***Федеральное агентство по рыболовству***



***Федеральное государственное бюджетное образовательное***

***учреждение высшего образования***

***«Астраханский государственный технический университет»***

**Система менеджмента качества в области образования, воспитания, науки и инноваций сертифицирована DQS**

**по международному стандарту ISO 9001:2015**

Институт Информационных технологий и коммуникаций

Направление подготовки 09.03.04 Программная инженерия

Профиль Разработка программно-информационных систем

Кафедра Автоматизированные системы обработки информации и управления

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(название темы)

Работа выполнена обучающимся группы ДИПРб-41

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия Имя Отчество)

Руководитель ВКР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(ученая степень, ученое звание, Фамилия Имя Отчество)

Консультант по   проектной документации ст. преп. Куркурин Н.Д.   
(название раздела, ученая степень, ученое звание, Фамилия И.О., подпись)

Консультант по   информационной безопасности к.т.н., доцент Белов С.В.  
(название раздела, ученая степень, ученое звание, Фамилия И.О., подпись)

Нормоконтролер   ст. преп. Толасова В.В.  
(ученая степень, ученое звание, Фамилия И.О., подпись)

Допущена к защите «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_г.

Заведующий кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_Хоменко Т.В.

Астрахань 2021

***Федеральное агентство по рыболовству***

***Федеральное государственное бюджетное образовательное***



***учреждение высшего образования***

***«Астраханский государственный технический университет»***

**Система менеджмента качества в области образования, воспитания, науки и инноваций сертифицирована DQS**

**по международному стандарту ISO 9001:2015**

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Стратегический партнер (работодатель) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (предприятие, должность, ФИО)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_»\_\_\_\_\_\_\_\_\_\_20 \_\_\_г. | УТВЕРЖДАЮ  Заведующий кафедрой  Автоматизированные системы  обработки информации и управления  д.т.н. профессор Т.В. Хоменко  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_»\_\_\_\_\_\_\_\_\_\_20 \_\_\_г. |

**З А Д А Н И Е**

на выполнение выпускной квалификационной работы

Обучающемуся   
учебной группы ДИПРб-41 института Информационных технологий и коммуникаций

(институт/факультет)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество - полностью)

**ТЕМА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Тема ВКР сформулирована в соответствии с запросом \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(стратегический партнер, работодатель, подразделение (службы) АГТУ и т.д.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Утверждена распоряжением директора института Информационных технологий и коммуникаций (распоряжение от « 07 » декабря 2020 г. № 49)

**РУКОВОДИТЕЛЬ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество – полностью, ученая степень, ученое звание)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Утвержден распоряжением директора института Информационных технологий и коммуникаций (распоряжение от « 07 » декабря 2020 г. № 49)

Представление выпускной

квалификационной работы на кафедру «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

Дата защиты «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г.Целевая установка и исходные данные:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| №  п/п | Перечень чертежей, подлежащих разработке | Формат, количество |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Руководитель выпускной квалификационной работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

|  |  |  |
| --- | --- | --- |
| №  п/п | Содержание расчетно-пояснительной записки  (перечень вопросов, подлежащих разработке) | Консультанты |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Руководитель выпускной квалификационной работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**Основная рекомендуемая литература**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Задание принял к исполнению «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_г.

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

|  |  |
| --- | --- |
| **УТВЕРЖДАЮ** | К заданию на выпускную квалификационную работу |
| Заведующий кафедрой  д.т.н., профессор Т.В. Хоменко  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_»\_\_\_\_\_\_\_\_\_\_20 \_\_\_г. |  |

**КАЛЕНДАРНЫЙГРАФИК**

подготовки выпускной квалификационной работы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| №  п/п | Разделы, темы и их содержание, графический материал | По плану | | По факту | | Отметка руководителя о выполнении |
| Дата | Объем  в % | Дата | Объем  в % |
| 1 | Составление и согласование технического задания на выпускную квалификационную работу | 10.12.2020 | 3 |  |  |  |
| 2 | Обследование предметной области | 01.03.2021 | 20 |  |  |  |
| 3 | Разработка технического проекта | 25.04.2021 | 30 |  |  |  |
| 4 | Получение задания на преддипломную практику | 30.04.2021 | 32 |  |  |  |
| 5 | Защита отчёта по преддипломной практике | 31.05.2021 | 60 |  |  |  |
| 6 | Подготовка к защите ВКР | 17.06.2021 –28.06.2021 | 100 |  |  |  |
| 7 | Предварительная защита ВКР | 14.06.2021–18.06.2021 | 100 |  |  |  |
| 8 | Защита ВКР | 28.06.2021 | 100 |  |  |  |

Руководитель выпускной квалификационной работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Обучающийся: \_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_г

**РЕФЕРАТ**

с. 65, рис. 15, табл. 24, прил. 2

**инновационный проект, малое предприятие, термодиструкция древесины, установка с чпу, автоматизация проектирования, сувениры, реклама, бизнес-план, project expert, sadt.**

Разработан проект малого предприятия по изготовлению сувениров с использованием технологии термодиструкции древесины на установке с ЧПУ при автоматизированном проектировании управляющих программ. Проведен анализ рынка сувениров и установлена ниша продукции предприятия - малые серии изделий по заказам к праздничным датам и с логотипом фирмы.

Определен состав оборудования предприятия и его персонал.

Поработана система мероприятий по охране труда сотрудников.

Проведен экономический анализ в форме бизнес-плана, обеспечивающего 2-х летний срок окупаемости капитальных вложений. Предложен состав мероприятий по рекламе продукции предприятия. Намечен план расширения предприятия за счет освоения новых типов продукции экспортного исполнения.

Управление проектом реализовано с помощью программных пакетов MS Project и технологии SADT.

**СОДЕРЖАНИЕ**

[Введение 7](#_Toc482787221)

[1 Технический проект 8](#_Toc482787222)

[1.1 Анализ предметной области 8](#_Toc482787223)

[1.2 Технология обработки информации 8](#_Toc482787224)

ВВЕДЕНИЕ

Кафедра «АСОИУ» астраханского государственного технического университета производит подготовку студентов IT-специальностей. Абитуриенты, ежегодно поступающие в университет, обладают разным уровнем подготовки к моменту началу обучения. Задача первого года обучения выровнять знания учеников. Для достижения данной цели используется среда Semantic IDE, которая обучает студентов правилам хорошего форматирования кода, позволяет плавно войти в программу обучения и подготовиться к более сложным языкам программирования.

Данный программный комплекс не позволяет решить одну из главных проблем современного процесса обучения программированию – обучение правильному выстраиванию логики поведения программы. Молодые программисты часто совершают логические ошибки, которые нарушают правильное поведение системы. На выходе получается продукт без синтаксических ошибок, но с практической пользой равной нулю.

Мышление человека работает с помощью представления образов. Графическую информацию мозг воспринимает в несколько раз быстрее текстовой, что позволяет потребить и проанализировать больше информации за единицу времени. Соответственно возникает резонный вопрос, а что произойдёт, если заменить текстовое программирование визуальным? Попыткой ответить на этот вопрос стало представление языка ДРАКОН – Дружелюбного Русского Алгоритмического Языка, Который Обеспечивает Наглядность. Суть его в представлении программ в виде блок-схем с определенной дисциплиной построений, в которых вся функциональная часть спрятана «под капотом», внутри функциональных блоков. Снаружи представлено лишь описание процессов формальным языком. Это позволяет существенно упростить процесс понимания алгоритма работы того или иного процесса.

Программно-аппаратный комплекс, позволяющий разрабатывать программные продукты с помощью языка ДРАКОН, позволит студентам правильно подходить к проектированию алгоритмов, что сократит время, затрачиваемое на разработку и тестирование.

Конвертация в высокоуровневые языки программирования позволит существенно популяризировать данное решение из-за возрастания практической пользы программно-аппаратного комплекса и избавит пользователей системы от необходимости переносить алгоритм на другой язык программирования вручную.

Существует ряд редакторов, в том числе WEB, такие, как «Dragon.tech», «ИС ДРАКОН», «DRAKON Editor». Все они имеют как преимущества, так и недостатки. Ни один из вышеперечисленных не обладает одновременно следующими важными качествами:

* платформонезавимость. Программный продукт должен быть кроссплатформенным;
* нетребовательность к ресурсам. Программный продукт должен рационально использовать ресурсу компьютера;
* конвертация в язык программирования высокого уровня. редакторы с открытым исходным кодом не обладает таким качеством.

Разработка и внедрение подобной информационной системы с возможностью создавать ДРАКОН-схемы, с последующей конвертацией в высокоуровневый язык программирования, на кафедру «АСОИУ» позволит существенно снизить нагрузку на преподавателя путём автоматизации процесса обучения студентов основам алгоритмизации. Снизит время, затрачиваемое на проектирование, и положительно скажется на поиске ошибок в логике поведения разрабатываемых программ.

Целью данной преддипломной практической работы является проектирование клиент-серверной информационной системы, которая осуществляет хранение в базе данных пользовательских ДРАКОН-схем, распределяет роли между пользователями системы с целью автоматизации процесса обучения и отображает графическое представление ДРАКОН-схемы.

Назначение: обучение пользователей основам алгоритмизации и автоматизации процессов, снижение нагрузки на преподавателей на кафедре «АСОИУ», повышение практической пользы языка ДРАКОН как инструментария для автоматизации процессов.

1 ТЕХНИЧЕСКИЙ ПРОЕКТ

* 1. Анализ предметной области
     1. Язык ДРАКОН

Язык ДРАКОН – дружелюбный российский алгоритмический язык, который обеспечивает наглядность. Дракон схема – диаграмма, состоящая из специальных икон и соединений между ними, которые формируются по определенным правилам, которые, в совокупности, формируют конечный алгоритм.

В первую очередь, язык ДРАКОН ориентирован на лиц, не владеющих языком программирования, но нуждающихся в программном обеспечении для удовлетворения задач в их предметной области.

Основа языка ДРАКОН – иконы и макроиконы. Иконы – минимальная единица в алгоритме, представляет какое-либо действие, событие. Объединяясь в группы иконы формируют макроиконы. Иконы также часто называют операторами, далее допускаются оба варианта наименования.

Любая дракон-схема имеет начало и конец. Они представлены в виде специальных икон, которые отличаются от всех остальных. Икона «Заголовок», изображенная на рисунке 1.1, указывает на начало алгоритма, и содержит в себе его название.



Рисунок 1.1 – Икона «Заголовок»

Икона «Конец», показанная на рисунке 1.2, обозначает точку завершения алгоритма. Иконы «Заголовок» и «Конец» не могут существовать в разрыве друг от друга и представляют собой неделимую основу для будущей последовательности действий.



Рисунок 1.2 – Икона «Конец»

Икона «Действие» содержит в себе некую операцию, проводимую над доступными данными, которая проводится безусловно. Пример представления такой иконы показан на рисунке 1.3.



Рисунок 1.3 – Икона «Действие»

Икона «Вопрос» представляет из себя условную конструкции, в которой есть два варианта исхода: положительный и отрицательный, в зависимости от результата проверки условия, хранящегося в иконе «Вопрос». Результаты проверки условия можно сопоставить с ответами на вопрос «Да» и «Нет». В языке ДРАКОН такая икона обозначается шестиугольником, пример показан на рисунке 1.4.



Рисунок 1.4 – Икона «Вопрос»

Для иконы «Вопрос» важную роль играет направление альтернативной ветки. Если она указывает на ту часть алгоритма, которая находится ниже иконы с условием, то для алгоритма она является обычным условным оператором. А если указывает на часть алгоритма выше её, то икона «Вопрос» представляет из себя цикл.

Для графического представления этих отличий для циклов указывается стрелка - направление, пример представлен на рисунке 1.5. Такие конструкции называются макроиконами, так-как образованы объединением икон в группы для решения одной конкретной задачи.



Рисунок 1.5 – Макроиконы «Вопрос» и «Цикл»

Другой важной иконой является «Вариант» её представление показано на рисунке 1.6. Она хранит схожее с иконой «Вопрос» условие, однако решение принимается исходя не из положительности проверки условия, а из точного соответствия конкретному значению.



Рисунок 1.6 – Икона «Вариант»

Каждое такое значение должно быть учтено в системе, для этого в языке ДРАКОН есть специальная икона «Выбор», в которой описывается точное состояние значения для иконы «Вариант». Графическое представление этой иконы показано на рисунке 1.7.

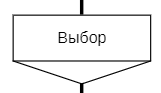


Рисунок 1.7 – Икона «Выбор»

Икон «Выбор» может быть несколько, в зависимости от потребностей разработчика алгоритма, ограничений на их количество условно нет.

Каждый «Вариант» формирует макроикону «Ветку» выполнения, каждая ветка выполняется независимо друг от друга и не может выполняться одновременно с другой. Таким образом разработчик может описывать последовательность действий для каждого состояния системы. Графическое представление макроиконы «Ветка» показано на рисунке 1.8.

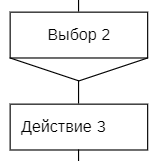


Рисунок 1.8 – Макроикона «Ветка»

Все макроиконы «Ветки», содержащие иконы «Выбор», вместе с привязанной к ним иконой «Вариант» образуют макроикону «Варианта». Конец всех «Веток» соединяется в единый поток, и управление переход следующей иконе. Графическое представление макроиконы «Вариант» показано на рисунке 1.9.

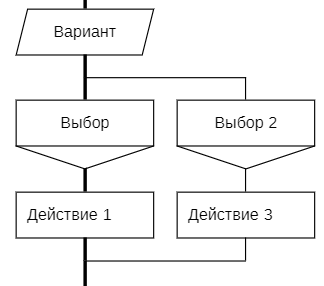


Рисунок 1.9 – Макроикона «Вариант»

Икона «Ввод» отвечает за получение входных данных от устройств ввода. Икона «Вывод», соответственно, за вывод информации в консоль или на экран, в зависимости от реализации системы. Обе иконы представлены на рисунке 1.10.



Рисунок 1.10 – Иконы «Вывод» и «Ввод»

Икона «Комментарий» не выполняет функциональной роли, только благоприятствует пониманию алгоритма. В ней пользователь описывает состояние системы, данных в ней или любую иную произвольную информацию, которую пользователь считает необходимой. Вид иконы «Комментарий» представлен на рисунке 1.11.



Рисунок 1.11 – Икона «Комментарий»

Икона «Пауза», как следует из её названия, отвечает за установку задержки перед выполнением следующей инструкции. Как правило задержку указывают в миллисекундах. Вид данной иконы представлен на рисунке 1.12.

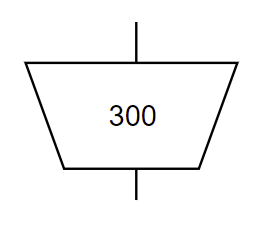


Рисунок 1.12 – Икона «Пауза»

Особой иконой является «Имя ветки». Она содержит произвольное количество икон внутри себя, которые описывают весь или часть какого-либо алгоритма. В некоторых случаях, например в макроиконе «Вопрос» и «Вариант» ветки не обозначаются графически, однако содержатся в каждом «Выборе». Но в отдельных случаях, которые описаны в пункте 1.1.2, у иконы «Ветка» есть графическое представление, оно также представлено на рисунке 1.13. Внутри данная икона содержит формальное описание содержимого в ней или просто название алгоритма внутри себя. Важно, чтобы это название было уникальным для недопущения разночтения.



Рисунок 1.13 – Икона «Имя ветки»

Каждая икона «Имя ветки» явно или неявно сопровождается иконой «Адрес», которая содержит название ветки, в который переходит процесс управления. Явно данная икона отображается в случае, если указывает на другую ветку, в таком случае она отображается перевернутой иконой «Имя ветки», её представление показано на рисунке 1.14.



Рисунок 1.14 – Икона «Адрес»

Если «Адрес» указывает на «Конец» ДРАКОН-схемы, то в таком случае икона явно не отображается. Такой способ представления ветки показан на рисунке 1.15.

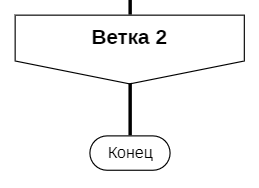


Рисунок 1.15 – Неявная икона «Адрес» в ветке

В уникальном представлении каждой иконы и заключается принцип наглядности: каждый тип иконы отличается от другой. На вопрос об отличии дракон-схемы от обычного графического представления алгоритма, однозначно можно ответить следующим образом: язык ДРАКОН дополняет стандартное графическое представление алгоритмов и устанавливает строгие правила взаимодействия икон друг с другом, что позволяет упростить написание программ, улучшить наглядность алгоритмов, а также даёт возможность реализовать транслятор удовлетворимой сложности для представления дракон-схемы в виде алгоритма на языке программирования высокого уровня.

* + 1. Правила построения ДРАКОН-схем

Самым простом способом представления ДРАКОН-схемы является «Примитив» - простая конструкция языка ДРАКОН, которая позволяет описать алгоритм в виде последовательности икон. Пример «Примитива» представлен на рисунке 1.13.

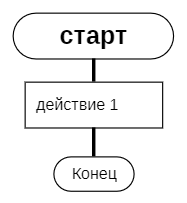


Рисунок 1.13 – Пример «Примитива»

Каждая икона в «Примитиве» следует одна за другой, формируя линию. Макроиконы, такие как «Вопрос» или «Вариант», могут формировать ответвления влево, однако все они соединяются с главной вертикалью в конце своих веток.

Вторым способом представления ДРАКОН-схем является конструкция «Силуэт». Её графическое представление показано на рисунке 1.14. Такая конструкция представляет из себя набор «Примитивов», каждый из которых содержит ссылку на следующий для выполнения набор инструкций, или на конец алгоритма. Каждый «Примитив» в «Силуэте» выполняется слева направо, если одна из веток выполнения ссылается на другую, то её необходимо размещать левее той, управление к которой перейдёт после.

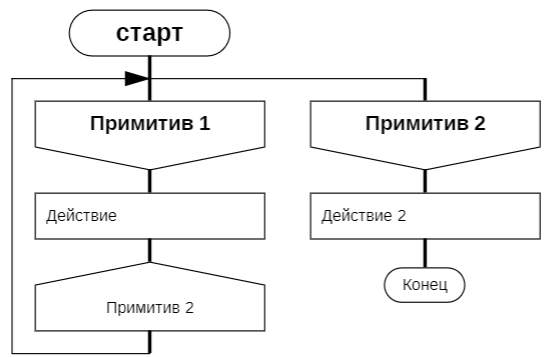


Рисунок 1.14 – Пример «Силуэта»

С точки зрения поведения «Силуэт» ничем не отличается от «Примитива». Главная задача силуэта упростить представление последовательного алгоритма. Каждый «Примитив» можно преобразовать к «Силуэту», справедливо и обратное утверждение.

Одним из главных правил построения схем в языке ДРАКОН является «Шампур». Оно гласит, что между иконами «Начало» и «Конец» может содержаться сколь угодно других икон, однако все они формируют прямую линию.

Данное правило должно реализовываться программно. Последовательность икон визуально должна ощущаться таковой, для этого их необходимо размещать в одной вертикальной плоскости.

Чтобы исключить возможность для пользователя нарушать основные принципы построения ДРАКОН-схем.

Графическое описание данного правила представлено на рисунке 1.15.

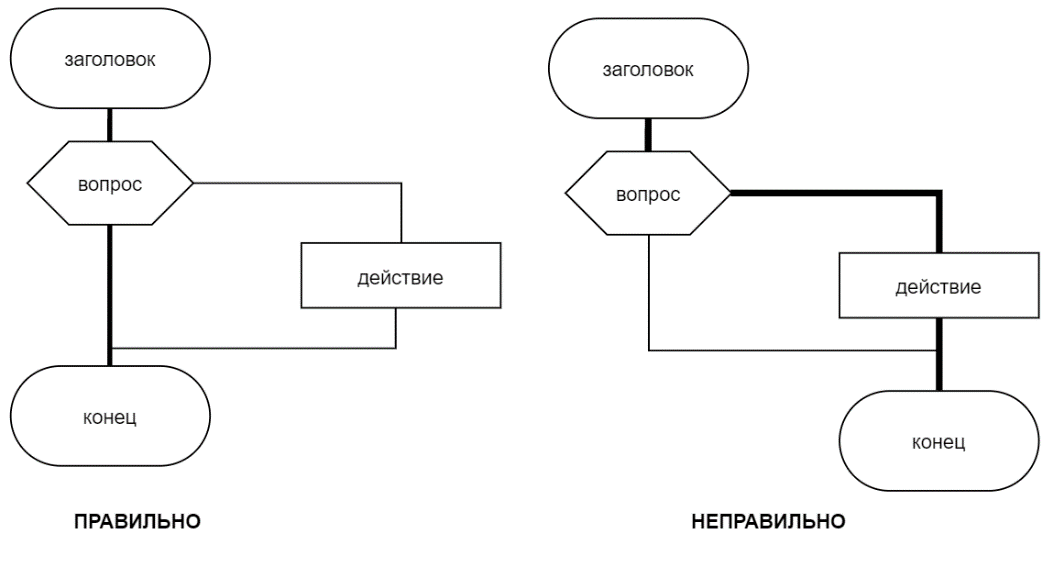


Рисунок 1.15 – Графическое представление правила «Шампура»

Второе важное правило, которое вытекает из первого – главный маршрут алгоритма всегда должен идти по «Шампуру». Применение данного правила на практике представлено на рисунке 1.16.

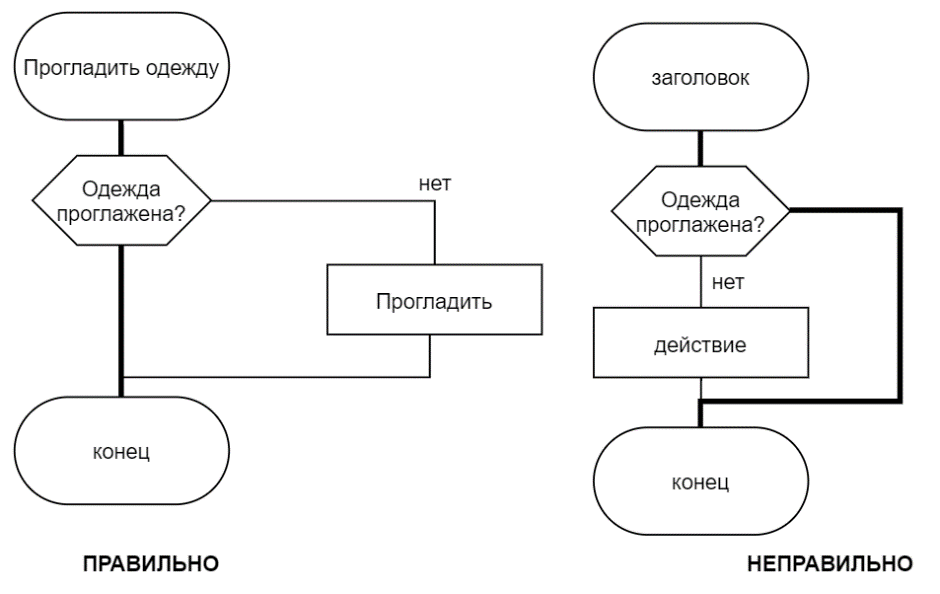


Рисунок 1.16 – Графическое представление правила: главный маршрут по вертикали

Под маршрутом, в данном случае, подразумевается путь, который отражает наиболее позитивный исход, достижение результата по наиболее благоприятному пути. Это правило регулируется привязкой координат начала и конца ДРАКОН схемы. Но построение правильной бизнес-логикой остаётся за пользователем.

Третье правило: побочные маршруты необходимо рисовать всегда справа от основного. Чем более неблагоприятное развитие события, тем правее оно описано. Это необходимо для улучшения читаемости алгоритма. Также его можно описать другими словами: «Чем правее, тем хуже». Пример реализации такого правила представлен на рисунке 1.17.

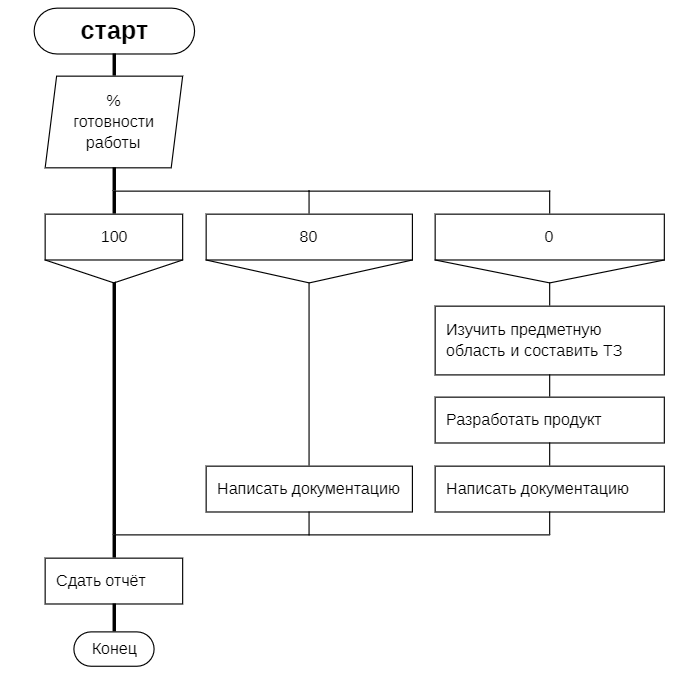


Рисунок 1.17 – Графическое представление правила: побочные маршруты справа

Программно данное правило реализуется тем, что альтернативные ветки для икон вопроса отрисовываются справа от основного маршрута. Но это не означает, что этого правила необходимо строго придерживаться. Иногда оно принципиально нереализуемо из-за совершенно иной логики программы, в котором отсутствует понятие «хуже». Тогда следует изменить логику работы программы, чтобы она удовлетворяла иному условию, по выбору пользователя.

Самое главное правило дракон-схем - пересечение линий запрещены. Любые две иконы должны быть соединены таким образом, чтобы не пересекать любые другие, соединяющие иконки. Для этого существует целый ряд правил для отрисовки элементов дракон-схем, которые хоть и не всегда способны защитить пользователя от нарушения этого правила, но свести такие ситуации к минимуму.

Стоит отметить, что пересечение линий всегда можно преобразовать к виду, в котором они пересекаться не будут.

* 1. Технология обработки информации

Разрабатываемая система будет представлять собой многопользовательское онлайн приложение. В системе необходимо реализовать идентификацию пользователя, сбор, хранение и предоставление ДРАКОН-схем. В информационной системе необходимо выделить следующие роли:

* пользователь – получающий доступ к своим схемам с целью разработки алгоритмов на языке ДРАКОН;
* куратор – получающий доступ к схемам курируемых пользователей с целью осуществления контроля;
* администратор – осуществляющий контроль над учётными записями пользователей и их ролями.

Диаграмма вариантов использования представлена на рисунке 1.18.

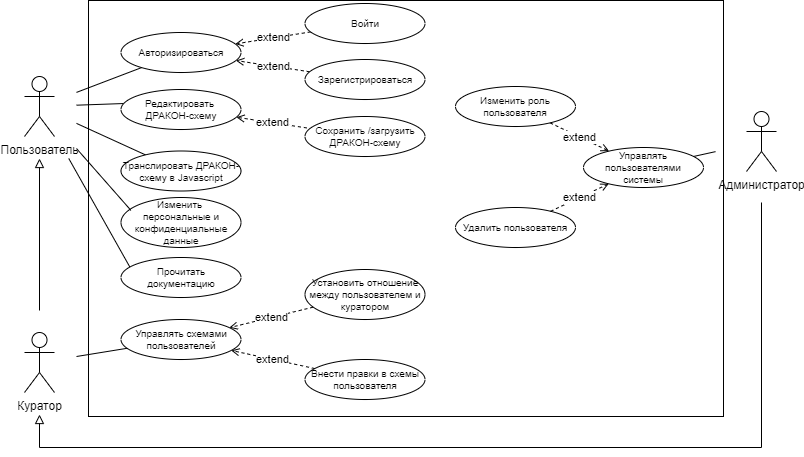


Рисунок 1.18 – Диаграмма вариантов использования

Основным актором в системе является пользователь. Пользователю разрешено управление собственными схемами, своими персональными данные, а также предоставляется возможность осуществлять конвертацию ДРАКОН-схем в JavaScript. Для идентификации используется механизм регистрации, для которой необходимо предоставить уникальную комбинацию символов для имени пользователя и пароль, который предоставляет защиту данных пользователя от доступа третьих лиц.

Куратор обладает аналогичными с пользователем возможностями по взаимодействию с системой. Куратор может добавлять существующих пользователей к себе в группу, что даёт возможность осуществлять просмотр и редактирование схем других пользователей.

Администратор обладает полным спектром возможностей по взаимодействию с пользователями и схемами, однако основная его задача: контроль над работой системы: администратор выдаёт роли зарегистрированным пользователем, удаляет аккаунты пользователей, которые не удовлетворяют пользовательским требованиям, например: некорректное имя пользователя, или учётная запись больше не является активной.

* + 1. Спецификация вариантов использования

Спецификация вариантов использования пользователя:

1. Зарегистрироваться.
   1. Краткое описание: пользователь в интерфейсе аутентификации вводит уникальное имя пользователя в системе, придумывает пароль, и нажимает кнопку «Регистрация».
   2. Ответ системы: система производит валидацию имени пользователя на уникальность и соответствие пароля минимальным требованиям, в случае успеха регистрирует пользователя в системе и позволяет войти под своими учётными данными.
2. Войти.
   1. Краткое описание: пользователь, ранее зарегистрированный в системе, вводит валидную комбинация имени пользователя и пароля и осуществляет вход через нажатие кнопки «Вход» интерфейса аутентификации системы.
   2. Ответ системы: система производит валидацию пользователя и выдаёт JWT-token для предоставления сессии пользователю и доступа к редактору ДРАКОН-схем.
3. Редактировать ДРАКОН-схему.
   1. Краткое описание: пользователь создаёт, удаляет или изменяет содержимое схемы или её название через интерфейс редактора ДРАКОН-схем.
   2. Ответ системы: система регистрирует изменения, произведенные пользователем, и обновляет данные в базе данных редактируемой схемы. В случае удаления схемы операция осуществляется без возможности восстановления.
4. Транслировать ДРАКОН-схему в JavaScript.
   1. Краткое описание: пользователь, предварительно открыв схему, через интерфейс системы выбирает пункт «В JavaScript» или «Сохранить код».
   2. Ответ системы: в случае выбора пользователем пункта «Сохранить код» система осуществляет трансляцию текущей открытой схемы в язык JavaScript. После чего генерирует файл на стороне клиента, и используя браузер, осуществляет сохранение на рабочей станции клиента файла с оттранслированной программой в формате «.js». В случае выбора пункта «В JavaScript» системы выводит оттранслированную схему в специальном компоненте интерфейса редактора.
5. Изменить персональные данные.
   1. Краткое описание: оператор выбирает пункт «Изменить имя пользователя» или «Изменить пароль» и вводит обновленные данные для своей учётной записи. Или выбирает пункт «Удалить аккаунт».
   2. Ответ системы: система регистрирует изменения в базе данных после проверки на корректность входных данных. В противном случае уведомляет об ошибке.
6. Прочитать документацию.
   1. Краткое описание: оператор выбирает пункт в меню интерфейса «Документация».
   2. Ответ системы: система отображает основную информацию об иконах и макроиконах системы вместе с иллюстрациями и способами трансляции в язык JavaScript.

Спецификация вариантов использования куратора:

1. Установить отношение между пользователем и куратором.
   1. Краткое описание: куратор в интерфейсе администрирования выбирает из списка или вводит имя пользователя. Для выбранного пользователя ему доступны следующие действия «Добавить/Удалить в список курируемых пользователей» и «Предоставить описание отношению между куратором и пользователем».
   2. Ответ системы: Система добавляет или исключает пользователя из списка курируемых для текущего куратора. Изменяет описание отношения в базе данных в ответ на ввод куратора.
2. Внести правки в схемы пользователя.
   1. Краткое описание: куратор в интерфейсе редактора в специальном выпадающем списке вводит имя курируемого пользователя, выбирает доступную схему из списка. После производит изменения в схеме, чтобы курируемый мог учесть правки.
   2. Ответ системы: система загружает схемы выбранного пользователя при наличии достаточных привилегий со стороны куратора. Регистрирует и сохраняет изменения схема в базе данных, включая информацию о времени и кураторе, который произвёл изменения.

Спецификация вариантов использования администратора:

1. Изменить роль пользователя.
   1. Краткое описание: администратор в интерфейсе администрирования выбирает из списка или вводит имя пользователя. Для выбранного пользователя «Администратор» выбирает желаемую роль и пункт «Установить роль»
   2. Ответ системы: Система проверяет привилегии пользователя, который производит изменение роли. В случае удовлетворительной проверки осуществляет обновление роли пользователя в базе данных.
2. Удалить пользователя
   1. Краткое описание: администратор в интерфейсе администрирования выбирает из списка или вводит имя пользователя. Для выбранного пользователя «Администратор» выбирает пункт «Удалить пользователя».
   2. Ответ системы: система проверяет привилегии пользователя. В случае удовлетворительной проверки осуществляет удаление пользователя из базы данных, всех его схем и отношений.
      1. Защита персональных данных

Одной из основных задач при проектирование многопользовательской информационной системы является обеспечение безопасности конфиденциальных данных.

Система осуществляет сбор и хранение логинов пользователей, их ДРАКОН-схем, а также почты. Все эти данные необходимо защитить от неавторизованного доступа и злоумышленников.

Каждому пользователю при регистрации необходимо указать пароль. На него накладывается ряд ограничений:

* Длина пароля не менее 6 символов;
* Пароль может состоять только из латинских символов, а также знака «\_» и цифр;
* Пароль обязательно должен содержать одну заглавную букву или цифру;
* Не допускаются пароли, состоящие полностью из прописных или заглавных символов.

Система не осуществляет хранение паролей в «чистом» виде. Предварительно все пароли шифруются с помощью функции bcrypt.

Bcrypt является односторонней адаптивной функцией криптошифрования ключей безопасности переменной длинны. Эффективность алгоритма обуславливается тем, что время работы функции можно изменять с помощью коэффициента «salt», а сам ключ при каждой операции шифрования для одинаковой последовательности символов отличается. Для проверки пароля на корректность необходимо осуществить шифрование с той же «солью», комбинацией символов пароля и сравнить с изначально зашифрованной последовательностью.

Пароль пользователя при регистрации отправляется на сервер, шифруется через функцию bcrypt и сохраняется в базе данных в связке с уникальным именем пользователя. В последующем обратно дешифровать ключ невозможно.

Для аутентификации пользователя используется схема Http Basic (RFC 7617). По которой для входа в систему пользователю необходимо отправить связку из логина и пароля.

Если система находит пользователя в базе данных, и пароли совпадают, то пользователю возвращается специальный JWT Access токен для доступа к защищенным путям приложения.

Авторизация осуществляется по схеме Http Bearer: в случае успешной авторизации пользователю выдаётся JWT токен, в котором зашифрованы его логин и роль в системе.

Все URL-пути приложения, которые осуществляют выдачу, обновление или удаление конфиденциальных данных защищены. Для выполнения любого запроса должен выполняться уровень привилегий пользователя.

Пользователь системы не может взаимодействовать со схемами других пользователей, изменять и получать конфиденциальные данные других пользователей.

Куратор обладает доступом к схемам только курируемых пользователей, не может изменять данные других пользователей.

Администратор получает полный доступ к схемам других пользователей, может удалять и изменять роль других учётных записей, но не имеет доступа к почте и паролю других пользователей системы.

* + 1. Способы взаимодействия с ДРАКОН-схемой

Минимальной схемой в примитиве является блок «Заголовок-Конец». Пользователь не имеет возможности удаления этого блока. Графически связь представляется отрезком между двумя иконами, как представлено на рисунке 1.19.



Рисунок 1.19 – ДРАКОН-схема «Примитив»

Все последующие иконы вставляются между ними. Количество их неограниченно. Местом для вставки является центр линии между двумя иконами. Область, в которую можно вставить икону подсвечивается. Удаление происходит с помощью выбора соответствующего пункта и нажатием на саму икону, удаляемые иконы также подсвечиваются.

Графически отображаемая икона ограничена размером, и не все данные можно уместить в данном блоке. Одним из вариантов решения этой проблемы – увеличение размера самой иконы, но тут существует проблема снижения читаемости схемы. Одним из оптимальных способов решения данной проблемы – представление полного содержимого иконы в отдельном элементе интерфейса.

В теле самой иконы описываются действия, производимые в иконе, для описания комментария необходимо воспользоваться специальной иконой «Комментарий».

* + 1. Методы представления ДРАКОН икон

Для проектируемой информационной системы необходимо реализовать представление ДРАКОН-схем в памяти программы. При этом контейнер должен обладать рядом качеств: предоставлять произвольный доступ к иконам, при этом обеспечивать последовательный доступ к иконам ДРАКОН-схемы для просмотра схемы от начала до конца. Учитывая наложенные ограничения, были разработаны классы, представленные на диаграмме классов в приложении 1.

Базовый абстрактный класс иконы устанавливает правила для всех наследуемых от него икон. Все иконы содержат информацию о своём типе, данные, которые они обязаны хранить в соответствии со своим типом и шаблон поведения на получение внешних данных. Помимо этого, каждая икона хранит информацию о своём родителе – макроиконе её содержащую. Все иконы внутри схемы имеют родителя.

Для проектируемого представления ДРАКОН-схем были разработаны следующие иконы:

* заголовок;
* конец;
* примитив;
* имя ветки;
* адрес;
* силуэт;
* действие;
* вопрос;
* ветка;
* цикл;
* вариант;
* выбор;
* разделитель;
* комментарий;
* пауза;
* ввод;
* вывод.

Каждая икона, наследуется от базового класса иконы, получая базовые поля для всех типов икон и методы взаимодействия друг с другом. Макроиконы содержат собственные дополнительные поля и правила поведения, которые используются с целью установки связи между другими иконами.

Каждая икона хранит массив вложенных в неё операторов, операторами могут служить как простые, так и макроиконы. Детали содержимого вложенных икон родительской неизвестны.

Вложенность икон удобно рассматривать как реализацию блочной видимости. Каждая родительская икона знает о своих вложенных операторах, в свою очередь о самой макроиконе знает информацию та икона, которая содержит её. Чтобы получить доступ к любой из икон, которая содержится внутри другой, необходимо или пройти все уровни вложенности, или с помощью ассоциативного доступа по уникальному идентификатору иконы. Таким образом обеспечивается оптимальное хранение и представление схем. Любой простой оператор содержит пустой массив: он не может содержать вложенных инструкций языка ДРАКОН.

Некоторые иконы не имеют графического представления. К таким относится частный случай иконы «Ветка». Если она представляет собой ветку «Выбор» или ветку условной конструкции, то графически она никак не отображается, кроме как с помощью связи между условным оператором и иконой «Разделитель». Которая также не имеет собственного графического представления, но служит для сбора всех веток внутри макроиконы.

Икона «Вопрос» содержит массив указателей на основную и альтернативную ветку выполнения. После веток располагается специальная икона «Разделитель», которая обозначает границу условной конструкции. Каждая ветка представляет из себя отдельную макроикону «Ветка», внутри которых содержится последовательность икон.

Каждая «Ветка» условно независима и является подобием «Примитива» по своему поведению, в неё так-же можно вставлять произвольное количество икон. Они формирует свою собственную последовательность операндов, правила отображения ДРАКОН-схем применяются точно такие же, разница заключается в том, что, с точки зрения языка ДРАКОН, эти две конструкции выполняют разные задачи. «Примитивы» в совокупности формируют «Силуэт», а «Ветки» являются содержимым различных макроикон условных конструкций. Реализация макроиконы «Вопрос» продемонстрирована на рисунке 1.20

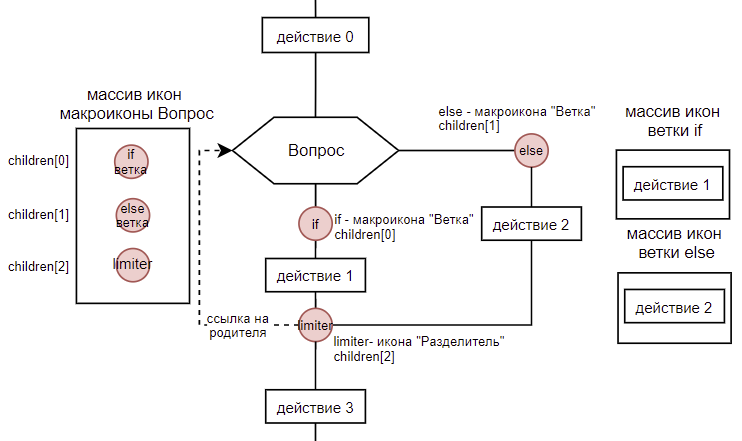


Рисунок 1.20 – Представление макроиконы «Вопрос»

Икона «Цикл» графически отображается в точности как «Вопрос» (см. рис 1.5), однако макроикона отличается направлением стрелки. Управление передаётся обратно, если условие не было выполнено.

«Цикл» содержит в себе обязательно одну «Ветку», в которой описывается набор инструкций, которые будут выполнены для попытки удовлетворения условию. Эта ветка является точкой входа в макроикону. Точкой выхода является сама икона «Цикл».

Вторая «Ветка» отсутствует, вместо неё икона цикла указывает на свою единственную ветку и на инструкцию, которая является внешней по отношению к ней (которая следует сразу за макроиконой «Цикл»).

Внешняя по отношению к циклу макроикона не знает подробностей реализации макроиконы «Цикл». Но в точке входа в неё автоматически считывается её внутреннее состояние, где доступна «Ветка» цикла и её содержимое.

На рисунке 1.21 показано представление макроиконы «Цикл».

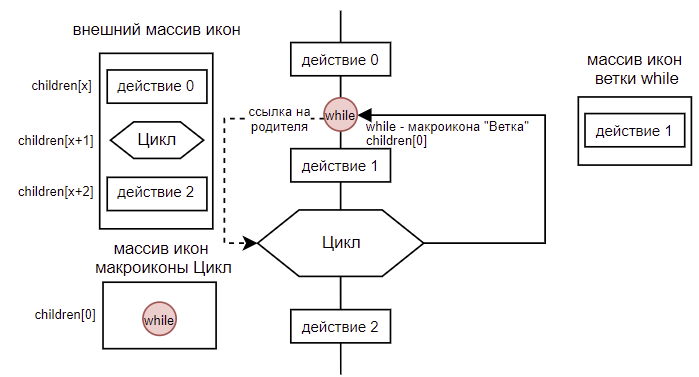


Рисунок 1.21 – Представление макроиконы «Цикл»

Макроикона «Вариант» подобно иконе «Вопрос» содержит несколько веток внутри себя. Но в отличии от неё количество этих веток условно неограниченно. В иконе «Вариант» указываются данные, состояние которых изменяются во времени, а сами состояния можно однозначно описать.

Каждая ветка именуются отлично друг от друга и располагается слева направо в зависимости от положения в массиве макроиконы «Вариант». Внутри каждой ветки свой набор икон, но каждая ветка хранит указатель на «Разделитель». Сам «Разделитель» хранит ссылку на икону «Варианта». Таким образом макроикона «Вариант» и её границы можно однозначно вычислить.

Представление макроиконы «Вариант» показано на рисунке 1.22.

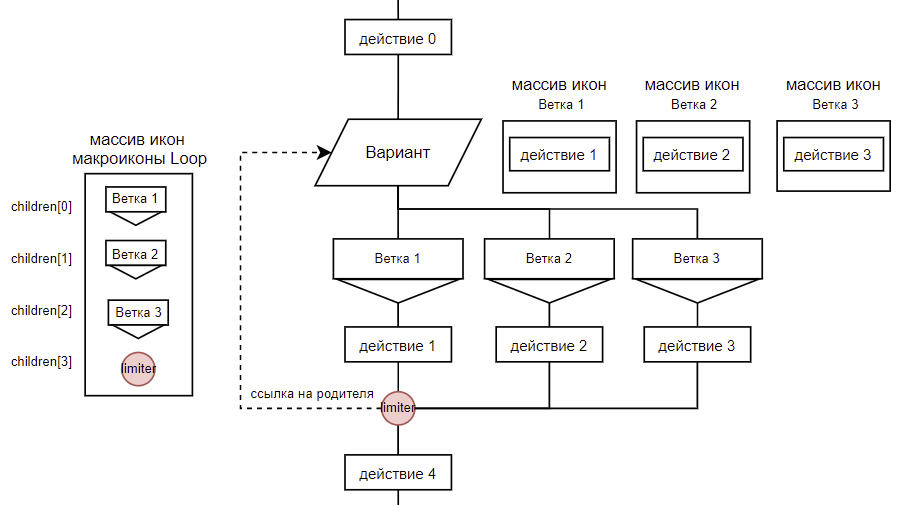


Рисунок 1.22 – Представление макроиконы «Вариант»

* + 1. Методы представления ДРАКОН-схем

Для однозначной идентификации каждой иконе присваивается уникальный ID, по которому можно обратиться как к самой иконе, так и к её родителю и всем вложенным в неё иконам. Благодаря уникальным номерам упрощается процесс вставки новых икон, их редактирование и удаление. Вторым преимуществом является упрощение сохранения дракон-схем в памяти компьютера. Каждой иконе соответствует уникальный идентификатор, что позволяет хранить иконы в любом желаемом виде, при условии читаемости данных на основе её ID.

К каждой иконе обеспечивается ассоциативный доступ. Для этого был разработан специальной контейнер - словарь. Для каждой иконы генерируется уникальный номер, который сохраняется во множестве использованных ключей, чтобы исключить вероятность дублирования идентификаторов. Ключ привязывается к иконе, устанавливаются ссылки на соседей, а сама икона сохраняется в словаре по указанному в ней ключу.

Благодаря такой реализации контейнер обладает качествами как контейнеров с ассоциативным доступом, так и с последовательным. Требуется это для того, чтобы при изменении (добавлении или удалении икон/макроикон) не было необходимости в обходе ДРАКОН-схемы.

При инициализации контейнера для ДРАКОН схемы генерируется базовая конструкция, которая включает в себя икону «Схема» и «Примитив», вложенную в неё. Это является базовой заготовкой для будущего алгоритма. Таким образом пользователь освобождается от лишней работе по созданию каркаса ДРАКОН-схемы.

Главной макроиконой в контейнере является «Схема». Её уникальным отличием является отсутствие родительской иконы. Она содержит название схемы, которое впоследствии выводится в иконе «Заголовок», и массив примитивов.

«Примитив» в схеме может быть один. В таком случае графически примитив отображается с помощью заключения содержимого между иконами «Заголовок» и «Конец». Количество вложенных икон может быть любым, но все иконы примитива располагаются на одной вертикали, только макроиконы могут осуществлять ветвления вправо, однако ход выполнения всё равно возвращается на главную вертикаль.

Пример отображения «Примитива» представлен на рисунке 1.23.

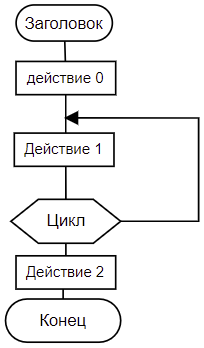


Рисунок 1.23 – Представление «Примитива»

Если отобразить все непечатаемые иконы, то можно графически показать на какие иконы ссылаются в качестве родителей инструкции в схеме. Например, как это продемонстрировано на рисунке 1.24, где для удобства содержимое икон «Схема» и «Примитив» выделены пунктирным квадратом.

А с помощью пунктирных стрелок показаны родительские отношения между иконами.

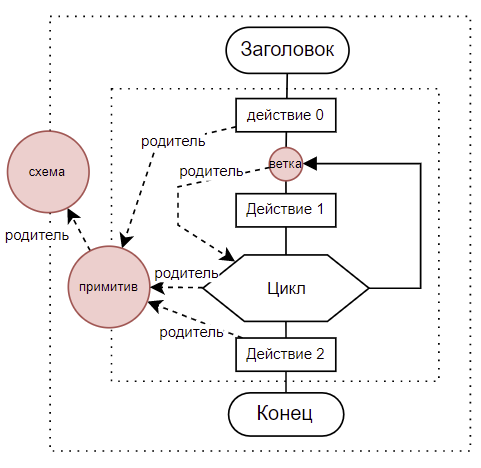


Рисунок 1.24 – Демонстрация родительской связи для икон «Ветка» и «Цикл»

Для иконы «Ветка» родителем является «Цикл». Для самой иконы «Цикл» родителем является «Примитив», который также содержит иконы «Действие 0» и «Действие 2». Сам «Примитив» является вложенной иконой «Схемы». Схема родительской иконы не имеет.

Стоит обратить внимание, что иконы «Заголовок» и «Конец» не представлены в схеме напрямую. «Заголовок» указывает на первую икону в «Схеме», а «Конец» на последнюю в иконе «Примитив», тем самым обеспечивается оптимизация при хранении ДРАКОН-схемы.

Для каждого типа икон есть шаблонные правила, по которым они могут быть вставлены в схему или быть удалены из неё. Таким иконам не нужна информация обо всей схеме, только данные родительской и соседних икон, где производится сама операция.

* + 1. Алгоритм вставки икон

Пользователь не может вставлять другие иконы во внутреннюю структуру макроикон, так как это может нарушить целостность всей схемы.

Для решения этой проблемы в каждой макроиконе есть специальные макроиконы «Ветка» и «Примитив», которые разрешают операцию вставки внутрь себя.

Пользователь может вставить до и после любой простой иконы в «Примитиве» и «Ветке». В случае, если они не содержат икон, то доступно только одно место для вставки: первым номером.

Доступные места для вставки отображаются графически с помощью круга зеленого цвета. Каждая такая икона содержит информацию о родителе, в котором она располагается и адрес следующей иконы, если он существует.

В первом случае новая икона вставляется в массив перед иконой, обозначенной как следующая. Во втором случае вставка осуществляется в конец массива.

Для макроикон определены собственные правила для вставки. Так для иконы «Вопрос» создаются две «Ветки», которые не добавляются в родителя, но в словарь, содержащий ID всех икон в схеме. Аналогично для всех остальных комплексных икон.

Алгоритм вставки новых икон показаны на рисунке 1.25.

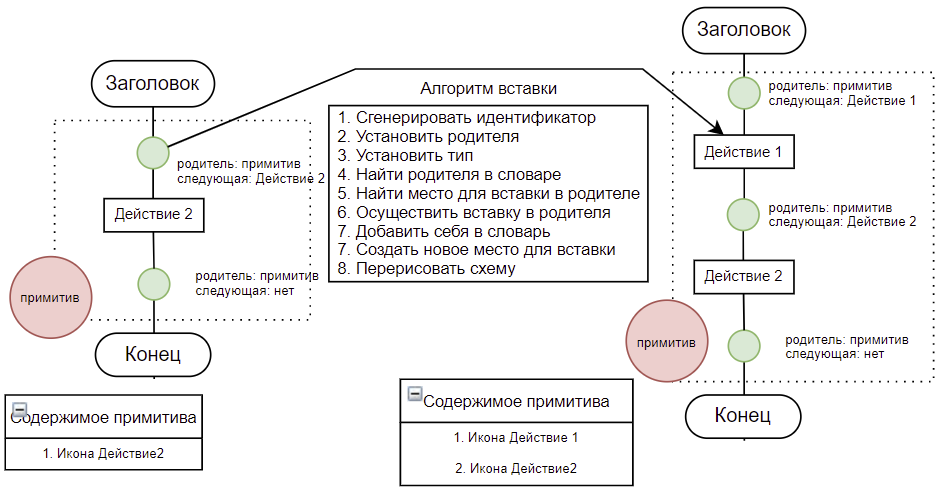


Рисунок 1.25 – Алгоритм вставки иконы

Каждая икона, добавляемая в схему, сохраняет себя в словаре, это позволяет получить доступ к её полям (см. приложение 1) по её идентификатору.

* + 1. Алгоритм удаления икон

Удалять из схемы можно практически все отображаемые иконы, кроме «Заголовок» и «Конец», которые неявно представляют икону «Схема». Алгоритм схож с вставкой, но есть некоторые значимые отличия.

Простые иконы, которые не содержат в себе других инструкций удаляются достаточно просто: по уникальному идентификатору в словаре находится ссылка на икону. После чего производится простая операция удаления иконы из массива со смещением. Также икона удаляется из словаря, после этого удаление можно считать завершенным.

Каждая удаляемая макроикона может содержать в себе некоторое множество других икон. В случае простого удаления все вложенные иконы останутся в висячем состоянии, когда формально иконы присутствуют схемы, но получить доступ к ним не представляется возможным. Поэтому все операции по удалению производятся рекурсивно.

Сначала производится поиск самой удаляемой иконы. Для всех икон, которые содержатся в удаляемой выполняется проверка: если содержимое присутствует, то осуществить переход в каждую икону и выполнить аналогичную проверку на наличие вложенных инструкций, в против случае произвести удаление.

* + 1. Описание проектируемых интерфейсов

Помимо решения проблемы хранения и представления ДРАКОН схемы в памяти ЭВМ, необходимо графически представить как саму схему, там и способы взаимодействия с ней.

Каждой иконе присваиваются координаты и размер в осях X и Y. Позиция каждой последующей иконы вычисляется на основе предыдущей.

Для отрисовки соединительных линий вычисляется центр получившегося прямоугольника, сформированного координатами, начала иконы и её размера как для иконы родителя, так и для иконы потомка.

При добавлении или удалении икон осуществляется пересчёт всех координат, так-как могут потребоваться существенные изменения в текущем построении.

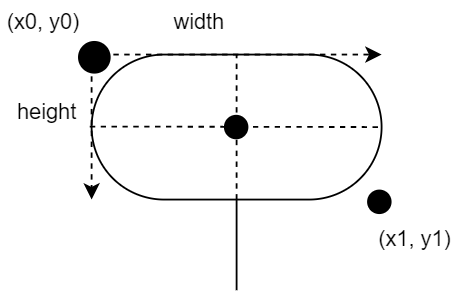


Рисунок 1.18 – Вычисление координат иконы

Места для вставки графически отображаются с помощью зеленого круга (см. рис. 1.25). Если пользователь хочет добавить икону в схему, для него необходимо графически показать все доступные места для вставки.

Все иконы, которые можно удалить также необходимо выделять. Если пользователь хочет исключить икону или макроикону из схемы, то необходимо подсветить все доступные для удаления иконы с помощью обводки красного цвета для всех икон, которые могут быть удалены из ДРАКОН-схемы.

Для ДРАКОН схемы необходима рабочая область, где будет отображаться сама схема. В ней также устанавливается разметочная сетка, которая служит для упрощения манипулирования схемой.

Манипуляции по добавлению новых икон должны проводиться через специальную панель, где пользователь выбирает нужную икону, а после место, куда требуется вставить икону.

Проектируемый интерфейс должен быть дружелюбен к пользователю. Весь основной функционал должен быть на главной странице приложения. Пользователь обязан иметь возможность как создавать и сохранять ДРАКОН схему, так и загружать уже существующую.

Программный продукт обязан предоставлять всю необходимую информацию о себе и его разработчике. Должен содержать в себе документацию, которая включает в себя учебник по работе с языком ДРАКОН и инструкцию по использованию самого программного продукта.

В рамках этой концепции был разработан прототип интерфейса редактора, показанный на рисунке 1.19.

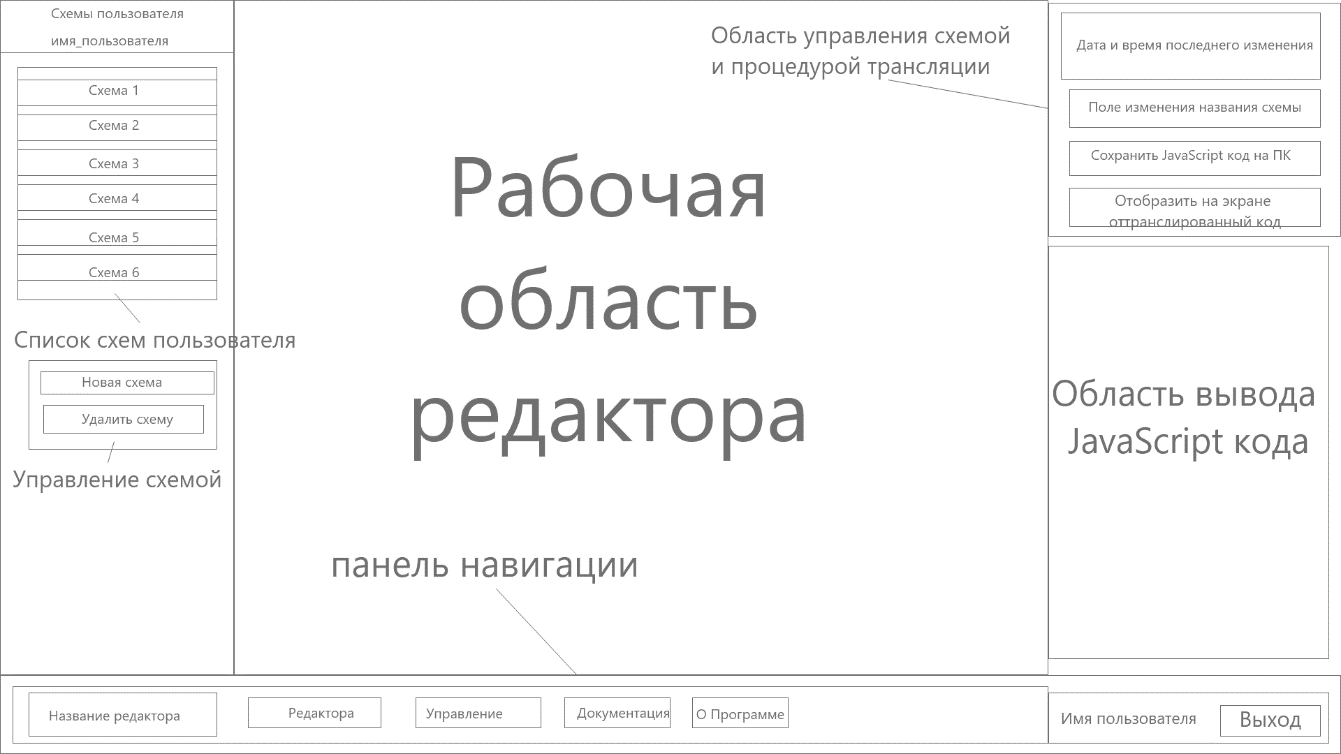


Рисунок 1.19 – Прототип графического интерфейса редактора

Редактор можно разделить на 4 основных блока:

* блок управления списком схем пользователя, куда включены методы добавления и удаления новых схем;
* рабочая область редактора, где отображаются ДРАКОН-схемы;
* блок управления схемой, в которой, в частности, выведена трансляция;
* панель навигации, которая содержит ссылки на основные страницы приложения и кнопку выхода.

На данном прототипе интерфейса основной графический блок с ДРАКОН-схемой располагается в центре. Именно на нём пользователь будет создавать дракон-схемы.

* 1. Инфологическая модель данных
     1. Сущности базы данных

Информационная система представляет из себя многопользовательское онлайн веб приложение, которое должно осуществлять сбор и обработку данных пользователей. База данных для разрабатываемой системы должна содержать следующее:

* информация о всех пользователях системы;
* роли пользователей в ИС;
* списки курируемых пользователей для кураторов;
* ДРАКОН-схемы каждого пользователя.

Таким образом, можно выделить следующие сущности базы данных, представленные в таблице 1.1.

Таблица 1.1 – Сущности базы данных

|  |  |
| --- | --- |
| **Сущность** | **Атрибуты** |
| Пользователь | Идентификатор пользователя, имя пользователя, почта, пароль, роль |
| Курируемое отношение | Идентификатор отношения, идентификатор куратора, идентификатор пользователя, имя отношения |
| ДРАКОН-схема | Идентификатор схемы, название схемы, идентификатор пользователя, структура схемы, дата последнего изменения, идентификатор пользователя, сделавшего последнее изменение |

Данные таблицы 1.1 также можно представить в виде ER-диаграмм (рис.1.26-1.28). На рисунке 1.26 изображен фрагмент ER-модели для сущности «Пользователь».

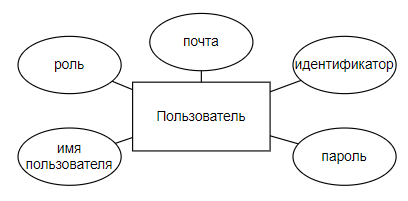


Рисунок 1.26 – Фрагмент ER-модели для типа сущности «Пользователь»

Фрагмент ER-модели сущности «Курируемый пользователь» показан на рисунке 1.27.

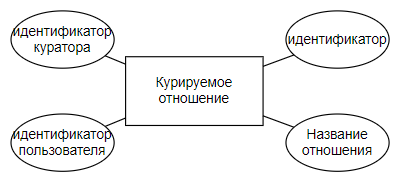


Рисунок 1.27 – Фрагмент ER-модели для типа сущности «Курируемое отношение»

Фрагмент ER-модели сущности «ДРАКОН-схема» показан на рисунке 1.28.

|  |
| --- |
|  |

Рисунок 1.28 – Фрагмент ER-модели для типа сущности «ДРАКОН-схема»

Из сущностей, представленных на рисунках 1.26-1.28, сформирована диаграмма «Сущность-связь», представленная в приложении 2.

* 1. Входная и выходная информация
     1. Входная информация

К входным данным относятся:

* Регистрационные данные пользователя (логин и пароль);
* Действия пользователя:
  + загрузка схем;
  + редактирование схем;
  + смена регистрационного имени;
  + смена пароля;
* Действия преподавателя в интерфейсе администрирования:
  + редактирование списка курируемых пользователей;
  + редактирование отношения между куратором и пользователем
* Действия администратора в интерфейсе администрирования:
  + выдача ролей;
  + удаление пользователей;
    1. Выходная информация

К выходным данным относятся:

* Файл ДРАКОН-схемы в формате JSON;
* Файл с исходным кодом оттранслированной ДРАКОН-схемы на языке JavaScript;
* Список ДРАКОН-схем и их содержимое из базы данных;
* Список курируемых пользователей для каждого куратора.
  1. Формат содержимого ДРАКОН-схемы

ДРАКОН-схема должна включать в себя способы собственного хранения на физических носителях, в частности в базе данных. В PostgreSQL наиболее удобным способом хранения набора данных с переменным числом атрибутов является формат JSON.

Для хранения дракон-схем был разработан специальный JSON-формат. Информацию о расположении хранить избыточно, так как представление осуществляется на стороне клиента. При загрузке дракон-схем производится перерасчёт расположения всех икон.

Каждая схема состоит из двух основных объектов: идентификатора макроиконы «Схема» и список всех икон в схеме, которые содержат следующую информацию:

* «id» - атрибут, содержит уникальный uuid иконы;
* «type» - тип иконы;
* «parent» - содержит uuid родительской иконы;
* «children» - содержит список из uuid всех своих дочерних икон.

К каждой иконе можно получить доступ через её уникальный идентификатор.

Пример представления ДРАКОН-схемы в формате JSON представлен на рисунке

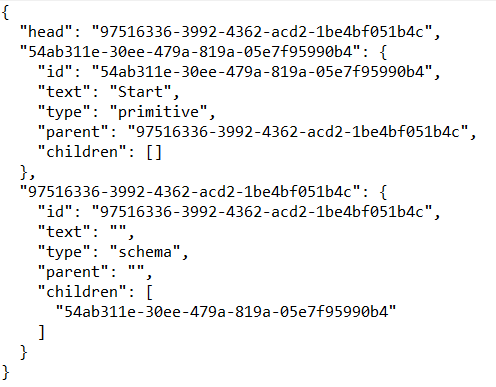


Рисунок 1.29 – Представление ДРАКОН-схемы в формате JSON

* 1. Трансляция ДРАКОН-схемы в JavaScript

Любую ДРАКОН-схему можно однозначно представить в синтаксисе языка JavaScript. Каждой иконе сопоставляется конструкция языка:

* действие – операторы объявления переменных, констант, математических операций;
* комментарий – синтаксис многострочных комментариев языка Javascript;
* вывод – операторы вывода в консоль console.log();
* ввод – не обладает единой реализации в JavaScript в различных средах выполнения реализации отличаются друг от друга;
* пауза – операторы синхронного ожидания таймера await SetTimeout();
* вопрос – операторы if-else;
* цикл – оператор do-while;
* вариант – оператор switch;
* ветка – конструкция блока {};
* примитив – функция function MyFync(){}.

Трансляция начинается с чтения иконы «Схема». Изначально формируется пустая строка, содержащую только идентификатор иконы «Схема». Впоследствии рекурсивно происходит обход всех дочерних элементов для каждой иконы, где идентификаторы заменяется на конструкции языка JavaScript.

Пример сопоставления содержимого ДРАКОН-модели с программой на языке JavaScript показан на рисунке 1.30. Слева направо показан уровень вложенности каждой иконы. Столбец представляет из себя массив икон, которые содержатся в родителе, с которым иконы в столбце связаны.

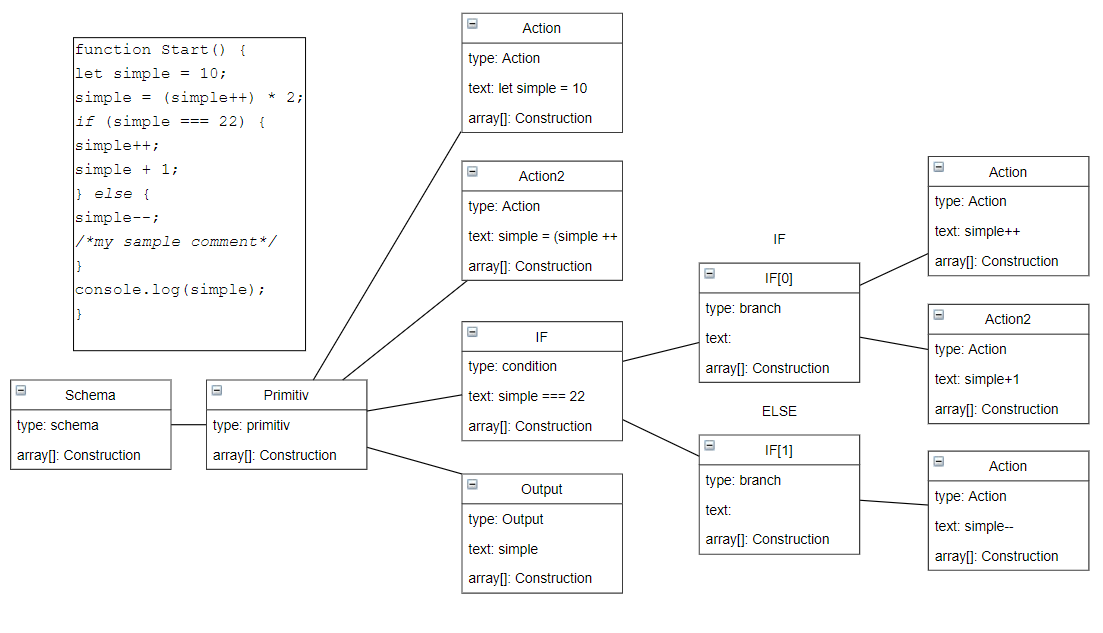


Рисунок 1.30 – сопоставление ДРАКОН-модели с кодом JavaScript

* 1. Требования к техническому и программному обеспечению

Рабочая станция:

* Intel-совместимый процессор с частотой не менее 2х1,6 ГГц;
* не менее 1024 МБ ОЗУ;
* не менее 200 MБ свободного места на диске;
* операционная система: ОС с поддержкой Chrome v.63+.

Сервер приложения и сервер базы данных:

Intel-совместимый процессор с частотой не менее 8х3,4 ГГц;

* не менее 16 384 МБ ОЗУ;
* не менее 100 ГБ свободного места на диске;
* дисковод CD-ROM/DVD-ROM;
* операционная система: Windows Server 2008+ или схожая по функциональным возможностям UNIX-подобная ОС.

2 РАБОЧИЙ ПРОЕКТ

* 1. Общие сведения о работе системы

Программный продукт разработан в среде разработки Visual Studio Code на языках TypeScript и JavaScript.

Программный продукт использует клиент-серверную модель архитектуры.

Для построения сервера используется библиотека NestJS v.7.5.0. В качестве СУБД используется PostgreSQL 13.2. В базе данных хранятся данные о пользователях системы, их ДРАКОН-схемы и отношения пользователей в системе. Для связи СУБД и сервера используется библиотека TypeORM v.0.2.3.

Все интерфейсы проектируемой системой разработаны с использованием библиотеки построения пользовательских интерфейсов – React v 17.0.0.

Клиент – одностраничное (SPA) веб-приложение использующий динамическую подгрузку HTML, CSS и JavaScript для построения компонентов интерфейса.

* 1. Функциональное назначение программного продукта

Разработанный программный продукт предназначен для получения базовых знаний по основам алгоритмизация с помощью построения схем на языке ДРАКОН. Программа должна предоставлять пользователю возможность создавать, изменять и удалять схемы. Программа имеет следующие функциональные возможности:

* регистрация нового пользователя;
* аутентификация пользователя в системе с помощью логина и пароля;
* удаление учётной записи пользователя;
* обновление пароля пользователя;
* изменение логина пользователя;
* создание схемы;
* загрузка схемы из базы данных;
* редактирование схемы и её содержимого;
* удаление схемы;
* просмотр и изменение куратором схем других пользователей;
* добавление/удаление пользователя в список курируемых;
* изменение роли пользователя администратором;
* удаление учетной записи пользователя администратором.

Программа имеет следующие функциональные ограничения:

* не реализованы механизмы изменения масштаба и прокручивания области редактирования;
* отсутствует возможность создавать силуэты языка ДРАКОН;
* положение икон строго зафиксированы на рабочей области.
  1. Выполнение программного продукта

Программный продукт располагается в папке dragon в каталоге с проектом. Для развертывания системы необходимо осуществить первоначальную настройку системы.

Информационная системе для развертывания необходима функционирующая база данных PostgreSQL v 13.0+. Сервер может располагаться как на удалённом сервере, так и на той же машине. Параметры подключения задаются в специальном конфигурационном файле «dragon/backend/ormconfig.json». Необходимо указать следующие параметры:

* host;
* port;
* username;
* password;

Предварительно необходимо инициализировать новую базу данных с именем «drakon\_db» удобным для пользователя способом.

SQL скрипт для инициализации базы данных располагается по пути «dragon/backend/scripts/drakon\_db.initscript.sql». После выполнения скрипта инициализации база данных готова к работе.

Для сборки программного продукта потребуется NodeJS версии 12.3 или выше. Для начала необходимо установить все зависимости для проекта «backend» с помощью команды npm install. После ввести последовательно команды npm run seed:config и npm run seed:run для создания роли администратора по умолчанию.

В файле .env в папке «dragon/backend» указать параметр PORT и HOSTNAME для прослушивания. Продублировать порт в «dragon/client-react/.env» в параметр REACT\_APP\_CLIENT\_PORT и указать хост и порт в параметре REACT\_APP\_SERVER\_DOMAIN.

Для запуска собственного сервера NodeJS достаточно выполнить команду npm run build в директориях «dragon/client-react» и «dragon/backend» для сборки приложения с текущими параметры порта и хоста. После выполнить команду node dist/main.js в папке «dragon/backend», которая запустит сервер Node. Приложение готово к работе.

Для начала работы необходимо открыть браузер по адресу, который был задан для прослушивания (по умолчанию http://localhost:5000). При переходе на домашний URL будет открыта страницу входа в приложение. В системе предварительно есть уже зарегистрированная учётная запись администратора с комбинациями логина и пароля «Administrator». Перед осуществлением эксплуатации системы настоятельно рекомендуется войти в систему под этой учётной записью и изменить пароль в панели администратора.

Для регистрации необходимо создать нового пользователя, необходимо придумать имя, пароль и ввести почту для регистрации. В случае успешной регистрации будет получен JWT токен для доступа к защищённым URL приложения, а пользователь будет перенаправлен на домашнюю страницу приложения.

* 1. Физическая архитектура системы

Проектируемая система состоит из нескольких компонент. В качестве СУБД используется PostgreSQL версии 13.2. Сервер представляет из себя NodeJS приложение, которое содержит библиотеку Express для обслуживания запросов от клиентов. Клиентами представляется одна или нескольких машин с доступом к сети, в которой работает сервер. Сервер предоставляет клиентам html страницы и данные из базы в ответ на запросы со стороны браузера. Диаграмма развертывания представлена в приложении 3 (см. рис. П.3).

* 1. Описание программы

Программный продукт следует разделить на 3 отдельных компонента:

* клиент;
* сервер;
* база данных.
  + 1. Описание сервера

Сервер отвечает за обслуживание запросов со стороны клиентов и взаимодействие с базой данной. NestJS предоставляет модульную архитектуру, в которой каждый компонент подключается к основному модулю системы, расширяя функциональность сервера.

Пользовательские запросы обрабатываются через URL вида «домен/api/\*» сквозь которые предоставляется API для получения информации из модулей сервера, которые обеспечивают связь с базой данных. Все модули спроектированы с учётом архитектурного стиля RESTful API.

В программе реализовано 7 модулей:

* UserModule;
* AuthModule;
* CuratorsModule;
* SchemaModule;
* TypeOrmModule;
* ServeStaticModule;
* ConfigModule.

UserModule отвечает за взаимодействие с сущностью «Пользователь» базы данных. Описание модуля приведено в таблице 2.1.

Таблица 2.1 – Описание модуля UserModule

|  |  |
| --- | --- |
| **Компонент** | **Назначение** |
| UserService | Обработчик запросов к базе данных к сущности «Пользователь» |
| UserController | Обработчик вызовов к api/users |

AuthModule обеспечивает механизмы регистрации новых пользователей и аутентификацию уже зарегистрированных. Описание модуля приведено в таблице 2.2.

Таблица 2.3 – Описание модуля AuthModule

|  |  |
| --- | --- |
| **Компонент** | **Назначение** |
| AuthService | Обработчик запросов к базе данных к сущности «Пользователь» с целью аутентификации пользователя |
| AuthController | Обработчик вызовов к api/auth системы авторизации |
| BrcyptService | Сервис шифрования и валидации паролей |
| JwtStrategy | Сервис стратегии по управлению защищёнными путями URL |
| LocalStrategy | Сервис стратегии по валидации учетных данных пользователя |

CuratorModule отвечает за взаимодействие с сущностью «Куриремое отношение» базы данных. Описание модуля приведено в таблице 2.1.

Таблица 2.3 – Описание модуля CuratorModule

|  |  |
| --- | --- |
| **Компонент** | **Назначение** |
| CuratorService | Обработчик запросов к базе данных к сущности «Куратор» |
| CuratorController | Обработчик вызовов к api/curator |

SchemaModule отвечает за взаимодействие с сущностью «ДРАКОН-схема» базы данных. Описание модуля приведено в таблице 2.1.

Таблица 2.3 – Описание модуля SchemaModule

|  |  |
| --- | --- |
| **Компонент** | **Назначение** |
| CuratorService | Обработчик запросов к базе данных к сущности «Куратор» |
| CuratorController | Обработчик вызовов к api/curator |

TypeOrmModule обеспечивает взаимодействие с базой данной. Его настройка осуществляется через файл «ormconfig.json». Помимо связи с базой данной, также предоставляет набор методов для осуществления запросов к словарям.

ServeStaticModule отвечает за обслуживание статических файлов. В программе таковыми файлами является собранный клиент приложения и его медиаресурсы (иконы, изображения, файлы markdown).

ConfigModule предоставляет поддержку конфигурационных файлов .env в среде NestJS.

Диаграмма компонентов сервера приведена в приложении 4(см. рис П4).

* + 1. Общее описание базы данных и даталогическая модель данных

Сервер осуществляет взаимодействие с данными через СУБД PostgreSQL v13.2. В ответ на SQL запросы по протоколу TCP/IP сервер получает и обновляет данные в базе. БД состоит из 3 справочников:

* пользователь;
* курируемое отношение;
* ДРАКОН-схема.

Справочник «Пользователь» предназначен для хранения информации о пользователях системы. Справочник содержит уникальный идентификатор пользователя, а также зашифрованный пароль с помощью функции bcrypt и информацию о роли пользователя в системе, а также почту. Идентификация пользователя осуществляется с помощью uuid, однако имя пользователя также должно быть уникальным для удобной идентификации через интерфейсы системы. Атрибуты справочника «Пользователи» представлены в таблице 2.6.

Таблица 2.6 – Справочник «Пользователи»

| **Параметр** | **Тип** | **Размер** | **Примечание** |
| --- | --- | --- | --- |
| id | uuid, primary key | 16 | Код записи. Уникальное значение. Первичный ключ. |
| username | varchar | 50 | Уникальное имя пользователя, используется для идентификации пользователя клиентом |
| email | varchar | 60 | Почта учетной записи пользователя |
| password | text | переменный | Хешированный пароль пользователя |
| role | varchar | 50 | Пароль учетной записи пользователя |

Атрибуты справочника «Курируемое отношение» показаны в таблице 2.7.

Таблица 2.7 – Справочник «Курируемое отношение»

| **Параметр** | **Тип** | **Размер** | **Примечание** |
| --- | --- | --- | --- |
| id | serial, primary key | 4 | Номер записи. Уникальное значение. Первичный ключ. |
| id\_curator | uuid, foreign key | 16 | Код записи куратора, Внешний ключ. Связан с первичным ключом таблицы «Пользователь». Вместе с id\_user образуют уникальную пару куратор-пользователь. |
| id\_user | uuid, foreign key | 16 | Код записи пользователя, Внешний ключ. Связан с первичным ключом таблицы «Пользователь». Вместе с id\_curator образуют уникальную пару куратор-пользователь. |
| relation\_name | varchar | 100 | Название для отношения кураор-пользователь |

Справочник «ДРАКОН-схема» содержит содержимое схемы, идентификатор пользователя, создавшего её, а также название, информацию о времени и пользователе, внесшим последнее изменение. Информация о последнем изменении сохраняется для того, чтобы идентифицировать куратора или пользователя, который последним изменял схему. Атрибуты справочника «ДРАКОН-схема» представлены в таблице 2.8.

Таблица 2.8 – Справочник «ДРАКОН-схема»

| **Параметр** | **Тип** | **Размер** | **Примечание** |
| --- | --- | --- | --- |
| id | uuid, primary key | 4 | Номер записи. Уникальное значение. Первичный ключ. |
| name | varchar | 100 | Название схемы |
| id\_user | uuid | 16 | Код записи пользователя. Содержит id владельца схемы |
| data | varchar | 100 | Содержимое ДРАКОН-схемы |
| last\_changed | datetime | 8 | Дата и время последнего изменения схемы |
| last\_changed\_by\_id | uuid | 16 | id пользователя, изменения которого были последними. |

* + 1. Общее описание работы клиента

Клиент написан с применением библиотеки React и реализован как одностраничное приложение. Это означает, что фактически в приложении существует одна единственная страница.

Сервер, при получении соответствующего URL запроса отправляет на клиент HTML страницу и скрипты JavaScript для взаимодействия с клиентской частью приложения. Для реализации возможности навигации в приложении клиента используется библиотеке «react-router», которая позволяет осуществлять рендер только тех интерфейсов системы, которые однозначно сопоставляются с URL путём.

Для рендера используется статическая модель, в которой все пути заранее известны. В системе выделяются следующие компоненты пользовательского интерфейса для рендера:

* страница регистрации;
* страница редактора ДРАКОН-схем;
* страница управления учетной записью и администрирования пользователей;
* страница документации по работе с ИС и языком ДРАКОН;
* страница информации о разработанном продукте;
* информационная страница о несуществующем пути приложения (форма ошибки 404).

Страница регистрации содержит все необходимые интерфейсы для авторизации пользователя в системе.

Страница редактора ДРАКОН-схем предоставляет функциональную возможность взаимодействия со схемами для пользователей системы.

На странице управление учетной записью пользователь может изменять конфиденциальные данные. Куратор редактировать список курируемых пользователей, а администратор отвечает за выдачу ролей в системе.

Интерфейс документации содержит всю основную информацию по способам взаимодействия с системой, а также содержит описание функциональных возможностей и ограничений программы, а также правила языка ДРАКОН.

Страница сведений о продукте содержит информацию о разработчике, разрабатываемой системе и все необходимые ссылки на рабочие репозитории проекта.

* + 1. Описание процесса аутентификации

По умолчанию по домашнему адресу приложения «/» осуществляется открытие интерфейса редактора. Но предварительно осуществляется проверка на наличие сохранённых данных пользователя.

Для хранения информации об учетной записи используется localStorage – интернет-хранилище в браузере клиента, которое позволяет осуществлять запись, чтение и удаление важных данных, которые необходимо сохранять локально на рабочей станции клиента.

Структура регистрационных данных пользователя, которая сохраняется в браузере клиента, имеет вид, представленный в таблице 2.22.

Таблица 2.8 – структура пользователя в localStorage

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Примечание** |
| username | Текстовый или NULL | Уникальное имя пользователя в системе |
| access\_token | Текстовый или NULL | Сохранённый JWT токен доступа |
| uuid | Текстовый или NULL | Уникальный идентификатор пользователя |
| role | Текстовый или NULL | Роль пользователя в системе |
| email | Текстовый или NULL | Почта пользователя |

Структура, описанная в таблице 22, полностью совпадает с сущностями из справочника «Пользователь» за исключением JWT токена. Если пользователь уже регистрировался в системе, то его данные буду сохранены в браузере, в противном случае необходимо пройти процедуру аутентификации.

Аутентификация пользователей осуществляется через специальную форму регистрации и входа.

Для успешного прохождения процесса регистрации необходимо ввести уникальное имя, по которому можно будет однозначно идентифицировать пользователя, пароль и почту. Если пользователь ранее зарегистрирован в системе, достаточно ввести валидную комбинацию логина и пароля.

На стороне клиента реализованы специальные сервисы формирования запросов и обработки результатов.

Сервис обработки запросов аутентификации отвечает за формирование пакета данных для отправки на сервер с целью регистрации пользователя, проверки валидности данных, выдачи и обновления JWT токенов. А также реализует обработчики для полученных с сервера ответов на запросы (Response).

Для валидации пользователя необходимо отправить на сервер форму с валидным именем пользователя и паролем. Сервер осуществит запрос на сравнение пользователя из базы данных с полученным от клиента именем. В случае, если пользователь найден, сервер получит объект сущности User, произведет хэширование пароля с той же солью, что и у пароля, хранимого в базе. Если пароли совпадают, сервер генерирует код возврата 200 и JWT токен регистрации, который будет валиден в течении 360 минут после успешного сеанса регистрации. После истечения срока валидности токена пользователю необходимо повторно ввести логин и пароль для обновления токена.

В случае возникновения ошибок при прохождении процедуры аутентификации выводится информационной сообщение со сведениями о произошедшей ошибке и о способах устранения.

Если аутентификация прошла успешна: пользователь в системе существует или же был создан, то ему выдаётся JWT токен доступа, который сохраняется в localStorage браузера вместе со всей информацией о текущем пользователе.

Такой подход позволяет сократить количество запросов к серверу, всю необходимую информацию для формирования запросов клиент хранит самостоятельно.

Эта же информация дублируется в специальный компонент библиотеки React – контекст. Контекст в React позволяет получить доступ к данным в любой точке приложения, принцип схож с глобальной переменной, однако для неё реализованы специальные методы по защите от случайных изменений.

* + 1. Описание процесса взаимодействия с ДРАКОН-схемой

Если аутентификация пройдена успешна, а значит пользователю выдан JWT токен доступа, то осуществляется переход на главную страницу приложения, которой является интерфейс редактора ДРАКОН-схем.

На данной странице отображаются все необходимые элементы интерфейса для взаимодействия с ДРАКОН-схемами, сохранёнными в базе данных. По умолчанию каждый новый пользователь не имеет ДРАКОН-схем, закрепленных за ним. Однако интерфейс системы предусматривает создание новых ДРАКОН-схем, которые, в момент создания, добавляются в словарь «ДРАКОН-схема» и закрепляются за пользователем.

Редактор ДРАКОН-схем реализован с помощью графической библиотеки «Konva» по работе с элементом HTML Canvas. Редактор предоставляет функциональные возможности по созданию, редактированию и удалению икон и макроикон в выбранной пользователем схеме, позволяет перемещать любые компоненты схемы в пределах рабочей области, а также заполнять иконы функциональным содержимым.

Изначально при входе в интерфейс редактора загружаются все идентификаторы ДРАКОН-схем из базы данных. Пользователю необходимо выбрать нужную ему схему для загрузки данных о её содержимом из базы данных.

ДРАКОН-схема, предварительно созданная пользователем, загружается в формате JSON. После осуществляется процесс её чтение с целью формирования ДРАКОН-модели, соответствующая экземплярам класса DragonModel (см. рис П.1).

На основе ДРАКОН-модели генерируется графическое отображение схемы библиотекой «Konva», где учитываются все требования, предъявляемые со стороны языка ДРАКОН.

Пользователь может добавлять или удалять иконы. Все возможные иконы для вставки предоставляются элементами интерфейса, как и сама возможность изменения содержимого ДРАКОН-схем.

После окончания редактирования схемы пользователю необходимо сохранить изменения, нажав соответствующую клавишу. Тогда схема снова преобразуется в формат JSON и отправляется формой на сервер, который реализует механизмы обновление данных в БД.

* + 1. Описание процесса управления пользовательскими данными

Каждому пользователю доступна возможность изменять свои конфиденциальные данные. Для этого в программе есть специальный интерфейс администрирования. Количество доступных вариантов использования зависит от роли пользователя в системе.

Обычному пользователю предоставляется возможность изменять своё уникальное имя в системе и пароль от учётной записи.

Куратору предоставляется возможность просмотра списка всех пользователей с функцией фильтра. Каждого простого пользователя Куратор может добавить в список курируемых, что позволит ему просматривать и изменять его схемы в интерфейсе редактора ДРАКОН-схем.

Администратор в системе необходим для установки ролей. Он также может просматривать любого пользователя в системе, но именно он назначает кто в системе является куратором, а кого назначить администратором. Дополнительно Администратор обладает функциональной возможностью удалять учётной запись пользователя из системы.

* 1. Обеспечение защиты паролей

Для обеспечения безопасности хранимых в системе паролей используется шифрование. За него ответственен BcryptService модуля AuthModule. Его описание приведено в таблице 2.20.

Таблица 2.1 – Описание сервиса BcryptService

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| hashPassword | Шифрование пароля с помощью функции bcrypt с заданной солью |
| checkPassword | Шифрование кандидата с помощью функции bcrypt с заданной солью и сравнение с валидным зашифрованным паролем. |

Шифрование с помощью функции bcrypt является необратимым процессом. Зашифрованную последовательность символов невозможно вернуть в исходную. В базе данных все пароли хранятся в зашифрованном виде. Даже если злоумышленник получит доступ к базе данных, пароли останутся защищёнными.

* 1. Определение целостности сущностей

Каждая запись в справочниках обладает своим уникальным идентификатором и содержит набор данных.

В таблице «Пользователь» первичным ключом является поле ID, которое представляет из себя генерируемый уникальный uuid, с помощью которого можно уникально идентифицировать пользователя в базе. Чтобы у пользователей системы не было необходимость запоминать длинную комбинацию произвольных символов при аутентификации, каждому ID сопоставляется уникальное имя, которое пользователь выбирает самостоятельно.

В таблице «Куратор» первичным ключом служит поле ID. Данный ключ позволяет осуществлять быструю выборки из списка кураторских отношений, а также позволяет однозначно интерпретировать это отношение в базе. Но уникальность записи обеспечивает уникальная пара id\_user и id\_curator, что исключает возможность добавления нескольких вариантов отношений для одной пары куратор + пользователь.

В таблице «ДРАКОН-схема» первичным ключом служит поле ID. Уникальный идентификатор позволяет однозначно интерпретировать схему в системе, чтобы у пользователя всегда оставалась возможность манипулировать только своими, или только доступными ему, схемами.

* 1. Определение целостности атрибутов справочников

Атрибуты в справочниках не могут быть пустыми. Каждое поле важно для целостности записи.

Для таблицы «Пользователь» поля идентификатора, имени пользователя и пароля нужны для идентификации пользователя в системе и обеспечения возможности входа. Роль необходима для предоставления различных привилегий. Для куратора такими привилегиями является возможность изменять схемы других пользователей, которые добавлены к нему в таблицу. Для администратора предоставляется возможность удаления пользователей и изменение ролей.

В таблице «Куратор» идентификаторы пользователя и куратора служат для однозначной интерпретации связи между двумя пользователями. А поле «название отношения» однозначно описывает эту связь.

В таблице «ДРАКОН-схема» поле названия необходимо для корректного отображения схемы на клиенте, поле «data» содержит саму схему, даже если схема пустая, в ней содержится метаинформация для построения новой схемы. Дата последнего изменения и идентификатор пользователя, который произвёл последние манипуляции с схемой позволяют однозначно идентифицировать куратора или пользователя, который произвёл изменения и время совершения операции.

* 1. Определение ссылочной целостности атрибутов справочников

Ограничение внешнего ключа необходимо для соединения двух таблиц вместе и используется в таблицах «Куратор» и «ДРАКОН-схема».

Внешний ключ «id\_куратор» в таблице «Куратор позволяет получить список всех курируемых пользователей из таблицы для текущего куратора. Внешний ключ «id\_user» позволяет получить доступ к имени пользователя и его схемам в таблице «ДРАКОН-схема».

Внешний ключ «id\_user» в таблице «ДРАКОН-схема» служит для получения только тех схем, которые принадлежат данному id. Внешний ключ «last\_changed\_by\_id» позволяет получить имя куратора или пользователя, который последним производил изменения в таблице.

* 1. Основные пользовательские функции, триггерные функции, триггеры

Для управления пользователями, отношениями между кураторами и курируемыми, а также схемами, были реализованы триггеры и триггерные функции в базе данных, чтобы исключить возможность передачи неправильных входных данных со стороны сервера для уязвимых к ошибочным данным полям. Основные триггерные функции и триггеры представлены в таблице 2.9.

Таблица 2.9 – Основные триггерные функции и триггеры

|  |  |  |
| --- | --- | --- |
| **Триггерная функция** | **Триггер** | **Назначение** |
| user\_registration | CREATE TRIGGER register\_new\_user\_tr BEFORE INSERT ON users EXECUTE PROCEDURE *user\_registration*(); | Генерация uuid для нового пользователя |
| user\_rename | CREATE TRIGGER user\_rename\_trg BEFORE UPDATE OR INSERT ON users EXECUTE PROCEDURE *user\_rename*(); | Проверка нового имени пользователя на наличие недопустимых символов |
| delete\_user | CREATE TRIGGER delete\_user\_trg AFTER DELETE ON users EXECUTE FUNCTION *delete\_user*(); | Удаление данных о пользователе, всех его схем из таблицы «ДРАКОН-схема» и записей в таблице «Куратор» |
| curator\_relation\_role\_check | CREATE TRIGGER validate\_user\_role BEFORE INSERT OR UPDATE ON curators EXECUTE PROCEDURE *curator\_relation\_role\_check*(); | Проверка отношения куратор-пользователь на корректность ролей. |
| scheme\_insert | CREATE TRIGGER scheme\_insert\_trg BEFORE INSERT ON dragon\_scheme EXECUTE PROCEDURE *scheme\_insert*(); | Генерация uuid ключа для таблица «ДРАКОН-схема» |

* 1. Использование JSON-объектов

JSON необходим для удобного и компактного хранения сложного массива данных, количество полей в котором является динамическим. В таблице «ДРАКОН-схема» используется поле data типа jsonb, который хранит сложную структуру дракон схемы в виде единого объекта, что позволяет быть уверенным, что данные не мутируют без ведома системы и пользователя, а значит могут быть однозначно интерпретированы.

* 1. Обеспечение безопасности базы данных

Одним из важных аспектов в клиент-серверной архитектуре является организация и обеспечение безопасности как самой системы, так и данных, обрабатываемой и хранимой ей. В текущей итерации программный данные в программном продукте защищены следующими факторами:

* каждая триггерная функция, выполняемая до операции, обрабатывает исключения, что позволяет избежать занесения некорректных данных в базу;
* каждый DTO объект, который поступает с запросом на сервер, проходит валидацию на соответствие полей типу;
* каждый путь приложения, через который можно получить доступ к API базы данных защищён с помощью стратегии безопасности JWT, т.е. каждый пользователь должен произвести авторизацию и получить валидный токен;
* все роли имеют ограниченные возможности по обращению к базе данных только через соответствующие реализованные методы;
* пароли учетных записей невозможно дешифровать, сами пароли всегда хранятся только в зашифрованном виде с помощью функции bcrypt.
  1. План обслуживания и резервного копирования базы данных

Резервное копирование базы данных осуществляется с помощью вспомогательной программы pg\_dump.

* 1. Описание основных классов системы
     1. Описание основных классов сервера

Для описания сущности «Пользователь» в системе был реализован класс пользователя, описание которого представлено в таблице 2.20.

Таблица 2.10 – Описание класса User

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| uuid | string | Id пользователя |
| username | string | Имя пользователя |
| email | string | Почта |
| password | string | Пароль |
| role | string | Роль |

Класс Curator представляет описание сущности «Курируемое отношение» внутри разрабатываемой системы. Описание класса размещено в таблице 2.21.

Таблица 2.11 – Описание класса Curator

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| id | number | Id отношения |
| uuid\_curator | string | Id куратора |
| uuid\_user | string | Id пользователя |
| relation\_name | string | Название отношения |

Класс Schema представляет описание сущности «ДРАКОН-схема» для связи с базой данных (табл 2.12).

Таблица 2.12 – Описание класса Schema

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| uuid | string | Id схемы |
| name | string | Название схемы |
| id\_user | string | Id пользователя |
| data | JSON | Содержимое схемы |
| last\_changed | Date | Дата и время последнего изменения |
| last\_changed\_by\_id | string | Id пользователя, сделавшего последнее изменение |

Для авторизации используется связка логин + пароль. Данные от клиента на сервер доставляются с помощью специальных DTO объектов, которые проходят проверку на корректность данных (табл 2.13).

Таблица 2.13 – Описание LoginUserDTO

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| username | string | Имя пользователя |
| password | string | Пароль |

Для создания новой учетной записи используется класс CreateUserDTO. Важным отличием от LoginUserDTO является наличия поля для почта и специального декоратора для поля пароля, который осуществляет проверку содержимого на удовлетворение минимальным требования, которые описаны в подразделе 1.2.2.

Если пароль недостаточно надёжный, то возвращается статус 400 (Bad request) и сообщение «Слабый пароль». Описание класса CreateUserDTO представлено в таблице 2.22.

Таблица 2.13 – Описание CreateUserDTO

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| username | string | Имя пользователя |
| email | string | Почта |
| password | string | Пароль |

Каждый новый пользователь в системе получает роль, по умолчанию «USER». Другие роли могут быть выданы администратором системы. Полный перечень ролей в системе представлен в таблице 2.22

Таблица 2.14 – Описание ролей пользователя

|  |  |
| --- | --- |
| **Роль** | **Описание** |
| USER | Простой пользователь системы |
| CURATOR | Куратор, может добавлять к себе в список курируемых пользователей системы с целью просмотра и редактирования их схем |
| ADMIN | Администратор, распределяет роли, обладает возможностью удалять аккаунты и просматривать и редактировать чужие схемы |

За обработку запросов к таблице «Пользователь» отвечает специальный UserController, который предоставляет набор URL путей, по которым пользователь может получить доступ к БД. Все пути в UserController защищены по схеме HTTP Bearer. Каждый запрос должен сопровождаться валидным JWT токеном в заголовке. Сопоставление методов контроллера с путями представлено в таблице 2.15.

Таблица 2.14 – Описание UserController

|  |  |  |
| --- | --- | --- |
| **Метод** | **HTTP запрос** | **Описание** |
| getAllNonPrivelegd() | GET host:port/api/users/nonpriveleged | Получить список всех пользователей с ролью USER |
| findOneById(@Param('id', new ParseUUIDPipe({version: '4'})) id: string) | GET  host:port/api/users/:id | Получить данные пользователя по ID |
| update(@Body() payload:UserDto) | PUT host:port/api/users/update | Обновить данные пользователя |
| deleteUserByID(@Param('id', new ParseUUIDPipe({version: '4'})) id: string) | DELETE  host:port/api/users/:id | Удалить данные пользователя |

Сервисы NestJS осуществляет непосредственную обработку данных, которые предоставляются методами контроллера. Для каждого контроллера реализован минимум один сервис обслуживания запросов. Описание UserService, которые обслуживает запросы UserController, приведено в таблице 2.15.

Таблица 2.15 – Описание класса UserService

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| usersRepository | Repository<User> | API к таблице «Пользователь» |
| **Метод** | | **Назначение** |
| async findAll() | | Получить все записи |
| async findByIds(users: string[]) | | Получить данные нескольких пользователей |
| async findAllNonPrivelegedUsers() | | Получить список всех простых пользователей |
| async findOneByEmail(email: string) | | Получить данные пользователя по его почте |
| async findOneById(id: string) | | Поиск пользователя по id |
| async findOneByName(username : string ) | | Поиск пользователя по имени |
| async create(userData : CreateUserDto) | | Создание нового пользователя |
| async update(id: string, payload: CreateUserDto) | | Обновление конфиденциальных данных пользователя |
| async updatePassword(userid: string, payload: Partial<CreateUserDto>) | | Обновить пароль пользователя |
| async delete(uuid: string) | | Удаление пользователя по id |

Для реализации сервиса авторизации необходим UserService, который предоставляет доступ к таблице «Пользователь». AuthController предоставляет набор URL путей для обеспечения механизма авторизации. Сопоставление методов контроллера с путями представлено в таблице 2.15.

Таблица 2.14 – Описание AuthController

|  |  |  |
| --- | --- | --- |
| **Метод** | **HTTP запрос** | **Описание** |
| login(@Request() req) | POST host:port/api/auth/login | Вход в систему зарегистрированного пользователя |
| signup(@Body() user: CreateUserDto) | POST host:port/api/ auth/signup | Регистрация нового пользователя |
| signup(@Body() user: CreateUserDto) | POST host:port/api/ auth/signup | Регистрация нового пользователя |
| verify(@Param('token') token: string) | GET host:port/api/ auth/verify/:token | Проверка валидности JWT токена |
| deleteUser(@Headers() req) | DELETE host:port/api/ auth/login | Удалить учётную запись пользователя |
| changePassword(@Body() changePasswordDto: ChangePasswordDto,) | PATCH  host:port/api/ auth/changePassword | Обновить пароль |

AuthController для изменения пароля использует специальный интерфейс ChangePasswordDTO, описание которого приведено в таблице 2.14. Для нового пароля используются декоратор проверки сложности пароля, аналогичный тому, которые применяется для CreateUserDTO.

Таблица 2.14 – Описание ChangePasswordDTO

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| username | string | Имя пользователя |
| oldPassword | string | Старый пароль |
| newPassword | string | Новый пароль |

Реализация защиты путей приложения от несанкционированного доступа осуществляется с помощью библиотеки «Passport» и реализуемых ею стратегий LocalStrategy и JwtStrategy, которые предоставляют методы авторизации по связке логин плюс пароль или через токен соответственно.

Защищены все URL пути приложения, кроме запросов регистрации нового пользователя и проверки токенов на валидность.

Методы авторизации реализуются через AuthService, описание которого представлено в таблице 2.22.

Таблица 2.17 – Описание класса AuthService

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| userService | UserService | Сервис пользователей для получения доступа к таблице «Пользователь» |
| jwtService | JwtService | Сервис безопасности и валидации JWT для идентификации и выдачи токенов |
| cryptoService | CryptoService | Сервис хеширования и валидации пароля |
| configService | ConfigService | Сервис управления файлами конфигурации |
| **Метод** | | **Назначение** |
| async signUp(user : CreateUserDto) | | Регистрация нового пользователя |
| async logIn(username: string, password: string) | | Вход зарегистрированного пользователя и выдача токена |
| async createToken(user) | | Создание и подпись токена |
| async verify(payload) | | Проверка, что пользователь зарегистрирован в системе |
| async verifyToken(token) | | Проверка токена на валидность |
| async changePassword(changePasswordDto: ChangePasswordDto) | | Изменение пароля учетной записи пользователя |

За обработку запросов к таблице «Курируемое отношение» отвечает специальный CuratorController. Его описание представлено в таблице 2.20.

Таблица 2.14 – Описание CuratorController

|  |  |  |
| --- | --- | --- |
| **Метод** | **HTTP запрос** | **Описание** |
| async getAll() | GET host:port/api/curators | Получить все курируемые отношения |
| getOneById(@Param('id') id : number) | GET host:port/api/curators/:id | Получить курируемое отношение по id |
| getByCuratorId(@Param('id', new ParseUUIDPipe({version: '4'})) id: string) | GET host:port/api/ curators/create | Получить все курируемые отношения для конкретного куратора |
| async createRelations(@Body() curatorObject: CreateCuratorDto) | POST host:port/api/ curators | Создать новое отношение |
| async update(@Body() payload: CuratorDto) | PUT host:port/api/ curators/:id | Обновить существующее отношение |
| async removeRelation(@Param() relationId: number) | DELETE  host:port/api/ auth/changePassword | Удалить отношение из базы данных |

Для создания нового курируемого отношения используется интерфейс CreateCuratorDTO. Его описание представлено в таблице 2.20.

Таблица 2.14 – Описание CreateCuratorDTO

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| uuid\_curator | string | id куратора |
| uuid\_user | string | id пользователя |
| relation\_name | string | название курируемого отношения |

Описание сервиса по взаимодействие с таблицей «Курируемое отношение» представлено в таблице 2.21.

Таблица 2.14 – Описание CuratorService

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| curatorRepository | Repository  <Curator> | API к таблице «Курируемое отношение» |
| **Метод** | | **Назначение** |
| private async findAll() | | Получить все записи |
| async findStudents(id\_curator: number) | | Получить всех пользователей, привязанных к куратору |
| async findOneById(id: string) | | Поиск отношения куратор-пользователь по id |
| async findCurators(id\_user : string ) | | Получить список всех кураторов, к которым привязан пользователь |
| async create(curator: CuratorDto) | | Создание нового отношения |
| async update(id: number, payload: CuratorDto) | | Обновление отношения куратора и пользователя |
| async delete(id: number) | | Удаление отношения куратора и пользователя |

За выдачу статических данных, такие как страница и медиаресурсы отвечает модуль ServeStaticModule, который входит в состав NestJS.

Для обеспечения взаимодействия с базой данной используется модуль TypeOrmModule, который входит в состав NestJS.

* + 1. Описание основных классов клиента

Взаимодействие с сервисами сервера осуществляется через методы и функции, реализованные на стороне клиента. Для управления состояниями приложения было разработано четыре сервиса:

* Сервис управления авторизацией;
* Сервис пользователя;
* Сервис управления схемой;
* Сервис управления курируемыми пользователями.

За управление авторизацией в приложении отвечает класс AuthStateService. Он хранит состояние текущего пользователя на протяжении всей его работы. Описание класса приведено в таблице 2.20. При проектировании сервисов на стороне клиента применялся паттерн «Одиночка».

Таблица 2.14 – Описание AuthStateService

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| static \_instance | AuthStateServie | Экземпляр класса |
| static \_accessToken | string | JWT токен доступа |
| static \_user | IUser | Данные авторизованного пользователя |
| **Метод** | | **Назначение** |
| getInstance(): AuthStateService | | Получить экземпляр класса |
| getToken() : string | | Получить JWT токен |
| getRole() : string | | Получить роль |
| getUUID(): string | | Получить id пользователя |
| getUsername(): string | | Получить имя пользователя |
| getEmail(): string | | Получить почту пользователя |
| async TokenVerification(token: string): Promise<boolean> | | Проверка токена на валидность |
| async Authentificate(username: string, password: string): Promise<ResponsePayload> | | Аутентификация пользователя по логину и паролю |
| async RegisterUser(username: string, password: string, email: string): Promise<ResponsePayload> | | Регистрация нового пользователя |
| Logout(): void | | Осуществить выход из системы |
| async DeleteRegistrationData(uuid: string) | | Удалить учетную запись пользователя |
| async create(curator: CuratorDto) | | Создание нового отношения |
| async changePassword(oldPassword: string, newPassword: string) | | Изменить пароль пользователя |

Для чтения ответов сервера на вызовы методов авторизации реализован интерфейс ResponsePayload. Описание приведено в таблице 2.20.

Таблица 2.20 – интерфейс ResponsePayload

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| status | number | Код ответа сервера |
| status\_text | string | Сообщения, сопровождающее ответ |
| body | loginResponseDTO | Содержимое ответа сервера |

В ответ на корректные данные аутентификации пользователя сервер возвращает JWT токен для доступа к защищённым путям приложения и поля класса User, которые помещаются в интерфейс loginResponseDTO.

Получения данных о других пользователях осуществляется через UserStateService клиента. Также данный сервер отвечает за изменение персональных данных текущего пользователя, поэтому для его работы необходим экземпляр класса AuthStateService. Описание класса UserStateService клиента приведено в таблице 2.14.

Таблица 2.14 – Описание UserStateService

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| static \_instance | UserStateService | Экземпляр класса |
| static \_authService | AuthStateService | Экземпляр класса регистрации |
| **Метод** | | **Назначение** |
| getInstance():UserStateService | | Получить экземпляр класса |
| setTokenBearer () : JSON | | Установка заголовка с токеном авторизации для выполнения запросов |
| getUserinfoById(uuid: string): Promise<UserDTO> | | Получить данные о пользователе по его ID |
| GetUnprevilegedUsers(): Promise<UserDTO[]> | | Получить список всех пользователей с ролью USER |
| GetAllUsers(): Promise<UserDTO[]> | | Получить список всех пользователей |
| async UpdateData(user: UserDTO) | | Обновить персональные данные пользователя |

Работа с курируемыми пользователями осуществляется через CuratorStateService. Описание класса приведено в таблице 2.15.

Таблица 2.15 – Описание класса CuratorStateService

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| static \_instance | CuratorStateService | Экземпляр класса |
| static \_authService | AuthStateService | Экземпляр класса регистрации |
| **Метод** | | **Назначение** |
| getInstance():UserStateService | | Получить экземпляр класса |
| createRelation(relation : CreateRelationDTO) | | Создание нового отношения между куратором и пользователем |
| DeleteRelation(id: number) | | Удаление существующего отношения |
| getRelationsByCurator(curator: string) | | Получить список всех курируемых пользователей по id куратора |
| updateRelation(relation: RelationDTO) | | Обновить выбранное отношение |

Взаимодействие со схемами пользователя на удалённом хранилище в базе данных осуществляется с помощью SchemaService. Описание данного класса приведено в таблице 2.20. Выбранная для редактирования схема помещается в статическую переменную класса.

Таблица 2.15 – Описание класса SchemaService

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| static \_instance | SchemaService | Экземпляр класса |
| static \_authService | AuthStateService | Экземпляр класса регистрации |
| static \_schema | DragonModel | Модель ДРАКОН-схемы |
| **Метод** | | **Назначение** |
| getInstance():UserStateService | | Получить экземпляр класса |
| public getModel(model: DragonModel) | | Получить модель ДРАКОН-схемы |
| public setModel(model: DragonModel) | | Установить данные для модели ДРАКОН-схемы |
| async getUserSchemas(user\_id : string) : Promise<SchemaDTO[]> | | Получить все схемы выбранного пользователя |
| async createNewSchema(user\_id: string) : Promise<SchemaDTO | undefined> | | Создание новой схемы для текущего пользователя |
| updateSchema(schema: SchemaDTO): Promise<SchemaDTO | undefined> | | Обновление схемы |
| deleteSchema(id: string): Promise<any> | | Удаление схемы |

Для представления модели данных были разработаны классы, которые представлены в приложении 1. Описание класса ModelDragon представлено в таблице 2.20.

Таблица 2.15 – Описание класса ModelDragon

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| \_head | string | Указатель на макроикону «Схема» |
| \_container | map<string, DragonInstruction> | Словарь для хранения ссылок на все иконы ДРАКОН-схемы |
| **Метод** | | **Назначение** |
| Insert(instruction: DragonInstruction, next?: string): void | | Вставка иконы в ДРАКОН-схему |
| toJSON(): JSON | | Преобразование схемы в формат JSON |
| public setModel(model: DragonModel) | | Присваивание |
| Delete(parent: string, uuid: string) | | Удаление иконы |
| private RecursiveDelete(parent: DragonInstruction) | | Рекурсивное удаление всех вложенных икон |
| validate() | | Проверка на висячие иконы |
| Find(uuid: string) | | Поиск инструкции в ДРАКОН-схеме |
| getDeepestLeftChild(uuid: string) | | Найти самого «левого» вложенную икону |
| Update(uuid: string, text: string) | | Обновление данных иконы |
| private castInstruction<T extends DragonInstruction>(instruction: T, instruction\_meta: metadataInterface) | | Преобразование абстрактной иконы к конкретной реализации |
| private parseInstructionFromJSON(instruction\_id: string, schema: any) | | Чтение иконы в формате JSON |
| static restoreFromJSON(schema: any) | | Статическая версия алгоритма чтения схемы в формате JSON |
| public restoreFromJSON(schema: any) | | Чтение схемы в формате JSON |
| public parseInstruction(instruction: DragonInstruction, code: string, deep = 0): string | | Трансляция иконы в язык JavaScript |
| public toJavaScript(): string | | Трансляция схемы в язык JavaScript |

Каждая икона представляет собой экземпляр от наследуемого класса DragonInstruction. Который описывает базовое поведение для всех икон. Описание его содержимого представлено в таблице 2.20.

Таблица 2.15 – Описание класса DragonInstruction

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Назначение** |
| \_head | string | Указатель на макроикону «Схема» |
| \_container | map<string, DragonInstruction> | Словарь для хранения ссылок на все иконы ДРАКОН-схемы |
| **Метод** | | **Назначение** |
| Insert(instruction: DragonInstruction, next?: string): void | | Вставка иконы в ДРАКОН-схему |
| toJSON(): JSON | | Преобразование схемы в формат JSON |
| public setModel(model: DragonModel) | | Присваивание |
| Delete(parent: string, uuid: string) | | Удаление иконы |
| private RecursiveDelete(parent: DragonInstruction) | | Рекурсивное удаление всех вложенных икон |

У каждой иконы есть свой тип, а также наследуемый от DragonInstruction класс. Полный список реализованных икон представлен в таблице 2.20.

Таблица 2.14 – Типы икон

|  |  |  |
| --- | --- | --- |
| **Тип** | **Описание** | **Реализуемый класс** |
| SCHEMA | ДРАКОН-схема | DragonSchemaInstruction |
| PRIMITIVE | Макроикона «Примитив» | DragonPrimitiveInstruction |
| ACTION | Икона «Действие» | DragonActionInstruction |
| CONDITION | Макроикона «Вопрос» | DragonConditionInstruction |
| LIMITER | Ограничитель условных конструкций | DragonLimiterInstruction |
| BRANCH | Макроикона «Ветка» | DragonBranchInstruction |
| INPUT | Икона «Ввод» | DragonInputInstruction |
| OUTPUT | Икона «Вывод» | DragonOutputInstruction |
| LOOP | Макроикона «Цикл» | DragonLoopInstruction |
| SWITCH | Макроикона «Вариант» | DragonSwitchInstruction |
| COMMENT | Икона «Комментарий» | DragonCommentInstruction |
| SLEEP | Икона «Таймер» | DragonSleepInstruction |

* 1. Реализованные меню и интерфейсы

ПРИЛОЖЕНИЕ 1 Диаграмма классов ДРАКОН-схемы

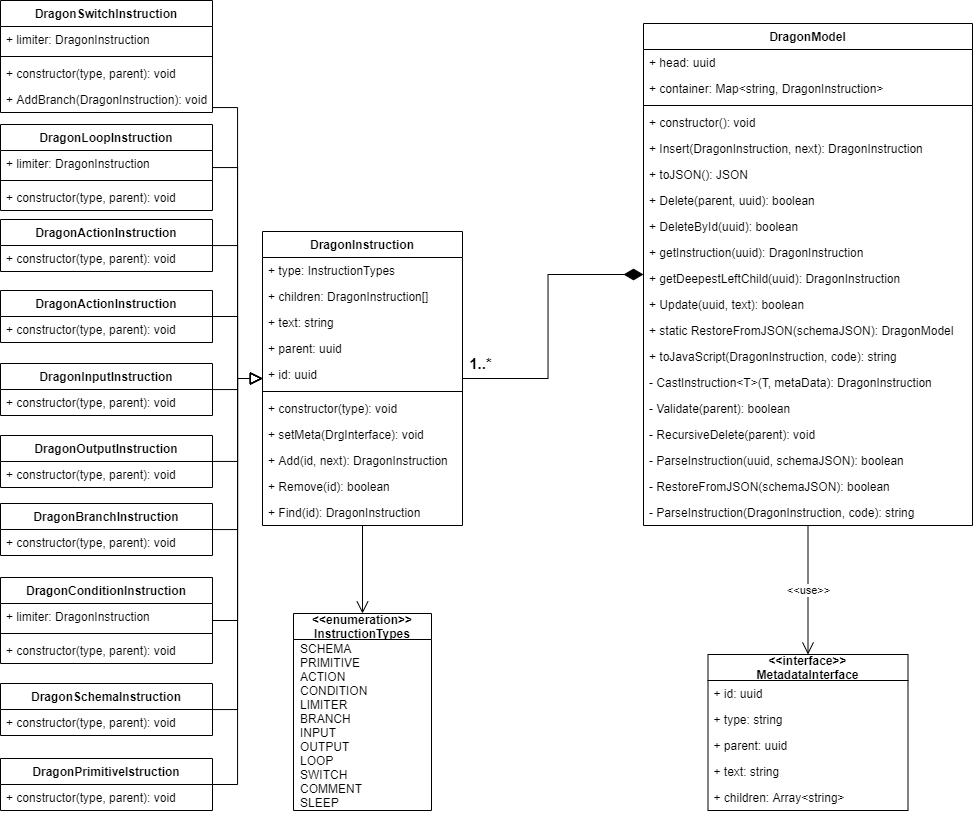


Рисунок П.1 – Диаграмма классов для представления ДРАКОН-схем

ПРИЛОЖЕНИЕ 2 Диаграмма классов ДРАКОН-схемы

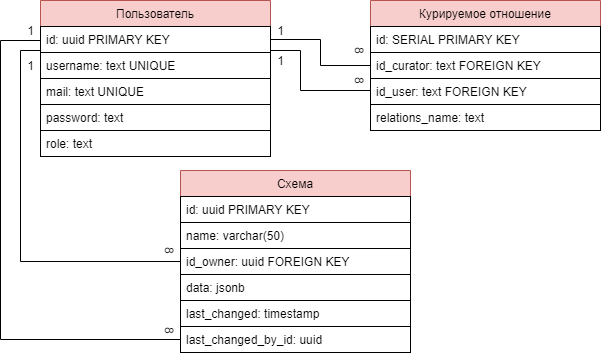


Рисунок П.2 – Диаграмма сущность-связь базы данных «БД\_Дракон»

ПРИЛОЖЕНИЕ 3 Диаграмма развертывания

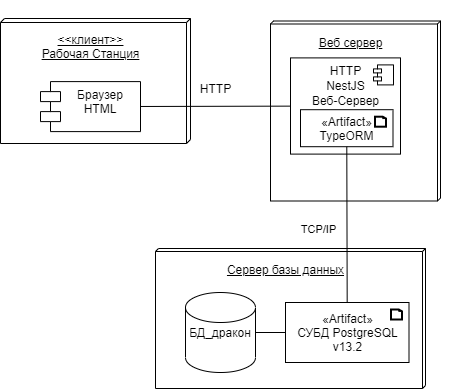


Рисунок П.3 – Диаграмма развертывания среды по обучению алгоритмизации DRAKON IDE

ПРИЛОЖЕНИЕ 4 Диаграмма компонентов сервера

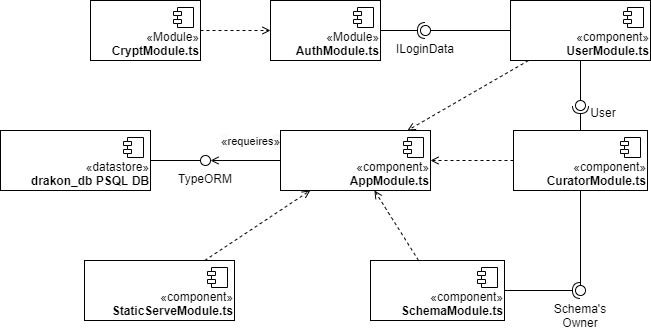


Рисунок П.4 – Диаграмма компонентов сервера NestJS