;

– сервер возвращает в ответ следующий вопрос;

– отображение следующего вопроса с новыми вариантами ответов.

Эта последовательность продолжается до тех пор, пока не закончатся вопросы.

Для отправки варианта ответа используется *GraphQL* мутация *SubmitAnswer* (рис. 2.28). Мутация может вернуть несколько ответов – следующий вопрос тренажера или перечень ошибок, например, что время на выполнение тренажера истекло. В зависимости от ответа выбирается дальнейшее поведение страницы.

Рисунок 2.28 – Мутация *SubmitAnswer*

Для отправки мутации на сервер используется хук *useMutation*. Он возвращает функцию отправки, полученные данные и флаг загрузки (рис. 2.29).

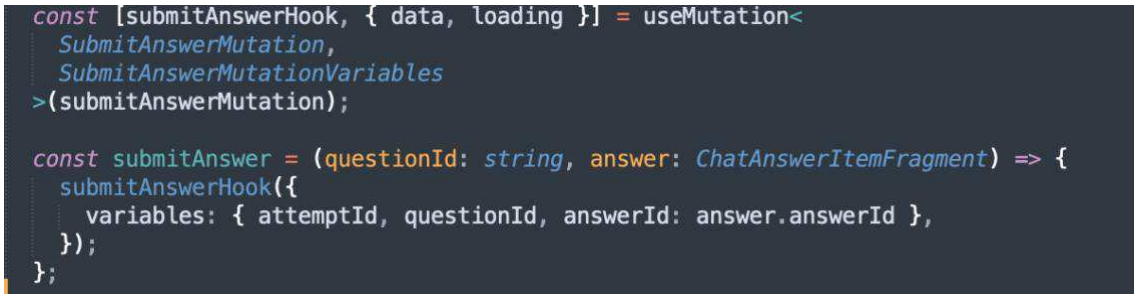


Рисунок 2.29 – Использование хука *useMutation*

Последняя страница – страница управления пользователями (рис. 2.30). Она доступна только для тренеров и позволяет создать нового пользователя или скачать отчет прохождений интересующего его пользователя.

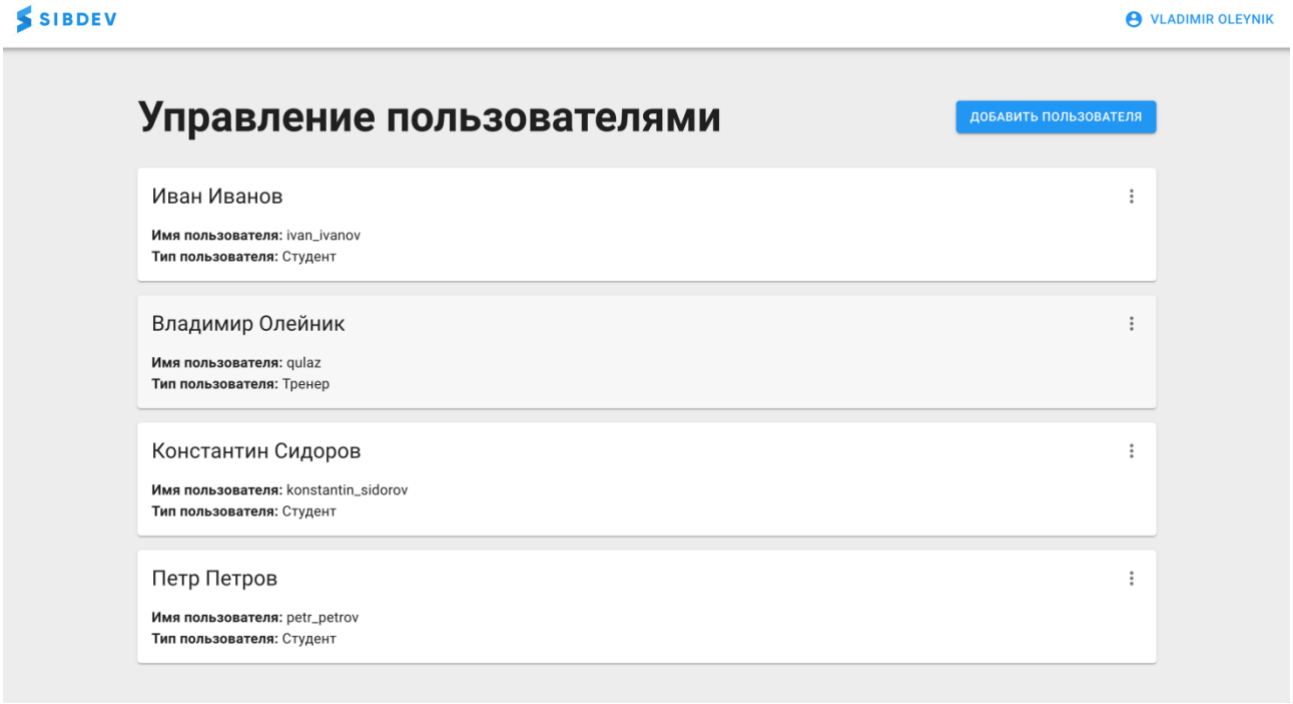


Рисунок 2.30 – Страница управления пользователями

Попасть на неё можно из меню, которое открывается при нажатии на имя текущего пользователя в правом верхнем углу (рис. 2.31).

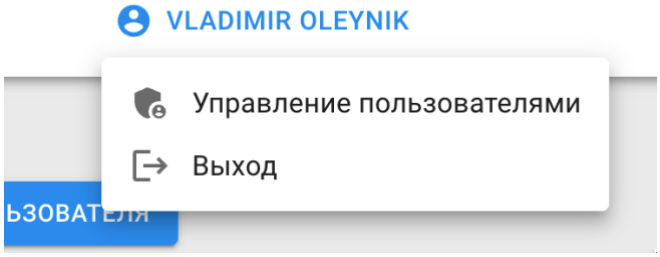


Рисунок 2.31 – Меню текущего пользователя

Форма создания пользователя показана на рисунке 2.32. На ней необходимо заполнить имя, фамилию, логин (имя пользователя, которое будет использоваться при входе) и выбрать тип пользователя: студент или тренер.

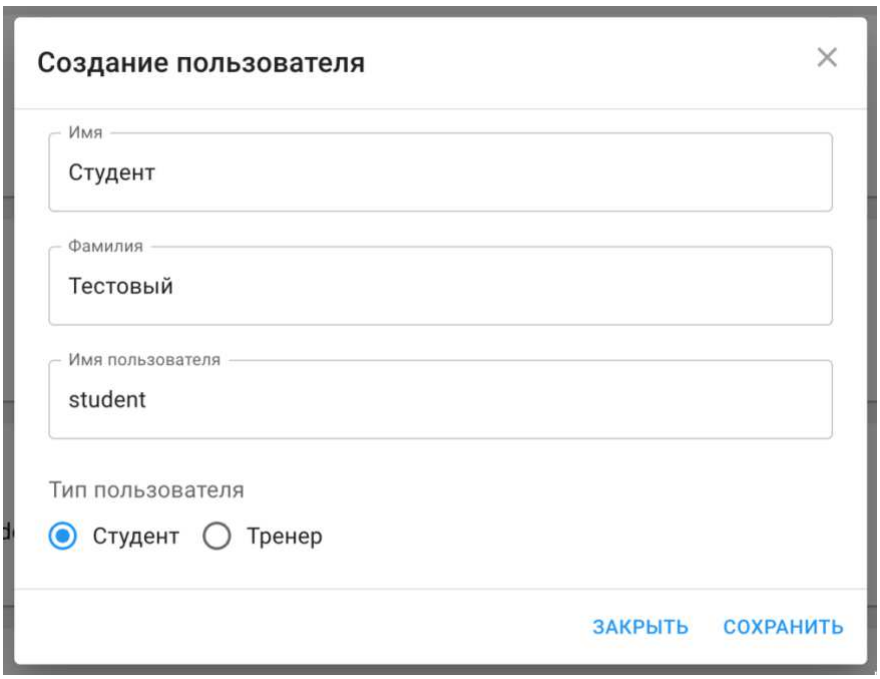


Рисунок 2.32 – Форма создания пользователя

После создания нажатия на кнопку «Сохранить» запрос отправляется на сервер и клиенту возвращаются данные о созданном пользователе, в том числе, автоматически сгенерированный пароль для него. На рисунке 2.33 показано отображение этих данных.

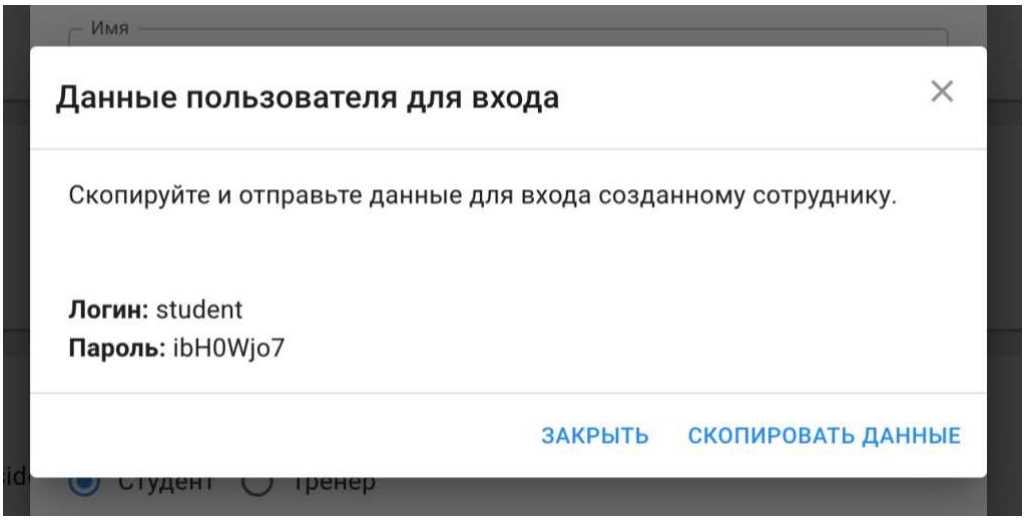


Рисунок 2.33 – Отображение данных для входа созданного пользователя

Предоставляется кнопка «Скопировать данные», которая сохранит данные для входа в буфер обмена. Их в дальнейшем можно вставить в сообщение для создаваемого сотрудника. Интересной особенностью этой кнопки является то, что она использует переменные окружения для формирования сообщения. В копируемое сообщение подставляется URL-адрес для входа, и он будет разным в зависимости от того, в каком окружении используется веб-приложение. То есть, на компьютере разработчика будет использоваться *localhost*, а на рабочем сервере уже домен, на котором работает веб-приложение. Реализация этого поведения представлена на рисунке 2.34.

```
const copyLoginData = () => {
  navigator.clipboard
    .writeText(
      data: `Данные для входа в диалоговый тренажер SibDev ${
        import.meta.env.VITE_BASE_URL
      }/login:\nЛогин: ${username};\nПароль: ${password}.`,
    )
    .then(() => close());
};
```

Рисунок 2.34 – Формирование копируемого сообщения

Последний шаг – реализация скачивания отчета о результатах студента. На рисунке 2.35 показано меню, которое открывается при нажатии на троеточие на карточке пользователя.

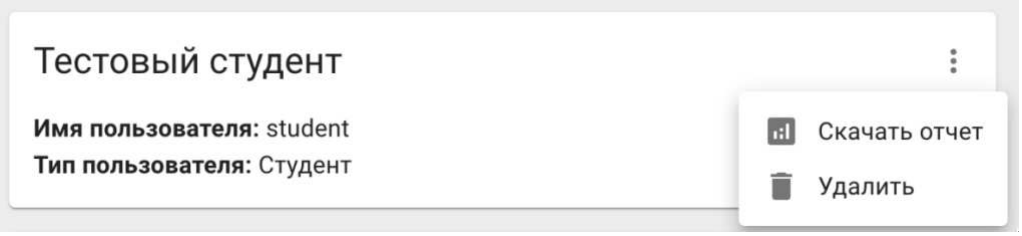


Рисунок 2.36 – Меню с дополнительным действиями над пользователем

При нажатии на кнопку «Скачать отчет» отправляется запрос на сервер. Сервер генерирует *Excel*-таблицу и отправляет её на клиент. Проблема заключается в том, что *GraphQL* не поддерживает отправку файлов. Один из вариантов решения, который отлично подходит для таких небольших файлов как таблицы – отправлять закодированную строку в кодировке *base64* и преобразовывать её в файл на стороне клиента. Для этого используется небольшая библиотека *download* (рис. 2.37).

```
const downloadReport = (userId: number) => {
  dowloadReportHook({ variables: { userId } }).then((res) => {
    download(res.base64, res.contentType, res.filename)
  })
}
```

Рисунок 2.37 – Код скачивания отчета

Таким образом, получилась готовая клиентская часть диалогового тренажера.

2.3 Тестирование диалогового тренажера на контрольном примере

Чтобы проверить, что разработанный диалоговый тренажер работает так, как было задумано, необходимо проверить его функциональность на контрольном примере. Так как в данной системе два типа пользователей с разными сценариями использования – тренер и студент, то и контрольный пример будет разделен на два этапа. Начать необходимо с тренера, так как именно этот тип пользователей является создателем контента в системе. В рамках контрольного примера тренер создаст категорию тренажеров, сам тренажер, относящийся к созданной категории и пользователя-студента. Студент же проверит, что созданный объекты действительно существуют, а

также что последовательность вопросов в тренажере совпадает с заданной тренером. Затем нужно будет опять зайти от лица тренера и скачать отчет о прохождении тренажеров студентом.

2.3.1 Тестирование функций диалогового тренажера в роли тренера

Изначально тренер попадает на пустую систему, в которой ещё не было создано ни одного тренажера. При этом система предлагает ему два варианта: вернуться назад или добавить новую категорию (рис. 2.38). В рамках примера необходимо выбрать вариант «Добавить категорию».

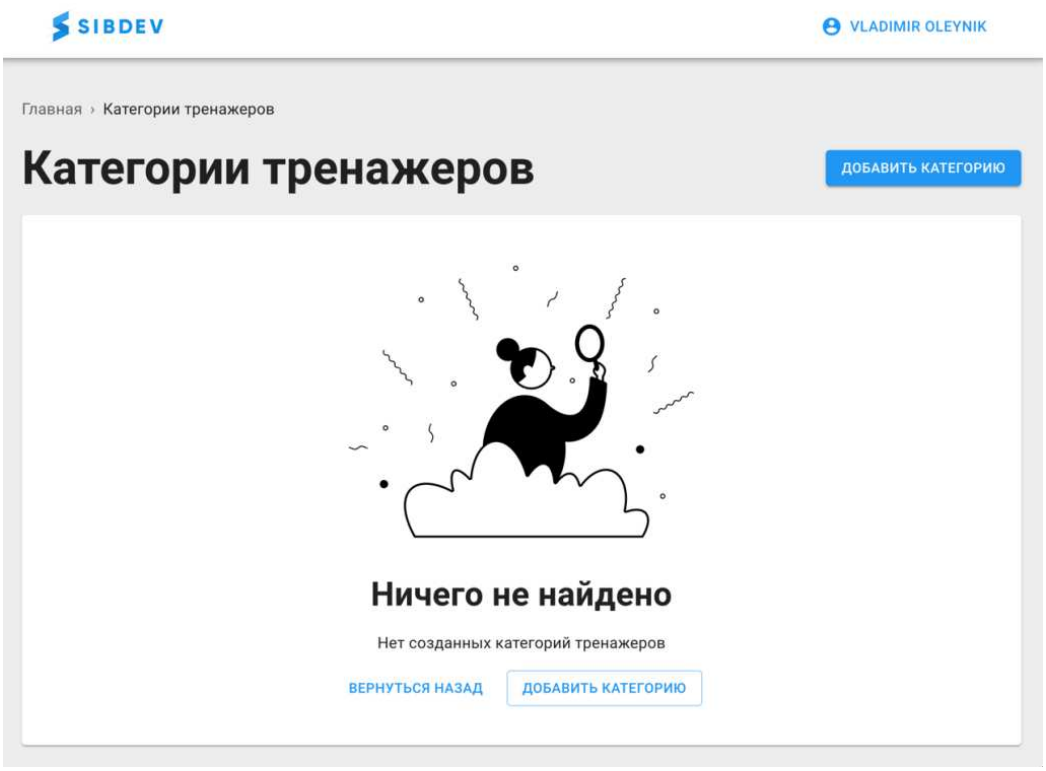


Рисунок 2.38 – Страница списка категорий

На рисунке 2.39 изображено открывшееся модальное окно с предложением ввести название и описание для будущей категории тренажеров.

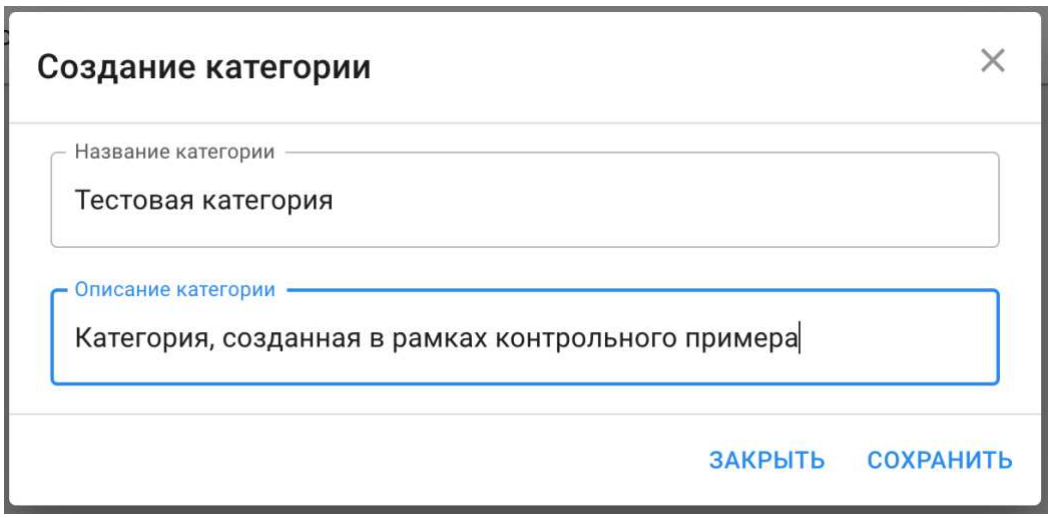


Рисунок 2.39 – Модальное окно с формой создания категории

Валидация полей формы также предусмотрена. Создать категорию, не введя данных в форму, не выйдет (рис. 2.40). Во всех остальных формах системы валидация также предусмотрена.

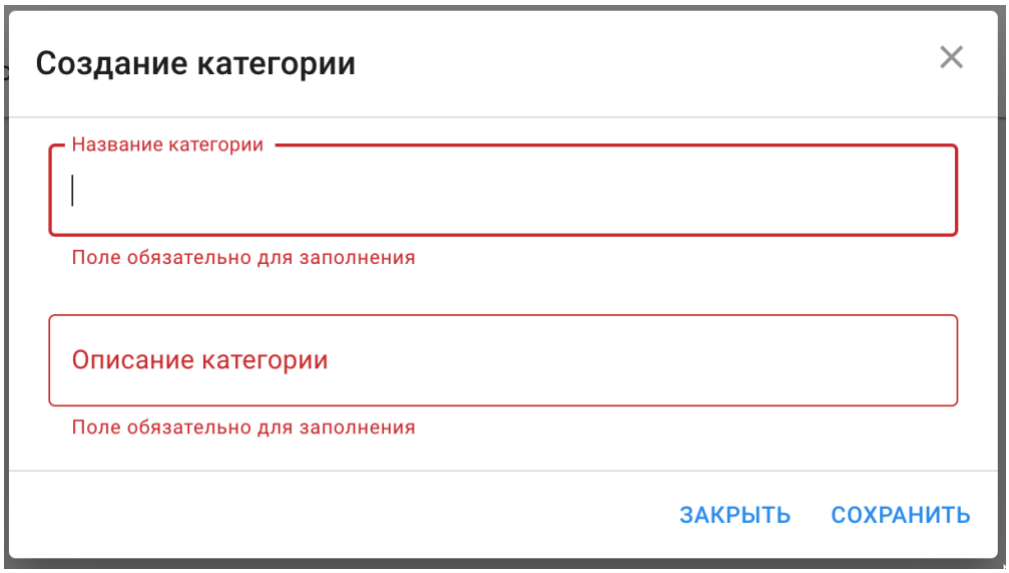


Рисунок 2.40 – Пример валидации формы создания категории

После успешного создания категория появляется в списке (рис. 2.41).

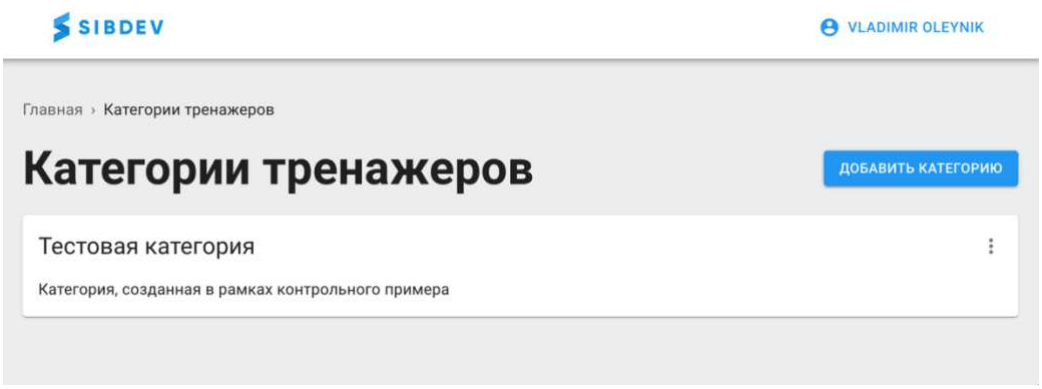


Рисунок 2.41 – Отображение созданной категории в списке

Перейдя в созданную категорию, система также предлагает два действия: вернуться назад или добавить тренажер (рис. 2.42). В данном случае выбрать необходимо «Добавить тренажер».

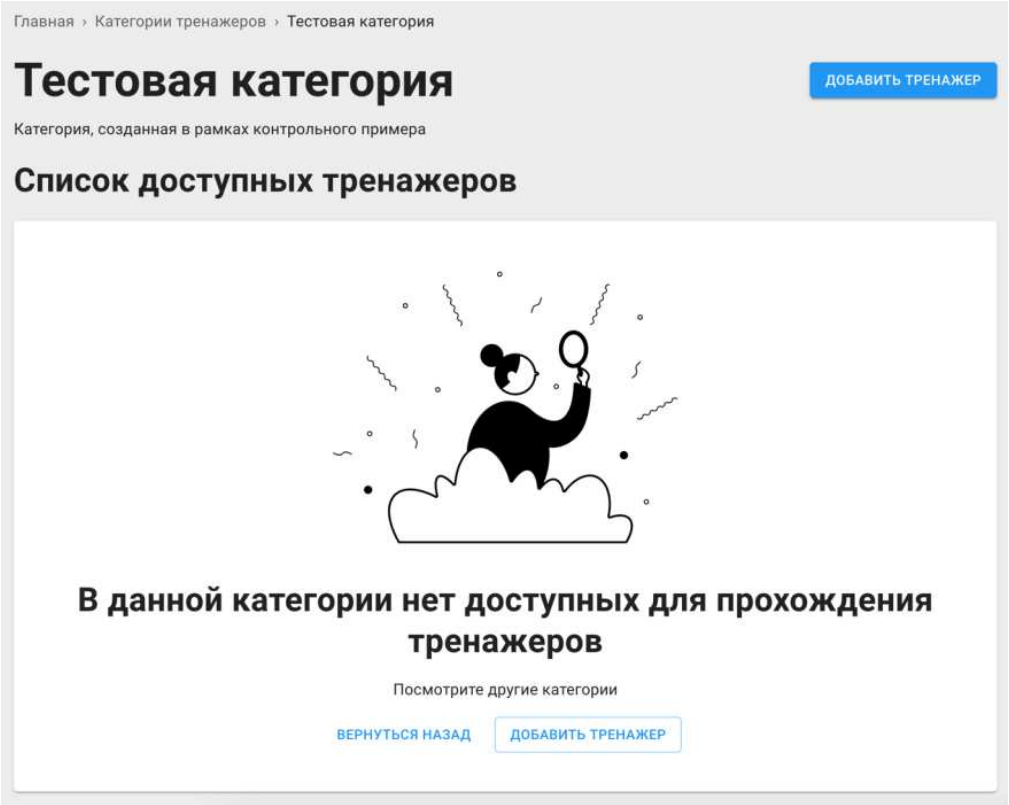


Рисунок 2.42 – Список тренажеров созданной категории

На рисунке 2.43 показан редактор тренажеров, в который попадает тренер после предыдущего действия. Здесь его задача – назвать тренажер, добавить к нему описание, выставить максимальное количество попыток прохождения, максимальное время на одну попытку и создать сценарий тренажера – вопросы и варианты ответа на них.

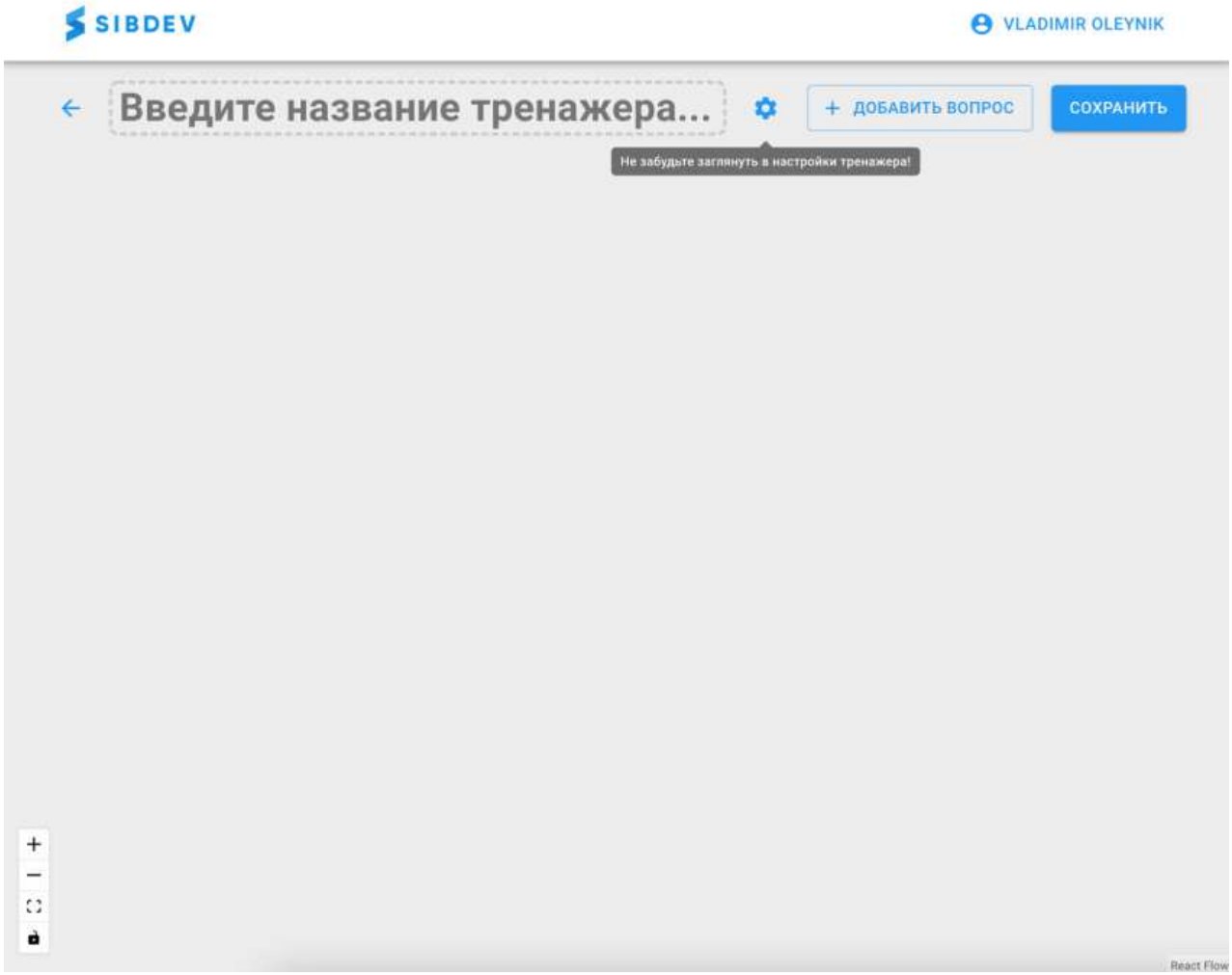


Рисунок 2.43 – Редактор тренажера

В поле «Введите название тренажера…» вводится название. По нажатию на иконку шестеренки, открывается модальное окно с основными настройками тренажера (рис. 2.44).

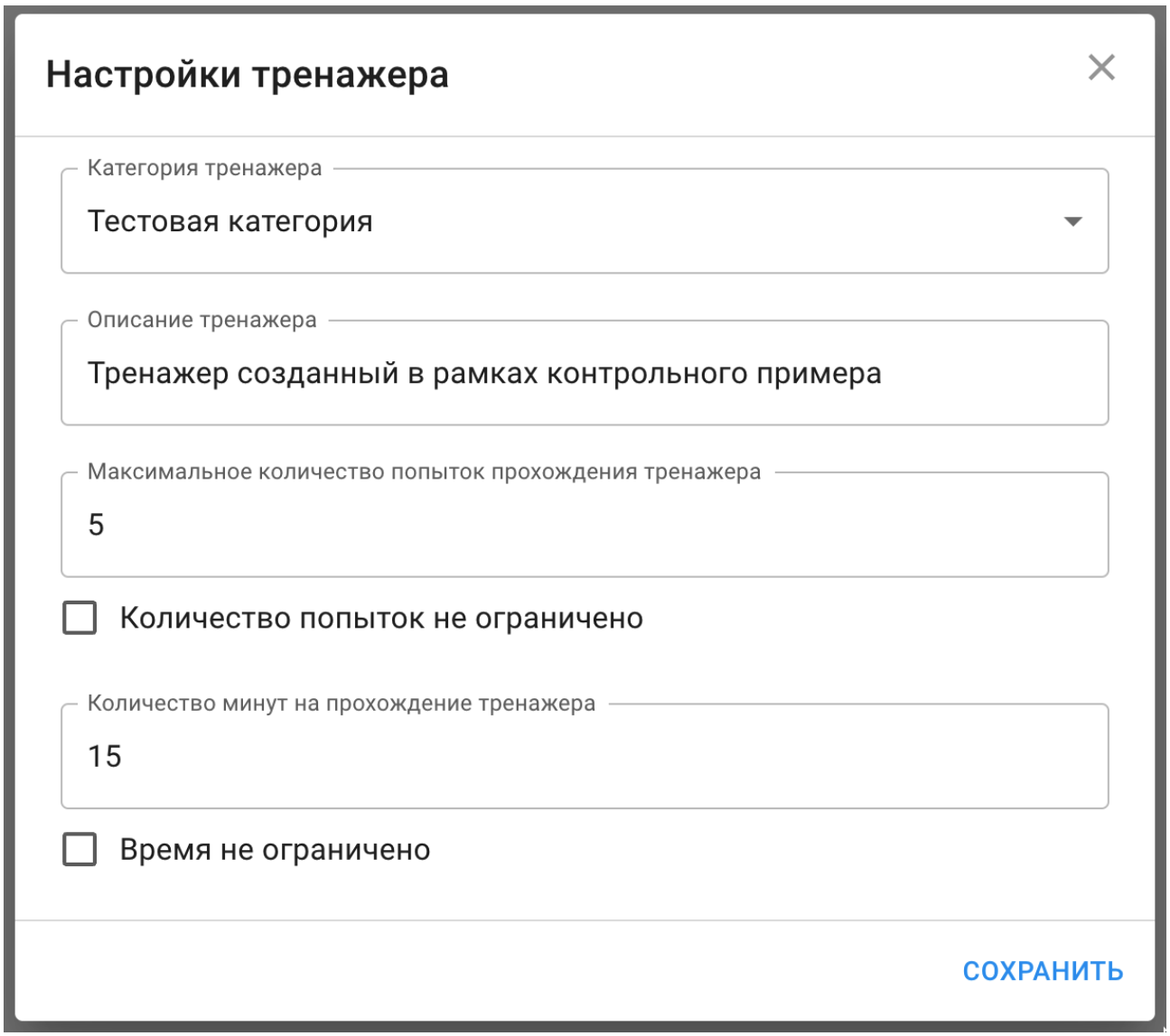


Рисунок 2.44 – Модальное окно настроек тренажера

Теперь, когда все настройки указаны, можно переходить к созданию сценария тренажера. Для этого необходимо нажать на кнопку «Добавить вопрос». В открывшемся модальном окне будет предложено ввести текст вопроса и ответы (рис. 2.45). Максимальное количество вариантов ответов не ограничено, их можно добавлять по нажатию на кнопку «+».

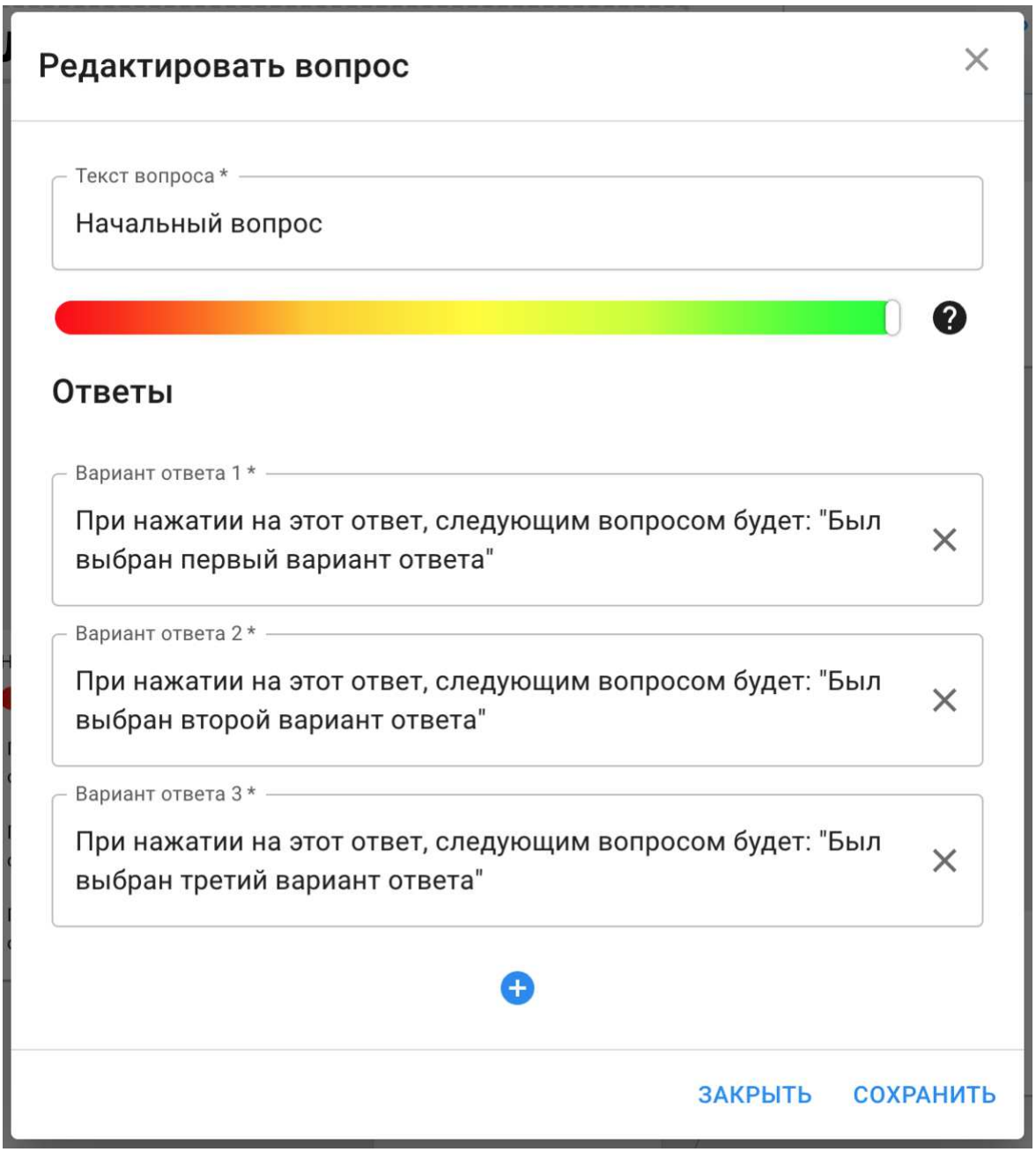


Рисунок 2.45 – Модальное окно добавления вопроса

После добавления вопроса он появляется в редакторе на общем полотне. По аналогии были добавлены остальные вопросы (рис. 2.46). Каждый из вопросов можно перемещать по полотну, удалять или редактировать.

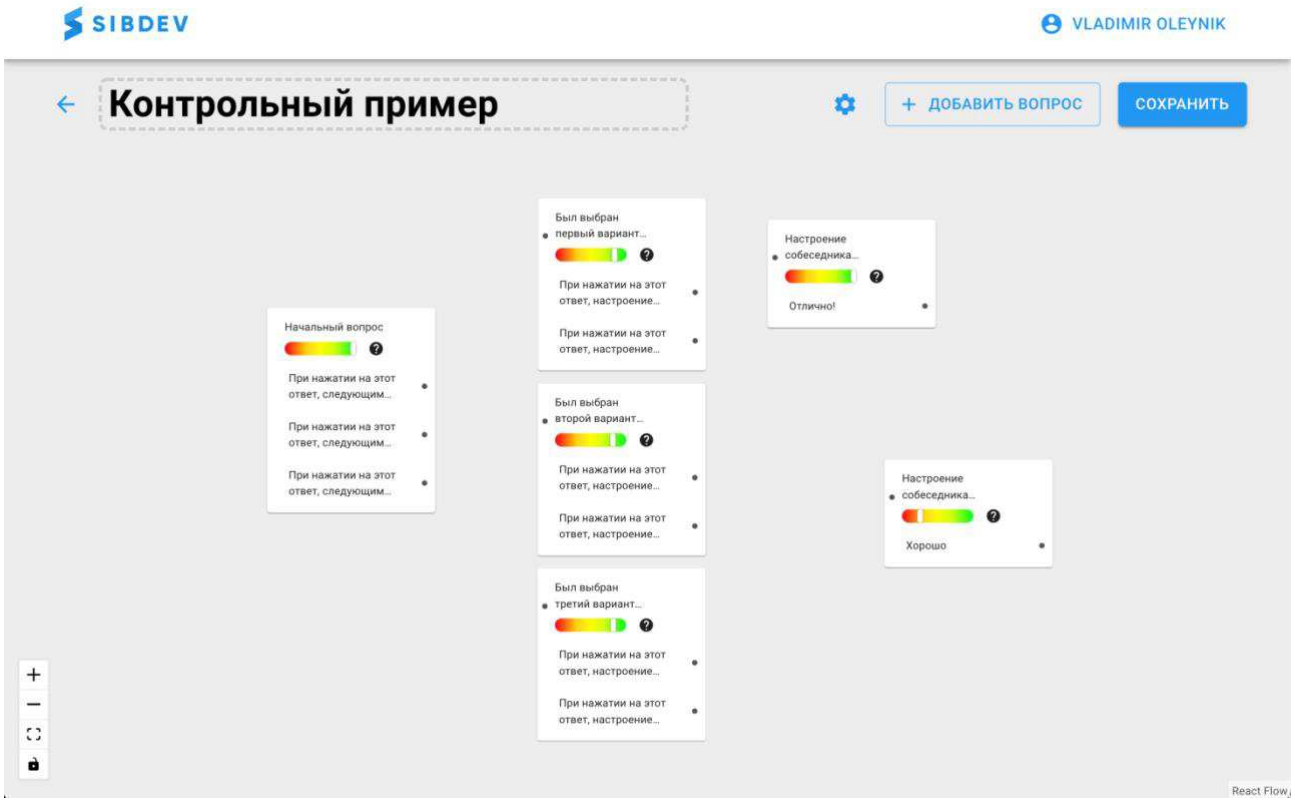


Рисунок 2.46 – Добавленные вопросы в редакторе тренажера

Следующим шагом нужно указать последовательность вопросов. Основная особенность заключается в том, что сценарий может быть нелинейным и меняться в зависимости от того, какой вариант ответа выбирает студент. С помощью данного редактора можно легко добиться такого поведения. Справа от каждого варианта ответа и слева от текста вопроса есть небольшие круглые точки – это места соединения. То есть, эти точки при помощи мыши можно соединить и это будет означать, что при выборе этого варианта ответа, следующим будет задан указанный вопрос. На рисунке 2.47 представлена итоговая последовательность вопросов.

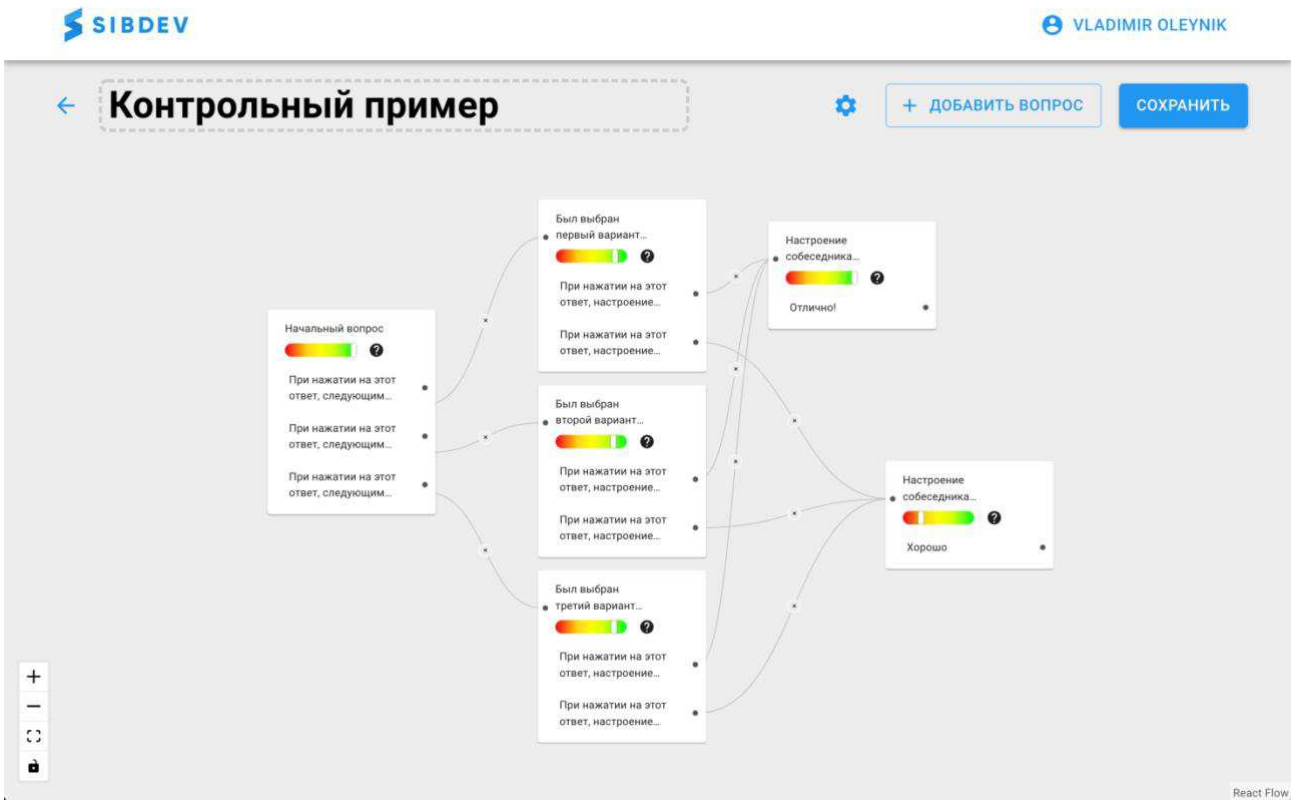


Рисунок 2.47 – Последовательность вопросов

После этого необходимо сохранить тренажер. После его сохранения он будет отображаться в созданной ранее категории (рис. 2.48).

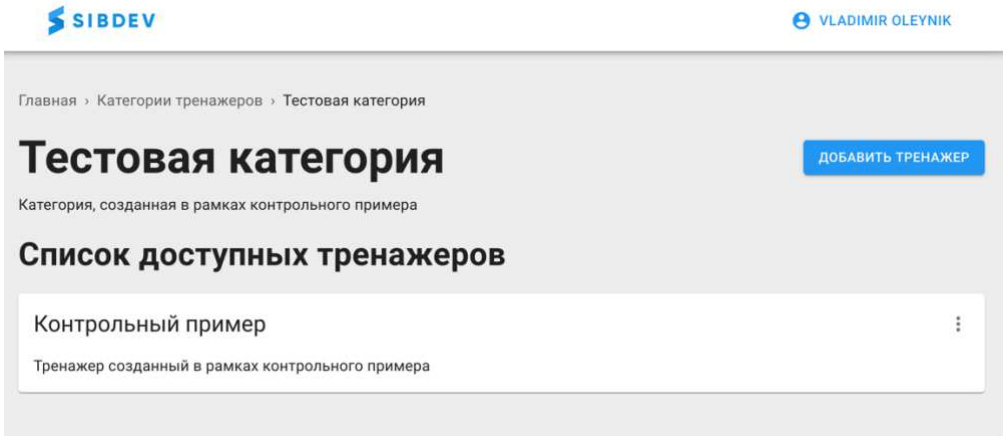


Рисунок 2.48 – Отображение созданного тренажера

Далее необходимо создать нового пользователя – студента, который будет проходить созданный тренажер. Для этого в правом верхнем углу нужно нажать на имя пользователя и в открывшемся меню выбрать пункт «Управление пользователями» (рис. 2.49).

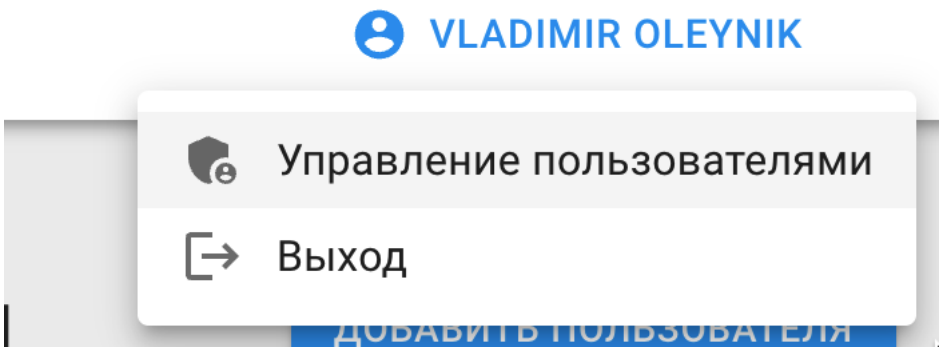


Рисунок 2.49 – Меню пользователя

На рисунке 2.50 показана открывшаяся страница управления пользователями.

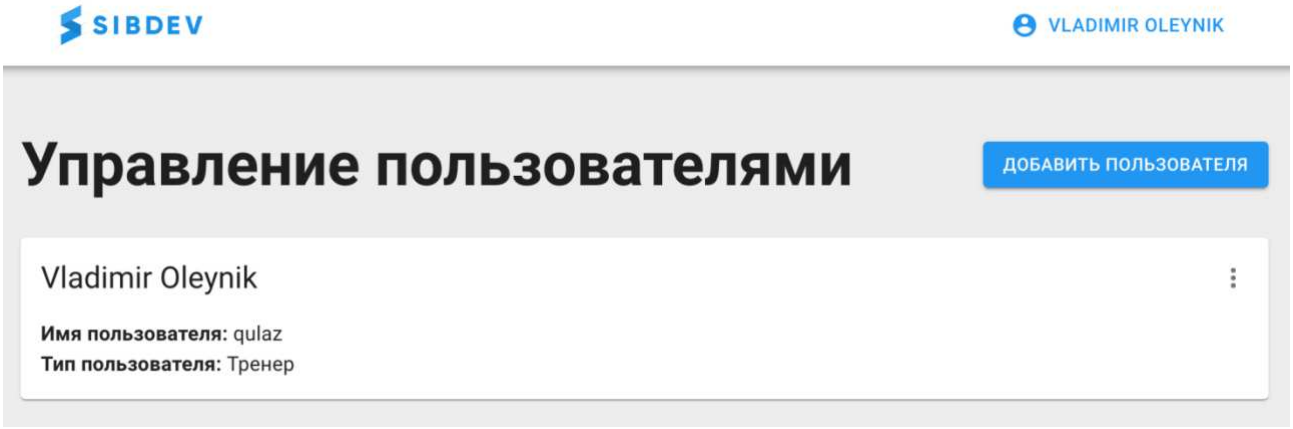


Рисунок 2.50 – Страница управления пользователями

Затем нужно нажать на кнопку «Добавить пользователя», после чего откроется модальное окно с формой создания пользователя (рис. 2.51).

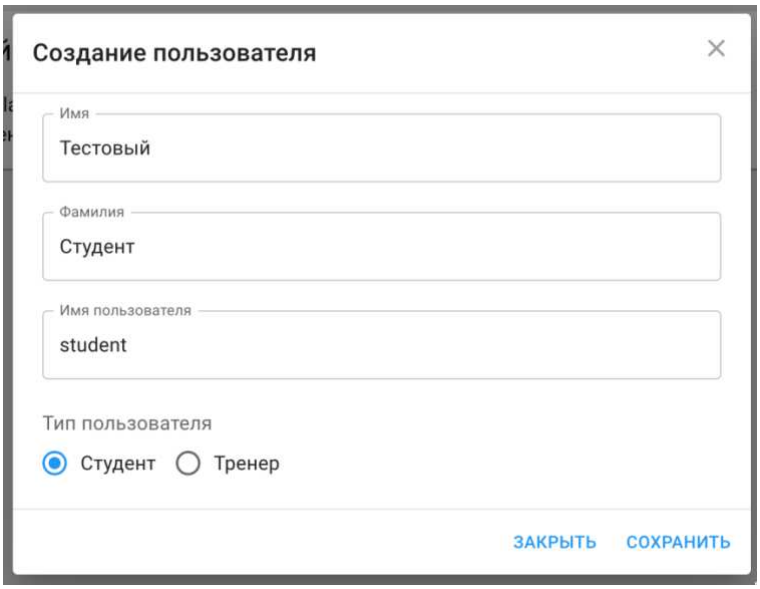


Рисунок 2.51 – Форма создания пользователя

На рисунке 2.52 показано модальное окно с данными для входа созданного пользователя, которое отображается при нажатии на кнопку «Сохранить».

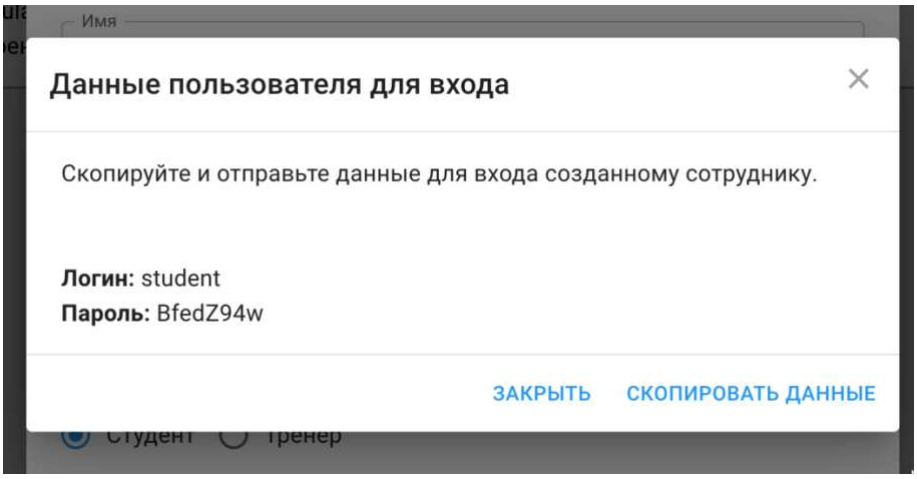


Рисунок 2.52 – Модальное окно с данными для входа

На этом контрольный пример в роли тренера завершен.

2.3.2 Тестирование функций диалогового тренажера в роли студента

После того, как тренер создал материал, студент может приступать к его выполнению. На рисунке 2.53 показано, что созданные в предыдущем пункте категория и тренажер доступны для студента.

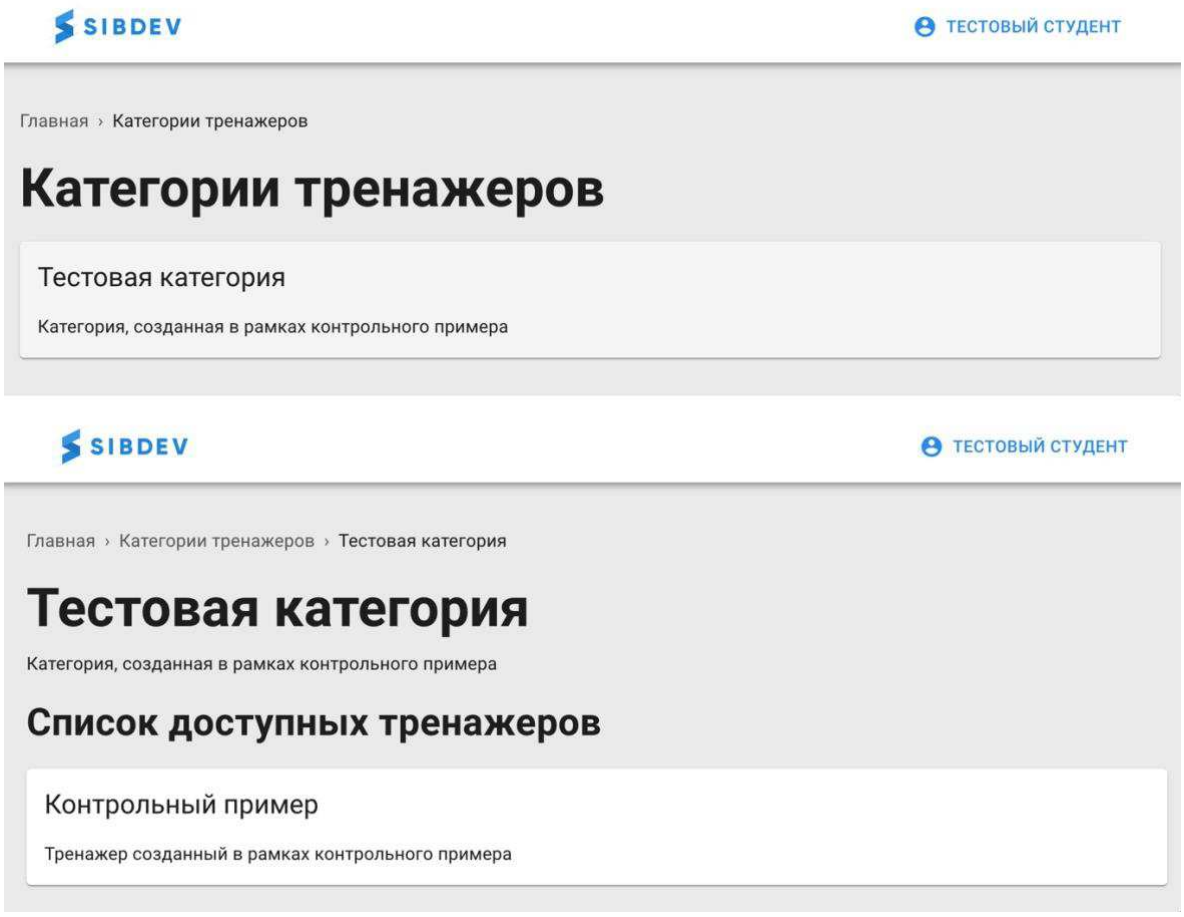


Рисунок 2.54 – Список категорий и тренажеров, доступных для студента

На рисунке 2.55 можно увидеть детали о тренажере – время выполнения, максимальное количество попыток и список попыток текущего студента. Как

можно заметить, все данные совпадают с указанными от лица тренера в прошлом пункте. Также список попыток выполнения закономерно пуст, так как данный студент ещё не приступал к его выполнению. Чтобы это сделать, необходимо нажать на кнопку «Начать выполнение».

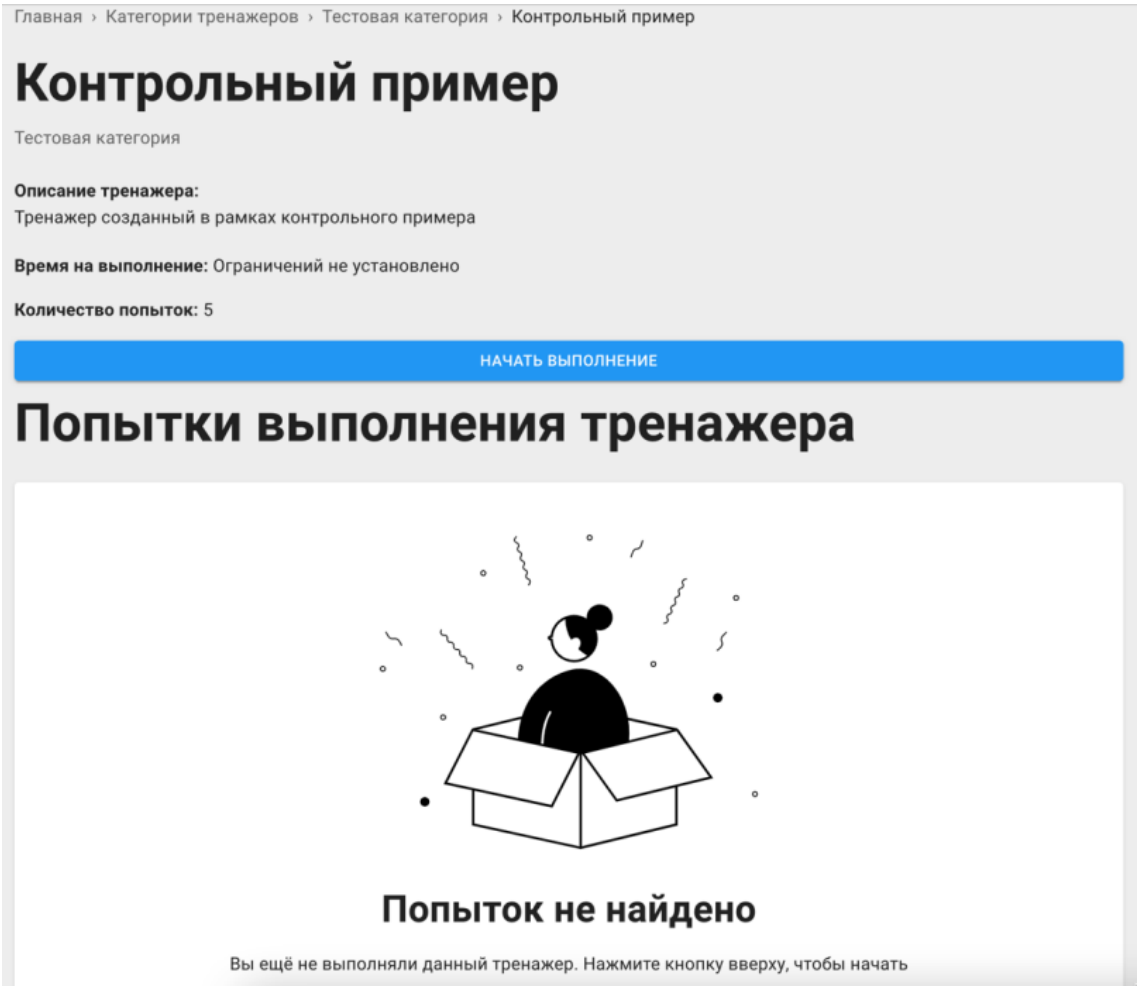


Рисунок 2.55 – Созданный тренажер от лица студента

На рисунке 2.56 показан процесс выполнения тренажера. Как можно заметить, он проходит согласно созданному сценарию. Он был создан специально так, чтобы было наглядно видно какой вопрос должен быть задан после выбора каждого варианта ответа.

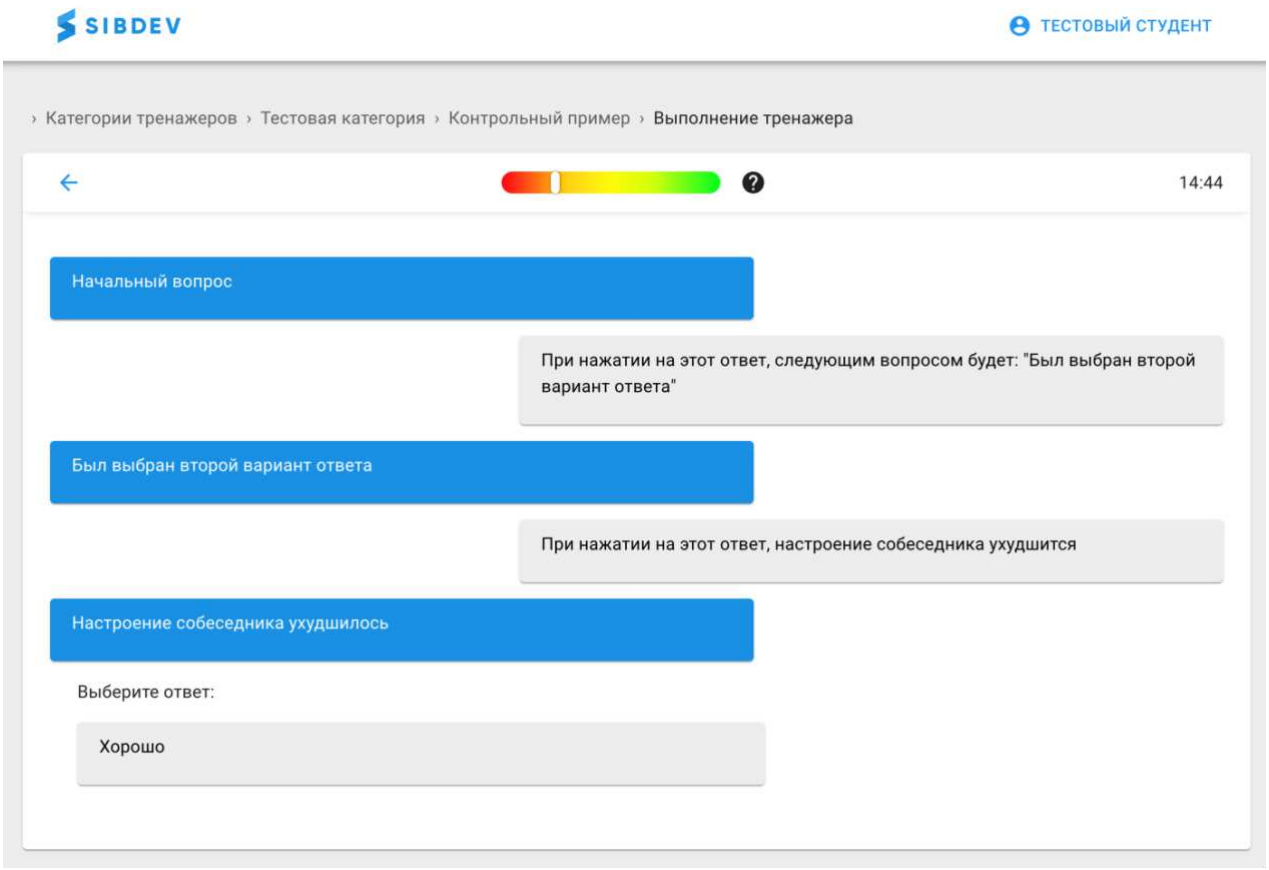


Рисунок 2.56 – Выполнение тренажера

После выполнения тренажера он попадает в список попыток выполнения (рис. 2.57).

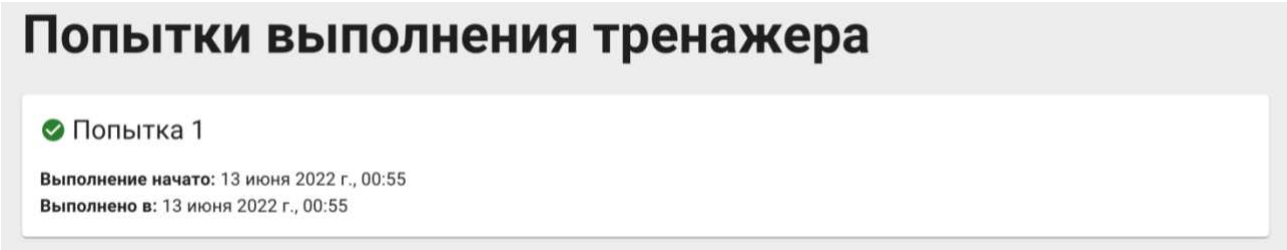


Рисунок 2.57 – Список попыток выполнения тренажера

Также если лимит попыток был истрачен, система не даст начать новую попытку (рис. 2.58).

Рисунок 2.58 – Страница тренажера с потраченными попытками выполнения

На этом контрольный пример в роли студента можно считать успешно пройденным.

2.3.3 Тестирование функций выгрузки отчета о результатах прохождения диалогового тренажера

Последний этап контрольного примера – скачивание отчета о пройденных тренажерах. Для этого нужно снова зайти от имени тренера на страницу управления пользователями и скачать отчет (рис. 2.59).

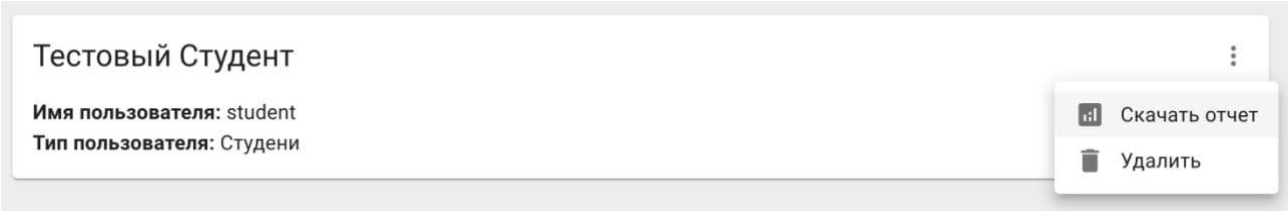


Рисунок 2.59 – Скачивание отчета о пройденных тренажерах

Открыв скачанный отчет можно увидеть, что тестовый студент выполнил 5 попыток прохождения созданного тренажера для контрольного примера. Из

них один не был завершен вовремя, другой ещё находится в процессе выполнения (рис. 2.60).

Пользователь Тестовый студент

Категория тренажера	Тренажер	Попытка	Дата начала прохождения	Дата завершения прохождения	Настроение собеседника	Среднее настроение собеседника
Тестовая категория	Контрольный пример	1	12.06.22 20:55	12.06.22 20:55	32	49.25
		2	12.06.22 20:55	12.06.22 20:55	80	
		3	12.06.22 20:55	12.06.22 20:55	85	
		4	12.06.22 20:56	НЕ ЗАВЕРШЕН	0	
		5	12.06.22 20:56	В ПРОЦЕССЕ	-	

Рисунок 2.60 – Отчет о прохождении тренажеров

На этом контрольный пример можно считать завершенным. Контрольный пример был успешно пройден и был получен ожидаемый результат.

Вывод по разделу «Описание разработки диалогового тренажера»

В ходе выполнения раздела разработаны серверная и клиентская части диалогового тренажера. Для разработки серверной части использовались язык *Go*, *GraphQL API* для обмена данными и библиотеки *gqlgen*, *mock* и другие.

Клиентская часть была разработана на языке *TypeScript* с использованием библиотек *React* и *Apollo*.

Было настроено клиент-серверное взаимодействие между обеими частями, и на выходе получился целостный программный продукт.

Функциональность системы была протестирована на контрольном примере как от лица тренера, так и от лица студента. В рамках контрольного примера тренером были созданы категория тренажеров, сам тренажер и пользователь с ролью студент. Студент же проверил, что созданные сущности доступны, прошел созданный тренажер, истратив при этом все 5 доступных

попыток. Затем тренер посмотрел результаты студента, скачав отчет о его прохождениях тренажера.

3 Оценка экономической эффективности разработки и внедрения диалогового тренажера

Совокупная стоимость владения или стоимость жизненного цикла – это общая величина целевых затрат, которые вынужден нести владелец с момента начала реализации вступления в состояние владения до момента выхода из состояния владения и исполнения владельцем полного объёма обязательств, связанных с владением [15].

3.1 Капитальные затраты

Капитальные затраты на диалоговый тренажер носят разовый характер и вычисляются по следующей формуле

$$К = К_{пр} + К_{тс} + К_{лс} + К_{по} + К_{ио} + К_{об} + К_{оэ}, \quad (1)$$

где $К_{пр}$ – затраты на проектирование диалогового тренажера;

$К_{тс}$ – затраты на технические средства управления;

$К_{лс}$ – затраты на создание линий связи локальных сетей;

$К_{по}$ – затраты на программные средства;

$К_{ио}$ – затраты на формирование информационной базы;

$К_{об}$ – затраты на обучение персонала;

$К_{оэ}$ – затраты на опытную эксплуатацию.

3.1.1 Затраты на проектирование диалогового тренажера

Затраты на проектирование диалогового тренажера рассчитываются по следующей формуле

$$К_{пр} = К_{зп} + К_{ипс} + К_{свт} + К_{проч}, \quad (2)$$

где $К_{зп}$ – затраты на заработные платы проектировщиков;

$К_{ипс}$ – затраты на инструментальные программные средства;

$К_{свт}$ – затраты на средства вычислительной техники;

$К_{проч}$ – прочие затраты.

Расчет заработной платы. Для разработки проекта требуется два специалиста – веб-дизайнер и программист со знаниями языков *Go* и *TypeScript,* фреймворка *React*, технологий *GraphQL, Apollo* и *Git*.

Для оплаты труда программиста и веб-дизайнера будет использована тарифная оплата труда, соответствующая минимальной оплате труда (МРОТ) в Красноярском крае [16]. Таким образом, заработная плата (ЗП) обоих специалистов равна 13890 руб. без учета НДФЛ, северного коэффициента (до 30%) и районного коэффициента (30%), но так как предполагается, что диалоговый тренажер будет разрабатываться младшими специалистами, стаж работы которых менее года, то применять к их заработной плате северный коэффициент не требуется.

Срок работы программиста – 33 полных рабочих дня (8 часов).

Срок работы веб-дизайнера – 4 полных рабочих дня (8 часов).

Расчет заработной платы, исходя из 23 рабочих дней и 8-часового рабочего дня, представлен в таблице 3.1.

Таблица 3.1 – Расчет заработной платы

Доходы, руб.		**Расходы, руб.**	
Оклад	13 890		
Северный коэффициент	0		
Районный коэффициент	4 167		
Итог в месяц / в час	18 057 / 98,14	НДФЛ	2 347,41
Итого на руки	15 709,57		

$К_{зп}$ – это величина Фонда оплаты труда (ФОТ), который включает в себя заработную плату, а также дополнительные средства для обязательных отчислений во внебюджетные фонды (пенсионное, социальное и медицинское страхование). Величина данных отчислений равна 30,2% от ЗП [17].

Таким образом, $К_{зп}$ будет равняться заработной плате программиста за 33 рабочих дня, заработной плате веб-дизайнера за 4 рабочих дня и отчислениям во внебюджетные фонды от этой суммы:

$$К_{зп} = \big((98{,}14 * 33 * 8) + (98{,}14 * 4 * 8)\big) * 1{,}302 = 37\,822{,}37 \text{ рублей.}$$

Затраты на инструментальные программные средства. Этап проектирования диалогового тренажера будет происходить с использованием бесплатного ПО, перечень которого представлен в таблице 3.2.

Таблица 3.2 – Перечень используемого программного обеспечения

№	**Наименование ПО**	**Количество**	**Стоимость, руб.**
1	Manjaro Linux	2	0
2	Figma	1	0
3	Google Chrome	2	0
4	Visual Studio Code	1	0

Таким образом, $К_{ипс} = 0$.

Затраты на средства вычислительной техники. Срок полезного использования техники составляет 5 лет. Перечень необходимых вычислительных средств для разработки представлен в таблице 3.3.

Таблица 3.3 – Перечень вычислительных средств

Название	**Комплектующие**			**Стоимость, ₽**
	Название	**Модель**	**Стоимость, ₽**	
Компьютер программиста	Процессор	Intel Core i5-9400 OEM	13799.0	49 122
	Материнская плата	GIGABYTE H310M H 2.0	4299.0	
	Корпус	DEEPCOOL MATREXX 30	2399.0	
	Кулер	AeroCool Verkho 2	999.0	
	Оперативная память	Patriot Signature Line 16 ГБ	5599.0	
	SSD-накопитель	Samsung 870 EVO 500 ГБ	5599.0	
	Блок питания	CoolerMaster MWE Gold 550 - V2	6350.0	
	Монитор	Xiaomi 23.8» Mi Desktop Monitor 1C	8999.0	
	Клавиатура	Sven Standard 303	699.0	
	Мышь	Jet.A OM-U50 Comfort	380.0	
Компьютер веб-дизайнера	Процессор	AMD Ryzen 3 2200G BOX	10699	35 093
	Материнская плата	GIGABYTE GA-A320M-H 2.0	4199	
	Корпус	DEXP DC-302B	1450	
	Оперативная память	Patriot Signature 8 ГБ	2650	
	SSD-накопитель	DEXP L4, 256 ГБ	2499	
	Блок питания	Corsair CV450	3999	
	Монитор	Xiaomi 23.8» Mi Desktop Monitor 1C	8999	
	Клавиатура	Oklick 305M	399	
	Мышь	Jet.A OM-U50 Comfort	199	

Компьютер разработчика будет использоваться 33 дня, а компьютер веб-дизайнера 4 дня. Исходя из этого, необходимо рассчитать амортизацию оборудования исходя из срока его использования в проекте.

Ставка амортизации за год:

$$А_{год} = С_t * Н_а, \quad (3)$$

где $С_t$ – стоимость оборудованиня;

$Н_а$ – норма автоматизации.

$$Н_а = 1/\text{нормативный срок службы} * 100\%, \quad (4)$$

Норма автоматизации для используемых компьютеров:

$Н_а = 1/5 * 100\% = 20\%$.

Для компьютера программиста ставка амортизации за год равна:

$А_{год} = 49\ 112 * 20\% = 9\ 922{,}4$ руб.

Для компьютера веб-дизайнера ставка амортизации за год равна:

$А_{год} = 35\ 093 * 20\% = 7\ 018{,}6$ руб.

Расчёт амортизации на проектирование выполнялся по формуле

$$А_{пр} = (А_{год} * К_{дэ}) / К_{дг}, \quad (5)$$

где $А_{год}$ – ставка амортизации за год;

$К_{дэ}$ – количество дней эксплуатации;

$К_{дг}$ – количество рабочих дней в году.

Амортизация для компьютера программиста:

$А_{пр} = (9\ 922{,}4 * 33) / 247 = 1\ 326$ руб.

Амортизация для компьютера веб-дизайнера:

$А_{пр} = (7\ 018{,}6 * 4) / 247 = 114$ руб.

Таким образом, затраты на средства вычислительной техники равны:

$К_{свт} = 1\ 326 + 114 = 1\ 440$ рублей.

Расчет прочих расходов. На прочие расходы принято выделять 3% от суммы проектных затрат:

$К_{проч} = (37\ 822{,}37 + 0 + 1\ 440) * 3\% = 1\ 178$ рублей.

Расчет затрат на проектирование ИС. Таким образом, учитывая расчеты, приведенные выше, следует, что затраты на проектирование равны:

$К_{пр} = 37\ 822{,}37 + 0 + 1\ 440 + 1\ 178 = 40\ 440$ рублей.

Затраты на проектирование диалогового тренажера представлены в таблице 3.4 и на рисунке 3.1.

Таблица 3.4 – Затраты на проектирование ИС

Затраты	**Состав затрат**	**Планируемая сумма, руб.**
Затраты на проектирование ИС	Затраты на ЗП	37 822,37
	Затраты на ПО	0
	Затраты на средства вычислительной техники	1 440
	Прочие затраты	1 178
Итого		40 440

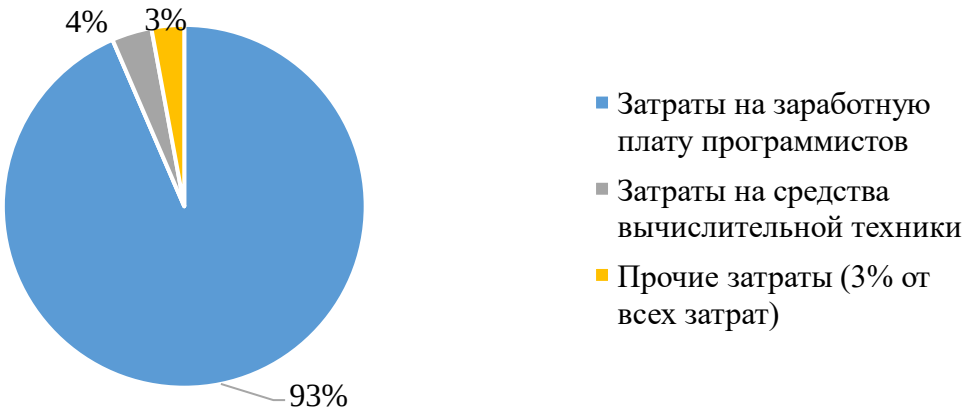


Рисунок 3.1 – Структура проектных затрат

3.1.2 Расчет капитальных затрат

Расчет капитальных затрат будет выполнен по формуле (1), которая представлена в п.3.1.

$К_{пр}$ подсчитаны в п.3.1.1 и составляют 40 440 рублей.

$К_{тс}$ равны 0 руб., так как величина амортизационных затрат, приходящихся на данный проект ничтожно мала, поэтому учитываться не будет.

$К_{лс}$ равны 0 руб., так как диалоговый тренажер не требует линий связи локальной сети.

$К_{по}$ равны 0 руб., так как для работы диалоговый тренажер используется только бесплатное ПО (веб-браузер с клиентской стороны и веб-сервер с серверной);

$К_{ио}$ равны 0 руб., так как информационная база диалогового тренажера была создана на этапе проектирования.

$К_{об}$ – для обучения персонала потребуется программист. На этом этапе он должен составить документацию по работе с диалоговым тренажером и лично объяснить ответственным основные принципы работы с диалоговым тренажером и ответить на вопросы. На это заложено 2 рабочих дня – 16 часов. $К_{об} = (98{,}14 * 16) * 1{,}302 = 2\ 044$ рублей.

$К_{оэ}$ – для того, чтобы удостовериться, что разработанная система работает верно, необходимо её протестировать. Для этого программист должен провести ручное E2E (end-to-end) тестирование и исправить все найденные недочеты, если таковые будут обнаружены. На это выделено 2 рабочих дня – 16 часов. $К_{об} = (98{,}14 * 16) * 1{,}302 = 2\ 044$ рублей.

Таким образом, капитальные затраты составляют:

$$К = 40\ 400 + 0 + 0 + 0 + 0 + 2\ 044 + 2\ 044 = 44\ 488 \text{ рублей.}$$

Список капитальных затрат представлен в таблице 3.5 и на рисунке 3.2.

Таблица 3.5 – Список капитальных затрат

Наименование затрат	**Планируемая сумма, руб.**
Затраты на проектирование ИС	40 400
Затраты на технические средства управления	0
Затраты на создание линий связи локальных сетей	0
Затраты на программные средства	0
Затраты на формирование информационной базы	0
Затраты на обучение персонала	2 044
Затраты на опытную эксплуатацию	2 044
Итого	44 488

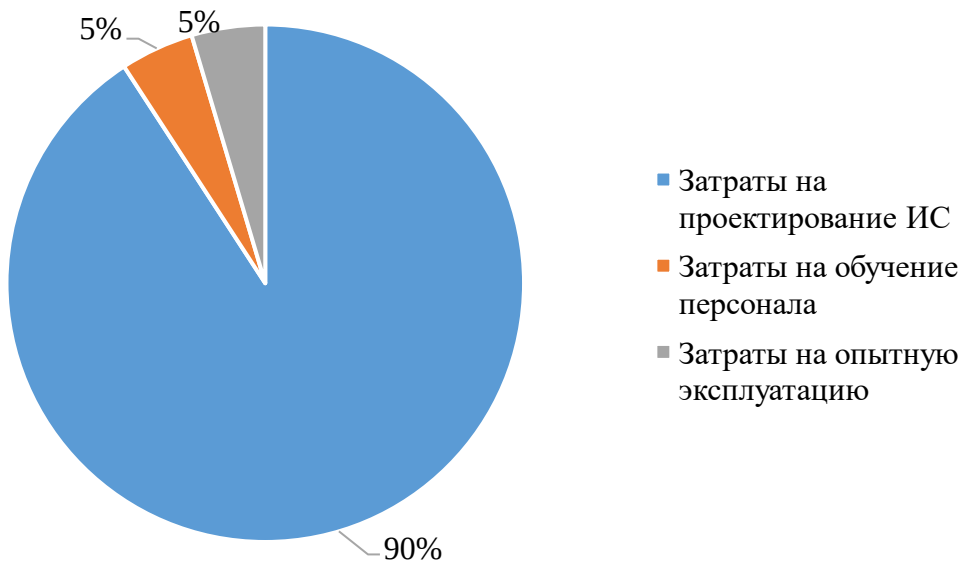


Рисунок 3.2 – Структура капитальных затрат

3.2 Эксплуатационные затраты

Эксплуатационные затраты, в отличие от капитальных, являются повторяющимися и рассчитываются в сумме за год. Рассчитываются они по следующей формуле

$$С = С_{зп} + С_{ао} + С_{то} + С_{лс} + С_{ни} + С_{проч}, \quad (6)$$

где $С_{зп}$ – зарплата персонала, работающего с информационной системой;

$С_{ао}$ – амортизационные отчисления;

$С_{то}$ – затраты на техническое обслуживание;

$С_{лс}$ – затраты на использование глобальных сетей;

$С_{ни}$ – затраты на носители информации;

$С_{проч}$ – прочие затраты.

Зарплата персонала, работающего с диалоговым тренажером. С данной информационной системой будет работать опытный менеджер компании, у которого, помимо всего прочего, в обязанности входит обучение и развитие сотрудников. Его задача – создавать материалы для обучения и проводить тренинги в различных режимах. Так как диалоговый тренажер используется для частичной автоматизации процесса обучения, дополнительных затрат на ЗП данных работников не потребуется.

$$С_{зп} = 0 \text{ рублей.}$$

Затраты на амортизационные отчисления. Планируется, что система будет работать на облачном сервере, базовые характеристики которого: 2 ядра, 2 гигабайта оперативной памяти, 40 гигабайта накопителя. Стоимость аренды такого сервера составляет 590 рублей в месяц. Таким образом затраты на амортизационные отчисления составляют:

$$С_{ао} = 590 * 12 = 1\,180 \text{ рублей.}$$

Затраты на техническое обслуживание. Проект в процессе эксплуатации может требовать своего обслуживания. Это необходимо учесть заранее и выделить предполагаемое время специалистов, которое у них займет это обслуживание. Предлагается выделить на это 20 часов рабочего времени программиста в год.

$С_{то} = (98{,}14 * 20) * 1{,}302 = 2\ 556$ рублей.

Затраты на использование глобальных сетей. Подключение к глобальным сетям включено в стоимость оплаты аренды сервера.

$С_{лс} = 0$ рублей.

Затраты на носители информации. База данных в проекте не предполагает хранения огромных массивов данных, поэтому для её хранения будет достаточно накопителя, предоставленного тарифом облачного сервера.

$С_{ни} = 0$ рублей.

Прочие затраты. Прочие затраты обычно составляют от 1 до 5 процентов от общей суммы эксплуатационных затрат. Оптимальным значением для данного проекта будет 3 процента.

$С_{проч} = (0 + 1\ 180 + 2\ 556 + 0 + 0 + 0) * 3\% = 112$ рублей.

Таким образом, эксплуатационные затраты равны:

$С = 0 + 1\ 180 + 2\ 556 + 0 + 0 + 0 + 112 = 3\ 848$ рублей.

Список эксплуатационных затрат представлен в таблице 3.5 и на рисунке 3.3.

Таблица 3.5 – Эксплуатационные затраты

Наименование затрат	**Планируемая сумма, руб.**
ЗП персонала, работающего с ИС	0
Амортизационные отчисления	1 180

Продолжение таблицы 3.5

Затраты на техническое обслуживание	2 556
Затраты на использование глобальных сетей	0
Затраты на носители информации	0
Прочие затраты	112
Итого	3 848

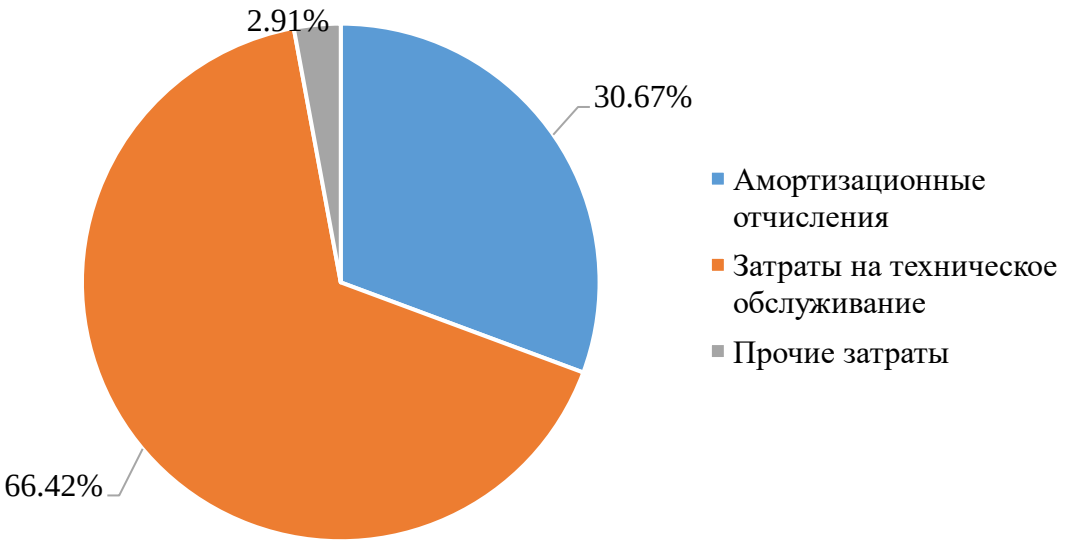


Рисунок 3.3 – Структура эксплуатационных затрат

3.3 Совокупная стоимость владение системой

TCO (*Total Cost of Owneship* – совокупная стоимость владения) – методика, разработанная в конце 80-х годов *XX* века компанией *Gartner Group* для расчета финансовых затрат на владение компьютерами. Она используется для того, чтобы рассчитать финансовые затраты на IT. Показатель совокупной стоимости владения диалоговым тренажером рассчитывается по формуле

$$TCO = DE + IC1 + IC2, \quad (7)$$

где DE (direct expenses) – прямые расходы;

IC1 (indirect costs) – косвенные расходы первой группы;

$IC2$ – косвенные расходы второй группы.

Для данного проекта косвенные расходы не являются значимыми, следовательно $TCO \approx DE$:

$$DE = DE_1 + DE_2 + DE_3 + DE_4 + DE_5 + DE_6 + DE_7 + DE_8, \quad (8)$$

где DE_1 – капитальные затраты;

DE_2 – расходы на управление ИТ;

DE_3 – расходы на техническую поддержку АО и ПО;

DE_4 – расходы на разработку прикладного ПО внутренними силами;

DE_5 – расходы на аутсорсинг;

DE_6 – командировочные расходы;

DE_7 – расходы на услуги связи;

DE_8 – другие группы расходов.

Собрав все данные вместе, получим:

$DE_1 = \text{К} = 44\,488$ рублей;

$DE_2 = \text{С}_{\text{зп}} = 0$ рублей;

$DE_3 = \text{С}_{\text{то}} + \text{С}_{\text{ао}} = 2\,556 + 1\,180 = 3\,736$ рублей;

$DE_4 = 0$ рублей;

$DE_5 = 0$ рублей;

$DE_6 = 0$ рублей;

$DE_7 = \text{С}_{\text{лс}} = 0$ рублей;

$DE_8 = \text{С}_{\text{проч}} + \text{С}_{\text{ни}} = 112 + 0 = 112$ рублей.

Таким образом, прямые затраты равны:

$DE = 44\,488 + 0 + 3\,736 + 0 + 0 + 0 + 0 + 112 = 48\,336$ рублей.

$TCO = 48\,336$ рублей.

3.4 Оценка рисков реализации проекта

При реализации проекта могут возникнуть различные проблемы. Чтобы они не стали неожиданностью, существует практика оценки рисков. С её помощью происходит анализ проблем, которые могут возникнуть, вероятность их возникновения, уровень влияния на реализацию проекта, а также продумываются пути их решения. Таким образом, при грамотной оценке рисков, проблемы не будут являться неожиданностью, будут уже проработанные пути решения, которые помогут в этой ситуации.

При разработке диалогового тренажера можно столкнуться со следующими рисками:

1. Риск соответствия имеет низкую вероятность, так как перед разработкой диалогового тренажера было проведено его моделирование. В рамках моделирования был разобран весь требуемый функционал, были представлены пользовательские сценарии взаимодействия с системой. Но все же имеется низкая вероятность, так как во время обсуждения моделей с заказчиком, могли возникнуть недопонимания или же он мог недостаточно точно трактовать свои желания.

2. Риск увеличения сроков реализации имеет среднюю вероятность, так как до непосредственной реализации, разработчиком был оценен проект, сложность его реализации и срок, в который он сможет его выполнить. Но, так как разработка информационной системы — это комплексный процесс, во время которого могут возникнуть непредвиденные трудности, рекомендуется заранее дополнительно заложить время на их решение.

3. Риск низкой эффективности. Под ним подразумевается, что разрабатываемая система даст меньший эффект для компании, чем предполагалось. Он имеет низкую вероятность, так как был проведен анализ предметной области, изучен опыт других компаний. Помимо этого, он имеет и

низкий уровень влияния, так как проект реализуется с этим учетом как *MVP* (от англ. *Minimal Viable Product*) – минимально жизнеспособный продукт, обладающий минимальными, но достаточными для удовлетворения первых потребителей функциями [18]. Его цель – быстро проверить гипотезу и собрать обратную связь для дальнейшего развития продукта.

Таблица 3.6 – Перечень рисков

№	Группы рисков	Перечень рисков проектов	Уровень влияния	Вероятность	Возможные решения риска
1	Риск соответствия	Недостаточность понимания заказчика и разработчика	Высокий	Низкая	Консультация разработчика с заказчиком
2	Риск увеличения сроков реализации	Непредвиденные расходы, срыв сроков внедрения	Средний	Средняя	Заложить дополнительное время на решение возможных технических проблем
3	Риск низкой эффективности продукта		Низкий	Низкая	Дополнительное изучение опыта других компаний, проведение опросов и аналитики

3.5 Оценка экономической эффективности реализации проекта

3.5.1 Анализ рынка продуктов-аналогов. Установление стоимости программного продукта

На рынке существует ряд аналогов разрабатываемого диалогового тренажера. Были выбраны 3 популярных аналогичных решения: комплекс *iSpring, Online Test Pad* и Неодиалог. Проведено сравнение их функционала, стоимости и общего удобства.

Сравнительный анализ продуктов-аналогов показан в таблице 3.7.

Таблица 3.7 – Сравнительный анализ продуктов-аналогов

Критерии оценки	**Комплекс iSpring**	**Online Test Pad**	**Неодиалог**
Функционал	Комплекс iSpring состоит из конструктора материалов и корпоративного портала. Такая комбинация позволяет создать перечень материалов и организовать их в единый обучающий курс, который будет доступен на корпоративном портале. Помимо этого, предоставляются инструменты по аналитике и отслеживания успеваемости.	Платформа по созданию тестов разных форматов, выполненная в виде веб-приложения. Она позволяет создавать тесты, опросы, кроссворды и диалоговые тренажеры. Платформа изначально не была рассчитана на корпоративное использование, поэтому не имеет функционала по созданию группы материалов, доступных определенному кругу лиц.	Неодиалог предоставляет конструктор диалоговых тренажеров и среду для их выполнения. Для администраторов доступна детальная статистика и отчеты по группам. Основной акцент делается на поддержку аудио/видео тренажеров. Обучаемый может отвечать на вопросы голосом.
Оценка функционала	4/5	2/5	4/5
Дополнительное ПО	Веб-браузер для прохождения тренажера и Microsoft PowerPoint для создания материалов	Веб-браузер	Веб-браузер
Стоимость за 25 человек	4 300 руб./мес. за корпоративный портал и 47 000 руб./год за конструктор материалов	Бесплатно	55 000 руб./мес.

Анализируя данные таблицы, можно сделать вывод, что значительным недостатком большинства аналогов является их цена – они предлагают множество функций, которые не планируются использовать, но они включены в стоимость.

Установление стоимости программного продукта.

1. Затратный метод.

Стоимость программного продукта рассчитывается на основании расчета капитальных затрат, определяя себестоимость продукта, прибавляя прибыль равная 10% от себестоимости, прибавляя затраты на рекламу и продвижение продукта.

Капитальные затраты на создание диалогового тренажера.

К = 44 488 рублей.

Прибыль в размере 10% от себестоимости:

П = 44 488 * 0,1 = 4 449 рублей.

Так как система разрабатывается под конкретного заказчика, необходимости в рекламе и продвижении нет. Поэтому затраты на рекламу равны 0 рублей.

Итоговая стоимость продукта:

44 488 + 4 449 = 48 937 рублей.

2. Доходный метод.

Для того, чтобы определить стоимость данного программного продукта доходным методом, необходимо посчитать доходы, которые будет приносить данное веб-приложение. Диалоговый тренажер создается для удовлетворения внутренних нужд студии веб-разработки «SibDev», поэтому она будет бесплатной и не будет приносить доход.

3.5.2 Экономическая эффективность реализации проекта

Под эффективностью автоматизированного преобразования

экономической информации понимают целесообразность применения средств вычислительной и организационной техники при форматировании, передаче и обработки данных. Обобщенным критерием экономической эффективности является минимум затрат живого и общественного труда. Для этого необходимо рассмотреть базовый вариант (до внедрения проекта) и проектный вариант (после внедрения проекта).

Значимость технических решений (ЗТР) вычисляется по следующей формуле

$$ЗТР = к_а * к_п * к_с + к_м * к_о * к_ш, \quad (9)$$

где $к_а$ – коэффициент актуальности;

$к_п$ – коэффициент соответствия программ важнейших работ научно-технического прогресса;

$к_с$ – коэффициент сложности;

$к_м$ – коэффициент места использования;

$к_о$ – коэффициент объема использования;

$к_ш$ – коэффициент широты охвата охранными мероприятиями;

В таблице 3.8 приведены коэффициенты и ЗТГ базового и разрабатываемого вариантов проекта.

Таблица 3.8 – Коэффициенты и ЗТР базового и разрабатываемого варианта

Коэффициенты	**Базовый вариант**	**Разрабатываемый вариант**
$к_а$	1	1,3
$к_п$	1	1
$к_с$	1	2,3
$к_м$	1	1
$к_о$	1	1,7
$к_ш$	1	1
ЗТР	2	4,69

Коэффициенты:

$к_а = 1.3$, т.к. диалоговый тренажер актуален для заказчика;

$к_п = 1$, т.к. реализация не требует научных технологий;

$к_с = 2.3$, т.к. для работы с диалоговым тренажером необходимы базовые навыки работы с компьютером и интернетом;

$к_м = 1$, т.к. диалоговый тренажер использует сеть Интернет;

$к_о = 1{,}7$, т.к. диалоговый тренажер широко используется в компании;

$к_ш = 1$, т.к. диалоговый тренажер не будет охватываться охранными мероприятиями.

Из данной таблицы видно, что разрабатываемый вариант имеет более высокий показатель эксплуатационно-технического уровня по сравнению с базовым вариантом.

Коэффициент эксплуатационно-технического уровня $к_{эту}$ рассчитывается по формуле

$$к_{эту} = \frac{ЗТР_{пр}}{ЗТР_{баз}}, \quad (10)$$

где $ЗТР_{пр}$ и $ЗТР_{баз}$ – это значимость технического решения для разрабатываемого и базового вариантов соответственно.

$$к_{эту} = \frac{4{,}69}{2} = 2{,}345$$

$к_{эту} > 1$, следовательно разработка проекта является оправданной с технической точки зрения.

Кроме того, для большей уверенности в обоснованности автоматизации можно использовать обобщающий индекс эксплуатационно-технического уровня $I_{эту}$ (комплексный показатель качества проекта по группе показателей),

который рассчитывается по формуле

$$I_{\text{эту}} = \sum b_i X_i, \tag{11}$$

где b_i – коэффициент весомости i-ого показателя;

X_i – относительный показатель качества, устанавливаемый экспертным путем по выбранной шкале оценивания.

Для оценки $I_{\text{эту}}$ рекомендуется пятибалльная шкала оценивания.

В таблице 3.9 представлены результаты расчета балльно-индексным методом.

Таблица 3.9 – Расчет балльно-индексным методом

Показатель качества	**Весовой коэффициент, b_i**	**Оценка, X_i**	
		Разрабатываемый проект	**Базовый проект**
Удобство работы (пользовательский)	0,3	4	2
Надежность (защита данных)	0,1	2	1
Функциональные возможности	0,3	3	1
Временная экономичность	0,2	2	1
Время обучения персонала	0,1	2	2
Комплексный показатель качества $I_{\text{эту}}$		2,9	1,4

Коэффициент технического уровня рассчитывается по формуле

$$\text{к}_\text{т} = \frac{I_{\text{этупр}}}{I_{\text{этубаз}}}, \tag{12}$$

где $I_{\text{этупр}}$ и $I_{\text{этубаз}}$ – это комплексный показатель качества для разрабатываемого и базового вариантов соответственно.

$$к_т = \frac{2,9}{1,4} = 2,07.$$

Для расчета экономического эффекта необходимо рассчитать приведенные затраты на единицу работ, выполняемых по базовому и разрабатываемому вариантам:

$$З_i = С_i + Е_н * З_{ппi}, \quad (13)$$

где $С_i$ – текущие эксплуатационные затраты i-ого вида работ;

$Е_н = 0,33$ – нормативный коэффициент экономической эффективности;

$З_{ппi}$ – суммарные затраты, связанные с внедрением проекта.

Для базового варианта текущие эксплуатационные затраты будут состоять из заработной платы за год сотрудника, который в данный момент проводит обучение менеджеров. Средний оклад такого специалиста равен 50 000 рублей. С учетом северного и районного коэффициентов, а также отчислений во внебюджетные фонды ЗП будет равняться: (50 000 * 1,6) * 1,302 = 104 160 рублей в месяц. Исходя из 23 рабочих дней и 8-часового рабочего дня зарплата в час будет равняться 104 160 / 23 / 8 = 566 руб./час. В среднем специалист тратит около 15% своего рабочего времени в год на деятельность, связанную с обучением, что составляет примерно 37 рабочих дней или 296 часов (из расчета 247 рабочих дней в году и 8-часового рабочего дня). Таким образом эксплуатационные затраты базового варианта:

$$С_{баз} = 566 * 296 = 167\,546 \text{ рублей},$$
$$З_{баз} = 167\,546 + 0,33 * 0 = 167\,546 \text{ рублей}.$$

При помощи диалогового тренажера, данный сотрудник будет тратить

меньше времени – около 8% в год, что составляет 20 дней или 160 часов. Таким образом, для проектного варианта эксплуатационные затраты будут равняться:

$С_{пр}$ =566 * 160 = 90 560 рублей.

$З_{пр} = 90\ 650 + 0{,}33 * 44\ 488 = 105\ 241$ рубль.

Экономический эффект от использования разрабатываемой системы определяется по формуле

$$Э = \left(З_{баз} * к_т - З_{пр}\right) * V, \quad (14)$$

где $З_{баз}$ и $З_{пр}$ – приведенные затраты на единицу работ, выполняемых с помощью базового и проектируемого вариантов процесса обработки информации;

$к_т$ – коэффициент эксплуатационно-технической эквивалентности (формула 11);

V – объем работ, выполняемых с помощью разрабатываемого проекта, натуральные единицы.

Экономический эффект от использования разрабатываемой системы:

$Э = (167\ 546 * 2{,}07 - 105\ 241) * 1 = 241\ 579$ рублей.

Также по формуле необходимо рассчитать срок окупаемости затрат на разработку проекта по формуле

$$Т_{ок} = \frac{З_{пп}}{Э}, \quad (15)$$

где $З_{пп}$ – единовременные затраты на разработку проекта;

Э – годовая эффективность.

Расчет срок окупаемости затрат на разработку проекта:

$$\mathrm{T_{ок}} = \frac{167\ 546}{241\ 579} = 0{,}6 \text{ года} = 8 \text{ месяцев.}$$

Таким образом, срок окупаемости проекта составляет около 8 месяцев.

Фактический коэффициент экономической эффективности разработки:

$$\mathrm{E_ф} = \frac{1}{\mathrm{T_{ок}}} \quad (16)$$

Нормативное значение коэффициента эффективности капитальных вложений $\mathrm{E_н} = 0{,}33$ если $\mathrm{E_ф} > \mathrm{E_н}$, то делается вывод об эффективности капитальных вложений.

$$\mathrm{E_ф} = \frac{1}{0{,}79} = 1{,}66$$

Так как $\mathrm{E_ф} > \mathrm{E_н}$, то разработка и внедрение разрабатываемого продукта являются эффективными, т.е. эффект от использования данной системы окупает все затраты, связанные с проектированием и эксплуатацией. В таблице 3.10 приведены сводные данные экономического обоснования разработки и внедрения проекта.

Таблица 3.10 – Сводные данные экономического обоснования

Показатель	**Величина**
Затраты на разработку проекта, руб.	48 937
Общие эксплуатационные затраты, руб.	3 848
Экономический эффект, руб.	241 579
Коэффициент экономической эффективности	1,66
Срок окупаемости, мес.	8

Вывод по разделу «Оценка экономической эффективности разработки и внедрения диалогового тренажера»

Таким образом, был проведен экономический анализ разработки диалогового тренажера. В результате выяснилось, что капитальные затраты составят 44 488 рублей, эксплуатационные 3 848 рублей, совокупная стоимость владения 48 336 рублей. Срок окупаемости разработки будет составлять около 8 месяцев.

ЗАКЛЮЧЕНИЕ

В рамках бакалаврской работы был разработан диалоговый тренажер для студии веб-разработки «SibDev». Система призвана помочь автоматизировать отрабатывание навыков у сотрудников, основной деятельностью которых является общение и коммуникация с клиентами.

Помимо этого, система привносит в компанию процесс стандартизации обучения за счет проработанных режимов. Это особенно полезно при найме новых менеджеров – они получают готовый тренажер, который является полигоном для испытаний, на котором без вреда для компании могут в приближенных к реальным условиям опробовать изученные методики.

При разработке были выполнены следующие задачи:

1. проведен анализ деятельности тренера, которая связана с обучением менеджеров по продажам, в процессе которого выяснилось, что тренер тратит много рабочего времени на обучение, особенно новых сотрудников;

2. проведено проектирование информационной системы при помощи диаграмм нотаций *IDEF0* и *IDEF3*;

3. проведен анализ программных средств разработки информационной системы и выбран оптимальный вариант: *MongoDB* в качестве базы данных, язык *Go* и фреймворк *gqlgen* в качестве серверной части, язык *TypeScript* и библиотека *React* для клиентской стороны;

4. разработана клиентская и серверная часть диалогового тренажера;

5. оценена экономическая эффективность разработки проекта, результат которой показал, что разработка экономически оправдана и срок окупаемости её составит около 8 месяцев.

Практическая значимость состоит в том, что полученные результаты могут быть применены в деятельности студии веб-разработки «SibDev» по обучению сотрудников, а также другими компаниями.

Созданный диалоговый тренажер был представлен заказчику, в частности

руководителю студии веб-разработки «SibDev». После демонстрации тренажера на контрольном примере, он был принят для проведения обучения тренером менеджеров студии.

Результаты бакалаврской работы были представлены на XVIII Международной конференции студентов, аспирантов и молодых ученых «Проспект Свободный–2022», посвященной Международному году фундаментальных наук в интересах устойчивого развития, г. Красноярск.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Профессия менеджер по продажам // Инфоподдержка вашей карьеры : [сайт]. – 2021. – URL: https://enjoy-job.ru/professions/manager-po-prodazham/ (дата обращения: 03.03.2022)

2. Менеджер ИТ-проектов: что это за профессия и почему она такая перспективная // RB.RU — новые технологии, бизнес и карьера в цифровой экономике : [сайт]. – 2020. – URL: https://rb.ru/opinion/it-projects-manager (дата обращения: 03.03.2022)

3. О профессии Менеджера по персоналу // Сибирский институт бизнеса и информационных технологий : [сайт]. – 2022. – URL: http://sano.ru/articles/o-professii-menedzhera-po-personalu.html (дата обращения: 03.03.2022)

4. Обзор рынка труда в ИТ-сфере в начале 2021 года в России и Санкт-Петербурге // HeadHunter : [сайт]. – 2021. – URL: https://hh.ru/article/28685 (дата обращения: 26.02.2022)

5. Диалоговые тренажеры. Что это такое и когда применяется. // Diskurs – электронные курсы : [сайт]. – 2017. – URL: https://kursnazakaz.ru/dialogovyye-trenajery-chto-eto/ (дата обращения: 16.12.2021)

6. Веб-приложение // Википедия – свободная энциклопедия : [сайт]. – 2001. – URL: https://ru.wikipedia.org/wiki/Веб-приложение (дата обращения: 18.12.2021)

7. Введение в React // Изучение React. Полное руководство по React : [сайт]. – 2017. – URL: https://learn-reactjs.ru/tutorial (дата обращения: 18.12.2021)

8. Бэнкс, А. GraphQL: язык запросов для современных веб-приложений / А. Бэнкс, Е. Порселло. – Санкт-Петербург : Питер, 2019. — 240 с. – ISBN 978-5-4461-1143-5.

9. Service Layer (Сервисный уровень). // Справочник «Паттерны проектирования» : [сайт]. – 2016. – URL: http://design-pattern.ru/patterns/service-layer.html / (дата обращения: 18.03.2022)

10. Данжу, Д. Путь Python. Черный пояс по разработке, масштабированию, тестированию и развертыванию. / Д. Данжу. – Санкт-Петербург : Питер, 2020. – 256 с. – ISBN 978-5-4461-1308-8.

11. JSX в деталях // React : [сайт]. – 2013. – URL: https://ru.reactjs.org/docs/jsx-in-depth.html (дата обращения: 23.03.2022)

12. Компоненты и пропсы // React : [сайт]. – 2013. – URL: https://ru.reactjs.org/docs/jsx-in-depth.html (дата обращения: 23.03.2022)

13. Краткий обзор хуков // React : [сайт]. – 2013. – URL: https://ru.reactjs.org/docs/hooks-overview.html (дата обращения: 23.03.2022)

14. Хук useEffect // Сайт о программировании : [сайт]. – 2022. – URL: https://metanit.com/web/react/6.3.php (дата обращения: 25.03.2022)

15. Совокупная стоимость владения // Википедия – свободная энциклопедия : [сайт]. – 2001. – URL: https://ru.wikipedia.org/wiki/Совокупная_стоимость_владения (дата обращения: 20.03.2022)

16. МРОТ в 2022 году по регионам России // Бухалтерский интернет-журнал : [сайт]. – 2022. – URL: https://buhguru.com/zp-i-kadri/mrot-2022-v-rossii-po-regionam.html (дата обращения: 22.03.2022)

17. Налоговый кодекс российской федерации // КонсультантПлюс : [сайт]. – 1998. – URL: https://www.consultant.ru/document/cons_doc_LAW_19671/ (дата обращения: 20.03.2022)

18. Что такое MVP, и как создать минимально жизнеспособный продукт // SectPoint – разработка ПО : [сайт]. – URL: https://stecpoint.ru/Practices-MVP/ (дата обращения: 25.03.2022)

Выпускная квалификационная работа выполнена мной самостоятельно. Использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

Отпечатано в одном экземпляре.

Библиография 18 наименований.

Один экземпляр сдан на кафедру.

«____» _____ 2022 г.

_____ Олейник Владимир Викторович
подпись

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Хакасский технический институт – филиал ФГАОУ ВО
«Сибирский федеральный университет»

Кафедра прикладной информатики, математики и естественно-научных дисциплин

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Е. Н. Скуратенко
подпись
«17» июня 2022 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.03 Прикладная информатика

Создание диалогового тренажера для студии веб-разработки «SibDev»

Руководитель _____ 17.06.2022 доцент, канд. пед. наук И. В. Янченко
подпись, дата

Выпускник _____ 17.06.2022 В. В. Олейник
подпись, дата

Консультанты
по разделам:

Экономический _____ 17.06.22 Е. Н. Скуратенко
подпись, дата

Нормоконтролер _____ 17.06.22 В. И. Кокова
подпись, дата

Абакан 2022